Sricharan Vadapalli

Hands-on DevOps

Explore the concept of continuous delivery and integrate it with data science concepts

Packt

Hands-on DevOps

Explore the concept of continuous delivery and integrate it with data science concepts

Sricharan Vadapalli

Packt

BIRMINGHAM - MUMBAI

Hands-on DevOps

First published: December 2017

Production reference: 1191217

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.
ISBN 978-1-78847-118-3

www.packtpub.com

Credits

Author
Sricharan Vadapalli

Reviewers
Jason Myerscough
Prakash Sarma

Commissioning Editor
Gebin George

Acquisition Editor
Prateek Bharadwaj

Content Development Editor
Abhishek Jadhav

Technical Editor
Aditya Khadye

Copy Editors
Safis Editing
Dipti Mankame

Project Coordinator
Judie Jose

Proofreader
Safis Editing

Indexer
Rekha Nair

Graphics
Tania Dutta

Production Coordinator
Deepika Naik

About the Author

Sricharan Vadapalli is an Information Technology leader with over 2 decades of experience in leading IT strategy and developing Next Generation IT solutions. Passionate with technology, build practices around DevOps, Big Data, Cloud, SAP HANA and EIM (Enterprise Information Management) Associated with Tech Startups, Instrumental in devising Go-Market strategy, strong proponent of open source tools adoption, Microservices, and Containers. Played leadership roles with MNC's like TCS, Infosys, CSC, Virtusa Polaris, handled fortune client engagements across multiple domains like BSFI, Retail, Telecom and Manufacturing.

Sricharan has a Masters of Technology from the Indian Institute of Technology, Delhi. His keen interest for continuous learning saw him certified in SAP HANA, In-Memory, PMP, ITIL, OCA, Function Point, and INS21. He is voracious reader, passionate about Yoga, nature lover, and is interested in traveling. His earlier book on *Reality of Relations* is a valuable for building successful relations, and key contributor for long-lasting happiness.

He lives in Hyderabad, India, with wife and two daughters.

About the Reviewers

Jason Myerscough was born in the UK but has been living in Vienna for the past 8 years with his wife Tina and their two young children, Harrison and James. Jason has 10+ years experience in the industry, where he started as a software engineer. Four years ago, Jason transitioned into the DevOps and cloud computing space, and today he manages a cloud engineering team that is working on migrating products to the cloud and evangelising DevOps. Jason has a BSc in Software Engineering and an MSc in Internet Technologies.

Prakash Sarma is the co-founder and Managing Partner of Newstar Corporation, focused on Digital Transformation in the Enterprise. Prakash has spent the past decade building high traffic and scalable e-commerce and digital platforms for Fortune 500 companies. A technologist at heart, with a product and revenue driven mindset, Prakash is passionate about E-commerce, Distributed Systems, Cloud Platforms, Big Data, and DevOps. He frequently advises startups and presents and writes on these topics as well as innovation and culture transformations in enterprises.

www.PacktPub.com

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www.packtpub.com/mapt`

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at `https://www.amazon.com/dp/1788471180`.

If you'd like to join our team of regular reviewers, you can email us at `customerreviews@packtpub.com`. We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!

Table of Contents

Preface

DevOps and big data are most sought-after technology trends bringing multifold benefits to business, industries, and R and D centers. Their widespread adoption is associated with the popularity and acceptance of Open source tools along with other technology advancements, such as cloud computing, containers, and the Internet of Things.

In this book, we will discuss the business drivers and market trends contributing to the evolution of technology. We will explore the key technological concepts, such as NoSQL databases, microservices, data science, and data visualization. We will build big data enterprise applications, using core architectural principles, based on the Spark in-memory engine. We will discuss DevOps practices and frameworks, and their application with popular tools, to prepare for continuous integration, continuous delivery, and continuous deployment. We will dive deeper into the popular open source tools, such as Chef, Ansible, Jenkins, Git, and Stack, which assist in the automation of the development to deployment cycle.

At every opportunity, this book introduces different flavors of popular vendor offerings with the key components and differences. This will provide the reader with information to aid their decision making. The technology choices discussed are for Hadoop platforms, cloud offerings, container and orchestration services, Internet of Things platforms, and NoSQL databases. The choice of popular tools for continuous integration, delivery, deployment, and monitoring is also thoroughly discussed. There are topics on capacity planning for Hadoop cluster, and security models for data and cloud based, discussed to provide reader with comprehensive knowledge on these subjects.

The migration strategy to DevOps for different systems, including big data systems, cloud platforms, microservices, data sciences, security authentications, and Internet of Things platforms, is discussed. ERP systems adoption to DevOps is also shared with detailed testing strategy.

Every effort is made to benefit the reader to learn the wide range of technology, tools and platform evolutions, frameworks, and methodologies from practical usage perspective. I feel confident that reader will find it very interesting and useful from multiple perspectives.

What this book covers

`Chapter 1`, *Introduction to DevOps*, focuses on business trends, drivers, market propellers of evolution, and adoption of DevOps. The key technological concepts are introduced for big data, cloud, data sciences, and in-memory computing. DevOps application scenarios and the benefits of adoption for an organization are presented.

`Chapter 2`, *Introduction to Big Data and Data Sciences*, starts with the introduction of big data concepts, discuss the features of in-memory databases for real-time systems, and presents different NoSQL database types for different uses as well as concepts relating to data visualization, data science, and data scientist roles.

`Chapter 3`, *DevOps Framework*, discusses for source code management, build, repository, release managements, and test automation were discussed. Continuous integration, delivery, deployment were covered along with infrastructure automation for configuration (Infrastructure as Code), application monitoring, and so on. DevOps maturity models, progress frameworks, and the Agile methodology are also discussed.

`Chapter 4`, *Big Data Hadoop Ecosystems*, focuses on big data clusters concepts such as data lake, processing frameworks, node types, roles, capacity planning, security, and authentications. Different flavors of Hadoop frameworks (Cloudera, Hortonworks, MapR, Pivotal, IBM Open platform, Amazon Elastic Mapreduce, and Micosoft Azure HDInsight) were discussed.

`Chapter 5`, *Cloud Computing*, discusses technology concepts for cloud, such as virtualization and hypervisor are discussed along with WEB protocols such as REST APIs. Cloud architecture layers and components are discussed, and different cloud models such as private, public, and hybrid cloud are presented. Offerings such as IaaS, PaaS, SaaS, and their variants are detailed. Features included in cloud vendor-based offerings, such as security, encryption, and business continuity models, are discussed.

`Chapter 6`, *Building Big Data Applications*, discusses life cycle stages, such as data discovery, quality, ingestion, analytics, visualization, governance, as well as their best practices. Data ingestion models for batch and real-time, streaming data, and respective tools are also discussed. Data storage concepts such as data lake, data analytics, Spark in-memory engine are also discussed. Core architecture components, such as workflow, security, data catalog, client services, and APIs. Data science algorithms; their business usage/application, different models, such as decision trees, clusters, and supervised and unsupervised learning, are amply covered.

`Chapter 7`, *DevOps - Continuous Integration and Delivery*, discusses the CI/CD methodology with open source popular tools such as Git, Maven, Gerrit, Nexus, Selenium, and Jenkins. Different tools and plugins compatible with Jenkins were discussed.

`Chapter 8`, *DevOps Continuous Deployment*, shows the popular continuous deployment tools Ansible and Chef; their advanced features such as Habitat, Automate, and Compliance for security are discussed. Core components, architecture and terminology (Chef, cookbooks knife, Playbooks, towers) are elaborated. Continuous monitoring tools Splunk and Nagios are covered in depth.

`Chapter 9`, *Containers, IoT, and Microservices*, covers virtualization, containers, orchestration, Internet of Things (IoT), microservices, hypervisors, and emulation. Containers offered by multiple vendors (AWS, Google, Pivotal, and Azure), including open source technologies such as J2EE and Docker, are covered at the component level. Orchestration tools (Mesos, Kubernetics, and Docker Swarm) are discussed. Mutiple IoT platforms, from both commercial vendors (ThingWorx, Electric Imp, and Predix) and open source (Eclipse and Contiki), are presented. Microservice design patterns, architecture models, client-side, server-side service models, and best practices are detailed.

`Chapter 10`, *DevOps for Digital Transformation*, covers the details of the digital journey for enterprise systems such as big data, cloud, data science, Internet of Things, and security authentication systems for progressive adoption with challenges and best practices. Mircoservice models are elaborated with different models and patterns.

`Appendix A`, *DevOps Adoption by ERP Systems*, discusses ERP systems like SAP have adopted DevOps process along with open source tools like Jenkins and Git to improve efficiency.

`Appendix B`, *DevOps Periodic Table*, depicts the entire list presented in the form of periodic table describing many open source DevOps tools for each stage of development, deployment, and monitoring.

`Appendix C`, *Business Intelligence Trends*, depicts shift in priorities, as we can see that data preparation and discovery are rated as high value added to master data management.

`Appendix D`, *Testing Types and Levels*, presents us end to end testing cycle at each of the layers DB, APP, and UI. Elaborate testing types are discussed.

`Appendix E`, *Java Platform SE 8*, shows Java enterprise edition layered architecture components are shared for reference.

What you need for this book

Prerequisite knowledge for fundamentals of software development life cycle and systems. It will help readers connect with basic topics and appreciate the application of various technologies. For advanced concepts of cloud offerings, Hadoop clusters, big data applications, microservices, containers, security, and digital migration, familiarity with architecture is required.

Who this book is for

If you are a big data architects, solutions provider, or any stakeholder working in big data environment and wants to implement the strategy of DevOps, then this book is for you.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning. Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "For example--`sqoop import –connect jdbc:mysql://:/ –username –password --table --target-dir`"

A block of code is set as follows:

```
 - hosts: webservers
serial: 6 # update 6 machines at a time
roles:
- common
- webapp
- hosts: content_servers
```

Any command-line input or output is written as follows:

```
C:>Java –jar Jenkins.war
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "The **Manage Jenkins** option in the dashboard will provide various options to configure various parameters."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply email `feedback@packtpub.com`, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `https://www.packtpub.com/sites/default/files/downloads/HandsonDevOps_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title. To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the internet, please provide us with the location address or website name immediately so that we can pursue a remedy. Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material. We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Introduction to DevOps

In the traditional approach to application development and maintenance, multiple stakeholders, departments, groups, and vendors are involved in the overall **software development life cycle** (**SDLC**). Most of us are familiar with the stages of application life cycle management: the business requirements are gathered by a business analyst, then developed by the development team (or could have been outsourced), and tested by QA teams (also could have been outsourced) for functionality and fitness for purpose. Performance and stress testing were also performed in applicable scenarios, by appropriate groups with relevant tools. Then the production deployment process, with a checklist and approvals, was managed by the IT teams at the organization, followed by monitoring and support by maintenance teams. And we notice that each stage of the maturity cycle, from functionality development to usability and maintenance, is managed in silos, by independent teams, departments, processes, and tools. This approach is often fragmented by techniques, frameworks, processes, people, and tools impacting the final product in terms of features, cost, schedule, quality, performance, and other administrative overheads such as interfacing and integration between vendors. Also, in this method the maintenance, support costs, and skill needs are often overlooked. However, both from application life cycle and business points of view, maintenance and support activities are key and important to assess, evaluate, and estimate well in advance.

In this chapter, we will cover the following topics:

- Introduction to DevOps
- Business application of DevOps.
- Business drivers/market trends
- DevOps strategy
- Benefits of DevOps

Many technological innovations have taken place to challenge the traditional method of IT management in almost every segment. The technological advances and changes are quite profound, rapid, and often intermingled, encompassing multiple fields such as agile methodology, DevOps, big data, cloud, and so on. A comprehensive and holistic approach will undoubtedly be rewarding and derive maximum value for organizations. Many institutions have already embarked on this journey towards the future, adopting these technologies.

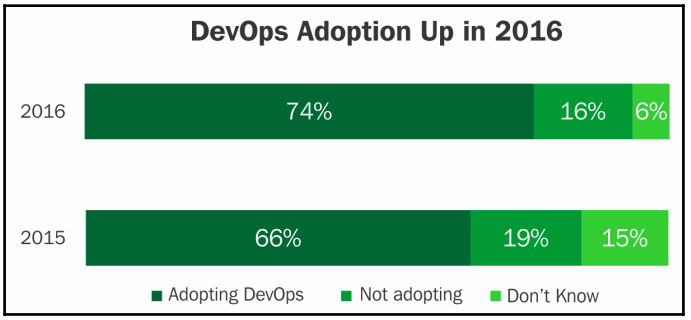
The pre-DevOps software development challenges are reluctant to change in systems; deployments fraught with risk, lack of consistency across environments (*it works on my machine* syndrome), the impact of silos--toss problems across *the wall* such as teams resulting in duplication of effort, skill sets, and in-fighting. To mitigate the mentioned issues and bridge this gap DevOps emerged as a popular choice.

DevOps (Development plus Operations) has recently taken center stage in the SDLC. DevOps offers process frameworks augmented with open source tools to integrate all the phases of the application life cycle, and ensure they function as a cohesive unit. It helps to align and automate the process across the phases of development, testing, deployment, and support. It includes best practices such as code repositories, build automation, continuous deployment, and others.

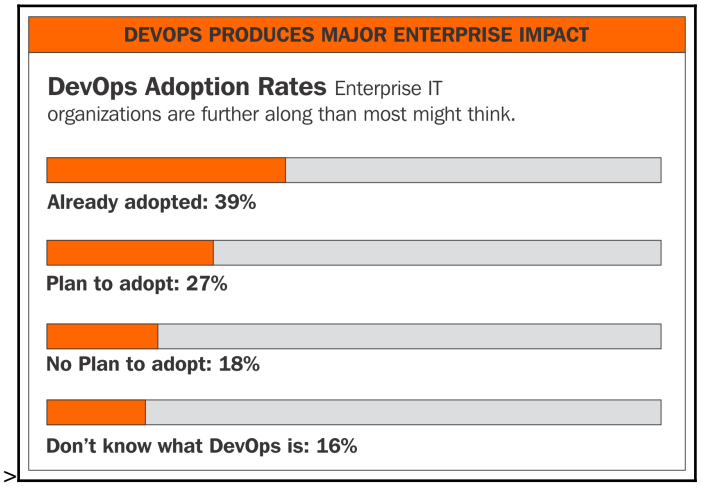
DevOps adoption for systems including big data systems and projects is a cultural shift compared to traditional development cycles. The purpose of this book is to put forth the concepts and adoption strategy for an organization, covering the technology areas of DevOps, big data, cloud, data science, in-memory technology, and others. Adopting and adhering to DevOps practices will be rewarding for any organization and allow it to improve on its performance and efficiency.

Acceptance of open source tools for each segment of IT functionality, their popularity, and versatility, is increasing day by day, across the world. As a matter of fact, many new tool variants have been introduced to the market for each segment. The open source tools for DevOps are major contributors to the success of DevOps' adoption in the market by institutions, which is discussed in detail in coming sections.

As we can see, across industries DevOps adoption has seen steady growth year on year:



DevOps penetration in enterprises shows a healthy trend, as per the following picture:



DevOps application - business scenarios

Application of DevOps varies for multiple scenarios, with accrued benefits as listed:

- **Automation of development cycle**: Business needs are met with minimal manual intervention, and a developer can run a build with a choice of open tools through a code repository; the QA team can create a QA system as a replica, and deploy it to production seamlessly and quickly.

- **Single version of truth - source code management***:* There are multiple versions of the code, but it is difficult to ascertain the appropriate code for the purpose. We lack a single version of the truth. Code review feedback is through emails and not recorded, leading to confusion and rework.
- **Consistent configuration management**: We develop, test, and build source code on different systems. Validating the platforms and compatibility versions of dependencies is manual and error-prone. It's really challenging to ensure all the systems speak the same language, and have the same versions of the tools, compilers, and so on. Our code works fine on build systems but doesn't when moved to production systems, causing embarrassment regarding business deliverables, and cost overheads to react.
- **Product readiness to markets***:* We have a process to develop code, test, and build through defined timelines. There are many manual checks and validations in the process; the integrations between different groups cause our commitments and delivery dates to be unpredictable. We wish to know how close our product is to delivery and its quality periodically, to plan in advance rather than being reactive.
- **Automation of manual processes**: We are following manual processes, which are often error prone, and wish to enhance efficiency by following an automation process wherever applicable. Testing cycle automation, incremental testing, and integrating with the build cycle will expedite product quality, the release cycle, and infrastructure service automation such as creating, starting, stopping, deleting, terminating, and restarting virtual or bare-metal machines.
- **Containers***:* Portability of code is the primary challenge. The code works in development and QA environments, but moving to production systems causes multiple challenges such as code not compiling due to dependency issues, build break down, and so on. Building platform agnostic code is a challenge, and maintaining multiple platform versions of development and QA platforms is a huge overhead. Portable container code would alleviate these kinds of issues.
- **On-premise challenges**: We have many on-premise systems. There are multiple challenges, from capacity planning to turnaround time. The Capex and operational expenses are unpredictable. Cloud migration seems to have multiple choices and vendors, so there needs to be an efficient adoption method to ensure results.

Business drivers for DevOps adoption to big data

Factors contributing to wide-scale popularity and adoption of DevOps among big data systems are listed as follows.

Data explosion

Data is the new form of currency--yes you read right, it's as much a valuable asset as oil and gold. In the past decade, many companies realized the potential of data as an invaluable asset to their growth and performance.

Let's understand how data is valuable. For any organization, data could be in many forms such as, for example, customer data, product data, employee data, and so on. Not having the right data on your employees, customers, or products could be devastating. It's basic knowledge and common sense that the correct data is key to running a business effectively. There is hardly any business today that doesn't depend on data-driven decisions; CEOs these days are relying more on data for business decisions than ever before, such as which product is more successful in the market, how much demand exists area-wise, which price is more competitive, and so on.

Data can be generated through multiple sources, internal, external, and even social media. Internal data is the data generated through internal systems and operations, such as in a bank, adding new customers or customer transactions with the bank through multiple channels such as ATM, online payments, purchases, and so on. External sources could be procuring gold exchange rates and foreign exchange rates from RBI. These days, social media data is widely used for marketing and customer feedback on products. Harnessing the data from all avenues and using it intelligently is key for business success.

Going a step further, a few companies even monetize data, for example, Healthcare IQ, Owens & Minor, State Street Global Corporation, Ad Juggler, comScore, Verisk Analytics, Nielsen, and LexisNexis. These organizations buy raw data such as web analytics on online product sales, or online search records for each brand, reprocess the data into an organized format, and sell it to research analysts or organizations looking for competitor intelligence data to reposition their products in markets.

Let's analyze the factors fueling the growth of data and business. Fundamental changes in market and customer behavior have had a significant impact on the data explosion. Some of the key drivers of change are:

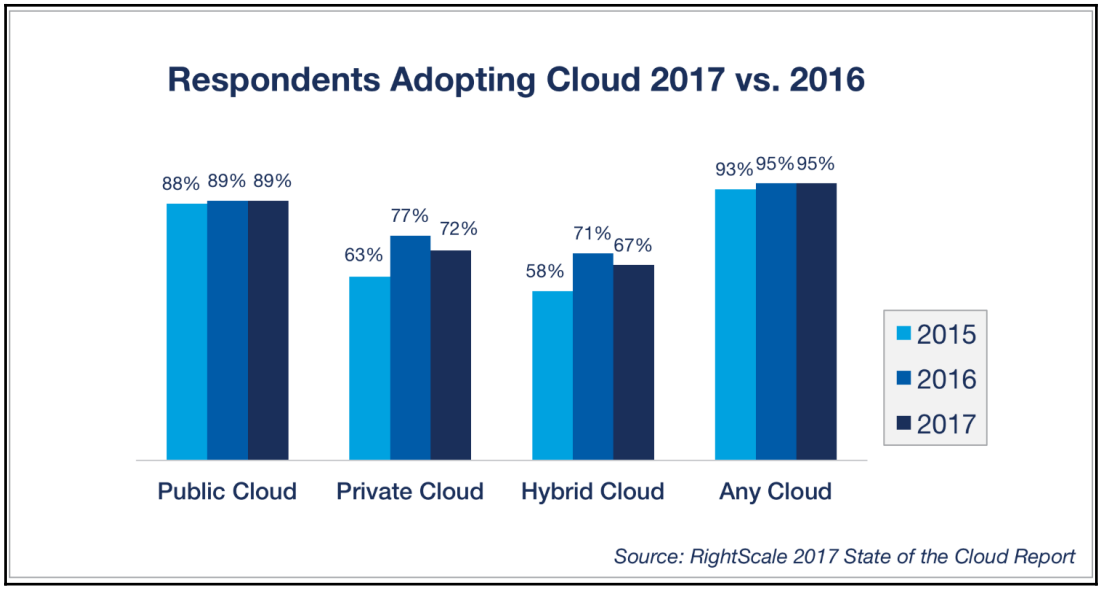
- **Customer preference**: Today, customers have many means of interacting with businesses; for example, a bank provides multiple channels such as ATM withdrawals, online banking, mobile banking, card payments, on-premise banking, and so on. The same is true for purchases; these can be in the shop, online, mobile-based, and so on, which organizations have to maintain for business operations. So, these multiple channels contribute to increased data management.
- **Social media**: Data is flooding in from social media such as Facebook, LinkedIn, and Twitter. On the one hand, they are social interaction sites between individuals; on the other hand, companies also rely on social media to socialize their products. The data posted in terabytes/petabytes, in turn, is used by many organizations for data mining too. This is contributing to the huge data explosion.
- **Regulations**: Companies are required to maintain data in proper formats for a stipulated time, as required by regulatory bodies. For example, to combat money laundering, each organization dealing with finance is required to have clear customer records and credentials to share with regulatory authorities over extended periods of time, such as 10 to 15 years.
- **Digital world**: As we move towards the paperless digital world, we keep adding more digital data, such as e-books and ERP applications to automate many tasks and avoid paperwork. These innovations are generating much of the digital data growth as well.

The next generation will be more data intensive, with the Internet of Things and data science at the forefront, driving business and customer priorities.

Cloud computing

Acceptance of cloud platforms as the de facto service line has brought many changes to procuring and managing infrastructure. Provisioning hardware and other types of commodity work on the cloud is also important for improving efficiency, as moving these IT functions to the cloud enhances the efficiency of services, and allows IT departments to shift their focus away from patching operating systems. DevOps with cloud adoption is the most widely implemented popular option. With cloud penetration, addition of infrastructure/servers is just a click away. This, along with credible open source tools, has paved the way for DevOps.

In a fraction of time, build, QA, and pre-prod machines can be added as exact replicas and configurations as required, using open source tools.



Big data

Big data is the term used to represent multiple dimensions of data such as large volumes, velocity, and variety, and delivering value for the business. Data comes from multiple sources, such as structured, semi-structured, and unstructured data. The data velocity could be a batch mode, real-time from a machine sensor or online server logs, and streaming data in real time. The volumes of data could be terabytes or petabytes, which are typically stored on Hadoop-based storage and other open source platforms. Big data analytics extends to building social media analytics such as market sentiment analysis based on social media data from Twitter, LinkedIn, Facebook, and so on; this data is useful to understand customer sentiment and support marketing and customer service activities.

Data science and machine learning

Data science as a field has many dimensions and applications. We are familiar with science; we understand the features, behavior patterns, and meaningful insights that result in formulating reusable and established formulas. In a similar way, data can also be investigated to understand the behavior patterns and meaningful insights, through engineering and statistical methods. Hence it can be viewed as data + science, or the science of data. Machine learning is a combination of data extraction, **extract, transform, load** (**ETL**) or **extract, load, transform** (**ELT**) preparation, and using prediction algorithms to derive meaningful patterns from data to generate business value. These projects have a development life cycle in line with a project or product development. Aligning with DevOps methodologies will provide a valuable benefit for the program evolution.

In-memory computing

Traditional software architecture was formerly based on disks as the primary data storage; then the data moved from disk to main memory and CPU to perform aggregations for business logic. This caused the IO overhead of moving large volumes of data back and forth from disk to memory units.

In-memory technology is based on hardware and software innovations to handle the complete business application data in the main memory itself, so the computations are very fast. To enable in-memory computing, many underlying hardware and software advancements have contributed.

The software advancements include the following:

- Partitioning of data
- No aggregate tables
- Insert the only delta
- Data compression
- Row plus column storage

The hardware advancements include the following:

- Multi-core architecture allows massive parallel scaling
- Multifold compression
- Main memory has scalable capacity
- Fast prefetch unlimited size

We will elaborate on these in detail in coming chapters.

Planning the DevOps strategy

A good DevOps strategy, discussed in this book, helps the user gain in-depth and wider understanding of its subject and its application to multiple technologies and interfaces, to an organization provides focus, creates a common (unbiased) view of the current problems, develops the future state, unveils opportunities for growth, and results in better business outputs.

A holistic DevOps strategy, at the most basic level, must answer the following questions:

- What are our business aims and goals?
- How do we plan the roadmap? Where do we begin?
- How should we channel our efforts?
- What are we trying to accomplish?
- What is the schedule for this?
- What is the impact to the business?
- How do our stakeholders see the value?
- What are the benefits and costs of doing it?

A good DevOps strategy for an organization will bring multiple benefits, channel energy to focus on high impact problems, produce clarity to develop the future state, identify growth opportunities, and pave the way for better business outputs.

A DevOps platform strategy will be a unique and extensive program, covering every aspect of the software life cycle, which integrates multiple technologies, platforms, and tools, and posing numerous challenges that need to be handled with skill, precision, and experience.

An organization can consider the introduction of DevOps to cater to specific purposes, such as the following:

- Automating infrastructure and workflow configuration management
- Automating code repositories, builds, testing, and workflows
- Continuous integration and deployment
- Virtualization, containerization, and load balancing
- Big data and social media projects
- Machine-learning projects

There are a wide variety of open source tools to select for adoption in specific segments of DevOps, such as the following:

- **Docker**: A Docker container consists of packaging the application and its dependencies all up in a box. It runs as an isolated process on the host operating system, sharing the kernel with another container. It enjoys resource isolation and allocation benefits like VMs, but is much more portable and efficient.
- **Kubernetes**: Kubernetes is an open source orchestration system for Docker containers. It groups containers into logical units for easy management and discovery, handles scheduling on nodes, and actively manages workloads to ensure their state matches users' declared intentions.
- **Jenkins**: Jenkins is a web-enabled tool used through application or a web server such as Tomcat, for continuous build, deployment, and testing, and is integrated with build tools such as Ant/Maven and the source code repository Git. It also has master and dump slaves.
- **Ansible**: Ansible automates software provisioning, configuration management, and application deployment with agentless, **Secured Shell** (**SSH**) mode, Playbooks, Towers, and Yum scripts are the mechanisms.
- **Chef and Puppet**: Chef and Puppet are agent-based pull mechanisms for the deployment automation of work units.
- **GitHub**: Git is a popular open source version control system. It's a web-based hosted service for Git repositories. GitHub allows you to host remote Git repositories, and has a wealth of community-based services that make it ideal for open source projects.

There are comprehensive frameworks readily available, such as RedHat Openshift, Microsoft Azure, and AWS container services, with pre-integrated and configured tools to implement.

A few popular open source tools are listed here:

- **Source code management**: Git, GitHub, Subversion, Bitbucket
- **Build management**: Maven, Ant, Make, MSBuild
- **Testing tools**: JUnit, Selenium, Cucumber, QUnit
- **Repository management**: Nexus, Artifactory, Docker hub
- **Continuous integration**: Jenkins, Bamboo, TeamCity, Visual Studio
- **Configuration provisioning**: Chef, Puppet, Ansible, Salt
- **Release management**: Visual Studio, Serena Release, StackStorm
- **Cloud**: AWS, Azure, OpenShift, Rackspace
- **Deployment management**: Rapid Deploy, Code Deploy, Elastic box
- **Collaboration**: Jira, Team Foundation, Slack
- **BI/Monitoring**: Kibana, Elasticsearch, Nagios
- **Logging**: Splunk, Logentries, Logstash
- **Container**: Linux, Docker, Kubernetes, Swam, AWS, Azure

We will explore DevOps frameworks, maturity models, and the application of these tools in the respective context elaborately in coming chapters.

Benefits of DevOps

Non-adherence to DevOps practices would be challenging for an organization, for the following reasons:

- High deployment effort for each of the development, QA, and production systems
- Complex manual installation procedures are cumbersome and expensive
- Lack of a comprehensive operations manual makes the system difficult to operate
- Insufficient trace or log file details makes troubleshooting incomplete
- Application-specific issues of performance impact not assessed for other applications
- SLA adherence, as required by the business application, would be challenging
- Monitoring servers, filesystems, databases, and applications in isolation will have gaps
- Business application redundancy for failover is expensive in isolation

DevOps adoption and maturity for big data systems will benefit organizations in the following ways:

- DevOps processes can be implemented as standalone or a combination of other processes
- Automation frameworks will improve business efficiency
- DevOps frameworks will help to build resilience into the application's code
- DevOps processes incorporate SLAs for operational requirements
- The operations manual (runbook) is prepared in development to aid operations
- In matured DevOps processes, runbook-driven development is integrated
- In DevOps processes, application-specific monitoring is part of the development process
- DevOps planning considers high availability and disaster recovery technology
- Resilience is built into the application code in-line with technology features
- DevOps full-scripted installation facilitates fully automate deployment
- DevOps operation team and developers are familiar with using logging frameworks
- The non-functional requirements of operability, maintenance, and monitoring get sufficient attention, along with system development specifications
- Continuous integration and continuous delivery eliminates human errors, reduces planned downtime for upgrades, and facilitates productivity improvements

Summary

In this chapter, we have learned about the concepts of DevOps, key market trends, along with business drivers leading to DevOps adoptions across systems like big data, cloud, data sciences, and so on. The business scenarios with ample examples for application of DevOps were presented. DevOps adoption with popular open source tools as detailed in coming chapters will enhance multifold productivity benefits to organizations.

In the next chapter, we will discuss the concepts of big data, in-memory technology, data science technologies, and data visualization techniques.

2
Introduction to Big Data and Data Sciences

We will discuss the key technology concepts such as In-memory computing and NoSQL databases that are the building blocks of the big data subject, along with concepts of data visualization and data sciences. These concepts are enriching for novice readers and will be appreciated while building big data applications in future sections. Readers familiar with these concepts can skip this section if they wish to. In this chapter we will cover the following topics:

- Big data attributes
- In-memory concepts
- NoSQL databases
- Data visualization
- Data science

Big data

Big data has many interpretations and definitions across industries, academics, organizations, and individuals. It's a very broad and evolving field, and many organizations are adopting big data in some shape or form to supplement their existing analysis and business tools. Big data systems are primarily used to derive meaningful value and hidden patterns from data. They are also implemented to supplement different types of traditional workloads for economies of scale that lower costs. The three key sources of big data are people, organizations, and sensors.

Big data systems are characterized by a few attributes such as volume, velocity, a variety of data, and value; additional characteristics are veracity, validity, volatility, and visualization:

- **Value**: The ultimate objective of big data is to generate some business value and purpose for the company by doing all the analysis with a big data project.
- **Volume of data**: Big data system volumes can scale as per business needs to gigabytes, terabytes, petabytes, exabytes, zettabytes, and so on. Each business has unique volume needs; for example, an **Enterprise Resource Planning** (**ERP**) system could run into gigabytes of data, while **Internet of Things** (**IoT**) and machine sensor data could run into petabytes.
- **Velocity of data**: The speed at which the data is accessed could be batch jobs, periodic, near-real-time, real-time data from web server logs, streaming data of live videos and multimedia, IoT sensor information, weather forecasts, and so on. We can correlate the quantity of SMS messages, Facebook status updates, or credit card swipes being sent every minute of every day by an organization.
- **Variety of data**: Variety of data is one of the key ingredients of big data. The data can be of many forms, such as a structured data format similar to a sales invoice statement date, sales amount, store ID, store address, and so on, which can easily fit into traditional RDBMS systems; semi-structured data, such as web or server logs, machine sensor data, and mobile device data; unstructured data such as social media data, including Twitter feeds and Facebook data, photos, audio, video, MRI images, and so on.

  Structured data has a form and rules for a metadata model, and dates follow a specific pattern. However, unstructured and semi-structured data have no predefined metadata model rules. One of the goals of big data is to gather business meaning from unstructured data with technology.

- **Veracity of data**: This is the trustworthiness of data; it should be devoid of bias and abnormalities. It's ensuring the quality and accuracy of data gathered from different source systems, and doing preprocessing quality checks to keep data clean and ensure no dirty data accumulates.
- **Validity of data**: Data should be correct and valid for its intended use, ensuring its appropriateness. Even in traditional data analytics, data validity is crucial for the program's success.
- **Volatility of data**: This is the shelf life of the data, its validity for the time period of intended use. Stale data will not be able to generate intended results for any project or program.

- **Visualization**: Visualization through pictures appeals to the human eye more than raw data in metric or Excel format. Being able to visualize the data trends and patterns from the input data systems or streams till the end analysis is an asset to big data programs.

In the traditional data warehouse, the data is structured, like RDBMS data, and the schema is modeled for the data to be loaded into the database.

Data handling in big data systems from ingestion to persistence, computation, analytics, and so on is quite different from traditional data warehouse systems as data volumes, velocities, and formats are quite divergent from source system ingestion to persistence. These systems require high availability and scalability for staging to persistence and analytics.

Scalability is achieved through cluster-resource pooling of memory, computation, and disk space. New machines can be added to the cluster to supplement the resource needs as per varying workload demands, or increased data volumes with business expansion. High availability is quite important for critical systems performing real-time analytics, production systems, or staging and edge systems holding real-time data. High availability clusters mean ensuring fault-tolerant systems even in the event of hardware or software failures, ensuring uninterrupted access to data and systems.

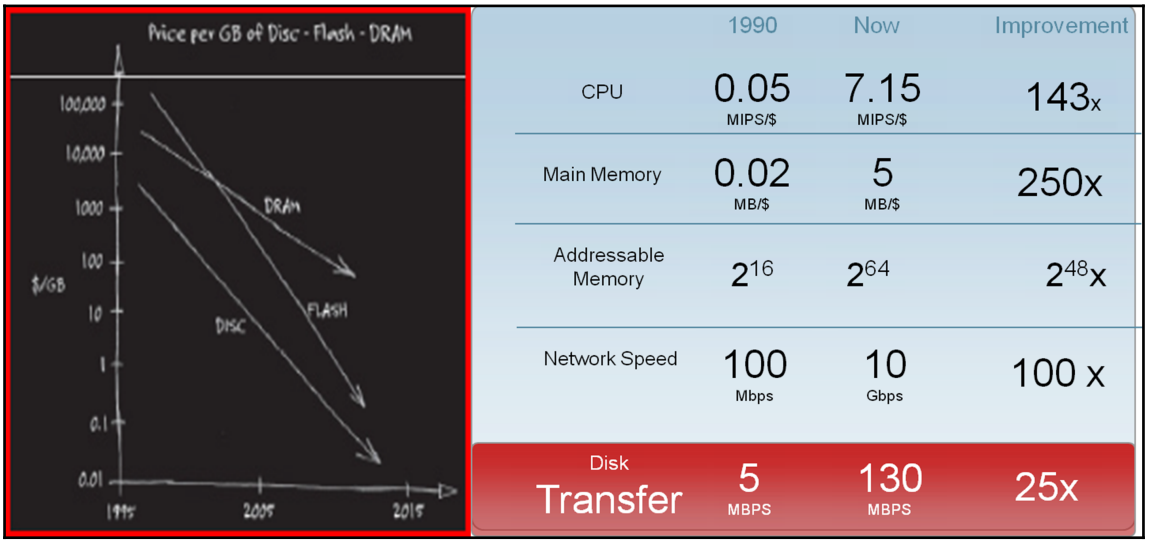
We will be discussing building Hadoop-based clusters as support in `Chapter 4`, *Big Data Hadoop Ecosystems*, including the various industry tools available.

Another prominent big data technology is In-memory computing, which encompasses both software and hardware technology advancements to handle the huge data loads of volumes, velocity, and variety in big data systems. We will discuss these in detail.

In-memory technology

In traditional application development, the disk was the main persistence for data storage. The challenge in this method was that, for business logic and application computation, data was transferred from storage disk to main memory, causing huge I/O overhead. Again, after the computations based on the business logic, the data from aggregation, computational, or analytic results was transferred from CPU and main memory to store, or the data was persisted back to storage disk, causing I/O overhead multiple times.

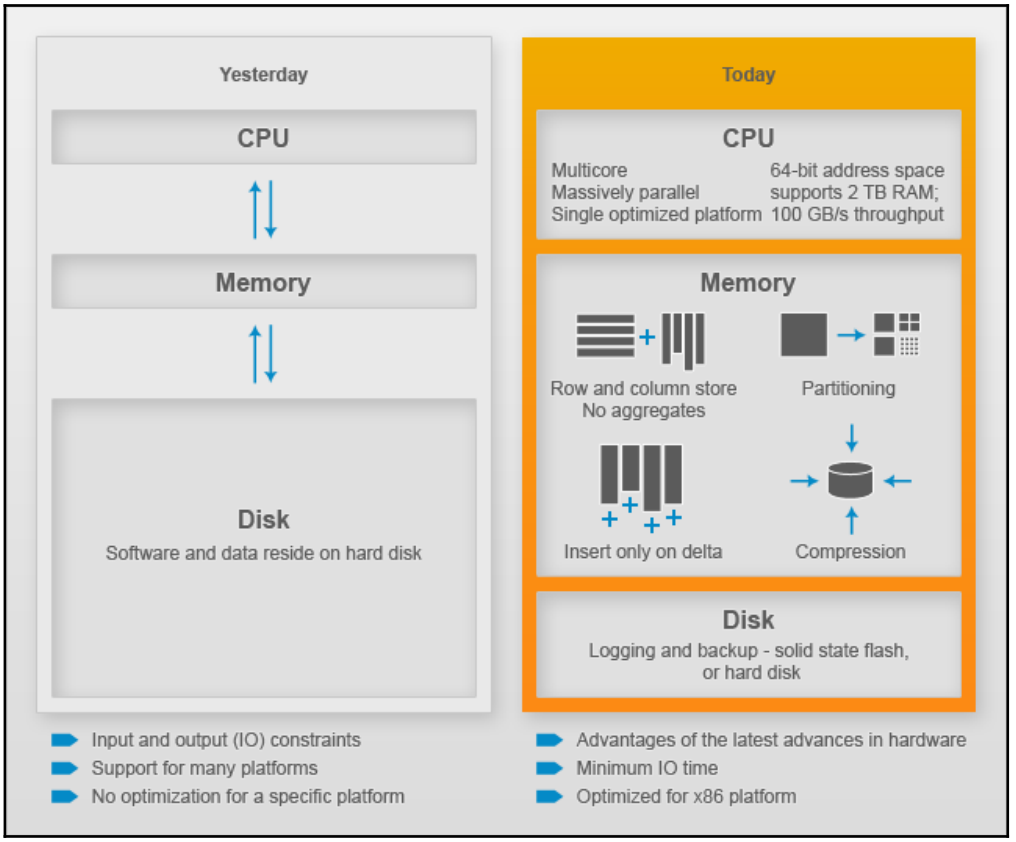
As the following simple illustration shows, disk speed is growing slower compared to other hardware components, while the need for higher performance and speed is increasing day by day:



In-memory database (IMDB)

With In-memory technology, in contrast to traditional disk-based data persistence methods, the complete application requires the data is loaded into the main memory of the system. That makes the applications perform 10 to 20 times faster.

- Data resides permanently in the main memory
- The main memory is the primary *persistence* of data storage
- The disk serves as persistent storage for logging and recovery from the disk
- Main memory access is crucial to performance
- Algorithms/data structures relying on the cache are more efficient in response



Hardware technology advances adopted for In-memory systems

Many hardware advancements have been integrated into modern In-memory computing architecture, such as multi-core architecture and massive parallel scaling.

The In-memory database is optimized to use the capabilities of multi-core processors to enable incredibly fast queries. Processor speed is no longer dependent on clock speed but rather on the degree of system parallelism. Modern server boards have many CPUs with several cores each.

Parallelism can be achieved at different levels, such as from the application level to query execution on the database level. Multi-threaded application processing is handled by mapping each of the queries to a single core, and hence multiple queries are distributed to multiple cores in parallel. Query processing also involves data processing (that is, the database needs to be queried in parallel). In-memory systems distribute the workload across multiple cores of a single system.

Software technology advances adopted for In-memory systems

Software technology advancements have contributed to the development of **In-memory Database** (**IMDB**) engines; they are listed as follows:

- Data compression
- No aggregate tables
- Insert-only tables
- Column, row, and hybrid storage
- Partitioning

Data compression

In-memory technology enables data compression techniques to achieve data compression of up to 20 times. There are multiple algorithms such as bitmaps, run length encoding, dictionary encoding, prefix/suffix encoding, cluster encoding, relative encoding, delta encoding, and indirect encoding. The In-memory engine will apply the most appropriate algorithm or combination of algorithms to achieve the optimum compression ratio and performance.

The objectives of data compression are to reduce the volume of data transfer back and forth from the system as quickly as possible. To accomplish this, use the appropriate algorithms and techniques to minimize the overhead associated with the compression and uncompression of data. Improving overall query performance is also an objective of effective data compression.

No aggregate tables

An In-memory database eliminates the need to maintain expensive pre-aggregate tables, since the data resides in the system memory computations and aggregations are on the fly, since there is no transfer of data back and forth. This eliminates the overhead of maintaining materialized aggregate views, which we can generate in real time.

Insert-only tables

In a typical database, deleted records are performance and overhead intensive. In-memory database instead of deleting a record its marked as obsolete record like in version control system, it not used for computation. So, the overhead associated with deleting a record and re-indexing the rest of records which are performance intensive are eliminated.

Column, row, and hybrid storage

Online Transaction Processing (**OLTP**) applications are row-oriented storage, wherein table data is stored as a sequence of records. Row-based tables and storage are more efficient in OLTP applications for the following reasons:

- Only a single record is processed by the application at any given time
- A single record can be subjected to many selects and/or updates at any point in time
- A complete record (or row) is typically accessed by the application
- Since the column values are many distinct values, compression rates are low
- In OLTP applications, aggregations or fast searching is required
- The configuration tables have a small number of rows

In **Online Analytical Processing** (**OLAP**) systems such as data warehouses, column storage is used, where aggregate functions play an important role; the entries of a column are stored in contiguous memory locations so the aggregations are quick and efficient in the column store. Column storage or tables are beneficial in the following situations:

- Calculations are typically executed on a single, or a few, columns only
- Based on values of a few columns, the table search is performed
- There are a large number of columns in the table

- Columnar intensive operations, such as aggregate, scan, and so on, are required to be performed on the table rows
- Since the majority of the columns contain only a few distinct values (compared to the number of rows in the table), high compression rates can be achieved

IMDB technology offers higher efficiency due to the usage of hybrid storage. It uses algorithms for selecting the appropriate combination of both row and columnar storage to gain maximum performance efficiency. It also provides users a choice to customize the storage options, such as select or alter columnar or table-wise storage for any specific table. In a few circumstances, using column stores in OLTP applications maximizes efficiency; it requires a balanced and well-understood approach to insertion and indexing column data storage for transaction systems.

Partitioning

Data partitioning is done for enhanced performance, easy management, convenient backup and recovery, and so on. There are several types of data partition techniques.

Partitioning based on some common affinity involves, for example, grouping of data based on equal time zone segments such as months, weeks, years, and partitioning by the size of tables, dimensions, functions, and so on.

Examples are tables preceding 1M records grouped together, tables fetching payroll data grouped together, and so on.

Various techniques are used in data partitioning strategies, such as group partitioning, horizontal partitioning, vertical partitioning, mixed partitioning, and so on.

In-memory database systems are listed, as follows:

- Apache Spark
- Pivot GemFire
- eXtremeDB
- SAP HANA
- IBM SolidDB
- MSSQL server
- Oracle TimesTen
- MemSQL
- VoltDB

In-memory technologies enabled to do business intelligence and visualization tools are listed, as follows:

- Microstrategy
- Tableau
- QlikView
- PowerBI
- TIBCO Spotfire

NoSQL databases

Traditional RDBMS are popular for their structured data, pre-configured schema, and rigid data consistency for transactional enterprise applications, and are characterized by the following attributes:

- Supporting centralized applications, such as ERP systems, which consolidate enterprise data
- Application availability could range from moderate to high availability
- The data velocity supported applications is moderate
- Typically, data input is limited to a few source systems
- The data they handle is primarily structured in nature
- The databases support complex and nested transactions
- The primary expectation is to scale up to support the read operations for multiple concurrent users
- Support moderate data volumes with cache and purge features

Most modern day applications are based on NoSQL databases to create flexible data schema, schema on read, or no schema to design web and cloud-based systems effectively. The key requirements are the ability to process very large volumes of data and quickly distribute that data across computing clusters to enable fast changes to applications that are continually updated. Traditional RDBMS systems are unable to cater to large-scale database clustering in cloud and web applications.

NoSQL database systems are designed for the following scenarios:

- Support decentralized applications which are spread across multiple locations such as web applications, mobile applications, IoT, and so on
- The applications are continuously available and can't afford downtime
- They support high-velocity data, which could be from devices, sensors, and so on
- The data ingested is not confined to a single location and could range to multiple locations
- Data forms are structured, semi-structured, and unstructured
- The transaction types are mostly simple; however, they maintain high data volumes and retain data for a very long time
- Scalability for systems with intensive write and read operations and data volumes
- Concurrency support for number of users

There are four types of NoSQL databases, with specific purposes:

- **Key-value database**: It is also called a **key-value store**. It stores data without the schema. The name derives from the fact that data is stored as an indexed or unique key and associated value. They are highly scalable and provide session management and caching in web applications, high performance, and schema-less design. This type of database is popular, and examples include Cassandra, DyanmoDB, **Azure Table Storage** (**ATS**), Riak, BerkeleyDB, Aerospike, and MemchacheDB.
- **Document databases:** These databases store semi-structured data such as document descriptions and information. Each document is assigned a unique key, which is used to retrieve the document data. An advantage is data records can be created and updated for storing, retrieving, and managing document based on unique key. For web-based applications, the data exchange is through JavaScript and **JavaScript Object Notation** (**JSON**), which is popular for content management and mobile application data handling. Popular examples of document databases are Couchbase server, CouchDB, DocumentDB, MarkLogic, and MongoDB.

- **Wide-column stores**: These are designed to organize data tables as columns instead of as rows. Wide-column stores can be found both in SQL and NoSQL databases, and offer very high performance and a highly scalable architecture. Wide-column stores are faster than traditional relational databases and can query large data volumes very fast, hence they are used for intensive data processing systems such as recommendation engines, catalogs, fraud detection, and so on. A few popular examples of wide-column stores are Google Bigtable, Cassandra, and HBase.
- **Graph database**: Graph data stores represent data as relationship models and organize data as nodes. They are designed to represent data relationships as interconnected elements, such as a graph with a pictorial representation of the number of relations between them. An example is connections between nodes; the graph data model can evolve over time so is used with a flexible schema. Graph databases are effectively used in systems that must map relationships, such as friends connected on a social network, reservation systems, or customer relationship management. Examples of graph databases include AllegroGraph, IBM Graph, Neo4j, and Titan.

The selection of NoSQL DBs are based on business requirements like building big data and web applications demanding high performance, scalability, flexibility, functionality, and complexity.

- **Architecture**: Basic requirements of web and cloud-based systems are constant uptime, multi-geography data replication, predictable performance, and so on. The architecture should be designed to support diverse workload patterns, technology to support ingesting data of high volume, variety, and velocity. Ability to perform transactions in real time, run real-time analytics on data lake or multiple systems. There are a few based on the master/slave model, such as MongoDB, and a few are masterless, where all nodes in database cluster perform the same role, as in Cassandra.
- **Data model scalability**: The data models are based on types, like a wide-row tabular store, document-oriented, key-value, or graph, and so on to scale very rapidly and elastically, in order to be applicable to all situations and times, scaling across multiple data centers and even to the cloud if needed.
- **Data distribution model**: Based on their inherent architectural differences, NoSQL databases perform differently for the reading, writing, and distribution of data. For example, Cassandra is popular to support writes and reads on every node in a cluster, and can replicate and synchronize data between many data centers across cloud providers.

- **Development model and support**: Based on their development APIs, NoSQL databases might have unique SQL-like languages (for example, Cassandra's CQL). Vendor or community-related support for technology will be an invaluable resource for the individuals and teams managing the environment.
- **Performance and continuous availability**: In an online world, big data must perform at extremely high velocities under varying workloads, so databases must scale and perform to support applications and environments, as nanosecond delays can cost you sales. Revenue generating systems like flight reservation systems (and data) need to be available 24*7 , as businesses can't afford any downtime.
- **Manageability and cost**: There is a need to balance the cost of NoSQL platform development and operational complexity, to be viable to businesses from a cost and usage perspective. Deploying a well-structured NoSQL program provides all of the benefits already listed, while also lowering operational costs.

Benefits of NoSQL

As we have seen, for cloud applications and decentralized systems NoSQL databases are the de facto databases to use, primarily as NoSQL databases offer many robust features and benefits compared to other database management systems, such as the following:

- **Continuous availability**: A database should be available 24/7 or 99.999% of the time, even during standing infrastructure outages.
- **Economical/minimal cost of operations**: Investment and expenses related to maintenance; scalability of the NoSql systems should be affordable to businesses and compatible to support existing applications.
- **Scalable architecture**: Web applications support multiple geographies, so architectural features of the database should be resilient and scalable. Data manipulation features and capabilities are to be supported for multiple concurrent enterprise systems.
- **High responsiveness**: Cloud and web-based applications are low latency applications that must respond as per business demands quickly. These applications should perform under various conditions like varying and mixed workloads and multiple data models, and integrations with third-party tools and applications.
- **Elastically scalable**: The database and supporting applications should be designed for current and future data needs linearly and predictably, and be operationally mature.

Data visualization

A picture is worth a thousand words. Data visualization is the representation of data in the form of graphs, charts, pictures, or any other visual means. It helps users to quickly understand and analyze the complex data patterns, variations, and deviations associated with data. As we can all agree, skimming through multiple records of numerical data would be very tiring. The ability to graphically visualize the same data would be a very effective, efficient, and time-saving way to identify the patterns we need. There are many tools that help with data visualizations; the simplest forms are bar charts, pie diagrams, and so on.

In big data visualizations, data patterns play an important role. A few benefits of data visualization are as follows:

- A quick, easy way to convey concepts in a universal manner
- Identify areas that need attention or improvement
- Clarify which factors influence customer behavior
- Help you understand which products to place where
- Predict sales volumes by segment and time period

Data representation and visualization methods are listed, as follows:

- Graph plots:
    - Area graph
    - Bar chart
    - Bubble chart
    - Density plot
    - Error bars
    - Histogram
    - Line graph
    - Multi-set bar chart
    - Parallel coordinates plot

  - Point and figure chart
  - Population pyramid
  - Radar bar chart
  - Radial column chart
  - Scatterplot
  - Span chart
  - Spiral plot
  - Stacked area graph
  - Streamgraph
- Diagrams:
  - Flowchart
  - Illustration diagram
  - Timeline
  - Tree diagram
  - Network diagram
  - Venn diagram
  - Pie chart
  - Pictogram chart
  - Bubble map
  - Treemap
- Tables:
  - Calendar
  - Gantt chart
  - Heatmap
  - Stem and leaf plot
  - Tally chart
  - Timetable
- Maps:
  - Choropleth map
  - Connection map
  - Dot map
  - Flow map

Data visualization application

Data visualization is quite valuable to preview the raw data in form of pictures and graphs as presentable formats. Few tools to aid the same are listed following:

- **Time series***:* It's used to plot the performance trend of a single variable, such as the sales of a particular car model over a time period of 5 years, with a line chart (`https://en.wikipedia.org/wiki/Line_chart`) to demonstrate the trend.
- **Nominal comparison**: Comparing general trends, say in sales volume by car model with a bar chart.
- **Ranking**: Used to compare car sales between different locations over the period of 5 years for a particular model. The performance over a period of time for different segments is represented by a bar chart (`https://en.wikipedia.org/wiki/Bar_chart`) to show the comparison across the zones on sales.
- **Part-to-whole***:* Used to measure a ratio to the whole (that is, a percentage out of 100%). The percentage of students securing *A* grade in a class is represented by a pie chart (`https://en.wikipedia.org/wiki/Pie_chart`) or bar chart to show the comparison ratio.
- **Frequency distribution**: Used to evaluate the trend of a particular variable for a given interval, such as the number of years in which the property price change is between intervals such as 0-10%, 11-20%, and so on. A histogram (`https://en.wikipedia.org/wiki/Histogram`) or bar chart may be used. A box plot (`https://en.wikipedia.org/wiki/Box_plot`) helps visualize key statistics such as median, quartiles, outliers, and so on for the distribution.
- **Correlation**: Used to compare dependency of trend movement between two variables (*X* and *Y*) to determine if they tend to move together or in opposite directions, for example, plotting unemployment (*X*) and GDP growth (*Y*) for a time period in months with a scatter plot (`https://en.wikipedia.org/wiki/Scatter_plot`).
- **Deviation**: Used for comparison of the actual versus the reference amount, such as the comparison of actual versus budget expenses for several portfolios of a business for a given time period, represented by a bar chart.
- **Geographic or geospatial:** Used to compare the spread of a variable across a map or geographical layout, such as the store locations by state. A cartogram (`https://en.wikipedia.org/wiki/Cartogram`) is a typical graphic used to plot a number of cars on the various floors of a parking lot.

A few of the commercial data visualization tools are as follows:

- Tableau
- QlikView
- Microstrategy
- Microsoft PowerBI
- TIBCO Spotfire
- Looker
- DOMO
- ZOHO
- Information Builders
- Chart

Some data visualization open source tools are as follows:

- Google Charts
- Js
- Ploty
- Chart Js
- Charted
- Leaflet
- FusionCharts

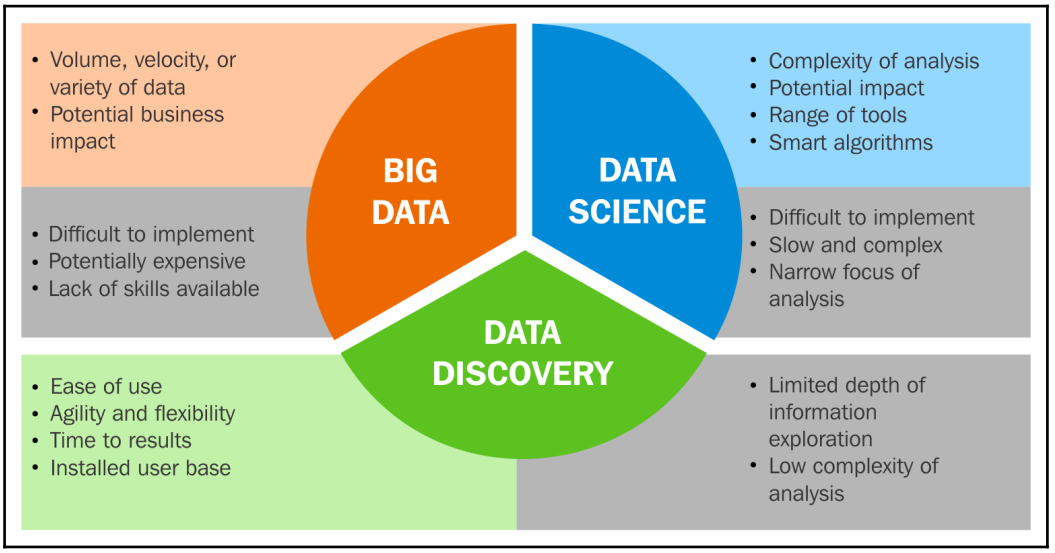
Data science

A few similar terms associated with data science include:

- **Data science**: An interdisciplinary field also known as data-driven science. There are few stages such as data discovery, the study of information originating from varied data sources in different forms, structured or unstructured, ingesting the data, and applying scientific methods and processes to gain insights and apply knowledge for the creation of value-added business and IT strategies.

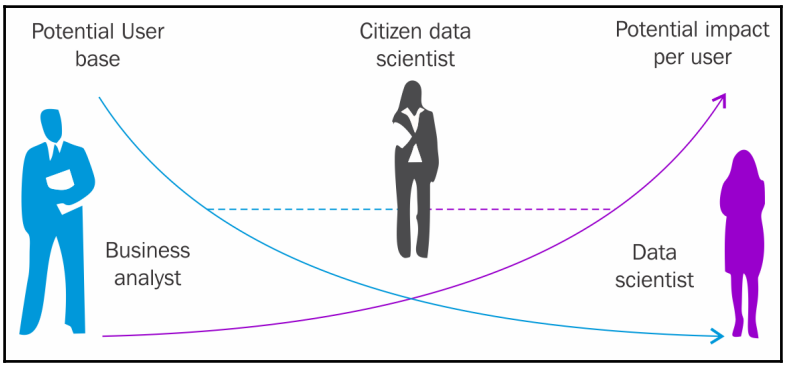
- **Data mining:** Data mining is a broad term for the practice of trying to find patterns in large sets of data. It is the process of trying to categorize a mass of data into a more understandable and cohesive set of information. Mining large amounts of structured and unstructured data to identify patterns can help an organization rein in costs, increase efficiency, recognize new market opportunities, and increase their competitive advantage.
- **Machine learning:** Machine learning is the study and practice of designing systems that can learn, adjust, and improve based on the data fed to them. This typically involves the implementation of predictive and statistical algorithms that can continually zero in on *correct* behavior and insights as more data flows through the system.

The interdependency and interfacing of data science with other interfacing sections of big data and data discovery are depicted, as follows:

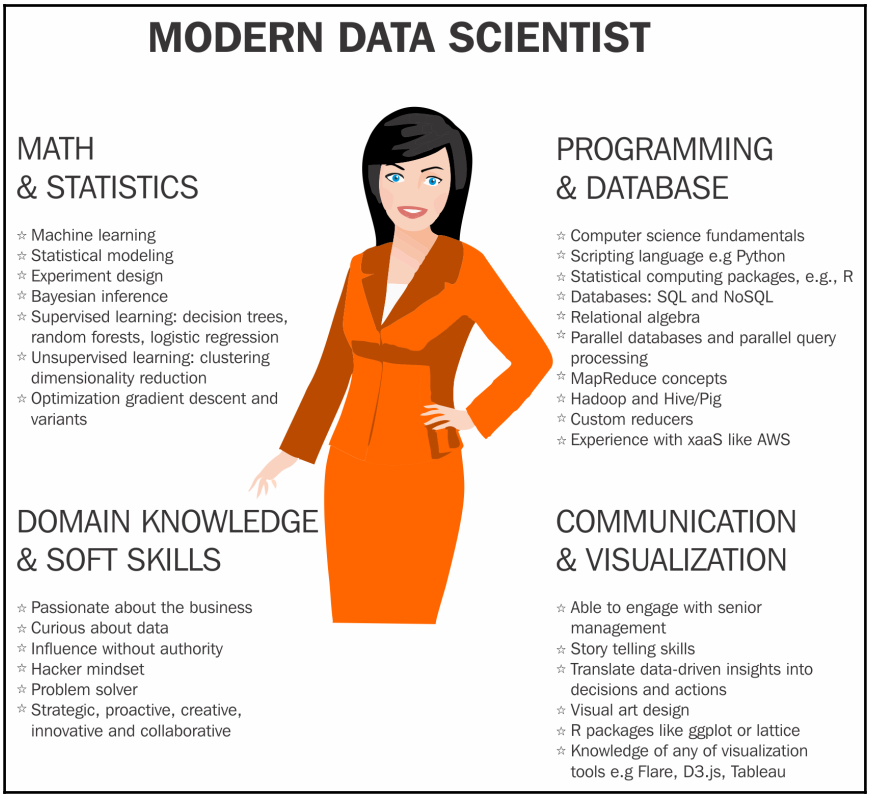


Big data discovery is the combination of big data, data science, and data discovery.

Gartner analysts have defined a new evolving role of **citizen data scientists** who, using these tools, will marry the skills of traditional business analysts with some of the expertise of expert statisticians.



The role of a modern data scientist requires a mixture of broad, multidisciplinary skills ranging from an intersection of mathematics, statistics, computer science, communication, and business understanding. A data scientist's most basic, universal skill is the ability to integrate systems and derive meaningful and reproducible patterns from the underlying data. More enduring will be the need for data scientists to communicate in a language that all their stakeholders understand, and to demonstrate the special skills involved in storytelling with data, whether verbally or visually, or ideally both.



Summary

In this chapter, we have learned about the concepts of big data, In-memory technology, NoSQL databases, data visualization, and data science.

In the next chapter, we will discuss the concepts of DevOps frameworks and best practices.

3
DevOps Framework

In this chapter, we will learn about different DevOps processes, frameworks, and best practices. We will present DevOps process maturity frameworks and progression models with checklist templates for each phase of DevOps. We will also become familiar with Agile terminology and methodology and the benefits accrued by an organization by adopting it. In this chapter, we will cover the following topics:

- DevOps process
- DevOps progression frameworks
- DevOps maturity models
- DevOps best practices
- Agile and DevOps

DevOps process

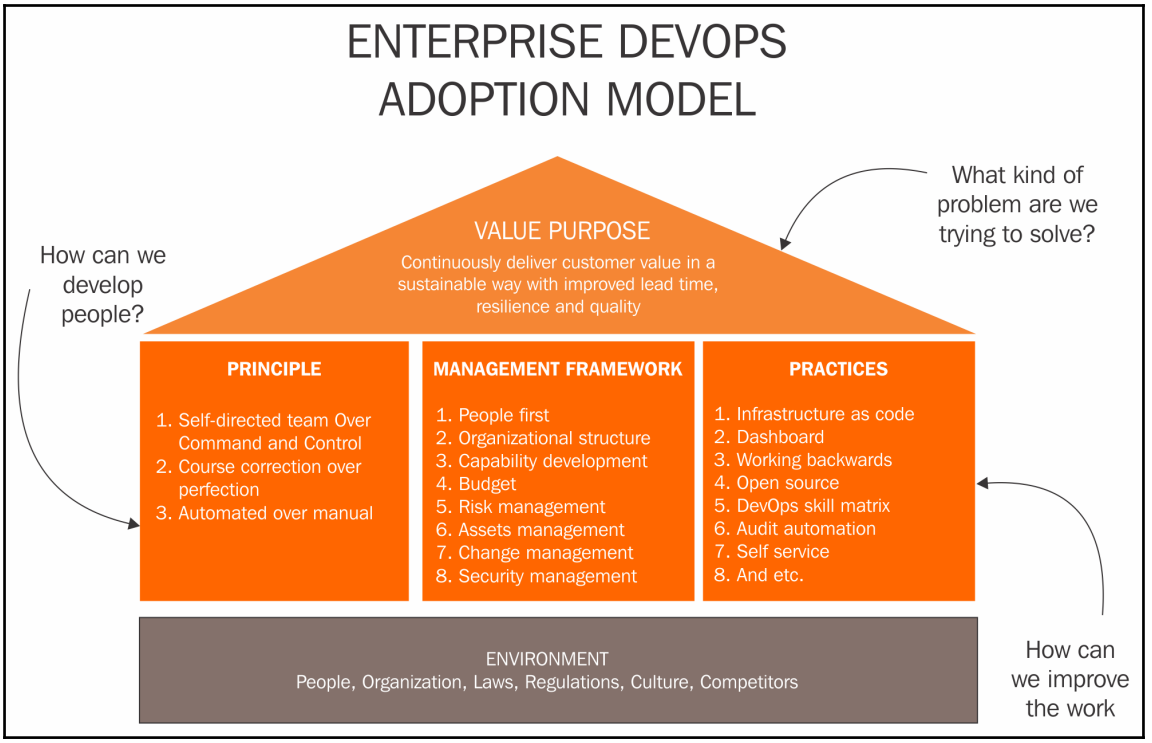
The DevOps standard processes prescribed across the industry and adopted by organizations are listed here; we will discuss them in detail:

- Source code management
- Source code review
- Configuration management
- Build management
- Repository management
- Release management

- Test automation
- Continuous integration
- Continuous delivery
- Continuous deployment
- Infrastructure as Code
- Application performance monitoring
- Routine automation/continuous improvement
- DevOps frameworks--under DevOps frameworks, we will study the life cycle models, maturity states, progression and best practices frameworks, and also Agile methodology:
    - DevOps project life cycle
    - Maturity states
    - Progression frameworks
    - DevOps practices frameworks
    - Agile methodology

DevOps best practices

The adoption of DevOps best practices will help to align people and progress towards organizational goals. DevOps offers multiple process frameworks at every stage of software development. Full-scale implementation of DevOps in an organization requires a cultural shift integrating departments, people, and the process of software life cycles. It enables organizations to move higher on the maturity road map in terms of compliance and process adherence:



DevOps process

The following are the DevOps standard processes prescribed across the industry and adopted by organizations, discussed in detail.

Source Code Management (SCM)

Source code management systems have been in use for decades, offering many functions and benefits. However, integrating them with DevOps processes offers robust integration and automation. A source code management system enables multiple developers to develop code concurrently across multiple development centers spread across diverse geographies. SCM helps in the management of code base and version control at the file level, so developers don't overwrite each other's code, and they have ability to work in parallel on files in their respective branches.

Developers merge their code changes to the main or sub branch which can be tracked, audited, enquired for bug fixes, and rolled back if needed. Branching is an important functionality of SCM, multiple branches of the software are maintained for different major and minor releases, tracking the features and bug fixes across various release versions. SCM enables managing process adherence across environments of development, test and production, facilitating entire software life cycle management from development to support.

The DevOps process framework emphasizes the adoption of SCM for accruing the following benefits for the organization:

- Coordination of services between members of a software development team
- Define a single source of truth for any version, minor or major
- Review changes before implementing
- Track co-authoring, collaboration, and individual contributions
- Audit code changes and rollback facility
- Incremental backup and recovery

SCM tools prevalent in the market are as follows:

- IBM ClearCase
- Perforce
- PVCS
- Team Foundation Server
- Visual Studio Team Services
- Visual SourceSafe

Open source SCM tools are as follows--their popularity is also attributed to DevOps' widespread adoption:

- **Subversion** (**SVN**)
- **Concurrent Version System** (**CVS**)
- Git
- SCCS
- Revision control systems
- Bitbucket

Code review

Code reviews are an important process to improve the quality of software instances before they are integrated into the main stream. They help identify and remove common vulnerabilities such as memory leaks, formatting errors and buffer overflows. Code review or inspection can be both formal and informal. In a formal code review, the process is through multiple methods such as formal meetings, and interactions to review the code line by line. Informal code reviews can be over the shoulder, emails, pair programming where a few authors codevelop, or tool assisted code reviews--these are also called **code walkthroughs**.

A code review process framework benefits the organization as follows:

- Collaboration between software development team members
- Identification and elimination of code defects before integration
- Improvement of code quality
- Quick turnaround of development cycle

Proprietary tools for code review automation:

- Crucible
- Collaborator
- Codacy
- Upsource
- Understand

Open source tools for code review automation:

- Review board
- Phabricator
- Gerrit
- GitLab

Configuration Management

Configuration Management (**CM**) is the broad subject of governing configuration items at enterprise level, as per **Infrastructure Library** (**ITIL**); even the **configuration management database** (**CMDB**) is part of the CM strategy. Configuration management includes identification, verification, and maintenance of configuration items of both software and hardware, such as patches and versions. In simple terms, it's about managing the configuration of a system and ensuring its fitness for its intended purpose. A configuration management tool will validate the appropriateness of the configurations on the system as per the requirements and its interoperability between systems. A common example is to ensure the code developed on a development system is effectively functional on a QA (test) system and production systems. Any loss of configuration parameters between the systems will be catastrophic for the application's performance.

As per DevOps, the benefits of incorporating configuration management processes and tools for an organization can be summarized as follows:

- Facilitates organizations with impact analysis due to the configuration change
- Allows automated provisioning on different systems such as dev, QA, and prod
- Facilitates audit, account, and verification of the systems
- Reduces redundant work by ensuring consistency
- Effectively manages simultaneous updates
- Avoids configuration related problems of a single version of the truth
- Simplifies coordination between team members of development and operations
- It is helpful in tracking defects and resolving them in time
- Helps in predictive and preventive maintenance

A few popular configuration management tools for infrastructure are as follows:

- BMC Software's Atrium
- Hewlett Packard Enterprise's Universal Configuration Management Database

A few popular software configuration management tools are as follows:

- Chef
- Puppet
- Ansible
- Salt
- Juju

Build management

Build management is the process of preparing a build environment to assemble all the components of a software application as a finished, workable product, fit for its intended purpose. The source code, the compilers, dependencies with hardware and software components, and so on, are compiled to function as a cohesive unit. Builds are manual, on demand and automatic. On-demand automated builds reinitiate with a script to launch the build and are used in few cases. Scheduled automated builds are the case with continuous integration servers running nightly builds. Triggered automated builds in a continuous integration server are launched just after being committed to a Git repository.

As per DevOps, the benefits of build management processes and tools for an organization can be summarized as follows:

- The vital function of ensuring software is usable
- Ensures reusability and reliability of the software in client environments
- Increases the efficiency and quality of software
- It's also a regulatory requirement

A few build tools that are in use are as follows:

- Ant
- Buildr
- Maven
- Gradle
- Grunt
- MSbuild
- Visual Build
- Make (CMake/QMake)

Artifacts repository management

A build Artifacts repository manager is a dedicated server for hosting multiple repositories of binary components (executables) of successful builds. By centralizing the management of diverse binary types, it reduces the complexity of access along with their dependencies.

The benefits are as follows:

- Manage artifact life cycles
- Ensure builds are repeatable and reproducible
- Organized access to build artifacts
- Convenient to share builds across teams and vendors
- Retention policies based on artifacts for audit compliance
- High availability of artifacts with access controls

A few repository tools that are in use are as follows:

- Sonatype Nexus
- JFrog Artifactory
- Apache Archiva
- NuGet
- Docker hub
- Pulp
- Npm

Release management

Release management is the process of a software life cycle to facilitate a release's movement from development, testing, and deployment to support/maintenance. It interfaces with several other DevOps process areas in the SDLC.

Release management has been an integral part of the development process for decades. However, its inclusion into the DevOps framework makes a complete cycle for automation.

Release management is an iterative cycle initiating by a request for the addition of new features or changes to existing functionality. Once the change is approved, the new version is designed, built, tested, reviewed, and after acceptance, deployed to production. During the support phase, there could be a possibility of enhancement or performance leading to the initiation of a new development cycle.

The benefits of adopting release management are as follows:

- Product life cycle holistic management, tracking and integrating every phase
- Orchestrate all the phase activities--development, version control, build, QA, systems provisioning, production deployment, and support

- Track the status of recent deployments in each of the environments
- Audit history of all activities of work items that are associated with each release
- The automation of release management relies on automating all of its stages
- Teams can author release definitions and automate deployment in repeatable, reliable ways while simultaneously tracking in-flight releases all the way to production
- Fine grain access control for authorized access and approval for change

A few release management tools are:

- Electric Cloud
- Octopus Deploy
- Continuum
- Automic
- Quikbuild
- UrbanCode Release
- CA Service Virtualization (LISA)
- BMC Release Process Management
- Plutora Release
- CA Release Automation
- Serena Release
- MS Visual Studio
- StackStorm
- Rally

Test automation

Testing manually for every possible scenario is tedious, labor intensive, time consuming and expensive. Test automation, or automatic testing, is for running test cases without manual intervention. Though not all test cases qualify to be automatically run, the majority can be scheduled. Automation is achieved by running the test cases with an automation tool or through the scheduling of automation scripts. Recent test data is used as input and the results are captured for analysis. The goal of test automation is to supplement manual testing by reducing the number of test cases to be run manually--not to replace manual testing all together.

Automation testing is for test cases that are repetitive, monotonous, tedious, and time consuming, that have defined input and boundary conditions. It's not suitable for frequently changing, ad hoc or first time execution test cases. Software automation testing can be based on a few types of frameworks data; keyword, modular, and hybrid.

Testing big data systems encompasses multiple technologies, integrations, frameworks and testing modules such as functional, security, usability, performance, integration testing, and so on.

The benefits of adopting test automation are as follows:

- Improve software quality and responsiveness
- Quick turnaround by substituting manual effort with automation
- Improve the effectiveness of the overall testing life cycle
- Incremental and integration testing for continuous integration and delivery

A few test automation tools are as follows:

- Visual Studio Test Professional
- QTP (UFT)
- SoapUI
- TestDrive
- FitNesse
- Telerik Test Studio
- Selenium
- TestComplete
- Watir
- Robotium

Continuous integration

Continuous integration is a DevOps best practice wherein developers continuously integrate their code in small logical units to a common shared repository with regularity (for example, once a day). The advantage of such a process is the transparency of the code's quality and fitness for its intended purpose. Otherwise, bulk code integration after the lapse of a fixed time period could expose many defects or integration challenges which could be expensive to resolve.

To achieve continuous integration, there are few prerequisites to be implemented, as follows:

- Using a version repository for source code
- Regular code check in schedule
- Automate testing for the code changes
- Automate the build
- Deploy build in preproduction

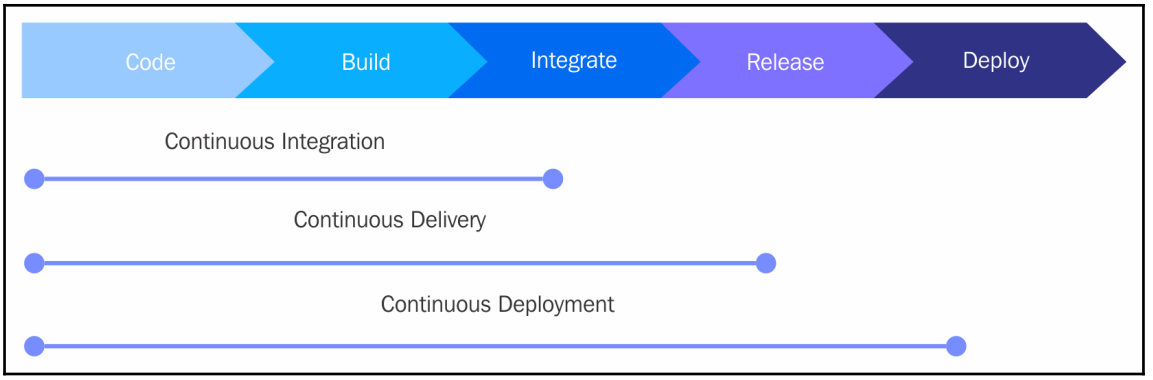
The benefits of continuous integration are as follows:

- Availability of latest code as we commit early and often
- Build cycles are faster as build issues are exposed early with check-ins
- Transparency in the build process means better ownership and lesser defects
- Automating the deployment process leads to quicker turnaround

Some continuous integration tools that are available are as follows:

- Jenkins
- TeamCity
- Travis
- Go CD
- Buddy
- Bitbucket
- Chef
- Microsoft Teamcenter
- CruiseControl
- Bamboo
- GitLab CI
- CircleCI
- Codeship

The following figure represents the roles of continuous integration, delivery, and deployment:



Continuous delivery

Continuous delivery is the next step of continuous integration in the software development cycle; it enables rapid and reliable development of software and delivery of product with the least amount of manual effort or overhead. In continuous integration, as we have seen, code is developed incorporating reviews, followed by automated building and testing. In continuous delivery, the product is moved to the preproduction (staging) environment in small frequent units to thoroughly test for user acceptance. The focus is on understanding the performance of the features and functionality related issues of the software. This enables issues related to business logic to be found early in the development cycle, ensuring that these issues are addressed before moving ahead to other phases such as deployment to the production environment or the addition of new features. Continuous delivery provides greater reliability and predictability on the usability of the intended features of the product for the developers. With continuous delivery, your software is always ready to release and the final deployment into production is a manual step as per timings based on a business decision.

The benefits of the continuous delivery process are as follows:

- Developed code is continuously delivered
- Code is constantly and regularly reviewed
- High-quality software is deployed rapidly, reliably, and repeatedly
- Maximum automation and minimal manual overhead

The tools that perform continuous integration do the job of continuous delivery as well.

Continuous deployment

Continuous deployment is the fully matured and complete process cycle of code change, passing through every phase of the software life cycle to be deployed to production environments.

Continuous deployment requires the entire process to be automated--also termed as automated application release--through all stages, such as the packaging of the application, ensuring the dependencies are integrated, deployment testing, and the production of adequate documentation for compliance.

The benefits of continuous deployment and automated application release are as follows:

- Frequent product releases deliver software as fast as possible
- Automated and accelerated product releases with the code change
- Code changes qualify for production both from a technical and quality view point
- The most current version of the product is ready in shippable format
- Deployment modeling reduces errors, resulting in better product quality
- Consolidated access to all tools, process and resource data leads to quicker troubleshooting and time to market
- Effective collaboration between dev, QA, and operation teams leads to higher output and better customer satisfaction
- Facilitates lower audit efforts owing to a centralized view of all phase activities

The tools that perform continuous integration do the job of continuous delivery as well.

Infrastructure as Code

Infrastructure as Code (**IaC**) is a means to perform infrastructure services through the defining of configuration files. In DevOps' scope, IaC is the automation of routine tasks through code, typically as configuration definition files, such as shell scripts, Ansible playbooks, Chef recipes, or Puppet manifests. It's usually a server and client setup with push or pull-based mechanisms, or agentless through **secured shell** (**SSH**). Many regular tasks on systems such as create, start, stop, delete, terminate, and restarting virtual or bare-metal machines are performed through software. In traditional on-premise systems, many of the system administrative tasks were manual and person dependent. However, with the explosion of big data with cloud computing, all the regular system activities and tasks are managed like any software code. They are maintained in code repositories, and the latest build updates are tested for deployment.

The advantages of IaC are as follows:

- The use of definition files and code to update system configuration is quick
- The version of all the code and changes is less error prone and has reproducible results
- Thorough testing of the deployment with IaC and test systems
- Smaller regular changes are easy to manage, bigger infrastructure updates are likely to contain errors that are difficult to detect
- Audit tracking and compliance are easy with definition files
- Multiple servers update simultaneously
- System availability is high, with less down time

Some tools for IaC are as follows:

- Ansible tower
- CFEngine
- Chef
- Puppet
- SaltStack

Routine automation

Every organization aims to automate routine, repetitive tasks; in fact the survival of most companies and software products is based on the degree to which they automate. ERP systems, data visualization, domain applications, data analytics, and so on; almost all segments are potential areas for automation. A few sections to automate are infrastructure (deployment, patching scalability), applications (development, integration, builds, delivery, deployment), load balancers, feedback, and defects/errors management.

There are several tools to automate each segment, as we have seen in the previous sections; we will explore their application in coming chapters.

Key application performance monitoring/indicators

Performance metrics are part of every tool, product and service. Accordingly, organizations are ever vigilant of the performance metrics monitoring of their applications, products and services. To achieve high-quality output for any product, achieving a high degree of standard in process and metrics is prerequisite. There are many parameters to gauge performance metrics, such as, for example, applications or hardware systems availability or uptime versus downtime and responsiveness, tickets categorization, acknowledgement, resolution time lines, and so on.

DevOps is all about measuring the metrics and feedback, with continuous improvement processes.

Several tools are available for application monitoring for various needs; we will cover the most appropriate and applicable tools in the context of the DevOps framework in further sections of this chapter.

DevOps frameworks

Under DevOps frameworks we will study the life cycle models, maturity states, progression and best practices frameworks, as well as Agile methodology.

Accomplishing DevOps maturity is a gradual progression to being well structured and planned, as stated in the following stages.

DevOps maturity life cycle

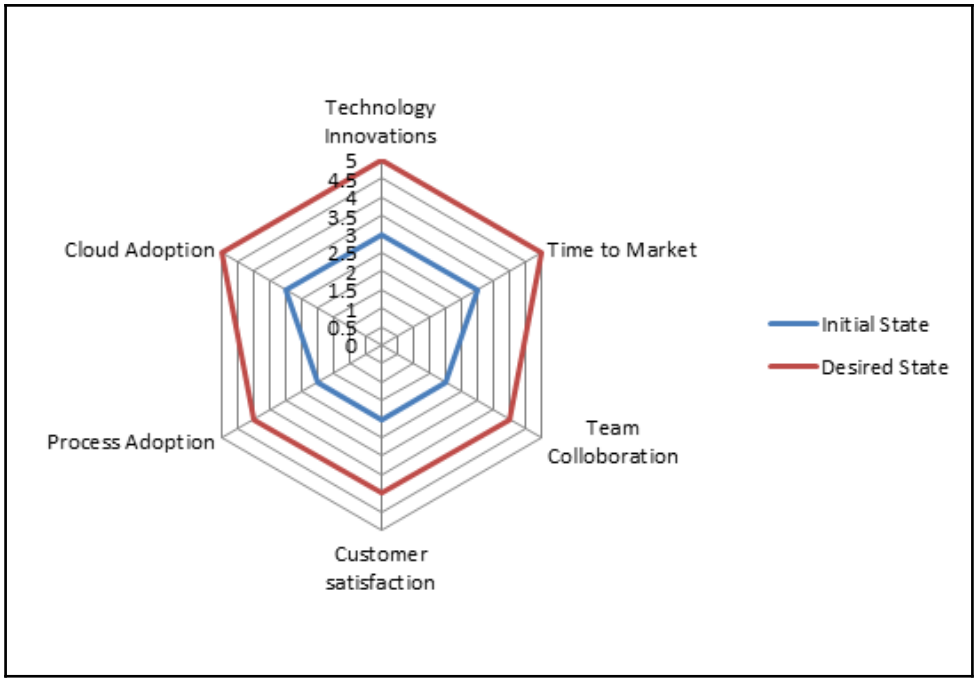
DevOps project phases are on lines of the software development life cycle as described here. We will dwell on each phase in detail:

- **Discovery and requirements phase:** The DevOps discovery phase is a highly interactive project phase for gathering inputs and feedback on the current state of process, frameworks and tools from key stakeholders. Templates and checklists are used to capture the inputs. The timeline for the phase depends on the availability of key stakeholders, the existence of requisite documents, and the complexity of the processes to explore. Discovery phase deliverables are as follows:
    - Templates detailing the current state of process, tools, frameworks
    - Signoff from key stakeholders on the details collated
    - Existing best practices and DevOps methods
    - Existing challenges, constraints as applicable
    - Reusable tools, process, artifacts
- **Design print phase:** The design phase is also the architecture phase; it's about producing a blueprint of the target state to accomplish. It's an iterative process of weighing alternatives for tools, and processes arriving at agreement by key stakeholders. The timeline and cost will be base lined and revisited and revised regularly based on new learnings from the project as we move forward towards the target state. The timeline for this phase depends on how acceptable the processes, tools, and budgets are to the key stakeholders. Design phase deliverables are as follows:
    - Target state is agreed upon
    - Baseline of DevOps process to be adopted
    - Baseline of most viable tools to be implemented
    - Baseline agreed timelines and cost

- **Development phase:** Artifacts base lined from the blueprint phase will be inputs for the development phase; the agreed upon process changes, tools to be implemented, frameworks to be adopted, and so on. A detailed project plan covering deliverables, schedules, dependencies, constraints, resource leveling, and so on will be quite handy. Agile scrum methodology will be the framework to implement the DevOps, which will be discussed in detail. The timeline for the development phase will be as per the project plan base lined initially, and revised regularly with the progress of milestones that have been accomplished. Development phase deliverables are as follows:
    - Initial project plan base lined and signoff
    - Incorporating regular feedback till project completion
    - Allocation of resources for each stage
    - Including new skills, methods, process, and tools
    - Work arounds for project risks, constraints, and so on
    - Deliverables as agreed in the project plan
- **Deployment phase:** The DevOps deployment phase is in accordance with the best practices outlined in the DevOps process framework detailed above. It depends on whether the deployment is a process, an application tool, or for infrastructure. The timeline will be evaluated as per experience gained in the development phase. Deployment phase deliverables are as follows:
    - Deployment guide--cutover plan to production
    - Deployment checklist
    - Signoff from key stakeholders
    - Rollback plan
    - Capacity planning
- **Monitoring phase:** Monitors the key performance factors for each phase's performance of development, build, integration and deployment over time duration. It's followed by tracking the defects, bug fixes, user tickets and plans for continuous improvement. Monitoring phase timelines are as per organization need and performance benchmarks. Monitoring phase deliverables are as follows:
    - Operations manual
    - Feedback forms and checklists
    - User guide, support manual
    - Process flow manual
    - Performance benchmark

DevOps maturity map

DevOps adoption is a value-added journey for an organisation. It's not something achieved overnight quickly, but matured step by step over a period of time with manifested results. Like any **Capability Maturity Model** (**CMMI**) or Process Maturity Models, the critical success factors are to be defined for the program's performance objectives. The initial maturity state of key evaluation parameters is agreed upon by key stakeholders. Then the target maturity level of the parameter variables to be accomplished will be defined in the project charter, along with detailed procedure, milestones, budgets and constraints as approved by stakeholders:



DevOps process maturity framework.

DevOps progression framework/readiness model

As discussed in the previous model, DevOps adoption is a journey for an organisation to higher maturity states. In the following table, different practice areas and maturity levels of DevOps at a broad scale are listed. DevOps maturity levels may vary across teams as per their standards, similarly even a common department or division of the same organization may have significantly more varied and advanced practices than others for the same process flow. Enhancing to achieve the best possible DevOps process workflow throughout the entire enterprise should be the end goal for all teams and departments:

PROCESS	FOUNDATIONAL	REPEATABLE	RELIABLE	OPTIMISED
TECHNOLOGY	Usage by team member	Usage by a Department	Usage by few Department	Enterprise wide Usage
TIME TO MARKET	Ad-hoc Release	Periodic Release	Frequent Release	Continuous Release
COLLABORATION	Team Isolated	Team Communicative	Team Collaborative	Team Unified
CUSTOMER SATISFACTION	Personal feedback	Service level feedback	Department level feedback	Organisation level feedback
PROCESS ADOPTION	Adhoc Process	Inconsistent Process	Shared Process	Aligned Process Corporate wide
CLOUD ADOPTION	Team working with VM's	Department level	Few Departments	Cloud fully embraced

DevOps maturity checklists

The process maturity framework, as seen in the preceding sections, is assessed with checklists and discussions. For each of the key focus areas, the detailed findings will indicate the maturity levels.

The findings provide a general estimate of the maturity level and the impact it is causing:

Development Framework :
Development work delivery (Schedule, Quality)
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Clients Interaction (functional Requirements, Status Reports)
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:

Team Collaboration :
Handshake between departments
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Integration Process Maturity
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:

Software Performance:
Rework vs. new functionality or value delivery
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Process Flow Framework :
Build Process workflow
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Deployment process Effectiveness
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:

Key Metrics Summary :
Code quality
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Test Cases Automation and Results
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Applications Monitoring- KPI's
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:

Production Framework :
Identification of defects
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Responsiveness/Performance
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:
Scalability of systems Architecture
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:

Cloud Technology Adoption Progress:
Initial Maturity Observations:
Initial Maturity Level Rating:
Target Maturity Rating:
Process Improvement Observations::
Process Adoption Benefits:
Best Practices:

Agile framework for DevOps process projects

DevOps projects are typically Agile-framework based, for the effective and quick turnaround of the development and implementation process cycle.

Agile software development-based projects have become widely accepted and adopted across the industry. The traditional waterfall model is outdated and unable to keep up with the advantages offered by Agile methodology.

Agile methodology owes its success to its core objectives:

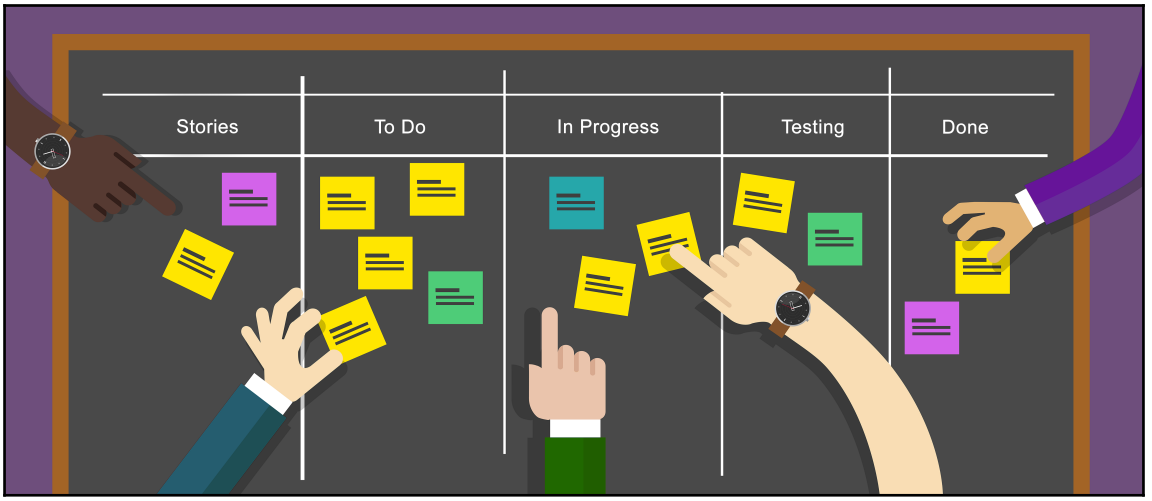
- Individuals and interactions are valued over process and tools
- Working software is valued over comprehensive documentation
- Customer collaboration is valued over contract negotiation
- Change adoption agility is valued over project plan adherence

Agile ways of development

Scrum is the Agile development methodology, focused on features development, from a team comprising of roles such as the following:

- The scrum master is responsible for team setup, conducting sprint meetings, and removing development obstacles
- The product owner creates and prioritizes product backlog, and is responsible for the delivery of the functionality at each sprint iteration cycle
- The scrum team manages and organizes the work to complete in the sprint cycle
- The product backlog is the list of features and requirements of functionality to be developed

The Agile method of development is an incremental and iterative approach for developing user stories, software features or functionality. Customers can see the product features early and make necessary changes, if needed. The development cycle is broken into sprint cycles of two to four weeks, to accomplish units of work. The idea is that smaller cycles can be developed and managed quickly with a team of developers and testers together. The structure and documentation are not important but a working feature of the code is considered valuable. The development process is iteratively accomplished in successive sprint cycles. Bugs identified are fixed at the earliest sprint with successful testing. Regression testing is performed when new functions or logic are developed. User acceptance tests are performed after the sprint cycle to flag the product for release:



The benefits of adopting the best practices of Agile software development are as follows:

- Working software makes the customer satisfied, as he can view the features
- Customers can add change requests at any phase of development
- Quick and continuous delivery of software in weeks
- Projects are built around motivated individuals, who should be trusted
- Sprint teams are highly skilled and efficient in delivery
- Since developers and testers codevelop, bugs are solved within sprint
- The communication mode is effective so quality of product delivered is higher
- Continuous attention to technical excellence leads to good design
- Self-organizing teams focus on optimal architectures, requirements, and designs
- The team is lean and effective, so productivity is maximised

Summary

In this chapter, we understood the application of DevOps processes, frameworks, best practices, and DevOps process maturity frameworks and progression models with checklist templates. We also looked into Agile terminology and methodology.

In the next chapter, we will cover in detail the big data ecosystem, different frameworks, Hadoop clusters, nodes, capacity planning, and so on.

4
Big Data Hadoop Ecosystems

We have discussed the key concepts of **big data technologies** in the preceding chapters. In this chapter, we will cover the building of big data clusters, frameworks, key components, and the architecture of popular vendors. We will discuss big data DevOps concepts in successive chapters.

We will cover the following topics in this chapter:

- Big data Hadoop ecosystems
- Big data clusters
    - Types and application
    - High availability
    - Load balancing
- Big data nodes
    - Master, worker, edge nodes
    - Their roles
- Hadoop frameworks
    - Cloudera CDH Hadoop distribution
    - **Hortonworks Data Platform** (**HDP**)
    - MapR Hadoop distribution
    - Pivotal big data suite
    - IBM open platform
    - Cloud-based Hadoop distribution
    - Amazon Elastic MapReduce
    - Microsoft Azure's HDInsight

- Capacity planning
    - Factors
    - Guidelines

Big data Hadoop ecosystems

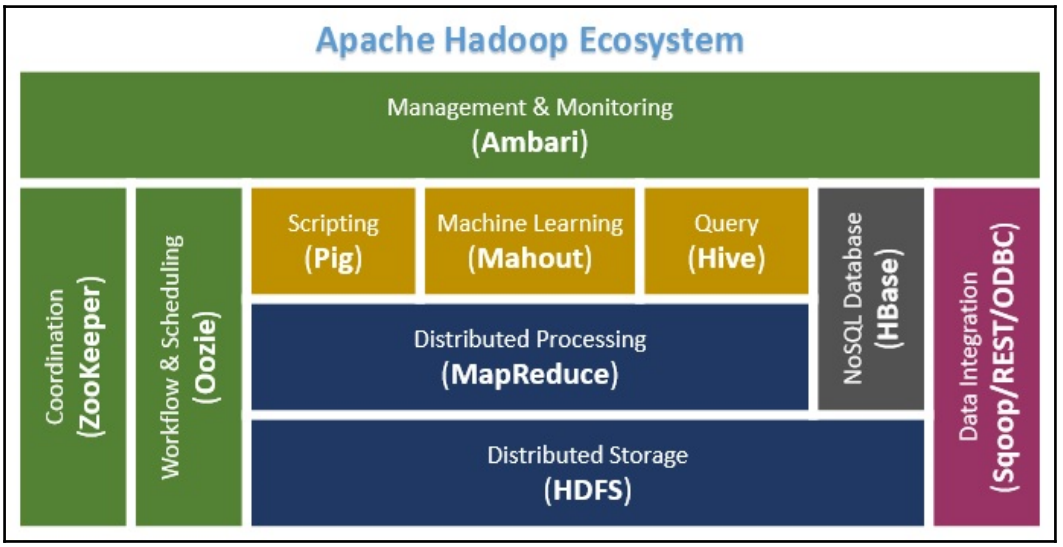
Apache Hadoop is an open source software platform built from commodity hardware and used to scale clusters up to terabytes or petabytes for big data, spanning across thousands of servers. It is highly popular and efficient for distributed data storage and distributed processing of very large datasets. Hadoop offers a full scale of services such as data persistence, data processing, data access, data governance, data security, and operations. A few of the benefits associated with Hadoop clusters are listed as follows:

- **Data scalability**: Big data volumes can grow exponentially to accommodate these large data volumes, Hadoop enables distributed processing of data; each node in the data cluster participates in storing, managing, processing, and analyzing data. The addition of nodes enables quick scaling of clusters to store data at a scale of petabytes.
- **Data reliability**: Hadoop cluster configurations provide data redundancy. For example, in case of accidental failure of one or more nodes, Hadoop cluster management software will automatically replicate the data and processing to the rest of the active nodes. Thus business continuity is assured with the application and data functional, even during the non-availability of a few nodes.
- **Data flexibility**: In traditional relational database management systems, schema tables are created before storing structured data into the system commonly referred to as *schema on write*. Based on the processing application data requirements, diverse data formats can be loaded into Hadoop systems such as structured, semi-structured, or unstructured data. Hence, the schema is dynamically created during the data load, and referred to as *schema on read*.
- **Economical**: Hadoop is open source and built on low-cost commodity hardware, and hence, is more economical than proprietary licensed software.

Organizations adopt the Hadoop system for its versatility and its ability to persist on large data volumes, managing, visualizing and analyzing vast amounts of data quickly, reliably, efficiently, and with a low cost for a variety of data formats, with data governance, workflow, security, and so on.

Inbuilt tools and capabilities in Hadoop ecosystem

Hadoop ecosystem offers many inbuilt tools, features, and capabilities, listed as follows:



- **Data storage**: The **Hadoop Distributed File System** (**HDFS**) provides scalable, fault-tolerant, cost-efficient storage. Hadoop can handle exponential data growth by distributing storage across many nodes; the combined storage capacity can grow with demand while remaining economical per unit of storage. There are other storage managers, such as HBase, Solr, and so on.
- **Data lake**: One of the key strengths of Hadoop is its ability to build a data lake economically. It will be a valuable asset for an organization to store all their relevant data needs, gathered and consolidated from various data sources. For example, in the manufacturing industry the machine maintenance data, inventory data, sales data, machine sensor data on performance, social media data on customer feedback, vendors and suppliers data, weather reports, and so on can be captured regularly as per the requirements of the data lake.

- **Data processing**:
    - **Hadoop ecosystem**: Hadoop ecosystem offers data processing in batch, stream, and hybrid systems.
    - **MapReduce**: MapReduce is an initial processing framework for batch jobs in Hadoop. MapReduce's processing technique follows the map, shuffle and reduce algorithm using key-value pairs. Batch jobs are like monthly telephone invoices for customers.
    - **Stream processing**: Like stock price information and airline reservation data, Apache storm is well suited for processing data in streams.
    - **Hybrid processing systems**: These processing frameworks can handle both batch and stream workloads, such as Apache Spark, Flink, and so on. One use case is the **Internet of Things** (**IoT**), truck sensor data capture, aggregation in the cloud and analytics to derive patterns, and so on.
- **Data access**:
    - **Hadoop**: Hadoop offers multiple ways of accessing and processing the data.
    - **Apache Solr**: Apache Solr provides indexing and search capabilities for the data stored in HDFS.
    - **Hive**: Hive provides data warehouse functionality for Hadoop with a simple SQL-like language called **HiveQL** that provides indexes, making querying faster. A standard SQL programming interface can be used and provides better integration with a few analytics packages such as Tableau, QlikView, and so on.
    - **HBase**: A NoSQL columnar database that provides capabilities such as the columnar data storage model and storage for sparse data to Hadoop systems.
    - **Flume**: Flume collects data from the source systems like a web server log data from Flume **agents** which it then aggregates and moves into Hadoop.
    - **Mahout**: Mahout is a machine learning library with a collection of key algorithms for clustering, classification, and collaborative filtering. These algorithms can be implemented from any processing framework or engines such as MapReduce, and are more efficient for in-memory data mining frameworks such as Spark.

    - **Sqoop**: Sqoop is a valuable tool for transitioning data from other database systems (mainly relational databases) into Hadoop.
    - **Pig**: Pig Latin is a Hadoop-based language that is adept at very deep, very long data pipelines (a limitation of SQL). It is relatively simple and easier to use than SQL.
- **Resource management**: YARN is a great enabler of dynamic resource utilization, an integral part of the Hadoop framework. It manages the increasing workloads of multi-tenant users, running various Hadoop applications without performance impact.
- **Unified administration**: Ambari is a RESTful API that provides a user-friendly web interface for Hadoop administration. It's a tool for Apache Hadoop clusters' provisioning, managing, and monitoring. Provisioning tasks for a Hadoop cluster include installing Hadoop services across multiple hosts and configuring Hadoop services for the cluster. Ambari manages Hadoop cluster services such as starting, stopping, and reconfiguring Hadoop services across the entire cluster through the central management console. Ambari monitors a Hadoop cluster with a dashboard for monitoring the health and status of the Hadoop cluster, and integrates with the Ambari Metrics System for metrics collection and the Ambari alert framework. This alerting system will notify if a node goes down, or disk utilization is higher than a threshold, and so on.
- **Workflow management**:
    - **Oozie**: A workflow processing system that manages and schedules a series of jobs. Jobs can be written in multiple languages such as MapReduce, Pig, and Hive, and linked logically to one another. Oozie allows scheduling dependent jobs as an output of one query to be completed to feed data into the next job as input.
    - **ZooKeeper**: ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications.

- **Comprehensive data security and governance:**
    - Security in Hadoop has three modes of implementation--authentication, authorization, and encryption.
    - Authentication ensures only genuine users have access to the Hadoop cluster. Currently, the tools used are MIT Kerberos, AD, OpenLDAP, and so on.

- Authorization grants users data privileges such as read-only, write, modify, delete, and so on. The currently used tool is Apache Sentry.
- Data encryption ensures data protection from unauthorized access to data, both at rest and in transit. The encryption tool for data at rest is Navigator Encrypt and the tools for data in transit can be implemented by enabling TLS/SSL.
- Access administration of Hadoop systems can be a challenging task in distributed environments that host the individual components of Hadoop on different clusters for optimal performance. For example, in a large production environment, there will be different cluster groups responsible for workflow, for data storage, data analytics, and so on; so managing the respective group access privileges could be a daunting task.

Big data clusters

A Hadoop cluster is a system comprising two or more computers or systems (called nodes). It represents a single unified system for the users. The nodes work together to execute applications or perform other tasks like a virtual machine. There are variants of Hadoop clusters that cater for different data needs. The key features in the construction of these platforms are reliability, load balancing, and performance.

The single node or pseudo-distributed cluster has the essential daemons such as NameNode, DataNode, JobTracker, and TaskTracker, all run on the same machine. A single node cluster is a simple configuration system used to test Hadoop applications by simulating a full cluster-like environment with a replication factor of 1.

A small Hadoop cluster comprises a single master and multiple worker nodes. The master node is comprised of a Job Tracker, Task Tracker, NameNode, and DataNode. A slave or worker node performs the roles of both a DataNode and TaskTracker if required; data-only worker nodes and compute-only worker nodes can be configured. Such nodes are used for full stack development of Hadoop application and projects with a replication factor of 3, such as a multi-node cluster for effective backup.

Multi-node or fully distributed clusters follow the master-slave architecture pattern of Hadoop cluster. The NameNode and TaskTracker daemon runs on the master machine and the DataNode and TaskTracker daemon runs on one or more slave machines. It is deployed for full stack production deployment of the Hadoop application and for projects with a replication factor of 3.

Hadoop cluster attributes

In this section, we will discuss the key attributes of Hadoop clusters, such as load balancing for high availability and distributed processing, and so on.

High availability

High availability (**HA**) clusters provide services and resources in an uninterrupted manner through the redundancy built to the system. High availability is to be accomplished both within the cluster and between clusters too.

High availability within a cluster is accomplished by a master node that monitors the worker nodes for any failure and ensures that the load is distributed to other active, working nodes.

High availability between cluster examples across regions is accomplished by having each cluster system monitor the others, and in the event of failover, replicating servers and services through redundant hardware and software reconfiguration. Fault tolerance for hardware is achieved with raid systems, and for network systems, in the event of link breakdown alternative link paths are provided for continuity of services.

Load balancing

Load balancing among servers is a key valuable functionality in the increasing and explosive use of network and internet-based applications. It is an important feature that distributes incoming traffic requests from clients evenly across all the active node machines of the cluster that are allocated to the application. In case of a node failure, the requests are redistributed among the rest of the active nodes available and responsible for processing the orders. The web cluster, to be scalable, must ensure that each server is fully utilized, providing increased network capacity and improving performance. Web application servers based on load balancing are called **web farms**, and redirect requests independently as they arrive, based on a scheduler and an algorithm. A few popular algorithms for load balancing are least connections, round robin, and weighted fair; each have unique applicability.

High availability and load balancing

High availability and load balancing combines the features of both types of clusters, increasing the availability and scalability of services and resources. Consistent HA and load balancing is the backbone of the entire web hosting and e-commerce project; it needs to ensure support of the scalability of traffic volume on networks, without eventually becoming a bottleneck or single point of failure. Apart from simply redirecting client traffic to other servers, the web systems need to have verification of servers, redundancy, and balancing characteristics such as full-time communication checks. This type of cluster configuration is widely used in airline and train reservation systems, e-commerce, banking, email servers, and 24/7 applications.

Distributed processing and parallel processing

Distributed processing involves dividing a large computational task into smaller tasks, for processing to run in parallel on individual smaller clusters of nodes. It represents a massively parallel supercomputer. This model of cluster is effective for application in large computational tasks, and improves the availability and performance. These cluster systems are used for scientific research-based computing, weather forecasts, and so on; tasks that require high-processing power.

Usage of Hadoop big data cluster

Big data clusters are built for varied purposes, such as storage, analytics, testing, development, and so on. It is imperative to have the right size of the cluster for the right kind of workload, so capacity planning for a cluster is an important and critical task.

Big data clusters catering to different purposes are as follows:

- **Development cluster**: There are many requirements for building a development platform, such as the technological validation of porting an application to develop functionality on a big data platform, so as to develop advanced analytics use cases with machine learning.
- **Test cluster**: A test platform is built to test the features and functionality developed in the development cluster.

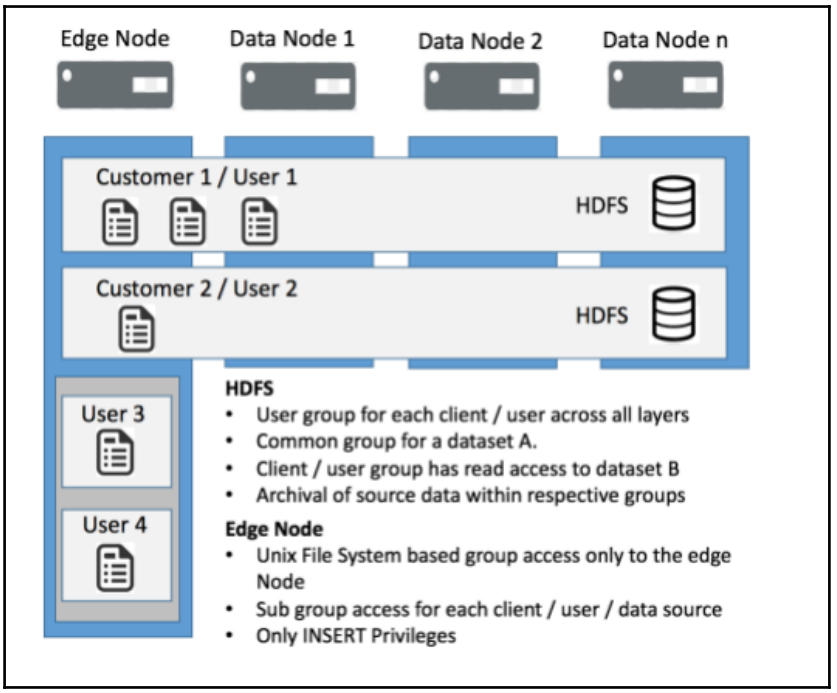
- **Data lake cluster**: A build used to provide extensive storage capacity, data from different source systems, including third-party data sources, are gathered into the data lake. There will be preprocessing activities to filter and perform aggregations on the incoming data before it is loaded into the data lake. A data lake serves multipurpose data needs for different departments of an organization.
- **Analytical cluster**: A platform for performing advance analytics using appropriate algorithms, and publishing the generated results.

Hadoop big data cluster nodes

We will discuss the different types of nodes along with their role and usage in Hadoop Ecosystem:

- **NameNode**: The NameNode is an important part of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks across where the cluster data files are stored. The data for these files is not stored at all. Client applications communicate with NameNode whenever there is a need to locate a file, or when they want to modify a file. The modifications are stored by NameNode as a log that is appended to a native file system file edits. When a NameNode starts up, it reads the HDFS state from an image file, fsimage, and then applies the edits to the log file.
- **Secondary NameNode**: Secondary NameNode's whole purpose is to have a checkpoint in HDFS. The Secondary NameNode is just a helper node for NameNode; it merges the fsimage and the edits log files periodically and keeps edits log size within a limit.
- **DataNode**: A DataNode stores data in HDFS. A functional file system has more than one DataNode, with data replicated across them. Client applications can talk directly to a DataNode, once the NameNode has provided the location of the data.
- **Edge/Hop Node**: Edge nodes are the interface between the Hadoop cluster and the outside network. Most commonly, edge nodes are used to run client applications and cluster administration tools. They're also often used as staging areas for data being transferred into the Hadoop cluster.
- **Cluster management**: Cluster management software applications provide end-to-end functionality and features for managing cluster landscapes. It facilitates the improvement of performance, enhances the quality of service, increases compliance and reduces administrative costs.

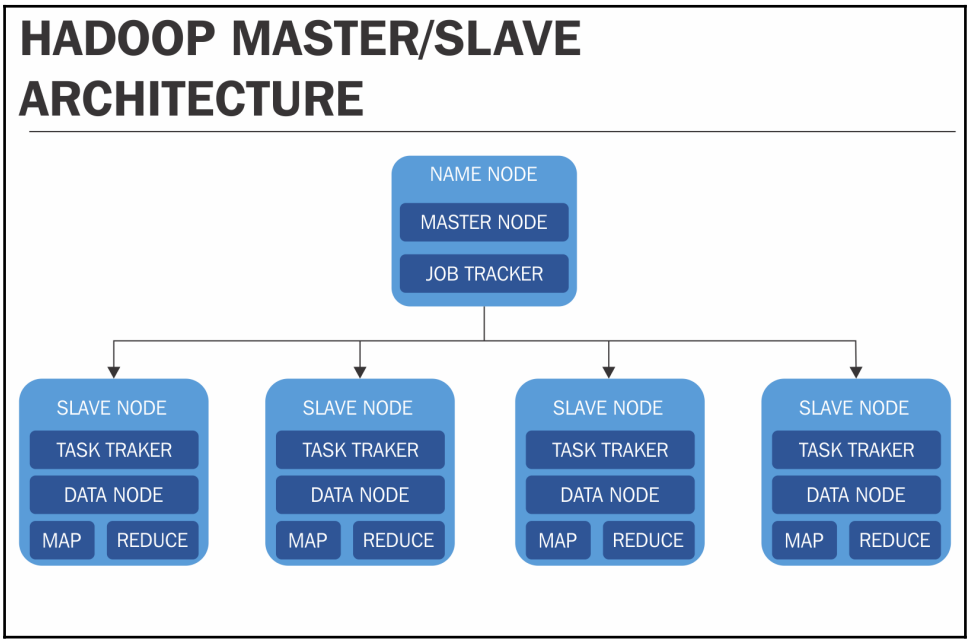
- **Security** for the HDFS file level and node level is depicted as follows:



Typically, the edge node is connected to the outside world through an external switch, that enables third-party systems to access through Kafka or STB-based, and SSH access for inbound users.

The preceding entire mentioned cluster servers interact with each other by a dedicated network switch, isolating traffic from the outside world.

Types of nodes and their roles



- **Nodes**: A Hadoop cluster can have a different configuration of servers based on the role they fulfill. This can be broadly divided into three types, with a different hardware configuration for each type:
    - **Master node** (also referred to as **name node**): In an enterprise deployment, this runs crucial management services. These nodes only store metadata so do not need a lot of storage, but since these files are critical and important, master node services include as follows:
        - Enterprise manager
        - Resource manager
        - Standby resource manager
        - NameNode
        - Standby NameNode

- Journal nodes
- HBase master
- Hive server
- Sqoop server
- ZooKeeper
- Oozie server
- Spark/Job history server
- Cloudera search
- Cloudera Navigator
- Hive metastore
- Kafka master
- Flume master

- **Worker node**: Worker/slave nodes in a Cloudera enterprise deployment are the ones that run worker services. Since tasks are performed by these nodes along with the storing of actual data, they are designed to be fault tolerant. Worker nodes can have the following roles:
    - Data node
    - Node manager
    - HBase region server
    - Impala daemons
    - Solr servers
    - Kafka broker
    - Flume agent
- **Gateway/edge node:** These are where Hadoop client services run, and include:
    - Third-party tools
    - Hadoop command-line clients
    - Beeline
    - Impala shell
    - Flume agents
    - Hue server
    - Spark and other gateway services
    - HA proxy

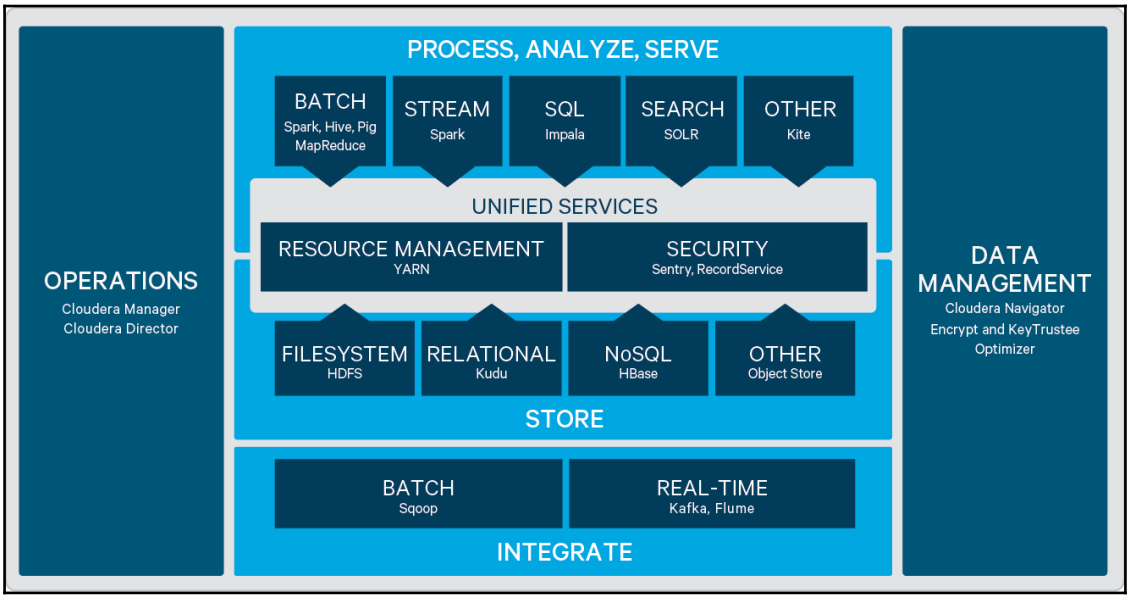
Commercial Hadoop distributions

As we have seen in the previous section, Hadoop is open stack community distribution with an integration of multiple components or interface layers. Many commercial vendors have built on the basic Hadoop platform and customized it to offer in the market, both as a hardware product platform and a service. We will discuss a few of the popular options as follows:

- Cloudera CDH Hadoop distribution
- **Hortonworks Data Platform (HDP)**
- MapR Hadoop distribution
- Amazon Elastic MapReduce
- IBM open platform
- Microsoft Azure's HDInsight--cloud-based Hadoop distribution
- Pivotal big data suite

Hadoop Cloudera enterprise distribution

The standard framework of Hadoop open source, with different layers and components, is presented as follows:



The Cloudera proprietary distribution framework is built on the Hadoop open source code, customizing the services as shown in the following topics. We will discuss the various components that make it a leading enterprise product.

Data integration services

Data from external source systems ingests into Hadoop systems can be through multiple modes based on the business need.

- **Batch transfer**:
    - **Apache Sqoop**: Sqoop is a command-line interface application for transferring data between relational databases and Hadoop. It supports the saved jobs that can be run multiple times helping us to import updates made to a database since the last import. It also supports the incremental loads of a single table or a free-form SQL query.
- **Real-time data transfer**:
    - **Apache Kafka**: Kafka is an open source message broker project developed by the Apache Software Foundation, written in Scala. The project aims to provide a complete, high-throughput, low-latency platform for handling real-time data feeds. It's a highly scalable pub/sub message queue architected as a distributed transaction log, making it very important for enterprise infrastructures to process streaming data.
    - **Apache Flume**: Flume adopts a simple, flexible and distributed architecture for streaming data. It effectively ingests large amounts of log data reliably and aggregates it. It has a simple, extensible data model, flexible for building and supporting online analytical applications. Flume is a pretty robust, fault tolerant, reliable service with built-in failover and recovery features.
    - **Apache Chukwa**: Chukwa is a framework for data collection and analysis on distributed file systems such as Hadoop, simplifying log analysis, processing and monitoring. Chukwa agents run on respective machines to collect the logs generated from various applications. It offers a high degree of flexibility to ingest the huge log data generated by servers. Collectors receive the data from the agent and write them to HDFS, which serves as storage, and the MapReduce framework will process, analyze, and parse the jobs and archive the huge log data.

- **Apache Avro**: Avro is a language-neutral remote procedure call and data serialization framework developed within Apache's Hadoop project. Since Hadoop writable classes lack language portability, Avro uses JSON for defining data types and protocols and serializes data in a compact binary format. Avro is quite helpful for dealing with data formats that can be processed by multiple languages such as Java, C, C++, C#, Python, and Ruby.

Hadoop data storage

Data storage in Hadoop is a key function and we will discuss various modes of accomplishing this:

- **Apache HDFS (Filesystem)**: The HDFS is a distributed, scalable, and portable file system written in Java for the Hadoop framework. HDFS stores large files--typically in the range of gigabytes to petabytes--across multiple machines and data nodes. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.
- **Apache HBase (NoSQL)**: HBase is an open source, non-relational, distributed data store running atop of HDFS providing a fault-tolerant means of storing large amount of sparse data. HBase is a column-oriented key-value data store and has been admired greatly due to its lineage with Hadoop and HDFS. It is suitable for faster read and write operations on extensive datasets with high throughput and low input/output latency.
- **Apache Kudu (Relational)**: Apache Kudu is an open source storage engine intended for structured data that supports low-latency random access, together with efficient analytical access patterns. It bridges the gap between HDFS and the HBase NoSQL database. Kudu tables look like those in SQL relational databases, to act as a storage system for structured data. Like RDBMS principles, primary keys are made up of one or more columns that enforce uniqueness and act as an index for efficient updates and deletes, as a storage system for tables of structured data.

Data access services

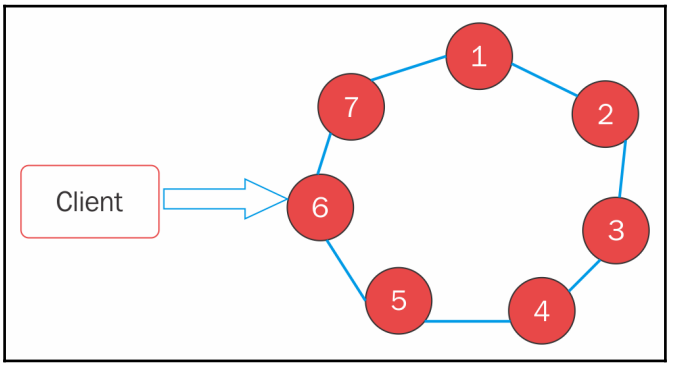
- **Apache Spark**: Spark is an open source framework for machine learning and stream processing based on in-memory technology for data processing. It provides programmers with a data structure called the resilient distributed dataset (RDD), an application programming interface. The RDD is read-only, distributing multiple sets of data items over a cluster of machines with fault tolerant features.
- **Apache Hive**: The Hive data warehouse software facilitates reading, writing, and managing large datasets that reside in distributed storage using SQL. The structure can be projected onto data already in storage. A command-line tool and JDBC driver are provided to connect users to Hive.
- **Impala**: Impala is Cloudera's SQL query engine for data stored in an Apache Hadoop cluster and running open source massively parallel processing (MPP). Impala enables users to run low latency SQL queries on data stored in HDFS and Apache HBase, without requiring additional data movement or transformation.
- **Solr**: Apache Solr is a search platform for websites, popular for enterprise search because it can be used to index and search documents and email attachments. It's built upon a Java library called **Lucene** (`http://whatis.techtarget.com/definition/Apache-Lucene`), written in Java, and provides both a RESTful XML interface and a JSON API that are used to build search applications. Solr can search and index multiple websites, returning content related recommendations based on the taxonomy (`http://searchcontentmanagement.techtarget.com/definition/taxonomy`) of the search query (`http://searchsqlserver.techtarget.com/definition/query`).
- **Apache Pig**: Pig provides a high-level language known as Pig Latin, a SQL-like language with many built-in operators for performing data operations such as joins, filters, ordering, and so on, and is used to perform all the data manipulation operations in Hadoop. The component of Apache Pig is Pig Engine that ingests the Pig Latin scripts as input and converts the scripts into MapReduce jobs. As a tool, it's very efficient, reducing development and coding time. As a platform, it adopts multiple query paths, representing them as data flows to analyze large sets of data.

- **Kite** is a high-level data layer for Hadoop, that provides an API and a set of tools to create logical abstractions on top of storage systems (such as HDFS) and operates in terms of records, datasets, and dataset repositories. It can access Mavin through plug-in and aid-in packaging, deploying and running distributed applications. It speeds up the development of stream processing ETL applications in Hadoop that extract, transform, and load data into target repositories such as Apache Solr, enterprise data warehouses, HDFS, HBase, and OLAP applications.
- **MapReduce** is a processing technique and framework based on Java for distributed computing. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job. A map job usually splits the input dataset into independent chunks where individual elements are broken down into tuples (key/value pairs). The reduce framework sorts the outputs of the map jobs, which are then input to the reduce tasks. The tasks are processed in a completely parallel manner to scale data processing over multiple computing nodes. The input jobs and the output jobs are stored in a file system. The framework takes care of scheduling tasks, monitoring them and re-executing failed tasks. The MapReduce model framework can easily scale the application to run over tens of thousands of machines in a cluster through a configuration change.

Database

Apache Cassandra architecture is a distributed NoSQL database management system known for its ability to scale, perform, and offer continuous uptime. Apache Cassandra is based on a ring design wherein all nodes play the equivalent role without any master concept. Compared to other architectures such as master-slave, legacy, or sharded design, Cassandra is quite easy to set up and maintain, and is designed for handling a high volume of structured data across commodity servers.

Apache Cassandra's high availability and scalable architecture enable it to handle large amounts of data, and thousands of concurrent users and operations spread across multiple data centers to ensure high-performance by distributing user traffic. Cassandra has built-in features such as data modeling, high availability clusters, monitoring tools, query language, and so on.



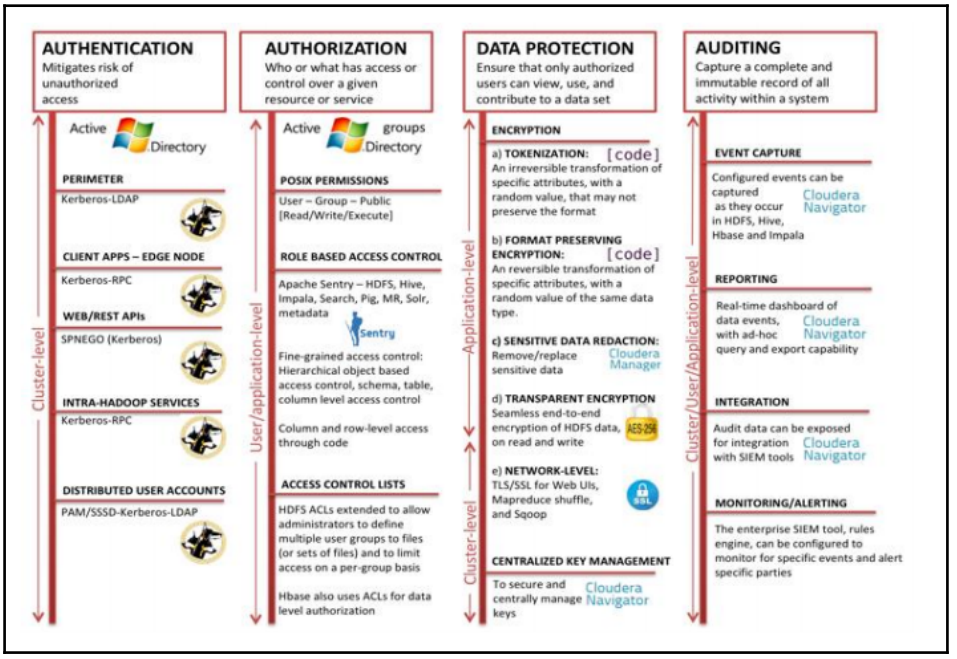
Unified (common) services

- **Resource management**: Apache Hadoop **Yet Another Resource Negotiator** (**YARN**) is a cluster management technology. YARN is one of the key features of the second-generation Hadoop 2 version of the Apache Software Foundation's open source distributed processing framework. It also enables versatility; the resource manager can support additional paradigms and not just map/reduce.
- **Apache Oozie**: Apache Oozie is a system to schedule a workflow to manage Hadoop jobs. The directed acyclic graph is the mode of representing workflows in Oozie, which are a collection of control flow and activity nodes. The beginning and the end of a workflow and the mechanism to control the workflow execution path are defined by control flow nodes. The execution of a computation processing task through workflow triggers happens on action nodes.

- **Apache Sentry**: Hadoop's strong security at the file system level lacks the granular support to adequately secure row-level access to data by users and BI applications. Sentry allows access control at the server, database, and table, and grants different privilege levels including select, insert, and so on. It provides for authenticated users privileges on data, the ability to control and enforce access to data, and so on. It enables fine-grained access control to data and metadata in Hadoop. The column level security can be implemented by creating a view of a subset of allowed columns by restricting the base table and granted privileges. Sentry administration is simple and convenient through role-based authorization. It is a policy engine that can easily grant multiple groups access to the same data at different privilege levels such as resource, roles, users, and groups.

The following diagram and table provide a comprehensive view of the security model adopted by Hadoop systems. There are security requirements at multiple levels, such as clusters, user level, and application level in an enterprise-wide implementation.

Cloudera offers four layers of security as follows; perimeter, access, visibility, and data. Cloudera Enterprise Security can be classified into four broad categories; authentication, authorization, data protection and auditing:



Security features offered by popular tools are listed in the following table:

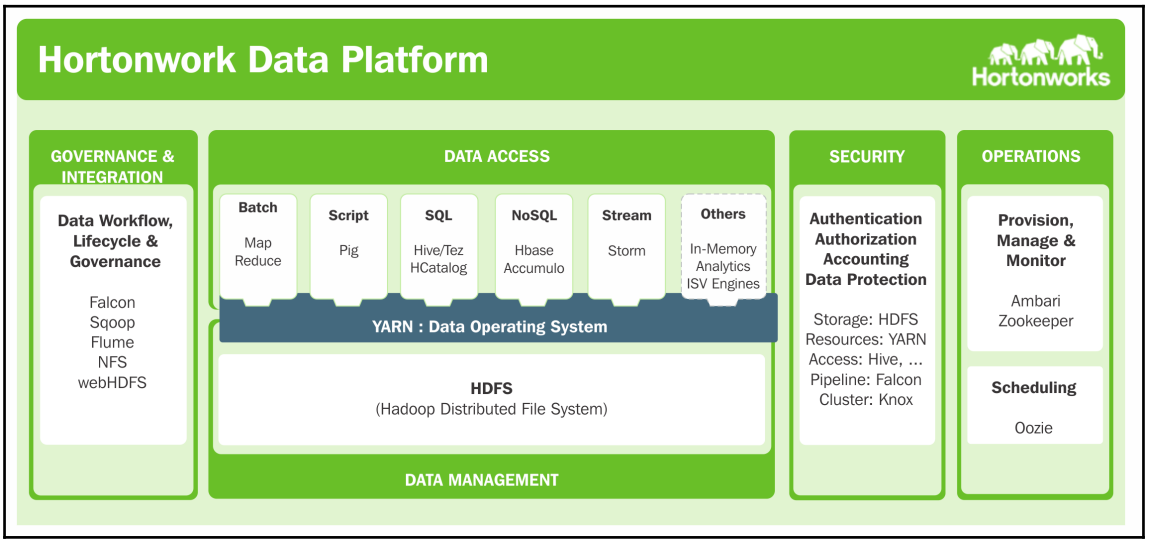
Key Security Requirements	Apache Sentry	Kerberos LDAP	Cloudera Navigator	Custom
Unique User Identification		Yes		
Emergency Access Procedure				Yes
Automatic Logoff				Yes
Encryption and Decryption at Rest			Yes	
Transmission Security- Encryption/Decryption			Yes	
Mechanism to Authenticate	Yes		Yes	
Authenticate. (Role based authorization)	Yes	Yes	Yes	
Transmission Security - Integrity Controls	Yes			
Audit, lineage			Yes	

Cloudera proprietary services and operations/cluster management

- **Cloudera Navigator**: Cloudera Navigator is part of Cloudera Enterprise, and is a fully integrated data management and security system for the Hadoop platform. Cloudera Navigator is the data governance solution for Hadoop, presenting crucial capabilities such as data discovery, continuous optimization, audit, lineage, metadata management, and policy enforcement. Cloudera Navigator supports continuous data architecture optimization and meeting regulatory compliance requirements.
- **Cloudera Manager**: Cloudera Manager is an end-to-end application for managing CDH clusters. Cloudera Manager sets the standard for enterprise deployment by delivering granular visibility into and control over every part of the CDH cluster—empowering operators to improve performance, enhance the quality of service, increase compliance and reduce administrative costs.
- **Cloudera Director**: Cloudera Director works with Cloudera Manager and the cloud service provider to provide centralized and programmatic administration of clusters in the cloud, including deployment, configuration, and maintenance of CDH clusters. With Cloudera Director, you can monitor and manage multiple Cloudera Manager and CDH deployments, across different cloud environments.

A Hadoop Hortonworks framework

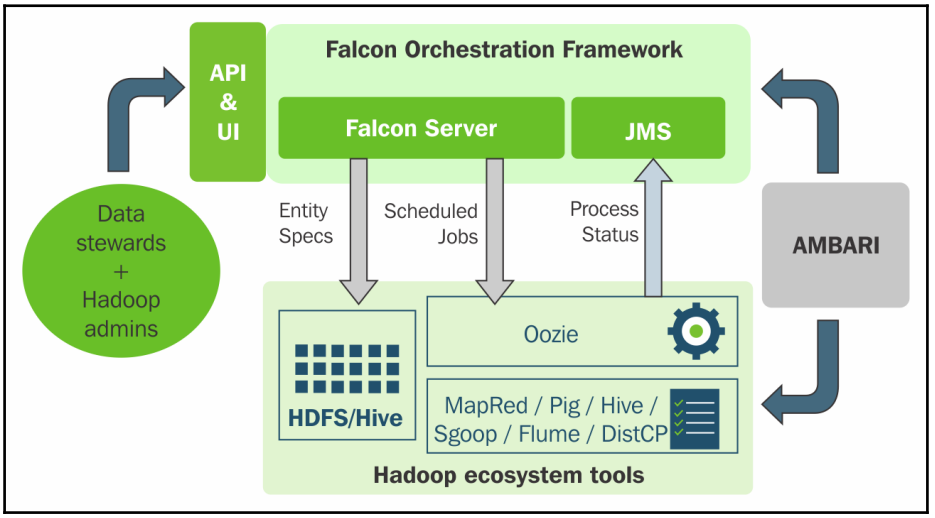
The following image is the framework of Hortonworks. Many components are the same as the Hadoop stack, as seen previously; we will discuss the components unique to this distribution:



Data governance and schedule pipeline

Apache Falcon is a data management tool for managing dependencies between the system infrastructure, data, and processing logic. Data administrators can define operational and data governance policies for Hadoop workflows for overseeing data pipelines in Hadoop (`http://searchdatamanagement.techtarget.com/definition/Hadoop-2`) clusters. Using Falcon, we can manage thousands of compute nodes, with a large number of jobs typically running on a cluster at any given time, ensuring a consistent and dependable performance on complex processing jobs.

Falcon relies on Oozie job scheduling software to generate the processing workflows, to set procedures for replication, and for the retention and archiving of incoming data. The data governance engine schedules and monitors data management policies such as enhanced monitoring, and so on. The other features are tracing jobs activities for failures, dependencies, audits, and lineage, and also tagging the data to comply with data retention and discovery requirements:



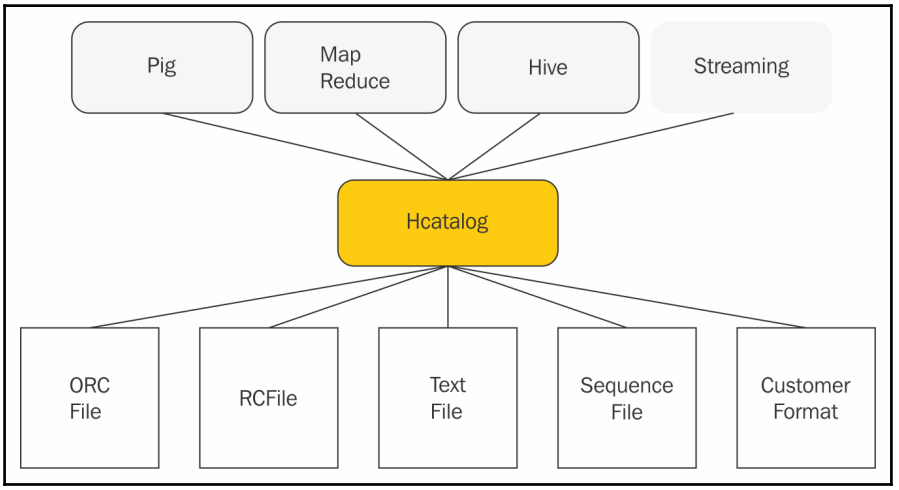
Cluster management

Apache Knox is a bastion security server; shielding direct access to Hadoop cluster nodes helps setups be more secure for enterprise-ready installations. Knox can easily scale horizontally by supporting stateless protocols. Knox provides authentication functionality to be managed by users and groups using LDAP or active directory. Identity Federation is SSO and HTTP header based.

Authorization is supported through an **access control list** (**ACL**) on service levels. The Knox Policy enforcement ranges from authentication, federation, authorization, audit, dispatch, host mapping and content rewrite rules. The policy is enforced through a list of providers, defined within the topology and the cluster definition, for purposes of routing and translation between user-facing URLs and cluster internals.

Data access

- **Apache Tez**: This is an application framework that allows for a complex directed-acyclic-graph of tasks for processing data. It is a resource-management framework, based on Apache YARN functionality, that is extensible for building high-performance batch and interactive data processing applications, leading to significant improvements in response times while maintaining MapReduce's ability to scale to petabytes of data. Tez caters for cases that require the near-real-time performance of query processing and machine learning, and so on, with a powerful framework based on expressing computations as a data flow graph.
- **Apache HCatalog**: This is a storage management layer for Hadoop, that facilitates the easy reading and writing of data from the Hadoop cluster grid with different data processing tools such as Hive, Pig, MapReduce, and so on. For different kinds of data formats stored on HDFS, such as RCFile, Parquet, ORC files, or Sequence files, it uses Hive **Serializer-Deserializer** (**SerDe**) to enable a relational view. Apache HCatalog provides features such as table abstraction and data visibility to tools for cleaning and archiving.

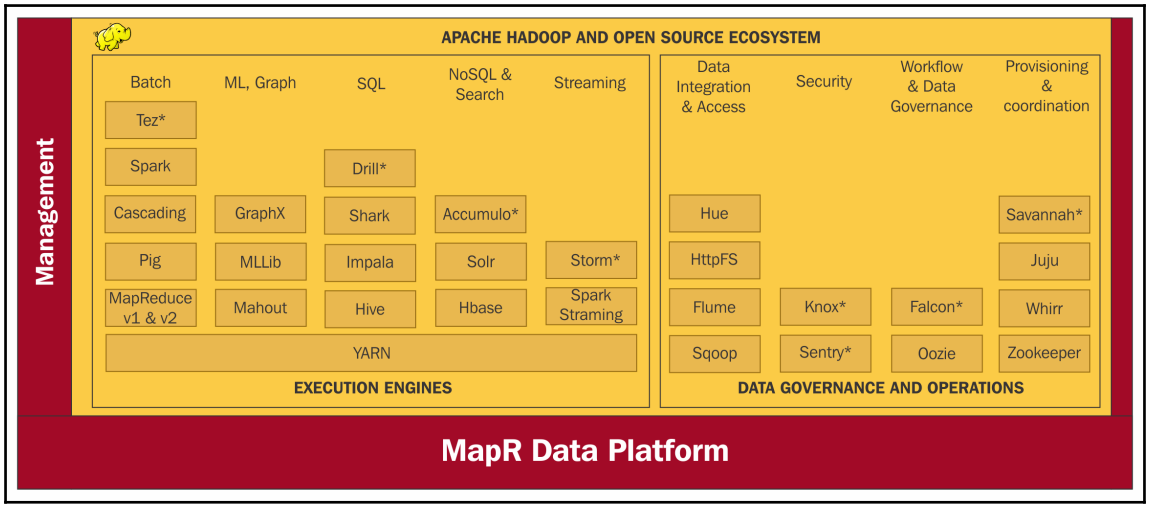


Data workflow

The WebHDFS protocol provides external applications over the internet, HTTP, or web access for managing files and data stored in the HDFS cluster, on a par with the high-performance native protocol or native Java API default with Hadoop Cluster. WebHDFS is based on an industry-standard RESTful mechanism that provides security on par with native Hadoop protocols. Using WebHDFS common tools such as `curl/wget`, users can access the HDFS for operations such as reading files, writing to files, making directories, changing permissions, renaming, and so on.

A Hadoop MapR framework

MapR is a commercial distribution of Apache Hadoop with HDFS replaced with MapR-FS. The following is the MapR framework with common Hadoop open source components; we will now review the components unique to this distribution:



Machine learning

MLLIB is a collection of machine-learning algorithms and utilities for prediction models and data sciences. There are some broad groups, such as classification, clustering, collaborative filtering, and so on. A few of the ML algorithms used for each category are listed following:

- **Classification**: Logistic regression, naive Bayes
- **Regression**: Generalized linear regression, survival regression, decision trees, random forests, and gradient-boosted trees
- **Clustering**: K-means, Gaussian mixtures (GMMs), frequent itemsets, association rules, and sequential pattern mining

GraphX is a new component in Spark for graphs and graph-parallel computation, used to implement new types of algorithms that require the modeling of relationships between objects. In many real-world applications, such as social networks, networking, and astrophysics, graph processing is very effective and efficient at representing a model relationship between entities visually.

SQL stream

Apache Drill is a distributed SQL engine that enables data exploration and analytics on non-relational data stores such as Hadoop, MapR, CDH, NoSQL (MongoDB, HBase), cloud storage (Amazon S3, Google Cloud Storage, Azure Blog Storage, Swift), and so on. It uses a shredded, in-memory, columnar execution engine for distributed query optimization and execution, for complex data and schema-free data. By using a query engine that compiles and re-compiles queries at runtime, high performance is achieved for any structure of data. Using standard SQL and BI tools, users can query the data without having to create and manage schemas. It supports schema-free JSON document models, similar to MongoDB and Elasticsearch, and industry-standard APIs--ANSI SQL, ODBC, JDBC, and RESTful APIs.

Apache Shark is a data warehouse-based system used with Apache Spark; its distributed query engine enhances high-end analytical results and the performance of Hive Queries multifold. Shark supports most of the Hive's features, such as query language, metastore, serialization formats, and user-defined functions. Apache Shark is built on top of Apache Spark, which is a parallel data execution engine; hence, Shark can respond to complex queries in sub-second latency. It provides the maximum performance gains offered by column-wise memory storage systems, as data is stored and processed within the cluster's memory or in a database with in-memory materialized views.

Storage, retrieval, and access control

Accumulo provides fine-grained data access control and cell-level access control, with complex policies governing access to sensitive data. It is a low-latency, large table data storage and retrieval system, with a key/value store based design. Accumulo provides extremely fast access to data in massive HDFS tables, while also controlling access to its millions of rows and columns down to the individual cell. It enables the intermingling of different data sets with access control policies for fine-grained access to data sets by encoding the policy rules for each individual data cell, and controls fine-grained access.

Data integration and access

Hue is an open source web interface for analyzing data within any HDFS cluster through Apache Oozie. It comes with many built-in features such as Hue's Editor to build workflows and then schedule them to run regularly and automatically. It has a dashboard for data querying, monitoring progress and logs, and performing actions such as pausing or stopping jobs. The applications supported are Apache Hive, Apache Impala (incubating), MySQL, Oracle, PostgreSQL, SparkSQL, Apache Solr SQL, Apache Phoenix, Apache Solr, and Apache Spark.

HttpFS--Apache Hadoop HttpFS is a service that provides HTTP access to HDFS through REST APIs, supporting all HDFS file system operations (both read and write). It supports data transfer between HDFS clusters running different versions of Hadoop (overcoming RPC versioning issues) or a cluster behind a firewall.

Provisioning and coordination

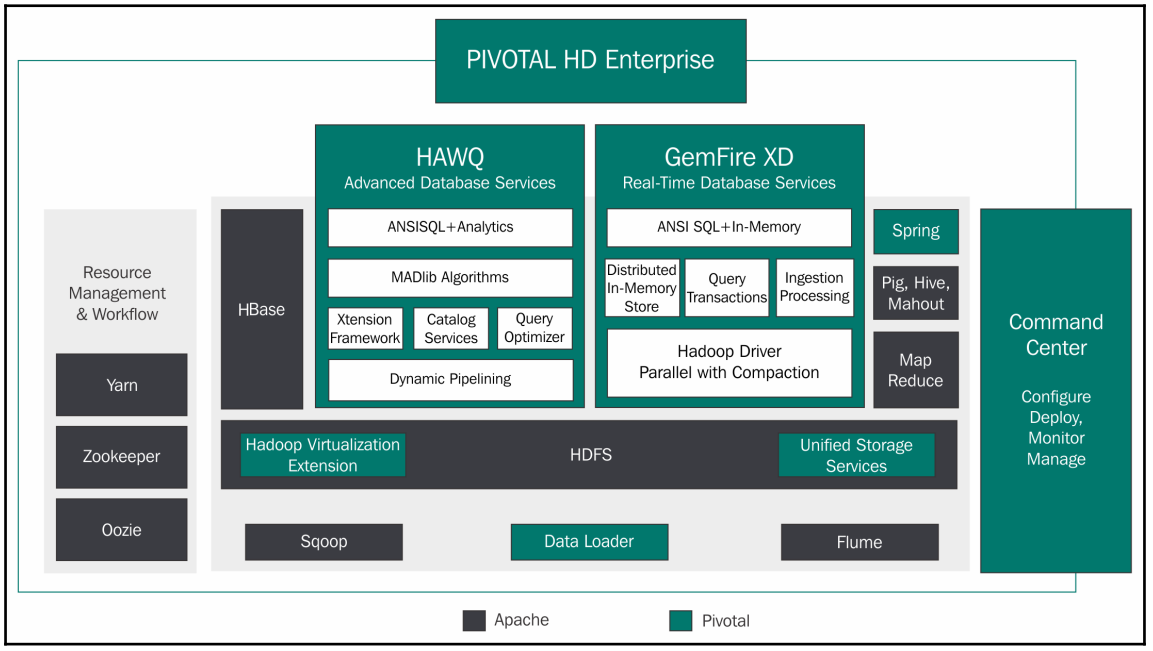
The data provision tools enable cloud hosting and coordination services for Hadoop systems, two popular choices are discussed following:

- **Juju**: The container option available with Hadoop distribution is the Juju framework. This allows users to deploy software built locally on a range of services, including MAAS, EC2, Azure, LXD containers. Juju can model, configure and manage services and deploy to all major public and private clouds with only a few commands. Hundreds of preconfigured services are available in the Juju store.

- **Apache Whirr--big data on clouds**: Apache Whirr can be used to define, provision and configure big data solutions on cloud platforms such as Amazon EC2, Rackspace servers, and CloudStack. Whirr automatically starts instances (set of libraries) in the cloud and bootstraps Hadoop through on them. It initiates cloud-neutral big data services to define and provision Hadoop clusters in the cloud, and adds packages such as Hive, HBase, and Yarn for MapReduce jobs.

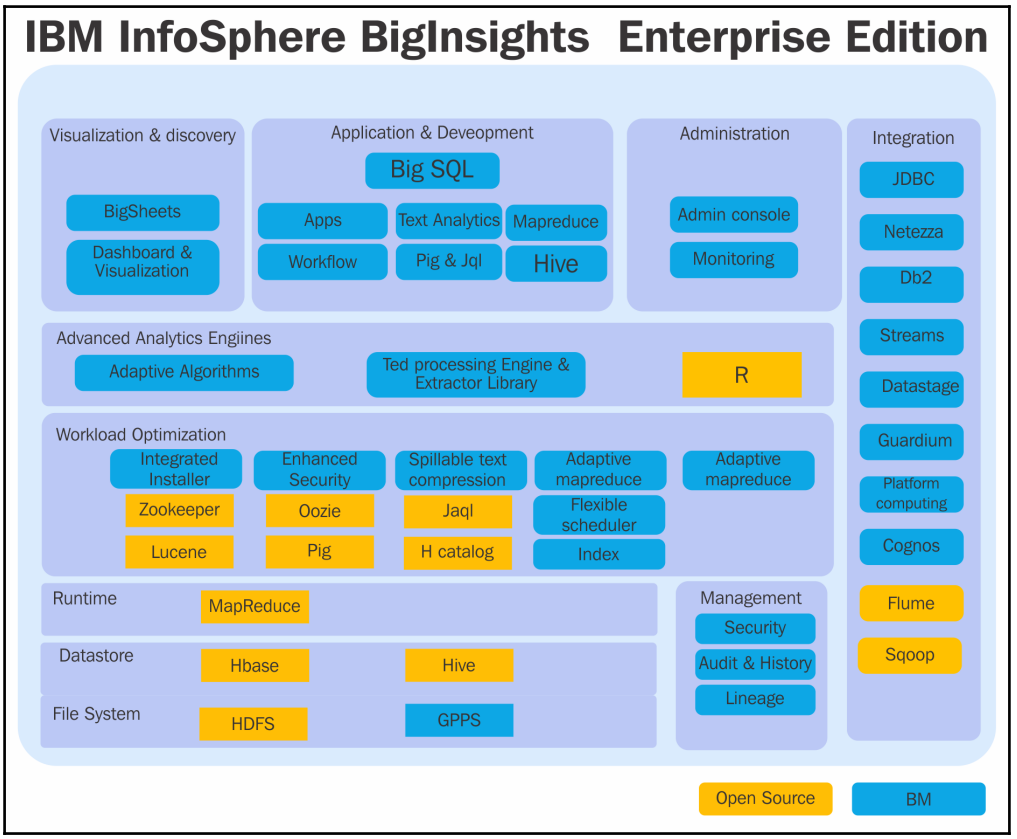
Pivotal Hadoop platform HD Enterprise

The following is the Pivotal HD Enterprise framework. The open source Hadoop components in this framework were already discussed earlier:



A Hadoop ecosystem on IBM big data

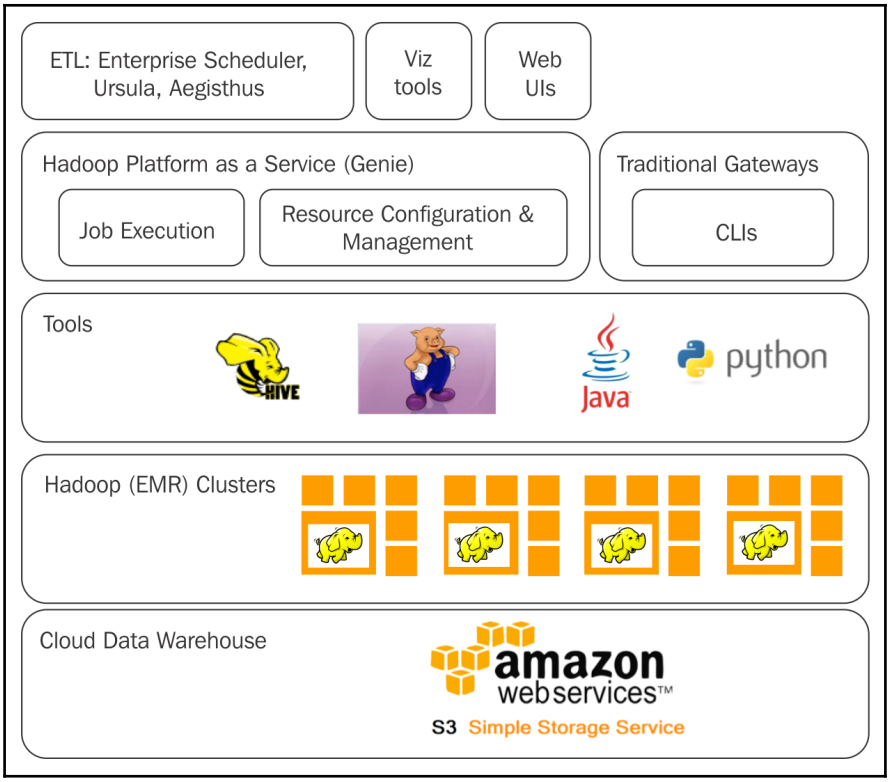
The following is a Hadoop ecosystem based on IBM big data. We are already familiar with most of the open source Hadoop big data components listed in this framework:



A Hadoop ecosystem on AWS

Amazon **Elastic MapReduce** (**EMR**) is a service that allows users to launch and scale Hadoop clusters inside of Amazon's web infrastructure. EMR instances use Amazon's prebuilt and customized EC2 instances, which greatly simplifies the setup and management of the cluster of Hadoop and MapReduce components. EMR can analyze large datasets on AWS cloud Hadoop clusters quite effectively.

An AWS EMR framework depicting multiple service layers is shown in the following diagram:



An AWS EMR framework that offers integrations with a wide choice of Hadoop open source components is presented as follows:

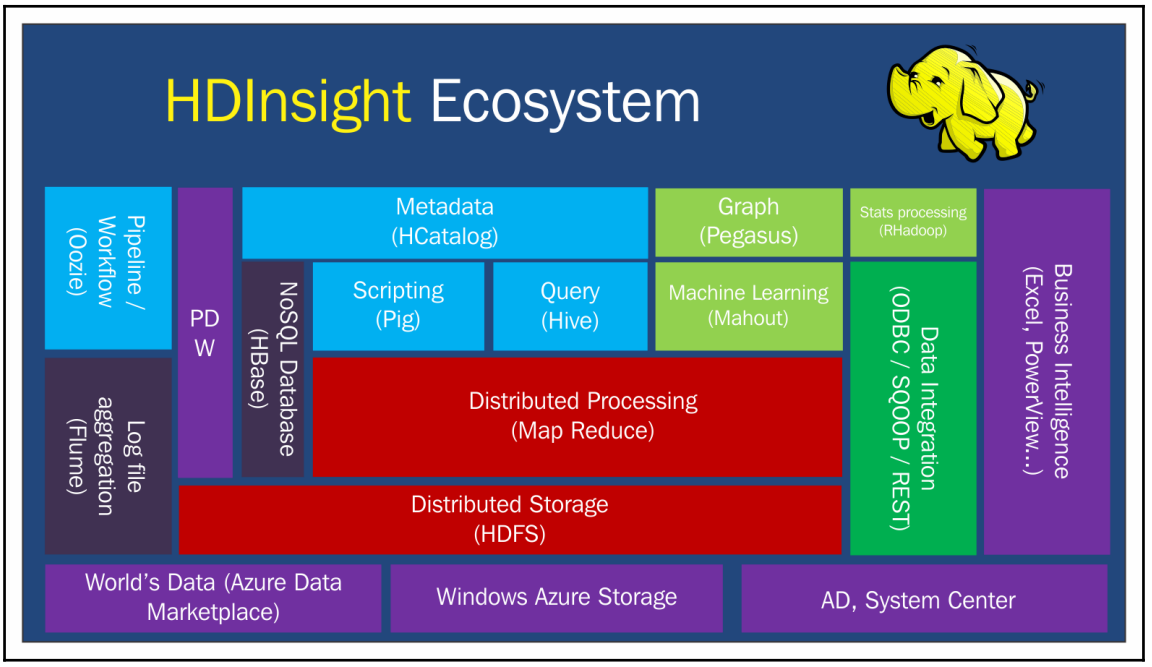
- **R**: It is a GNU package, open source programming language and software environment for statistical computing and graphics, and is widely used among statisticians and data miners for developing statistical software and data analysis. R as a language is both flexible and powerful.

- **Presto**: Apache Presto is a distributed parallel cross-platform query execution engine on the Hadoop platform. Presto supports using standard ANSI SQL to query multiple sources such as HDFS, MySQL, Cassandra, Hive, relational databases, and other data stores. Presto runs multiple analytic queries, optimized for low latency and interactive query analysis, and scales without downtime. Presto supports most of today's best industrial applications such as Facebook, Teradata and Airbnb, and so on.
- **Gradle**: The landscape of modern software development is continuously evolving, and so are the needs for build automation. Projects involve large and diverse software stacks with multiple programming languages and a wide spectrum of testing strategies. Adopting agile practices leads to the early integration of code, as well as frequent and easy delivery to both test and production environments, as supported by builds. Gradle is an open source build automation system that builds upon the concepts of Apache Ant, and Apache Maven. Gradle uses a directed acyclic graph to schedule the order of the tasks and introduces a Groovy-based domain-specific language for declaring the project configuration. Gradle was designed for multi-project builds for its ability to manage dependencies. Gradle can define and organize large project builds, as well as modeling dependencies between projects. It supports incremental builds by intelligently determining the build tree dependencies and need for re-execution.
- **Cascading**: Cascading is an application development platform for building big data applications on Apache Hadoop, providing an abstraction layer for Apache Hadoop and Apache Flink. Cascading is used to create and execute complex data processing workflows on a Hadoop cluster, hiding the underlying complexity of MapReduce jobs. Cascading provides an optimal level of abstraction with the necessary options through a computation engine, systems integration framework, data processing, and scheduling capabilities. Cascading offers Hadoop development teams portability for simple or complex data applications without incurring the costs of rewriting them.

- **Apache Phoenix**: Apache Phoenix is an open source, massively parallel relational database engine that uses Apache HBase as a base to support OLTP for Hadoop. It provides random, real-time access to large datasets with a familiar SQL interface to Hadoop systems such as Spark, Hive, Pig, Flume, and MapReduce. Apache Phoenix abstracts away the underlying data store. Aggregation queries are executed on the nodes where data is stored, reducing the need to send massive data over the network.
- **Apache Mahout**: Apache Mahout is a suite of scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering, and classifications.

Microsoft Hadoop platform is HDInsight hosted on Microsoft Azure

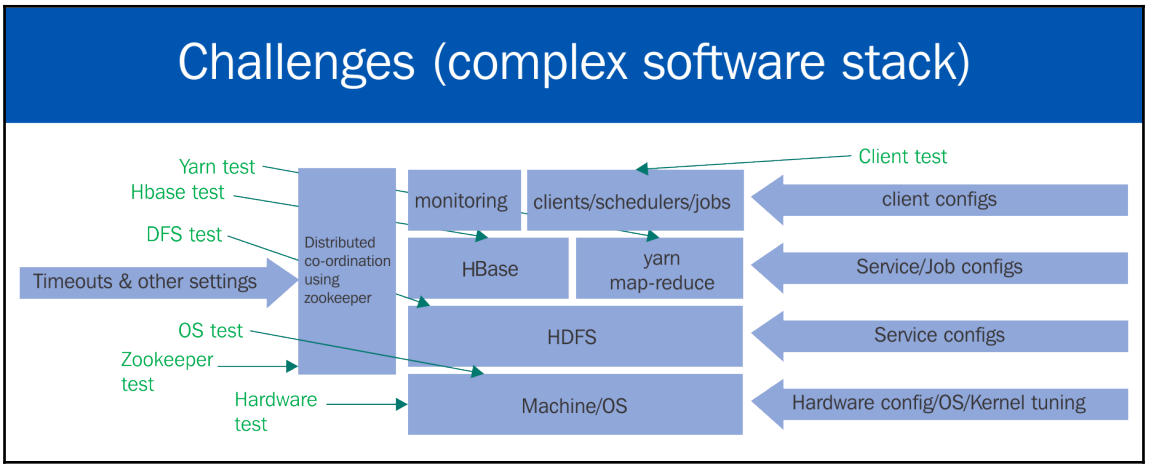
The following is the architecture of an HDInsight Ecosystem hosted on Microsoft Azure. While some of the native open source layers are embedded as is, some others are tailored and customized as per Microsoft proprietary offerings:



Capacity planning for systems

Sizing a Hadoop cluster is an important task as there are many factors influencing the performance. Capacity planning and the sizing of a Hadoop cluster are imperative for optimizing the distributed cluster environment with its related software. The number of machines, specifications of the machines, and effective process per node planning will allow you to optimize the performance effectively.

Within the Hadoop ecosystems, different layers (components/services) interact with each other, leading to performance overheads associated within a complex cluster stack between any of the layers; hence the need for requisite performance tests at each interface and appropriate tuning, as depicted in the following diagram:



There are many factors that influence the capacity planning, sizing, and performance of a complex Hadoop-distributed cluster. The following are a few factors for consideration:

- **Amount of data**:
  - The volume of data and growth
  - The data retention policy of how many years to hold data before discarding
  - Also the data storage mechanism (data container, type of compression used if any)

- **Type of workload**: If workloads are CPU/IO /memory intensive, we will have to consider hardware accordingly. If processing might grow rapidly, we have to consider adding new data nodes.
- **Frequency of workload**: If the data load is batch or real-time streaming data, would it be a few times a day, nightly, weekly or monthly loads?
- **Type of security**: Authentication, authorization, and encryption
- **Type of services required**:
    - What is the business SLA for the cluster? Is there is a requirement for real-time support?
    - What types of services are running other than core Hadoop services?
    - How many third-party tools will be installed/used?
- **Choice of an operating system**: Selecting an operating system depends on multiple factors, such as your team's administration competency, the cost of procurement and maintenance, stability, performance, reliability, support availability, and so on.
    - **CentOS**: Linux CentOS is functionally compatible with RHEL, and a popular choice towards work nodes in Hadoop clusters.
    - **RedHat Enterprise Linux (RHEL)**: Linux RHEL is widely used for servers in Hadoop clusters.
    - **Ubuntu**: A very popular distribution, based on Debian – both desktop and server versions available.
    - **SLSE**: A Linux Enterprise Server developed by SUSE. It is designed for servers, mainframes, and workstations but can be installed on desktop computers for testing as well.
    - **Solaris, OpenSolaris**: Not popular in production clusters.
    - **Fedora Core:** Linux distributions for servers and workstations.
- **Network considerations**: Hadoop is very bandwidth-intensive, since most of the time all the nodes are communicating with one other simultaneously. Consider the usage of dedicated switches, a network Ethernet bandwidth of 10 GB/sec, and racks that are interconnected through switches.

Guideline for estimating and capacity planning

Proper cluster sizing includes selecting the right hardware for a master and a worker, as well as edge nodes, while keeping the cost low.

Here are a few factors to consider, along with some general guidelines for planning capacity sizing:

- **Data size**:
    - For data sizing, the recommended replication factor is 3
    - If the total data to be stored is *Y* TB, it will become *3Y* TB after replication
    - If any compression techniques are used to store data then a compression factor can be considered
    - For disk efficiency, only 60-70% usage is recommended for the total disk availability
    - So, the total disk capacity including disk compression factor = *3 x Y x 7*

- **Data growth rate**:
    - Considering data growth factor, say 1 TB per year
    - Consider a 3 x replication factor
    - Disk efficiency factor of 70%
    - We need to calculate *1 TB * 3 / .6 = 4-5 TB*
    - Storage capacity per 1 TB of data growth
    - It is equivalent to adding a new node, considering the data processing needs and growth of data volume
    - The number of data nodes planner:
        - Let's consider that we need to store 200 TB of data in HDFS
        - With 3 x replication, it will be 600 TB
        - Considering the replication factor, it will be *600 * 1.3 = 780* TB storage (nearly)
        - Assuming per node there are 12 disks with 2 TB capacity = 24 TB per node

- The number of nodes required would be *780/24 = 33* nodes
- Considering data growth requirements, we need to plan for cluster expansion in terms of additional nodes required per month, week, year, and so on

Cluster-level sizing estimates

Cluster sizing and capacity planning is important for various nodes as per their roles.

For master node

Capacity planning for master mode is critical and needs consideration of system resources and services hosted on it as detailed following:

- **Memory**: Capacity sizing of the master name node memory is a very important task. The rule of thumb is to have 1 GB of heap size to store 1 million blocks. For example:
    - Consider a 5-node cluster
    - Raw storage on each node is 20 TB
    - Consider that the HDFS block size is 128MB
    - Total number of blocks = *5 * (20*1024*1024) = 33 million* (approximately)
    - Total HDFS block size is 33 million
    - Based on a replication factor of 3, the required heap size = *33/3 = 11*
    - The total number of actual blocks would be approximately 11 GB
    - There are other factors that we need to consider as well:
        - Name node would be the stable entity, so you should plan for the future growth prospects as well, while planning the sizing configuration
        - Services such as resource manager, HBase master, zookeeper, and so on would also run on this node

  - Rule of thumb is to keep a minimum of 2 GB for services such as HBase and 4 GB for the resource manager
  - Based on normal usage, the memory configuration for the master node should be 128 GB to 512 GB

- **CPU**: Cores of 2 GHz or above of processor per node are reasonable, depending on the number of services running on the node.
- **Disk**: The local disks on the master node should be configured as RAID10 with hot spares to give good performance and fast recovery on the loss of a disk. There should be a separate partition for `/var` with a minimum size of 1 TB, or based on the capacity required for log storage. It is strongly recommended to configure High Availability in production.

Worker node

Capacity planning for worker node needs due consideration for system resources (memory, CPU, disk) as per the services it runs:

- **Memory**: Memory on the worker nodes is based on the type of workload and daemons they will be running. You should consider 256 GB to 512 GB of memory for each of the worker nodes.
- **CPU**: CPU capacity is based on the type of workload and numbers of parallel MapReduce tasks that are planned. By enabling hyperthreading, the total number of MapReduce tasks should be 1.5 times of the number of the cores. Consider a minimum of 24 cores with processors greater than 2 GHz.
- **Disk**: You should consider having a large number of small SATA disks (2 to 4 TB), instead of a small number of large disks. All disks should be configured as JBODs with noatime, nodiratime along with `/var` as a separate partition from the root OS partition. It is recommended you add additional nodes to enhance storage, as it will increase processing power too.

Gateway node

Gateway nodes do not run any specific Hadoop services; their configuration should be based on what jobs they will run.

- **Network**:
  - Network is a core component and should be considered very carefully as processing in Hadoop is based on data proximity.
  - If workloads consist of map-only jobs that are only transforming the data, there isn't a lot of data movement over the wire. If workloads have a lot of reduce actions such as aggregation, joins, and so on, then there is a lot of data movement between the nodes, in which case we should consider a minimum network capacity of 10 GB.
  - Running other services such as HBase, Spark should also be taken into account while considering network configuration.
  - Consider an average of 15 to 20 nodes per rack.
  - Dual Ethernet cards bonded together are recommended to support failover.
  - It's good practice to set up data locality and replication configured across the rack (rack awareness), so that even if one rack goes down, data is still available.
  - Core switches connected to top of the rack switches should be of high bandwidth (10 GB/sec or more). Consider redundancy for top of rack as well as core switches.

Summary

In this chapter, we have covered big data Hadoop components, popular frameworks, and their unique components for various services. In the next chapter, we will cover the terminology and technologies of the cloud such as public, private, hybrid models, service offerings for infrastructure, platform, identity, software, and network as service. We will also present a few popular market vendors.

5
Cloud Computing

Cloud technologies are integral to the building of big data, data sciences, and Internet of Things-based systems to accommodate huge data volumes, as they bring agility and scalability. DevOps is more prominent for cloud, to reproduce environments as replicas with configurations to be executed as code on par with software projects. In this chapter, we will discuss cloud computing terminology, architecture models, building blocks for service offerings such as platforms, infrastructures, and so on. We will also discuss the popular market vendors, and so on.

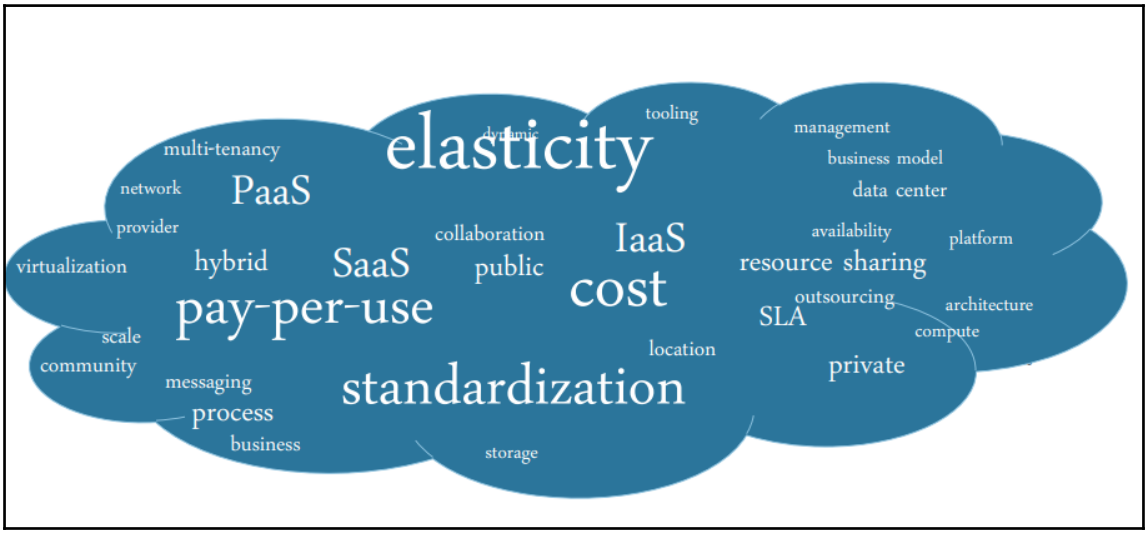
In this chapter, we will cover the following topics:

- Cloud computing concepts
- Architecture models:
    - Public
    - Private
    - Hybrid
    - Community
- Service offerings:
    - PaaS
    - IaaS
    - SaaS
    - IDaaS
    - NaaS

- Market vendors:
    - Amazon
    - Azure
    - SalesForce
- Cloud computing security
- Cloud backup solution

Cloud computing technologies

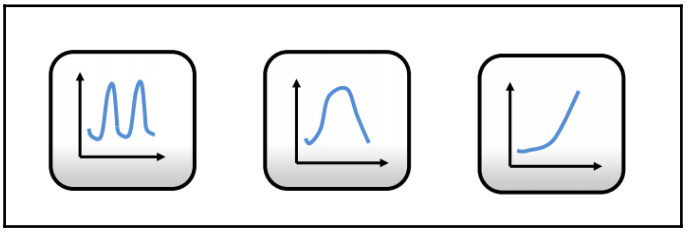
Cloud technologies encompass a wide umbrella of multiple service offerings and underlying technologies, as depicted in the following diagram:



The increasing popularity of cloud computing has resulted in its wide adoption across industries such as banking, insurance, hi-tech, pharmacy, manufacturing, and so on. Big data systems ingest huge volumes of data from multiple source systems with variety of formats. Therefore, its quite economical to stage or store data on cloud compared to on-premise systems.

Even with Internet of Things, sensor data from machines runs into terabytes and petabytes quickly, therefore only cloud systems are economical to hold such data volumes. With strategic and systematic adoption of business processes, applications, platforms, and infrastructures, businesses can accrue the following multiple benefits:

- **Dynamic load patterns and elastic scalability**: A well-known application of the cloud is where the workload is varying and unpredictable. In such scenarios, clients scale your IT capacity by off loading high-demand compute requirements to the cloud without provisioning for peak-load capacity, thus saving money for the organization.

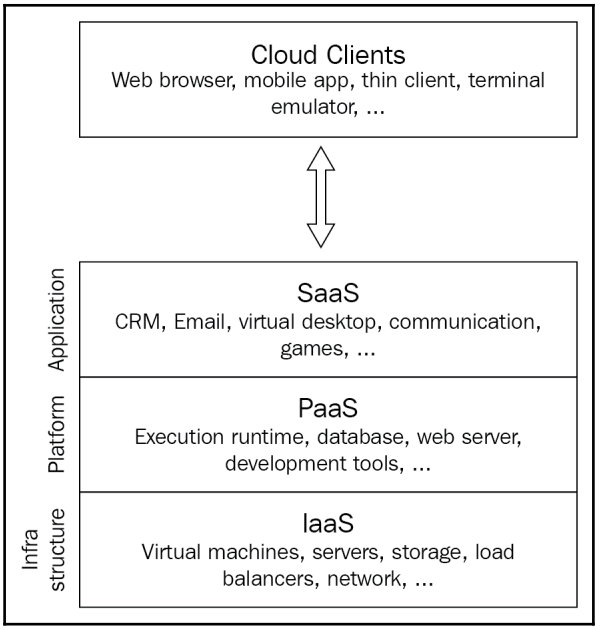


- **Quick setup**: The capacity sizing and procurement time for the resources is drastically reduced multifold, so with cloud setup, application deployments are relatively easier and faster; allowing for the offloading of basic IT requirements to the cloud service providers.
- **Pay as you go**: Pay for what you need and use. The big advantage is in saving the effort and cost for many of the products used to support the network and systems, such as spam/antivirus, encryption, data archiving, email services, and off-site storage.
- **Higher value**: The administrative activities for the IT support systems move to the cloud service provider so internal IT resources can focus on higher-value business activities, and capital expenditure can be directed for more pressing business investments.

The adoptions for cloud are across the following layers:

- Business process
- Application
- Platform
- Infrastructure

We will discuss these in detail in the coming sections. They are also represented in the following diagram:

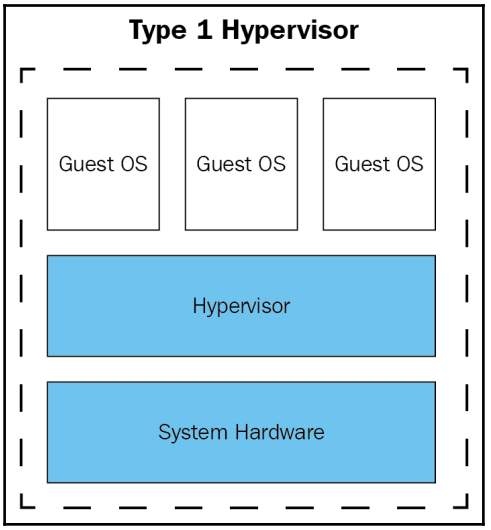


Cloud technology concepts

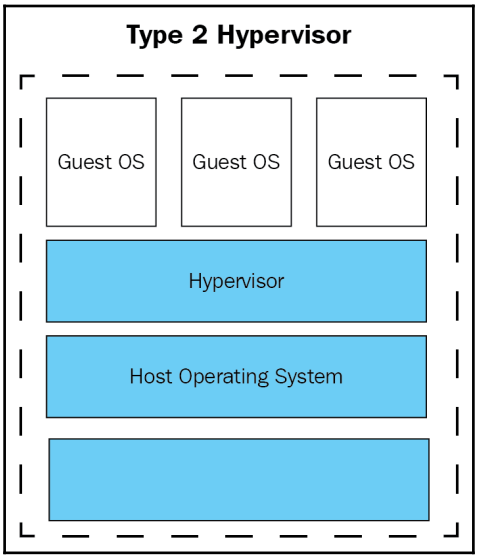
Virtualization is one important technology often referred to alongside cloud technologies. It is the prime technology framework associated with cloud computing to create various dedicated execution environments sharing the same physical infrastructures, such as an operating system, server, storage device, or network. Virtualization software manipulates hardware, while cloud computing refers to the service, and delivers the value of shared computing resources software or data as a composite service.

Virtualization is creating a virtual machine over bare metal, or an existing operating system with hypervisor.

There are a few models, such as Oracle VM, Sun xVM Server, LynSecure, and VirtualLogicVLX, that have hypervisor running on a bare system without an operating system:



Another category involves having hypervisor software on top of an existing operating system. A few examples are VMWare Fusion, Virtual Server 2005 R2, Containers, Microsoft Hyper V, Windows Virtual PC, and VMWare workstation 6.0:



Virtual Machine Monitor (**VMM**) or virtual manager, is technology to create virtual versions of a device or resource to run multiple applications on multiple divergent operating systems on the same server simultaneously. DevOps ensures the configuration is consistent between environments like development, QA, pre-prod and production. The usage of virtual machines on cloud and their replication consistently with DevOps is effective mode to quickly turn around. A hypervisor or VMM installed on top of the operating system makes servers, workstations, storage, and other systems independent of the physical hardware layer. A **virtual machine** (**VM**) is an emulation of a computer system based on architectures, and provides the specialized functionality. A host machine is a computer on which a hypervisor runs one or more virtual machines, and each virtual machine is called a **guest machine**.

Virtualization offers many benefits, including the following:

- **Maximizing resources**: Traditionally built systems are mostly underutilized. Virtualization can reduce the number of physical systems to acquire by efficiently using the capacity. Virtualization helps to maximize the use of the hardware investment to get more value out of the servers.
- **Multiple systems**: These can help you run multiple types of applications and even run different operating systems for those applications on the same physical hardware, such as Linux and Windows, on the same server and deploy the respective applications.

Virtualization can be full virtualization or para-virtualization.

Full virtualization is where the server with all the software is emulated to another system. It's more applicable in scenarios such as the following:

- Sharing a computer system among multiple users
- Isolating users from each other and the control programs
- Emulating a server system to clients or host systems

Para-virtualization allows for multiple OS to run on a single hardware device, and it's more apt in scenarios such as the following:

- Convenient migration to new systems, as guest systems can be easily moved
- Capacity management to add additional resources is easily managed
- Disaster recovery for guest instances to be managed

Virtualization offers benefits such as systems consolidation, and fully-managed virtualized and cost-efficient systems.

Cloud solutions beyond virtualization, however, offer the following capabilities, which we will discuss in detail:

- Self-service
- Elasticity
- Automation management
- Pooled resources and scalability
- Pay as you go service

Stateful, stateless servers, and REST protocols are frequently referred with cloud computing technologies.

Stateful servers maintain a user's state information in the form of sessions. Once a user logs into the website, a unique identifier is created and tracked by the web service for that session. The session information will be retained for all other requests by the user till the browser is closed; this session information helps to offer a personalized service to the user. The details of the sessions also help to monitor concurrent web traffic and incorporate security features, such as preventing a malicious user trying to flood the web server with multiple simultaneous requests through an established connection.

Stateless servers, as the name indicates, do not maintain any state information for the user, so each request is completely independently handled as a new request unrelated to the previous request.

Web API's design follows stateless architecture, and uses the **REpresentational State Transfer** (**REST**) protocol. The user identity in this stateless design is through a unique user ID allocated to the user, which needs to be passed with every request.

REST is an architectural approach focusing on resources (or data) as a prime objective, and not on the function (functionality) that should be provided by the API. It provides better scalability, interoperability, and is aptly used in the following scenarios:

- Client server
- Stateless servers
- Cacheable web systems
- Layered system
- Code on demand
- Uniform interface

REST is a set of constraints that ensure a scalable, fault-tolerant, and easily extensible system adopted by the **World Wide Web** (**WWW**). A RESTful API can use other transfer protocols, such as SNMP, SMTP, and others. **HyperText Transfer Protocol** (**HTTP**) is a protocol used to link pages of hypertext in the WWW. This is also a popular way to transfer files apart from other transfer protocols available, such as FTP, gopher, and so on. RESTful APIs use HTTP as a transport layer in most scenarios, since the infrastructure, servers, and client libraries for HTTP are widely adopted. A RESTful API adheres to all the REST constraints, and an HTTP API makes use of HTTP as its transfer protocol, including SOAP APIs relaying on HTTP.

Working with REST APIs is like the navigation between websites. HTTP and HTTPs are two-key protocols used in implementations of REST APIs. A **Uniform Resource Identifier** (**URI**) is used as an address by the REST API to access a particular website. The URI is comprised of two parts--a **Uniform Resource Locator** (**URL**) and a **Uniform Resource Name** (**URN**). The URL represents the location of a specific resource on a computer network, and defines a mechanism for retrieving it. A URN is the name of a web resource.

For example, `https://en.wikipedia.org` is our URL, and `/wiki/Cloud_computing_architecture` is our URN.

The combination `https://en.wikipedia.org/wiki/Cloud_computing_architecture` is a URI unique resource address or identifier.

There are four basic operations when working with any REST API to interact with a server through the HTTP protocol. They are GET, POST, PUT, and DELETE.

- GET will retrieve the element addressed
- PUT will create or replace the addressed element
- POST is used to create another element
- DELETE will delete the addressed element

These are also used to get the status of a call response; for example, the HTTP status code of 404 is for a page or address *Not Found*.

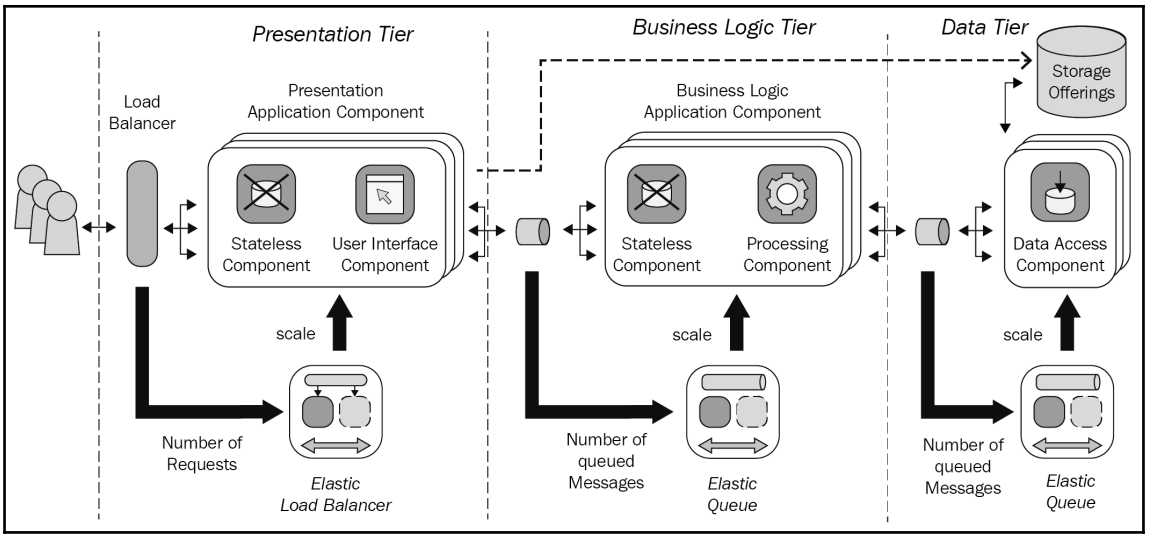
The two types of data formats to present the output results when working with REST APIs are **JavaScript Object Notation** (**JSON**) and **eXtensible Markup Language** (**XML**). The advantage of JSON is that it's easier to parse, and hence more preferred to XML in most cases. While XML has some more advanced features in terms of result validation and navigation, it's more complex. Using `.json` at the end of the URI, JSON formatted output is delivered.

Authentication and security

REST APIs through HTTP protocol ensure authentication, a security mechanism, and data protection, they encrypt traffic between the client and server. Some APIs rely on GET requests to view data without authentication; however, for any data modifications, authentication is strictly enforced. Username and password for HTTP authentication is basic model called with Base64 hash, which not a scalable and secure model, token-based authentication mechanism known as **Authtype 1/2** is now more prominent for API calls.

Multi-tier cloud architecture model

Cloud architecture also follows the pattern of layered representation of presentation, business logic, and data to support the application multi-tier models. A typical architecture model with underlying components is shown in the following picture:

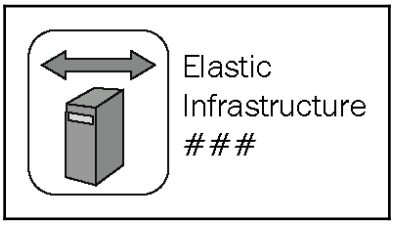


This layered architecture model is supported by popular cloud vendors offering multiple value-added features and functionality at each service layer as listed.

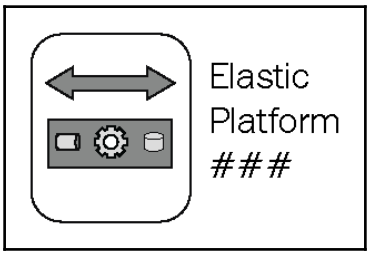
Presentation tier

The presentation tier is the frontend user-facing and interacting tier. It displays on the website/webpage the information related to available services by communicating with other tiers. It sends the results to the browser for the service and query requests by coordinating with other tiers in the network such as business and data layers:

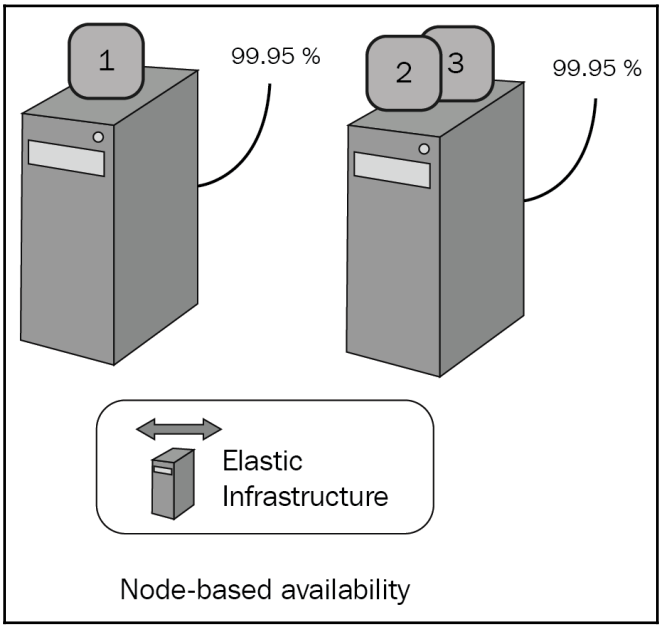
- **Elastic infrastructure**: To meet the real-time demand, hypervisors are enabled to create virtual machines or containers towards cloud infrastructure elasticity with the resources. They are enabled with self-service BI for features such as monitoring information about resource utilization, traceable billing, and provisioning of resources.



- **Scale vertically or scale-up**: Moving the application to a bigger virtual machine, or by resizing the VM (claiming back the virtual machine's capacity) to accommodate a scalable environment for the application is termed as vertical, or scale-up. This is associated with multiple dependencies.
- **Elastic platform scalability**: This is often associated with the application layer for its capability to handle a growing amount of work. It could be possible to add or append the system, network, or process in order to cater to the business growth.

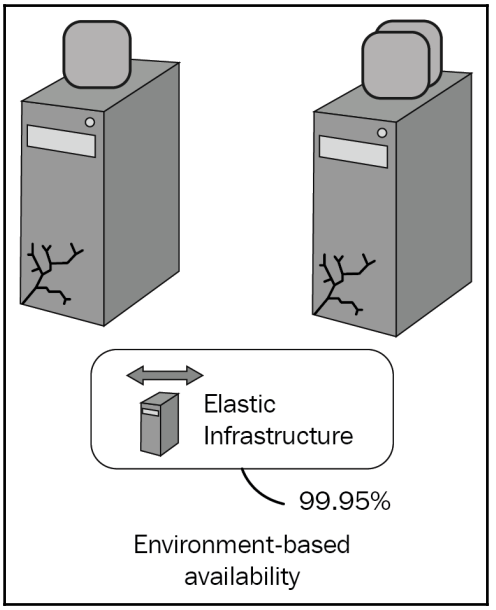


- **Scale-out or scale horizontally**: This is achieved by provisioning more instances of the application tiers on additional machines (through additional resources), and then dividing the load between them. Though horizontal scaling allows the re-division of resources between applications by provisioning, the application architecture should be able to scale by additional nodes, and by distributing the load.
- **Node-based availability**: Most legacy applications have been developed to run on a single machine. This is where each node plays a unique role for hosted applications, and its availability is measured. These applications require recoding to adapt for both the scalability and elasticity that the cloud provides:



Node-based availability

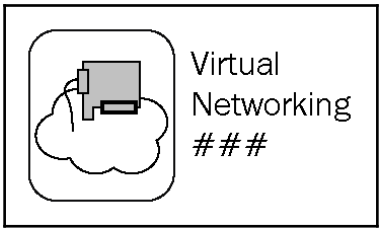
- **Environment-based availability**: Applications built with multi-threaded architecture benefit from cluster-based environments with built-in elasticity and scalability. This is crucial with uneven usage, or spikes during certain periods. The applications should be designed to detect variations in the real-time demand for resources, such as bandwidth, storage, and compute power and automatically adopt:



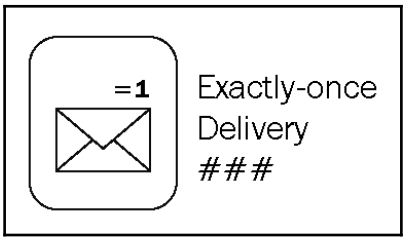
Business logic tier

The business logic tier is also called the **application tier**. As the name indicates, business logic performs all the processing required by the application to send the requisite details for the presentation tier by communicating with the underlying data tiers.

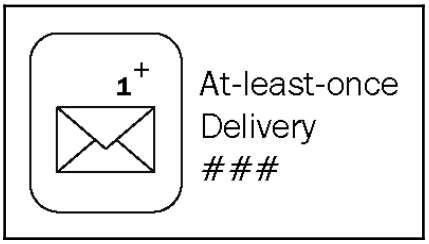
- **Virtual networking**: This is a technology to access remote systems on the networks. It is also called **remote desktop sharing**, and enables visual display and control of one system through other protocols, such as **Remote Frame Buffer** (**RFB**), **Remote Desktop Protocol** (**RDP**), and **Apple Remote Desktop** (**ARD**).



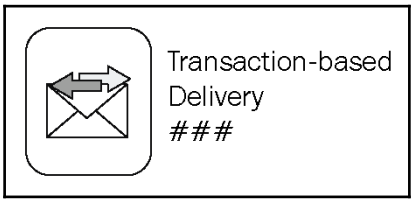
- **Message-oriented middleware**: **Message-oriented middleware** (**MOM**) enables the sending and receiving of messages between application modules distributed over heterogeneous platforms. MOM reduces the complexity of developing applications that span multiple operating systems and network protocols through software or hardware infrastructure support. The following are some of the variants of MOM:
    - **Exactly-once delivery**: For many, critical systems require exactly-once delivery of messages. By using filter conditions duplicate messages are avoided.



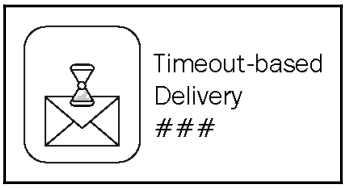
- **Atleast-once delivery**: To ensure that the messages are delivered at least once without loss, for each message, an associated acknowledgement is added.



- **Transaction-based delivery**: This ensures that the message is fully delivered, and received as a transaction with ACID compliance.



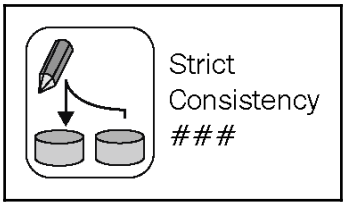
- **Timeout-based delivery**: In this process, the message is not deleted from the system until the client system acknowledges receiving and reading the message. So, messages are stored in the system till that time, hidden from client systems as backup.



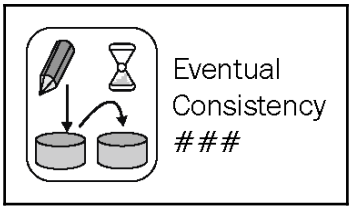
Data tier

The data tier is the repository of the data; it supports the presentation/business/application tier for processing the browser request. The request from the browser is translated into the application logic by sharing the underlying data, ensuring the confidentiality and consistency per the business demands.

- **Strict consistency**: Multiple replicas of data are maintained at different locations to serve data redundancy in the event of failures, to ensure consistency of replicas, and also improve response time. Data replicas can be either read or write operations; their equation N (number of replicas) < R (replicas access to read) +W (replicas accessed to write), N <R+W, ensures data consistency.

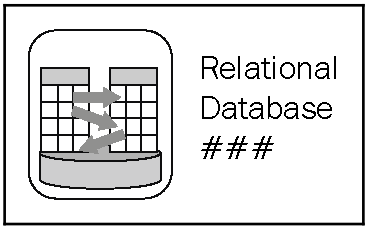


- **Eventual consistency**: Eventual consistency is employed by data stored at different locations; data is not consistent all the time so that performance and availability of the data are ensured with network partitioning. All replicas are updated with data alterations by asynchronously propagating over the network.



Relational databases

Many of the traditional relational databases, where data is structured according to schema and enforced with data manipulation, such as Microsoft SQL Server, Oracle Database, MySQL, IBM DB2, and Amazon Relational Database Service, are supported.



NoSQL database

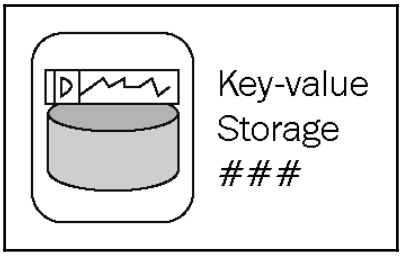
Cloud platforms offer and support many of the new age NoSQL databases versions, such as the following:

- Document databases are offered, such as Couchbase Server, CouchDB, DocumentDB, MarkLogic, and MongoDB
- Graph databases are available, including AllegroGraph, IBM Graph, Neo4j, and Titan
- Column stores are offered, such as Google Bigtable, Cassandra, and HBase

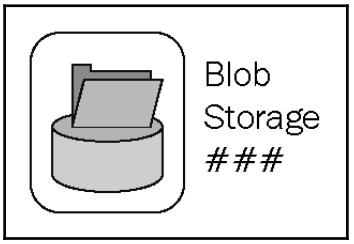
Data storage

Cloud platforms support many types of storage, including the following:

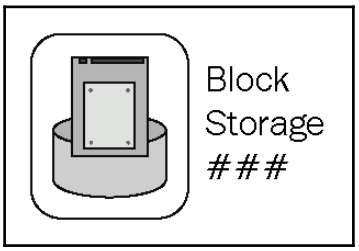
- **Key-value storage**: We have seen that in the Map Reduce framework, the values are stored as a key-value pair, there is no schema, and the value of the data is in the form of numbers, strings, counters, JSON, XML, HTML, binaries, images, short videos, and so on. In a key-value storage, or key-value database, values are identified and accessed via a key. This form of data storage is appropriate for storing, retrieving, and managing associative arrays. This method of designing a data structure is referred to as a dictionary, or hash. Some popular databases are Aerospike, Apache Cassandra, Berkeley DB, Couchbase Server, Redis, Riak, and so on.



- **Blob storage**: Blob is a binary large object. It is used for storing large amounts of unstructured binary data, such as text or binary data that can be accessed through web protocols, such as HTTP or HTTPS.



- **Block storage**: Block blobs are composed of storage blocks with different sizes, and are identified by block ID. Block storage is ideally suited for streaming data, where a large video is split into smaller segments and uploaded in parallel to decrease the upload time, and assembled. This is applicable for file-sharing applications to file blocks uploaded to the storage service in parallel and then assembled into a block blob:

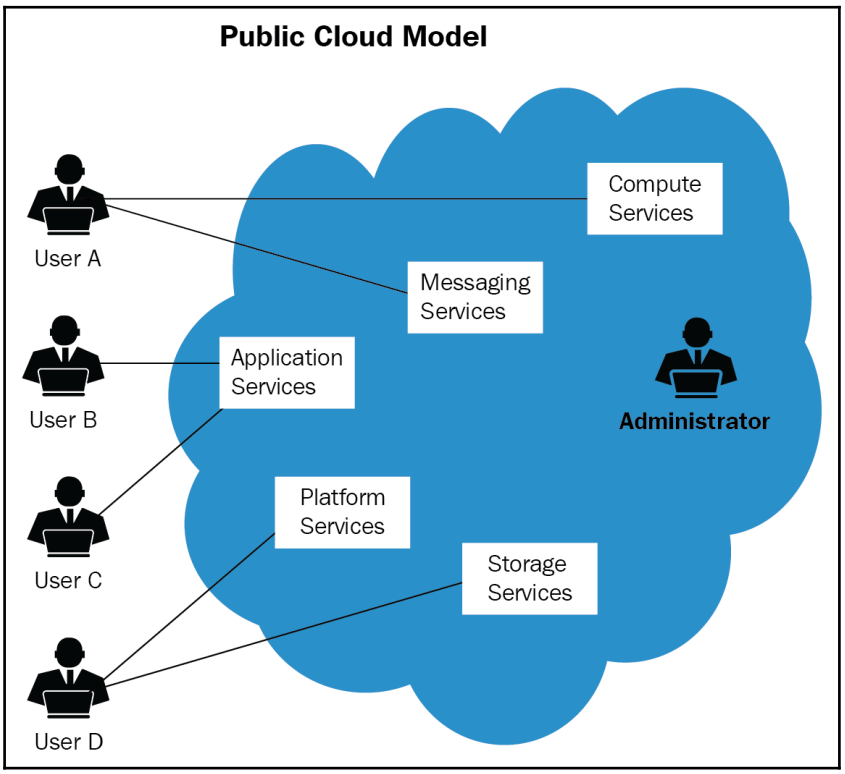


Cloud architectures

The most common implementation models are presented next.

Public cloud

In a public cloud, the applications and services are hosted on a public cloud platform. The systems and services are easily accessible to the general public; for example, Amazon, Microsoft Azure, Google, Salesforce, and so on:



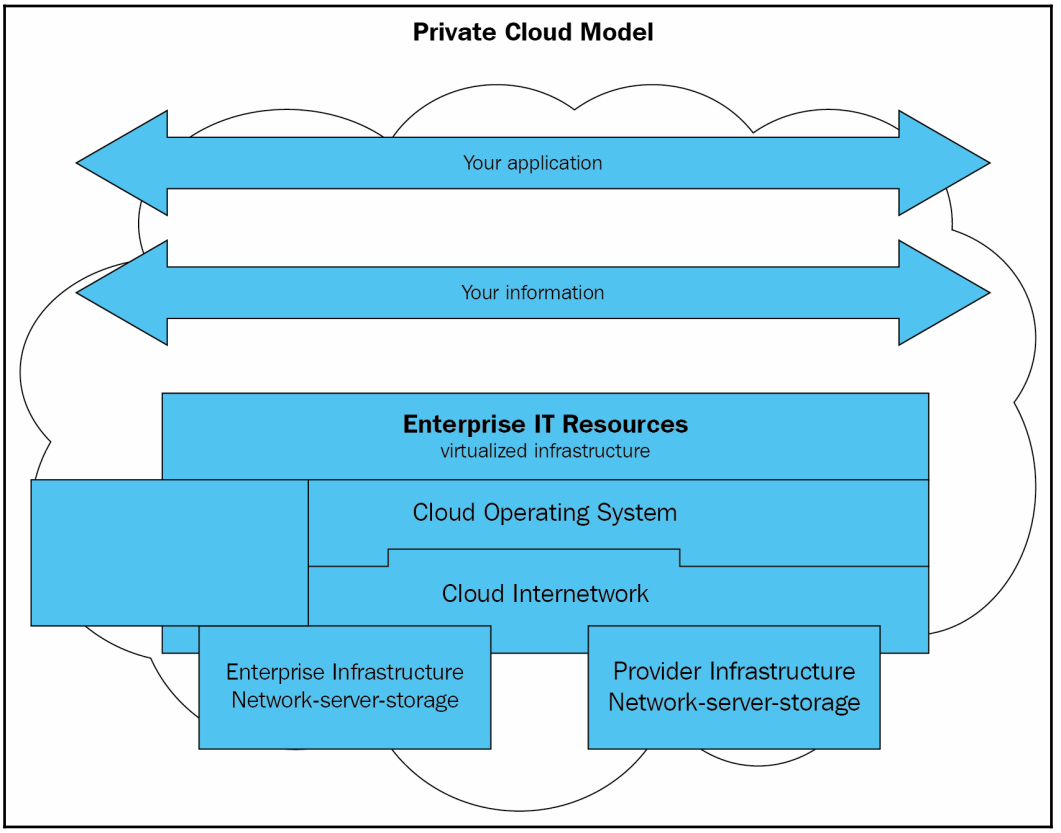
Benefits of the public cloud include the following:

- **Economical scale**: The public cloud is cost-effective by sharing the same resources with a large number of consumers
- **Reliability**: The public cloud maintains replicas at different locations; they serve for redundancy in case of failover, providing a high level of reliability
- **Elastic scalability**: The public cloud can be flexibly scaled up or down, depending on demand from a pool of resources

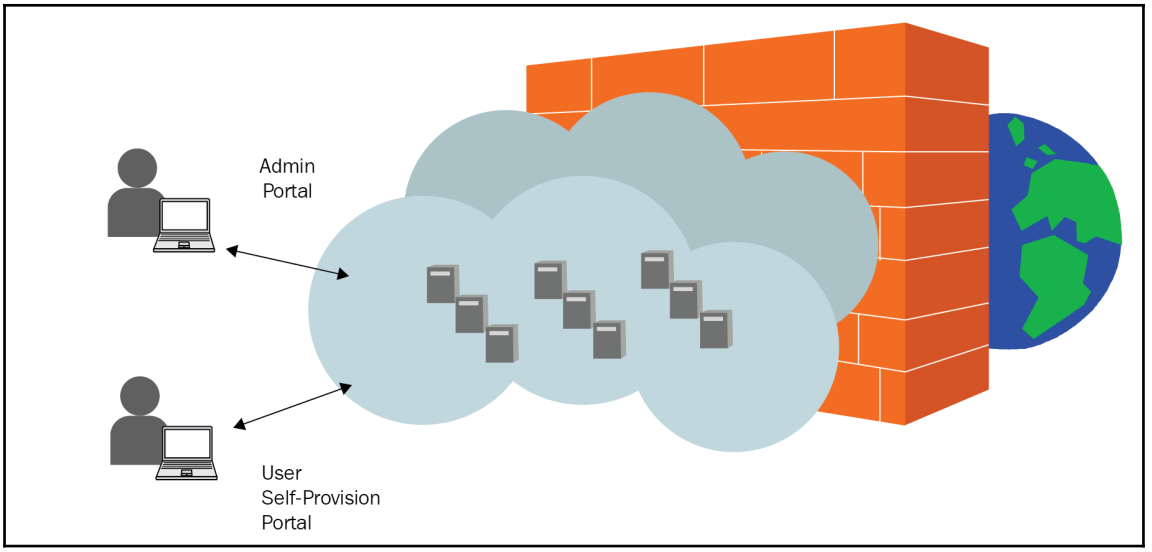
- **Flexibility**: Many options to choose among vendors, and also to integrate the public cloud with the private cloud and so on, are available
- **Pay per use**: This feature enables that resources are accessible whenever a consumer needs them

Private cloud

In a private cloud, the applications, operating systems, cloud infrastructure, and storage are all inside the private cloud. They are operated solely for the purpose of an organization. The private cloud can be managed internally by the organization nor by a third party:



The big security feature is that a firewall separates the private cloud from the external world, enhancing the security multifold from data theft:



The benefits of a private cloud are that privacy and security of organizational information is maintained within the organization, and there is more control within the organization. Although the cost is higher to adopt for the organization, it also offers higher efficiency though enhanced security.

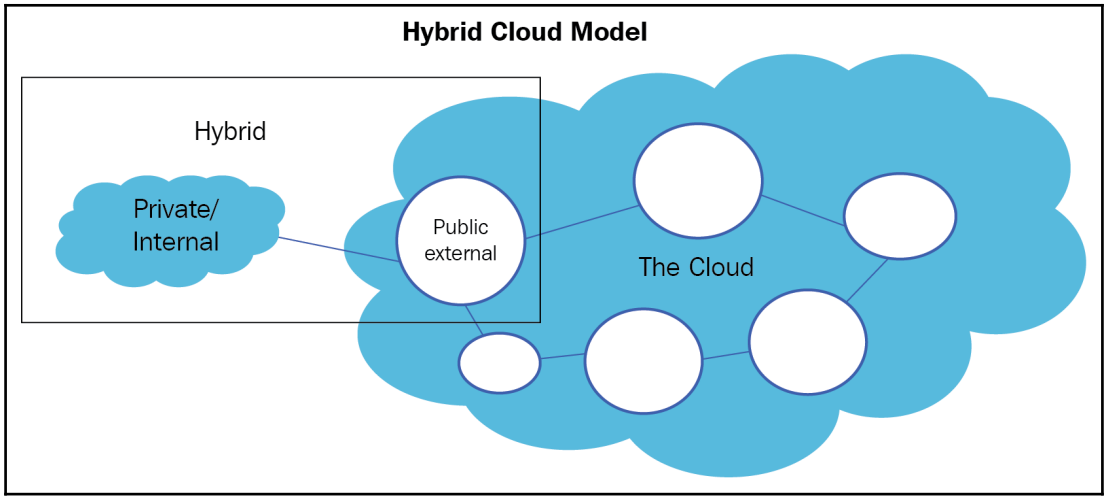
The challenges are that a private cloud's scalability is limited and depends on the organization's resource availability. Procurement could lead to inflated or inflexible pricing. The private cloud may be confined to a local geography, and may not extendable to other regions.

Hybrid cloud

A hybrid cloud offers the best of both private and public clouds, including:

- Security features customized and incorporated as per the organization's needs
- Scalability features, architecture flexibility, and also the optimization of the cost efficiencies balancing between public and private models

The challenges are network integrations and associated complexities, and security compliance while integrating the public and private clouds:



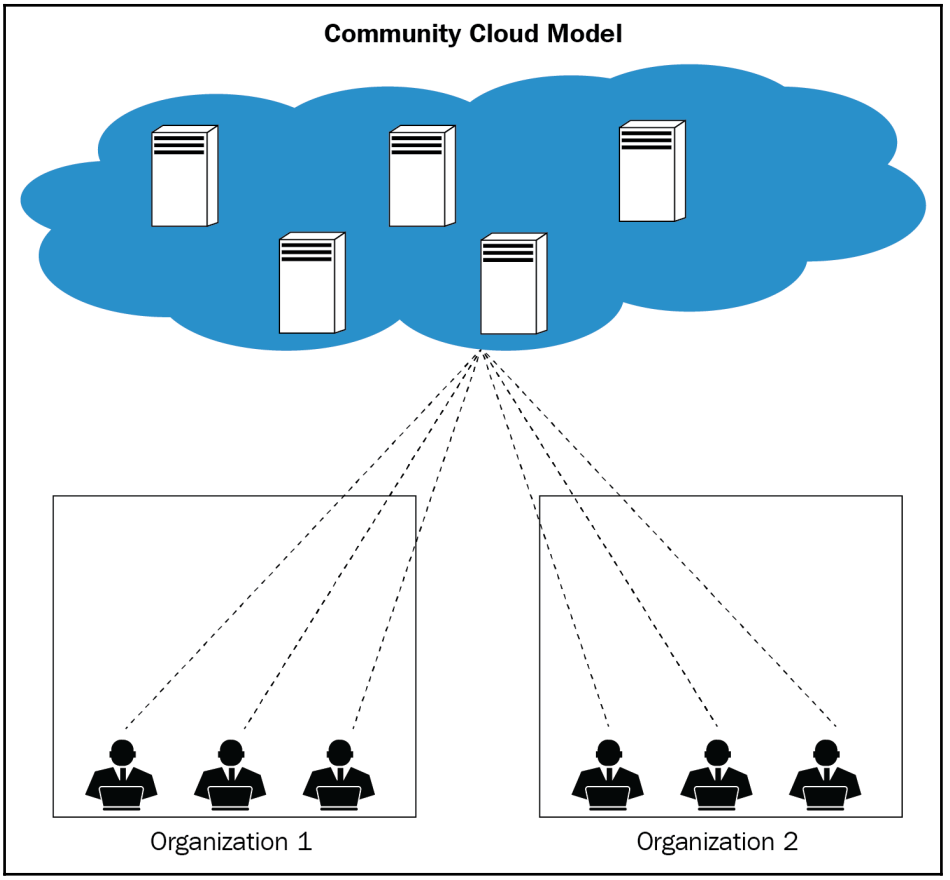
Community cloud model

The community cloud facilitates a group of institutions, sharing systems, and services, to be accessible by all the partner groups; for example, in education, research can be set up across institutes, and the infrastructure and services could be managed by a third party. It could also be effective between various supplier groups of a large organization for effective collaboration.

Benefits include cost-effectiveness, and the advantages of quick community collaboration.

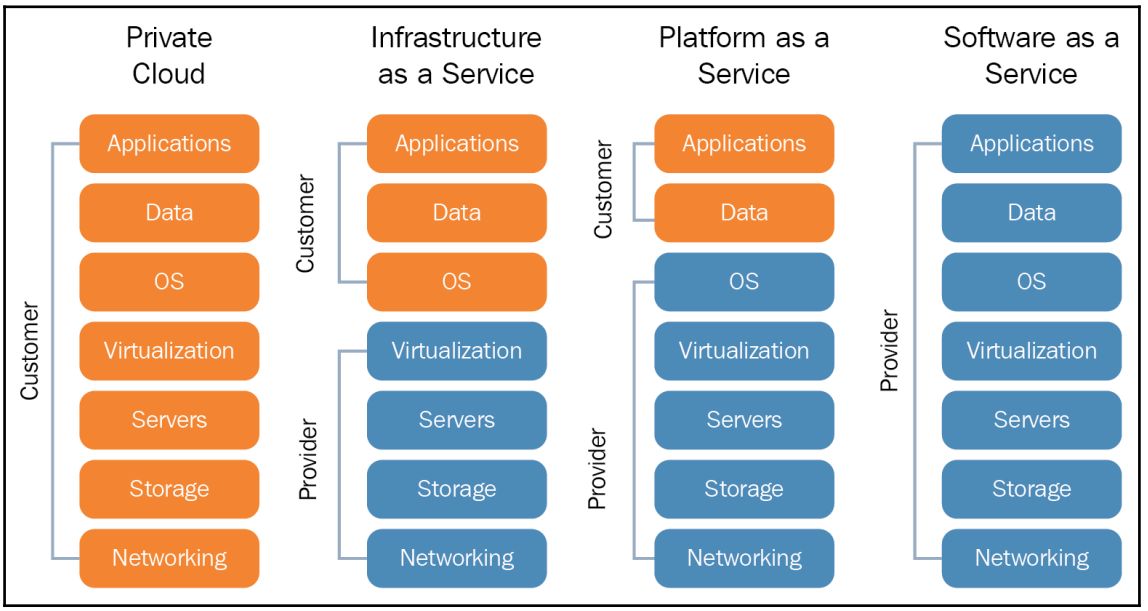
Security features are optimal as per the community needs, and are better than the public cloud.

Challenges could include data governance, security, and cost. Since all the data is hosted together, security should be stringent so as to avoid unauthorized access:



Cloud offerings

Cloud offerings are broadly categorized as the **Private Cloud**, **Infrastructure as a Service**, **Platform as a Service**, and **Software as a Service**. Each service variation depends on the layers managed between the provider and customer:



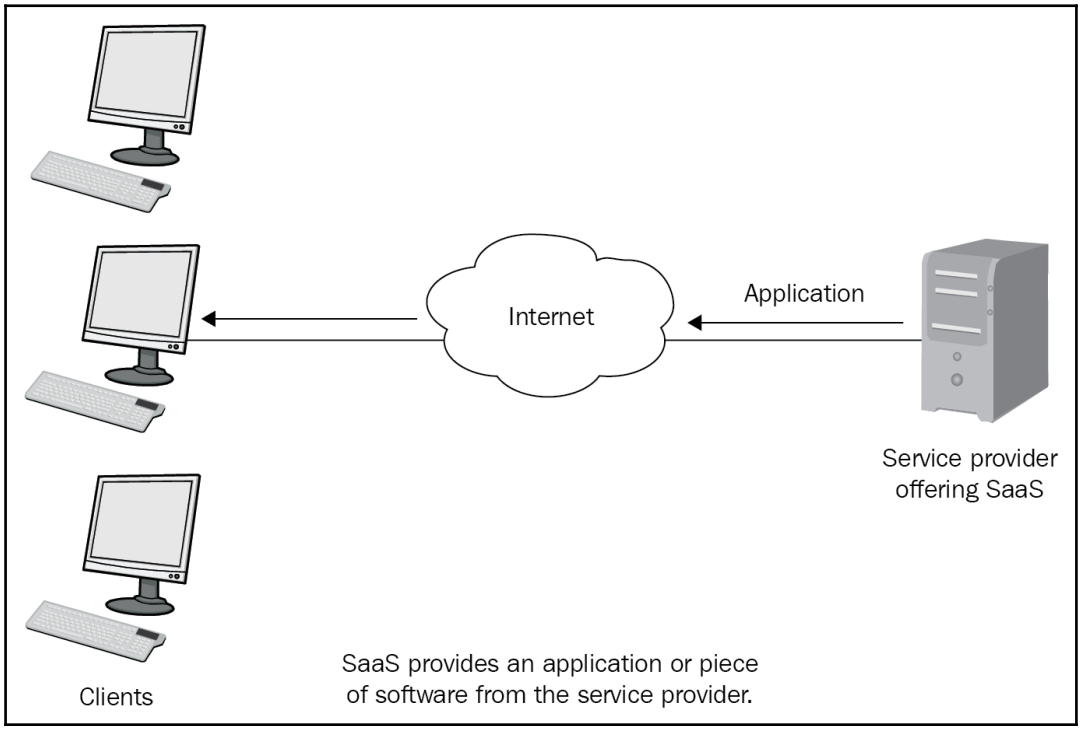
Software as a Service (SaaS)

As represented in the preceding diagram, in SaaS all the components and layers of offerings are owned by the provider.

SaaS is a model where applications are hosted on a cloud as a service that clients access through a web interface, such as browsers, over the internet. Unlike in the traditional system of installing on individual machines, clients don't need to install any specific applications on their machines nor maintain or support the applications on SaaS. The cloud provider is responsible for upgrading it or hosting any integrations, and so on. The provider does all the patching, upgrades, and ensures that the infrastructure is running, by charging fees appropriately.

SaaS applications are designed and built to support multiple concurrent users with web browser tools. These applications are popular for standard commercial usage in most cases, and in some scenarios offer custom-built applications, such as the following:

- Video conferencing
- Content management
- HR payroll applications
- Team collaboration and sharing the screens, applications, and so on:



Software as a service has a few options, as discussed next.

Single tenant

Every customer as a tenant has a separate instance of the software, and the entire supporting infrastructure serves that single customer. Each tenant has their own independent database and instance of the software. There is no sharing, so every tenant is isolated from each other. The advantages are as follows:

- Enhanced security
- Reliability of services
- Easy to backup and restore
- Easy to migrate from SaaS to self-hosted
- Flexibility
- Upgrade control

Multi-tenancy

Multi-tenancy means multiple organizations accessing the same server or resources; each organization (or company) is treated like a tenant. The architecture has a centralized compute, storage, and networking, serving to multiple customers by sharing the infrastructure and software. Each customer shares the supporting infrastructure and a single database. To each tenant, the system looks like it has been customized for them. The data is partitioned between the customers and encrypted to serve them separately; all the customers share the same copy of the application code, making it a multi-tenant cloud application. Fewer IT resources are needed to patch, maintain, and upgrade the application (reduced TCO).

Multi-tenancy is achieved by two methods: multi-tenancy through instance replication, and through data segregation effective multi-tenancy. Instance replication becomes an overhead as the number of tenants grows and accordingly the VM's explode, causing performance and maintenance overheads.

Multi-instance

Multi-instance instances are deployed on a per-user demand to scale horizontally and infinitely the multi-instance cloud. A separate application logic and database processes are deployed for every client, together with a dedicated set of unique instances as per the organization's needs. The multi-instance system offers data isolation, as customers are not sharing databases and infrastructure.

The multi-instance environment offers many benefits, including greater flexibility, and higher data security, preventing unwarranted intrusion. Its architecture makes it easy to deploy and scale to meet higher performance and SLA's adherence or 24×7 seamless service experiences. It facilitates for better manageability of product updates, upgrades, configuration management, and customization. The customer's services can continue to be available, while migrating to an on-premise server, or to another cloud for routine maintenance and unexpected issues. To create and manage multiple application instances, it's an effective model.

The challenges are that to maintain and upgrade the multiple-instance environments, such as databases and web application instances, it is expensive compared to the multi-tenancy model as propagation of functionality, or features from one instance to multiple instances is not supported.

Benefits of SaaS

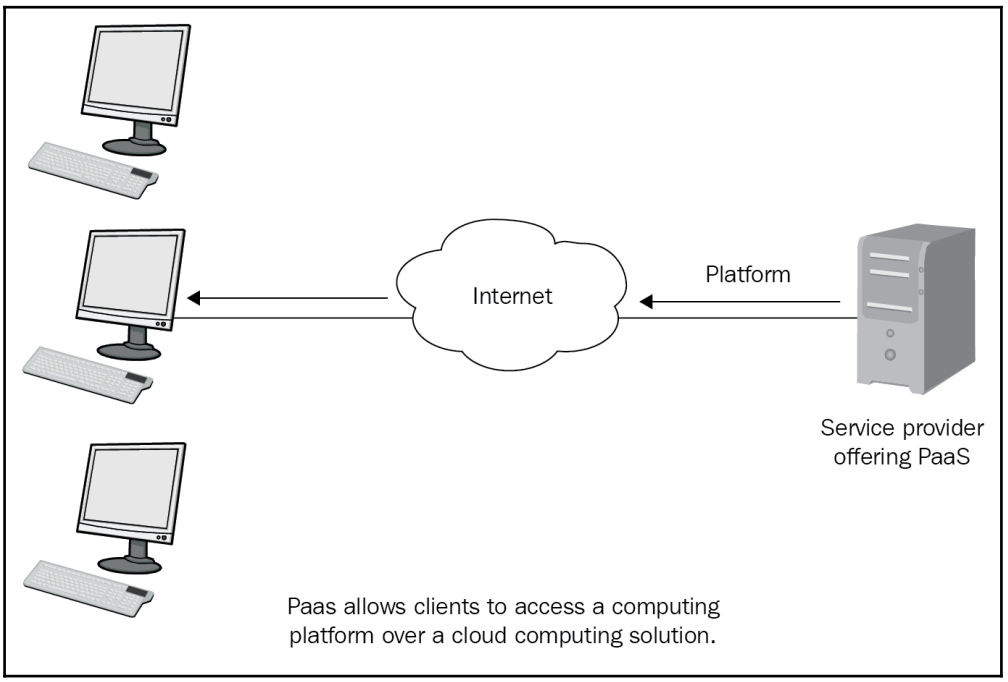
The benefits of SaaS are the following:

- Quick setup time to enable the business to adapt to the SaaS platform
- Lower procurement cost for IT system's infrastructure, hardware, licenses, software, and so on
- Lower maintenance costs in terms of upgrade of software, and so on
- **Secure Sockets Layer** (**SSL**) is well established for secured access, and there is no need for VPNs
- SaaS applications are quite reliable and stable

Some challenges with SaaS include low-security features, and it is less customizable.

Platform as a Service (PaaS)

In PaaS, the users are offered an infrastructure, along with a platform for their customized requirements, to develop into applications, and the ability to deploy and maintain them as well:

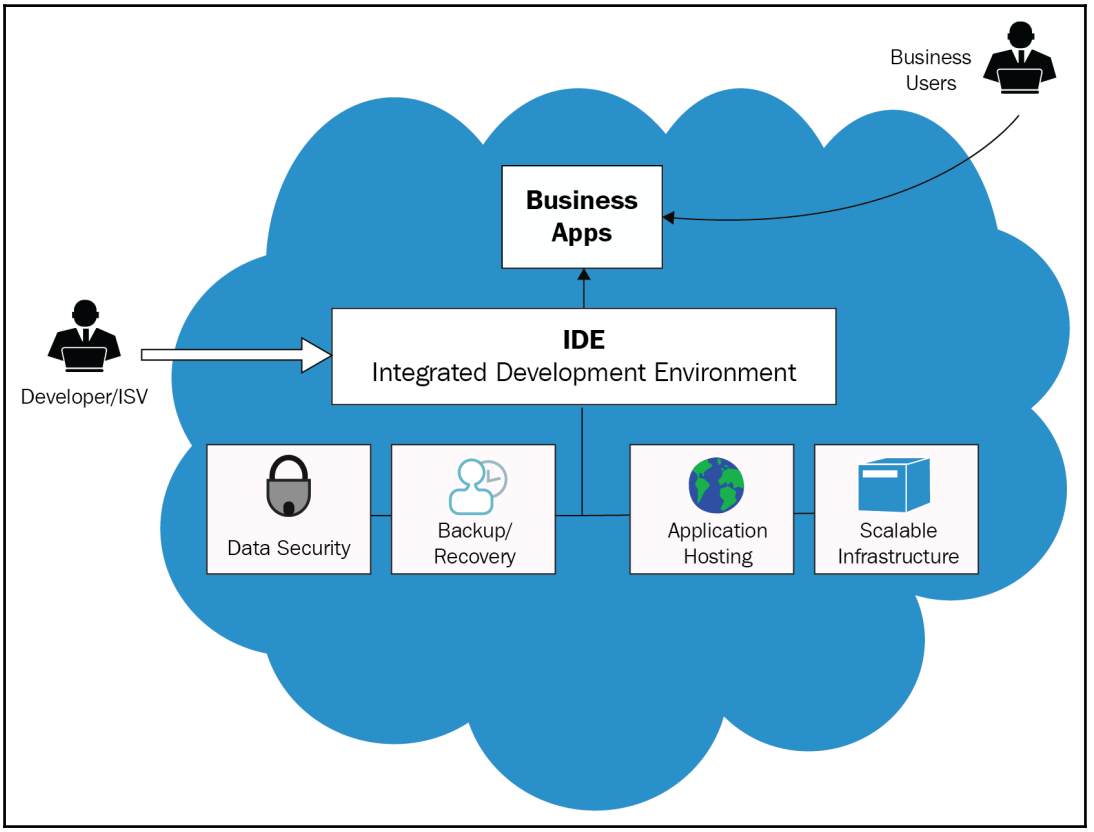


There are different types of Platform as a Service:

- PaaS Development as a Service--standalone development environment.
- PaaS with Data as a Service
- PaaS with Database as Service
- PaaS associated with SaaS
- PaaS along with the operating environment
- Open source platform PaaS

Development as a Service (DaaS)

Development as a Service is a web-based, community-shared development tool equivalent to locally installed development tools in the traditional (non-cloud computing) delivery model. In the standalone development model, the runtime environment, along with integrated development tools, is offered for application development. Many choices of open source development and deployment tools are available to develop applications, and are included with the package:



Data as a Service with Paas

Data as a service is often considered as a specialized subset of a Software as a service offering. It's a web-based design construct whereby data from cloud-based application is accessed through some defined API layer, underlying data layer of cloud based web application could be RDBMS, filesystem, or Amazon S3.

Database as a Service with Paas

Database as a Service is like a Platform as a Service offering, with databases such as traditional RDBMS or non-relational databases as choices.

PaaS tied to SaaS environment

Many of the successful product vendors have created a business model for their partners and **independent software vendors** (**ISVs**) to co-develop applications to leverage from extended talents, cost benefits, and speed to market. To support this, the ecosystem or environment is created as PaaS, enabling the partners and ISVs to develop as per the stipulated requirements.

A few examples of PaaS that are anchored to an SaaS environment to collaborate are:

- **Force.com by salesforce.com**: Customers and ISVs can develop on the `salesforce.com` ecosystem and can sell by their own modes or by the `salesforce.com` application marketplace, AppExchange.
- **Workday PaaS**: With workday solutions, customers can integrate applications and leverage services for building, configuring, testing, and deploying integrated solutions. Example financial management, human resource SaaS solutions, and so on.
- **Google App environment**: This is PaaS for ISVs to develop applications for Google Apps. The Google Apps engine is a scalable model, and allocates resources as per the elastic demand. The Google Apps marketplace is a store house of all the Google Apps, and offers a search facility as well. An open source cloud platform for the Google App engine is AppScale, which can be deployed on both public clouds and private clouds.
- **Intuit developer network**: This is PaaS for QuickBooks accounting software for both on-premises and cloud options. ISVs can develop customized software and integrate and market their applications through the Intuit marketplace.

PAAS tied to an operating environment

Many platform (IaaS) vendors have moved away from just nuts and bolts providers of operating systems, networking, and so on; they are providing other value-added offerings such as tools for application development and performance metrics. They offer platforms along with an operating environment to enable better value-added offerings so that developers can build or deploy applications with this support.

Examples of PaaS tied to an operating environment include the following:

- **Windows Azure**: Maturing from just an infrastructure (IaaS), Azure is offering many tools from Windows, SQL abstractions, development tools, management, and services, to make it a fully-fledged PaaS.
- **AWS Elastic Beanstalk**: This is a PaaS that Amazon packages with **Amazon Web Services** (**AWS**) for deploying applications. Elastic Beanstalk has in-built automation, which benefits organizations to automate, auto-scale as per the demands change, and control for the underlying IaaS resources.
- **AT & T Platform as a Service**: AT & T offers synaptic **Compute as a Service** (**CaaS**) to easily build and deploy applications from scratch. Pre-built application templates are facilitated on this platform.
- **IBM SmartCloud**: This is a hybrid offering with the ability to shift workloads with increases in demand to public clouds. The applications supported are Java, web, and enterprise applications on their private cloud.

Open-platform PaaS

Open-platform PaaS are not tied to any single cloud implementation; developers can develop their own custom tools or open source tools as per the platform. Migration between clouds is convenient by an open platform PaaS. These platforms are well suited for a hybrid cloud environment, supporting deployment on both public and private clouds.

Here are some examples of open-platform PaaS:

- **OpenShift**: From Red Hat, this is fully integrated with advanced offerings also with the JBoss application server and middle layer. Applications supported are Java, Python, Perl, PHP, Ruby, and so on.
- **Cloud Foundry**: This offers VMware public clouds, vSphere, and vCloud, for building, deploying, and operating applications based on the cloud, supporting a wider range of languages, such as .NET, Java, Scala, and Ruby.

- **Engine Yard**: This is a fully managed PaaS used in conjunction with a number of private clouds and public clouds such as Azure, and AWS. Applications can be built for languages such as Ruby on Rails, and PHP.
- **CloudBees**: This is a platform that offers a full life cycle application for public and hybrid clouds. It's primarily a Java-based platform to support, build, test, run, manage, and develop, independent of the underlying platform.
- **OrangeScape**: This offers support for a private or public cloud, and offers portability as well. The main feature is process-oriented application development with OrangeScape Studio for non-programmers. It has pre-built business templates ready for use.
- **Apprenda**: This is specific to .NET applications development predominantly for Azure PaaS. Organizations can redeploy the applications to other on-premises, or public clouds.
- **DotCloud**: This is a multi-stack solution with the flexibility of multiple languages, databases, caching, and messaging components. The languages supported on the platform are Java, Perl, Ruby, and PHP.
- **CumuLogic**: This is a platform for cloud services that supports automation, autoscaling, monitoring, resource management, and user management for Java-based platforms for both public and private clouds.

An open source platform as service would include the entire suite of open source applications. Cloudera Enterprise distribution offers an entire bundle of components for the big data Hadoop platform as Platform services comprising of 20 components (listed following) ranging from **Extract Transform Load** (**ETL**) applications to **Machine language Libraries** (**MLLs**) as a full suite of Platform as a Service:

Here, we are giving an example of the Cloudera Enterprise package for PaaS so it will be in accordance with its own specification.

- **Apache Sqoop**: Provides data movement between RDMS and HDFS; it is a highly scalable architecture.
- **Apache Spark**: This makes the processing of jobs faster and easier to write based on in-memory.
- **Apache Sentry**: Provides access control for Hadoop users, both role-based and granular level.

- **Apache Pig**: This is a batch analysis framework for the processing of large datasets with the high-level language.
- **Apache Parquet**: Provides columnar data and a compressed format for Hadoop quite efficiently.
- **Apache Oozie**: A workflow scheduler for efficiently scheduling all the Hadoop jobs efficiently.
- **Apache Mahout**: Machine learning libraries for various tasks such as classification clustering, collaborative filtering, and many more.
- **Apache Kafka**: Kafka messaging service for Hadoop based on publish-subscribe; its architecture is distributed, and resilient.
- **Apache Impala**: This is a SQL query programming language for HDFS, S3, or HBase, with high-concurrency and low-latency.
- **Apache Hive**: This is batch processing for ETL transformations of Hadoop data with the SQL framework.
- **Apache Hbase**: This is popular for columnar storage for Hadoop; it's scalable with random read/write access.
- **Apache Flume**: This is a stream data processing framework to collect and aggregate event data in real time into HDFS or HBase .
- **Apache Hadoop**: This is the most popular storage; it also offers data processing and resource management.
- **Apache DataFu**: Statistical evaluation-oriented user-defined functions for large-scale data analysis written in PIG.
- **Apache Avro**: This is the framework for data serialization with rich data structures in binary format and RPC.
- **Apache Crunch**: This is a framework with user defined functions written in Java for writing, testing, and running Map Reduce pipelines.
- **HUE**: Hadoop User Experience for web-based graphics user interfaces; it is user friendly to work with Hadoop data.
- **Kite SDK**: Software development kit with the Application Program Interface. It has examples and documentation for creating applications on Hadoop

- **Cloudera search**: This is the search engine for free-text, and is Google style for business users using Hadoop data.
- **Apache Zookeeper**: This is the resource coordination service. It is highly effective and reliable, and distributes the Hadoop environment **Infrastructure as a Service** (**IaaS**).

IaaS provides access to infrastructure resources. Customers can benefit from the service on lease to use for their own use as per the timeline agreed upon. The fundamental resources are physical machines, virtual machines, virtual storage, and so on.

Apart from these primary resources, IaaS also extends resources to the end user through server virtualization; they include virtual machine disk storage, **virtual local area networks** (**VLANs**), load balancers, IP addresses, and software bundles.

Infrastructure as a Service is a flexible and efficient mode of renting hardware computer resources such as virtual machines, storages, bandwidth, IP addresses, monitoring services, firewalls, and so on. The consumer has to pay based on the length of time a consumer retains a resource.

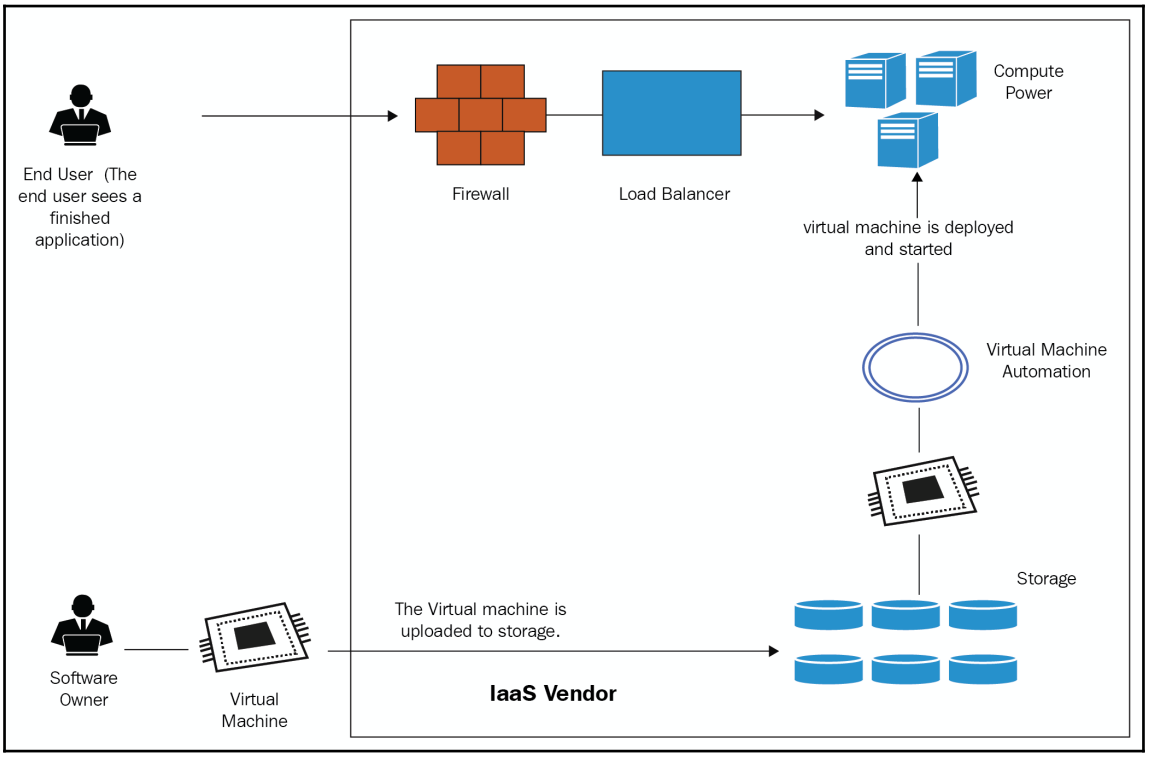
The process is also quite simple and convenient, with administrative access to virtual machines. The consumer can also run any software, start web servers, install new applications, or even custom operating systems.

IaaS platforms also support portability and interoperability between cloud platforms. For example, network applications such as web servers and email servers that are running on consumer-owned server hardware, can be ported to run from VMs in the IaaS cloud.

The challenges with IaaS are as follows:

- Similar to the challenges faced by PaaS and SaaS, network dependence and browser-based risks exist for IaaS.

- IaaS supports the consumer to run legacy software in the VM infrastructure; it could expose security vulnerabilities associated with such legacy software:



IaaS for Linux platforms offers all the following popular models on cloud as virtual machines:

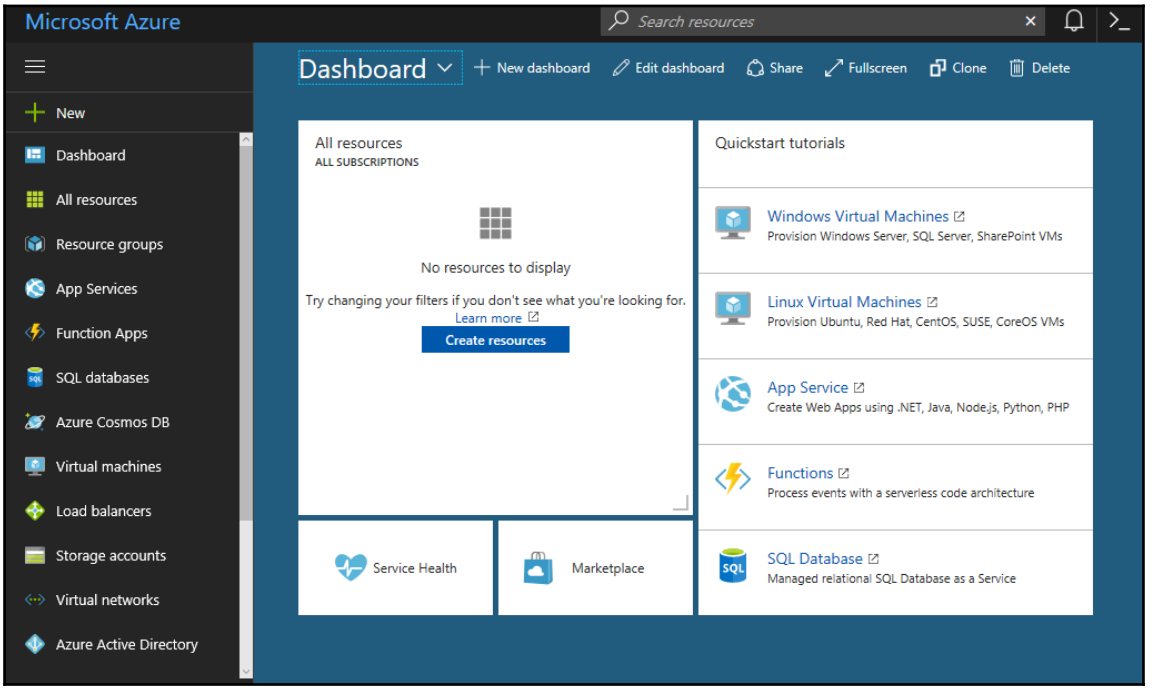
- CentOS
- Debian
- Kali
- Red Hat
- SUSE
- Ubuntu

The two popular cloud platforms offering IaaS are Amazon Web Service, and Microsoft Azure. These platforms, as discussed, offer much more than IaaS; they have offerings also as PaaS and SaaS.

Microsoft Azure Portal

Microsoft Azure Portal is an enterprise cloud platform offering, with a collection of versatile products, such as follows:

- **Virtual machines**: These provide virtualization with Linux, Windows Server, SQL Server Oracle, and so on
- **App service**: To build web, mobile, and API for the enterprise grade
- **Cloud services**: Applications and services to be built and deployed on the Cloud
- **SQL database**: Relational database on the cloud
- **Azure Cosmos DB**: Globally distributed database across multiple regions
- **Azure Active Directory**: Identity and access management for the cloud
- **Storage**: Types are file, disk, table, queue, and blob
- **Backup**: Cloud-integrated backup as a service

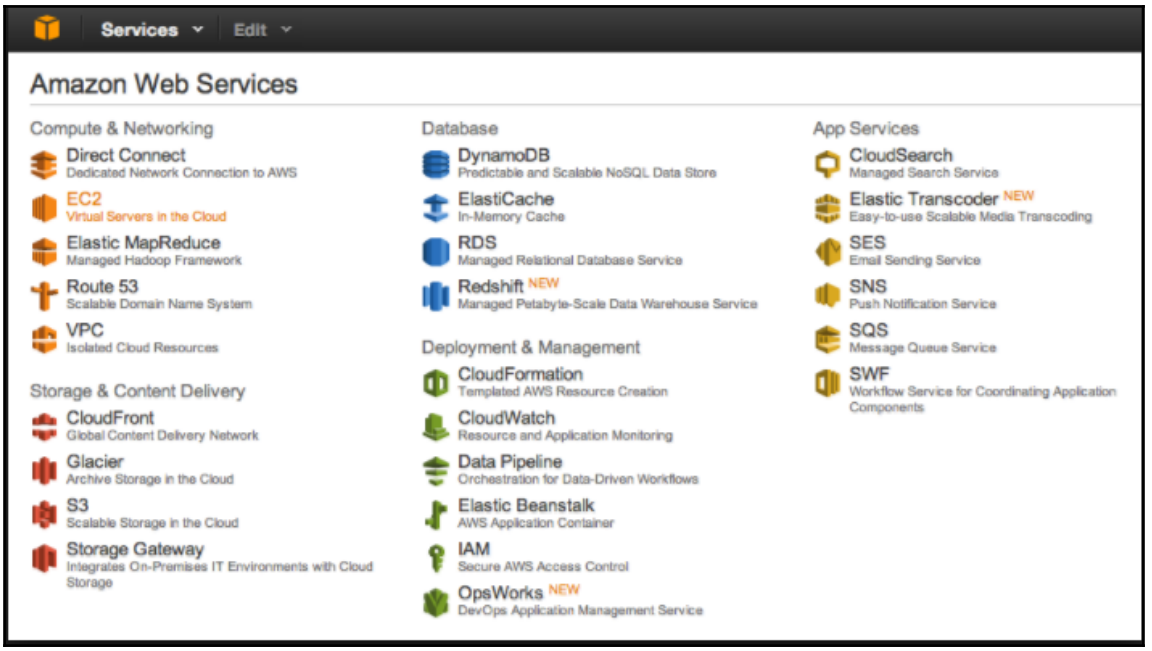


Amazon Web Services

Amazon cloud-based services offer many diverse offerings, including the following high-level features, which are integrated and accessible from its portal:

- **Compute services:**
  - **Amazon EC2**: Virtual servers in the cloud
  - **AWS Elastic Beanstalk**: Web apps hosting
  - **AWS EC2 Container Registry/Services**: Store/retrieve/run/manage Docker image
  - **AWS Lambda**: Events-based code execution
  - **Auto scaling**: Automatic elasticity
  - **AWS batch**: Scalable to run batch jobs
- **Storage services:**
  - **Amazon S3**: Scalable storage in the cloud
  - **Amazon Glacier**: Storage for archive in the cloud
  - **Amazon EBS**: Block storage for EC2
  - **AWS Storage Gateway**: Hybrid storage integration
  - **Amazon Elastic File System**: Managed file storage for EC2
- **Database features:**
  - **RDS**: Relational Database Service, MSSQL, Oracle, PostgreSQL, SQL Server
  - **DynamoDB**: Managed NOSQL database
  - **Aurora**: High-performance relational database
  - **ElasticCache**: In-memory caching system
  - **RedShift**: Effective data warehousing.

- **Migration functionality:**
    - Application discovery service
    - Server migration service
    - Snowmobile for data transport
    - Database migration service
    - AWS Migration Hub



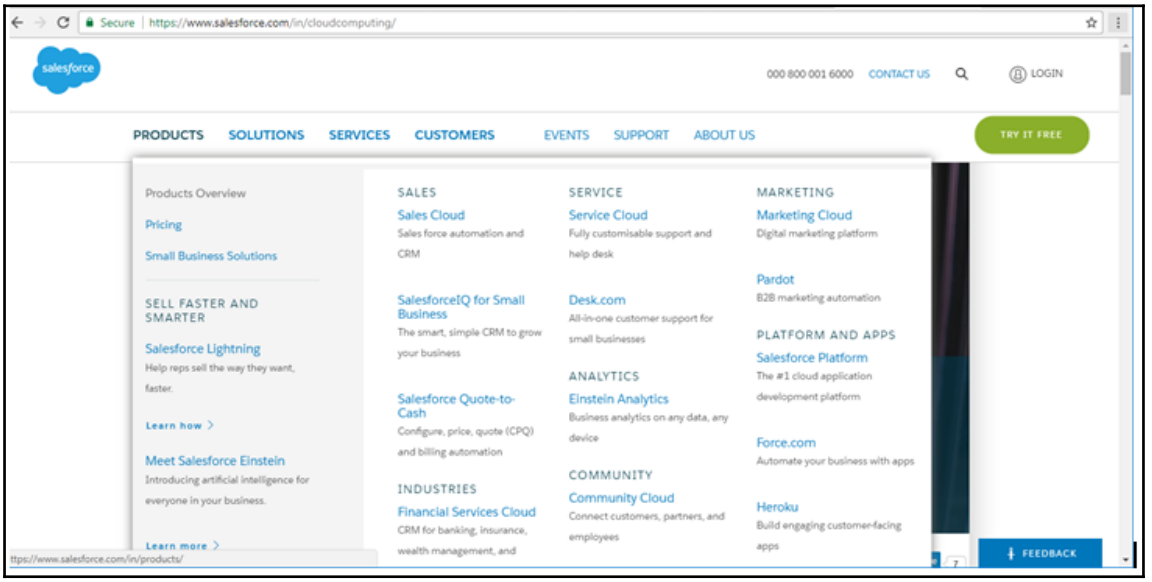
Salesforce offerings on cloud

Salesforce has multiple service offerings on par with other cloud vendors.

Sales channel:

- **Sales cloud**: CRM solutions provided on the cloud.
- **SalesforceIQ**: CRM solutions supported with email intelligence.

- **Salesforce quote-to-cash**: The cycle from accurate quotes, proposals, and contracts. Follows automated billing, subscriptions, and revenue recognition to the complete cycle.
- **Service cloud**: Cloud-based CRM, customer relationship management.
- **Marketing cloud**: Cloud-based marketing activities of email, mobile, social, advertising, the web, and so on.
- **Community cloud**: For collaboration, participation, and interaction between community groups for co-development.
- **Einstein analytics**: Analytics on the cloud, with drilldown options.
- **App cloud**: Cloud-based platform for apps development.
- **IoT cloud**: Cloud-based opportunity for the Internet of Things.



Network as a Service (NaaS)

In line with other offerings such as IaaS and PaaS, NaaS is also made available as a value-added service and network as a utility based on the pay-per-use model. NaaS uses a virtualized network infrastructure to provide network services; it enables access to the network infrastructure directly and securely. Customers can deploy custom routing protocols; however, the NaaS provider is responsible for maintaining network resources to ensure workload is supported.

NaaS offers multiple benefits, including being part of portals such as AWS and Azure. Customer networks are independent and segregated logically, as is traffic. Network bandwidth capacity can be accommodated, and customers only pay for usage. There is redundancy and resilience built in for backup. NaaS is convenient for the addition of new service elements. It offers a high-sensitive data protection specialized solution.

Identity as a service (IDaaS)

Identity as a service (**IDaaS**) is an important service and key feature that is the backbone for other service offerings such as IaaS, PaaS, and SaaS. IDaaS enables a set of identity and access management functionalities, to facilitate customers with the service through systems setup either on the client premises, or in the cloud.

IDaaS functionality includes the following broad areas:

- **Identity governance and administration**: This is the ability to secure digital information assets of the organization and adapt to governance policies
- **Identity access management**: This provides access to users for things such as authentication, **Single Sign-on** (**SSO**), and authorization as per role enforcement
- **Intelligence metrics**: Logging events, audit and compliance, and reporting details on access such as what was accessed, by whom, and when?

IDaaS solutions are flexible to store the organization's identity directory either on-premises, or in the cloud. On-premise Active Directory, or LDAP, is required in organizations for higher control and safety, which are important. They are supported with failover and automatic load balancing for continued service. Cloud-based IDaaS is also quite popular, and includes Google Directory, and so on. LDAP or Active Directory, are integrated through firewalls as per business needs. Managing identities requires a fine balance of security and productivity, with the optimal plan meeting the criteria. Audit compliance is an important feature to enforce policies, track cloud activities, and detect anomalies, by controlling access privileges to prevent unauthorized data loss through applications or environments.

It is critical that IDaaS has the ability to enforce granular policies to provide secure access and control of cloud apps to prevent data loss or theft. The business needs to acquire and deploy sanctioned and unsanctioned cloud applications in order to address critical organization needs. A strict policy discipline for data access is therefore imperative and mandatory.

IDaaS offers management of identity (information) as a digital entity. This identity can be used during electronic transactions.

A unique identification attribute is assigned to uniquely identify a digital asset or an object as a digital entity. All objects may have the same attributes, but their identity cannot be the same, so a set of attributes associated with them are identified to make them uniquely recognizable. These identity services are in demand for deployment to validate services such as websites, transactions, transaction participants, clients, and so on.

We have already seen the three broad areas of IDaaS. The following shows the finer segregation of the services:

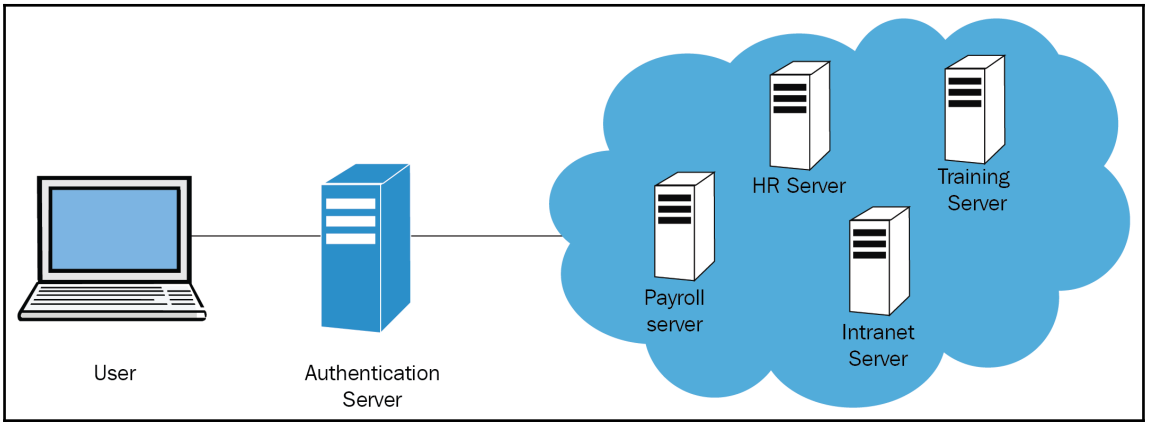
- Federated services
- Single sign-on services
- Registration
- Authentication services
- Risk and event monitoring
- Identity and profile management
- Directory services

We will look at the identity solutions next.

Single Sign-On

SSO is the most popular technology adopted across industries and companies to alleviate the need to use multiple username and password combination credentials for different servers. Single Sign-On software enables the user to login only once with credentials, and provides the user access to all the systems.

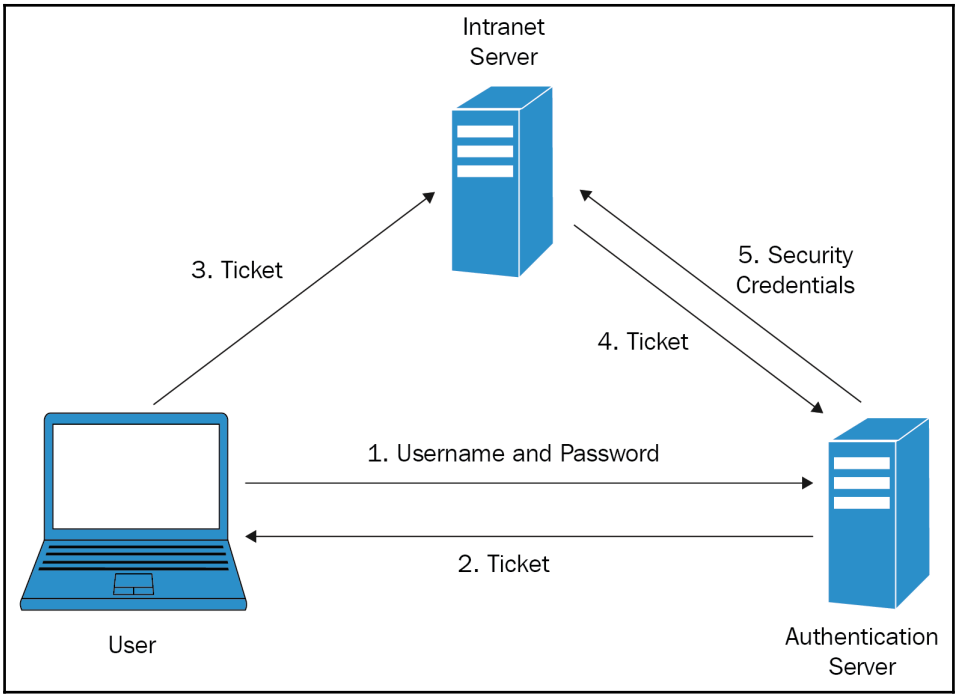
The SSO landscape has a single authentication server and, as shown in the following diagram, manages access to multiple systems:



The SSO modus operandi is as follows:

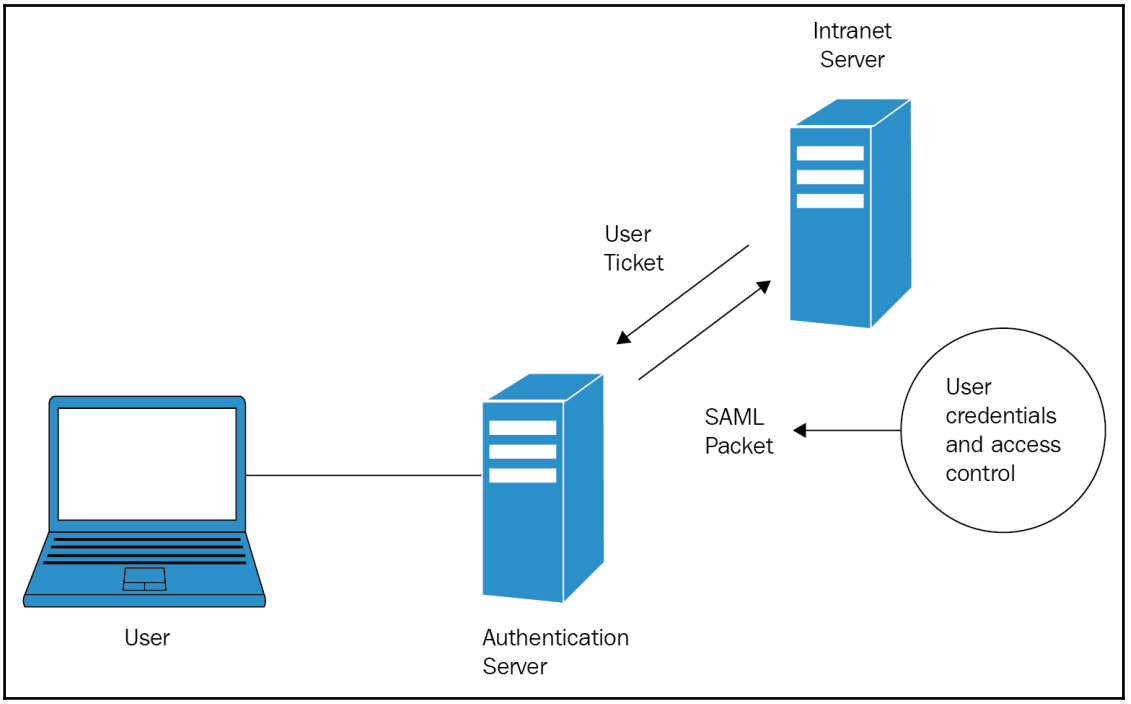
- With the credentials of username and password, log into the authentication server
- The user browser gets a ticket from the authentication server
- The ticket is sent to the intranet server by the user's browser
- The ticket is sent to the authentication server by the intranet server
- The user security credentials are sent by the authentication to the intranet server

Maintenance is easy for the addition or deletion of access with just the authentication server, which in turn reflects the user's access to all the supporting systems:



Federated Identity Management (FIDM)

As the name indicates, the security credentials are federated across security domains supported with technologies and protocols. **Security Assertion Markup Language** (**SAML**) is a prime mover to package a user's security credentials, as shown in the following diagram:



OpenID

OpenID is a service that facilitates users to login to multiple websites with a single account. Companies supporting OpenID includes Google, Yahoo, Flickr, MySpace, WordPress, and so on.

Some of the associated benefits include the following:

- Increased site conversation rates
- Greater access to cumulative user profiles
- Lower maintenance issues for users to avoid multiple credentials
- Ease of content integration into social networking sites

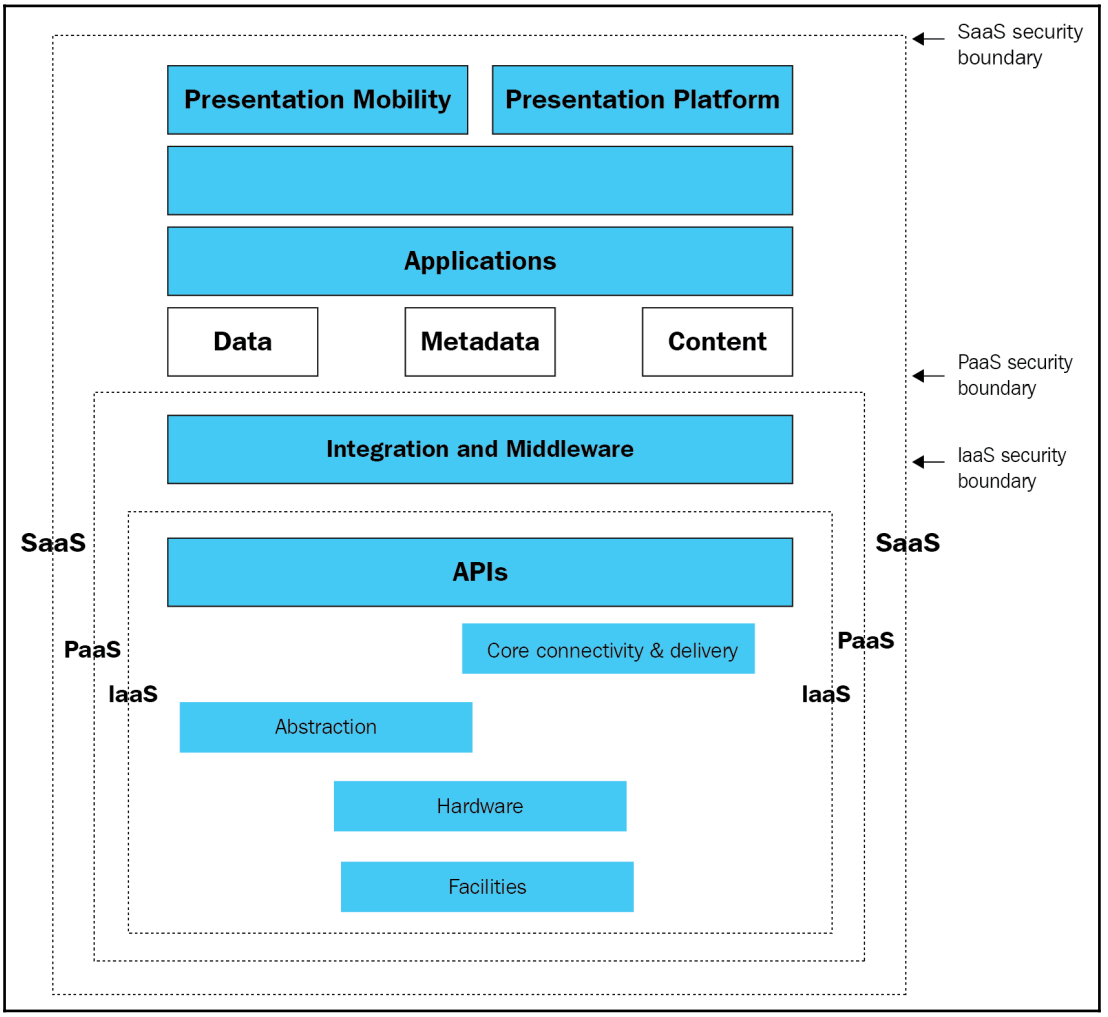
Cloud security

Security is major focus and concerning area while referring to cloud computing. There are multiple means of handling it as per the type of cloud, be it public, private, communal, or hybrid. It begins with effective planning to analyze several attributes on the sensitivity to risk in the context of the various models--IaaS, PaaS, and SaaS. With each of these models, the responsibilities of the client versus the cloud provider for security at different levels of the service need to be thoroughly understood.

Some pointers to gather relevant inputs in this light are the following:

- How the application or resource is deployed on the cloud
- Data encrypted form for storage in the cloud
- Proxy and brokerage services should be employed
- The need to restrict direct access to the shared data
- The cloud provider's system and tools for data transfer into and out of the cloud

Each cloud deployment model, depending on the service models and cloud types, brings in its own risks. The responsibilities between the cloud service provider and consumer are defined based on the offerings and business requirements:



Source: Cloud Security Alliance (CSA) (https://cloudsecurityalliance.org/csaguide.pdf)

As represented in the preceding diagram, IaaS is the basic level of service offering, which is followed by PaaS, and then SaaS. The security features and concerns are inherited from one layer to another, compounded in the order of from IaaS (infrastructure) to PaaS, and to SaaS. Thus, in the integrated system, IaaS has the lowest level of integrated functionalities and security, while SaaS has the highest level of offerings. The responsibilities are shared between vendor and client as follows:

- Depending on the service boundary
- As agreed upon in the business terms
- Depending on the cloud model adopted (public, private, hybrid, and so on)

Cloud data security is of vital importance, irrespective of the model and service selected. Data security enforcement includes the following options:

- Access control
- Auditing
- Authentication
- Authorization

Data stored in the cloud can be accessed from any location; therefore, an adequate data protection mechanism to isolate the data directly from client access needs to be incorporated. One of the effective approaches for isolating storage in the cloud is **Brokered Cloud Storage Access**.

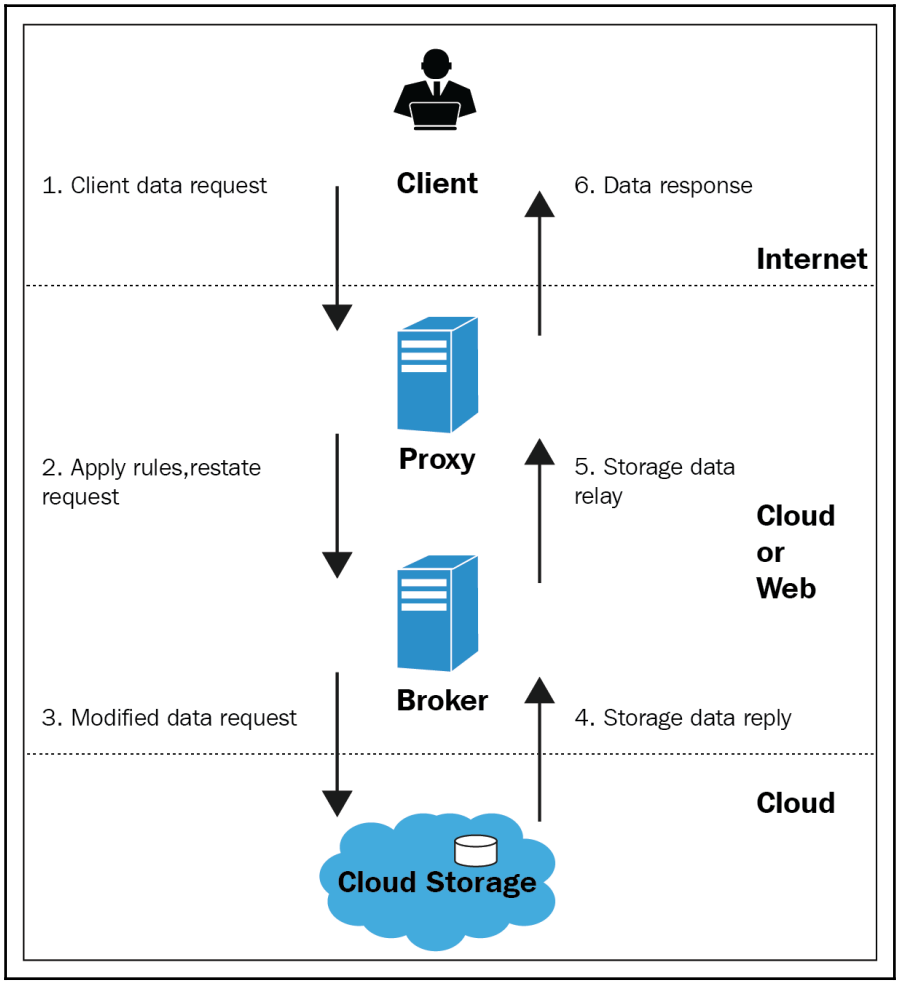
In this method, two services are created:

- A broker with full access to storage, but no access to the client
- A proxy with no access to storage, but with access to both the client and broker

When the client tries to access the data in the Brokered Cloud Storage system, it follows the following workflow:

1. The client data request hits the proxy's external service interface.
2. The proxy forwards the request to broker.
3. The broker makes data request to the cloud storage system.
4. The cloud storage system returns the data to the broker.
5. The broker returns data to the proxy.
6. The proxy sends data to the client.

These are depicted in the following diagram:



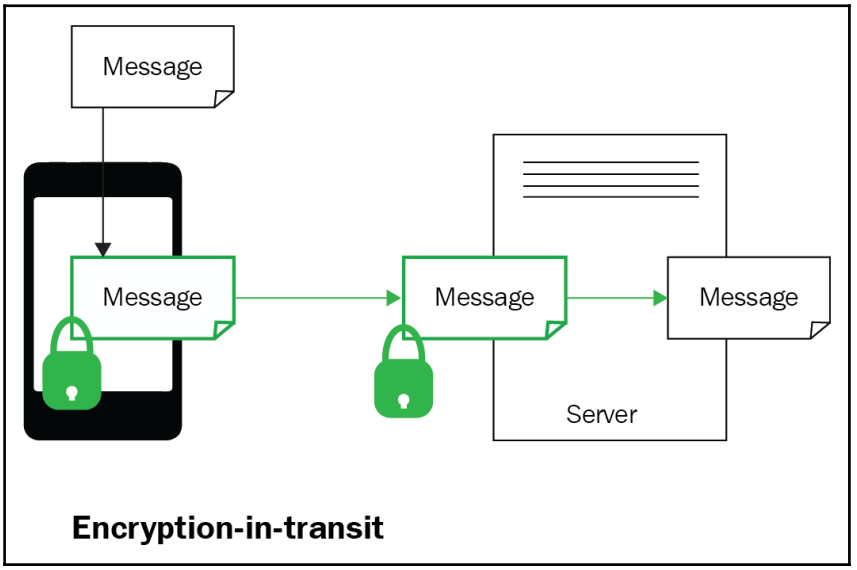
Source: Cloud Security Alliance (CSA) (https://cloudsecurityalliance.org/csaguide.pdf)

Encryption safeguards the data from unauthorized access. It prevents data being compromised for the data in motion (in transition, or being transferred), and also data at rest (static, stored in the cloud). However, encryption cannot avoid data loss like disk failure, and so on.

Data encryption

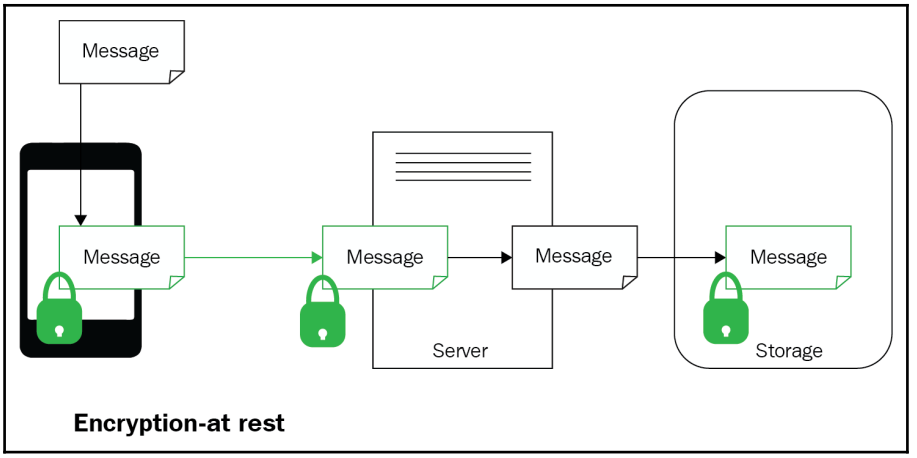
Cloud services are centralized repositories of often sensitive information, so the security aspects should be strengthened more vigorously than a single user laptop or mobile. The cyber security industry has been innovating and investing in technologies and processes to protect cloud servers such as firewalls, threat detection and analysis, and encryption mechanisms. There are different modes of data encryption, as shown in the following topics.

Encryption in transit



Using technologies such as **Secure Sockets Layer** (**SSL**), or **Transport Layer Security** (**TLS**), the encryption in transit is used to secure a message being transmitted from a phone or computer to the server. This will be secure while the message is on internet, and the decrypted message is available to both the device and the server. This makes the server vulnerable to attack, as the data is in unencrypted format, and messages are available in decrypted format:

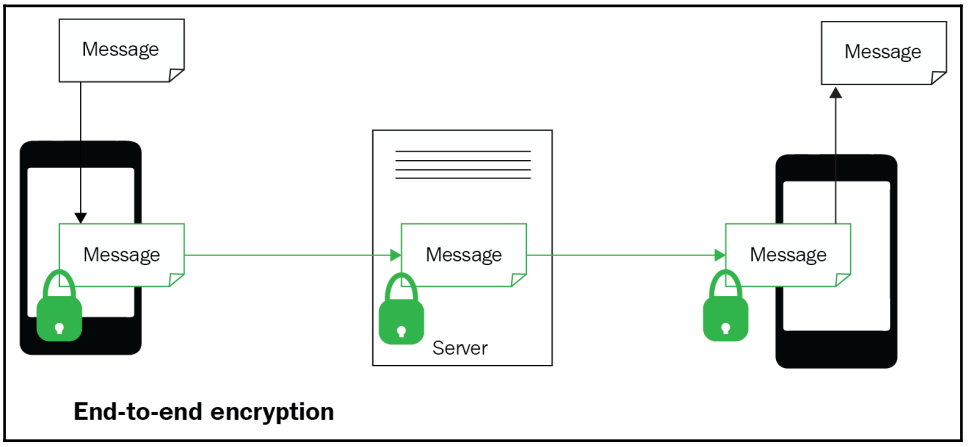
Encryption-at-rest



Encryption-at-rest means that data is encrypted when not being used, like in the storage media on cloud servers. However, since the server has decrypted information, it's vulnerable, and not completely safe.

End-to-end encryption

This will provide the complete solution by securing the data, not just the devices:



In the end-to-end encryption process, the messages are encrypted right from the device of the sender, till they reach the device of the recipient to be decrypted. So, with end-to-end encryption, the server never has access to the decrypted data, and will not compromise any user information. So here, the security is on the data itself, and not limited to the devices.

Additional data security measures are implanted by several features, such as the following:

- Secure processor acting as a microcontroller
- Secure encrypted virtualization
- Secure memory encryption
- Secure off-chip storage for firmware and data
- Cryptographic functionality for secure key generation and key management
- Hardware validated boot (TPM)

DRAM-level encryption is achieved with the AES-128 engine directly attached to the MMU to safeguard against physical memory attacks.

The OS, or hypervisor, can select the pages to encrypt via page tables.

However, DMA engine- encrypted pages are accessed by external devices such as network storage, and graphics cards.

With its own encryption key, each VM, or container, works in isolation in protecting against cross-contamination. The keys are transparent to the VMs themselves, and managed by the protected hypervisor.

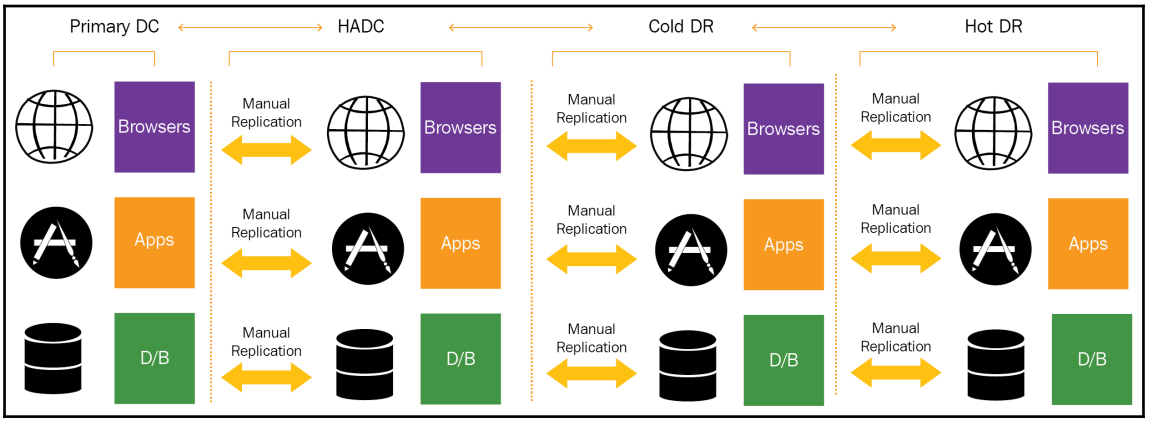
Data contamination is handled with reporting, and a machine check recovery mode.

Backup and recovery

Best practices for cloud computing backup and recovery follows:

- Four copies of the data are maintained for zero data loss and easy disaster recovery
- Two copies of the data would be kept in the primary data center with synchronous replication and for high availability
- The third copy of the data will be kept in another data center as a backup

- The fourth will be kept in a backup in a different data center and seismic zone using asynchronous replication:



Summary

In the next chapter, we will learn on big data applications core architecture principles, step by step building of systems with Spark platform. We will also study many data science algorithms to address business challenges.

6
Building Big Data Applications

In this chapter, we will learn to build big data applications, analyze a traditional end-to end data workflow life cycle, and on similar lines build a big data application step by step. We will cover the big data process--discovery, ingestion, visualization, and governance. The emphasis will be on the Spark platform and data science prediction models. DevOps applications to various phases of big data will be explored in the subsequent chapters.

- Traditional data platforms
- Big data platform core principles
- Big data life cycle:
    - Data discovery
    - Data quality
    - Data ingestion
    - Data analytics
    - Spark platform
    - Data visualization
    - Data governance
- Building enterprise applications
- Data science--prediction models

Traditional enterprise architecture

Traditionally, an **enterprise data warehouse** (**EDW**) system is considered as a core component of the business intelligence environment. Data warehouse systems are central repositories built by integrating data from multiple disparate source systems, used for data analysis and reporting the needs of the enterprise.

Let's review the end-to end data life cycle components of the traditional system:

- The **data discovery** phase is where the source systems are explored and analyzed for relevant data and data structures. If the analyzed data is valid, correct, and usable, it is ingested into the data warehouse system. For example, if we need customer ID information, we should be connecting and extracting data from the correct columns and tables.
- **Data quality** ensures that the ingested data is acceptable and usable. A simple example is name formats of the first name and last name convention, which should be adhered to and, as appropriate, corrected for a few records.
- **Data transformation** is the phase where data manipulation rules as per business logic are applied in tune with business needs. For example, every employee's yearly salary is computed from multiple systems and saved in the system.
- **Extract**, **Transform**, and **Load** (**ETL**) is the common term for consolidated reference for all the preceding phases together (data discovery, data quality, and data transformation) as a cycle.
- **Data staging** is the landing area on your systems for data collected from source systems.
- **Data lineage** traces the origin and credibility of the data ingested into the system to ensure only authentic, trusted, and authorized data is inducted into the system.
- **Metadata** is data about data. For example, a sales receipt is the origin of the record with transaction details, from where the requisite data from our computations is extracted. Details such as store ID, sales amount, item ID, date of transaction, and so on, are extracted from a sales receipt.
- **Data warehouse** is the storage layer, into which the transformed data is loaded as consolidated copy. It is time-variant, consistent, and read-intensive.
- **Data marts** are data serving repositories specializing in some category, such as customer data, product data, employee data, and so on.
- **Data analytics** is the common term for an analysis to address all the business needs. Its building queries address business demands, such as how many customers were added last month, or what products are selling above targets this week.
- **Semantic layer**--its business interface builds queries on the database with business intelligence tools, hiding complex data tables from business users.
- **Reporting**--reports are for business use, such as all products sold last month in a state.

- The **dashboard** provides a consolidated quick view of important key performance indicators. An analogy is a car dashboard with speed, battery, petrol reserve, and so on.
- **Data visualization**--finding key performance business trends solely based on Excel reports can be a daunting task. Presenting them in a visual form is quite appealing, such as representing them as charts, histograms, and pie diagrams.

Principles to build big data enterprise applications

Big data platforms and applications manage, integrate, analyze, and secure analytics on many data types both within the enterprise as well as in external data. They integrate multiple data sources in real time, taking into account volume, velocity, and variety. The platform can be built as a repository of an enterprise knowledge base with the organization's collective data assets.

Some of the salient features for building these platforms are discussed and as we see DevOps is very appropriate and instruments to enhance value at every stage, like versioning systems for building algorithms, data models, scalable reproducible platforms with virtual machines as seen in previous chapter:

- **Flexible data modeling**: Big data systems integrate many different forms of data from multiple data sources. Rather than a pre-defined schema of rigid rows and columns, the schema is to be defined on the fly and data modeled to reflect how information is to be assimilated. To reflect real-world entities, it is also flexibly specified as a graph of objects with relationships.
- **Knowledge management**: Version controlled knowledge base with accumulated insights of an organization can be leveraged as an enterprise asset.
- **Privacy and security controls**: The platform is designed for data lineage, multi-level security, and audit compliance. Every object integrated into the platform is traced to its original data source, where access restrictions are in place including authorization and authentication.
- **Algorithms for data processing**: These are massive datasets to be compiled and analyzed with built-in machine learning algorithms to augment the human user's ability to make sense of large-scale data by identifying patterns in the data.

- **Scalable platforms**: These platforms handle petabyte-scale data through a combination of a scalable architecture with federated data storage to hold large types of unstructured data, such as documents, emails, audio, video, images, and so on. These platforms are designed as open platforms extendable at every layer of the stack. The provision for efficient data discovery, data lineage, and elastic search tools should be considered.
- **Collaboration**: This platform enables multiple users, within and across organizations, to seamlessly, securely collaborate to analyze the same data concurrently, from low-level data integration, importing pipeline customizations, to building custom user interfaces. Data that has been integrated can be accessed as objects via APIs or can be exported for other frameworks and tools
- **Building models on models**: Simple models can serve as building blocks for more complex models, building out sophisticated analyses to be streamlined as a modular process. Models can be built using various in-built rich reusable libraries of statistical and mathematical operators.
- **Data visualization**: This is an interactive user interface to provide a seamless holistic view of all the integrated data of interest in the form of rich visualizations, such as tables, scatter plots, and charts. These visualizations in real-time are up to date with the source data so that users always see the most accurate and current information at any given time.

Big data systems life cycle

Big data systems are built in accordance with the data life cycle model, which can be broadly categorized in the following stages:

- Data discovery
- Data quality
- Ingesting data into the system
- Persisting the data in storage
- Analytics on the data
- Data governance
- Visualizing the results

We will study them in detail next.

Data discovery into the system

Data discovery, like in the traditional process, ingests raw data from multiple source systems; however, the data will be divergent in volume, variety, and velocity when it comes to transforming it into business insights. Leveraging the power of big data, the data discovery process enables data wrangling and data enrichment facilitates combining datasets to recreate new perspectives and interactive visual analytics. An interactive data catalog facilitates guided search capabilities and enables us to thoroughly analyze and understand the data quality. A matured and robust data discovery process ensures possible data correlations; it lets users define attribute-based rules, relationships between data sources, harmonize data sources, and create enriched data based on a data mart.

To expand the boundaries of traditional business intelligence systems, harnessing the potential of big data is the key for business success. It helps to unlock the insights from new sources of information effectively. Organizations have the potential to access a wealth of knowledge to be tapped appropriately, with the proliferation of new sources of digital information.

Data discovery with big data tools and technology facilitates deep exploration for visibility into business performance with a big variety of data across the organization and even beyond. The business can explore new dimensions, transforming the way that business analytic systems are built and used more efficiently. Data discovery with any combination of data sources allows rapid, intuitive exploration and analysis of information; it enables deeper insight into the business, and the opportunity for greater efficiency. It builds new relations redefining roles between business and IT, adding new roles, new leadership, and revised means of data governance as well.

Many organizations have updated business intelligence decision systems to improve business performance based on existing data and systems to understand and monitor. These days, the vast majority of data growth is from systems beyond the reach of traditional BI environments, such as websites, social media, content management systems, emails, documents, sensor data, external databases, and so on. Hence the need to adopt to new age data discovery tools, also to consider that this diverse and changing data is growing exponentially. Data types to be dealt by the discover phase are varied, ranging from structured database tables to semi-structured forms containing a mix of numbers and free-form text to wholly unstructured documents.

The biggest challenge along with the volume of data is the variety rather and its uncertain value. An internet-savvy business culture and the impact of consumer interaction with enterprise business software with mobile and web applications have created an urgent need to quickly explore relevant information to discover new insight for business prospects and decisions.

Datafication is the process of quantifying data from all types of sources, in all types of formats. Datafication allows information to be collected, tabulated, and analyzed, so that the potential uses of the information are limited only by the ingenuity of the skilled business user. The data's true value is like an iceberg floating in the ocean. Only a tiny part of it is visible at first sight, while much of it is hidden beneath the surface. Innovative companies that understand this can extract that hidden value and reap potentially huge benefits.

Enterprise-class data discovery systems enable rapid, intuitive exploration and analysis of data from a wide combination of structured and unstructured sources. They enable organizations to channel their existing investments to extend business analytics capabilities to new combinations of a greater variety of sources, including social media, websites, content systems, e-mail, and database text, providing a new level of visibility into data and business processes, saving time and cost and leading to better business decisions. Some of the advantages of this approach are:

- You can gain deeper insight into the business by enabling users with increased insight and visibility to find the data they want to analyze.
- Near real-time data and content can be delivered by access to fresher information, helping people make decisions based on the most current information.
- Increased reuse of assets. You are able to reuse information assets and eliminate the costs of re-creating these assets.
- Ease of use for BI professionals to develop and deliver analytic consumer-style applications for business professionals, leading to higher adoption rates, lower training costs, and faster time to value.

Data discovery stages

The data discovery process involves stages such as prototyping, visualization, bridging, replication, and transformation. These concepts applied in tandem in the context of data discovery provide value out of big data:

- **Prototyping**: Generally, in any business context, projects are always under pressure and constrained to deliver under-limited resources of time and budgets. For a complex project, prototyping is an effective way of making progress when challenges are pressing, demanding, seem difficult, and are under time constraints. Prototyping is about acting before you've got the answers, about taking chances in the absence of a proven formula or a known way of solving a problem. Prototyping lets us test a hypothesis and explore alternative approaches, building in small blocks to get the big picture. DevOps expedites the process of prototyping to make it a continuous cycle of development

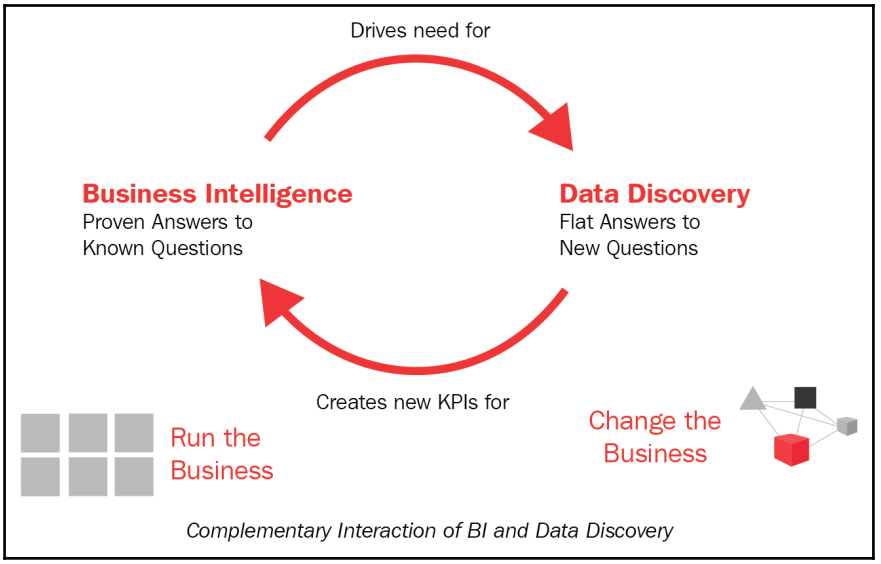
  With data discovery, people are empowered to wander around in the data, try new combinations of sources, filter and refine the analysis, find previously hidden patterns, or jump to another experimental thread if the first one yields no insight. One of the key challenges is synchronizing data schema changes at the source systems with staging and development systems in real time; the DevOps methodology and process can be adopted to address this.

  Experimentation through data discovery comprises three main tasks--asking new questions, seeing new patterns, and adding new data. These steps comprise a continuous process which is iterative and which flows in any direction, depending upon what the user sees or theorizes at any given moment. DevOps integrated with the data discovery phase makes it an automated process from source systems identification to ingestion of data to staging systems.

- **Visualization**: Data visualization helps data analysts gain immediate feedback on their hypotheses, as graphics are more convenient instruments for realizing patterns in quantitative information. Viewing the results of empirical business analysis in real time enables data analysts to determine what refined or further search could lead to a deeper understanding of a performance gap or market opportunity. Effective data discovery is augmented by quick visualization of analysis and results to aid both experienced data analysts and non-technical business users. Data visualization techniques such as drag and drop dashboards, quickly produced charts, easy to use wizards, and consumer-style navigation, accelerate understanding of patterns of data, and its behavior attributes quickly.

- **Bridging**: Business and IT can have a more effective working relationship through the data discovery process. In a traditional setup, business and IT perform separate activities as a provider and user in the data discovery phase. This enables them to work together as a team to combine efforts to jointly explore and learn. IT analysts help their business partners, showing them how to add data sources, refine searches, and explore new questions, stepping through the capabilities of the software. They can work with the business directly on how to build discovery applications in real time, and how to become self-sufficient in analysis of the possibilities. Conventional extracting requirements, writing complex specs, and going through a lengthy development and deployment process are tedious in terms of time and effort. IT gains huge efficiencies bypassing much of the SDLC process and by empowering business users via self-service data.
- **Systematizing**: As data exploration becomes more versed with deep insight skills, in some cases a certain path of investigation could be repeatable a few times. It could also yield valuable insights beyond a limited one-time scenario to more effectively grasp a part of the business question to continually repeat and track the outcome. Once the business queries and responses are well established by the metrics and dimensions that are used to describe key business processes, they can be considered to roll the analysis and metrics into the organization's enterprise BI system. Business intelligence platforms excel in allowing users to perform standardized queries for well-known questions on standard datasets.

Data discovery and business intelligence systems complement each other, with data discovery excelling in unknown questions and BI focusing on systematized analysis, as depicted in the following diagram:



Complementary Interaction of BI and Data Discovery

DevOps will standardize the process of data models, dashboards, and visualization reports can be maintained into the repository and automated testing and deployment from development systems to QA and production as continuous integration and deployment models.

- **Transformation**: The data discovery process can help explore an unlimited number of new avenues to address business problems and find new opportunities previously hidden in the data. Uncovering these new dimensions through a process of experimentation uncovers valuable new insights, paving the way for transformation. However, for standard known areas, they will continue using existing BI systems, and use data discovery to explore ways to give insights for new questions and problems. Thus data discovery is proving its value in deploying an easy exploration of diverse data to uncover insights that drive dramatic increases in revenue and productivity while improving the alignment and relationship of business and IT. It enables a transformation in the world of business analytics.

There are several tools in big data discovery, such as Apache PIG, Oracle Big Data Discovery, Zoomdata, Exasol, Revolution, GridGain, and so on.

Traditional BI tools such as Tableau, QlikView, Microstrategy, Informatica, and so on, are also offering extensive data discovery features for big data.

Data quality

One of the biggest challenges is ensuring data quality and accuracy, which is easier said than done. DevOps will augment the data quality process to ensure the data quality cycle from scripts to automation of the entire validation process. Let's look at some of the challenges:

- **Data variety**: The data coming from diverse data sources such as mobile devices, web technologies, sensor data, social media, and so on brings with it multiple complex data types and data structures, increasing the difficulty of data integration. These data sources produce data types such as unstructured data in the form of documents, video, audio, and so on, and semi-structured data like software package, modules, spreadsheets, financial reports, and structured data. The valuable insights gained depend on the data collected, stored, and verified. They come from so many divergent sources and rely on the effectiveness of the integration process. When working with data-intensive and sensitive industries such as life sciences, this process has to be foolproof.
- **Complexity of data**: The data becomes complex, accounting for multiple attributes such as raw data from a variety of different sources directly from consumers, salespeople, operations, and other sources within the organization. The timeliness of the data is crucial; you need to gather the required data in real time or deal with the data needs in real time, otherwise the data can become stale, outdated, and invalid. Processing analysis based on the data will produce useless or misleading information and conclusions, misleading the decision-making systems.
- **Ensuring data security**: There are so many technologies that contribute to multiple channels of communications across applications of mobile, web, and ERP, which adds to the complexity of maintaining data security. New age technologies such as cloud, big data, and mobile are expanding their reach very quickly and becoming popular. However, data security needs and challenges are more complex than before.

General guidelines to ensure data quality are the following:

- **Data availability**: The data intended for consumption to the big data systems should be appropriately available for the intended use. It should be timely data with either real-time streaming or batch mode. The data should be accessible with API interfaces available and also the data should be authorized for use.
- **Data appropriateness**: The data should be credible; otherwise, the purpose is defeated. The data lineage process traces the origin of the data. The data definitions should be appropriate and acceptable on freshness and regular updates for the data. The data documentation and metadata should be audited for correctness within an acceptable range of values.
- **Data accuracy**: The data should be reliable; the data value should represent the true state of the source information and the data should not be ambiguous and should be a single version of a fact. Data should be consistent and verifiable during the intended time as per the time stamp. The data should be complete and auditable.
- **Data integrity**: The data format should be clear and meet the set criteria of consistency with the structure and content. Its integrity should be intact. The data should be relevant and match the required purpose, and be complete in all aspects and fit to use for its intended purpose.
- **Presentation quality**: The data content and format for the data should be clear and understandable. The data description, classification, and coding content should be easy to understand and meet the stated objectives and specifications.

There are many big data open tools providing multiple features for data quality, such as Talend, AB Initio, Data Manager, Datamartist, and iManage Data.

In the data ingestion process, raw data is added to the system from multiple data sources. The process can be complex, depending on the format and quality of the data from the source systems and the state of the data to be processed from the desired target state for consumption. There are multiple ways and means of ingesting data into big data systems, depending on the type of big data ingested. To prepare raw data for the system's use, some level of analysis, sorting, and labeling usually takes place during the ingestion process. Extending conventionally from the legacy data warehousing processes consists of extracting, transforming, and loading, referred to as the ETL process. Some of the same concepts apply to data entering the big data system as well.

The process of data ingestion involves massaging the input data for proper formatting, categorizing, and labeling. The data structure should adhere to predefined standards by eliminating unwanted data. The data thus captured from multiple source systems in large volumes is used for further processing. It is stored in a data lake in raw format

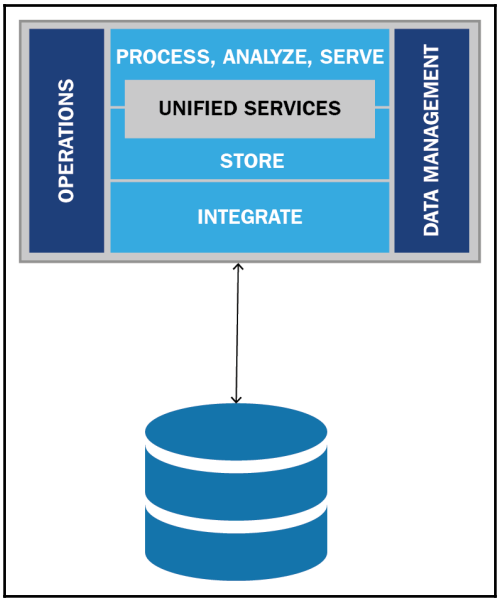
Batch processing

Batch jobs, as the name indicates, are dataset jobs which are non-time sensitive and processed in batches in large datasets. The processing of batch jobs can happen in multiple modes, as described next.

RDBMS to NoSQL

Most of the legacy data stored in RDBMS can be imported into NoSQL databases on HDFS using Sqoop, DevOps can aid for baseline of the Sqoop scripts, automating the process of importing and exporting the data, scalability of the storage and computing systems, test automation of the data integrity, and automated deployment.. The data migration process is described here:

- Sqoop is a command-line interface application for transferring data between relational databases and Hadoop. It supports incremental loads of a single table or a free form SQL query as well as saved jobs which can be run multiple times to import updates made to a database since the last import. Imports can also be used to populate tables in Hive or HBase.
- Oozie can be used to schedule and create flows for importing/exporting data.
- Full as well as incremental imports can be configured in Sqoop/Oozie
- We can directly import and create Hive tables, but if we build a layered architecture it is suggested to import to the staging.
- For example--`sqoop import –connect jdbc:mysql://:/ –username –password --table --target-dir`.

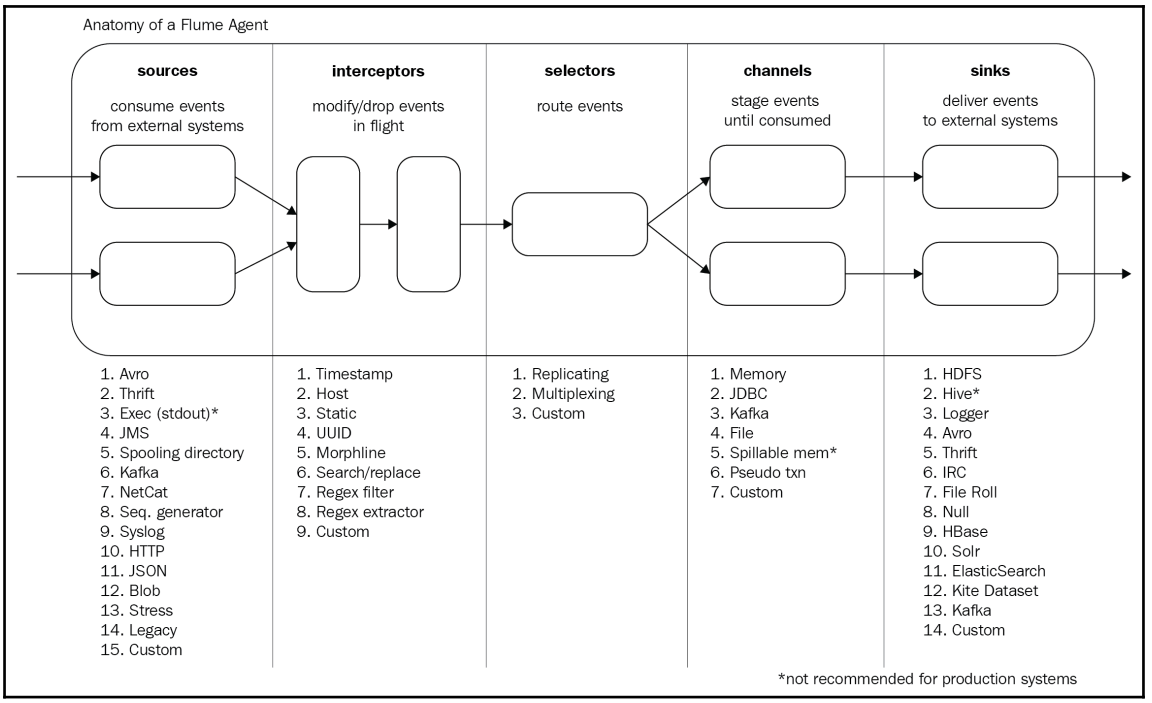


Flume

For other batch sources, we can deploy Flume. It will watch for new source files and push the data to HDFS.

- Flume is effective and reliable for moving log data in large volumes to a distributed system, collecting and aggregating it in accordance with business demand
- The architecture is simple and flexible, matching the incoming streaming data flow needs
- Flume is robust, fault-tolerant, customizable, with advanced features such as failover and a recovery mechanism, and so on

- A simple extensible data model supports online analytic applications



Stream processing

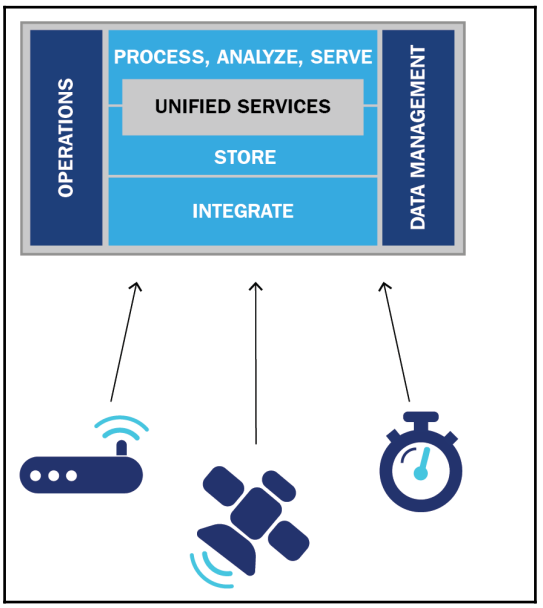
Stream processing, as the name indicates, is the computing of real-time data which is time-sensitive, usually with high-velocity metrics. Analytics are usually performed on the streaming data while it is being ingested for quality checks, and so on.

Real-time

If data is available in streams such as application logs, program output (like web scrapper), sensors, geo-location, or social media, this can be collected using Kafka on a real-time basis.

- Kafka is an open-source message broker project that aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. It is, in essence, *a massively scalable pub/sub message queue architected as a distributed transaction log, making it highly valuable for enterprise infrastructures to process streaming data*.

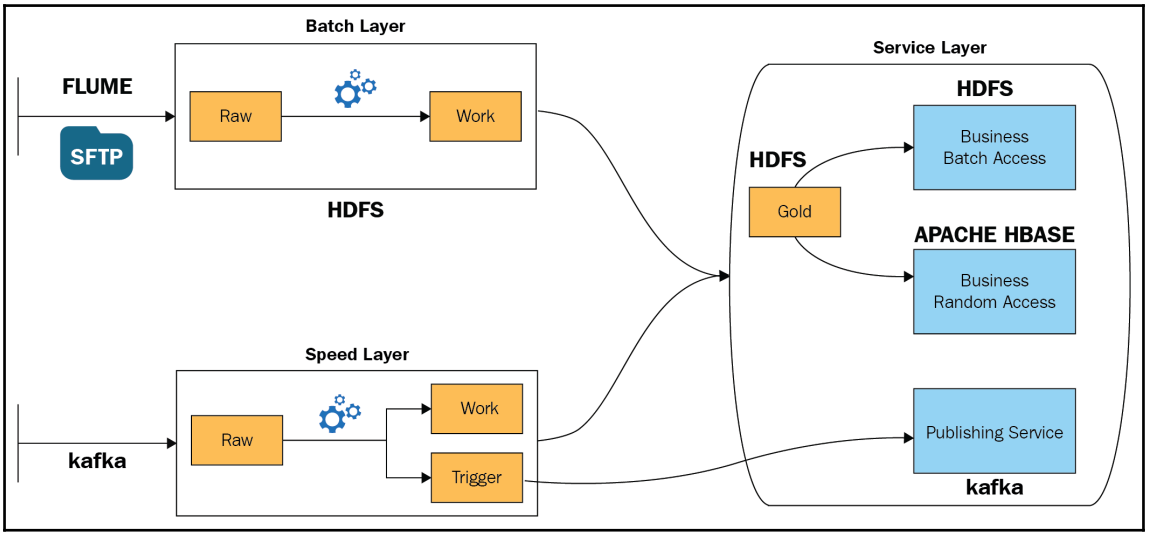
- Kafka can be easily scaled to support more data sources and growing data volumes.
- Kafka also supports direct connectivity to Spark.
- There is one topic per incoming data source and one topic per consumer group.
- The number of partitions per topic will depend on the data size.



Apart from Sqoop and Kafka, Some other specialized data ingestion tools for importing and aggregating both server and application logs are Apache Flume and Apache Chukwa. **Gobblin** also offers a data ingestion framework to aggregate and normalize date.

Lambda architecture

The lambda architecture is effective when both batch and streaming data are ingested to systems at the same time, as shown in the following diagram:



The data storage layer

Persisting big data has multiple storage options, such as Data Lakes and data warehouse; cloud technologies for data storage greatly augment the needs of big data storage systems, scalable to terabytes and petabytes through simple storage devices and virtual machines. DevOps is an effective solution for managing scalable data storage with infrastructure as code discussed in detail in this book:

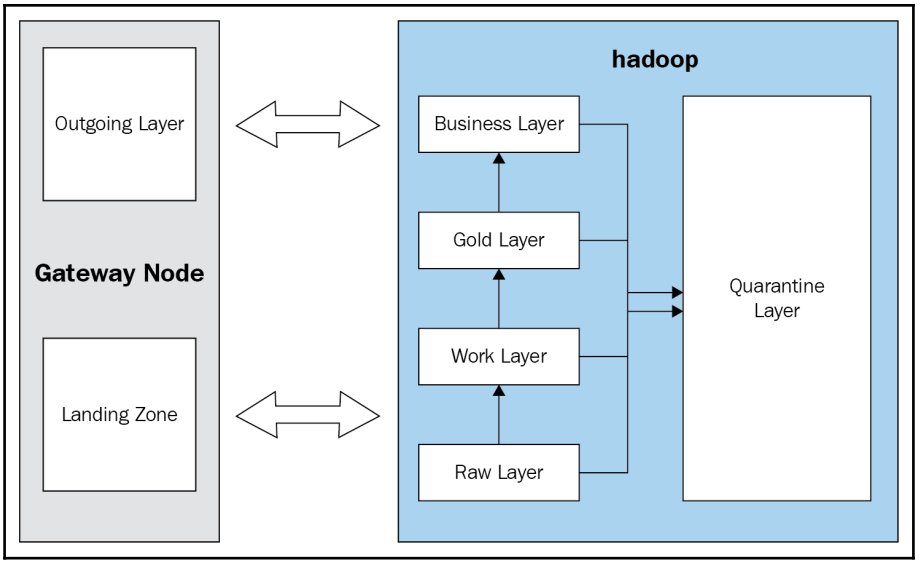
- **Data Lake**: Data Lake is synonymous with a water lake where water is stored and consumed by many people. A Data Lake is the repository for the collection of raw data. The data collected could be unstructured and frequently changing, so, initially the data is pooled into the large repository to be consumed by users as per their scheduled needs.
- **Data warehouse**: A data warehouse is an ordered repository for large volumes of data to be used for analysis and reporting. The data in a data warehouse is typically being cleaned, is well-ordered, and is integrated with other sources. It is generally more prominent with conventional systems.

The ingestion process ensures the incoming data is processed as per business needs to be persisted reliably in the storage disk. The ingestion process can be complex, depending on the volume and variety of the data from the source systems. The availability of the distributed storage system is accomplished by Apache Hadoop's HDFS filesystem. With HDFS, large quantities of raw data are written to multiple nodes simultaneously with redundancy.

Ceph and GlusterFS are other filesystems offering all these capabilities.

Distributed databases such as NoSQL are well placed to import data for more structured access. They have features such as fault tolerance, and they have the capability to ingest heterogeneous data formats. Based on the organization's business needs, appropriate databases can be selected from a variety of available choices.

Data storage - best practices for better organization and effectiveness



Landing

A landing zone is where in the initial data lands from different source systems in the as it is state to the storage system.

- The landing zone is where incoming data is stored
- All input validation should be done here
- The folder structure can be `<source>/<type of data>/<yyyymmddhhisss>`
- The archive mechanism should be applied as well (day/week/month)
- Access should be restricted to only processing users and not end users

Raw

Once the data lands in the landing zone, it undergoes sanity checks as to its appropriateness, format, and quality; upon satisfactory compliance, it is stored as raw data.

- This is where the raw data is stored in its original format
- Validated input from the landing layer is stored here
- The directory structure is managed by the ingestion framework
- Only selected super-users and system users will have access to this data
- Snappy/LZO compression should be applied
- A data classifier should be applied (hot, cold) and set to the archive policy
- The folder structure can be `<base dir>/<system type>/<dataset Source Name>/<Source Type>`

Work

The work area stores identified clean data to be used for business purposes.

- This is the temporary working area and cleanup up should take place after related jobs unless the jobs require data for debugging
- The checkpoint location for Spark streaming will be located here as well

Gold

Gold data is the valuable data that is the transformed, partitioned, and classified as master data.

- This is the location that stores transformed data
- Partitioning is important in this layer and should be done based on the most frequently accessed columns
- Classification should be done for very hot, hot, cold, and very cold data
- Very cold data should be archived to blob storage (or any other cheap storage)
- The folder structure could be `<base dir>/<system type>/<dataset Source Name>/<Source Type>/<Job ID>`

Quarantine

This is the place where unwanted and stale data, which is of no active immediate use, is saved.

- All rejected files from the various steps will be stored here, such as ingestion, transformation, as well as validation exception records
- Classification should be done for very hot, hot, cold, and very cold data
    - Very cold data should be archived to blob storage (or any other cheap storage)

Business

Business data is the master data for a client's particular details, such as address, product preference, and so on.

- This layer will have application/client-specific data
- All data must be store in parquet format since all data at this stage will be structured

Outgoing

Outgoing data is what is to be shared with external entities, such as suppliers, vendors, or third-party APIs.

- This is the location from where data can be shared with the external application
- Files need to be archived once they are copied
- Clients can have temporary or permanent access

A good easy-to-use backup and disaster recovery solution provides integrated data sync between Hadoop clusters. It enables data protection by replicating data stored in HDFS platforms and across data centers.

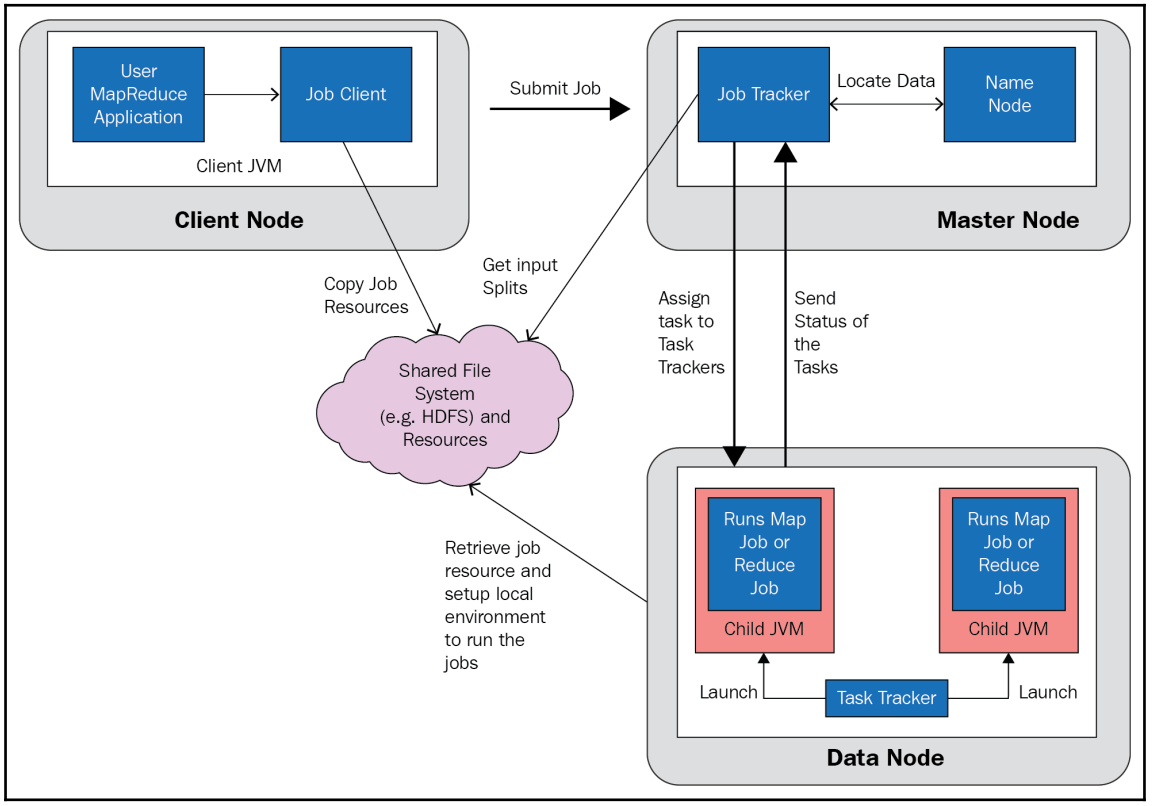
Computing and analyzing data

The data made available from the preceding processes can be analyzed to unearth the actual information value. The computational layer performs diverse functions where data is often processed iteratively with a combination of tools. The different types of insight needed for business needs are extracted by tailored practices that vary from organization to organization.

Batch processing is one method of computing over a large dataset where data is ingested in batch mode either daily or hourly. Apache Hadoop's MapReduce is the most prominent and powerful batch processing engine and it is known as a distributed Map Reduce algorithm; it adopts the following strategy and is most useful when dealing with very large datasets that require quite a bit of computation:

- **Splitting**: In this process, the work is divided into smaller pieces
- **Mapping**: This is the process of scheduling each piece on an individual machine
- **Shuffling**: Reshuffling data based on the intermediate results
- **Reducing**: Processing each group of output data
- **Assembling**: The final result is assembled together

The MapReduce framework forms the compute node while the HDFS filesystem forms the data node. Typically, in the Hadoop ecosystem architecture, both the data node and compute node perform similar roles. The delegation tasks of the MapReduce component are performed by two daemons, the **Job Tracker** and **Task Tracker**, Their activities are shown in the following diagram:



Data processing in batch, real-time, and stream processing is discussed next. DevOps is integral to big data systems for high volume data processing from source system discovery to the scalability of infrastructure to support storage needs.

- **Batch processing**: It is very efficient in processing high volume data.The data is ingested to the system, processed, and then results are produced in batches. The computational power of the system is designed based on the size of the data being processed. The systems are configured to run automatically without manual intervention. The system can scale very quickly to accommodate the entire dataset for computational analyses of the huge volume of data files. Based on the volume of data processed and the computational power of the system defined, the output timelines can vary significantly.
- **Real-time/stream processing**: Though batch processing is a good choice for certain types of data and computation, other workloads require a more low-latency turnaround. There are real-time systems which are required to respond in real-time as they process information and make the analytics or visualizations readily available to the business while assimilating the new information continuously. These systems are called stream processing, which operates on a continuous stream of data composed of individual items. These systems, real-time or stream processing systems, utilize the real-time processing capability of in-memory engines; computational analytics are performed in the cluster's memory to avoid having to write back to disk as in traditional disk-based persistent systems. Real-time processing is best suited for analyzing smaller chunks of data that are changing or being added to the system rapidly.

There are many platforms and tools to achieve real-time processing, such as Apache Storm, Apache Flink, and Apache Spark. Each of them is designed as different ways of achieving real-time or near real-time processing. Apart from these listed computational frameworks, there are many other means of analyzing data or performing computations within a big data ecosystem. These tools frequently plug into the aforementioned frameworks and provide additional interfaces for interacting with the underlying layers. We have already discussed in the previous chapter their applicability to different scenarios and the best application for any individual problem, but we will recap a few of the tools here again:

- Apache Hive provides a data warehouse interface for Hadoop
- Apache Pig provides a high-level querying interface
- Apache Drill, Apache Impala, Apache Spark SQL, Presto, and so on, provide SQL-like interactions with data

- R and Python are popular choices for simple analytics programming
- Apache SystemML, Apache Mahout, and Apache Spark's MLlib, provide building prediction models for machine learning libraries

Apache Spark analytic platform

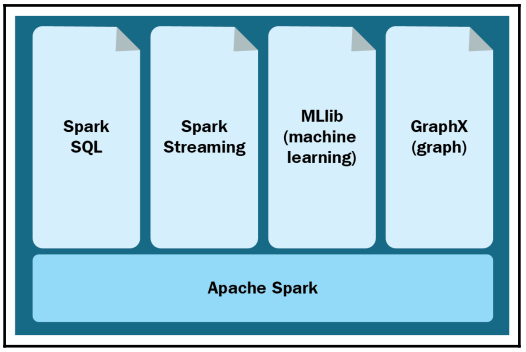
Apache Spark is a next-generation in-memory open-source platform, combining batch, streaming, and interactive analytics under one umbrella. Spark facilitates ease of use, providing the ability to quickly write applications with built-in operators and APIs along with faster performance and implementation.

Spark provides a faster and more general data processing platform and runs programs up to 100x faster in memory, or 10x faster on disk, than Hadoop, with lightning-fast cluster computing. The Spark framework is built on top of Hadoop clusters to process data from structured system such as Hive and stream data from Flume and Kafka. It has many advanced features and supports a variety of languages, including Java, Python, and Scala. It has extensive features for analytics, out-of-the-box algorithms, machine learning, interactive queries, and complex function analytics.

Spark's popularity is due to many advantages, including:

- Spark is a subset of the Hadoop ecosystem. It integrates well with the reliable and secure storage platform, HDFS of the ecosystem. It is compatible with other data sources, such as Amazon S3, Hive, HBase, and Cassandra. It can run on clusters managed by Hadoop YARN or Apache Mesos, and can also run as a standalone.
- Spark is a fast technology to enable large-scale data processing. This framework provides Java-, Scala-, and Python-based high-level APIs with a rich set of data stores for stream processing and machine learning. Though primary APIs are for Scala, Java, and Python, languages such as R are also supported.
- Spark ensures parallel processing for data, very well integrated with Hadoop/HDFS for data storage, and to support variety of filesystems and databases.

- Spark's machine learning capabilities are proved to be excellent solution for stream processing. With Spark REPL writing code is easier and quicker with inbuilt high-level (80) operators . Spark's **Read Evaluate Print Loop** (**REPL**) is interactive (out-of-the box) shell is a modified version of the interactive scala REPL. With REPL, no need to compile and execute the code. User expressions are evaluated and REPL will display the results of the expression. The *Read* takes expression as an input and parses and stores in memory as an internal data structure. *Eval* traverses the data structure, evaluates the called functions. *Print* displays the results with print ability. *Loop* iterates going back to read state to terminate the loop on exit. REPL expedites the turnaround time also supports ad hoc data query analysis.
- Spark is more nimble and well suited for big data analytics. Continuous micro-batch processing with integrated advanced analytics is based on its own streaming API, which is developer-friendly.
- Spark is more efficient compared to MapReduce. It is 100 times faster than MapReduce for the same process.
- The popular batch job processing engine for Hadoop, MapReduce poses a significant challenge due to its high-latency to the batch-mode response and it is difficult to maintain due to inherent inefficiencies associated with its architecture design and code. Spark's main component is the Spark Core Engine. It is complemented by a set of powerful, higher-level libraries that can be seamlessly used in the same application:
    - Spark SQL is a powerful query language with inbuilt DataFrames
    - Spark Streaming engine is for data streaming
    - Spark MLlib for machine learning along with machine learning pipelines models
    - GraphX with GraphFrames stores relationship between entities as a graphical representation.



Spark Core Engine

The base engine for Spark Core to perform large-scale parallel and distributed data processing is the Spark Core Engine. It performs the following functions:

- Fault-tolerant and recovery-based memory management
- Cluster job scheduling, distributing, and monitoring
- Storage device systems interfacing

Spark is built and based on an immutable, fault-tolerant, distributed collection of objects called **Resilient Distributed Dataset** (**RDD**) that can be operated on in parallel. Objects are created through loading an external dataset or distributing from the internal driver program. The operations performed on objects through RDD are transformations.

Transformation operations include map, filter, join, and union, and are performed on RDD and yield a new result.

Action operations include reduce, count, first and they return a value after running a computation on RDD.

The Spark Engine is designed efficiently. The transformations are actually computed when an action is called and the result is returned to the driver program. Transformations are lazy and do not compute their results right away; however, they remember the task to be performed and the dataset (for example, a file) to perform the operation. The transformed RDD can be recomputed for the next transformation. The advantage is avoiding unnecessary changes and computations in the process. All this is achieved by the in-memory engine where the data is persisted and cached for the RDD objects. Spark will keep the elements around on the cluster-cached memory for much faster access the next time you query it.

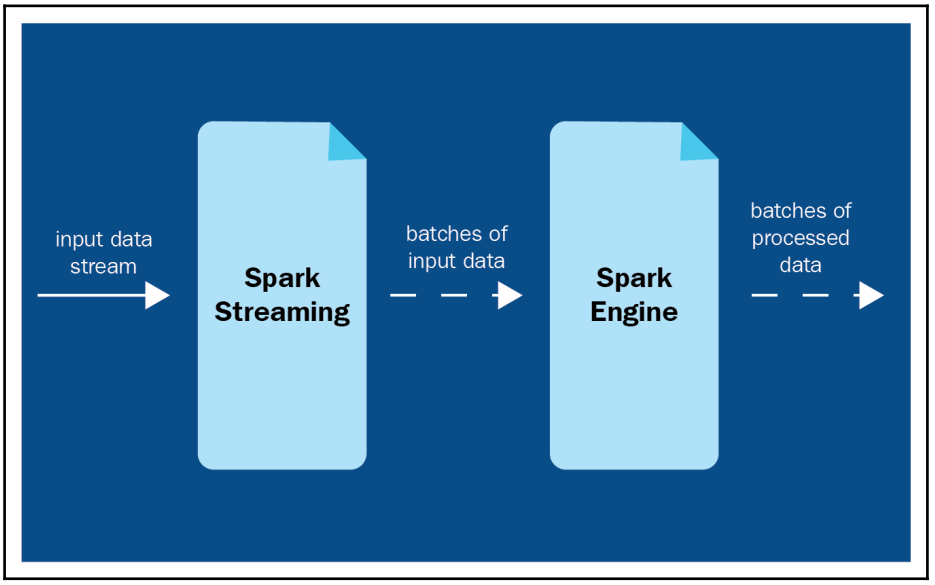
Spark SQL

The Spark component, Spark SQL, supports querying data either through SQL or through the Hive query language. Spark SQL is integrated with the Spark stack providing support for various data sources. It allows you to weave SQL queries with code transformations, making it a very powerful tool.

Spark Streaming

Spark Streaming is a powerful functionality built on in-memory technology. The Spark Streaming API is compatible with the Spark Core Engine. It facilitates both batch and streaming data to process real-time streaming data from systems such as web server log files, Twitter-based social media data, and so on. Spark interfaces with other tools, such as Kafka, for various messaging queues.

Spark Streaming receives the input data from upstream systems such as Apache Flume, Kafka, and so on, and divides the data into batches to process them with the Spark Engine and generate a final stream of results in batches to store them on HDFS/S3, and so on.



MLlib is a set of library functions that provides various algorithms of machine learning, such as classification, regression, clustering, and collaborative filtering. Apache Mahout (a machine learning library for Hadoop) is integrated into Spark MLlib. A few algorithms such as linear regression or k-means clustering also work with streaming data designed to scale out on a cluster.

GraphX provides ETL functionality, exploratory analysis, and iterative graph computations. It provides a library for manipulating graphs and performing graph-parallel operations on common graph algorithms such as **page rank**.

Spark is ideal to simplify challenging and compute-intensive task for real-time data processing of high volumes of streaming or archived data, both structured and unstructured, seamlessly integrating relevant complex capabilities, such as machine learning and graph algorithms.

Some challenges include operational complexity and the high skills required to develop and manage applications. Spark performs well with Hadoop to take advantage of Hadoop's HDFS. Performance tuning of both systems is imperative; otherwise, Spark's nuances can lead to out-of-memory error issues and memory lag, if jobs are not tuned well.

Visualization with big data systems

We are well versed with the saying: *garbage in, garbage out*. Identifying and recognizing trends, variations, and changes in data over time is often more important and data visualization is an inevitable step. Due to the complexity of information being processed in big data systems, visualizing data is one of the most important and useful ways to spot trends and create meaningful insights from a large number of data points.

We will discuss some real-time processing tools here:

- **Prometheus**: To visualize application and server metrics in real-time processing, data streams as a time-series database and visualizes that information. The health of the systems is gauged by the frequent data changes and large variations in the metrics typically indicate significant KPIs.
- **Elastic Stack**: It is popular for visualizing big data systems to visually interface with the results of calculations or raw metrics. It is also known as the ELK stack, composed of Logstash for data collection, Elasticsearch for indexing data, and Kibana for visualization.
- **SILK**: It is a similar stack achieved by using Apache Solr for indexing and a Kibana fork called **Banana for visualization**.
- **Jupyter Notebook** and **Apache Zeppelin** offer visualization interfaces for interactive exploration and visualization of data in a format conducive to sharing, presenting, or collaborating. This technology is typically used for interactive data science work and is termed a data *notebook*.

Data governance

Data governance is the process of classifying enterprise data and providing the right access and privileges for appropriate roles and personnel. It refers to the overall process of managing the availability, usability, integrity, and security of the data assets in an organization. A matured data governance model is a defined set of strategies, and includes a governing group and a well-orchestrated plan to execute those procedures.

Adopting an open source software development approach for platform/product development will ensure many advantages, such as the following:

- Seamless collaboration across different diverse technology teams spread across the organization
- Leverage and re-use of existing software and knowledge assets across the organization
- It ensures region, client, and country-specific needs are addressed

This approach for software development helps a governance mechanism in place to ensuring avoid duplication of work and enable transparency following a common standard framework.

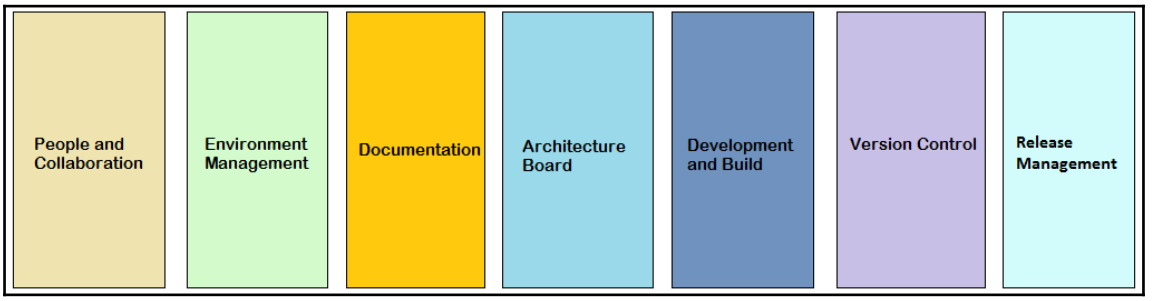
Data governance should define the minimum necessary rules instead of being an overhead. It should also balance across:

- Rules versus public (free for all)
- People versus process
- Empowerment versus directing

Open source governance operates on three pillars:

- Transparency
- Set governance parameters
- Faster delivery of every team and every member

Open source communities that practice transparency, encourage active participation, and recognize the contributions of all constituents are more likely to thrive, iterate, and strengthen their prospects. The guiding principles of governance in the open source community development model can be better demonstrated by describing the seven pillars illustrated in the following diagram:



People and collaboration in accordance with DevOps core concept

Technology is driven by people; therefore, the success of technology depends on a few attributes--it should be quite easily adoptable for people, it should be flexible, easy to learn, and convenient to collaborate with. We will discuss them here:

- Establishing ownership for the process example code, submission, and review process
- A common dashboard to check the status of any component, changes made, review status, testing, impacted components, and so on
- Adopt forums for discussions for resolution of queries than e-mail, Yammer Groups, and so on
- Regular (weekly, bi-weekly) all-hands deployment to discuss changes in components and roadmaps

Environment management

Management of environments in information technology is a complex and critical function.

- Parallel environments for development to be maintained, for example, current programs and other production fixes
- Define access restriction for components to interact with the database
- Define allocation of resources (disk space, threads/mappers, and so on) for each component/program

Documentation

Documentation of the ongoing work is important for the shell life of the project, features upgrades, and also to support and maintain as an operations manual.

- Comprehensive documentation of the core components, business/assembly line usage, governance, and so on. It is important for collaboration across teams, new members, and so on.
- The artifacts documents to help each team member could be revised regularly to add more granular details:
    - Master architecture reference document--created in TOGAF suggested format to speak in a common language
    - Developer guides
    - Governance guide
    - Deployment guide

Architecture board

The architecture board has the responsibility for the long-term enterprise and for ensuring the architecture meets the business goals such as service-oriented architecture, usage of components meeting the security guidelines, open source tools usage percentage as a roadmap, and so on.

- The **Architecture Review Board** (**ARB**) is the authority for defining and participating in Tollgate (or milestone) planning
- Define the Tollgate, the milestone for every program's high-level design
- Budget the re-factoring effort in every program and approve this in Tollgate meetings

- Centralize decision making for design approvals
- Review checklists used for Tollgate and reviews

Development and build best practices

Development best practices are the adoption of coding standards, adequate documentation, peer reviews, quality of the code, and so on, to ensure high quality of the code and performance. Build is a complex task with many interfaces. Adherence to proper guidelines will make it robust and well functioning as per the organization's needs.

- Automated builds with automated testing processes
- Peer review of source code based on sample
- Common IDE, code review tools (PMD, check style, and so on), build tools (Hudson and Maven)
- Standard build promotion procedures and schedules
- Publish environment stability on a weekly basis through continuous integration and testing
- Define a test bed and regression suite to execute impacted modules

Version control

Version control will ensure code changes are well tracked for traceability and accountability.

All of the organization teams using an enterprise standard tool for version control is the ideal. Governance in versioning has the following attributes:

- Automated email for every check-in to a controlled group of supervisors mailing list
- Define contributors for every program component and restrict access to the components
- Periodic audit of check-ins and approval from a component owner

Release management

A mature release management process is a strong asset for the organization to deliver timely dependable products to its customers with quality.

- Centralized release management
- Common priority defined across programs and features
- Employ microservices/incremental deployment--architecture for independent deployments.

Building enterprise applications with Spark

For enterprise applications to be successful, it is very important that you carefully define the data access, processing, and governance framework.

Client-services presentation tier

This graphical user interface will be backed by a set of APIs to help on-board new users. Some features that can be supported are as follows:

- Manage client data sources, file formats, delivery frequency, validation rules, join conditions (if multiple datasets are present), and so on.
- Validate and transform datasets
- Manage access to datasets
- Additional data delivery requirements from Eureka

Data catalog services

This graphical user interface will be backed by a set of APIs to provide data-related services. Some of the features that can be supported are:

- Search for any dataset/data in the data lake in a fashion similar to Google Search
- Browse (preview with pagination) search results
- Display the lineage and data profile of the selected data set

Workflow catalog

This graphical user interface will let you define workflows for an application and schedule runs. The main functionalities include:

- Create workflows for an application and schedule them
- Display the execution status of workflows, the time taken, as well as the datasets involved along with the lineage
- Ability to restart the process in case of a failure or from any given point
- Configure status updates, notifications, and alerts

Usage and tracking

This graphical user interface will be in use for Sentry and Navigator to track the usage of datasets across the cluster. The main functionalities include

- **Valid usage tracking of datasets**: How many times a dataset was accessed and by who
- **Invalid usage tracking**: Who tried to access a dataset they did not have access to
- Process tracking in terms of which user is running what process and what resources are being consumed
- Need to define dashboards based on requirements from the operations team

Security catalog

This graphical user interface will be backed by REST APIs to configure access control for user groups. The features included are as follows:

- Manage users and groups
- Manage user access to various datasets across the cluster and applications that can run in the data lake

Processing framework

This is where all transformations and processing on the data will take place. Processing can be batch as well as real-time with support for various frameworks such as Spark and MapReduce, along with querying engines such as Hive and Impala.

Ingestion services

Data can be ingested from various sources, ranging from batch to real-time streams using tools such as Sqoop, Flume, Kafka, and SFTP.

- Ingestion will be metadata-driven
- In order to ingest new data sources, new code is not required as long as it makes use of the supported ingestion methods, such as Kafka, Flume, Sqoop, and so on
- We should be able to register datasets and start cataloging them into the data catalog as soon as they are ingested

The ingested data can be used in multiple forms: stored, persisted into a device, published to external vendors. It can be accessed by other third-party programs through APIs.

- **Storage layers**: In order to maintain data integrity and isolation, we can spread our data in HDFS across multiple layers so that each layer defines a certain stage between ingesting raw data and generating insights.
- **Publishing**: This service will be used to publish data to external users and subscribers, as well as applications, as an outward push from the data lake.
- **Bulk API**: This service will be used to download data from the system using an asynchronous API. Users will make requests for data retrieval and they will be notified when the data set is ready. Delivery of the data can be provided in multiple ways:
    1. Download link.
    2. Push to SFTP location.
    3. HDFS location for internal users (same cluster or another cluster).
    4. API contracts need to defined.

- **Data Access API**: This API is similar to the Bulk API, but will only support small datasets such as a credit score to be synchronized frequently.
- **Notebooks**: These can include interfaces such as Apache Zeppelin or the Hue data science workbench, which will give a GUI interface for users to access datasets in the cluster and query them using Impala, Spark, R, and so on.

Data science

Data science as a field has many dimensions and applications. As we all are familiar with science by formulating reusable and established formulas, we understand the features, behavior patterns, and meaningful sights. In a similar way from relevant data too through engineering and statistical methods, we understand the behavior patterns and meaningful sights. Thus, it's also viewed as data plus science, the science of data or data science.

Data science has been in use for decades across industries. Many algorithms have been developed and are in use across the industries, including:

- K-means clustering
- Association rule mining
- Linear regression
- Logistic regression
- Naïve Bayesian classifiers
- Decision trees
- Time series analysis
- Text analytics
- Big data processing
- Visual work flows
- Apriori
- Neural networks

Combinations of the preceding algorithms are used to solve popular business problems such as the following, and new business opportunities are surfacing continuously:

Cross-selling	Find relationship among customer characteristics Match campaigns to potential customers
Product yield analysis	Classify product defects
Direct marketing	Classify customers Match campaigns to potential customers
Churn analysis	Predict the tendency of churning Win back customers
Cross-selling	Find relationship of products in transactions
Segmentation analysis	Classify customers Match campaigns to potential customers
Inventory analysis	Find relationship of products in transactions Make replenishment decision
Product-mix-analysis	Classify customers Match campaigns to potential customers Estimate the revenue of product mix
Fraud detection	Classify customers Detect unusual activities
Credit-rating	Classify customers Credit rating customers

For example, in data classification itself we can implement the following three models to get a refined and accurate model:

- **Random forests**: Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks. They operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Please check the following link: `https://en.wikipedia.org/wiki/Naive_Bayes_classifier`.

- **Naive Bayes**: Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters to be linear in the number of variables (features/predictors) in a learning problem.
- **Support vector machine**: Support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories. An SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

The following are the steps towards building a prediction model to solve a business problem:

- Defining the tangible business goal is the most important criteria. The business value and purpose of the data science problem agreement by different stakeholders is the most important step.
- Sponsorship and buy-in from key stakeholders is crucial to the success of the project.
- Collaboration among data engineers and data scientists is critical; otherwise, working in silos will not lead to project success.
- A data lake is a repository that gathers useful data from different source systems of appropriate, valid, meaningful, useful, and historical data required for the business problem. For building a data lake, capacity planning for the initial data and growth considerations for the future should be taken into account.
- Cleanse data as per quality norms to ensure only valid and appropriate data is used and no dirty or stale data enters the system.
- Feature engineering is core engineering of the data science project to extract meaningful insights (features) from the raw data.
- Feature selection is to eliminate irrelevant, redundant, or highly correlated features.

- Test the prediction models by following the proper validation methods, such as K-Fold Cross Validation or 70:30 models.
- Establish the model results. As with any project, the repetition of results accuracy validates the model's effectiveness.
- Optimize the model by continuous improvement with new data iteratively, and by fine-tuning.

Many popular statistical modeling tools are on the market, such as:

- SPSS modeler
- KNIME
- Microsoft Revolution Analytics
- RapidMiner
- SAP Predictive Analytics
- SAS Enterprise Miner
- Oracle Advanced Analytics (Oracle Data Miner, Oracle R Advanced Analytics)

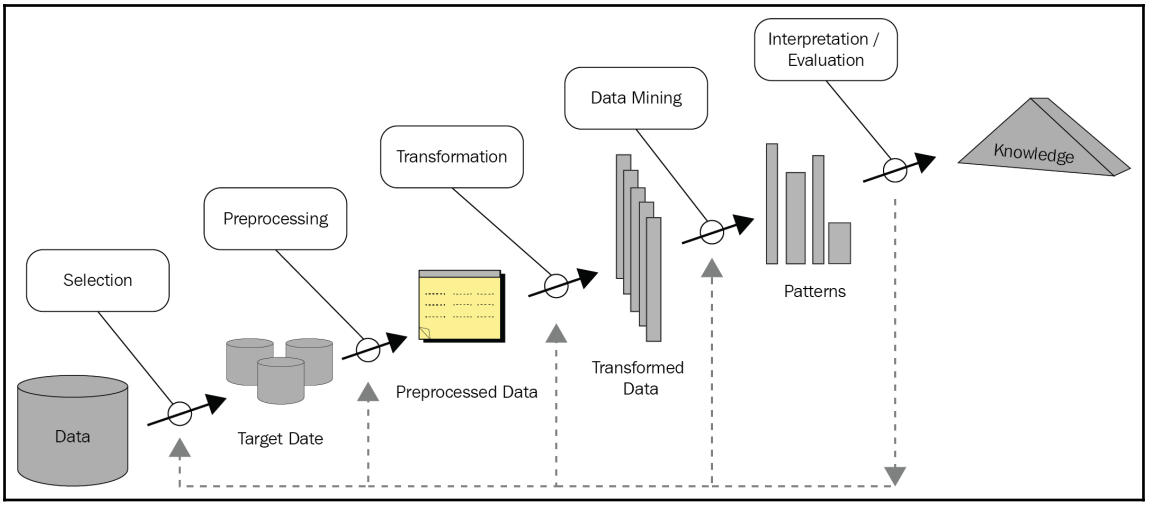
A few popular open source tools offer all the functionality on a par with commercial established statistical packages:

- R
- Python
- Scala
- MatLab
- Julia

The recent surge in low-cost technology availability such as Hadoop Eco systems, cloud computing, big data and open source tools has led to large-scale adoption by every industry from small enterprises to large giants.

Approach to data science

The approach to data science solutions involves the following staged approach:



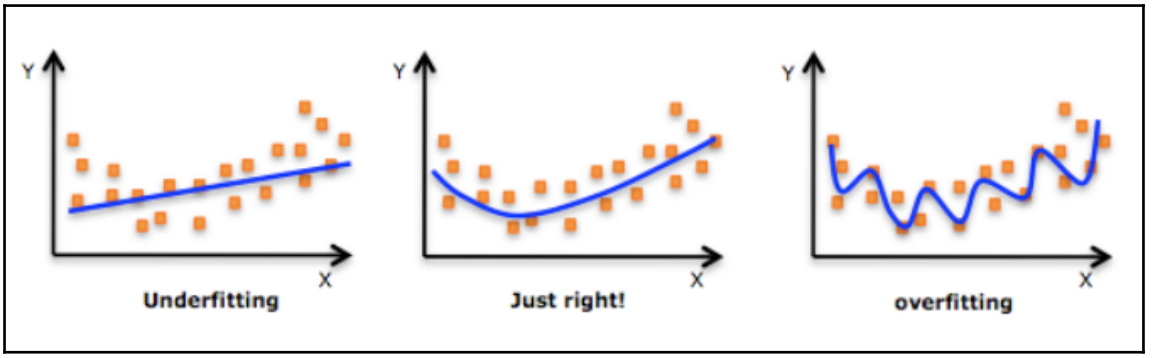
Knowledge mining (discovery) in datasets is an interactive and iterative process involving several steps for identifying valid, useful, and understandable patterns in data.

- **Data processing**: Data cleansing pre-transforms the raw data into an easy and convenient format for usage; a few related tasks are:
    - Sampling, that is, selecting representative subsets from a large population of data
    - Remove noise
    - Missing data handling for incomplete rows
    - Normalization of data
    - Feature extraction: data useful in a particular context is extracted
- **Data transformation**: Ensure usability of data by missing data treatment methods like:
    - Use only rows with relevant data
    - Substitution of values:
        - Mean value of particular attribute is used
        - Regression substitution with historical value from similar case
        - Matching imputation, similar attribute correlation case is used
        - Maximum likelihood, EM, and so on

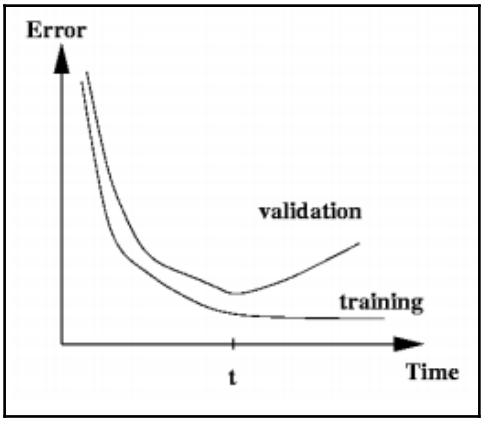
- **Data mining**: It is automating the searching patterns in the data by methods like:
    - Association rules
    - Sequence and path analysis
    - Clustering analysis:
        - Partitioning a data set into clusters or subsets based on some common attributes
    - Classification methods:
        - Division of samples into classes
        - Trained set is used previous labelled data
    - Regression models
        - Prediction of new values based on past data by inference
        - Compute new values for dependent variable values based on few other measured attributes
    - Visualization patterns
    - Classification is similar to clustering but requires classes to be defined ahead of time:
        - Classification with classifier based on input label is returned
        - Probabilistic classification is where classifier returns probable values to assign them to a class
        - Specific criteria like data more than 90% to avoid costly mistakes
        - Assign objects to class based on probability limits (greater than 40%, and so on)
    - Regression and forecasting
        - Data table statistical correlation:
            - Data distribution in functional forms with prior assumptions
            - Machine learning-based algorithms
        - Curve fitting:
            - A well defined and known function underlying the data is explored
            - Theory- and expertise based

- Machine learning:
    - Supervised:
        - Both input and desired results are part of training data
        - Model training process inputs are based on correct known target and results
        - Proper training, validation, and test set construction are crucial
        - Fast and accurate results
        - Ability to generalize, new data should produce correct results without prior knowledge of target
        - Generalization means the ability to produce valid outputs for inputs not available during training
    - Unsupervised:
        - Correct results are not provided to the model during training
        - Input data clustered to classes based on their statistical properties alone
        - Significance of cluster and label
        - Even small number of objects which are representative of desired classes and labels can be applied
- Curve fitting
    - Is by using proper subsets and early stopping
    - Based on data learning and not just underlying function

- Data used in training should perform well with new data

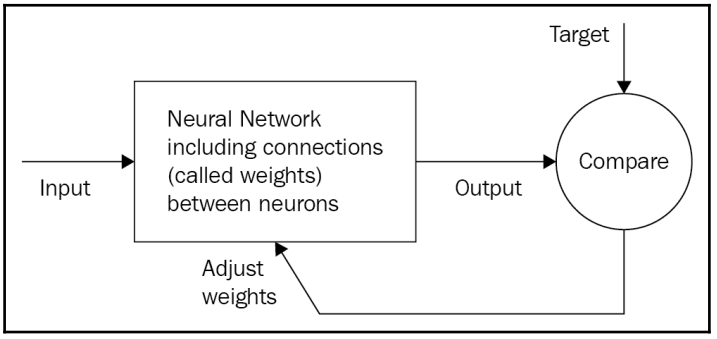


- Data sets:
  - **Training set**: Used for learning where target value is known.
  - **Validation set**: Used to tune classifier architecture to estimate error.
  - **Test set**: Performance of classifier is assessed only; it's never used in the training process. Test set error should provide unbiased generalization error estimate.

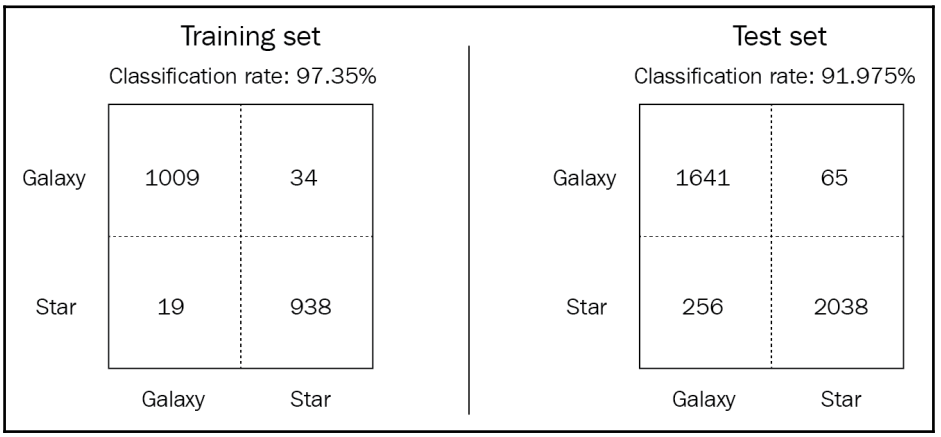


- **Data selection**: Garbage In , Garbage Out, underlying model representation should be training, validation, and test data.
- **Unbalanced datasets**:
    - Network minimizes overall error so the proportion of types of data in the set is critical
    - Loss matrix inclusion
    - An even representation of different cases is the best approach to interpret networks decision

- Learning process:
    - Back propagation:
        - Output values compared with target value to compute the predefined error function
        - Error is fed back into the network
        - Using these inputs the algorithm adjusts the weights of each connection to reduce the value of error function.
        - The network will converge after repeating the process for longer training cycles for sufficient numbers



- Results:
  - Confusion matrix: The prediction results on X axis are compared to target values on Y. Rows represent the true classes and columns predicted classes.



- Completeness and contamination
  - Performances are rated as the following criteria for the classifiers, for example between two classes:
    - **Completeness**: The percentage of objects of class A correctly classified
    - **Contamination**: The percentage of objects of class A incorrectly classified as objects belonging to class B
    - **Classification rate**: The overall percentage of objects correctly classified

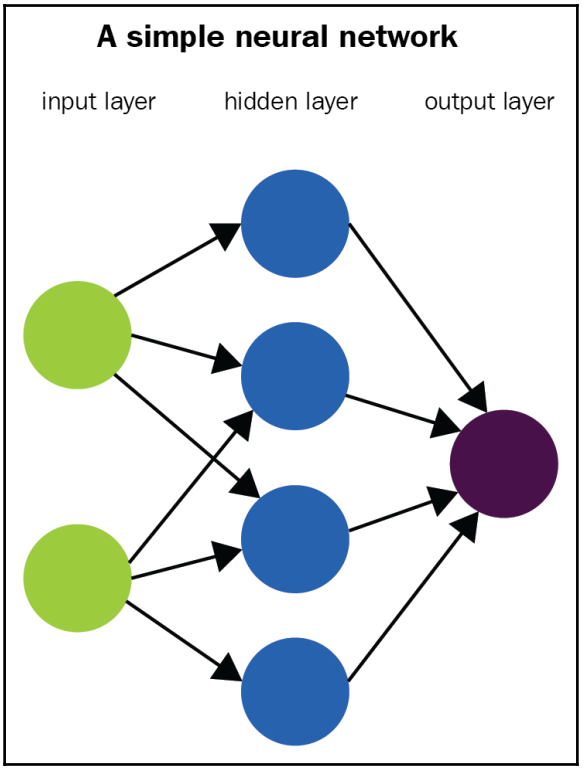
Supervised models

- Neural networks
- Multi layer perceptron
- Decision trees

Neural network

It is a structured flow consisting of the input layer of neurons and the output layer of neurons with one or more hidden layers in the middle.

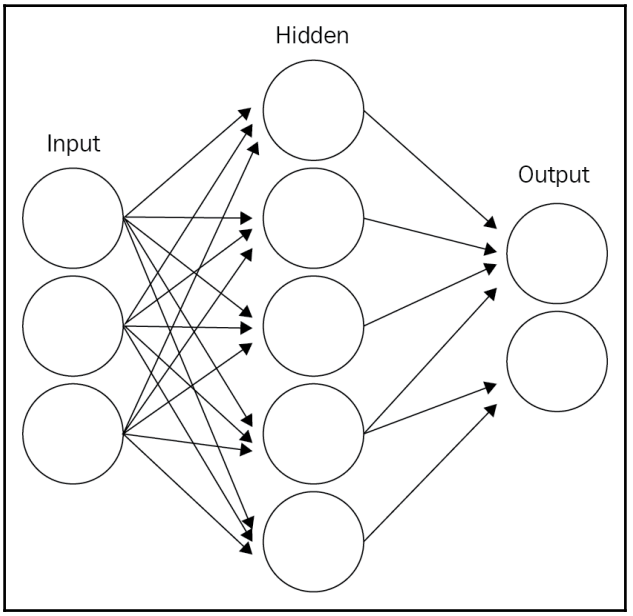
Neurons are well connected with adjacent layers though being in different topologies, they are connection by a choice of activation function, the **weights** are assigned as function values associated with the connections of various types and architectures.



Artificial neural networks this is inspired by the biological nervous system process. An artificial neural network is an information-processing mechanism.

Highly interconnected large number of simple processing neuron elements working together to address specific problems.

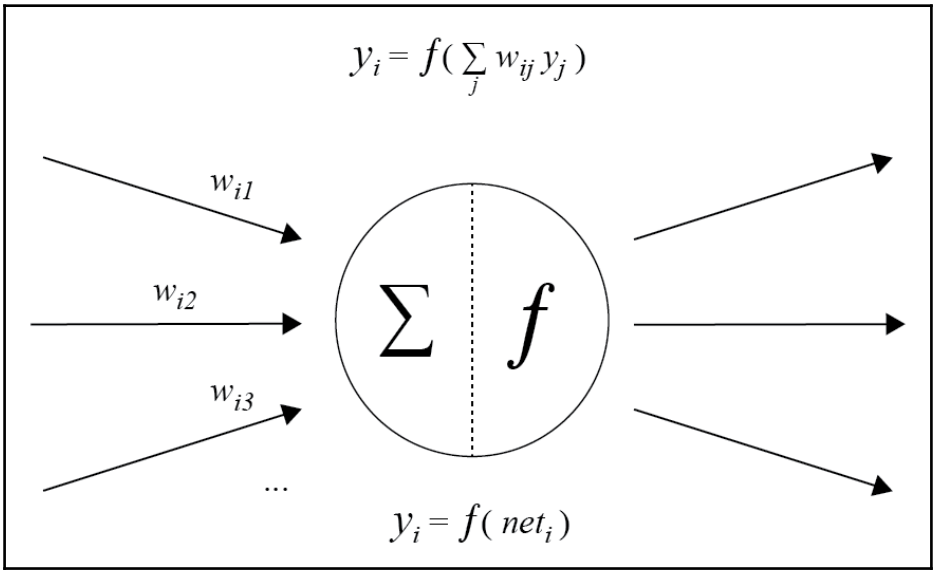
A simple artificial neuron:



A node or unit is a basic computational element that receives input from other units or external source.

To model synaptic learning, each input is considered with associated weight w.

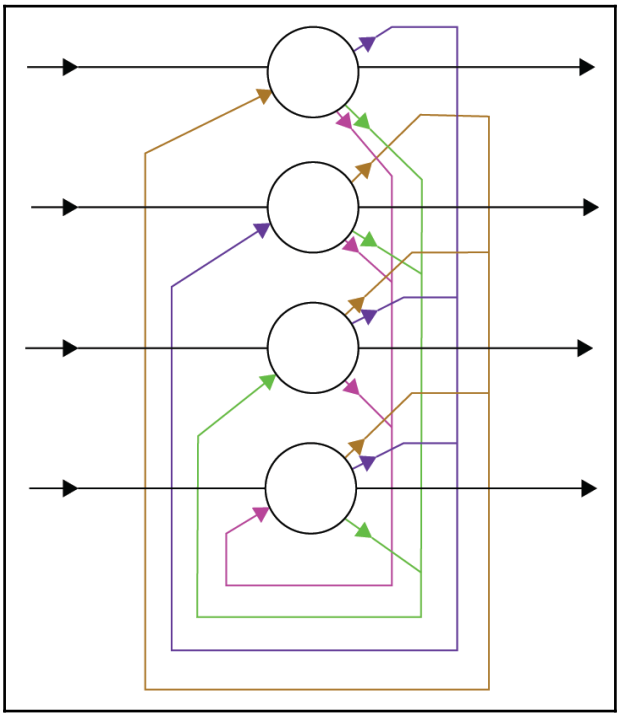
The weighted sum of its inputs is computed as a function by the unit:



The different types of neural network are:

- **Feedforward**: **Adaptive Linear Neuron** (**ADALINE**), RBF, single layer perception
- **Self-organised**: SOM (Kohonen Maps)
- **Recurrent**: Simple recurrent network, Hopfield network
- **Stochastic**: Boltzmann machines, RBM
- **Modular**: **Associative Neural Networks** (**ASNN**), committee of machines

- **Others**: NeuroFuzzy, Cascades, PPS, GTM, Spiking (SNN), Instantaneously Trained



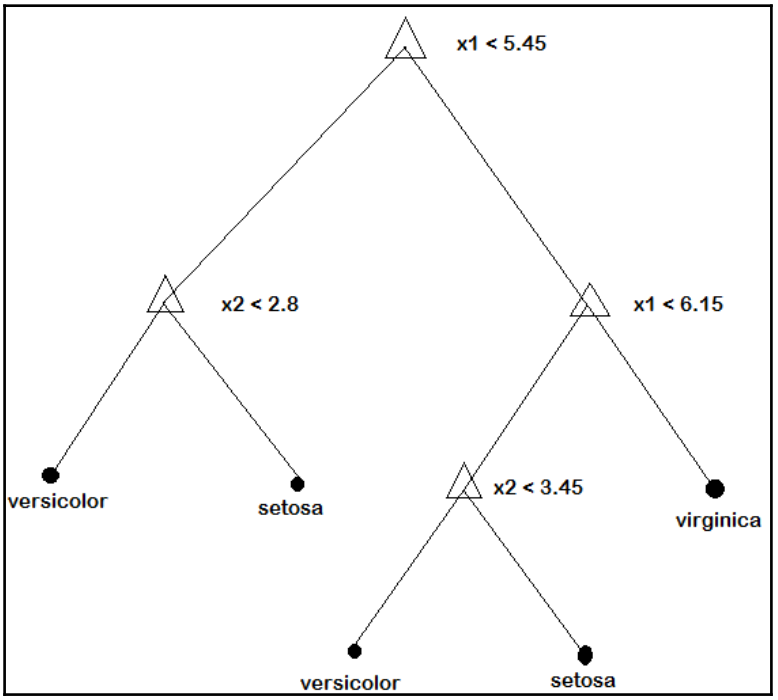
Multi layer perceptron

This is a most popular supervised model consisting of multiple layers of computational units inter connected in a feed-forward way usually. Each neuron in one layer is connected to subsequent layer neurons through direct connections.

Decision tree

This is a classification method with a set of simple rules; they are non-parametric without the need for any assumptions on distribution of the variables in each class.

As example decision tree is depicted in following:



Unsupervised models

- Clusters
- Distances
- Normalization
- K-means
- Self-organizing maps

Clusters

Clusters are hierarchical, it finds successive clusters using previously assigned clusters with bottom up (agglomerative) or top-down (divisive) and partitional type cluster depicted below as right and left side respectively.

Distances

To determine the similarity between two clusters and the shape of clusters.

Normalization

- **VAR**: For a transformed set of data points for each attribute, the mean is reduced to zero; this is by subtracting the mean of each attribute from the values of the attributes and dividing the result by the standard deviation of the attribute.
- **RANGE (Min-Max Normalization)**: It subtracts the minimum value of an attribute from each value of the attribute and then divides the difference by the range of the attribute. The advantage is preserving all relationship in thedata precisely, without adding any bias.
- **SOFTMAX**: It is a way of reducing the influence of extreme values or outliers in the data without removing them from the dataset. It is useful when you have outlier data that you wish to include in the dataset while still preserving the significance of data within a standard deviation of the mean.

K-means

K-means is popular model as it's fast and simple; however it does not yield the same result with every run.

- Partitions data into K clusters based on their features
- Each cluster is represented by its centroid, the center of the cluster points
- Each point is assigned to the nearest cluster
- The goal is to minimize intra-cluster variance or the sum of squares of distances between data and the corresponding cluster centroid
- Computes the mean point--centroid

A new partition is built by associating each point with the nearest centroid. Computes the mean point or centroid of each set.

The recent surge of low-cost technology availability such as Hadoop eco systems, cloud computing, big data, and open source tools has led to large-scale adoption by every industry from small to large giants. Data science penetration is also witnessed across every industry with eco system being available.

Summary

In this chapter, we have covered the key concepts of building big data applications, covering the tools for data discovery, quality, and ingestion. We discussed the Spark Stream in-memory engine and its versatility; data science models for various industry solutions were also discussed.

7
DevOps - Continuous Integration and Delivery

In this chapter, we will learn about implementing DevOps core process such as source code repository, code review, artifacts repository, continuous testing, continuous development, continuous integration. As discussed in previous chapters on big data and cloud, these processes are highly valuable to apply to every phase. We will focus on few popular tools like Git, Jenkins, Maven, Gerrit, Nexus, Selenium, and so on.

- **Continuous integration** (**CI**)
- **Continuous delivery** (**CD**)
- Jenkins tool setup
- Configuration management-Jenkins
- Source code management-Git
- Build management- Maven
- Source code review-Gerrit
- Repository management-Nexus
- Test Automation- Selenium
- Continuous deployment-Pipelines
- Jenkins client setup
- Jenkins security
- Jenkins metrics

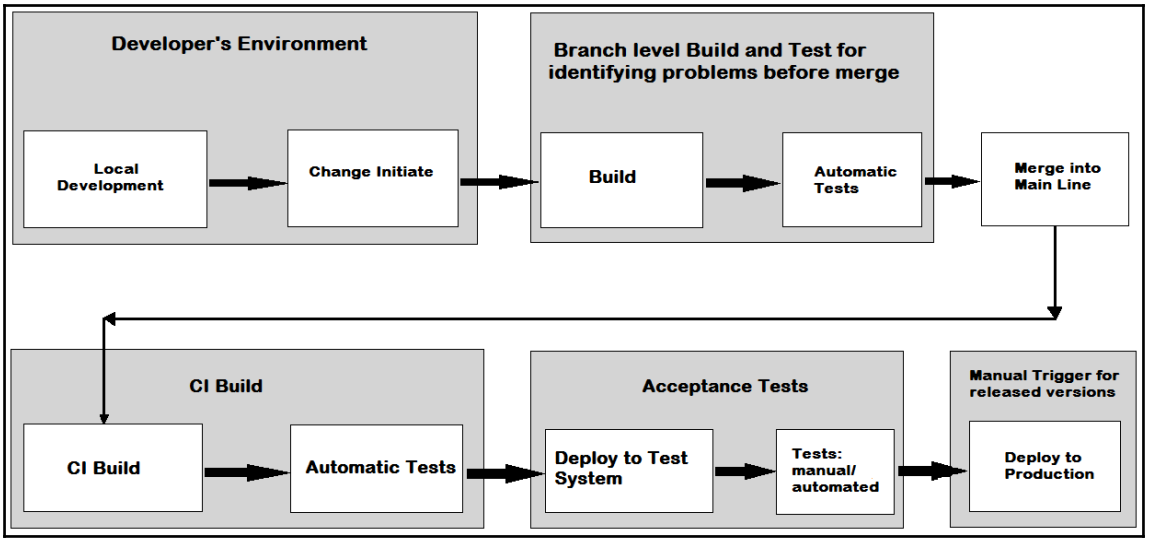
Continuous integration and continuous delivery are popular and valuable processes to ensure high-quality and timely software delivery. Continuous integration is the integrated software development process where multiple developers adhere to the agile methodology and adapt it to best practices like the following:

- Ensure all development code is subject to a version control system
- An adequate code review process is incorporated
- Changes to code are integrated, tested, and built quickly
- Build process is integrated to run unit tests and automated
- Attend to the build errors immediately, turn around quickly
- Tracking and Metrics of build results and repository management.
- Transparency and a user-friendly build process

Continuous delivery is the process of extending the continuous integration.

- The most current and latest version of the software is readily available
- Changes passing through the testing cycle from the technical and quality standpoint are ready for deployment
- Automate the shipment and deployment process

The continuous integration process is depicted as follows:



The continuous integration process is detailed following:

- **The developer's environment**: Developers create code changes in a local workspace with an Integrated Development Environment runtime and with build tools physically installed on PC, or a cloud-based (Web IDE). They do unit level testing, data validations, code performance checks, and so on. The code changes done by the developer are pushed to the source code management system.
- The typical continuous integration and deployment cycle is comprises of setting up a CI/CD infrastructure and processes as listed:
    - The source code version and repository management system
    - A process scheduler to initiate the orchestration pipeline
    - A build process to manage code builds and scheduled tests
    - Build nodes for executing the build
    - Testing process on identified test nodes for automated testing
    - Build outcome artifact repository
    - Artefact repository to store build results
    - Scenario and acceptance tests on test nodes
    - Application installation with deploy tool on to runtime systems
    - Acceptance tests for applications deployed on the runtime systems

The quality manager will approve the acceptance tests to agree to deployment test systems.

The delivery manager will approve the application deployment to production.

Best practices for CI/CD

- **Using version control**: In collaborative development environments with simultaneous development there will be multiple challenges:
    - A source code management system defines a single source of truth for the code after placing the code under a version control system. The source code will be reproducible by effectively adopting the merge process for mainline development and loop lines for bug fixes and so on in the system. Git is a popular source code management system and GitHub is a cloud variant as a **Software as Service** (**SaaS**) model:

- **Automate the build**: Standardized automated build procedure will stabilize the build process to produces dependable results. The matured build process must contain the build description and all the dependencies to execute the build with a standardized build tool installation. Jenkins is the most versatile tool for build schedules; it offers a convenient UI and also has plug-ins integrating most popular tools for continuous integration.
- **Tests in the build**: A few tests are to be performed to validate effectiveness and fitness of code beyond just the syntactical correctness of the code as follows:
    - Unit tests operate directly on build results
    - Static code checks on source code prior to developer check-in. Git pre-commit triggers or CI system could be used to set up a gating or non-gating check
    - Scenario tests for new build applications to be installed and started
    - Functional performance of the code:

  Unit test frameworks are popular across source code technologies like JUnit for Java. Selenium Framework provides graphical user interfaces and browser behavior.

  Implementing these tests on the developer's workstation early as part of the build saves time and effort addressing bugs discovered later in the development process.

- **Early and frequent commit of code**: In a distributed development environment with multiple projects, each team or developer intends to integrate their code with the mainline. Also, the feature branches change to be integrated into the main line. It's a best practice to integrate code quickly and early. The time delay increases between new changes and merging with the mainline will increase the risk of product instability, the time taken, and complications as the main-line evolves from the baseline. Hence each developer working with the feature branch should push their code at least once per day. For main branch inactive projects, the high effort for constant rebasing must be evaluated before implementing.
- **Every change to be built**: Developer changes are to be incorporated into the mainline, however, they can potentially destabilize the mainline affecting its integrity for the developers relying on the main line.

Continuous integration addresses this with the best practice of continuous build for any code change committed. Any broken build requires immediate action as a broken build blocks the entire evolution of the mainline and it will be expensive depending on the frequency of commits and such issues. These issues can be minimized by enforcing branch level builds.

Push for review in Gerrit or pull request in GitHub are effective mechanisms to propose changes and check the quality of changes by identifying problems before they're pushed into the mainline, causing rework.

- **Address build errors quickly**: The best practice of building at the branch level for each change will put the onus on the respective developers to fix their code build issues immediately rather than propagate it to the main branch. This forms a continuous cycle of Change-Commit-Build-Fix at each respective branch level.
- **Build fast**: The quick turnaround of builds, results, and tests by automatic processes should be vital inputs for the developer workflow; a short wait time will be good for the performance of the continuous integration process on overall cycle efficiency.

This is a balancing act between integrating new changes securely to the main branch and simultaneously building, validating, and scenario testing. At times, there could be conflicting objectives so trade-offs need to be achieved to find a compromise between different levels of acceptance criteria, considering the quality of the mainline is most important. Criteria include syntactical correctness, unit tests, and fast-running scenario tests for changes incorporated.

- **Pre-production run**: Multiple setups and environments at various stages of the production pipeline cause errors. This would apply to developer environments, branch level build configurations, and central main build environments. Hence the machines where scenario tests are performed should be similar and have a comparable configuration to the main production systems.

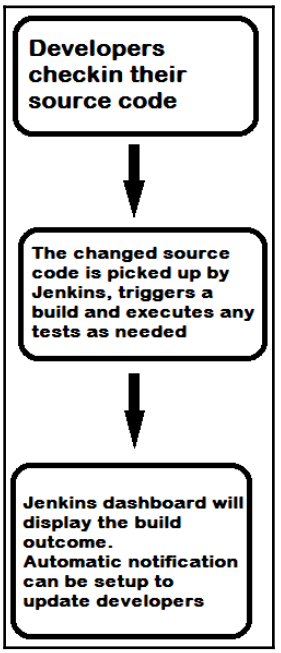
Manual adherence to an identical configuration is a herculean task; this is where DevOps value addition and core value proposition and treat the infrastructure setup and configuration similar to writing code. All the software and configuration for the machine are defined as source files which enable you to recreate identical systems; we will cover them in more detail in `Chapter 8`, *DevOps Continuous Deployment*.

- **The build process is transparent**: The build status and records of the last change must be available to ascertain the quality of the build for everyone. Gerrit is a change review tool and can be effectively used to record and track code changes, the build status, and related comments. Jenkins flow plugins offer build team and developers a complete end to end overview of the continuous integration process for source code management tools, the build scheduler, the test landscape, the artifact repository, and others as applicable.
- **Automate the deployment**: Installation of the application to a runtime system in an automated way is called deployment and there are several ways to accomplish this.
    - Automated scenario tests should be part of the acceptance process for changes proposed. These can be triggered by builds to ensure product quality.
    - Multiple runtime systems like JEE servers are set up to avoid single-instance bottlenecks of serializing test requests and the ability to run parallel test queries. Using a single system also has associated overheads in recreating the environment with change overhead for every test case, causing a degeneration a performance.
    - Docker or container technology to install and start runtime systems on demand in well-defined states, to be removed afterward (we will discuss container technology in `Chapter 9`, *Containers, IoT, and Microservices*, in depth).
    - Automated test cases, since the frequency and time of validations of new comments, is not predictable in most cases, so scheduling daily jobs at a given time is an option to explore, where the build is deployed to a test system and notified after successful deployment.
    - The deployment to production is a manual conscious decision satisfying all quality standards and ensure the change is appropriate to be deployed to production. If it can also be automated with confidence, that's the highest accomplishment of automated continuous deployment too.

Continuous delivery means that any change integrated is validated adequately so that it is ready to be deployed to production. It doesn't require every change to be deployed to production automatically.

Jenkins setup

We will start with Jenkins as it's the core component of the continuous integration process. The Jenkins process workflow is shown as follows:



See the Jenkins homepage at: `https://jenkins.io/index.html`, shown as follows:

Prerequisites to install Jenkins

Jenkins installation and configuration requirements should be planned well as prescribed on the Jenkins homepage based on the following parameters:

- Operating system--Linux versions of Ubuntu/Debian, Red Hat/Fedora/CentOS, openSUSE, FreeBSD, OpenBSD, Gentoo, Windows, macOS X
- JDK version
- Memory
- Disk space
- Java Containers--The Jenkins WAR file can run on any servlet-supported engine such as tomcat or Glassfish application servers.

Jenkins can be installed in different modes as per its utility:

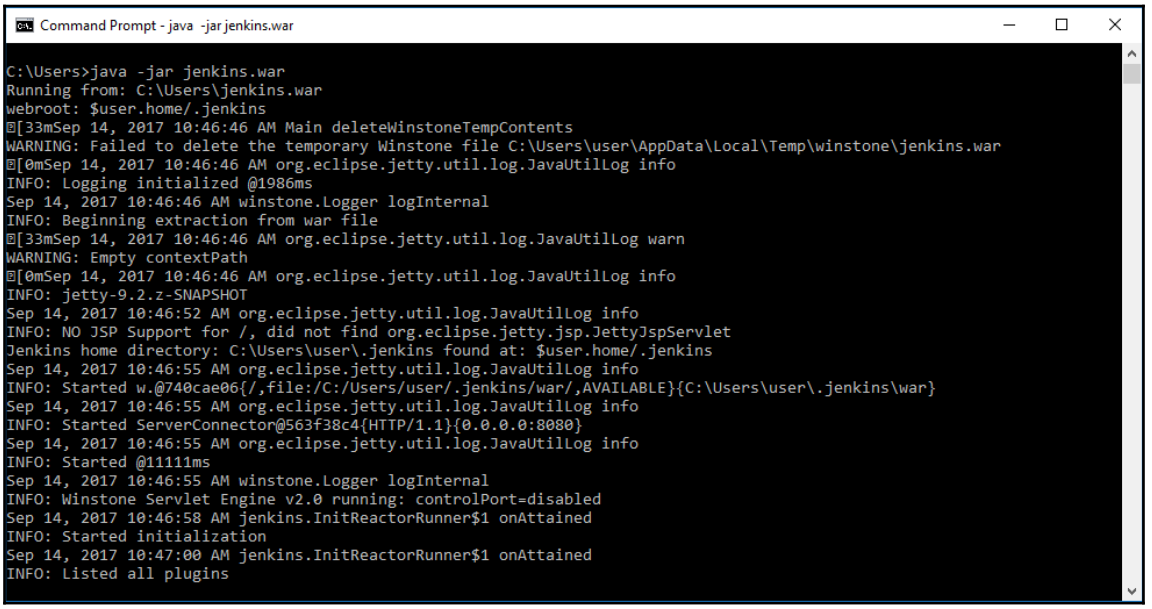
- **Standalone**: Jenkins can run standalone in its own process using its own built-in web server (Jetty) for experimentation and small projects
- **Servlet-based**: It can also run as one servlet framework for development projects.
- **Multi-node setup for staging or production**: Distributed client-server setup; the Jenkins advanced installation procedure is recommended.

Standalone Installation

A standalone installation as suggested in the name is all by itself on a single machine (as opposed to multiple systems for different tasks):

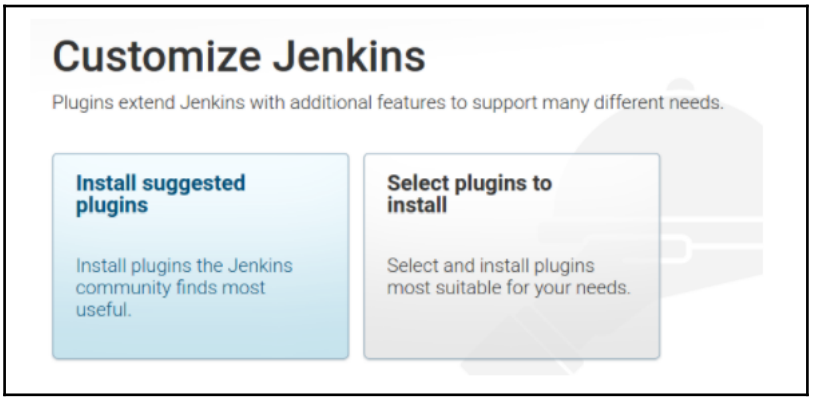
1. Standalone installation requires JDK to be installed on the system.
2. Download the `Jenkins.war` file
3. Open the command prompt and, at the location of the `Jenkins.war` file, run the command:

```
C:>Java –jar Jenkins.war
```

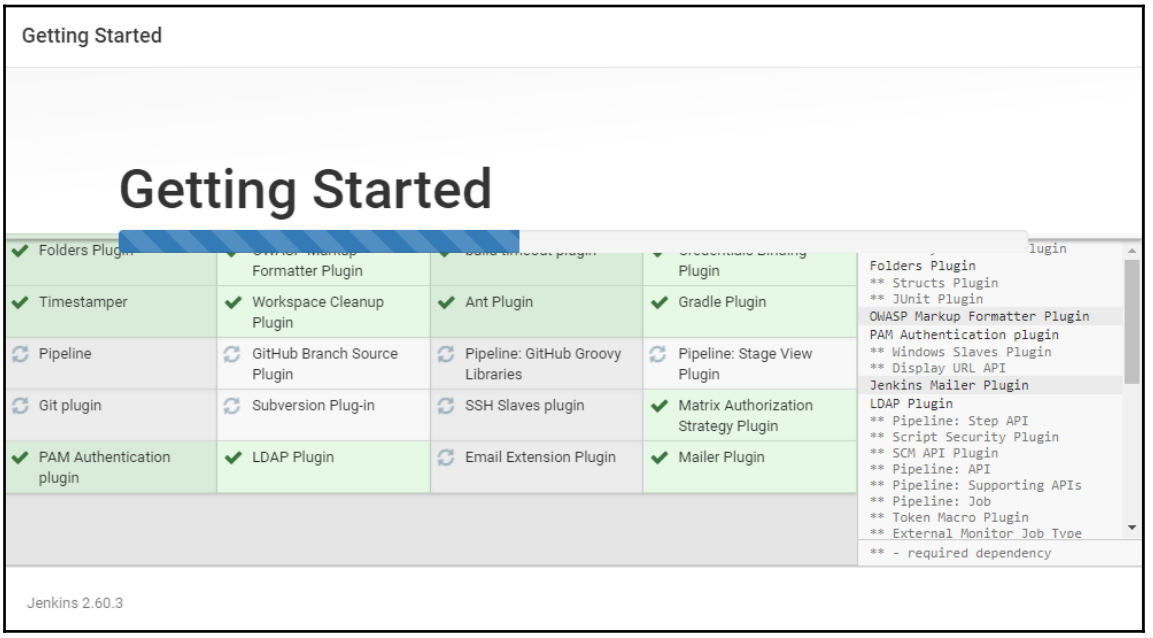


During initialization, a few tasks will run and the following screen will appear during the installation process:

1. The initial screen page will ask about the plugin options:



2. Plugins will be installed as per the selected configuration in the preceding option:

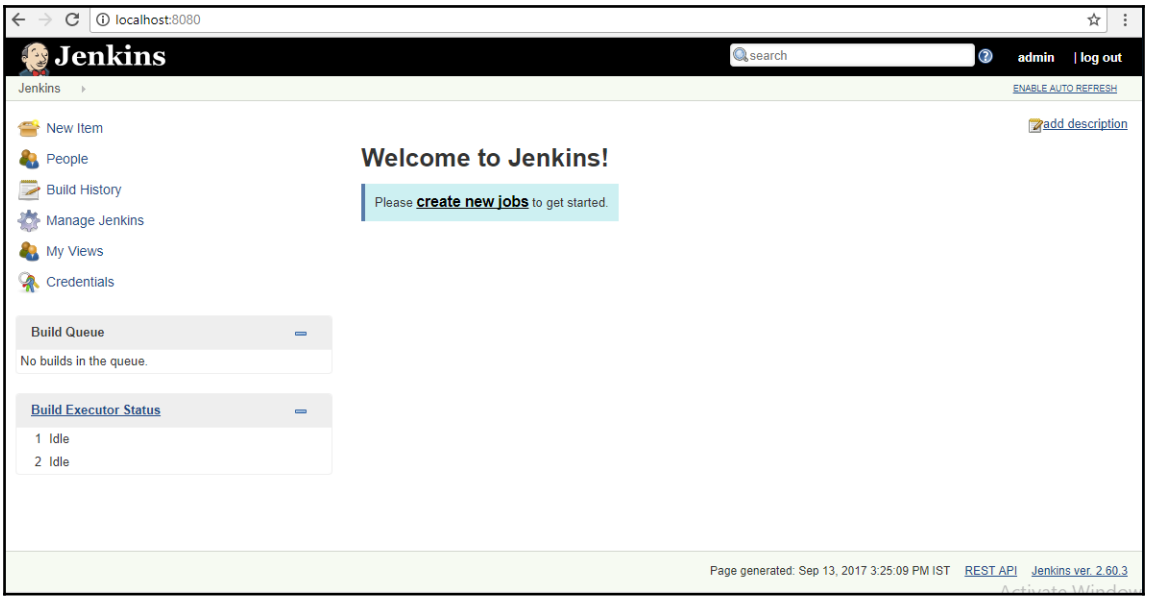


3. After successful installation, the following admin credential creation page will pop up:

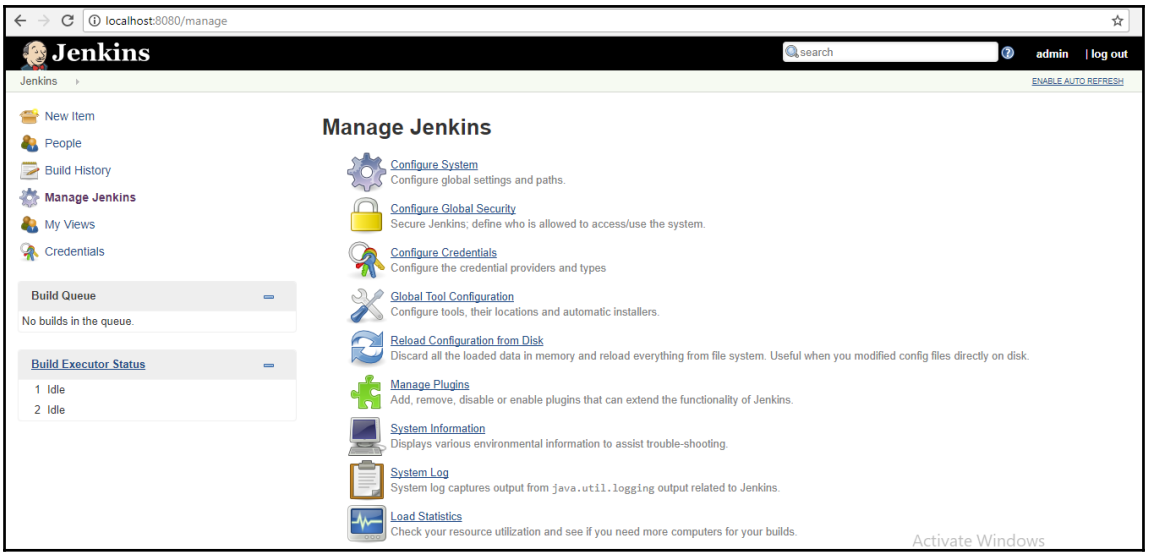
4. **Accessing Jenkins**: After successful installation, Jenkins can be accessed through a web browser from your local machine as follows:

```
http://localhost:8080
```

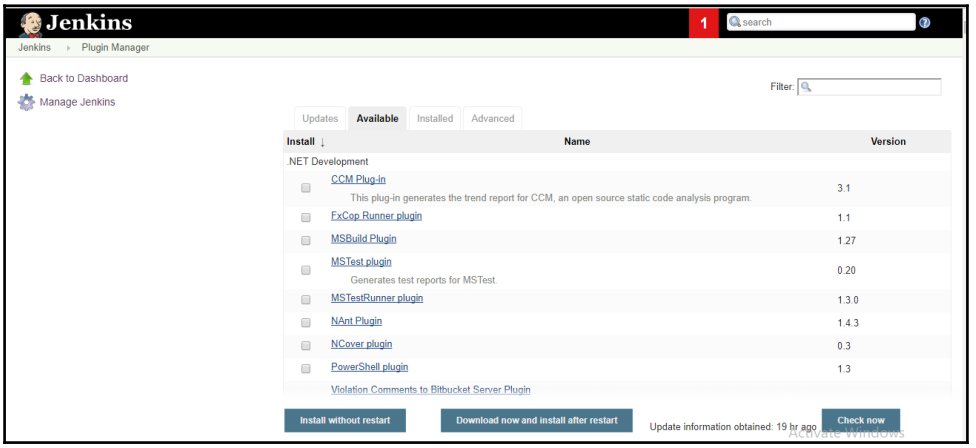
5. The Jenkins dashboard will open at this link:



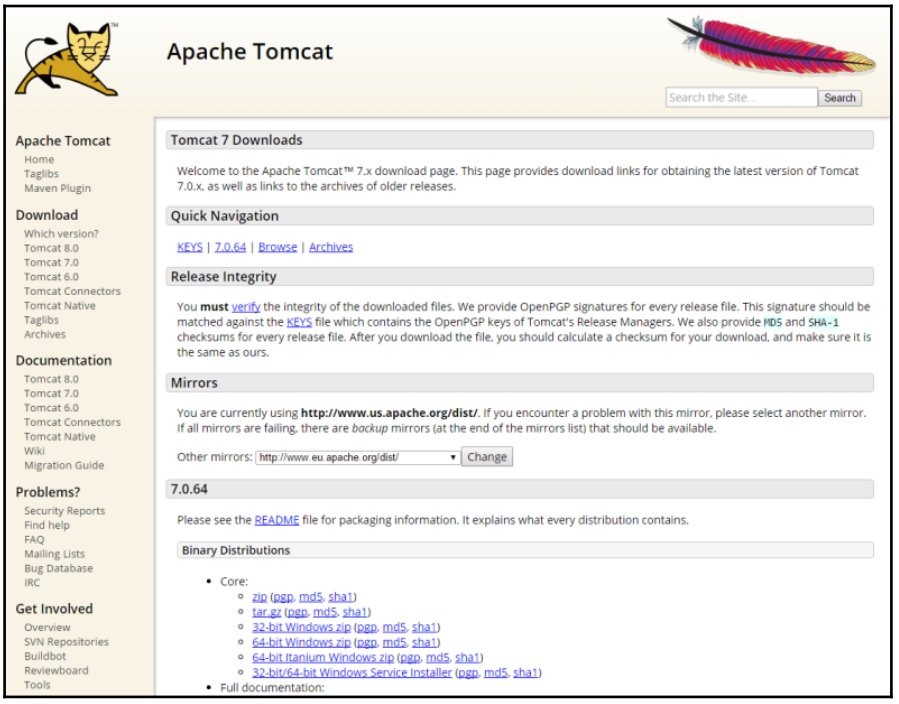
6. The **Manage Jenkins** option in the dashboard will provide various options to configure various parameters:



7. The **Manage Plugins** option in the dashboard is an important option with a very wide choice of plugins to integrate with source code systems, authentication systems, various development platforms, and so on.



Installing Jenkins on a Servlet Engine needs the installation of Tomcat or Glassfish.



1. Copy the `Jenkins.war` file it to the web apps folder in the `tomcat` folder.
2. Start the Tomcat server from the Tomcat `bin` directory.
3. `http://localhost:8080/Jenkins`--access Jenkins on Tomcat server.

Linux system installation on Ubuntu

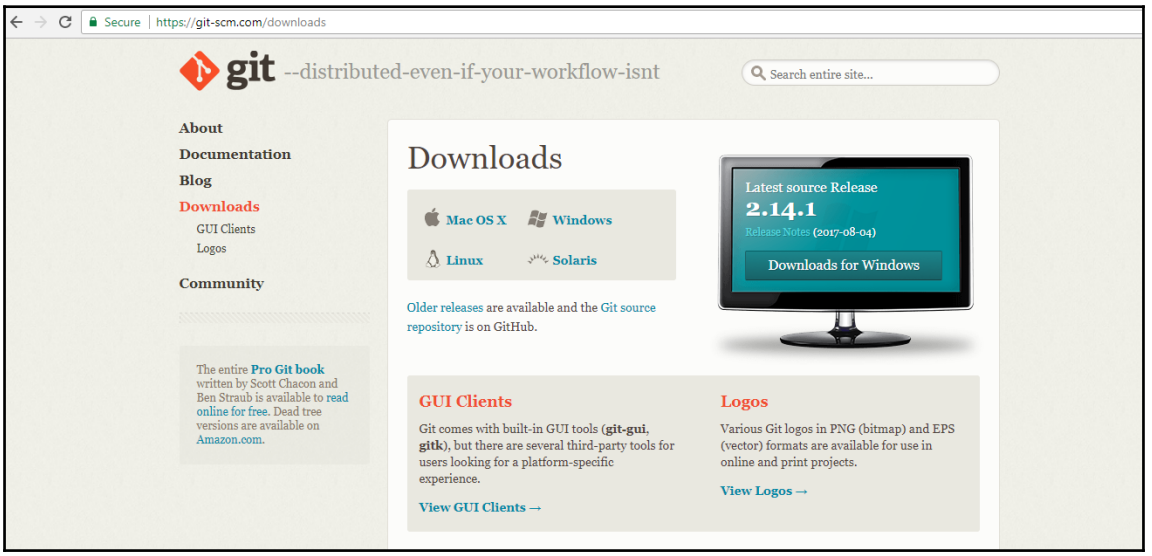
1. Log into the server and update: `sudo apt-get -y update`.
2. Install Java: `sudo apt-get install -y default-jdk`.
3. Download an Ubuntu version from `Jenkins-ci.org` site using `wget` command: `wget http://pkg.jenkins-ci.org/debian-rc/binary/jenkins_2.0_all.deb`.
4. Package install--`sudodpkg - i Jenkins.zip`.
5. Dependency resolve by `sudo apt - get -f install`.
6. Access Jenkins on port `http://localhost:8080/Jenkins`.
7. Continue with the steps listed in preceding figure.
8. To initialize Jenkins at startup, add the command `/etc/init.d/jenkins` start in `/etc/rc.local` file.

Git (SCM) integration with Jenkins

Git is the most popular source code management system and offers extensive benefits such as:

- Version control lets you maintain multiple versions of the code for different purposes
- A code repository is required to keep all project-related code in one place
- Collaboration among users and intervention for debugging purposes

Git can be downloaded from `https://git-scm.com/downloads`:



Multiple platforms versions such as Linux, Windows, and so on are available within desktop and web flavors.

There can be multiple types of repositories:

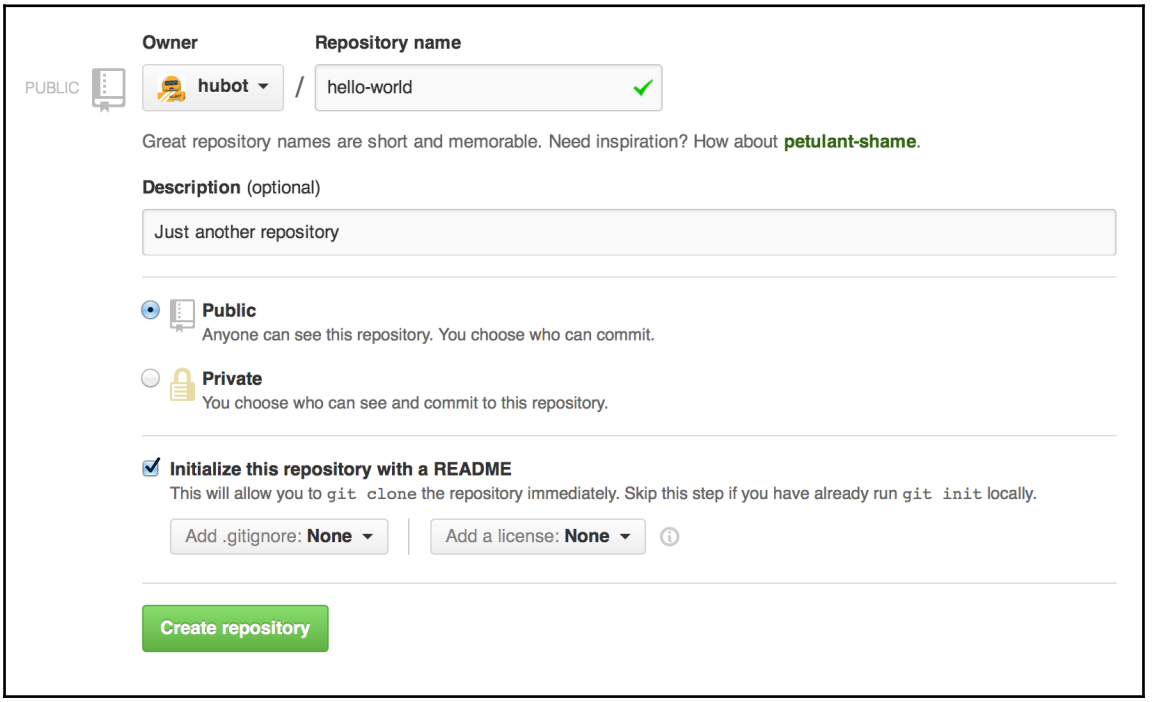
- A public repository created on GitHub can give read access to everyone but write or commit access is given to chosen individuals or groups
- A private repository permits collaborators for participation and is a paid subscription to GitHub
- A local repository is a desktop version without the need for an internet connection
- A remote repository is a web-based repository for extended features like issue management and pull requests

GitHub provides options to synchronize code changes from a single computer or between multiple computers.

Pull changes will sync code changes from a desktop with an online repository and clone options will create a new copy of the repository on the computer.

Performing these tasks enables us to maintain source code on cloud-based SaaS system

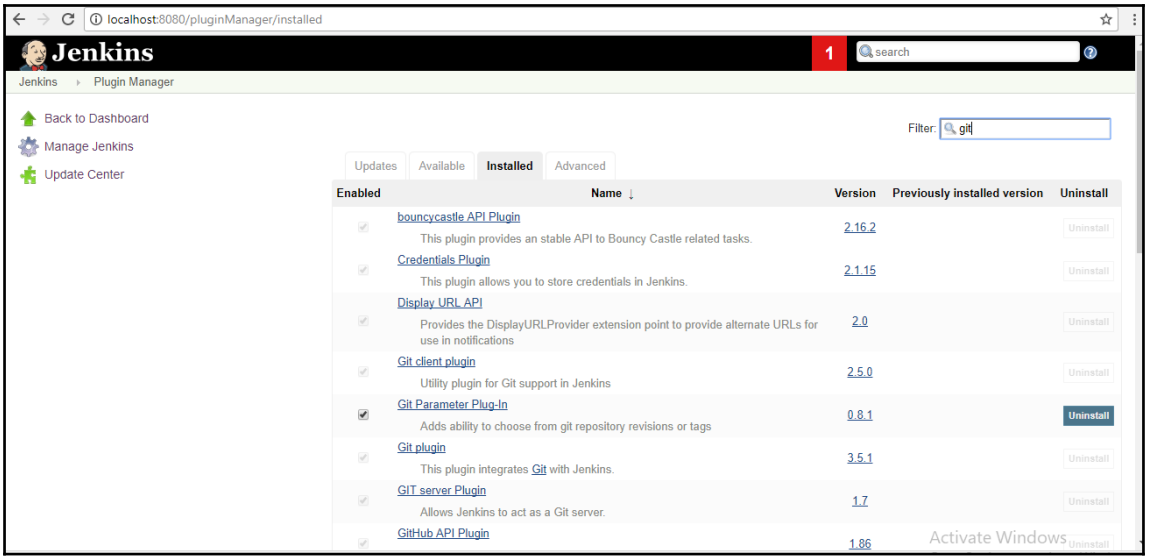
1. Create a sign-in account on GitHub.
2. Create a project repository for organizing your project-related code.



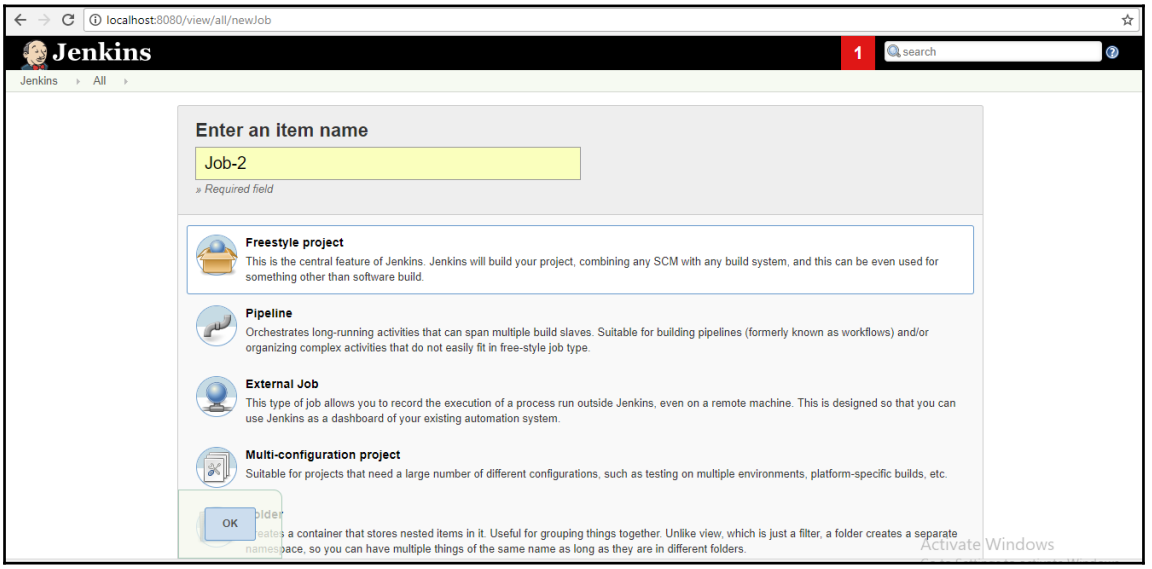
Integrating GitHub with Jenkins

To integrate a GitHub repository with Jenkins, follow these steps:

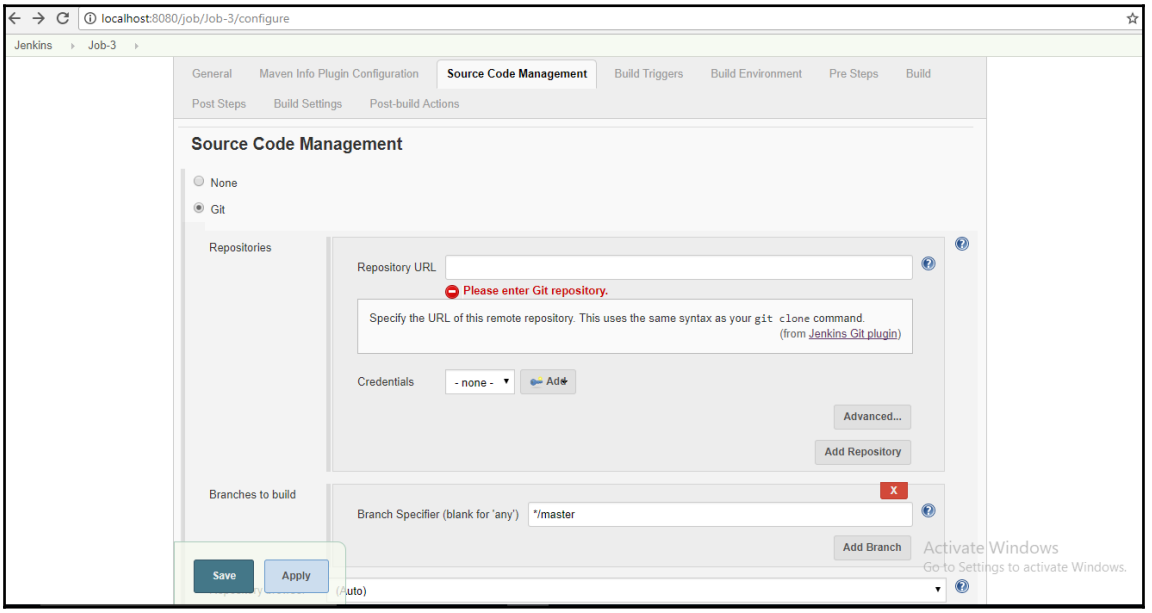
1. In **Manage Plugins**, search for Git plugin under the filter section and install it.
2. If it's installed by default, we can find it on the **Installed** tab as follows:



3. After Jenkins is restarted, create new item on the Jenkins homepage will give the following screen:



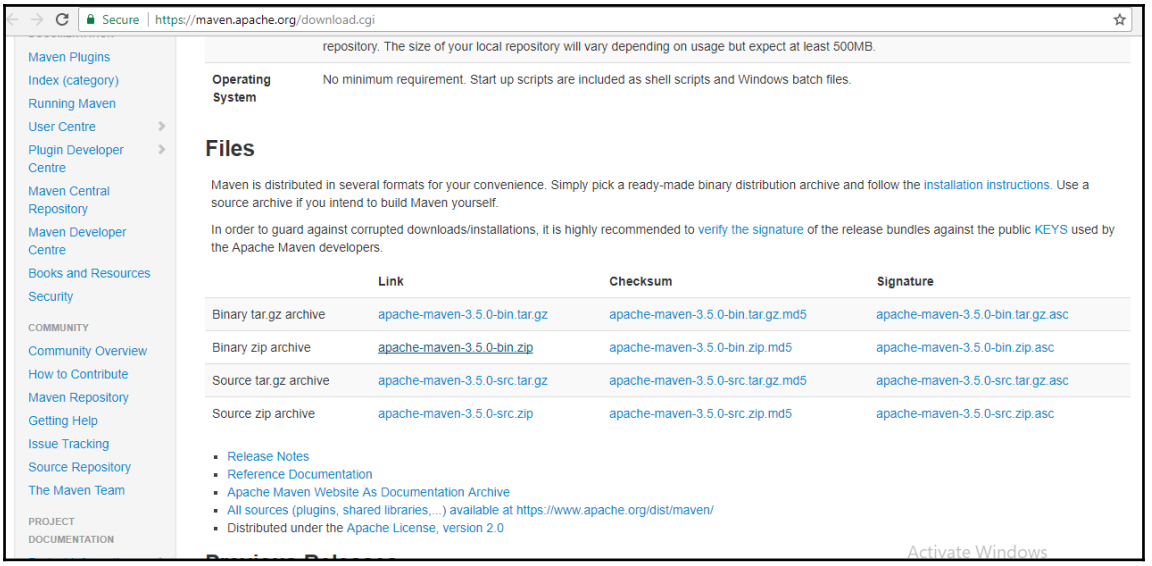
4. Select a job name and the next screen will show Git options as following, under the **Source Code Management** tab. You can add other SCM tools like CVS, subversion, and so on, in a similar manner:



5. Enter the Git repository address of the local machine or a web link in the preceding repository URL placeholder to configure Git with Jenkins.

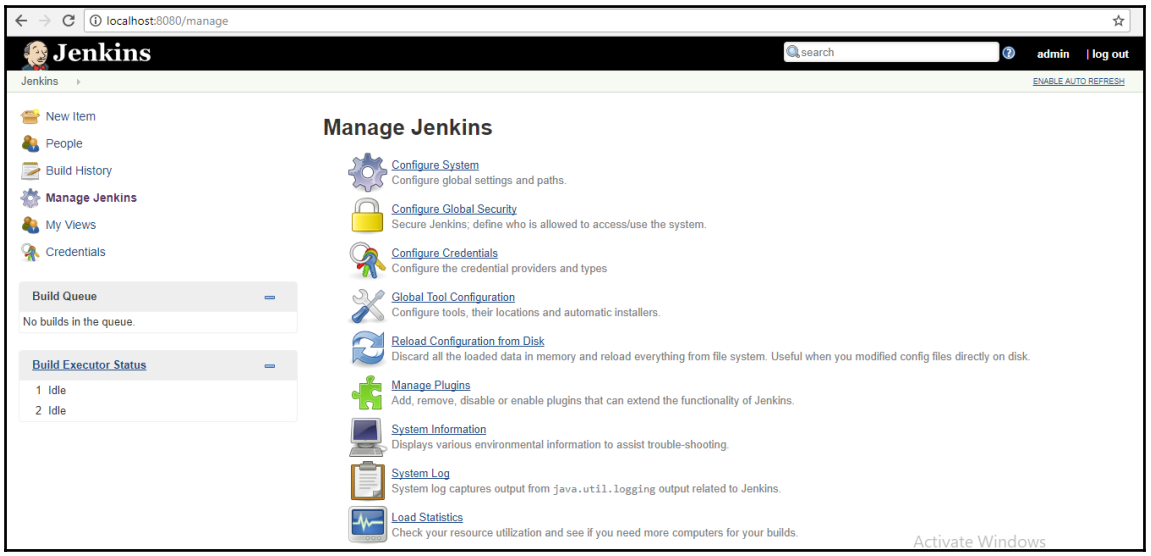
Maven (Build) tool Integration with Jenkins

1. Download Maven from `https://maven.apache.org/download.cgi`; this is the latest version of the binary file:

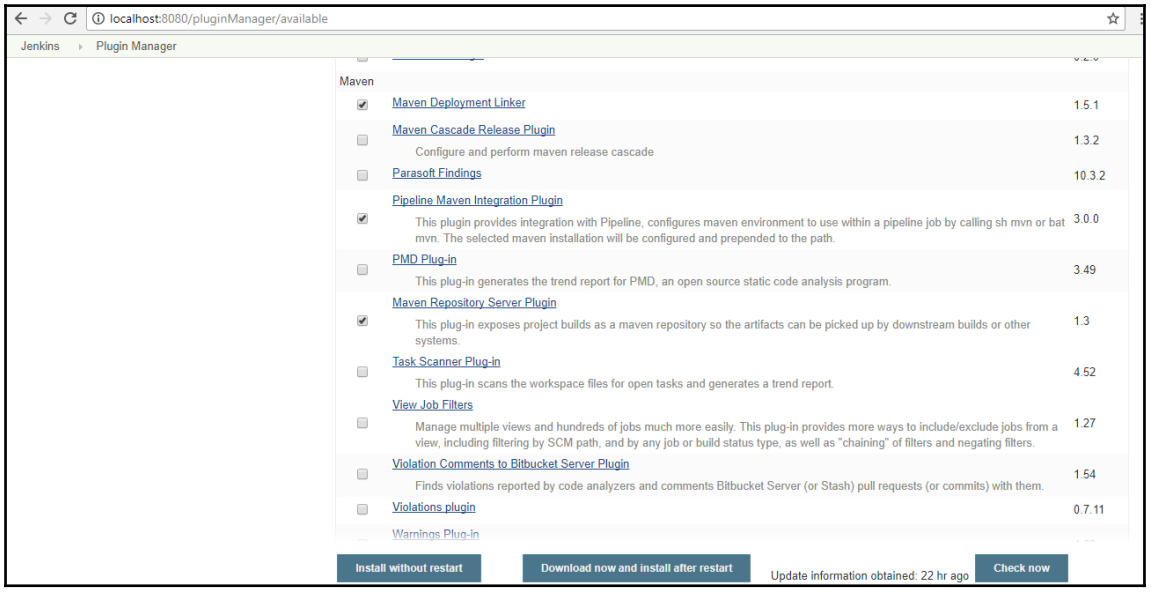


2. Extract the downloaded Maven file to a folder.

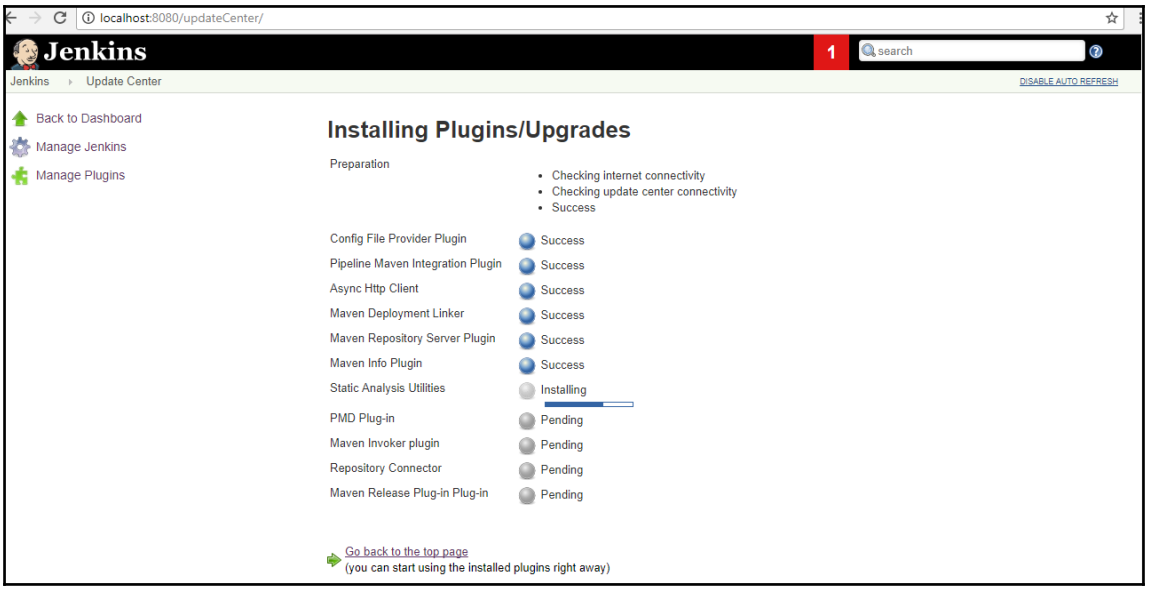
3. Open **Manage Jenkins**:



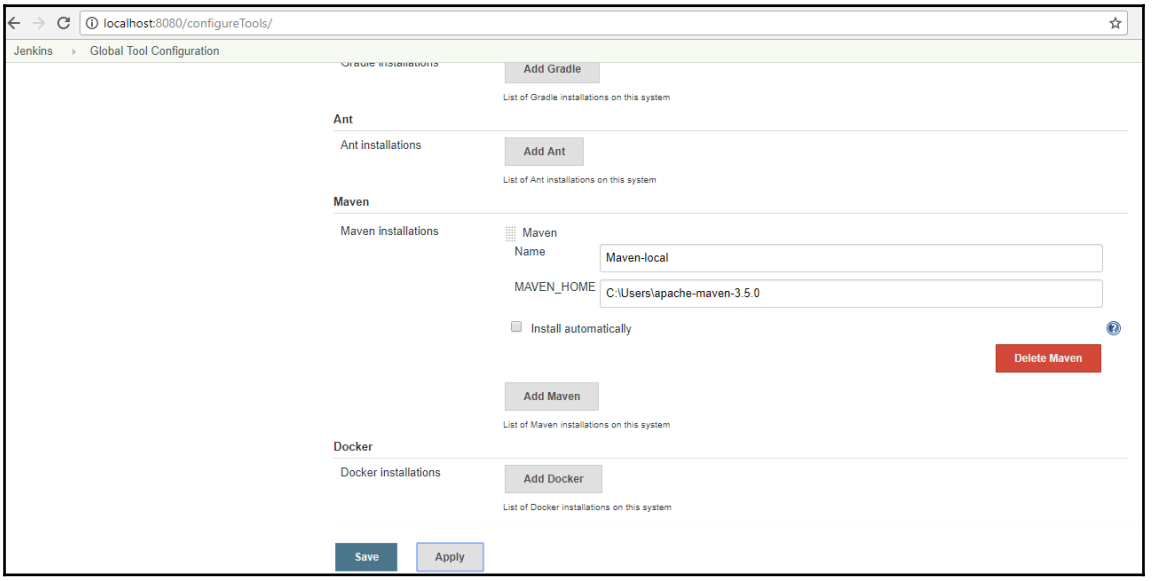
4. Select Maven Plugins as follows and install them without the restart option.



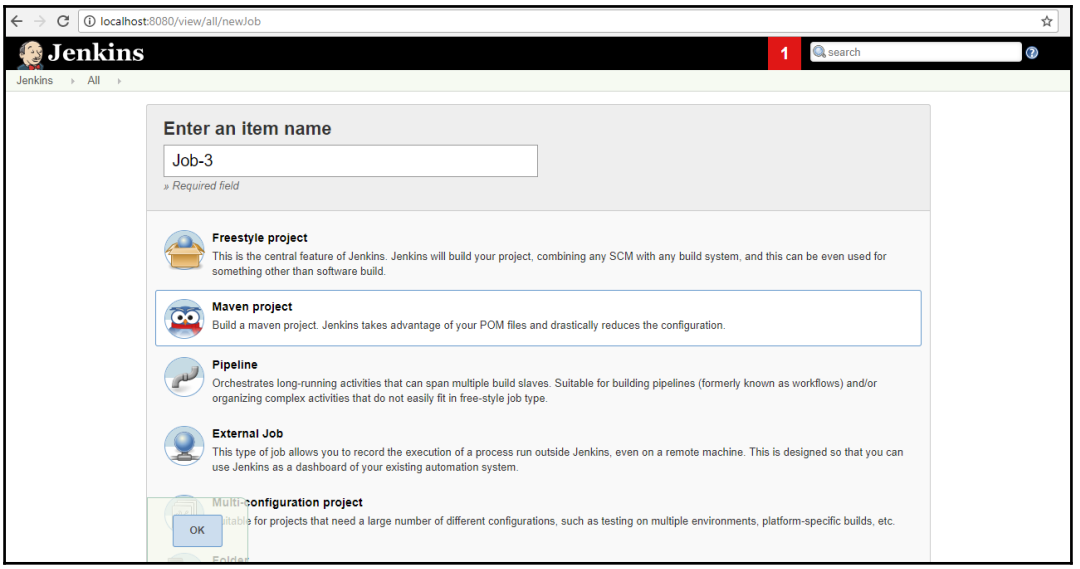
5. Monitor plugin progress as follows:



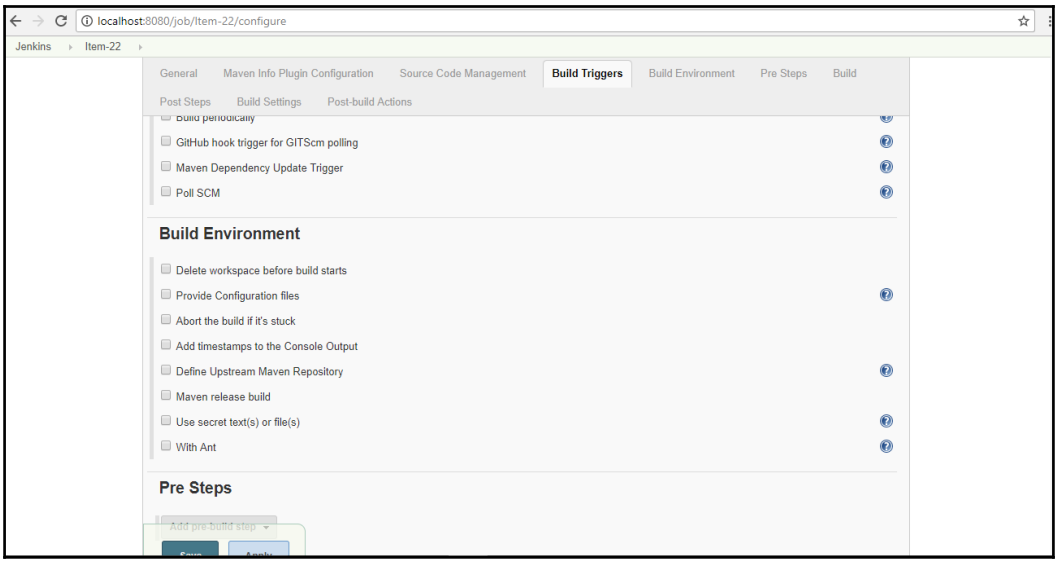
6. Under **Configure** tools, add Maven by giving the repository location:



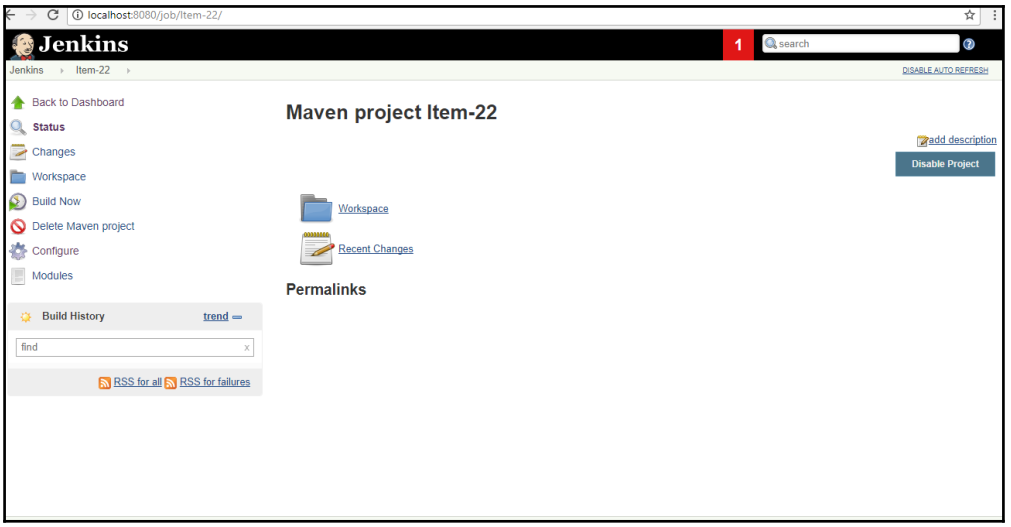
7. Create a new item job with the Maven project option:



8. The Maven option in build environment is as follows:

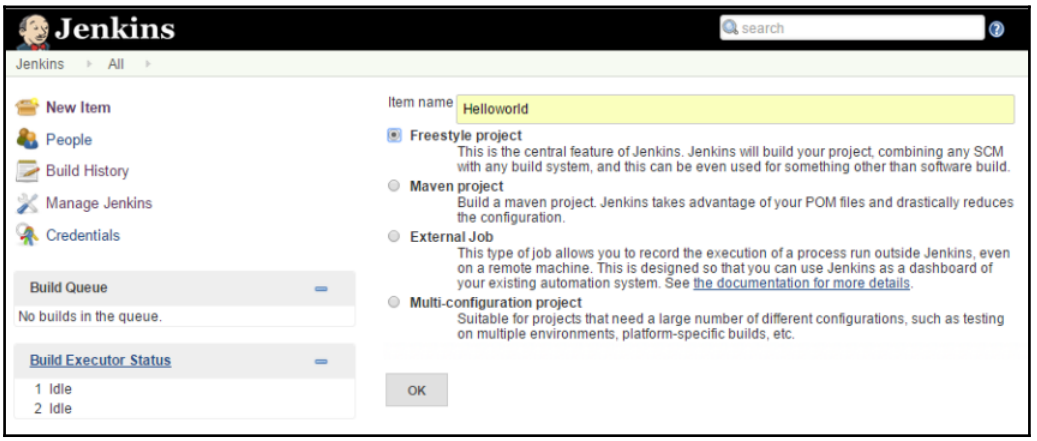


9. The project is created as follows:

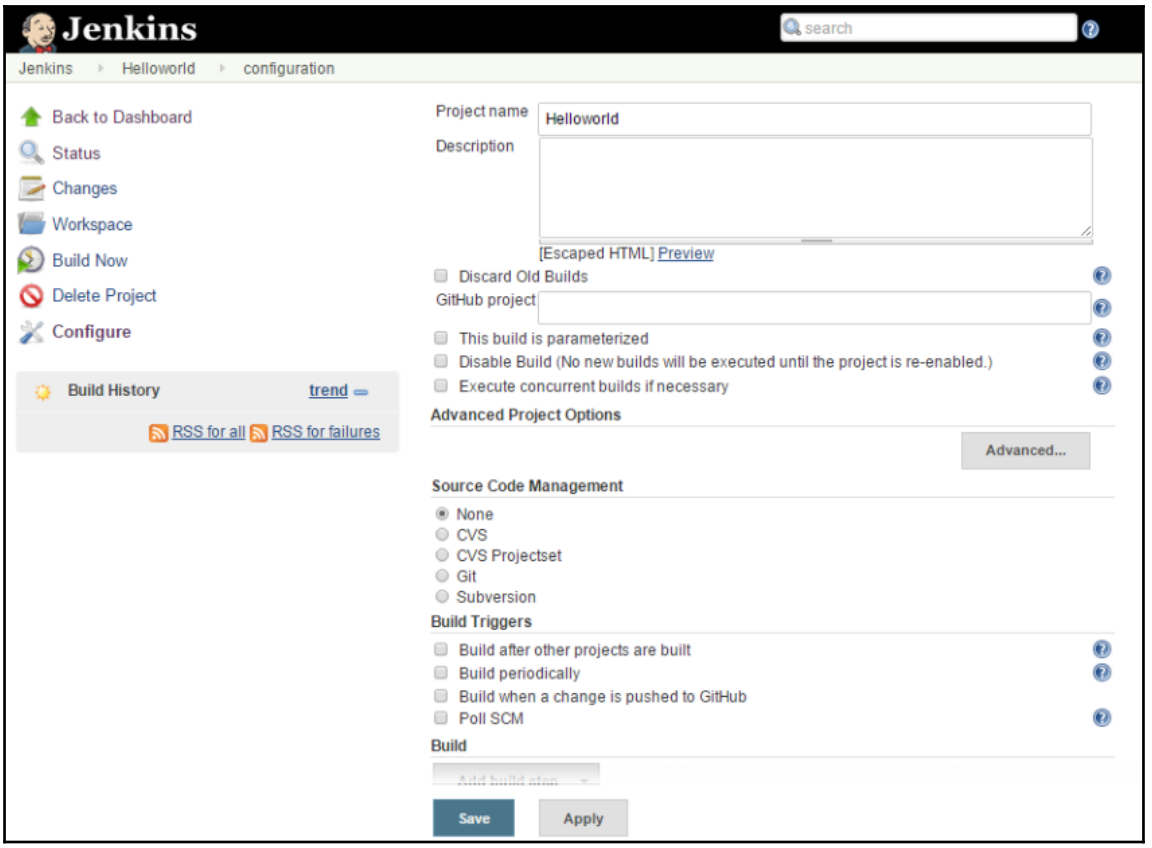


Building jobs with Jenkins

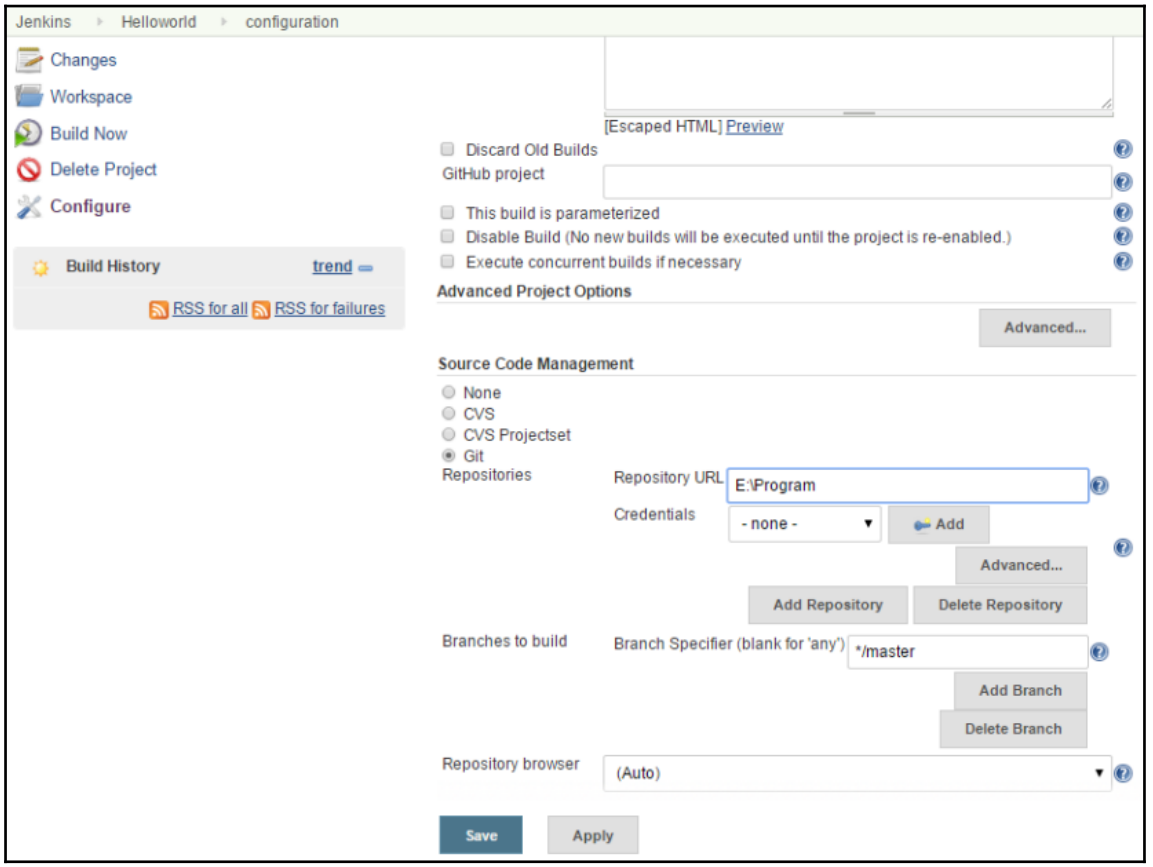
1. A simple application builds and runs the program:



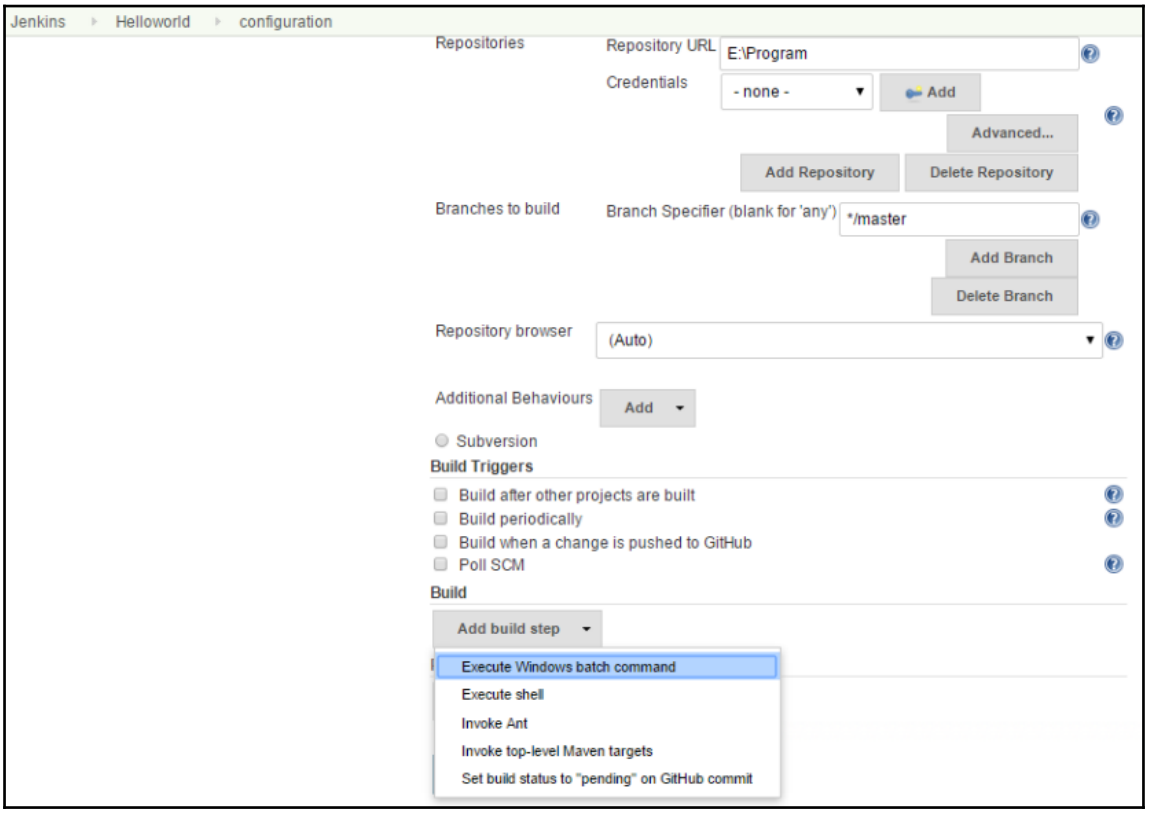
2. The source code repository options as listed as follows:



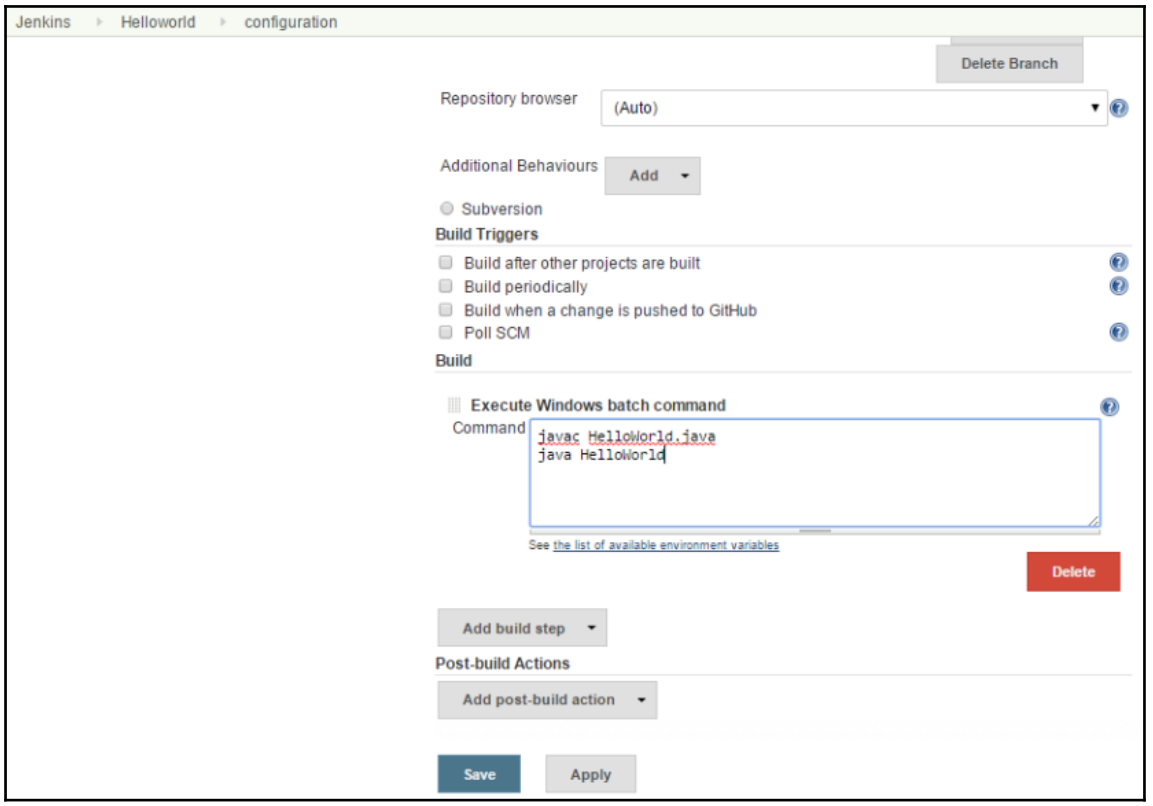
3. We can specify the location of files which need to be built either from a source Git code repository or the URL from GitHub:



4. Builds can be executed with multiple options, command modes, and Maven and so on:

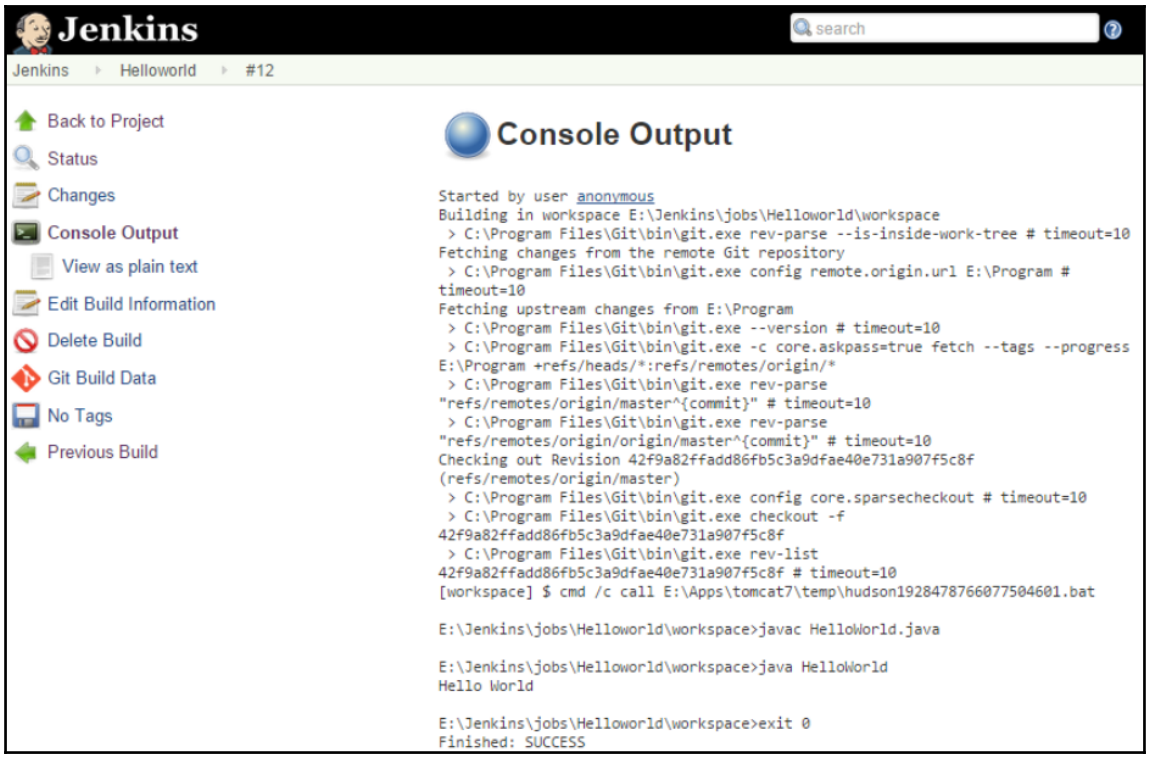


5. Command-line programs can be executed as follows:



6. After saving, the build option is visible, and history is also available:

7. Build progress can be seen and repository available as follows:

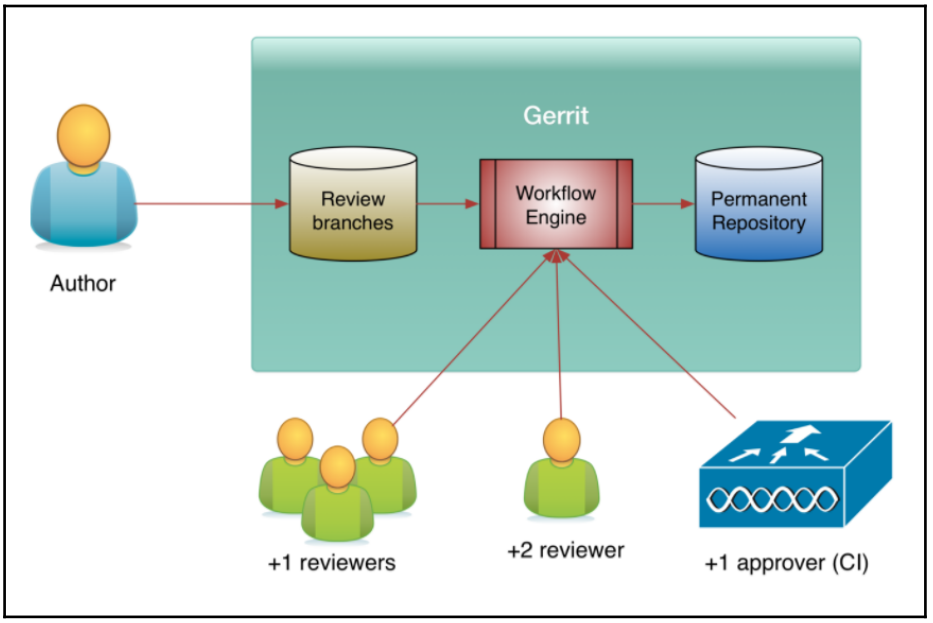


Source code review - Gerrit

Code review is an important function in the software development framework. Having a good collaborative tool like Gerrit for a code review process is very appropriate and needed. Gerrit initiates a pull-based workflow to initiate change requests, wherein comments are included even for source code to allow the change to be merged into the code repository through the workflow process. Gerrit maintains a local repository of the mirrored Git project repositories with reference repositories. Gerrit creates another maintenance branch from master branch to track reviews to the code; it creates a change-id identifier for the commit message to keep track of each change in a code review.

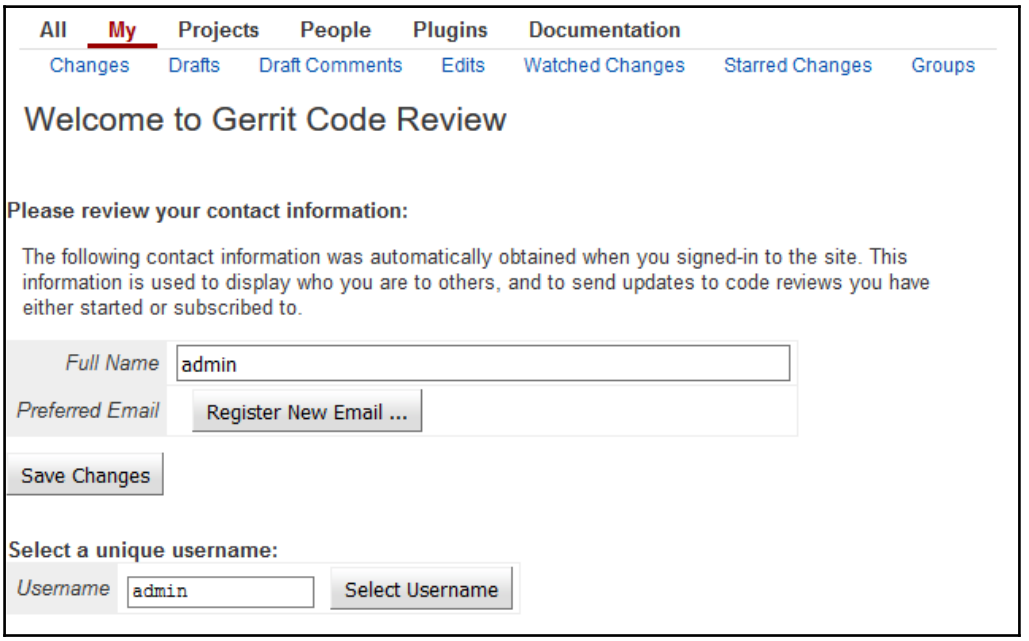
Gerrit allows for code change comparisons and a reviewer can give one of five ratings:

- **+2**: Looks good, approved
- **+1**: Looks good, but needs additional approval
- **0**: No comments
- **-1**: Suggest not submit this
- **-2**: Block the submit

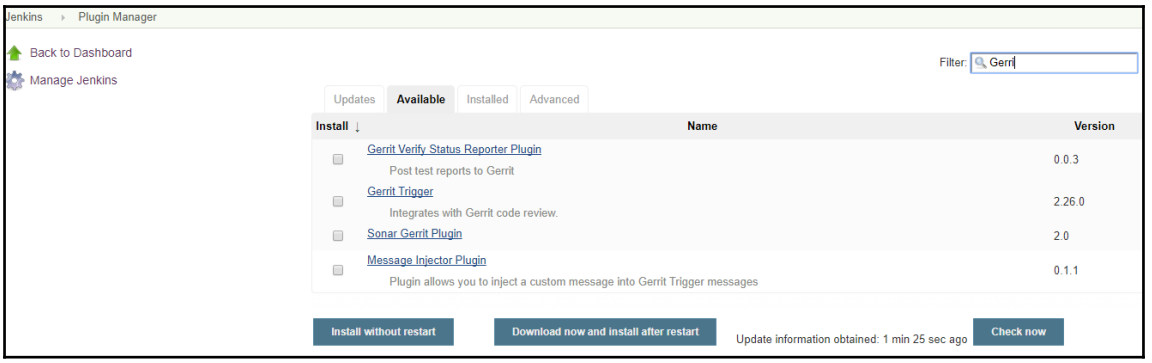


Installation of Gerrit

1. Download Gerrit from `https://www.gerritcodereview.com/`.
2. Follow the installation instructions as per the platform option and access Gerrit on port `8080` as follows to create users and projects:



3. Configure in Jenkins under **Manage Plugins** for Gerrit:



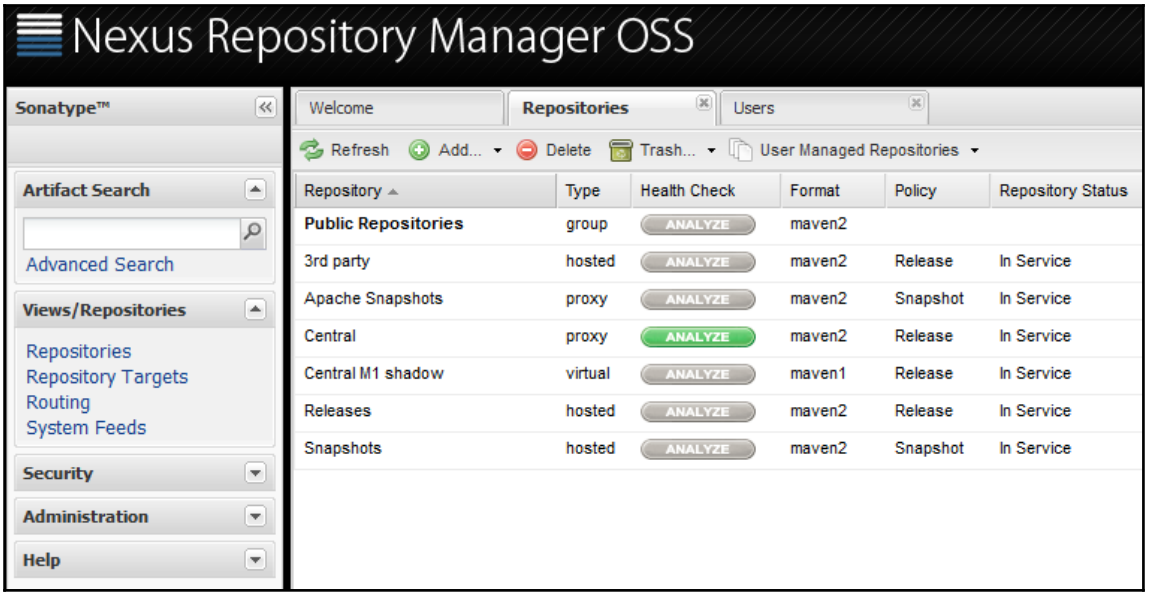
Version control tools listed in the `Chapter 3`, *DevOps Framework*, for example Gerrit the web-based code review interface, allow reviewing changes online to push changes from any git client and then auto-merging them with the master; it can also be configured as a remote git repository.

Gerrit configuration includes user creation, **Secure Shell** (**SSH**) set up to exchange data with a Gerrit server. The configuration file `/etc/gerrit.config` has extensive parameters you need to set as per configuration requirements.

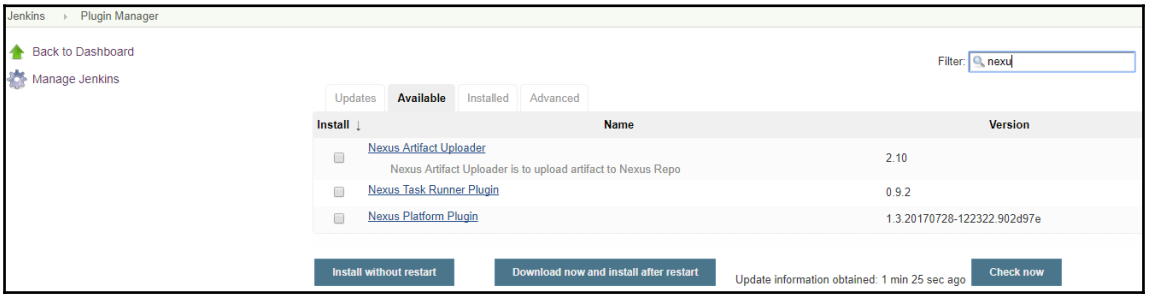
Repository management

Maintaining multiple build version artifacts is the key feature of repository management and Nexus is a popular repository manager. It can be downloaded from `http://www.sonatype.org/nexus/downloads/`.

After installation, it can be accessed from `http://<nexus host>:8081/nexus`:

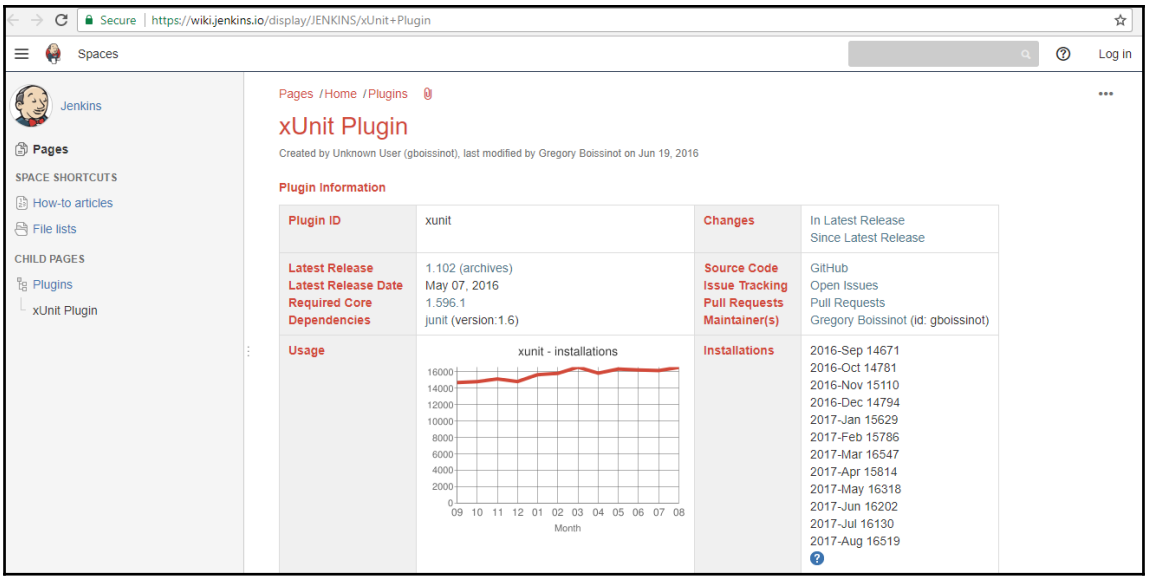


Nexus can be configured with plugins for Jenkins integration:



Testing with Jenkins

Jenkins provides many out-of-the-box functionalities and plugins for testing. The site `https://wiki.jenkins.io/display/JENKINS/xUnit+Plugin` provides the plugins:



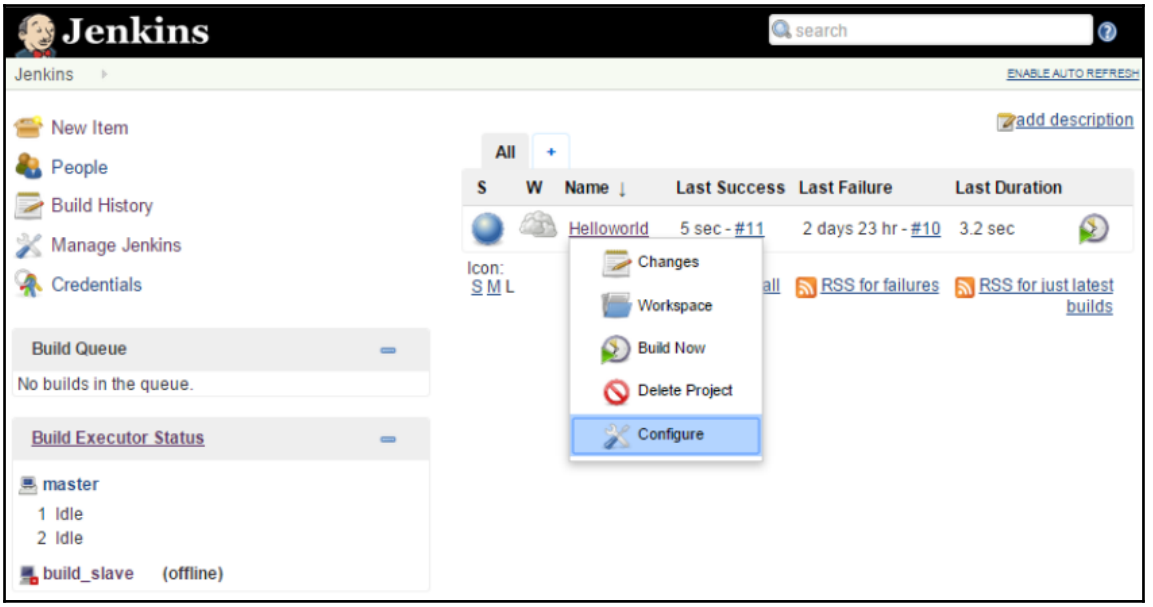
A list of available testing plugins is shown as follows:

- JUnit itself
- AUnit
- MSTest (imported from MSTest Plugin)
- NUnit (imported from NUnit Plugin)
- UnitTest++
- Boost Test Library
- PHPUnit
- Free Pascal Unit
- CppUnit
- MbUnit

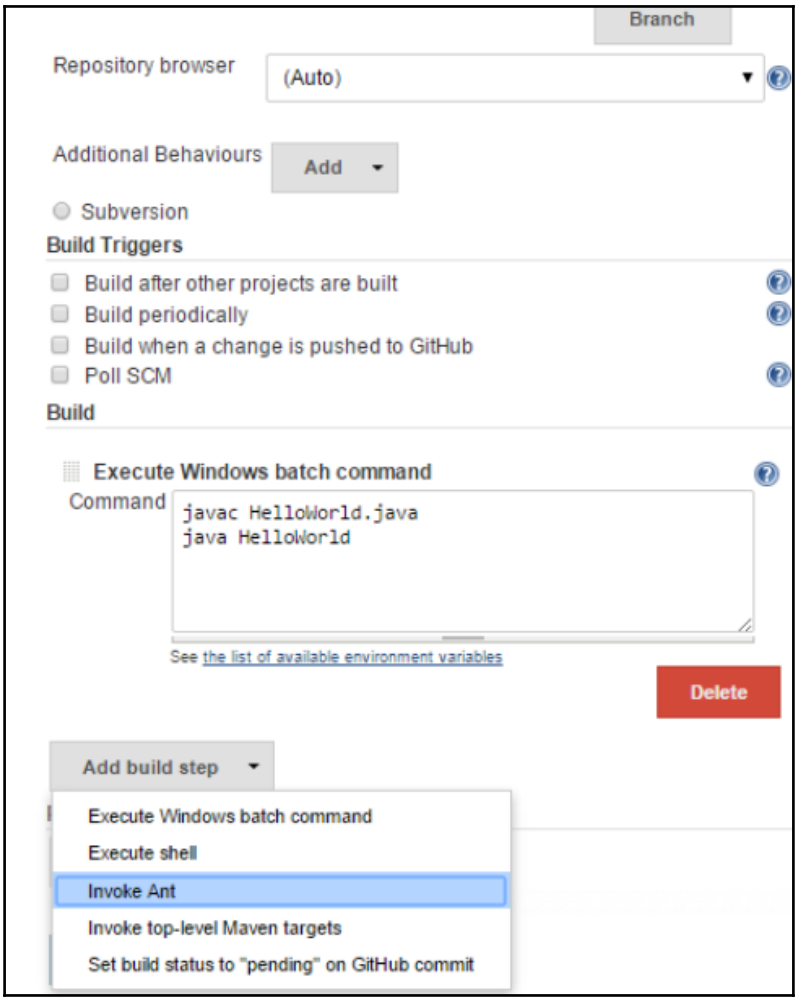
- Google test
- EmbUnit
- gtester/glib
- QTestLib

Setting up unit testing

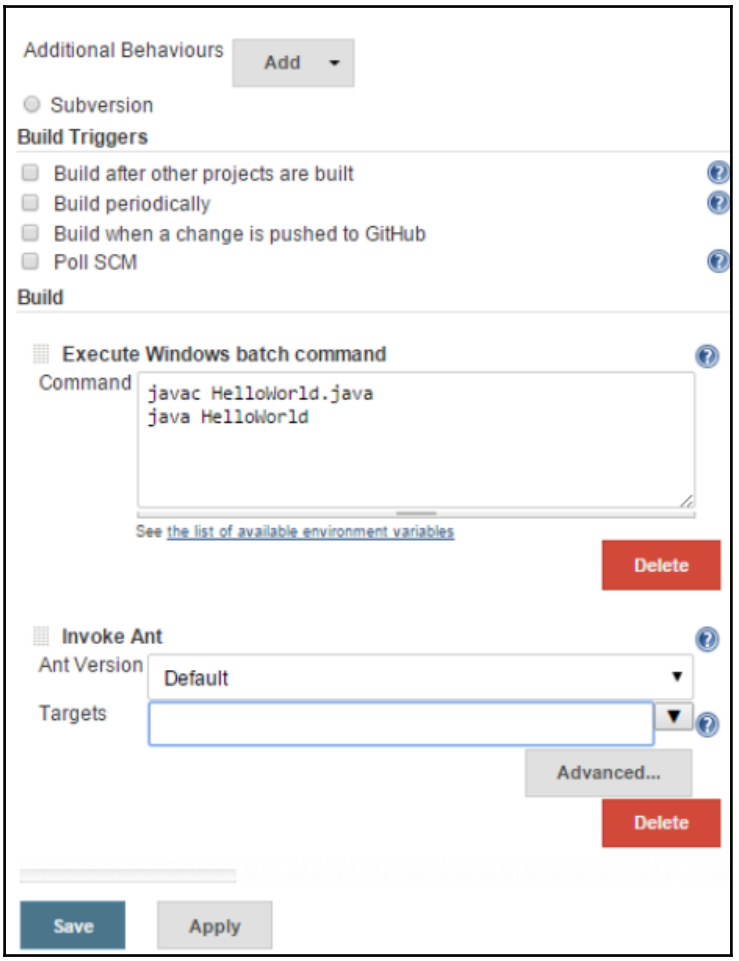
1. Pick up the project we have set up:



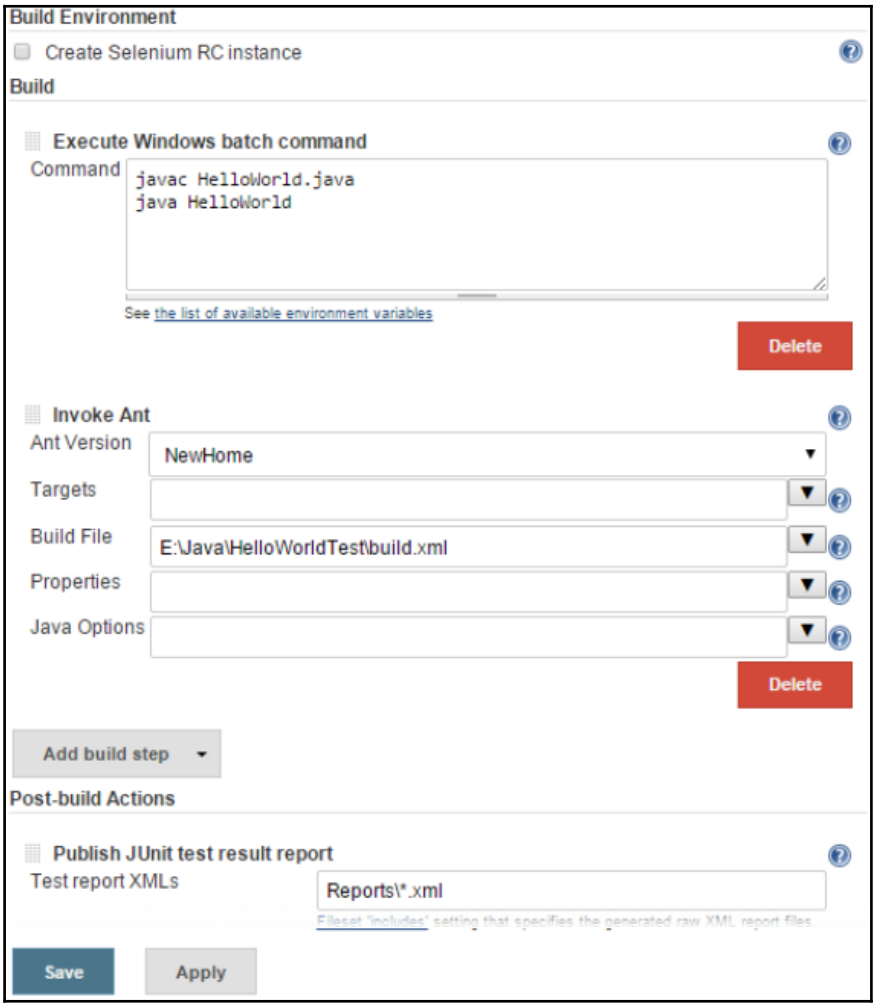
2. Choose build option:



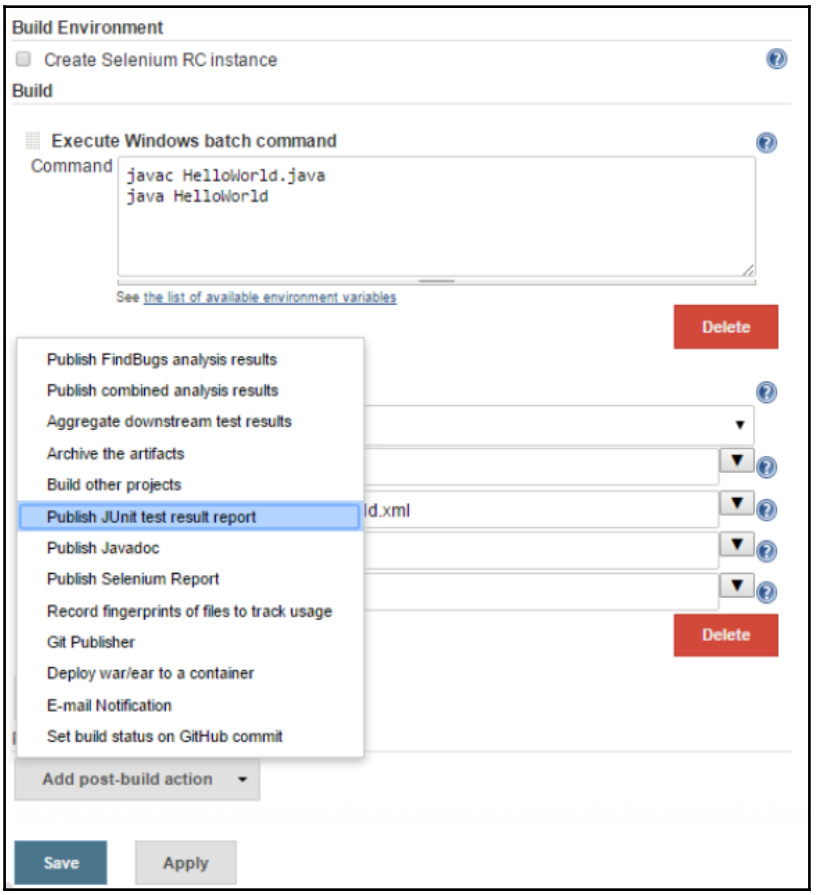
3. Choose an **Advanced** option:



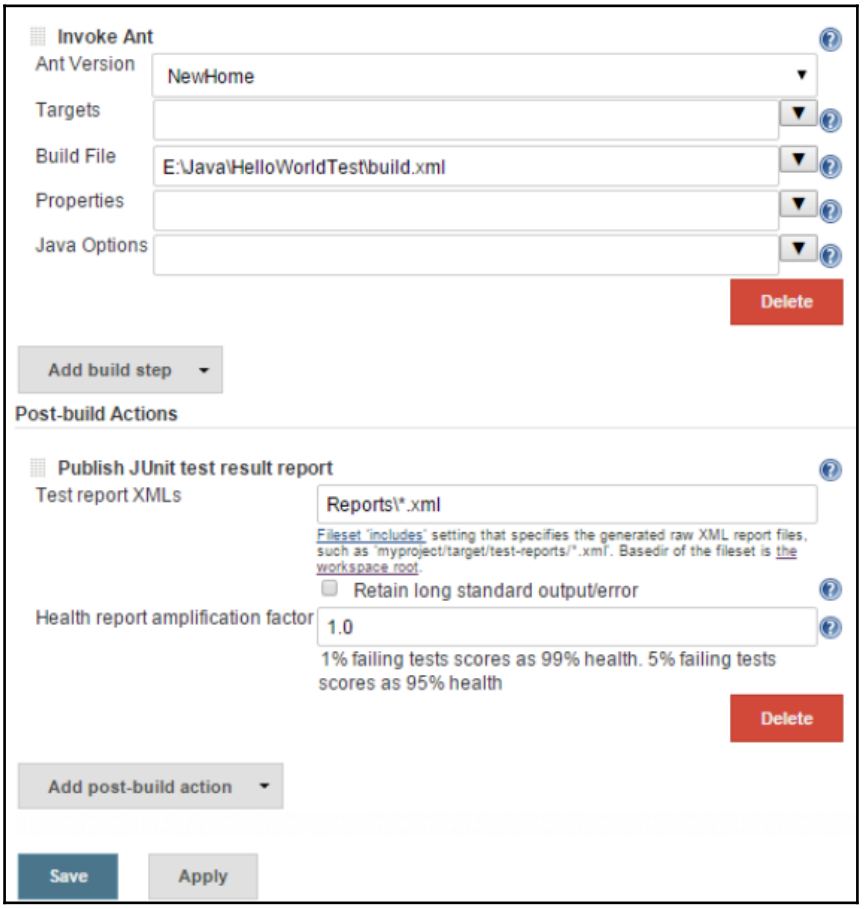
4. Enter the location of `build.xml`:



5. Select the option of post-build option and choose - `Publish JUnit test result report`:



6. In the test `reports.xml`, enter the reports created a folder in our project so that Jenkins picks the resulting XML files produced by running of the JUnit test cases:

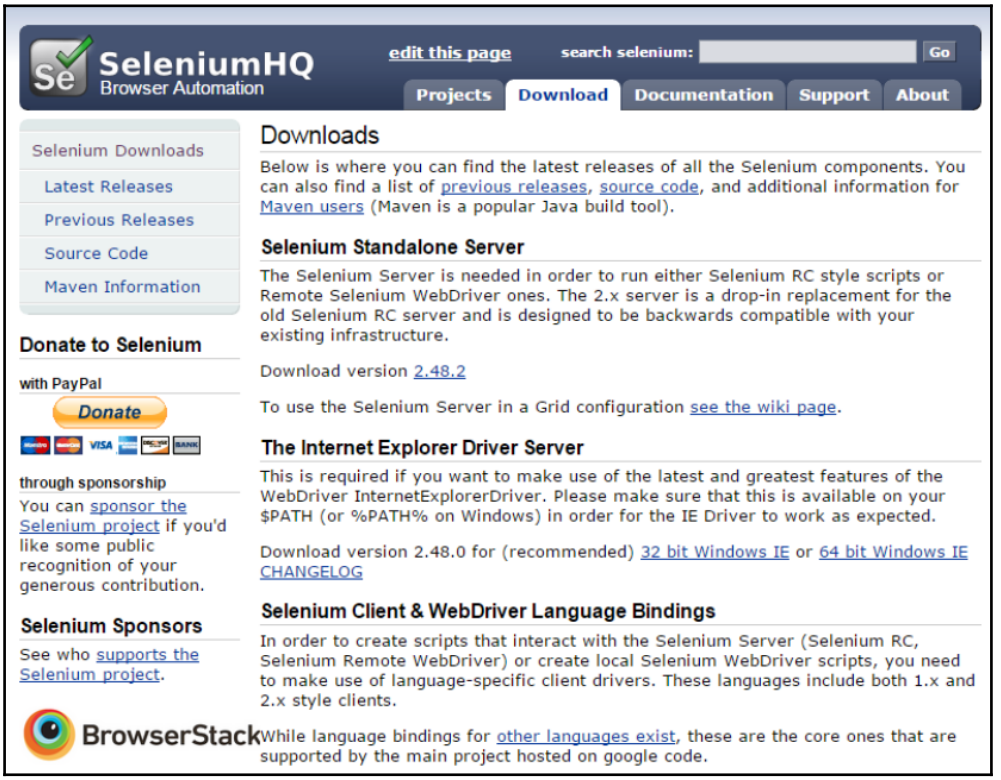


We can select the build and drill-down to the test results.

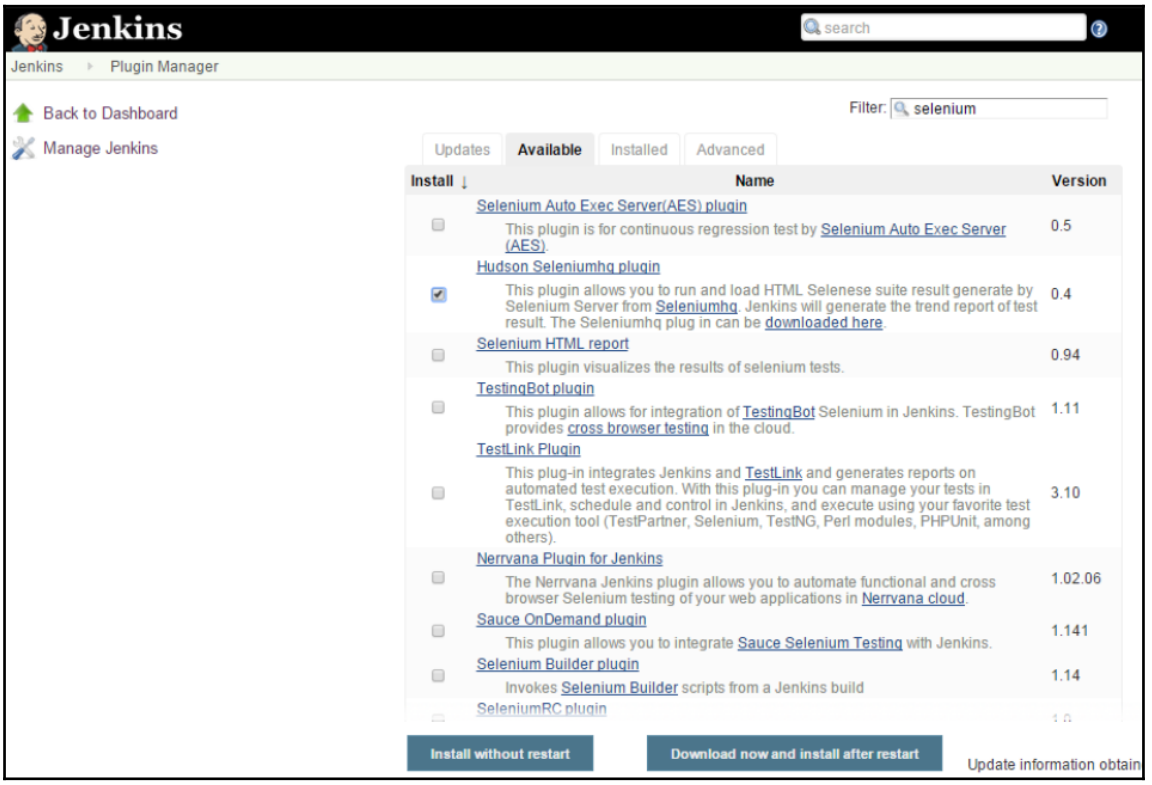
Automated test suite

Continuous integration is the process of verifying a build to objectively access its readiness for the next level; this is accomplished with automated testing. So, the build artifacts are set to be tested automatically; Selenium is the most popular framework for this.

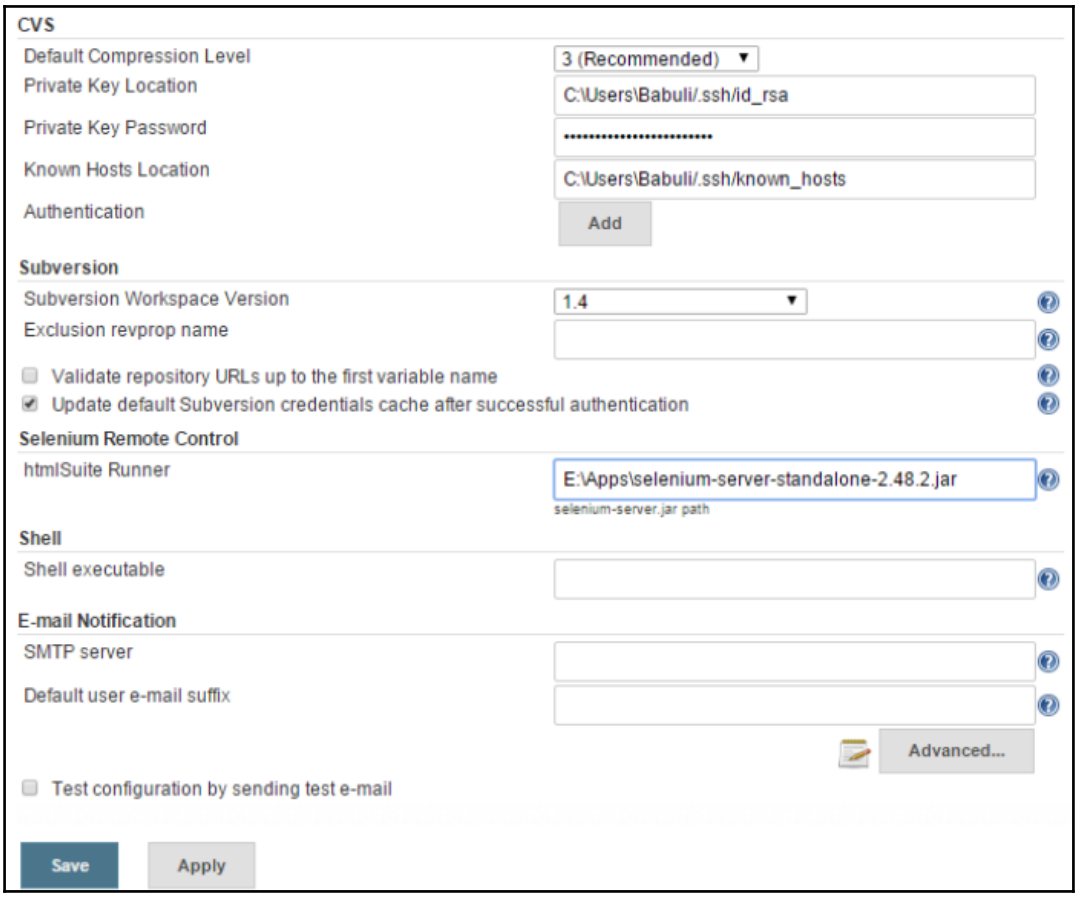
It can be downloaded from the following site:



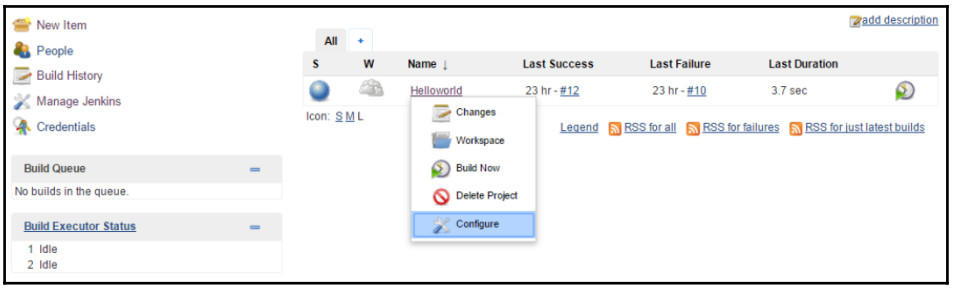
1. Under **Jenkins**, **Plugin Manager**, select the Selenium plugin and install, restart to initiate:



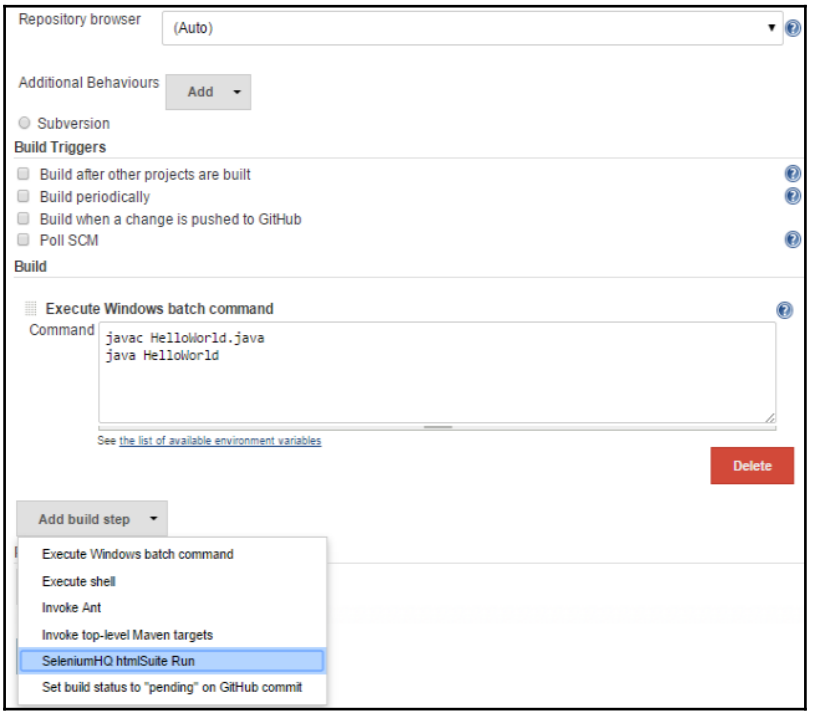
2. Configure the selenium server JAR file:



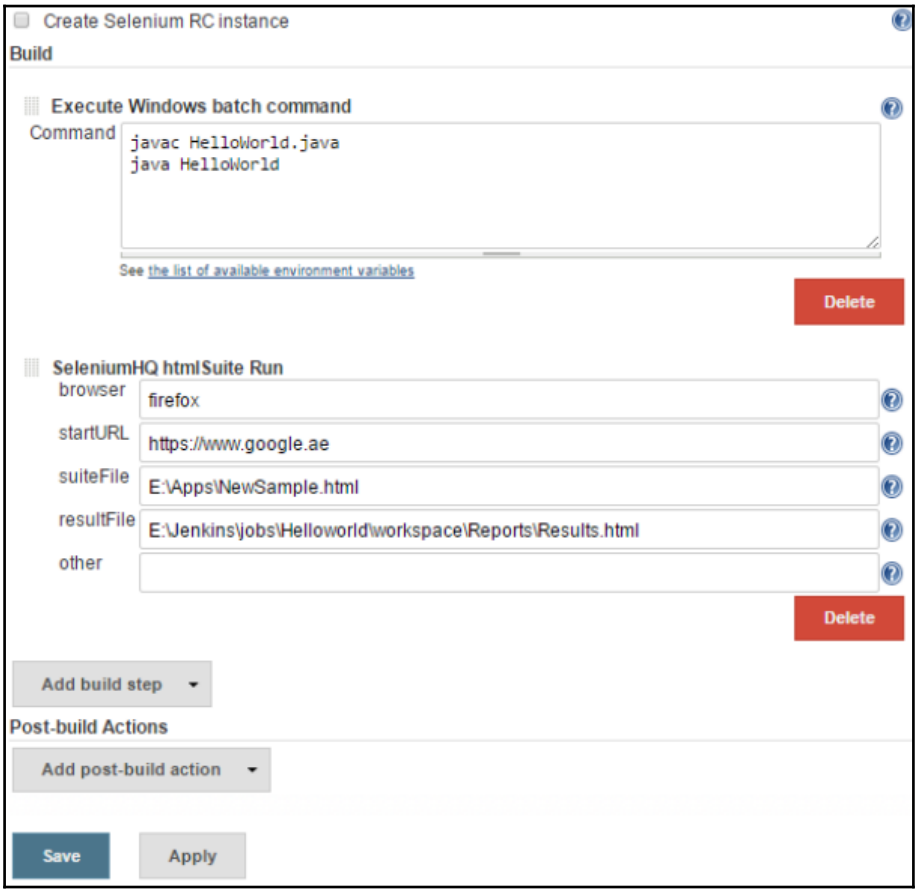
3. Configure the project we created to be set for this automated framework:



4. In the build process, add the option, `SeleniumHQhtmlSuite Run`:

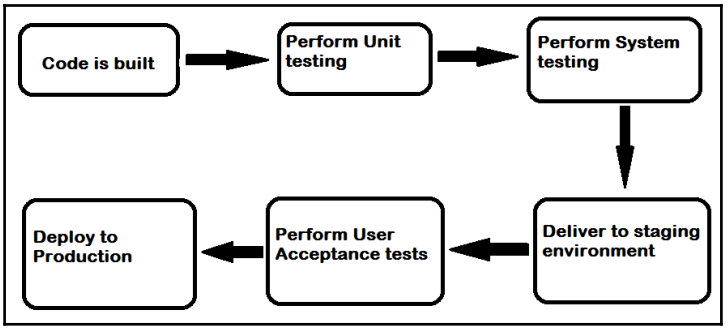


5. Selenium IDE will generate TestSuite, the Selenium test is enabled with SuiteFile by launching the selenium driver:

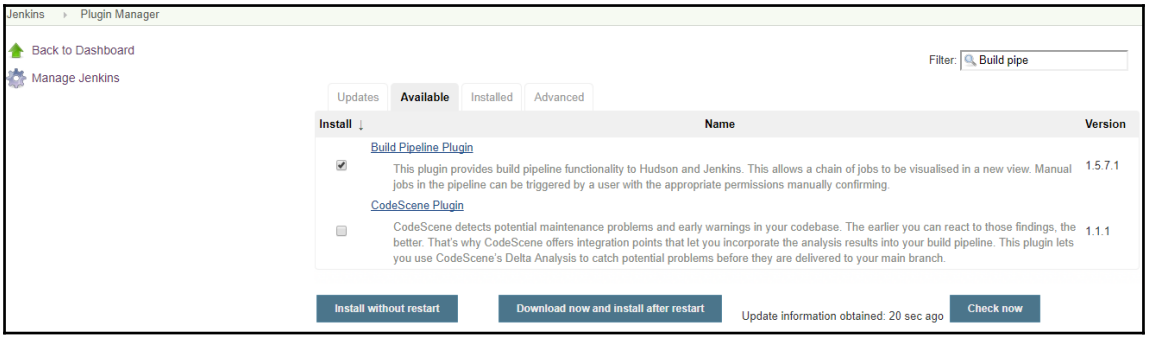


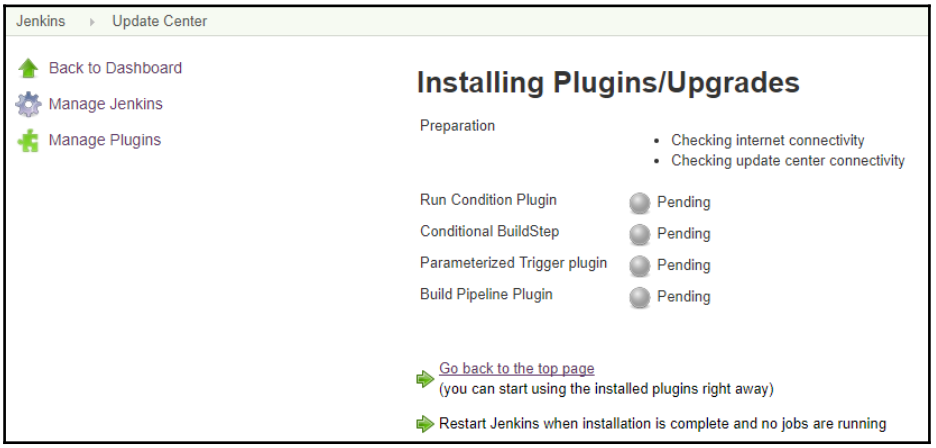
Continuous delivery- Build Pipeline

Continuous delivery is the process of building a robust pipeline from software development to deployment.

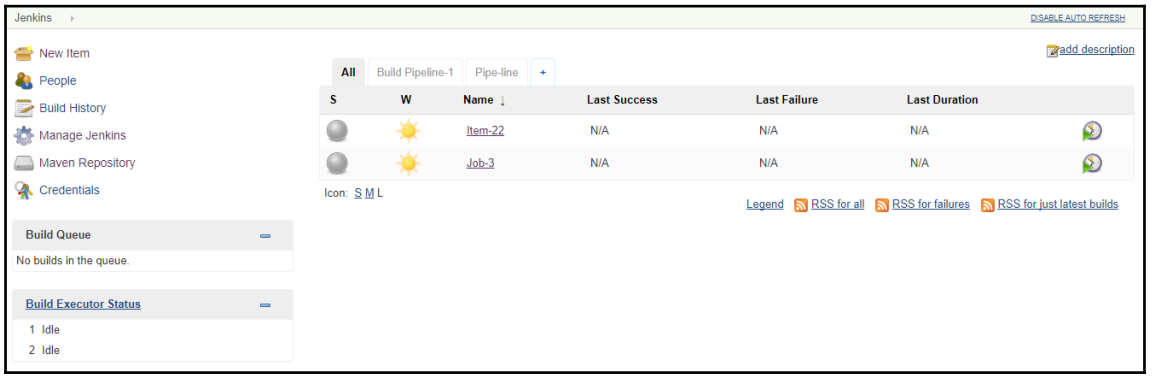


1. Install the Build Pipeline plugin from **Manage Plugins** as follows:

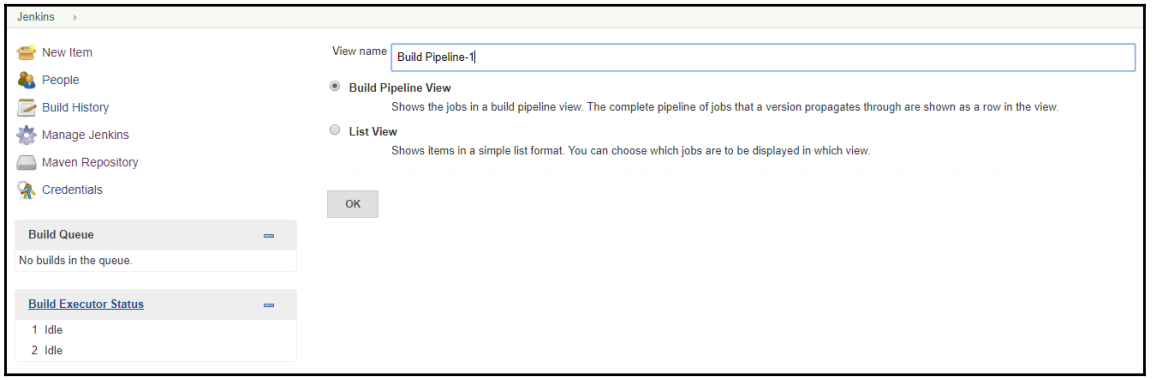




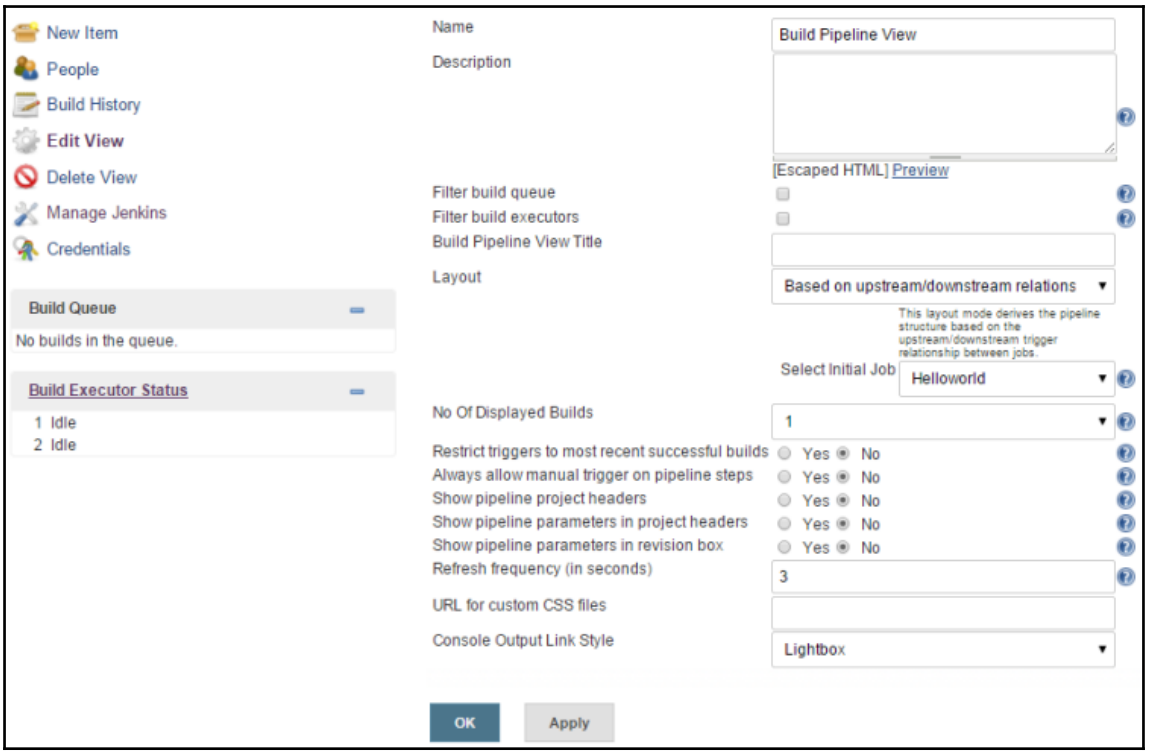
2. To set up the Build Pipeline, Click on the + symbol, next to the **All** tab on the dashboard:



3. Select **Build Pipeline View** and choose a name for the pipeline:



4. Select the **Options** and the project created:



5. The delivery pipeline view is created with the status of each stage of the project.

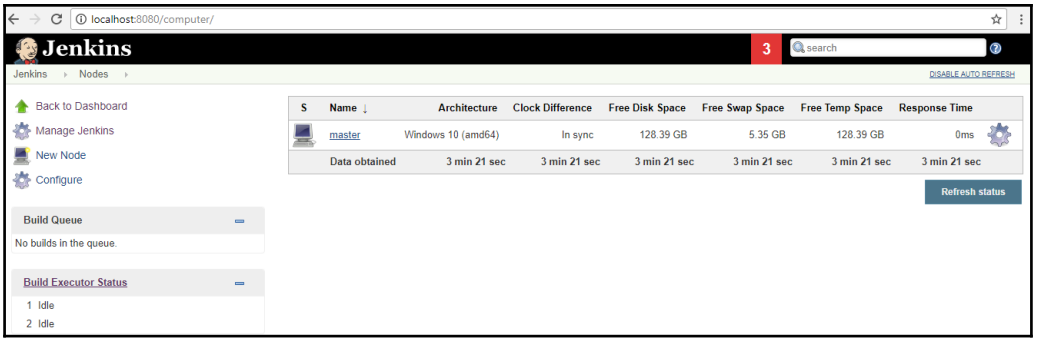
Jenkins features

- Client-server
- Security
- Reporting

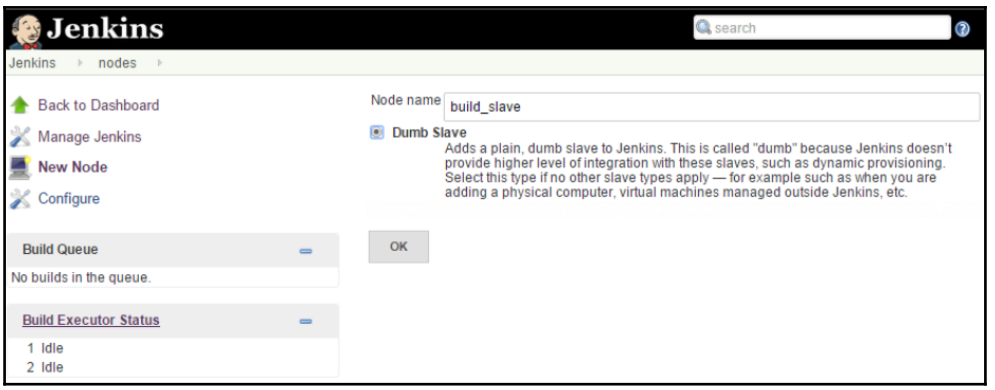
Larger projects need multiple machines to be configured instead of centralized builds on one machine. Also, there are requirements for several different environments for test builds. Slave machines are effective to offload these loads from a master server.

They need a bi-directional communication link from the master through a TCP/IP socket, with only a slave agent instead of the full Jenkins package or compiled binaries.

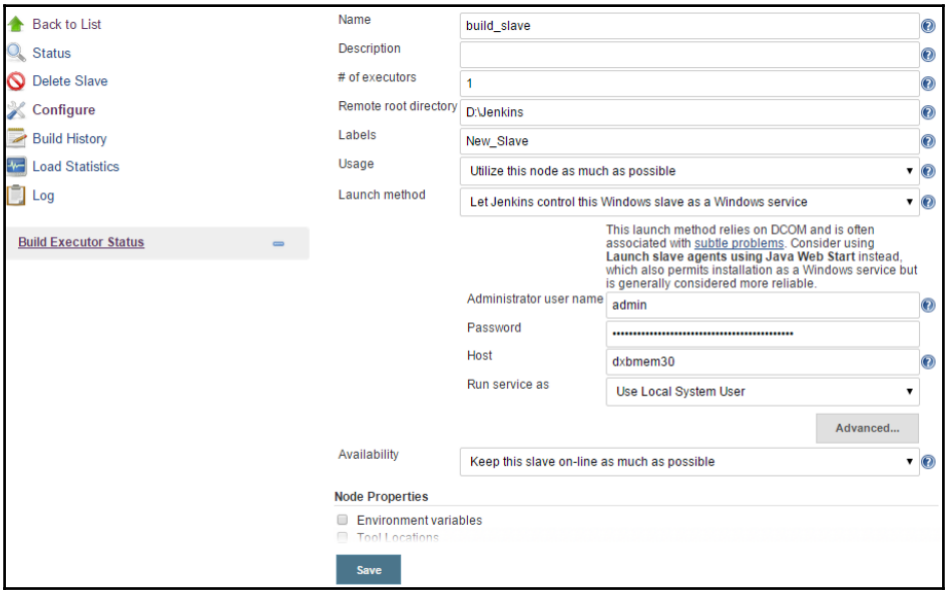
1. To set up slave/nodes under Jenkins, configure and select the manage nodes option and create a new node:



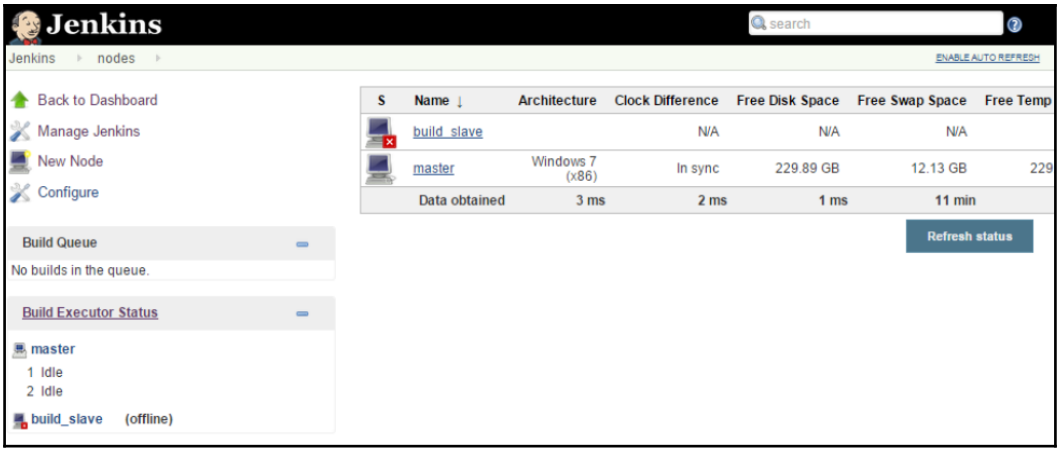
2. Select name and **Dumb Slave** option.



3. The slave node details are to be given, then choose to let Jenkins consider the Windows slave as a Windows service. Details such as name node and login credentials of the machine are required.



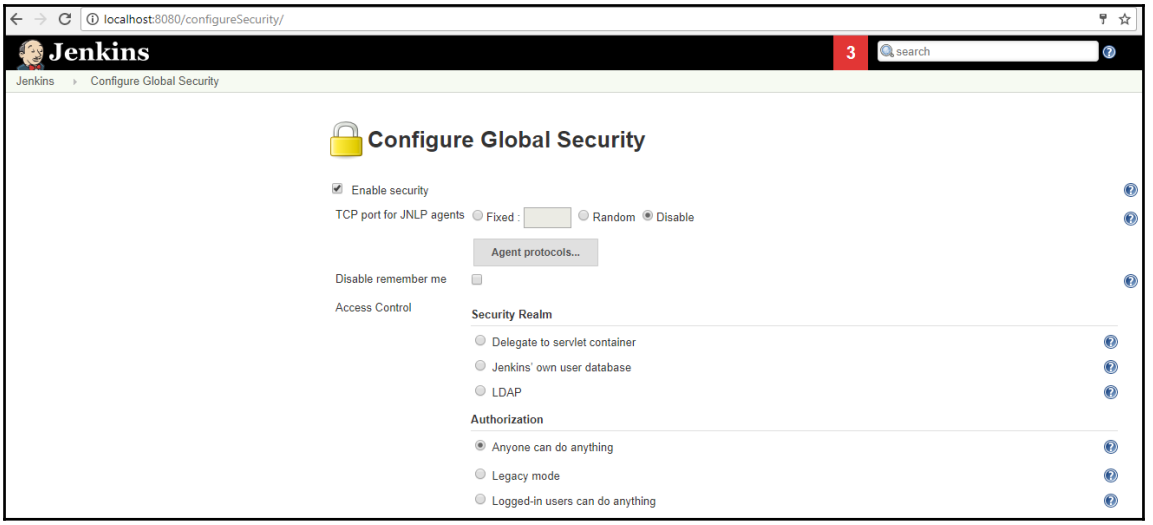
4. The slave machine will be available as follows; new jobs can be configured to run on this slave machine.



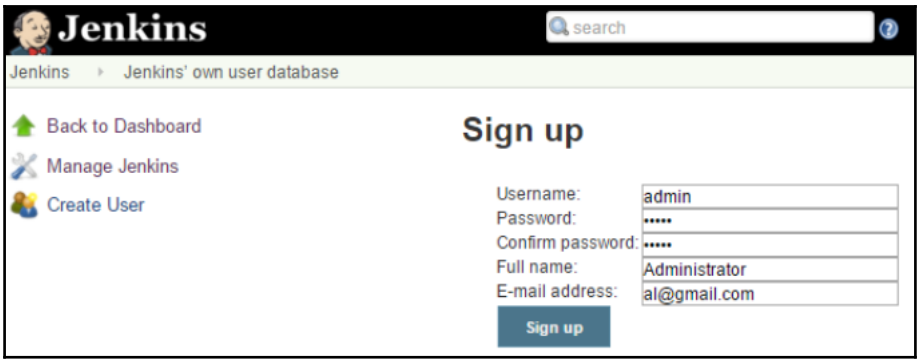
Security in Jenkins

Users with relevant permissions can be set up with security configuration:

1. Under **Manage Jenkins**, select **Configure Global Security**, and choose to **Enable security** option:

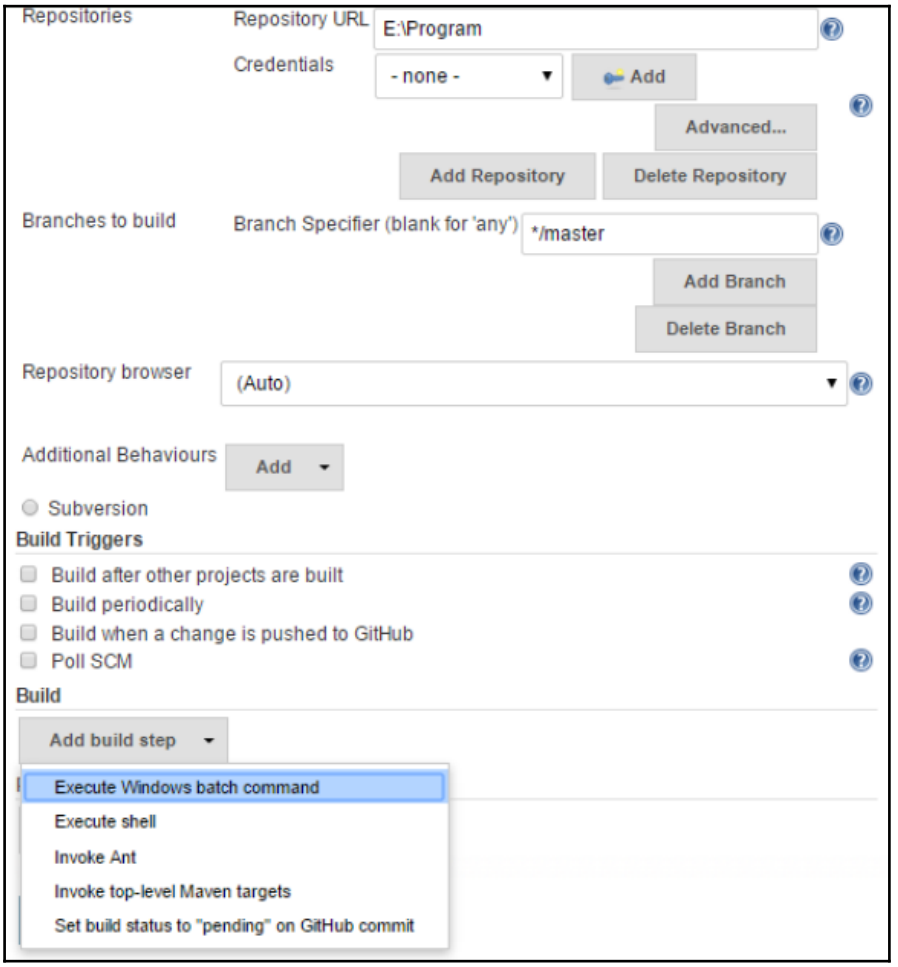


2. Once you save the options, you will be prompted for an admin user.



3. Under **Jenkins Manage** setup, choose **Manage Users Options** to create users and then set up authorizations required to execute jobs with matrix based security:

4. The **Reporting Options**, **Metrics Options**, and **Reporting Plugins** can be installed.



5. Many Metrics are available such as the Build History Metrics Plugin:
    - **Mean Time To Failure** (**MTTF**)
    - **Mean Time To Recovery** (**MTTR**)
    - Standard deviation of build times

MITTR	Last 7 days	0 ms
	Last 30 days	23 hr
	All Time	23 hr
MITTF	Last 7 days	0 ms
	Last 30 days	2 days 4 hr
	All Time	2 days 4 hr
Standard Deviation	Last 7 days	0 ms
	Last 30 days	52 sec
	All Time	52 sec

6. It can be installed under **Manage Plugins** choosing the **Build History Metrics Plugin**, the above metrics will be reflected on the job page.
7. To see a graphical representation, use Hudson global-build-stats and **Global Build Stats** plugins under **Manage Plugins**. Setting the options, initialize stats, create new chart options, and all the existing builds records will be displayed.

Summary

In this chapter, we learnt about processes and tools for implementing continuous development, continuous integration, and continuous deployment with the use of repository management, code reviews, and test automation.

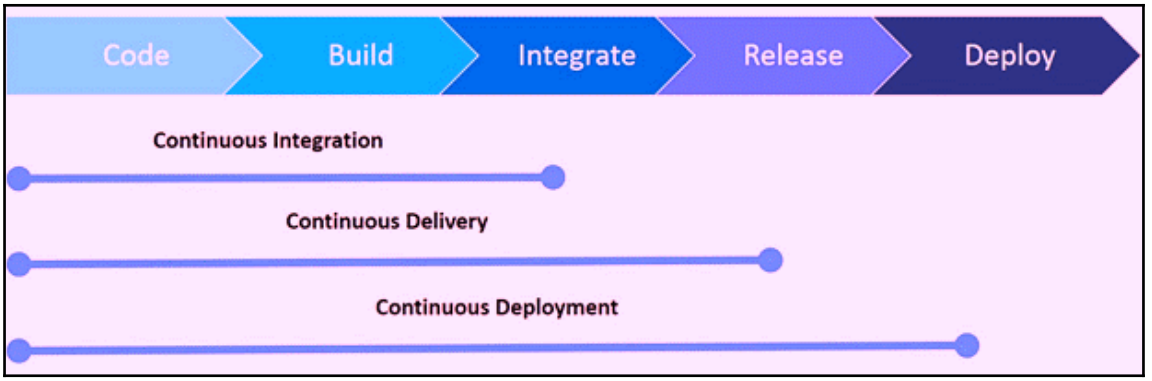
In the next chapter, we will cover the topics of infrastructure configuration management as code for continuous deployment with tools such as Chef, Puppet, and Ansible. We will discuss on continuous monitoring process with tools Splunk and Nagios.

8
DevOps Continuous Deployment

DevOps continuous deployment enables changes to be ported quickly from development to production. Infrastructure and automation play key roles for enabling continuous deployment. In this chapter, we will learn about configuration automation and the implementation of infrastructure automation (Infrastructure as Code) with tools such as Chef and Ansible. We will also discuss the continuous monitoring process with the tools, Splunk and Nagios:

- Continuous deployment
- Chef
    - Components
    - Terminology
    - Architecture
- Ansible
    - Components
    - Terminology
    - Architecture
- Continuous Monitoring
- Splunk
- Nagios

As we have discussed in the previous chapters, the following figure shows the process of continuous integration, continuous deployment, and continuous delivery alignment.



Continuous Integration (**CI**) is the process of making the development, unit test and build process on a continuous mode as opposed to staggered (step-by-step) methodology. In the CI process, every developer merges their code changes to a central version control system, each commit triggers automated build. So the latest versions are always available in the code repository and also built executable is from latest code.

Continuous Delivery (**CD**) is a next step to the continuous integration process for software engineering to produce software in short cycles of testing, and releasing software faster and more frequently. The automated testing process ensures that the software can be reliably released at any time.

Continuous deployment is the process to minimize lead time (the elapsed time) between the development of new code and its availability in production for usage. To accomplish this, continuous deployment relies on infrastructure that automates various steps, after each successful code integration meeting the release criteria, leading to deployment, the live application is updated with new code.

Traditionally, a new machine is built by administrators and system engineers from documentation and customized scripts, and so on. Managing Infrastructure through manual procedures such as custom scripts, golden image configurations, and so on, are both time consuming and error-prone. Organizations looking for faster and matured deployments and concepts adopt infrastructure configuration automation, which means managing Infrastructure like a software code for reproducible results, hence it's also termed as **Infrastructure as Code**.

Just like the SDLC process, infrastructure can be managed with similar tools and processes such as version control, continuous integration, code review, and automated testing extended to make configuration changes for infrastructure robust and automated. The infrastructure code and configuration changes are consistently tested, shared, and promoted across all the environments from development to QA test systems and to production more easily, rapidly, safely, and reliably with the detailed audit trail of changes. With infrastructure code as a service, the configuration of the new machines to the desired state can be written as a code to set up multiple machines at the same time. This scalable model is more effective by leveraging the elasticity of the cloud. Adopting DevOps to Infrastructure as Code str, and so on goes beyond simple infrastructure automation to extend multiple benefits as below:

- Ensure error-free automation scripts are repeatable
- To be redeployed on multiple servers
- Ability to roll back in case of issues
- Infrastructure code testing standards such as unit testing, functional testing, and integration testing can be effectively enforced
- Since the documented state of the machine is maintained as code and made up-to-date, written documentation is avoided
- Enable collaboration between dev and ops around infrastructure configuration and provisioning, infrastructure code as change management

DEV			PROD			
Dev	Build	Test	Deploy	Provision	Monitor	Operate
· Ant · Eclipse · Git · GitHub · Gradle · IDE · IntelliJ · JIRA · JUnit · Maven · MSBuild · Nant · NUnit · Subversion · TeamCity · Visual Studio	· Ant · AWS CodePipeline · Bamboo · Concourse · Electric Cloud · Gradle · Jenkins · Maven · MSBuild · Nant · TeamCity · Visual Studio · Visual Studio TFS	· Bamboo · Blazemeter · Gatling · Jenkins · JMeter · LoadRunner · MSTest · Selenium · SilkPerformer · SoapUI · TeamCity · Visual Studio · Visual Studio TFS	· Ansible · AWS CodeDeploy · Bamboo · Chef · Cloud Foundry · Deis · Docker · Electric Cloud · Jenkins · OpenShift · Puppet Labs · Salt · TeamCity · Wercker	· Ansible · AWS CodeDeploy · Bamboo · Chef · Cloud Foundry · Deis · Docker · Electric Cloud · Jenkins · OpenShift · Puppet Labs · Salt · TeamCity · Wercker	· Elasticsearch · SPLUNK • Nagios	· HipChat · OpsGenie · PagerDuty · ServiceNow · Slack · VictorOps

We will discuss continuous deployment from the perspective of popular tool features and functionality listed in the preceding figure.

Chef

Chef is one of the prominent configuration management and infrastructure automation platforms; it provides a full suite of enterprise capabilities such as workflow, visibility, and compliance. It enables continuous deployments for both infrastructure and applications from development to production. Infrastructure configuration automation as code is written, tested, deployed, and managed by Chef across networks such as the cloud, on-premises, or hybrid environments with comprehensive 24 x 7 support services. Examples are client systems, security patches can be updated from master server by writing configurations as a set of instructions and executed on multiple nodes simultaneously.

The Chef platform as shown in the following figure, supports multiple environments such as Amazon Web Services, Azure, VMware, OpenStack, Google Cloud, and so on. Platforms such as Windows, Linux, VMware, and so on, are available. All the popular continuous integration tools such as Bitbucket, Jenkins, GitHub, CircleCI, and so on, are supported for workflow integration. The runtime environment is available on Kubernetes, Docker, and Swarm.

Chef landscape components

The Chef landscape comprising various elements of Chef, including the nodes, the server, and the workstation along with their relationships is shown here. We will discuss each of the components, the terminology, and the role it plays in the ecosystem to enable a Chef client to execute the job assigned. Chef terminology resembles food preparation. Cookbooks are the formula to make food dishes and recipes are ingredients.

The components of Chef are:

- Chef server
- Chef client
- Chef workstation
- Chef repo

Chef server

The Chef server is the hub for maintaining the configuration data across the network, storing cookbooks, applying policies to nodes, and each registered node detailed metadata managed by the Chef client. Chef server provides configuration details, such as recipes, templates, and file distributions through the Chef client installed on the respective nodes. Accordingly, the Chef clients implement the configuration on their nodes relieving the Chef server of any processing tasks. This model is scalable to consistently apply the configuration throughout the organization.

Features of Chef server

Web-enabled user interface with management console and search.

- Management console on the chef server is a web-based interface for managing multiple functions such as:
    - Nodes in the network
    - Cookbooks and recipes
    - Roles assigned
    - Data bags--JSON datastores, might include encrypted data
    - Environment details
    - Search facility for indexed data
    - Administrative user accounts and data for chef server access
- Search functionality facilitates querying any types of data indexed on Chef server such as nodes, roles, platforms, environments, data bags, and so on. The Apache Solr search engine is the base search engine and extends all the features such as pattern search with exact, wildcard, range, and fuzzy. A full indexed search can be run with different options within the recipe, command line, management console search feature, and so on.
- Data bags are located in a secure sub-area on the Chef server; they store sensitive data such as passwords, user account data, and other confidential types of data. They can only be accessed by nodes with the authentic SSL certificates validated by the Chef server. A data bag is accessed by the Chef server with its global variables stored in JSON format. It can be searched and accessed by recipe and loaded.

- A policy defines how roles, environments, and cookbook versions are to be implemented in the business and operational requirements, processes, and production workflows:
    - A role is a way to assign the tasks based on specific functions, patterns, and processes performed in an organization such as power or business user, and so on. Each node, web, or database server consists of unique attributes and a run list is assigned per role. When a node is to perform a task, it compares its attributes list with those required to execute the function. The Chef client ensures the attributes and run lists are up-to-date with those on the server.
    - Environments reflect organizations real-life requirements such as development, staging, or production systems, each are maintained with a cookbook version.
    - Cookbooks maintain organization-specific configuration policies. Different cookbook versions are maintained such as source control with associated environments, metadata, run lists for different needs; they are uploaded on to a Chef server and applied by a Chef client while configuring the nodes. A cookbook defines a scenario and everything that is required to support that scenario is contained such as:
        - Recipes that specify which resources to use and the order as well
        - Attribute values
        - File distributions
        - Templates
        - Chef extensions such as custom resources and libraries
    - A run-list contains all the required information for Chef to configure a node to a desired state. It is an ordered list of roles and recipes specified in the exact order to be run to reach its intended state. It's tailored for each node and stored on the Chef server as part of the node object. It is maintained using knife commands or using the Chef management console on the workstation and uploaded to the Chef server.

Chef client on nodes

Chef clients can be installed on different node types--physical, virtual, cloud, network device, and so on, that are registered with Chef server.

- Types of nodes:
    - A physical node is an active device (system or virtual machine) attached to a network on which a Chef client is installed to communicate with a Chef server.
    - A cloud-based node is hosted in external cloud environments such as AWS, Microsoft Azure OpenStack, Google Compute Engine, or Rackspace. Knife with plugins provides support for external cloud-based services and creates instances to deploy, configure, and maintain those instances.
    - A virtual node is a system that runs like a software implementation without direct physical machine access.
    - A network node such as a switch can be configured with Chef and automated for physical and logical Ethernet link properties and VLANs. Examples of network devices are Juniper Networks, Arista, Cisco, and F5.
    - Containers are virtual systems running individual configurations sharing the same operating system. Containers are effective at managing distributed and scalable applications and services.
- Chef client:
    - The Chef client does the actual configuration. It contacts the Chef server periodically to retrieve the latest cookbooks to update the current state of the node, if required, in accordance with the cookbook instructions. This iterative process is enforced by business policy to ensure the network is in accordance with the envisioned target state.
    - The Chef client is the local agent that is installed and runs on every node registered with the Chef server to ensure the node is at the expected state. Chef client does most of the computational effort. It's typically a virtual machine, container instance, or physical server.
    - Authentication between Chef client with the Chef server happens through RSA public key/pairs for every transaction request. The data stored on the Chef server is shared after authentication of registered nodes. Any unauthorized data access is avoided.

- After installation of the chef client, the nodes become compute resources on infrastructure that is managed by Chef for performing the tasks such as:
    - Registering the node with the Chef server
    - Authentication services
    - Creating the node object
    - Synchronizing cookbooks with Chef server
    - The required cookbooks, with recipes, attributes, and all other dependencies, are compiled and loaded
    - Configuring the node as per the requirements
    - Exception handling and notifications

Ohai

Ohai is a tool run by Chef client to collect system configuration and metrics data with many built-in plugins to determine the system state for use within cookbooks. The metrics collected by Ohai are:

- Operating System
- Kernel
- Host names
- Fully-qualified domain names
- Virtualization
- Cloud service provider metadata
- Network
- Memory
- Disk
- CPU

Attributes that are collected by Ohai are automatically used by the Chef client to ensure that these attributes remain consistent with the definitions on the server.

Workstations

Workstations facilitate users to author, test, and maintain cookbooks and interact with the Chef server and nodes. The Chef development toolkit is also installed and configured on a workstation. The Chef development kit is a package comprising prescribed sets of tools, and includes Chef, the command-line tools, Test Kitchen, ChefSpec, Berkshelf, and a few others. Users use workstations for:

- Developing the cookbooks and test recipes
- Testing the Chef code in different environments
- Version source control synchronized with Chef repo
- Defining and configuring roles and environments and organizational policy
- Enforcing data bags are used for storing the critical data
- Performing a bootstrap operation on nodes

Cookbooks are repositories for files, templates, recipes, attributes, libraries, custom resources, tests, and metadata. Chef client configures each node in the organization through cookbooks and recipes, the fundamental unit of configuration is the cookbook and provides structure to your recipes. Infrastructure state is defined as a file, a template, or a package in policy distribution as per the required scenario.

The programming language for Chef cookbooks is Ruby as a full-fledged programming language with syntax definitions. Recipes are simple patterns for specific configuration items such as packages, files, services, templates, and users with blocks that define properties and values that map to them. Recipes are the fundamental configuration element in a cookbook. A Chef recipe is a file that groups related resources, such as everything needed to configure a web server, database server, or a load balancer. Recipes are stored in cookbooks and can have dependencies on other recipes.

Chef repo

The Chef repo, as the name suggests, is the repository artifact to author, test, and maintain the cookbooks. The Chef repo is managed like source code, synchronizing with a version control system (such as GitHub, Bitbucket, and so on). The Chef repo directory structure can contain a Chef repo for every cookbook or all of their cookbooks in a single Chef repo.

The `knife` is a command interface to communicate with the Chef server from the workstation to upload the cookbooks. To specify configuration details, the `knife.rb` file is used, `knife` helps to manage:

- Nodes bootstrapping
- Recipes and cookbooks
- Environments, roles, and data bags
- Various cloud environment resources
- Chef client installation to nodes
- Chef server indexed data search features

The package of tools and utilities to work with Chef is called **Chef Development Kit** (**Chef DK**). It includes command-line tools interacting with Chef such as `knife` Chef server and Chef clients and with local Chef code repository (`chef-repo`). The components of Chef DK are as follows:

- Chef client
- Chef and `knife` command-line tools
- Test Kitchen, Cookstyle, and Foodcritic as testing tools
- Compliance and security requirements with InSpec as executable code
- Cookbooks are authored to upload to Chef server
- To encryption and decryption of data bag items is with Chef-Vault using the public keys for registered nodes
- Cookbooks dependency manager
- Workflow tool Chef
- Unit testing framework Chef Specto tests resources locally
- For style-checking to write clean cookbooks Rubocop-based tool Cookstyle
- Continuous delivery workflow on Chef Automate server also command-line tools to set up and execute
- For static analysis of recipe code Foodcritic is a lint tool
- It is to test cookbooks across platforms, an integration testing framework tool is Test Kitchen
- For rapid cookbook testing and container development `kitchen-dokken` is `test-kitchen` plugin with a driver, transport, and provisioner for using Docker and Chef
- Kitchen driver for Vagrant is `kitchen-vagrant`

- People to work together in the same `chef-repo` and Chef server knife workflow plugin is `knife-spork`
- The preferred language for Chef is Ruby

A recipe is the collection of resources, defined using patterns such as resource names, attribute-value pairs, and actions. It is the fundamental configuration element designed to read and act in a predictable manner and authored in Ruby programming language.

A few properties are as follows:

- Include all that is required to configure the system
- To be stored in a cookbook
- For the Chef client to be used, it must be added to a run list
- It is executed in the same sequence as listed in a run list
- Chef client will run the recipe only when instructed
- Could be included in another recipe
- Might read the contents of a data bag (encrypted data bag)
- Might input the results of a search query
- Might have dependency on other recipes
- Facilitate the creation of arbitrary groupings by tagging a node
- If the recipe is constant, then there won't be any change by repeated execution

Recipe DSL is a Ruby DSL that is used to declare resources primarily from within a recipe. It also helps to ensure recipes interact with nodes (and node properties) in the expected manner. Most of the Recipe DSL methods find a specific parameter to advise Chef client on actions to take according to the node parameter.

A resource is a configuration policy statement that:

- Describes the configuration item desired state
- Declares the steps on the item required for the desired state
- Resource type is specified such as package, template, or service
- Lists additional resource properties
- Are grouped into recipes, that describe working configurations

Chef has built-in resources to cover common actions across common platforms and can be built to handle any customized situation.

With different versions of cookbooks, multiple environments of production, staging, development/testing are managed.

Cookbook template resources are used to add to recipes for dynamic generation of static text files.

To manage configuration files, **Embedded Ruby** (**ERB**) templates are used.

The cookbooks/templates directory contains ERB template files with Ruby expressions and statements.

The cookbooks are written consistently as per standards and tested for same.

With unit and integration testing, the cookbooks recipes are validated, testing code quality is also called **syntax testing**.

Test Kitchen, ChefSpec, and Foodcritic, and so on, are tools for testing Chef recipes.

The attribute files are executed in the same order as defined in the cookbook.

Chef is built on top of Ruby, it is a thin **domain-specific language** (**DSL**) with built-in taxonomy for customizations need of organization.

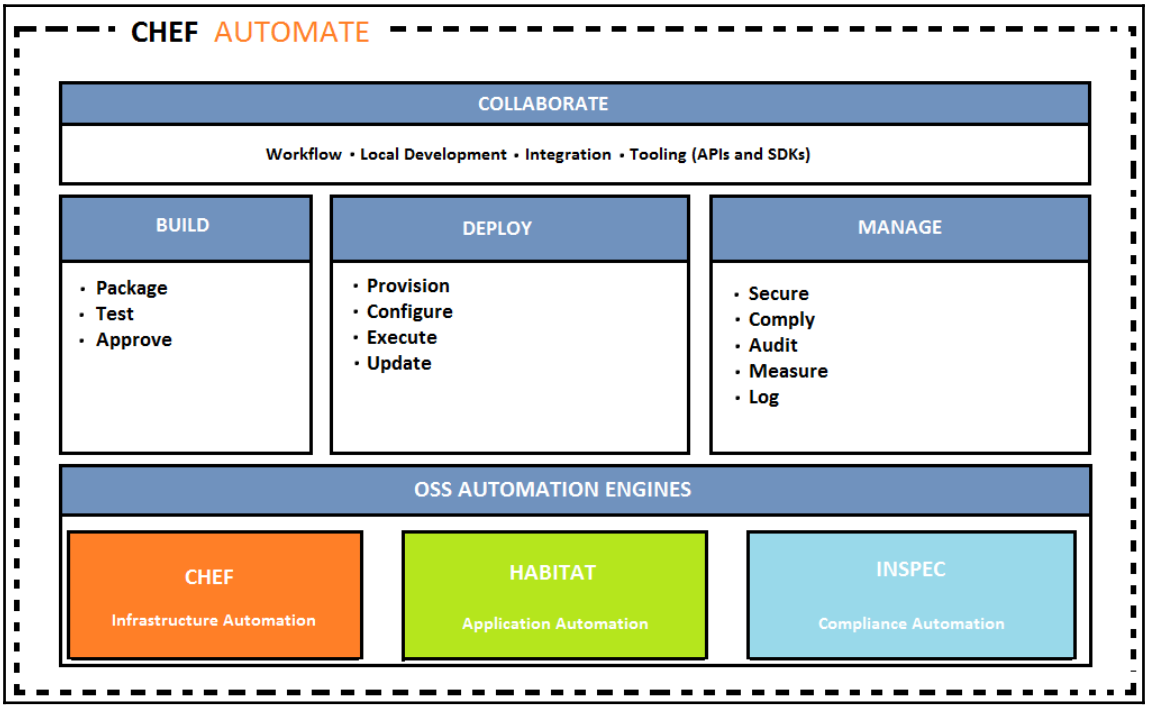
To manage environments, cookbooks, data bags, and to configure role-based access for users and groups, attributes, run lists, roles, and so on, the Chef server user interface is the Chef management console.

Chef Supermarket is the community location to share and manage. Cookbooks may be used by any Chef user or organization.

Extended features of Chef

It is a powerful automation platform that transforms infrastructure into code that operates on the cloud, on-premises, or in a hybrid environment. Infrastructure is configured, deployed, and managed across your network irrespective of the organization size with Chef Automate. Integral parts of Chef Automate are Chef, Habitat, and InSpec.

Three open source power-packed engines are shown in the following image:



Chef is the core engine for infrastructure automation. Habitat is an application automation tool emulating concepts of containers and microservices. InSpec ensures compliance and security requirements by specifying executable code.

Habitat

Habitat comes with a prescribed packaging format for application automation; the Habitat supervisor and application dependencies are packaged and deployed as one unit. The Habitat package format defines on how to be structured, these are isolated, immutably executed for any kind of runtime environments such as a container, bare metal, or PaaS. The Habitat supervisor manages the package's peer relationships, upgrade strategy, and security policies, which are auditable as well.

InSpec

InSpec is an open source to test for adherence to security policies. It's a framework for specifying compliance, security, and policy requirements to automatically testing any node in the infrastructure. Compliance can be expressed as code and integrated into a deployment pipeline:

- InSpec using the Compliance DSL enables you to write auditing rules quickly and easily
- InSpec examines infrastructure nodes to run the tests locally or remotely
- Security, compliance, or policy issues noncompliance is logged

The InSpec audit resource framework and Chef Compliance are fully compatible.

It runs on multiple platforms with remote commands such as SSH or using Docker API, apart from ensuring compliance using APIs, it can access the database, inspect, and can restrict usage of services or protocols and the configuration of virtual machines. An example is to Restrict Telnetd or the FTP service on the client or server machines.

The continuous deployment full-stack pipeline is Chef Automate. It includes automated testing for compliance and security. The workflow provides visibility for both applications and infrastructure, as well as changes propagating throughout the pipeline from development production.

Chef High Level Architecture components are Chef DK, Chef Server, and clients:

The Chef server plays multiple roles and acts as a hub for configuration data. It stores cookbooks, applies the policies to the systems in accordance with the infrastructure, and metadata defined for each system.

Cookbook development workflow is prescribed by the Chef Development kit as below:

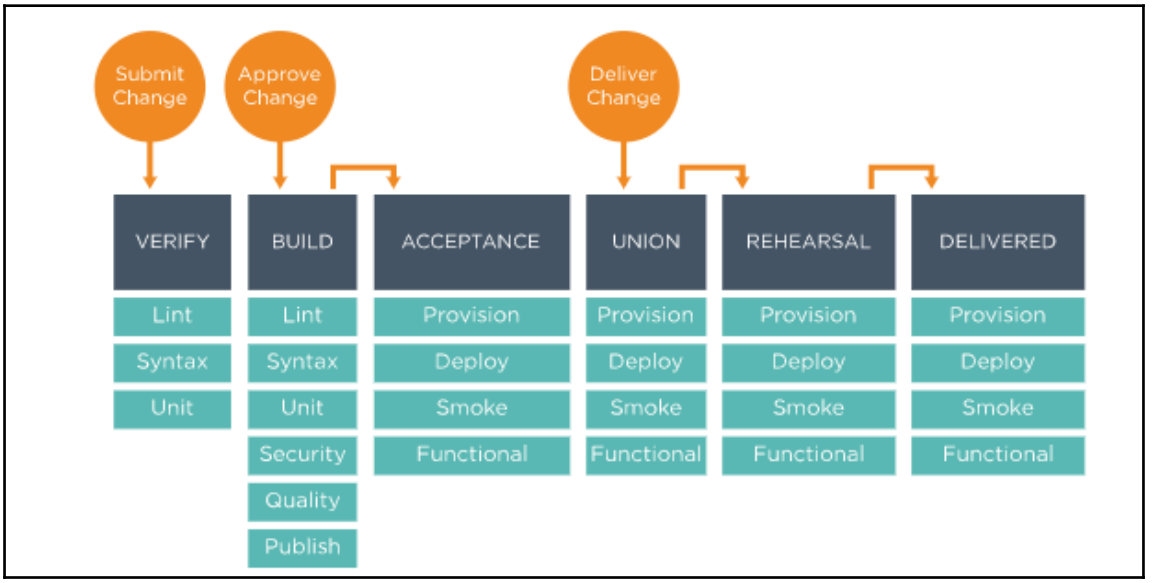
- Skeleton cookbook creation: A cookbook with the standard files already part of the Chef Development kit, the Berkshelf is the package manager that helps manage cookbooks and related dependencies.
- Virtual machine environment creation using Test Kitchen: Environment that develops the cookbook with the location details for performing automated testing and debugging of that cookbook during development.
- Prepare and debug the recipes for the cookbook: An iterative process to develop and test cookbooks, fix bugs, and test till they meet their purpose. Cookbooks are authored with any text editor such as Sublime Text, vim, TextMate, EditPad, and so on.
- Conduct acceptance tests: These tests are done against a full Chef server using a near production environment as opposed to development environment.
- The cookbooks that pass all the acceptance tests in the desired manner are deployed to the production environment.

Chef Automate workflow

Chef Automate pipeline is for continuous delivery of full-stack approaches for infrastructure and applications. It facilitates safe deployment with any application, changes at high velocity, and relates infrastructure changes.

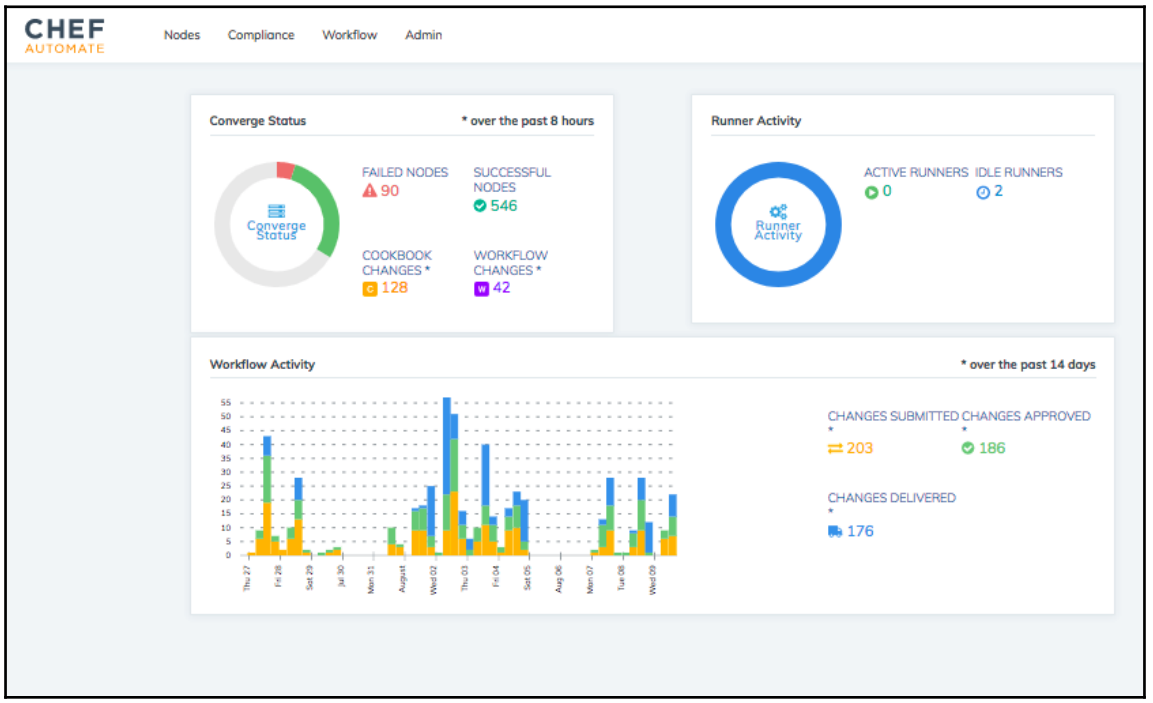
The Chef Automate pipeline quality gates are automated to move changes from a developer's workstation from deployment to production. A proposed change is approved by a team and afterwards, acceptance tests are approved and released to the respective artefact for delivery into production.

This diagram shows the workflow from development, test, and deployment of Chef code:



The artefact moves through the pipeline after the acceptance stage, moves to the union stage of quality assurance, rehearsal (pre-production), and delivered (production).

The Chef Automate graphical user interface provides views into operational and workflow events. Its data warehouse collects inputs from Chef, Habitat, Automate workflow, and compliance. Dashboards track each change status through the pipeline and query languages available to customize dashboards.

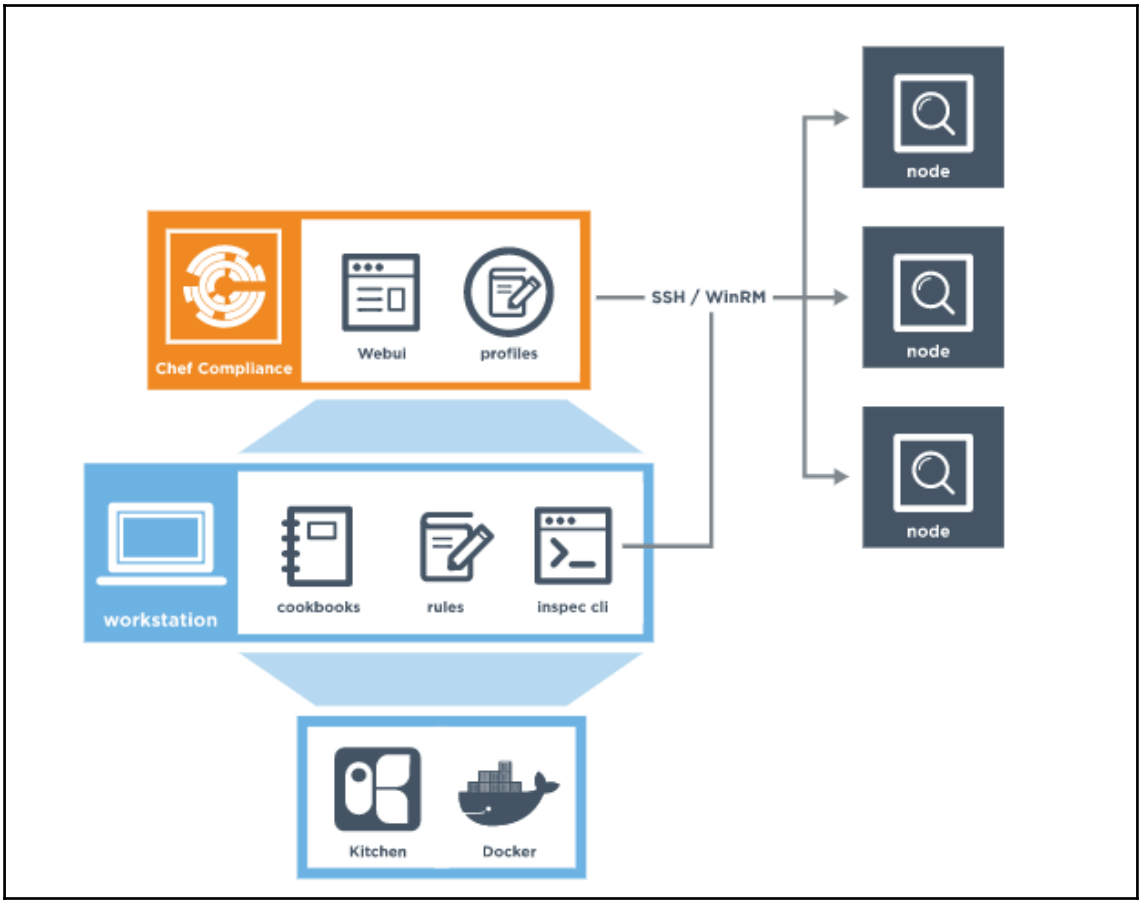


Compliance

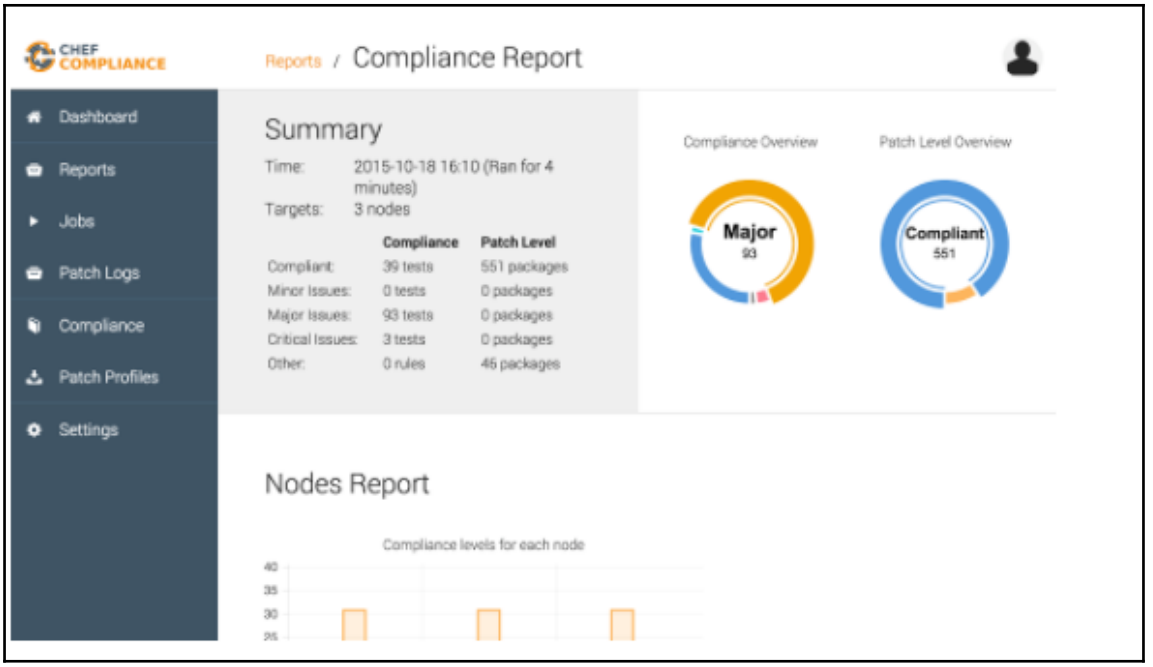
Compliance issues, security risks, and outdated software can be identified by creating customizable reports with compliance rules in InSpec. There are built-in profiles with predefined rule sets for security frameworks such as **Centre for Internet Security** (**CIS**) benchmarks, and so on. Compliance reposting can be standalone or integrated. Also, the Chef Automate server provides high availability with fault tolerance, real-time data about your infrastructure, and consistent search results.

The Chef Compliance server facilitates centralized management of the infrastructure's compliance, performing the following tasks:

- Create and manage profiles of rules
- Test nodes as per the organization's security management life cycle regularly
- The scans are fully executed remotely; no footprint is installed on the node
- Compliance reports ensure infrastructure meets security requirements
- Auditing statistics for nodes compliance are available



Chef compliance reports detailing multiple parameters such as node wise for patch and compliance are shown in the following:



Chef Compliance Report views from Automate.

Chef Automate provides the ability to analyze compliance reports to pivot the data for either nodes, platform of the node, environment, or profiles with the ability to drill down on the information.

Chef Automate compliance control status report provides a comprehensive dashboard on major, minor, critical, patch levels, and so on.

Ansible

Ansible is a popular and powerful automation framework for continuous delivery with features and benefits are listed in the following topics:

Prominent features

Ansible provides following features:

- **Modernize**
  - Automate existing deployment process
  - Manage legacy systems and process, updated like DevOps
- **Migrate**
  - Define applications once and redeploy anywhere
- **DevOps**
  - Model everything, deploy continuously

Benefits of Ansible

Using Ansible provides multiple advantages as listed following:

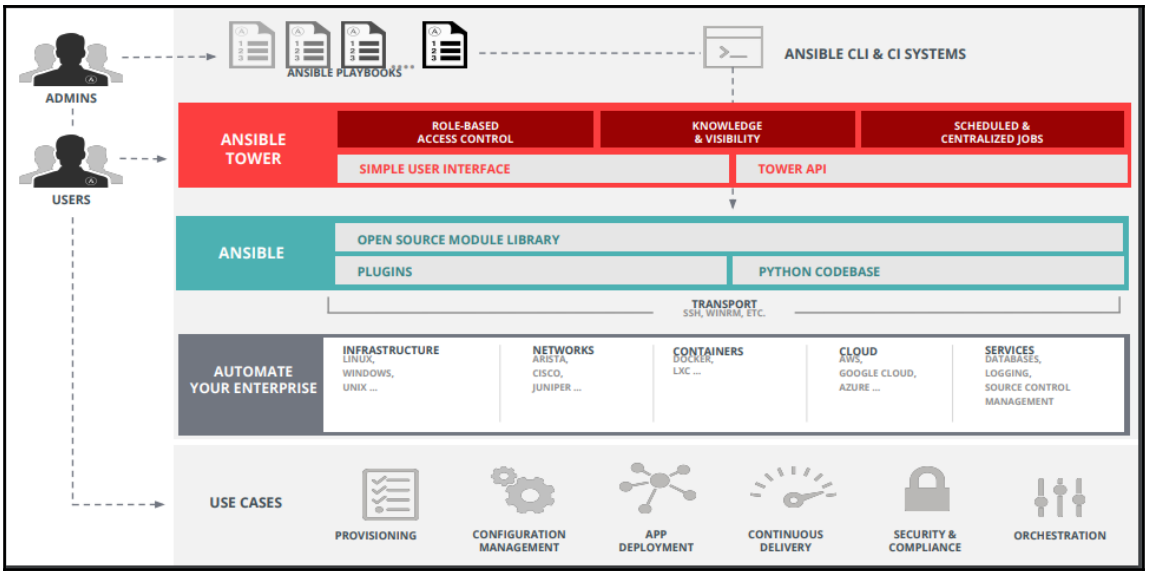
- **Simple to use**
  - Special coding skills not required
  - Tasks are executed sequentially
  - Get productive quickly
  - Automation is easily understood
- **Powerful with functionality**
  - App deployment
  - Configuration management
  - Orchestration of workflow
  - Orchestration of app life cycle
- **Agentless**
  - Agentless architecture
  - Uses OpenSSH and WinRM
  - No agents to exploit or update
  - More efficient and secure

Ansible is a multi-dimensional IT automation engine that simplifies automation of cloud provisioning, intra-service orchestration, configuration management, application deployment, and many other IT functionalities.

Ansible models your IT infrastructure by prescribing to interrelate systems for multi-tier deployments against managing the systems individually.

As discussed under features, there are neither client-side agents nor additional custom security infrastructure for Ansible. It makes deployment very simple by describing automation jobs in a plain English language called YAML and in the form of Ansible playbooks.

Ansible architecture is as shown in the following:

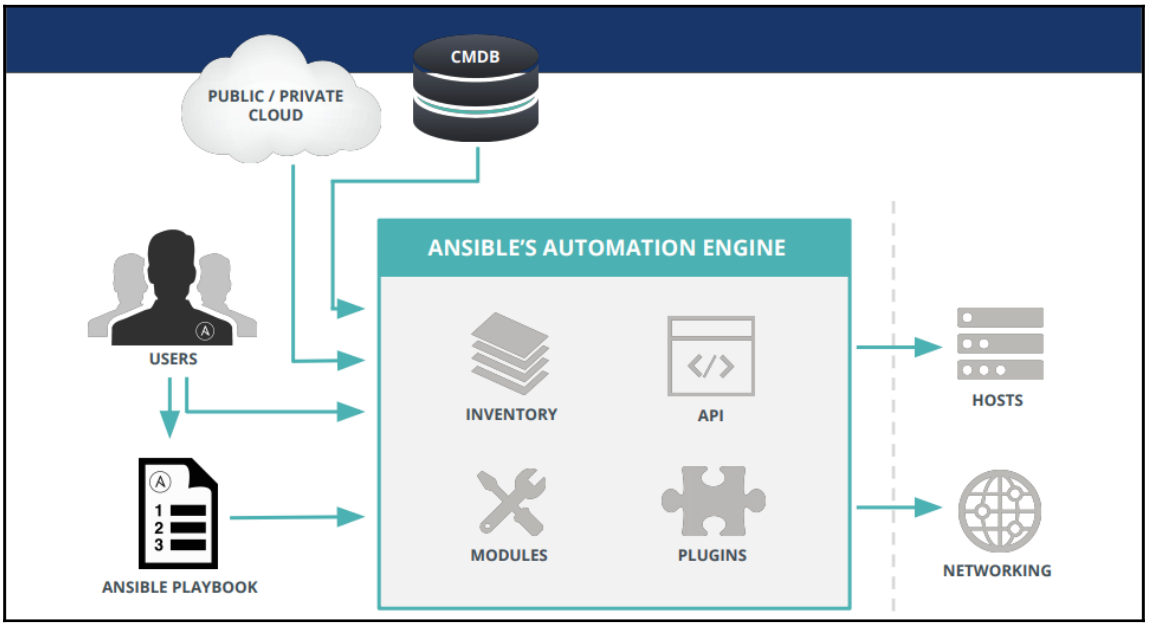


Ansible terminology, key concepts, workflow, and usage

Ansible Tower is a web-based solution for enterprise automation frameworks designed to be the hub for controlling, securing, and managing your Ansible environment with a user interface and RESTful APIs. It provides the following rich features:

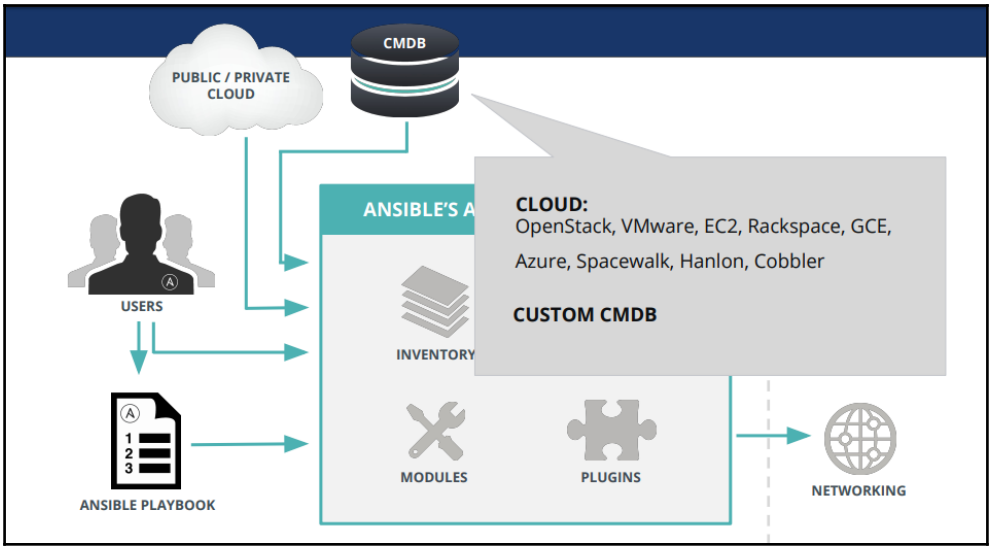
- Access control is role-based to keep the environment secure, and efficient in managing – allows sharing of SSH credentials but not transfer
- With push-button deployment access even non-privileged users can safely deploy entire applications by providing access on the fly

- Ensuring complete auditability and compliance as all Ansible automations are centrally logged
- Inventory with a wide variety of cloud sources, can be graphically managed or synced
- It's based on a robust REST API, integrates well with LDAP, and logs all jobs
- Easy integration with the continuous integration tool Jenkins, command-line tools options are available
- Supports auto scaling topologies though provisioning callback
- Ansible Tower is installed using Ansible playbooks



CMDB

Ansible **configuration management database** (**CMDB**) maintains the entire configuration information of the enterprise in the database and supports cloud creation options in multiple formats for different vendors.



Playbooks

Playbooks are the configuration programs written in YAML to automate the systems. Ansible can finely orchestrate multiple instances of infrastructure topology with very detailed control over many machines at the same time. Ansible's approach to orchestration is finely-tuned automation code managed through simple YAML on syntax or features.

Ansible playbooks describe a policy to be orchestrated for enforcement on remote systems for configuration and deployment to enforce general IT process adherence steps.

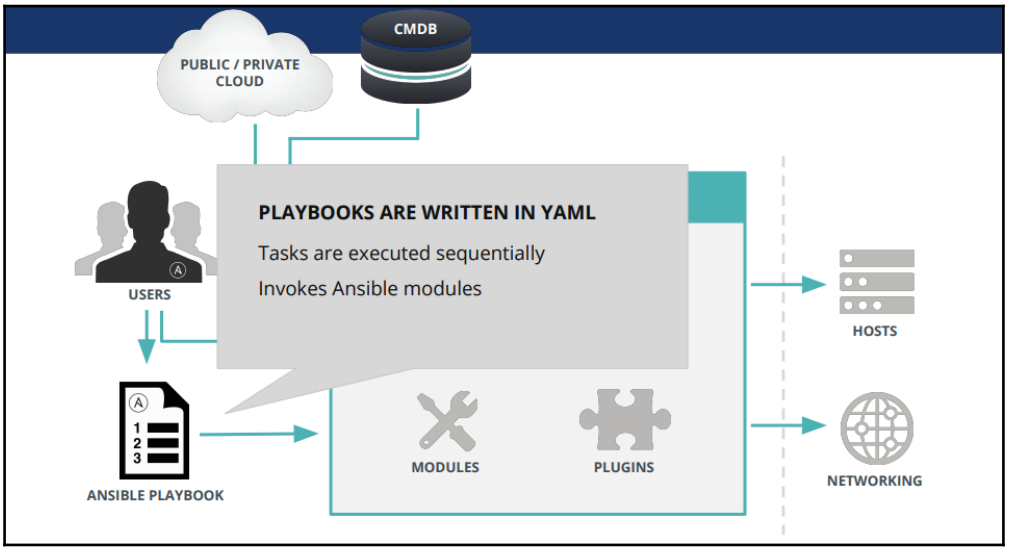
A simple analogy is, an inventory of hosts is raw material, instruction manuals are playbooks, and Ansible modules are the tools in the workshop.

To manage configurations of deployments to remote machines, playbooks can be used at a basic level. They can sequence multi-tier rollouts involving rolling updates on a more advanced level, to interact with monitoring servers and load balancers along the way and delegate actions to other hosts.

Playbooks are developed in a basic text language conveniently designed to be human-readable. Organizing playbooks and the files can be done in multiple ways.

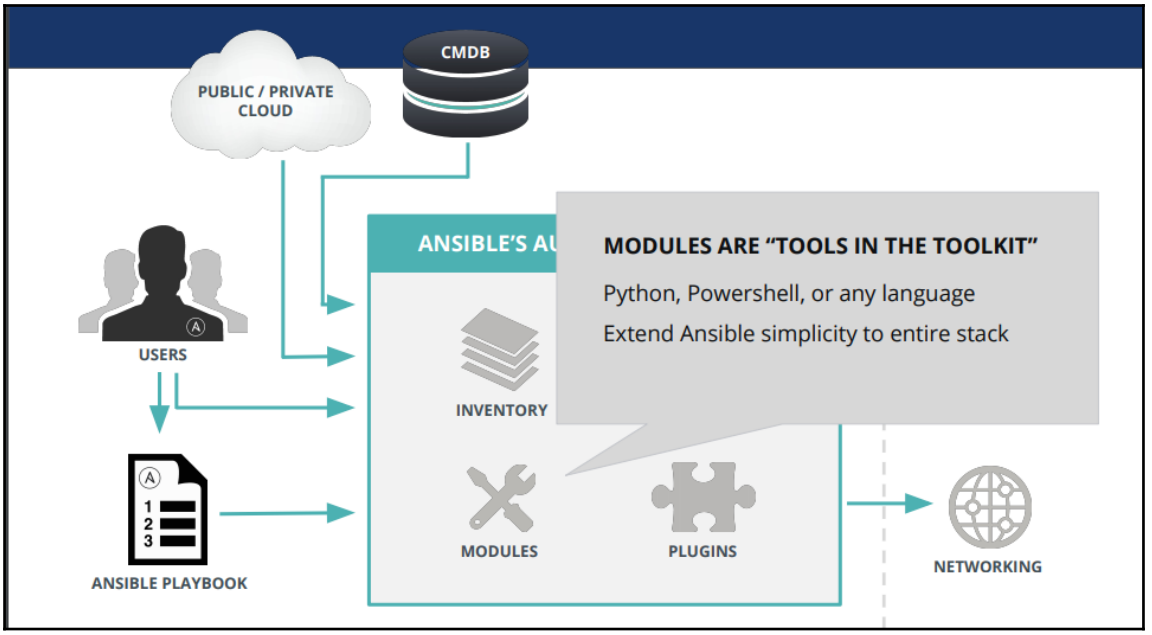
A simple playbook example:

```
- hosts: webservers
serial: 6 # update 6 machines at a time
roles:
- common
- webapp
- hosts: content_servers
roles:
- common
- content
```



Modules

Ansible modules can control system resources, such as services, packages, or files to handle and execute system commands. These resource modules are pushed by Ansible on nodes to configure them to the desired state of the system. These Ansible modules are executed over SSH (Secured Shell) on the target nodes and removed after accomplishing the task. The module library is shipped by default with a number of modules to be executed through playbooks or directly on remote hosts. The modules can reside on any machine, there is no concept of servers, daemons, or databases to maintain them. The modules and libraries are customizable, typically created with any terminal program, a text editor, and to keep track of changes to the content, the version control system is used effectively.



Inventory

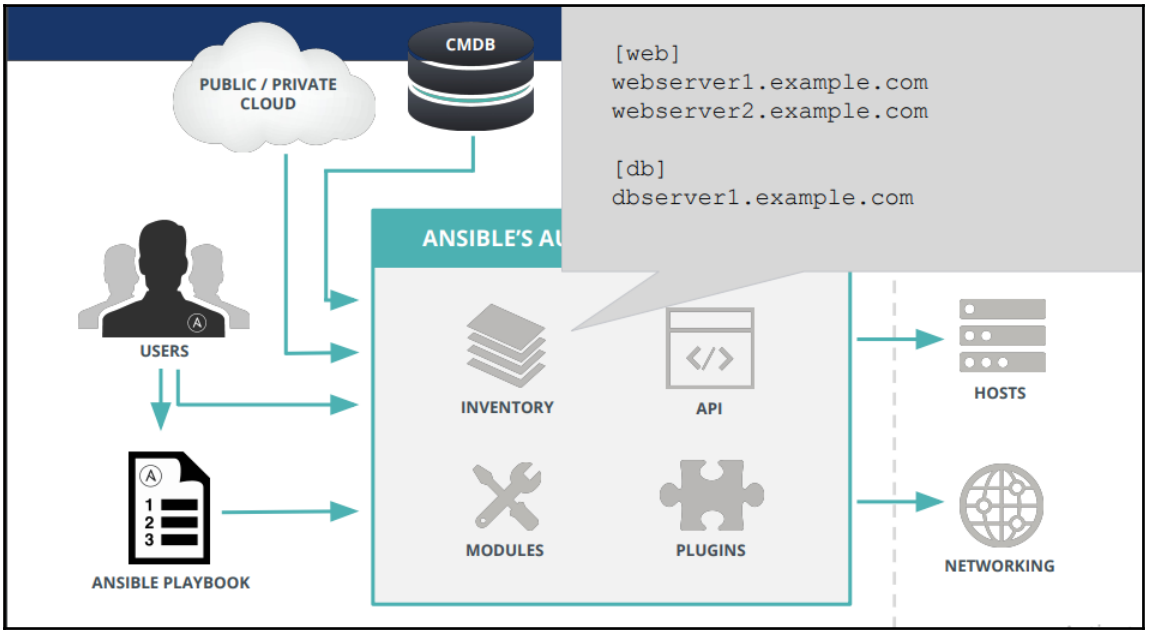
Ansible inventory is a list of resources:

- Hosts and groups
- Host variables
- Group variables
- Groups of groups and group variables
- Default groups
- Splitting out group and host and specific data
- List of inventory behavioral Parameters
- Non-SSH connection types

Ansible, through the inventory list, works on multiple systems in the infrastructure simultaneously. The dynamic inventory mechanism allows multiple inventory files to be flexible and customizable at the same time through inventory plugins. The inventory list can be in a default location or specify inventory file location of your choice from dynamic or cloud sources, EC2, Rackspace, OpenStack, or different formats.

Here's what a plain text inventory file looks like:

```
[webservers]
www1.example.com
www2.example.com
[dbservers]
db0.example.com
db1.example.com
```



Plugins

Ansible's core functionality is augmented by a number of handy plugins and can be customized in JSON (Ruby, Python, Bash, and so on). Plugins can connect to any data source, extend the connection types for transport other than with SSH, call back for logs, and even add new server-side behaviors.

Ansible Tower

Offers multiple features such as:

- LDAP, AD, SAML, and other directories can be connected
- Access control engines that are role based
- Credentials without exposure storage
- Simple for first time users
- Smart Search enabled information lookup
- Configure automation at runtime
- REST API based integration with processes and tools
- Tower clusters to extend capacity

Ansible Tower can invoke multi-playbook workflows to link any number of playbooks, with different inventories, run as different users, run as batch, or with different credentials.

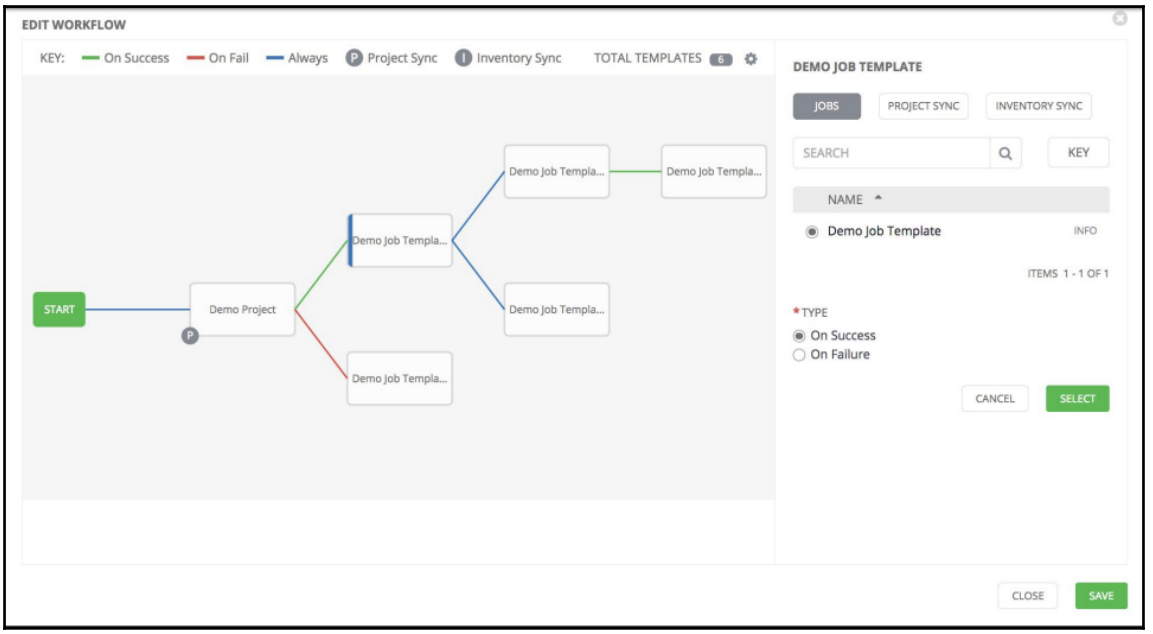
Ansible Tower workflows facilitate many complex operations, build workflows to provision the machines, apply base configurations of systems, and deploy the applications by different teams maintaining different playbooks. A workflow can be built for CI/CD to build an application, deploy to a test environment, run tests, and based on test results, automatically promotes the application. Ansible Tower's intuitive workflow editor easily models complex processes with different playbooks set up to run as alternatives in case of success or failure of a prior workflow playbook.

A typical workflow may be as follows, it can be effectively used on multiple systems quickly without taking their infrastructure offline. To achieve continuous deployment, automated QA is vital to mature to this level:

- Script automation to deploy local development VMs
- CI system such as Jenkins to deploy to a staging environment on every code change
- The deploy job executes the test scripts on build for pass/fail for every deploy
- Upon success of the deploy job, the same playbook is run against production inventory

The Ansible Tower workflow brings the following features and functionality:

- Jobs schedule
- Built-in notifications to inform the teams
- Stabilized API to connect to existing tooling and processes
- New workflows to model entire processes



The Ansible Tower dashboard (refer to the image) offers functionality as listed:

- Dashboard and real-time automation updates
- Graphical inventory management
- Integrated RBAC with credential management

Ansible Vault

Ansible Vault is a feature to keep sensitive data in encrypted form, for example passwords or keys as opposed to saving them as plain text in roles or playbooks. These vault files can be placed in source control or distributed to multiple locations. The data files such as Ansible tasks, handlers, arbitrary files, even binary files can be encrypted with Vault as well. These are decrypted at the destination on target host.

Ansible Galaxy

Ansible Galaxy is an open source website designed for community information and contributing to collaborate on building IT automation solutions to bring together administrators and developers. There are preconfigured roles to downloaded and jump start automation projects with Galaxy search index. These are also available with a GitHub account.

Testing strategies with Ansible

Though testing is a very organizational and site-specific concept, Ansible Integrated Testing with Ansible playbooks is designed as an ordered and fail-fast system. It facilitates embed testing directly in Ansible playbooks through a push-based mechanism.

Ansible playbooks are models of desired-state of the system that will ensure the things declared, such as services to be started and packages installed, are in accordance with declarative statements. Ansible is an order-based system on unhandled errors. A host will fail immediately and prevent further configuration of that host and shows them as a summary at the end of the Ansible run. Ansible is a multi-tier orchestration system to incorporate tests into the playbook run, either as tasks or roles.

Testing the application for integrating tests of infrastructure before deployment in the workflow will be effective to check the code quality and performance before it moves to production systems. Being push-based, the checks and balances in the workflow and even upgrading is very easy to maintain on the localhost or test servers.

Monitoring

Enterprise monitoring is a primary activity and it categorizes monitoring development milestones, application logs, server health, operations, infrastructure, vulnerabilities, deployments, and user activity. These are accomplished with:

- Collecting and key messages
- Mature monitoring tools
- Avoid perceptions and making decisions based on uncertainty
- Participative monitoring and evaluation
- Selecting and using right indicators

- Interpreting indicator results in business context
- Real-time data collection
- Managing data and information

Development Milestones: Monitoring of development milestones is an indicator of how well your DevOps adoption strategy is working by gaining insights of the actual process and team effectively. Some of the metrics are sprint scope changes, bugs count of field and fixed, and the ratio of promised-to-delivered features. These metrics are the drivers on team effectiveness and adherence to the schedule, this monitoring is built-in as an Agile plugin for issue tracking.

Code Vulnerabilities: Monitoring vulnerabilities in application code, lists the weaknesses induced in the top-level code by insecure coding practices. These can be addressed by conducting regular code reviews or changing third-party dependencies , and so on.

Deployments: Deployment monitoring is configuring your build servers to have some monitoring built into the process to notify the team. Notification-capable continuous integration servers communicate with chat servers and promptly alert teams of failed builds and deployments.

Application log output: Application log output to be planned for centralized logging if services are distributed to gain full benefit, errors and exceptions provides value in real-time. The ability to trace notifications from error-producing code in a searchable format generates benefit, before production move.

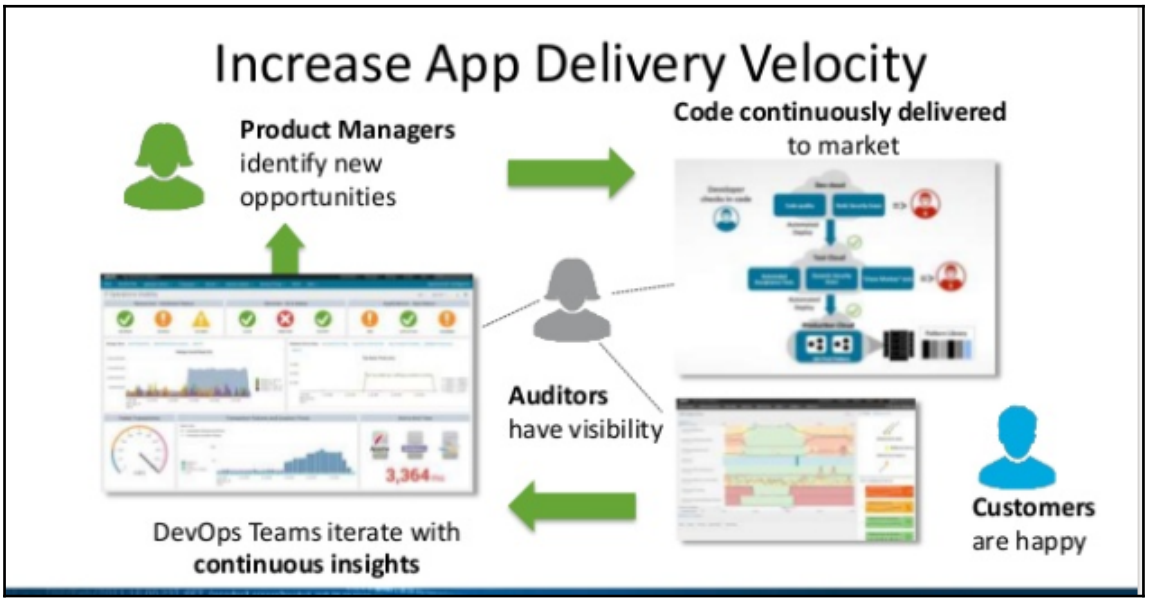
Server Health: Monitoring of uptime and performance of available resources downed or over-utilized servers fall in this category. Intrusion detection and health monitoring systems being on the same notification pipeline will provide additional value.

Activity Monitoring: User activity monitoring is both feature development and the scaling of infrastructure. Along with monitoring development milestones volume of data is monitored.

The centralized storage of consolidated logging data for application logs, user activity monitoring, project history enhances the value to detect and analyze in a global context correlating different log sources about the state of the application and the project.

Splunk

Splunk is a popular application monitoring tool to gain real-time visibility into DevOps-driven application delivery for Continuous Delivery or Continuous Integration to move from concept to production quickly. Splunk enterprise helps improve the business impact of application delivery by enhancing velocity and quality.



Splunk improves code quality with the following benefits:

- Resolve code issues before customers see them
- Detect and fix issues related to production faster
- Objective metrics are available to ensure code is operational and meets quality SLAs

Splunk is a platform to capture and record all the activity and behavior of your customers, machine data, users, transactions, applications, servers, networks, and mobile devices.

The Splunk platform enhances its business impact by integrated real-time insights from application development to testing to production monitoring. It provides cohesive views across all stages of the delivery life cycle as opposed to discrete release components.

Real-time visibility into business-relevant data for business and DevOps leaders on both development and operations, such as application performance, usage, revenue systems, cart fulfillment, and registration data provides insights to better plan inventory and report and improve the customer experience.

Development life cycle integration and visibility across diverse, multiple supported phases and applications is supported:

Operations lifecycle integration and visibility across diverse, multiple supported phases and application is supported. Applications are delivered faster using analytics:

- End-to-end visibility across every DevOps delivery tool chain component
- Correlated insights that iterate faster across the application delivery lifecycle
- Measuring and benchmarking release contributions and improving DevOps team efficiency

Splunk helps organizations by enabling a feedback loop to business leaders, evaluating the real impact of code changes on their customers. Continuous interaction helps to build more intelligence about machine behavior and deep asset models.

The benefits reflect the business impact of application delivery:

- Gain new business insights by correlating business metrics with code changes
- Enhance user experience through delivery of better-performing code
- Delivering more secure and compliant code improves reputation

Nagios monitoring tool for infrastructure

There are multiple variants of the Nagios open source tool for monitoring mission-critical infrastructure components specific to each segment on any operating system:

- Network monitoring software
- Network traffic monitoring

- Server (Linux, Windows) monitoring
- Application monitoring tools
- Web application monitoring
- Monitoring core engine and a basic web interface
- Nagios core plugins package with add-ons
- Nagios log server security threats with audit system

Nagios facilitates monitoring of the network for problems such as overloaded data links, network connections, monitoring routers, switches, problems caused by overloaded of crashed servers, and so on.

Nagios can deliver the metric results in a variety of visual representations and reports to monitor availability, uptime, and response time of every node on the network with both agent-based and agent-less monitoring.

Effective application monitoring with Nagios enables organizations to quickly detect applications, services, or process problems, and take corrective action to prevent downtime for your application users.

Nagios tools for monitoring of applications and application state extends to Windows applications, Linux applications, Unix applications, and web applications. It has an active community collaboration network.

The router monitoring capabilities offer benefits such as immediate notification on unresponsive machines, early warning by detecting network outages and protocol failures, increased servers, services, and application availability.

Windows monitoring with Nagios enables increased servers, services, and application availability, quick detection of network outages, failed services, processes, batch jobs and protocol failures, and so on. The extensive metrics are gathered for system metrics, event logs, applications (IIS, Exchange, , and so on), services (Active Directory, DHCP, service states, process states, performance counters, and so on).

Nagios – enterprise server and network monitoring software

Built-in advanced features are:

- Integrated overview of sources, checks, network flow data, and so on, provided with comprehensive dashboard
- Alert of suspicious network activity by security and reliability network analyzer.
- Insights and drill down options on network traffic, bandwidth, overall network health, and so on, with advanced visualizations
- Monitor network usage of specific applications, custom application monitoring, custom queries, views, and reports are available
- Historical network flow data with subsets of network flow information through specialized views
- Abnormal activity alerts with automated alert system example bandwidth usage exceeds specified thresholds
- Integrated metrics of network analyzer server loads with hard disk space availability

Integrated dashboards for network analysis, monitoring, and bandwidth

The Nagios dashboard with multiple monitoring options such as source groups, Server CPU, disk usage, and so on, can be extended and customized with many more choices based on business requirements.

Summary

In the next chapter, we will discus on advanced topics of visualization, containers offered by multiple vendors, orchestration options, Internet of Things, microservices, and so on.

9
Containers, IoT, and Microservices

In this chapter, we will study the concepts of Containers, Virtualization, Kubernetes, Internet of Things, microservices, and so on.

- Virtualization:
    - Para virtualization
    - Container-based virtualization
- Introduction to containers:
    - Types of containers
    - Dockers
    - Java container services
    - Amazon container services
    - Pivotal container services
    - Google container services
- Container orchestration:
    - Kubernetes
    - Docker Swarm
    - Mesosphere
- IoT
- Microservices

Virtualization

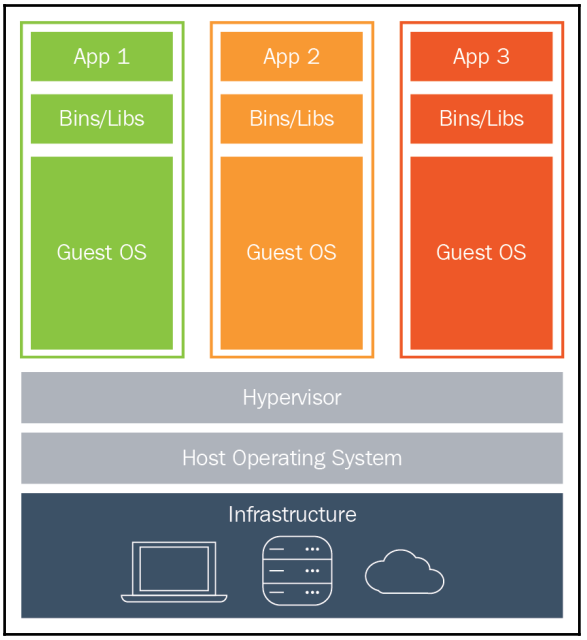
It is a method of logically dividing mainframes to simultaneously allow multiple applications to run concurrently. Bare metal applications were unable to cope up with the advancements to use abundance availability of resources such as the server processing power and capacity improvements. This paved way for designing **virtual machines** (**VMs**), by running specialized software on top of physical servers to emulate a type of underlying hardware system.

The same physical server can host multiple VM, each with different operating systems. Each VM runs a unique operating system and its own binaries/libraries and applications that it supports and services. VMs can be many gigabytes large. Server virtualization benefits are like a consolidation of applications onto a single system, with reduced server footprint, quicker server provisioning, improved disaster recovery, and cost savings.

Hypervisor

A hypervisor is software, firmware, or hardware that creates and runs on VMs also named as a virtual machine monitor. It is a layer between the OS and hardware to virtualize the server.

The hypervisor creates the virtual environment to host the guest VM. It supervises the guest systems and allocates resources to them as required. The hypervisor emulates the operation on the host machine's operating system, and provides virtualization services to the VMs in between the physical machine and virtual machines:



Virtualization technologies have become popular with rapid development in cloud using hypervisors, such as Xen, VMware Player, and KVM, and in incorporation of hardware support in commodity processors, such as Intel VT and AMD-V.

Types of virtualization

The virtualization types are categorized depending on the way it mimics hardware to a guest operating system, and also emulates a guest operating environment.

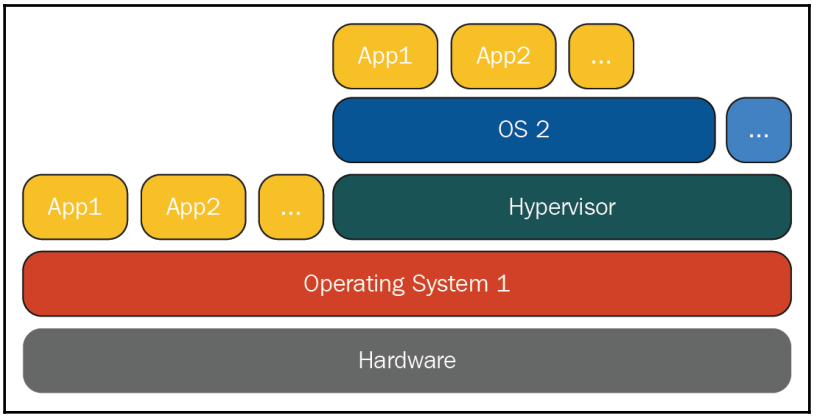
Primarily, there are three types of virtualization:

- Emulation
- Paravirtualization
- Container-based virtualization

Emulation

Emulation is full virtualization to run the OS kernel of VM entirely in software. This type of hypervisor is termed as *Type 2 hypervisor*, and is installed on the top of the host operating system in order to translate a guest OS kernel code to software instructions. There is no hardware involvement, and the translation is done entirely in software layer. By emulation to any operating system that supports the underlying environment is emulate; however, the overhead of additional system resource leads to performance reduction as compared with other types of virtualizations.

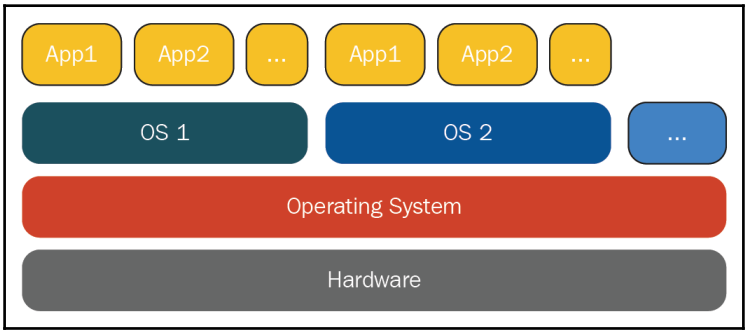
A few examples are VMware player, VirtualBox, QEMU, Bochs, parallels, and so on are shown in the following digram:



Paravirtualization

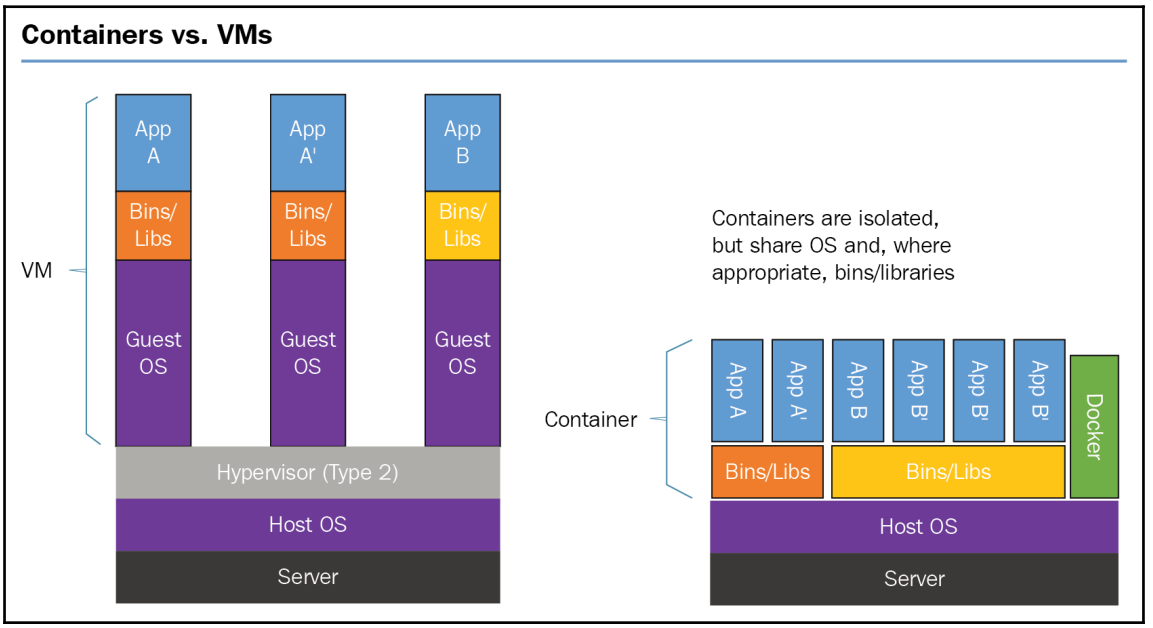
Paravirtualization, also referred to as *Type 1 hypervisor*, runs directly on the bare-metal hardware to provide virtualization services to the VM directly running on it. It supports collaboration between the operating system, the virtualized hardware, and the bare-metal hardware to accomplish optimal performance. These hypervisors do not require extensive resources, and typically run on small footprint.

A few examples are Xen, KVM, and so on:



Container-based virtualization

Container-based virtualization is also referred to as operating system-level virtualization; within a single operating system kernel, it enables executions of multiple isolated environments. This isolated virtual execution environment is called container-managed with a group of processes. It offers features and benefits such as high performance and dynamic resource management:

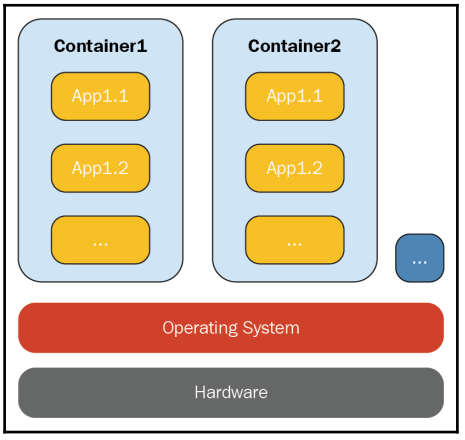


Containers

The concept of containers in IT is synonymous to the concept of containers used in the transportation sector. The basic purpose of containers is to carry or port items from source to destination.

Extending the same analogy, containers in IT operate on this basic purpose to port software from one server to another safely and securely. Moving an application from development server to QA (test) server and to production server is usually associated with multiple complexities, such as preparing infrastructure environment checklists, validating the compliers, libraries, runtime dependencies, and so on. The container concept is to ensure that it carries along with it the ecosystem required for an application to run from one bare-metal system to another. A container in that sense is self-sufficient, with all the requisite components, and the environment for the application to run on any server is installed. A container image is a standalone executable package, an abstraction that packages the code and dependencies at the application layer; they consume less space, and operate within the allocated resources.

Containers are a logical packaging mechanism to abstract the applications from the underlying environment they actually operate. Container-based applications because of decoupling can be deployed easily and consistently on a variety of target environments from private data center to the public cloud, or a personal laptop. Containerization separates roles, responsibilities, and concerns; this facilitates developers to work on the application logic and dependencies. IT operations teams, without diving into the specifics for software versions and configurations specific to the applications, can focus on pure deployment and management:

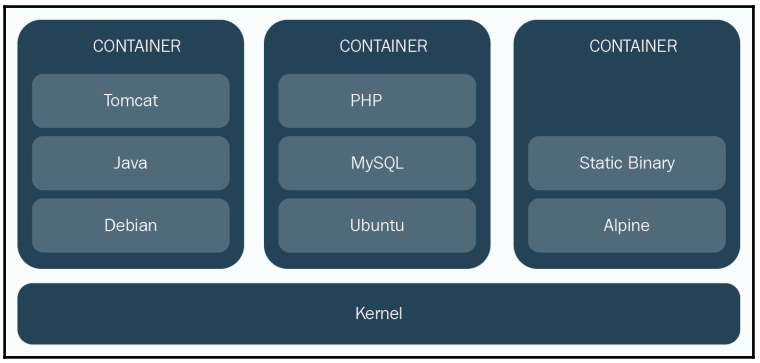


The attributes of a container making them quite a popular choice are:

- **Lightweight**: These containers are to use fewer resources and start instantly; many containers can share the operating system kernel and use less resources. They share common files, and images are built from filesystem layers to minimize disk consumption and faster image downtime.
- **Interoperability**: Containers should be operating on open standards, and portable between platforms of Linux distributions (Debian, Ubuntu, and CentOs), VMs, and Windows.
- **Security**: Containers offer a high degree of security by isolating and separating both in between the applications and with underlying infrastructure. The isolation of issues related to an application should be contained within the container, and it neither impacts the other applications, nor the entire machine.

Multiple tenancy that containers share a common kernel (system resources) through unique mount process. For each container, the commonly shared components are write-enabled, and they have read-only access. Due to this sharing concept, containers bring the following benefits:

- Being exceptionally lightweight
- Quick startup time
- Portability on a variety of cloud deployments, public, private, and so on
- Development and testing phases are accelerated by quick assembling (package) of applications with dependencies
- Management effort reduced to manage a single system compared with multiple servers, reduced effort for patches, and upgrades
- Guest OS and Host OS should be compatible, so both should match like Linux, neither should be run on Windows, and vice versa:



- **Container security**: The namespaces feature included with Linux kernel led to the creation of the concept of a container. Separate instances are created for global namespaces; the way it works is for every process the container adds a unique ID, and also for every system call, new access control checks are added. These serve as isolated containers wherein the outside objects have neither access nor visibility to the processes running inside the container. Although they share the underlying kernel, each container is treated separately; for example, the root user of each container is restricted for the respective container, adding a high security feature.
- **Container management**: The common kernel has full visibility among multiple containers, so it's imperative to limit the resource allocation to containers. This is accomplished by the Linux **Control Groups** (**cgroups**) subsystem for resource allocation and scheduling. Container virtualization is enabled by grouping processes to manage their aggregate resource consumption to limit memory and CPU consumption. The management tools to limit resource allocations for Linux containers are LXC, LXD, Linux-VServer, OpenVZ, Docker, systemd-nspawn, lmctfy, Warden, and so on.
- **Scalability**: Cluster management infrastructure supports scalability to install and operate, to schedule container-enabled applications across the cluster based on resource needs, and availability. It supports the ability to grow from single to multiple instances without additional complexity on how the applications, batch jobs, or microservices are managed, abstracting the complexity of infrastructure.
- **Task definitions**: Tasks are defined with declarative JSON template called **task definition**. Multiple containers required for the task are specified inside task definitions, and they include Docker repository/image, CPU, memory, and shared data volumes. The way containers are to be linked to each other can be part of a single task definition file registered with service. Version control for application specification is also part of task definition.
- **Programmatic control**: Container services provide APIs to integrate and extend services such as creation and deletion of clusters, registration and deregistration of Docker containers, including launch and termination tasks, and provide details on the cluster state and information of its instances.
- **Scheduling**: Applications, batch jobs, and services are managed with schedulers to run as they are designated. Schedulers place containers based on availability requirements and resource needs (such as RAM or CPU) onto the clusters. Provision for custom schedulers and integrate third-party schedulers, container management, and orchestration support with open source projects.
- **Container auto-recovery**: Containers automatically recover unhealthy containers, to ensure application support with the required number of containers.

- **Container deployments**: By uploading a newer version of application task definition, the scheduler will automatically stop the old version containers and start the containers with new versions. Ease of updating containers to new versions by conveniently registering and deregistering containers.
- **Load balancing**: Distribute traffic across containers with **Elastic Load Balancer** (**ELB**), by specifying in the task definition to add and remove containers from ELB. A dynamic port allocation in task definition gives unused port access to the container while scheduled; to share multiple services with ELB path-based routing is also an option.
- **Local development**: Docker composes an open source tool to define and run multicontainer applications; this can be extended to both work machine and production server.
- **Monitoring**: Clusters and containers to run tasks, average and aggregate of CPU and memory utilization, grouped by task definition, service or cluster, provision to set up alarms in order to alert to scale up or down the containers.
- **Logging**: The details of agent logs, API calls, and Docker logs are captured for issue diagnostics, the time of the API call, the API call source IP address, the request parameters, response elements, security analysis, compliance, audit, and resource change tracking.
- **Repository support**: Container service along with third-party, private Docker registry, or Docker hub image repository accessibility to retrieve appropriate images.
- **Security**: Inbuilt features to gain visibility into roles and access at task level, and manage instance-based roles and task-based roles separately with least privilege policy.

As we have seen, the containers are standardized units of software development, incorporating everything as code, runtime, system tools, and system libraries for software applications to run. The image is a read-only template to create containers. Application components must be architected to deploy and run in containers.

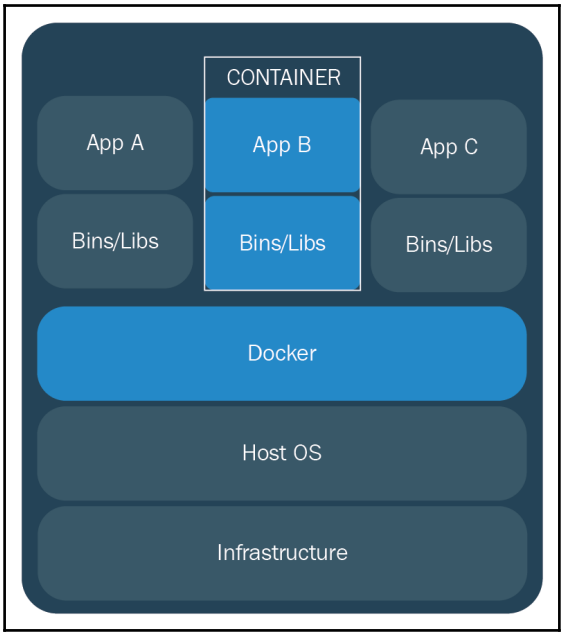
Containers have become popular in digital technology and offered by many vendors a like. Docker is a popular container choice, so its adoption and support is a natural choice by other vendors:

- Dockers
- Java container services
- Amazon
- Pivotal

- Azure
- Google
- IBM Bluemix container service

Docker containers

Docker container platform is available as **Community Edition** (**CE**) and **Enterprise Edition** (**EE**), with optimized installers for a variety of infrastructure:



The features of Docker container services are as follows:

- **Universal packaging**: This packages apps of any programming language or service into containers to port them, without the risk of incompatibilities, or version conflicts.
- **Developer toolkit**: Readily available containers in the Docker store offer everything needed to build, test, and run multicontainer apps for any programming language.

- **Container orchestration in-built**: In-built clustering at scale with sophisticated scheduling to monitor, build highly available and fault-tolerant services to run apps.
- **Highly secure**: Security out-of-the-box with mutual TLS, certificate rotation, container isolation, and image signing makes it secure and easy-to-use container app runtimes.
- **App-centric networking**: Containers connected together with software-defined networking to intelligent routes and load balances traffic. Container-defined networks abstract configuration and deploy apps from underlying network infrastructure.
- **Extensible architecture**: Integration with third-party systems for Open APIs, plugins, and drivers is convenient to change storage and networking backends with minimal or no code changes.

Java EE containers as a part of Java EE

Thin-client multi-tiered applications to handle transaction and state management, multithreading, resource pooling, and other complex low-level details involve many lines of intricate code and hard-to-maintain Java EE architecture is component-based and platform independent, ensuring convenience for Java EE applications, as business logic is organized into reusable components, and provides underlying services in the form of a container for every component type. Rather than investing to develop these services, concentrate on solving the business problem at hand.

Java containers are the interface between a component and the low-level platform-specific functionality that supports the component. For a web service, the enterprise beans, the application client component must be assembled into a Java EE module, and deployed into its container to be executed.

For the assembly process, container settings are specified for each component of the Java EE application, and also the Java EE application itself. Container settings can be customized with the underlying features of the Java EE server, such as transaction management, **Java Naming and Directory Interface** (**JNDI**) lookups, security, and remote connectivity.

Some features are:

- The Java EE security model-authorized users are configured access for a web component or enterprise.
- The Java EE transaction model treats all methods in one transaction as a single unit, by specifying relationships among methods to cluster them as a single transaction.
- A JNDI lookup service helps application components access a unified interface in the enterprise to the multiple naming and directory services.
- The Java EE remote connectivity model mimics VM by invoking methods to manage low-level communications between clients after enterprise beans are created.

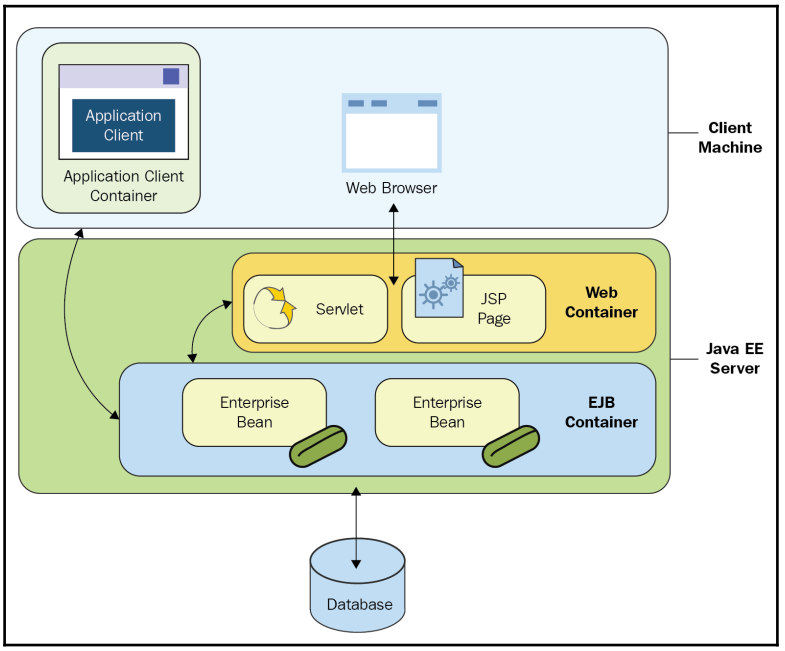
The Java EE architecture features configurable services, and application components within itself, so can provide different functionality like security settings levels of access to database data customized for each production environment. The Java EE container facilitates nonconfigurable services such as enterprise bean and servlet life cycles, data persistence, database connection, resource pooling, and access to the Java EE platform APIs.

Java EE server and containers

The container types in the Java EE application components are:

- **Java EE server**: This is the runtime environment of Java EE, and provides EJB and web containers.
- **Enterprise JavaBeans (EJB) container**: It's part of Java EE server, and it's responsible for the execution of enterprise beans for Java EE applications.
- **Web container**: They are part of Java EE server. They manage the execution of JSP page and servlet components for Java EE applications. Web components and their containers run on the Java EE server.
- **Application client container**: This hosts application clients and their containers' execution.

- **Applet container**: This is responsible for execution of applets, and hosts web browser and Java plugin together to run on the client.



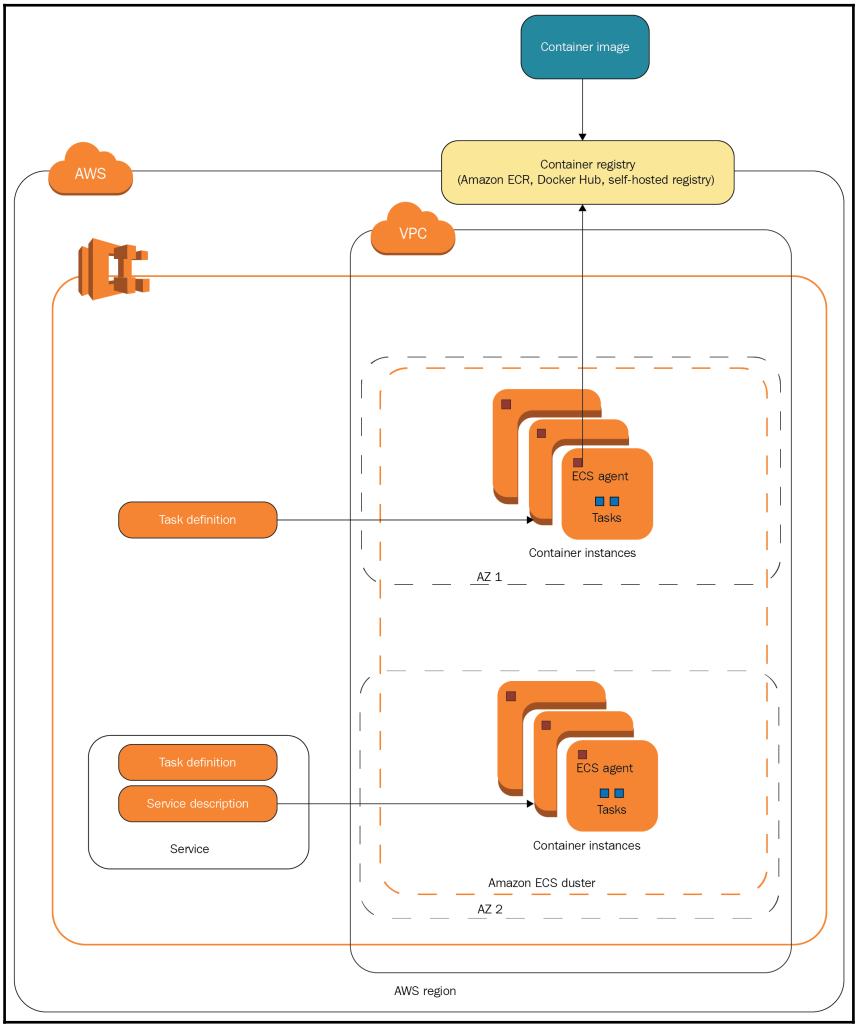
Amazon ECS container service

Amazon ECS comes with a variety of following features:

- Highly scalable, fast container management service
- Manage Docker containers are easy to run, stop on a cluster of **Amazon Elastic Compute Cloud** (**Amazon EC2**) instances
- Container-based applications launch and stop through simple API calls
- Centralized service to monitor state of your cluster, and access to many familiar Amazon EC2 features
- Schedule the containers placement across the clusters based on resource needs, isolation policies, and availability requirements
- Consistent deployment and build experience, manage scale **Extract-Transform-Load** (**ETL**) and batch workloads

- Build sophisticated microservices and model-based application architectures
- Provides more fine-grained control and access to a wider set of use cases, hence more popular compared with AWS Elastic Beanstalk

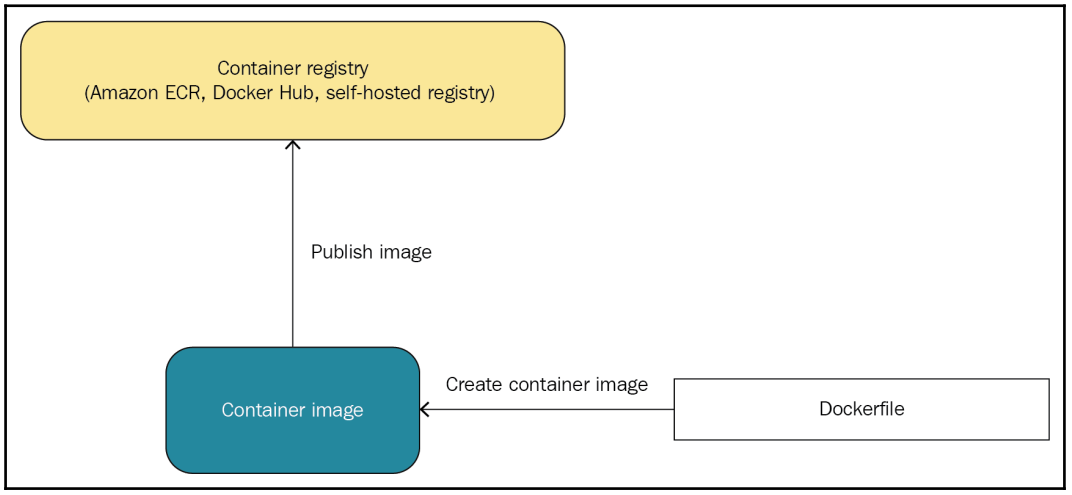
Amazon ECS is also highly available, which supports the concept of running application containers across multiple availability zones. Container images are stored in and pulled from container registries (either within or outside of AWS). Task definitions and services are defined to specify alignment of Docker container images to run on respective clusters:



The Amazon ECS multizone architecture components are discussed further.

Containers and images

Application components must be architected to run in containers for deployment to ECS. The standardized unit is Docker container for software development, comprising application requirements to run code, runtime, system tools, system libraries, and so on. An image template is used to create containers, and it specifies all of the components to be part of the container. It is typically a read-only format file created from a plain text Dockerfile, stored in a registry for download and run on container instances:

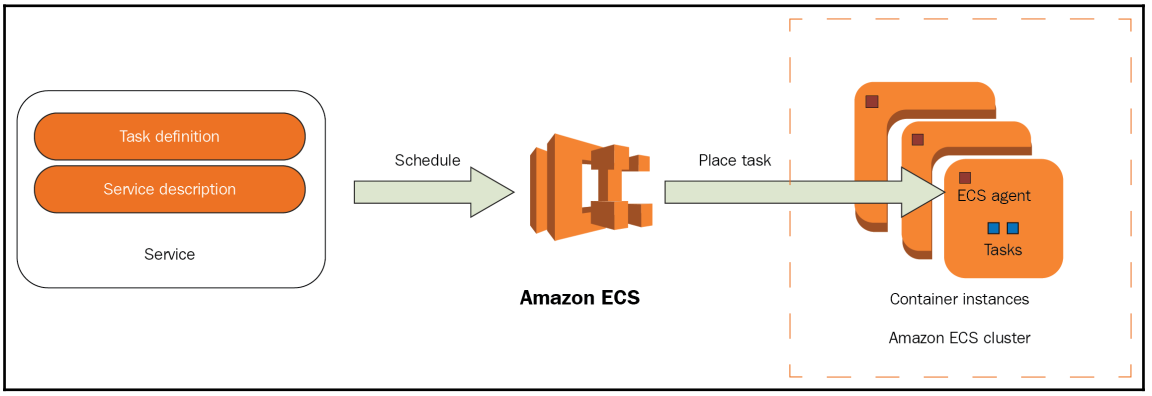


Task definitions

Task definitions prepare the application to run on ECS as a blueprint for the application, a JSON format text file. It describes the application use by various parameters as to which containers to use, the corresponding ports to be opened, the repositories to be located, and the data volumes to be used with the containers for the tasks on a web server such as Nginx.

Tasks and scheduling

With task definition, the number of tasks are specified for instantiation on a container instance within the cluster. Task scheduler is responsible for placing and scheduling tasks on container instances simultaneously:

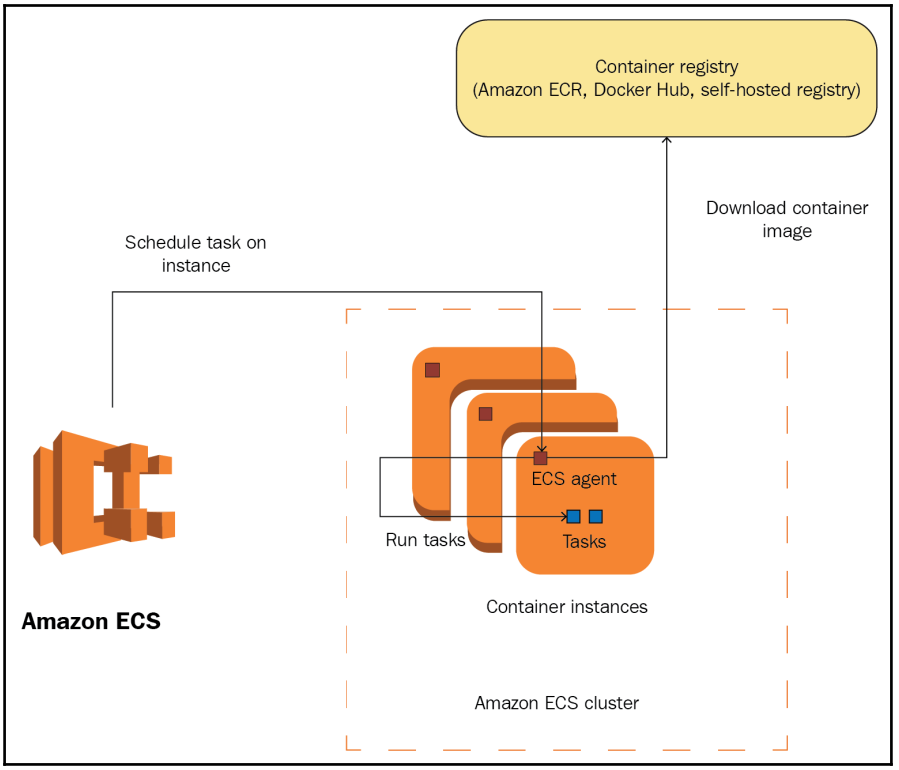


Clusters

A cluster is a logical grouping of instances to run tasks. The container images are downloaded from a specified registry to run them on the container instances within the cluster.

Container agent

The container agent runs on each instance within the cluster. It relays the instance's information such as current running tasks and resource utilization to also start and stop tasks as per the requests:



Amazon ECS functionality can be augmented using additional services in conjunction:

- **Identity and access management**: IAM is a web service to control access to resources securely for users by authentication to control use of resources and authorization on how to access resources. Using IAM roles control access at the container instance level, and also the task level are managed.
- **Autoscaling**: Autoscaling is a web service to scale out and scale in the container instances, automatically launching or terminating EC2 instances. It could be defined in user-defined policies, health status checks, and schedules.
- **Elastic Load Balancing**: ELB accomplishes high levels of fault tolerance for applications by distributing the incoming application traffic automatically across multiple EC2 instances, and across services in a cluster.
- **EC2 Container Registry:** Docker registry service is secure, scalable, and reliable. Resource-based permissions enabled by IAM to access repositories and images on Docker private repositories. Docker CLI can be used to push, pull, and manage images.

- **AWS Cloud Formation**: AWS Cloud Formation is a convenient way to create, manage, provision, and update a collection of related AWS resources. Cloud Formation script can help define clusters, task definitions, and services in an orderly and predictable fashion as entities in an AWS.
- **Amazon ECS CLI**: Amazon ECS CLI using Docker compose from a local development environment. It provides high-level commands to create, update, and monitor clusters and tasks.
- **AWS SDKs**: The SDKs support programming languages, and take care of tasks automatically such as:
    - Signing of service requests cryptographically
    - Retrying requests
    - Error responses handling

Pivotal container services

Pivotal technologies offer container services with the following core features:

- Containerized workloads are reliably deployed and run across private and public clouds.
- Container orchestration is built in with high availability, automated health checks, monitoring, and so on.
- Suited for Spark and elastic search workloads needing access to infrastructure primitives.
- Apt for apps that require specific colocation of container instances, and where multiple port binds are needed.
- High operational efficiency for Kubernetes and Open Source Kubernetes with latest stable OSS distribution no proprietary extensions.
- On-demand provisioning with BOSH, a powerful release engineering tool chain, a reliable and consistent operational experience on any cloud.
- Multicloud flexibility to deploy and consume Kubernetes on-premises with vSphere, or in the public cloud.
- Network Management and Security to programmatically manage software-based virtual networks, out-of-the-box network virtualization on vSphere and VMC.
- Fully automated Ops can deploy, scale, patch, and upgrade the system without downtime also for Kubernetes clusters.

- Easy integration with VMware tools like vRealize Operations Manager, vSAN network storage, and Wavefront for a full-featured on-prem deployment.
- Harbor, an enterprise-class container registry server, is part of PKS. Harbor extends features for an open source Docker like vulnerability scanning, identity management, and support for multiple registries.
- Integrated with the PCF Service Catalog, easily adds APM tools, database services, and the Service Broker API. Extend PKS with a growing library of add-on services.
- Constant compatibility with GKE, can easily move workloads to (and from) GKE.
- PKS is built on top Kubo, an open-source project managed by the Cloud Foundry Foundation.

Google container services

Google container services is a popular choice, with features as listed here:

- Google containerized application management offers multiple advanced features and flexibility
- Container engine provides a managed environment for containerized applications to deploy, manage, and scale on a container cluster
- Kubernetes open source cluster management system powers container cluster engine
- Provides tools and interface to perform administration tasks, manage, deploy applications, and set up policies
- Monitors status of application containers, the health of deployed workloads
- Load-balancing for compute engine instances
- Node pools within a cluster as a subset for additional flexibility
- Automatic scaling of cluster's node instances
- Automatic upgrades for cluster's node software versions
- Auto-repair of node to maintain node health and availability
- Stack driver logging and monitoring for visibility into your cluster
- Master and node architecture for the cluster
- IP aliases, IAM, role-based access, and so on
- IP rotation and IP Masquerade agent

Container orchestration

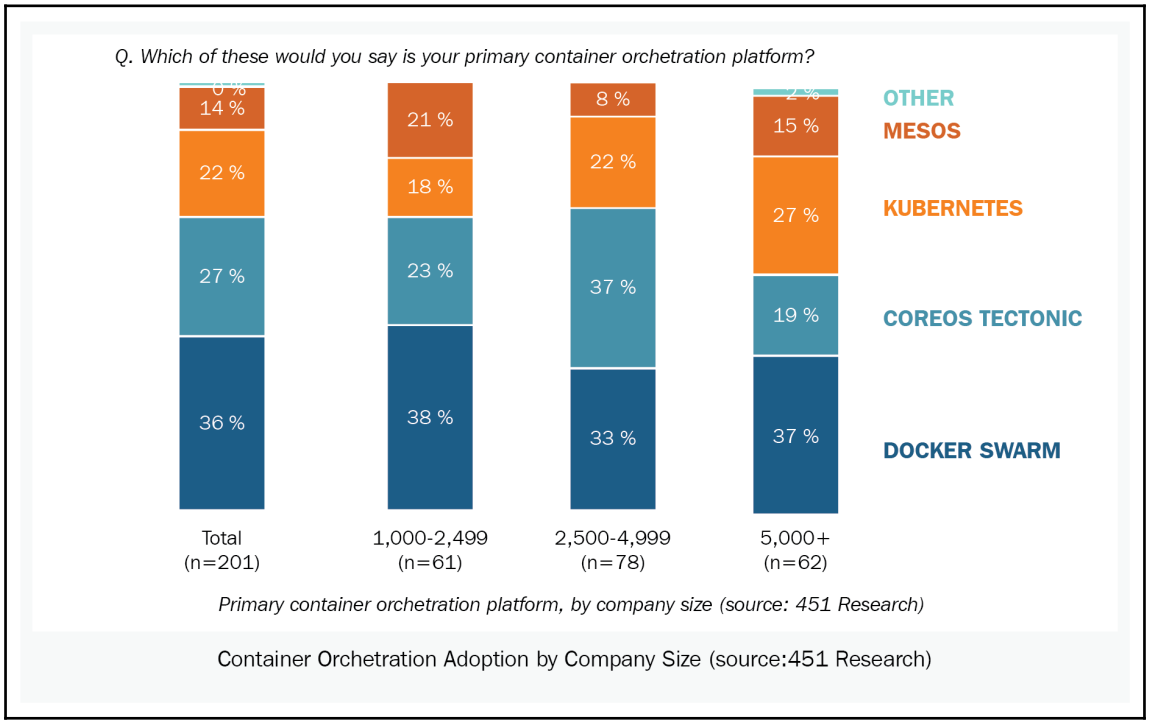
Container orchestration is the process of automatically deploying multiple containers in an optimized manner to implement an application. This is quite important with a growing number of containers and hosts day by day. Orchestration means the automation of the process, and includes a number of features:

- Hosts provisioning
- Set of containers instantiation
- Failed containers rescheduling
- Containers linking together with interfaces
- Exposing services outside of the cluster to machines
- Scaling by adding or removing containers, out or down the cluster

Orchestration tools

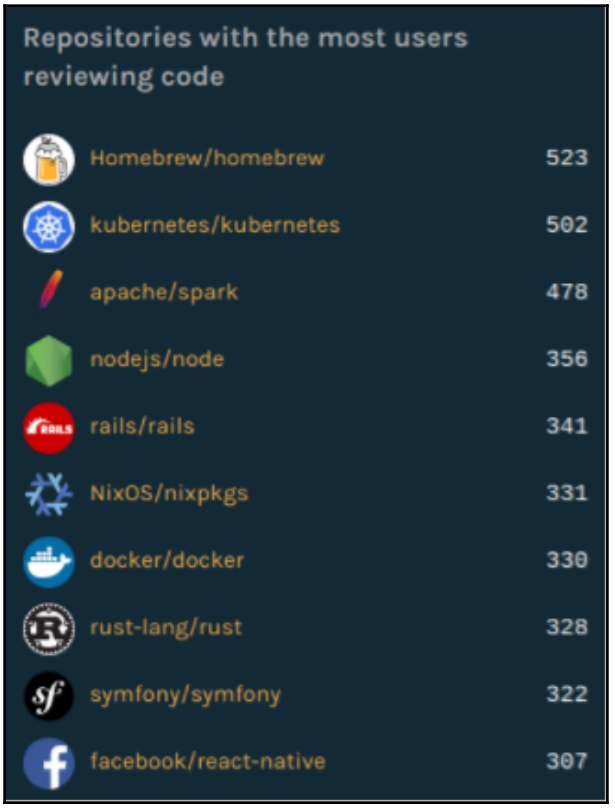
A few popular orchestration tools are listed here:

- Mesos
- Kubernetics
- CorCos Tectonic
- Docker Swarm



Container Orchetration Adoption by Company Size (source:451 Research)

The popularity of repositories as per their usage is shown in following image:



The three key differentiators among the orchestration tools to select for organization are:

- **Level of abstraction**: Support for containers or services that are container-based
- **Tooling**: Orchestration management and integration with other services
- **Architecture**: How does it support scalability and recover from failure?

We will discuss the following popular tools in detail:

- Kubernetes
- Docker Swarm
- Mesosphere

Each orchestration platform has advantages compared with the others. There are multiple evaluations to consider, such as:

- Enterprise DevOps framework and orchestration methodology along with APIs
- Number of hosts if over thousands of physical machines
    - Mesos can be considered for large farm
- Are the containers based on bare metal, private VMs, or in the cloud?
    - For cloud deployments, Kubernetes is popular
- Need for automated high availability
    - Kubernetes failed pods/containers will be automatically rescheduled by replication controller
    - In Mesos application's framework, code performs that role for automated high availability
- Grouping and load balancing requirement for services
    - Kubernetes provides this, but Mesos application's framework code performs it
- Organization skills
    - Mesos allow application to run as a framework programmatically with custom coding
    - Kubernetes is more declarative
- Setting up orchestration frameworks infrastructure can have challenges

Kubernetes

Kubernetes was created by Google, and now with **Cloud Native Computing Foundation** (**CNCF**). Its concept is to build orchestration for container deployments across multiple domains public clouds to hybrid deployments.

Kubernetes makes deploying and managing application easier, automates deployment, scaling, and management of containerized applications. Kubernetes adds the higher-level functions, such as load balancing, high availability through failover (rescheduling), and elastic scaling, as follows:

- Automatic health checks against the services
- Self-healing, restarting containers that fail, or have stalled
- Horizontal scaling, scale services up or down based on utilization
- Service discovery and load balancing

- Version-controlled automated rollouts and rollbacks
- Secret and configuration management
- Storage orchestration only running what is needed
- Batch execution, declaratively manages your cluster

The key components making up Kubernetes are:

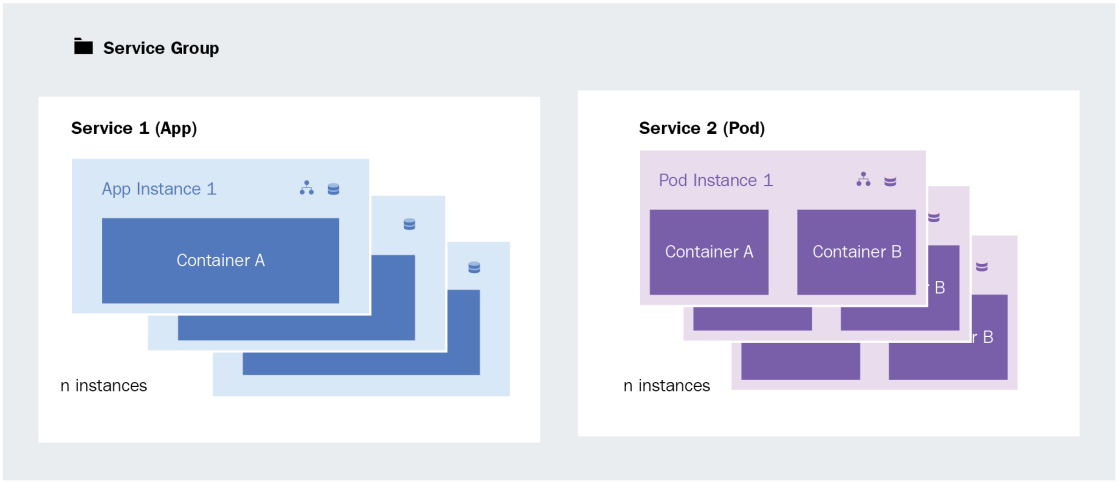
- A cluster is a collection of nodes, either bare-metal servers or VMs providing the resources to run one or more applications.
- Pods are co-located resources on the same host groups, such as containers and volumes. Pod-based containers share the same network namespace and use localhost to communicate. Pods are the basic scheduling unit, ephemeral not designed as durable entities.
- Labels are tags that allow them to be managed as a group. Like assigned to entities such as containers to be exposed as a service to the outside world.
- Services are basic load balancers and references for exposing them to the outside world and other containers.
- The replication controller manages the scheduling of pods across the cluster.

Docker orchestration tools

Docker orchestration most common tools are described here:

- **Docker Swarm**: It is one of the most easy-to-use orchestrators, with just a couple of commands. It lets you spin up your first cluster much like the first container.
- **Docker Engine**: It is the lightweight runtime and tooling engine used to run Docker containers.
- **Docker Machine**: This provisions hosts and installs Docker Engine software on them.
- **Docker Swarm**: By clustering multiple Docker hosts together produces a single, virtual Docker host. It enables Docker API to integrate with tools compatible with a single Docker host.
- **Docker Compose**: Applicable for development, testing, and staging environments. Creates required containers for deploying requisite application from a file defining a multicontainer application along with its dependencies.

- **Apache Mesos**: It is adopted by large enterprises, such as Twitter, Airbnb, and Apple, as it's designed to scale to tens of thousands of physical machines. A framework in Mesos is an application running on one or more containers. Each frame can accept the resources offered by Mesos. Compared to Kubernetes, Mesos is less in features, involves extra integration work, is programmatic, defining services or batch jobs. Mesos also supports the fine-grained resource allocation across the nodes in a cluster of Kubernetes pods:



Mesophere DC/OS Pods (Image Credit: Mesosphere Blog)

Mesos is proved to be efficient wherein the application is collocated with other services such as Hadoop, Kafka, and Spark. Mesos is the foundation for few distributed systems as follows:

- **Apache Aurora**: A highly scalable scheduler service for long-running services and cronjobs such as adding rolling updates, service registration, and resource quotas
- **Chronos**: A fault-tolerant service scheduler for orchestrating scheduled jobs within Mesos as a replacement for Cron
- **Marathon**: A simple-to-use service scheduler; it enhances performance of Mesos and Chronos by running two Chronos instances simultaneously

Internet of Things (IoT)

IoT integrates sensory data, big data, networking, robotics, and artificial intelligence technology into an advanced automation and analytics system. IoT, when applied to any industry or system, can bring in greater transparency, control, and performance to deliver a complete product or service.

IoT systems span across industries using smart devices and enabling powerful technology to enhance data collection, analysis, achieve deeper automation, operations and integration through applications apt in any environment.

IoT benefits span across multiple business domains and even lifestyles providing benefits of improved customer engagement, technology optimization, reduced waste, enhanced data collection, and so on.

IoT-perceived challenges are as follows:

- **Security**: An ecosystem of constantly connected devices communicating over networks even with security measures is vulnerable
- **Privacy**: Substantial personal data is captured without the user's active participation
- **Complexity**: Design, deployment, and maintenance of IoT systems integrating multiple technologies is a complex system
- **Flexibility**: IoT systems with several interfacing and locked systems are tightly coupled
- **Compliance**: When standard software compliance to comply with regulations is challenging, complexity of IoT makes the issue of regulatory compliance much more challenging

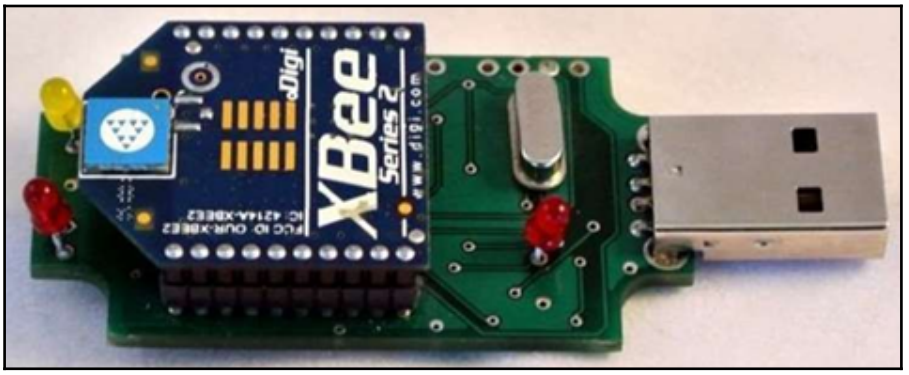
The most important features of IoT include connectivity, sensors, active engagement, and being a small device and artificial intelligence combination:

- **Connectivity**: New enabling technologies between system devices, specifically IoT networking on a much smaller and cheaper scale, need not be exclusively tied to major providers.
- **Sensors**: IoT capability comes with sensors that transform IoT from a standard passive network of devices into an interactive integrated system to address real-world needs
- **Active engagement**: IoT introduces real-time interaction with connected technology, a new paradigm for active content, product, and service engagement

- **Small devices**: Purpose-built small devices extend IoT capabilities to deliver its precision, scalability, and versatility at low, affordable costs
- **Artificial intelligence**: IoT essentially enhances every aspect of life with the power of data collection and analytics with artificial intelligence algorithms

IoT - eco system

IoT systems capture data with hardware devices such as remote dashboard, control devices, sensors, servers, and routing bridge device. Key tasks and functions managed with these devices can extend to system activation, security, communication, action, and detection.



Multiple devices sensors for different functions are:

- Accelerometers
- Magnetometers
- Gyroscopes
- Acoustic sensors
- Pressure sensors
- Humidity sensors
- Temperature sensors
- Proximity sensors
- Image sensors
- Light sensors
- Gas RFID sensors

Standard devices

The standard devices such as desktop, tablet, and cell phone are also integrated with IoT for command interfaces and remote management.

Desktop and tablet can offer the highest level of control for the system and its settings.

Cell phone can also provide remote functionality to modify some settings or configurations.

Routers and switches are standard network devices, and key to connected devices.

Data synthesis

IoT systems effectiveness is through data collection, device integration, real-time analytics, networking, and action through platforms, embedded systems, partner systems, and middleware. These individual and master applications are responsible for integration with critical business systems, such as ordering systems, robotics, and scheduling within the IoT network.

Data collection

The data collection software collects and eventually transmits all collected data to a central server. It collects variety of data such as sensory data, measurements by applying data filtering, data security, and aggregation of data. Through protocols, data from multiple devices and sensors are connected real time with machine-to-machine networks. It also can reverse transmit distributing data back to the devices.

Device integration

Integration of all connected system devices through dependency and relationships binds the IoT ecosystem. It ensures that the necessary cooperation manages the various applications, protocols, and limitations of each device to allow communication and stable networking between devices.

Real-time analytics

The analytics applications collect real-time data input from various devices, and convert it into clear patterns for human analysis and viable actions. The information analysis and visualization techniques can be extended for automation-related tasks specific to industry requirements.

Application and process extension

These IoT applications integrate predefined devices to extend the reach of existing systems such as allowing certain mobile devices, or engineering instruments access and software to allow a wider, more effective system to improved productivity and more accurate data collection and analysis.

Technology and protocols

IoT technologies, apart from standard networking protocols, are RFID, NFC, low-energy radio protocols, low-energy Bluetooth, low-energy wireless, LTE-A, and WiFi-Direct, all of which support the specific networking functionality needed in an IoT system. We will review these technologies.

Radio-frequency identification (**RFID**) and **near-field communication** (**NFC**) are simple, low-energy options connection bootstrapping, and payments for identity and access tokens.

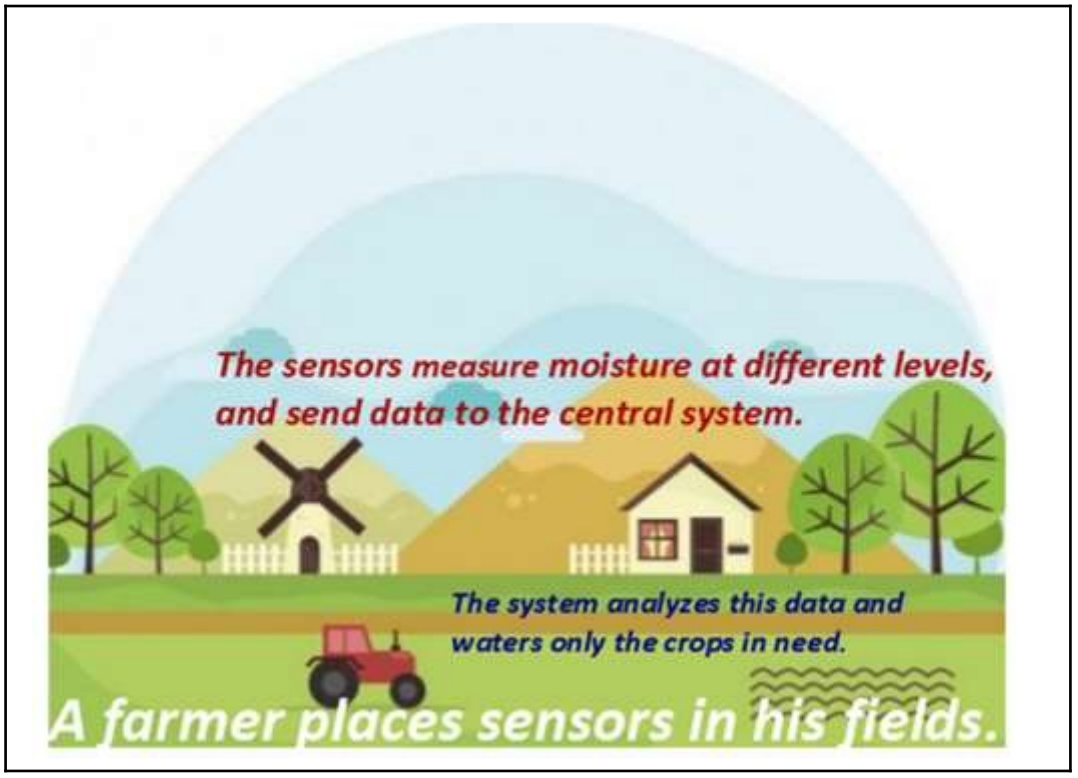
More information on the IoT technologies and Protocols can be found at: `https://www.tutorialspoint.com/internet_of_things/internet_of_things_quick_guide.htm`

IoT - application in multiple fields

The following are the applications of IoT across multiple fields:

- **Wearable electronics**: The penetration of IoT smart wearable electronics, such as helmets, watches, shoes, and glasses

- **Manufacturing and engineering industry**: Dynamic response to market demands, malfunctions in equipment, problems in the distribution network, customer needs, nonconforming product, lower costs, optimized resource use, and waste reduction
- **Product safety**: Avoid malfunctions, nonconforming product, and other hazards, avoiding recalls, and controlling nonconforming or product distribution to market
- **Healthcare applications**: Healthcare in remote areas can be extended by IoT applications to offer high level of medical assistance as in developed areas supporting mobile clinics
- **Housing, environment, health, and safety applications**: Also use IoT to extend their productivity, benefits for improved quality of life
- **Transportation application**: Extend to commercial vehicles on road, trains, UAVs provide improved communication, control, and data distribution
- **Commercial farming**: Exploiting advanced biotechnology, IoT enables deeper automation and analysis



IoT platforms for development

There are many IoT development platforms with integrated development tools to support connectivity, analysis for the rapid development, and deployment of smart, connected devices:

- ThingWorx by PTC
- Virtualized Packet Core by Cisco
- Electric Imp- Salesforce
- Predix by GE
- Eclipse IoT
- Contiki is open source

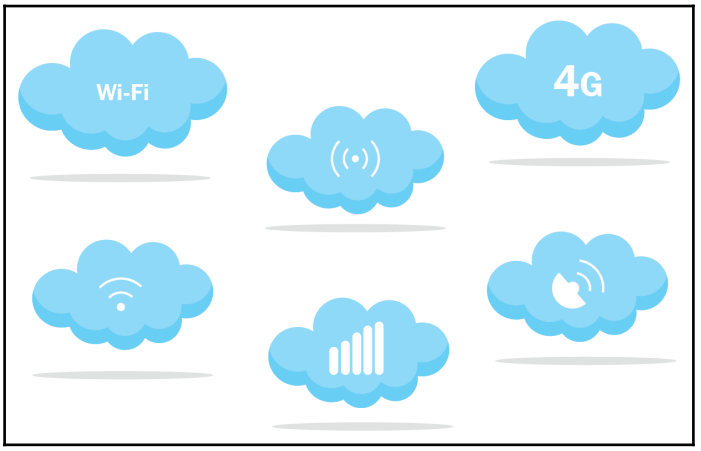
ThingWorx

Enables rapid development with interfaces and embedded tools such as Vuforia, Kepware, Composer, Mashup builder, search engine for storage, collaboration, and connectivity:

- Vuforia for reality development
- Kepware for single-point data distribution to facilitate interoperability in alignment
- ThingWorx agent for industrial connectivity
- The composer is the modeling environment for design testing
- The Mashup builder is a dashboard to build components
- SQUEL is the search engine, extension means search, query, and analysis; it's for analyzing and filtering data
- Data shapes describe data structures of custom events, infotables, streams, and data tables
- Thing templates allow new devices to inherit properties in large IoT systems
- Thing shapes define templates, properties, or execute services, allowing developers to avoid repeating device property definitions

Virtualized Packet Core (VPC)

VPC technology provides core services for 4G, 3G, 2G, Wi-Fi, and small cell networks with key features such as packet core service consolidation, dynamic scaling, and system agility. The networking functionality is delivered as virtualized services for greater scalability and faster deployment at a reduced cost of new services. It distributes and manages packet core functions, whether virtual or physical, across all resources:



VPC application is more prominent for network function virtualization, **software-defined networking** (**SDN**), and rapid networked system deployment by supporting low-power, high-flow networking, and the simple deployment of a wide variety of small devices. VPC introduces direct communication over a standard network, enhanced automated monitoring, automatic data updates through smart signs, and native IP networks along with **Power over Ethernet** (**PoE**) technology for all devices, improving overall safety and quality of service.

Electric Imp

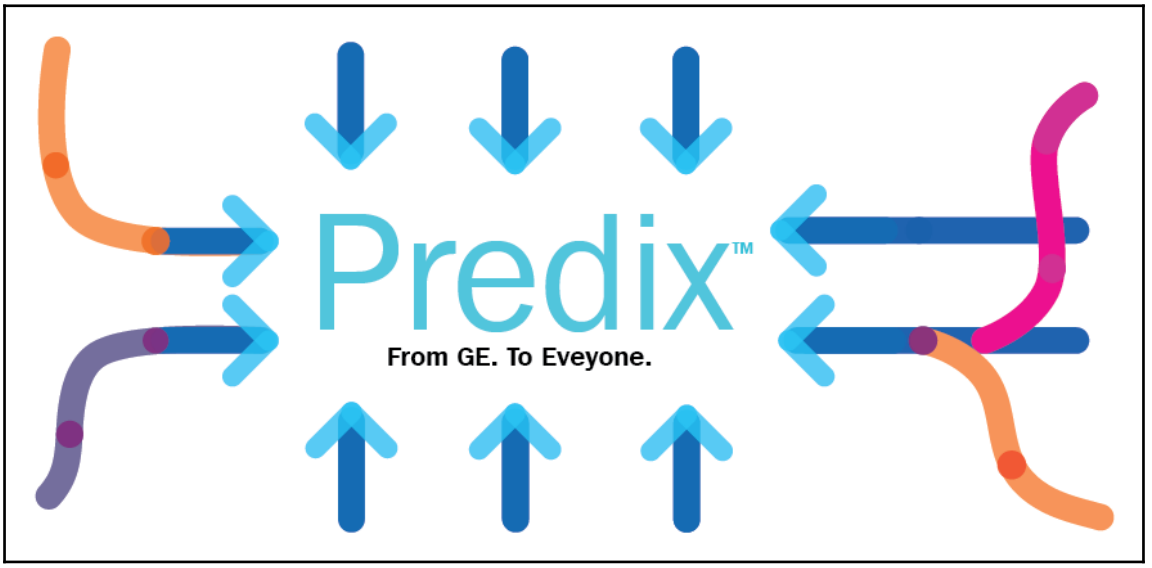
Salesforce Electric Imp platform is for quickly connecting devices to the cloud, and developing applications through a high-level, OO, lightweight scripting language named Squirrel language. Applications consist of two modules:

- The device module runs on the device
- The agent module runs in the Electric Imp cloud

The Electric Imp platform ensures secure communication messages with a simple call between the modules, standard web application development coding for device interaction, monitoring, and response with a simple, easy-to-learn syntax.

Predix

General Electric (**GE**) Predix is a software platform for industrial instruments. It's a cloud-based **Platform as a Service** (**PasS**), data collection platforms to enable industrial-grade analytics. It connects factories data, individuals, and equipment in a simple way for operations optimization and performance management. A predix ecosystem consists of an Intel Edison processor module, of a dual core board and a Raspberry Pi board. Developers provide an IP address, Ethernet connection, power supply to automatically establish the connection, register with the central Predix system, to transmit data from sensors.



Eclipse IoT

An open source technology-based Eclipse IoT is an ecosystem of entities (industry and academia), creating open source frameworks and services for utilization in IoT solutions, developing tools for IoT developers, open source implementations of IoT standard technology. There are a few utilities, as mentioned next:

SmartHome

Eclipse IoT's major service SmartHome is a framework for building smart home solutions, with assorted protocols and standards integration for heterogeneous environments. It facilitates interaction between devices by uniform device and information access consisting of OSGi bundles to deploy in an OSGi runtime, with OSGi services. OSGi bundles are Java class groups and other resources with manifest files containing information on file contents, services to enhance class behavior, and the nature of the aggregate as a component, and so on.

Eclipse SCADA

Eclipse state-of-the-art open source SCADA is to connect various industrial instruments with a shared communication system to post-processes data. The technologies incorporated are shell applications, JDBC, Modbus TCP and RTU, Simatic S7 PLC, OPC, and SNMP. The SCADA system is with communication service, monitoring system, archive, and data visualization for developing custom solutions.

Contiki

Open source operating system Contiki provides functionality for small IoT devices for management of programs, processes, resources, memory, and communication. Its ecosystem is an operating system, a web browser, web server, calculator, shell, telnet client and daemon, email client, VNC viewer, and FTP.

Its popular with academics, organization researchers, and professionals being very lightweight quite apt for devices with limited memory, power, bandwidth, and processing power requires a few kilobytes to run, and within a space of under 30 KB.

Contiki communication

Standard protocols supported by Contiki, and also enabling protocols for IoT include:

- **uIP (for IPv4)**: This TCP/IP supports 8-bit and 16-bit microcontrollers.
- **uIPv6 (for IPv6)**: Extension to uIP is a fully compliant IPv6.
- **Rime**: This offers a set of primitives for low-power systems and alternative stack when IPv4 or IPv6 are not applicable.
- **6LoWPAN**: This stands for IPv6 over low-power wireless personal area networks. A low data rate wireless compression technology to support devices with limited resources.
- **RPL**: This distance vector IPv6 protocol can find the best path for devices with varied capability in complex network of LLNs (low-power and lossy networks).
- **CoAP**: This protocol is for simple devices requiring a heavy remote supervisor.

Dynamic module loading

Dynamic module loader loads, relocates, and links ELF files to load and link at run-time supports environments to support application behavior changes after deployment.

The Cooja network simulator

The Cooja Contiki network simulator spawns to compile for working Contiki and control system by Cooja simulator.

IoT devices, security, compliance and maintenance are important features to be thoroughly considered while adopting.

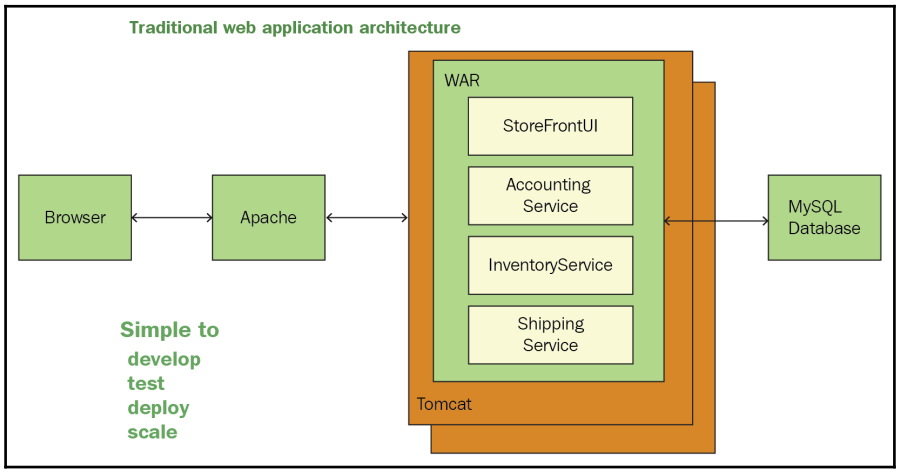
Microservices

It is an architecture pattern to structure as loosely coupled services to implement business capabilities, enabling an organization to evolve its technology stack on continuous delivery/deployment of large, complex applications.

Microservices core patterns

Microservice architecture is the core differentiator compared to monolithic architecture.

Monolithic architecture was based on unique requirement for building server-side enterprise application. It has to support a variety of clients, such as browsers from desktop, mobile, expose itself to third-party, and integrate with other applications through web services or message broker. Business logic is executed by handling HTTP requests and messages with a database, and returning a HTML/JSON/XML response:



The challenges associated with such architecture are:

- The large monolithic code base is difficult to maintain, modularity breaks down over time because there are no hard module boundaries; hence, implementing a change becomes cumbersome over time.
- **Overloaded IDE**: The slower the IDE with the larger the code base lower productivity.
- **Overloaded web container**: The larger the application, the longer it takes the container to start up, deployment lowers developer productivity too.
- **Continuous deployment is difficult**: Large monolithic application frequent deployments for updates are challenge. The user interface need to be developed iterative and redeployed frequently, the risk associated with redeployment increases.
- **Scaling the application can be difficult**: A monolithic architecture can only scale in one dimension with an increasing transaction volume by running more copies of the application, adjusting the number of instances dynamically based on load. However, with an increasing data volume, the architecture can't scale. Each application instance copy will access all of the data, making the caching less effective and increasing memory consumption and I/O traffic. With a monolithic architecture scaling, each component independently for different resource requirements will be challenge such as CPU intensive and might memory intensive.
- **Obstacle to scaling development**: A monolithic application prevents the teams from working independently, so it should be coordinated development between the UI team, accounting team, inventory team, and so on. Once the application gets to a certain size, it is an obstacle to scalability to develop involving multiple teams.
- **Long-term commitment to a technology stack**: A monolithic architecture could be tied to a technology stack, and a particular version of upgrading with some newer technology framework will be tedious. If platform framework subsequently becomes obsolete, then to adopt a newer platform framework rewriting the entire application could be a risky proposal.

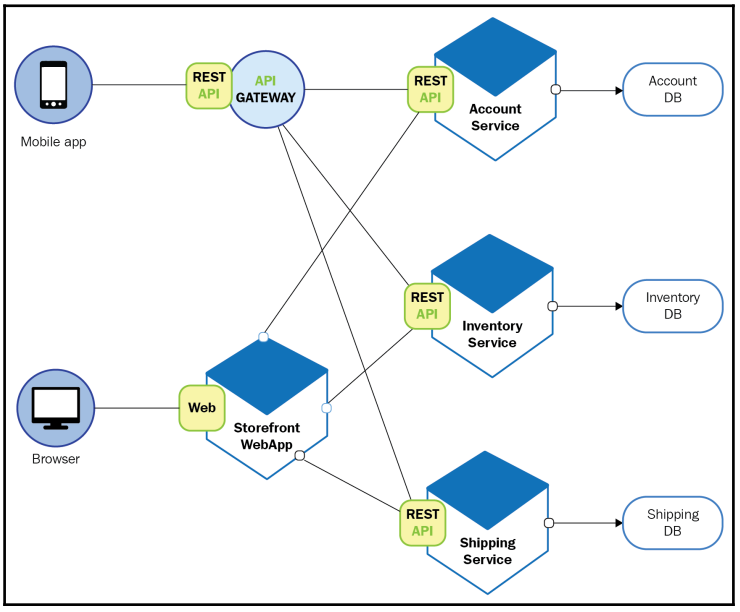
Microservices architecture

Microservices are a set of loosely coupled collaborating services that are the building blocks of applications. Each service implements a set of narrowly, related functions such as order management and the customer management services. The services communicate with each other with synchronous protocols such as HTTP/REST, or asynchronous protocols such as AMQP. The services are independently developed and deployed with its own database decoupled from other services with data consistency enforced.

The divers for microservices architecture are:

- To build application quickly, easy to understand, maintain and continuous deployment.
- Scalability and availability to run on multiple machines, multiple copies of the application
- Emerging technologies adoption for frameworks, programming languages, and so on

An application built on these lines could be of several components such as StoreFrontU to implement the user interface, backend services to check credit, maintaining inventory, and shipping orders. This can take orders from customers, check inventory, credit availability, and ship them:



This solution has a number of advantages:

- Each microservice is relatively nimble to understand, and can build faster and deploy; the IDE is faster and more productive
- Each service can be developed by their respective teams, and deployed of other services independently
- Frequent deployment for new versions is easier
- Improved fault isolation any service memory leak will affect only that service, and other services will continue to handle requests unaffected
- Flexible to adopt new technology stack

Challenges associated with microservices-based solutions are:

- Complexity of creating distributed systems with multiple services transactions.
- IDE support and testing is difficult, including the inter-service communication mechanism for coordinating multiple services between the teams.
- Deployment/operational complexity of deploying and managing a system comprised of many different service types.
- Increased memory usage: The microservice architecture runs its own VM to isolate the instances. If there are *M* instances, there will be *M* times the VMs causing the overhead.

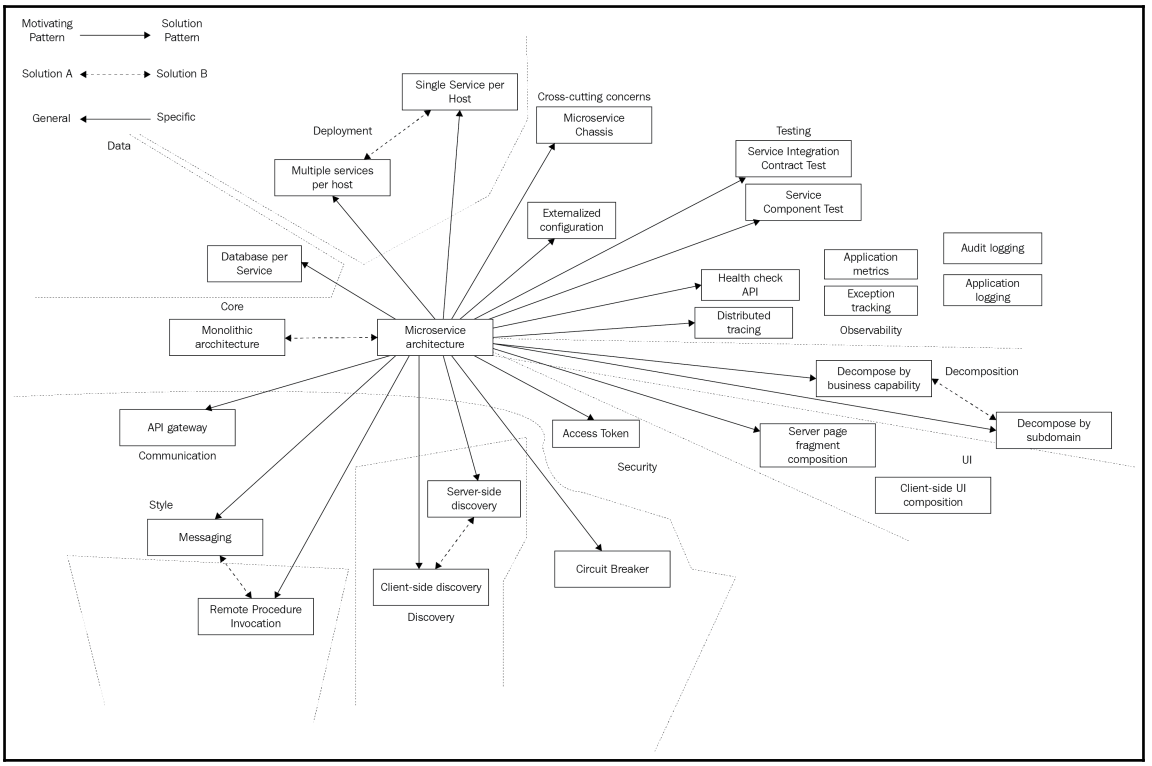
Microservice decision

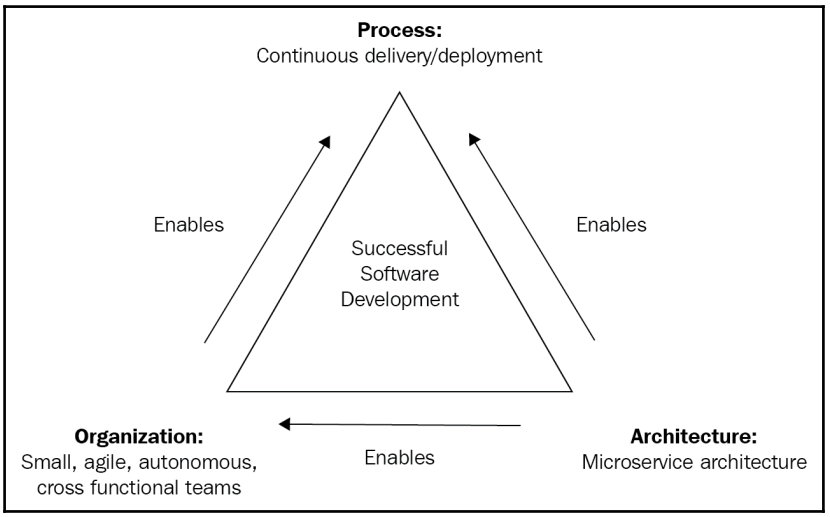
Microservice architecture is more appropriate for larger, complex scale of application, rather than small or startup applications where in monolithic is more appropriate. Microservice architecture structures an application as a set of loosely coupled services to accelerate software development by enabling continuous delivery/deployment.

Microservice decision is based on the following:

- Microservice based on business capability
- Microservice based on subdomain
- **Object-oriented design** (**OOD**)
- **Single Responsibility Principle** (**SRP**)

- **Common Closure Principle** (CCP)





Microservice architecture--service must be adequately planned to be developed by a small team and to be easily tested. The SRP is a basis for service design to define responsibility of a class and reason to initiate the change. It creates cohesive design of services and implements a small set of strongly related functions. The **common closure principle** (**CCP**) means classes that change for the same reason should be in the same package. If the same business rule is implemented in different aspects by two classes, then for any business rule change only small modifications to be done in code to accommodate the same.

Microservices should correspond to business capabilities/business object to generate value:

- Inventory management
- Order management
- Delivery management

The corresponding microservice architecture would have services aligned to each of these capabilities. Following this pattern has the benefits such as:

- The business capabilities are relatively stable so as the architecture
- Development teams are delivering business value based on cross-functional, autonomous, and organized, rather than technical features
- Services are loosely coupled and cohesive

Identifying business capabilities, and hence services require an understanding of the business. An organization's business capabilities and services are identified by analyzing the organization's purpose, business processes, structure, and areas of expertise.

Organization structure can be based on **domain-driven design** (**DDD**) subdomains or business capability groups. DDD is related to the application's problem space as the domain criteria; for example, business groups organized on a basis of regions, domains, locations, and so on.

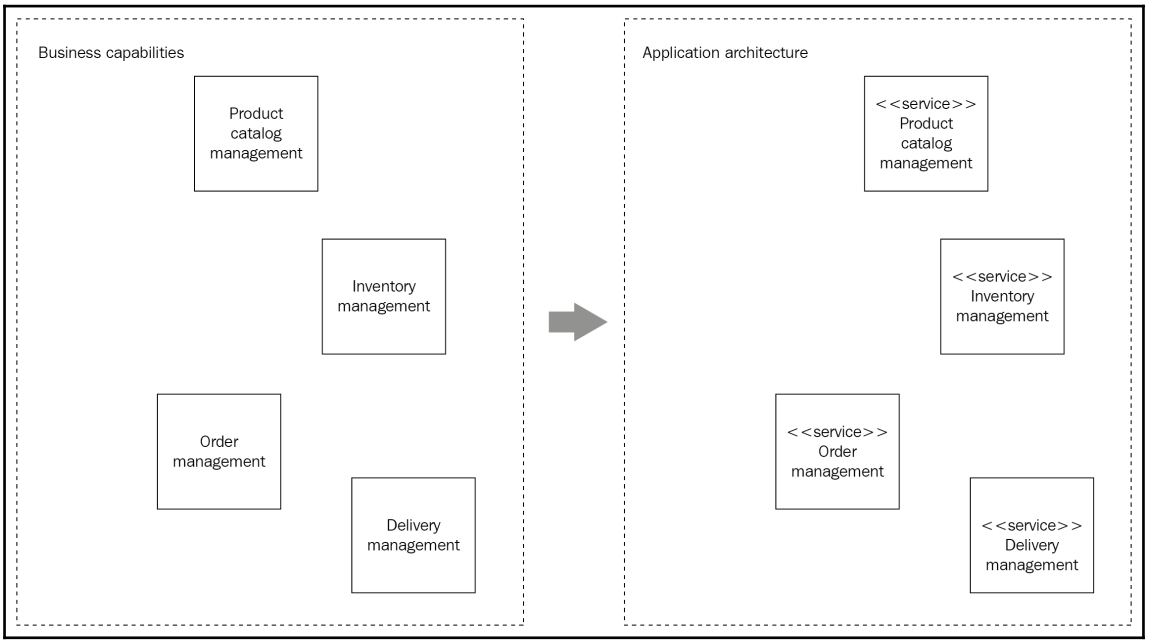
A domain is made up of multiple subdomains. Different parts of the business corresponds to subdomain as:

- **Core subdomain**: Key business differentiator and the most critical part of the application
- **Supporting**: Not key business differentiator, can be implemented in-house or outsourced
- **Generic**: Not business specific and implemented using off-the-shelf software

The subdomains of an online store application can be as follows:

- Product catalog
- Inventory management
- Order management
- Delivery management

The corresponding microservice architecture would have services corresponding to each of these subdomains:



Microservices deployment patterns

The guiding principles for services are as follows:

- A variety of languages, frameworks, and framework versions can be used for services
- Multiple service instances for each service for throughput and availability
- Independently deployable and scalable services

- Isolated service instances from one another
- Faster build and deploy ability for a service
- The resources (CPU and memory) consumed by a service should be constrained
- Each service instance should be transparent to monitor behavior
- Reliable and cost-effective deployment of service
- Application metrics and health check API's
- Audit logging and compliance
- Distributed tracing and management
- Exception tracking and management
- Log aggregation, log deployments, and changes
- Service component testing and service integration contract testing
- UI composition (server-side page fragment, client-side UI)
- Security--Access Token based on JSON Web Token identifying the requestor securely to each service

Distribution patterns

- Multiple service instances per host, like multiple instances of different services running on a physical or virtual host machine
- Deploying a service instance on a shared host
- Deploy each service instance as a JVM process, per service instance
- Deploy multiple service instances in the same JVM
- Avoiding conflicting resource requirements, or dependency versions
- Service instance per VM
- Service instance linked to each container
- Serverless deployment options are explored as follows:
    - AWS Lambda
    - Google Cloud Functions
    - Azure Functions

Microservice chassis

- Externalized configuration for credentials management, and of external services such as databases and message brokers for network locations
- **Logging**: Usage of logging framework like log4j or logback
- **Health checks**: Determine the health of the application through a URL-based monitoring service
- **Metrics**: Measurement and insight for application performance
- **Distributed tracing**: A unique identifier between services traced with instrument code-based services

For example:

- Java:
    - Spring Boot and Spring Cloud
    - Dropwizard
- Go:
    - Gizmo
    - Micro
    - Go kit

Communication mode

The various types of communication used in microservices are shared following:

- Remote Procedure Invocation for interservice communication for client requests by services as listed as follows:
    - REST
    - gRPC
    - Apache Thrift

- Messaging requests from clients through asynchronous mode by channels as follows:
    - Apache Kafka
    - RabbitMQ
- Domain-specific protocol:
    - Protocols such as SMTP and IMAP for emails
    - Protocols such as RTMP, HLS, and HDS for media streaming

Data management options

The various modes of data management process in use for microservices are listed following:

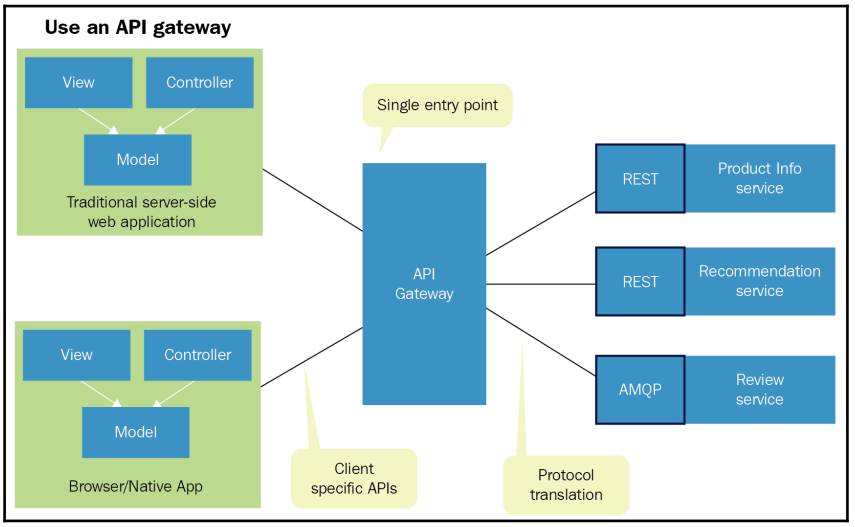
- Database per service ensures loosely coupled services, each service can choose use type of database that's best suited.
- Shared database helps, as developer uses ACID transactions to enforce data consistency that are familiar and straightforward.
- Event sourcing persists the state of a business entity as a sequence of state-changing events. A new event is appended to the list of events with the state of a business entity changes.
- Transaction log tailing.
- Database triggers insert events into an EVENTS table to be polled by a separate process that publishes the events.
- Application events.

API interface

The different APIs used in microservices are as following:

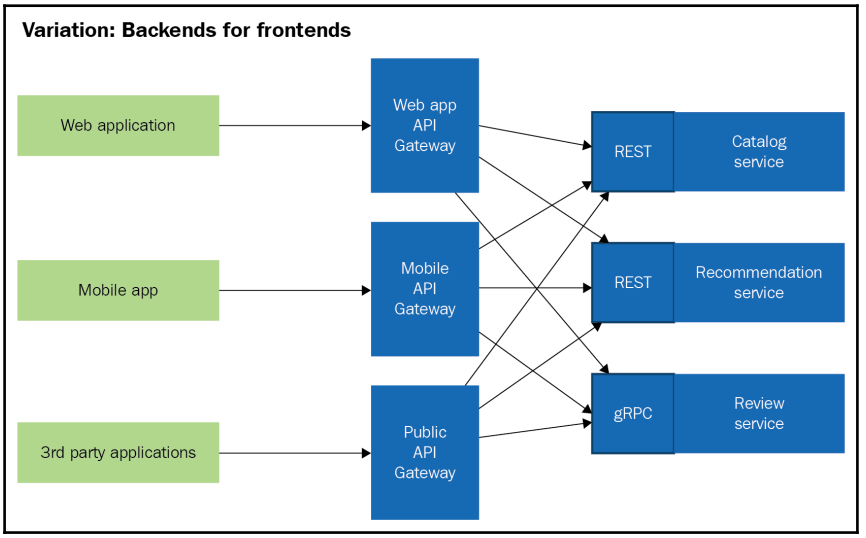
- UI for desktop and mobile browsers is HTML5/JavaScript-based
- Server-side web application generates HTML
- Native Android and iPhone clients interact through REST APIs with the server

- For third-party applications through the online exposure of details is by REST API



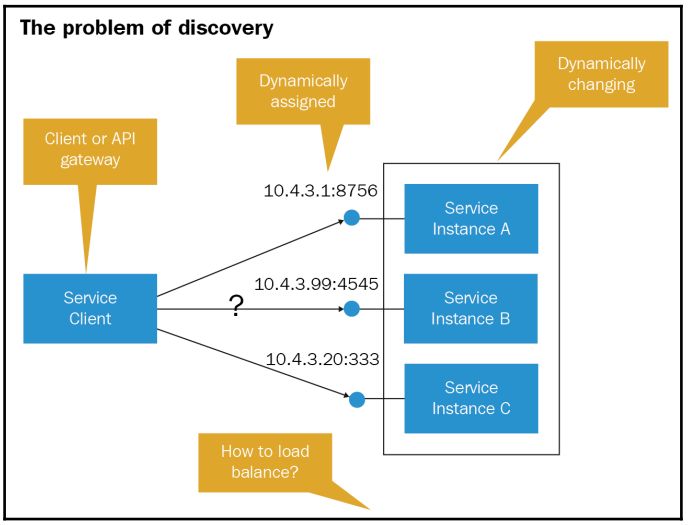
Usage of Application Program Interface (API) gateway protocols

The following figure shows usage of backend for frontend APIs:

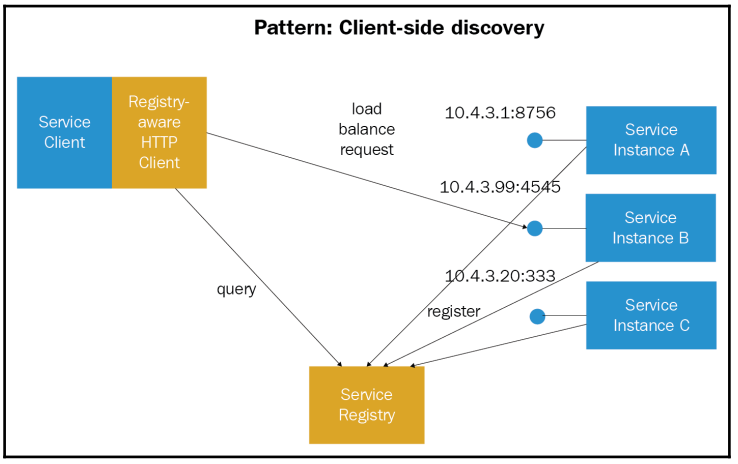


Service discovery

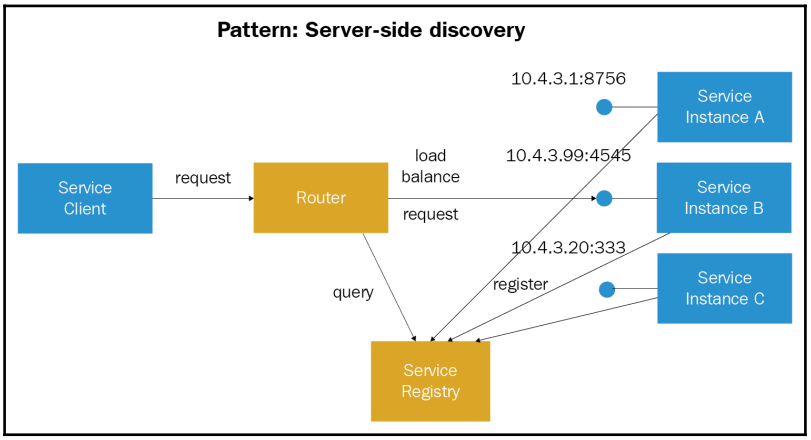
Microservice application is designed where the number of instances of a service and their locations change dynamically to run in a virtualized containerized environment, so the typical problem associated with service discovery is shown here:



Client-side discovery exposes a remote API such as HTTP/REST or Thrift at a particular location (host and port) for each instance of a service:



Server-side discovery is making a request to a service by the client through a router (that is, load balancer) that runs at a well identified location address like a service registry, which the router queries and forwards the request to an available service instance:



Summary

In this chapter, we covered the topics of containers and flavors offered by different vendors, virtualization methods, container orchestrations, Internet of Things, and microservices applications and architectures.

10
DevOps for Digital Transformation

In this chapter, we will explore the digital transformation journey in the light of DevOps adoption by systems of big data migration, cloud migration, microservices migration, data sciences, authentication security, and the **Internet of Things** (**IoT**).

The following topics will be covered:

- Digital transformation
- DevOps for big data
- DevOps for Cloud
- DevOps for microservices
- DevOps for data sciences
- DevOps for authentication systems
- DevOps for IoT

Digital transformation

Journey to digital transformation is unique for every company; hence, the very definition will vary. On a broad note, it's about the adoption and integration of digital technology making the fundamental shift of business operations to deliver value to customers.

To accomplish this, companies need to adopt a culture to challenge continuously the status quo of the long standing established processes, and experiment often in favor of relatively new practices that are evolving.

The incentives for this digital journey are as follows:

According to a Forrester Research report, executives predict that nearly half of their revenue will be driven by digital by the year 2020.

According to the MIT Center for Digital Business, *companies that have embraced digital transformation are 26 percent more profitable than their average industry competitors and enjoy a 12 percent higher market valuation*.

Research shows, *nine out of ten IT decision-makers claim legacy systems are preventing them from harnessing the digital technologies they need to grow and become more efficient.*

Their customers and internal employees will whole-heartedly support them as digital practices have made inroads into people's lives for all facets, such as shopping online via mobile devices and remotely adjusting their home heating systems.

Digital strategy adoption should be considered as a long-lasting cultural, technological change and not a tactical means. It's re-evaluation of the software platform and architecture, development methodologies, technologies, business processes, roles, and responsibilities. It's evolutionary, incremental, and iterative and not meant to be revolutionary or disruptive. It's about the alignment of organization to the change rather than a push from **Chief Information Officers** (**CIOs**) alone to match the competition.

The steps to a successful digital journey are:

- Formulating digital strategy for organization
- People involvement as important as technology
- Legacy architecture transformation
- Legacy processes and attitudes modernize
- Using mobile as a catalyst for change
- Harnessing power **application programming interfaces** (**APIs**)
- Planning to stay secure

Digital transformation framework varies widely as per organizations' specific challenges and demands. The common themes for business and technology leaders as they embark on digital transformation are often cited to aid in developing their own digital transformation strategy, as follows:

- Customer experience
- Operational agility
- Culture and leadership
- Workforce enablement
- Digital technology integration

Digital transformation cultural shift for organizations is enabled by the fundamental shift in the role of IT.

A recent report from Harvey Nash and advisory firm KPMG found that, *CEOs are focused on IT projects that can make money (63 percent), rather than those that save money (37 percent)*.

Research from Forrester suggests, on average, CIOs spend an average of 72 percent of their budgets on existing IT concerns, whereas only 28 percent goes to new projects and innovation.

In the past 4 years, long-standing CIO priorities have seen the biggest change in priorities of relative importance. For example, increasing operational efficiencies has dropped 16 percent, and delivering stable IT performance has dropped 27 percent.

Bryson Koehler, CIO of The Weather Company, says, *There is a very different mindset at work when you take IT out of an operating mode of, 'Let's run a bunch of packaged solutions that we've bought and stood up' to 'Let's build and create new capabilities that didn't exist before'*.

Marc Carrel-Billiard, global tech R&D lead on digital transformation at Accenture, is quoted as saying in an article for TechCrunch that *Finding ways to help people across this digital divide and the culture shock that rapid change brings is going to be just as important as the technology we use to get there*.

Dr. David Bray, CIO of the U.S. Federal Communication Commission, speaks on how this cultural shift sets the stage for transformation: *Throughout human history, the things that we could do with tools changed what we could do as humans, and as a result, changed what we could do as cultures*. Bray says, *Our species is 'smart' because we know how to use tools collaboratively together. At the end of the day, when we talk about technology change - whether it's the Internet of Everything, big data, or machine learning - it's really about people and organizational cultures, first and foremost. Then, it's about how those people get stuff done together - and that's really what it comes down to when you talk about transforming organizational cultures*.

Nextgov reports, *Around three-quarters of the $80 billion the federal government spends on information technology each year is used just to keep legacy systems running*.

According to a study by Tiger Text and using research conducted by HIMSS Analytics, in the healthcare industry, *Despite widespread use of smartphones and other mobile devices among healthcare providers, 90 percent of hospitals still use pagers and overpay by 45 percent to maintain legacy paging services*.

Companies are no longer building software or managing IT for cost savings and operations, but rather IT has become the primary enabler and driver for business innovation. To embrace adopt the shift company roles need realignment with the impact of IT in day-to-day experience. In driving digital transformation strategy, though IT will play an important role, the massive changes that go along with digital transformation implementation and adaption is everyone's responsibility. Digital transformation is primarily related to people, such as shedding of outdated processes and legacy technology to adopting agile principles and modernization across the business.

Big data and DevOps

Organizations that tend to consider DevOps as pure process maturity versus big data as technology stream tend to treat them in silos, leading to inefficiencies. DevOps' goal is to make software production and delivery more efficient, and including data subjects within the scope of continuous delivery processes to embrace DevOps will be a big asset for accomplishing organizations. Many IT leaders are now under increased pressure to produce results for investments in big data and data science projects. Big data projects are becoming more challenging. Applications are now showing up in big data projects forcing analytics scientists to revamp their algorithms. Major changes in analytic models cascades to revised resources and infrastructure in short duration. The entire process slows down if the operations team is kept out of the loop, negating the competitive advantage that big data analytics provide and warranting the need for DevOps collaboration.

In big data projects, the three big components to consider are:

- Making sure that things are reliable
- Making sure that things are scalable
- Making sure that they perform

The most challenging thing for big data projects is performance, dealing with hundreds and thousands of computers, dealing with huge volumes of data sets, dealing with rapid data changes, and simultaneously dealing with multiple things and people. That combination of variables makes performance critical for big data systems.

By integrating big data and DevOps, organizations can achieve the following benefits.

Planning effectively on software updates

Software invariably interacts with data, so while updating, redesigning an app includes understanding of the types of data sources. Collaborating with data experts along with programmers and writing new code can help plan updates effectively from a data perspective.

Lower error rates

Data handling issues account for a major share of errors with software development and testing, which compounds the complexity of application and the variety volume of data it handles, increasing the chance of errors. DevOps principle of *shift left testing*, which emphasizes to enable testing of code changes early in the cycle, referred as *left* part of the development cycle. This is enabled with DevOps practice to drive further automation with continuous integration processes. Strong collaboration between data teams and the DevOps team is crucial to avoid data-related errors in an application, during continuous delivery and deployment processes.

Consistency of development and production environments

Big data and DevOps teams working together and being part of the software delivery process can help understand the types of data challenges involved in building apps quickly to mimic real-world behavior in development and testing environments. Types and diversity of data in the real world can vary enormously; DevOps recommends non-data specialists be involved in the process.

Prompt feedback from production

A continuous delivery process includes gathering metrics from production environments after software release to analyze the strengths and weaknesses and plan further updates. Involving data teams to monitor and maintain software in production, to analyze production-related data such as app health statistics (CPU time, memory usage, and so on), can help the organization better understand the DevOps chain for continuous delivery.

Agility of big data projects

An agile environment facilitates adaptive streaming, and evolutionary development facilitates streaming between the software streams. Enterprises are moving their big data and data science projects to public cloud services for more agility to spin up virtual Hadoop or Spark clusters within minutes. DevOps adoption brings agility to projects and businesses, as we have seen in previous chapters.

Big Data as a service

The benefits of DevOps to big data can be extended using Docker containers to provide big data as a service. Data scientists, through self-service, can spin up instant clusters with big data tools, Hadoop, Spark, and so on.

Big data processes to be adopted for DevOps are:

- ETL
- Analytics
- Visualization
- Security/Kerbos
- Data sciences
- Monitoring

Big data involves migration of a large quantity of data from source systems to destination systems; it needs proper design, data extraction, data loading, data verification, and data cleansing to accomplish the process. The upcoming discussed methodologies, along with the DevOps process, aid the expected migration in a short time even with huge data.

Big data formats and varieties can be structured, semi-structured, and unstructured data contributed by social media, machine data, server logs, and so on, encompassing multiple domains including medicine, biology, physics, healthcare monitoring, astrometry, transportation, and manufacturing.

Big data implementations have multiple options and process methodologies, as discussed next, to evaluate and adopt DevOps for both applications and infrastructure.

The ETL datamodels

Extract, **Transform**, and **Load** (**ETL**) is the process in which the data is extracted from source systems, transformed with business logic, and loaded into the system for business usage. Data models and schemas are built in accordance with the business rules based on input and output data.

There is an other model named **Extract**, **Load**, and **Transform** (**ELT**), where the data is loaded in raw form in the staging layer, and transformation, business logic, rules are applied for further usage to business.

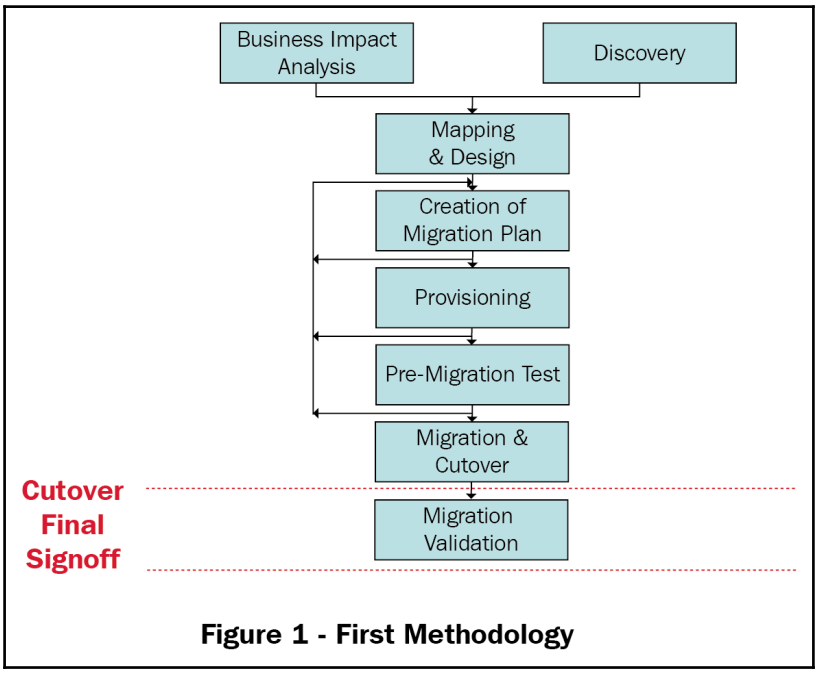


Figure 1 - First Methodology

Methodology 1

This methodology has the following eight steps: business impact analysis, discovery, mapping and design, creation of migration plan, provisioning, pre-migration test, migration and cutover, and migration validation:

- **Business Impact Analysis** is identifying the business and operational requirements for the migration process
- **Discovery** is about the details of data sources, migration hardware, and software environment collected
- **Mapping and design** identifies how and where the data is to be moved
- **Creation of Migration Plan** is the blueprint specifying customer expectations along with project deliverables
- **Provisioning** prepares the destination storage environment for the data hosting
- **Pre-Migration Test** tests and validate migration components
- **Migration & Cutover** is for data from source-to-destination migration
- **Migration Validation** confirms that all expectations are met in the post-migration environment

Methodology 2

This methodology ensures that data is properly evaluated, reviewed, and restored before it is migrated to target systems.

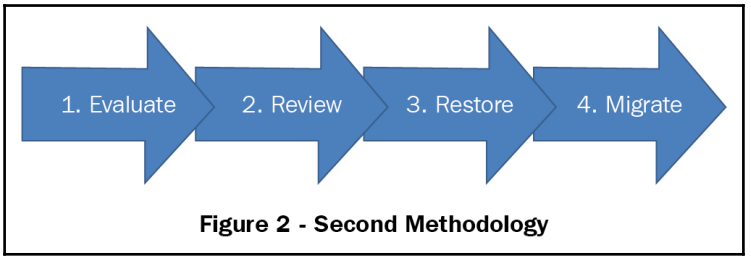


Figure 2 - Second Methodology

- **Evaluate** phase analyzes source type, state of media, and effort estimation
- **Review** phase evaluates customer requirements and criteria, and also the targeted system
- **Restore** phase identifies and restores individual files and recovers data before the data extraction process
- **Migrate** phase indexes and reduplicates the restored data and migrates data to target delivered to customer

Methodology 3

This methodology was introduced specially to handle methods, such as the database migration readiness stage, migration stage, ETL stage, and application stage.

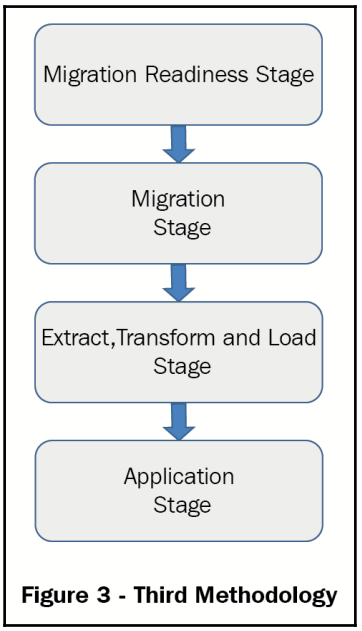


Figure 3 - Third Methodology

- **Migration Readiness Stage** creates a business case, evaluates risk, creates engagement model, assesses the database environment, infrastructure planning, and includes hardware and software.
- **Migration Stage** analyzes the present database and target database design based on business objectives. It builds database objects, such as tables, views, and triggers. It transfers data; the prioritized migration roadmap verifies database schema and validates that data is migrated aptly.
- **Extract, Transform, and Load Stage** designs and develops ETL packages to handle parallel data load, validates technical accuracy, functionality, and security and tests data load performance.
- **Application Stage** performs integration testing for the applications and user acceptance testing to implement in the production environment.

Methodology 4

This was introduced by Gershon Pick with four steps:

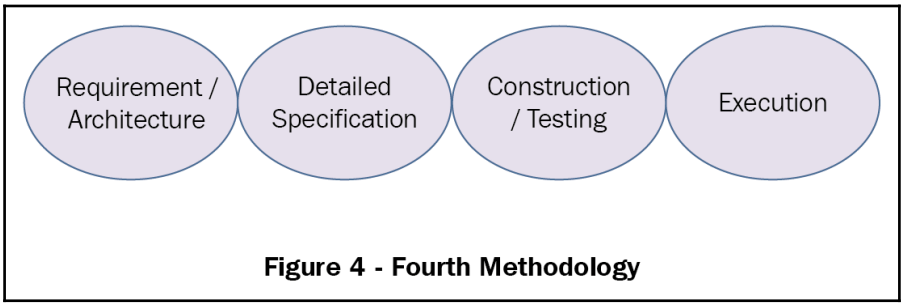


Figure 4 - Fourth Methodology

- **Requirement / architecture** phase defines high-level requirements such as data, performance requirements, and also a detailed project plan.
- **Detailed Specification** phase defines transformation, validations, and structure changes.
- **Construction / Testing** phase builds the migration solution and tests.
- **Execution** phase ensures the target system validates results to implement in production.

Methodology 5

This was introduced by Dell. It has five steps shown as planning, pre-production testing, migration, validation, and cutover.

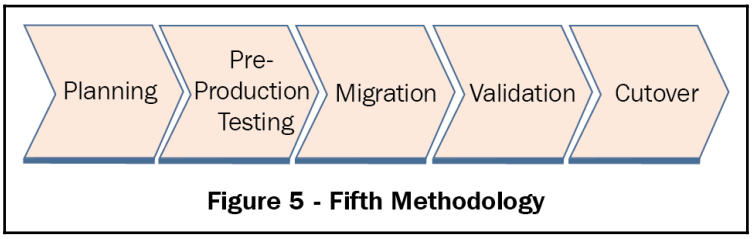


Figure 5 - Fifth Methodology

- **Planning** phase defines migration goals and requirements and creates a migration plan defines HW/SW and tools.
- **Pre-Production Testing** phase tests the migration environment and collects and validates data. It validates HW/SW and migration tools, and updates the final migration plan.

- **Migration** phase installs migration software and performs migration based on the plan.
- **Validation** phase verifies the completion of the migration, collects migration statistics, and prepares a migration report.
- **Cutover** migrates application to the target environment and creates a final report.

Methodology 6

This methodology has eight steps explaining the common and necessary steps involved in the migration process.

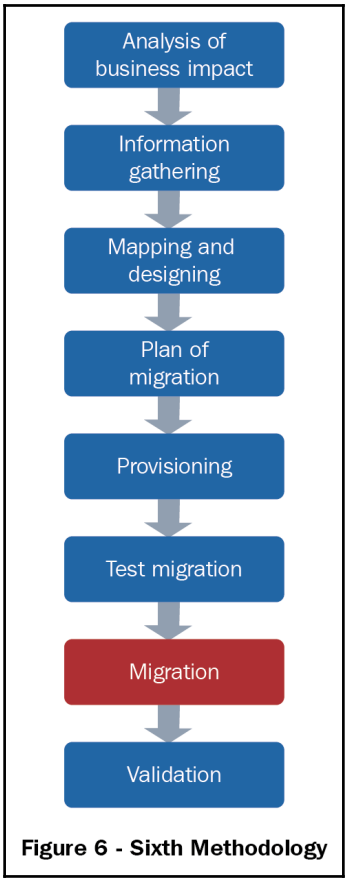


Figure 6 - Sixth Methodology

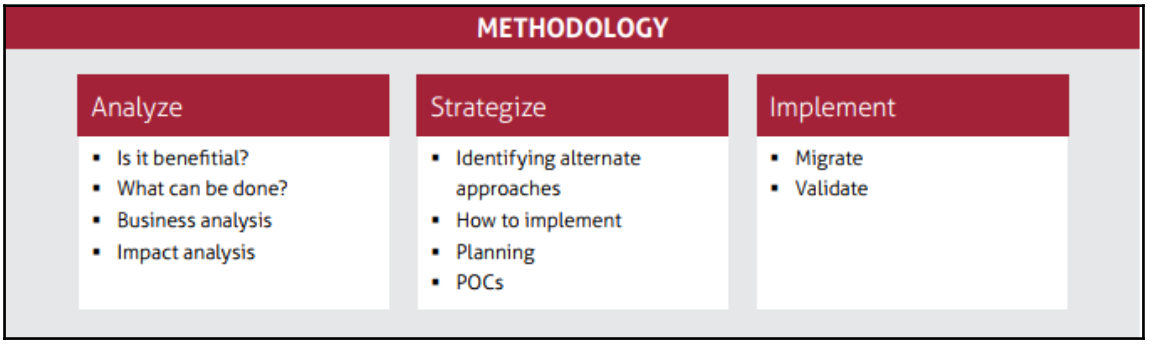
- **Analysis of business impact**: Data is the basic and important aspect that should be analyzed to enhance the business process. This step analyzes customers' requirements for business.
- **Information gathering** phase collects the details about source and destination systems. Software and hardware information collection can be manual or an automatic process.
- **Mapping and designing** phase maps source and destination systems maintain two types of design layouts. One for source and destination layouts is the same, and relay layout for source and destination schemas are different.
- **Plan of migration** phase considers business and operational constraints to migrate data, along with its attributes and tools for the process.
- **Provisioning** phase replicates the source structure of files, data volumes, and attributes for the environment to receive the streams of actual data.
- **Test migration** phase ensures that all the presumptions are valid, and tools are appropriate to minimize the risk of time and money wasted.
- **Migration** phase determines data movement from source to destination with two possibilities of moving data out of path and into path.
- **Validation** phases checks data access, file permissions, and structure of directories to validate working of applications.

As we have discussed, every phase of big data migration process from ETL, analytics, and visualization can be analyzed as a process methodology to apply DevOps maturity and tools at every stage for performance enhancement and productivity improvement.

Cloud migration - DevOps

Cloud migrations can be very expensive if they are not done correctly. Applications perform differently in the cloud versus on-premise, especially if it's a complex application where matured testing strategy makes the difference. DevOps methodology incorporation at each stage of the migration process will surely add multi-facet value. Each stage of cloud migration is detailed here for adopting DevOps strategy for applications, infrastructure, and tools as per the organization's needs.

Several aspects are to be considered for an application migration to the cloud, as follows:



- **Application feasibility**: This is architectural compatibility of the application for cloud hosting.
- **External/internal dependencies**: Applications accessibility of internal and external dependencies from cloud to be understood.
- **Application class**: High-demand applications from a business perspective classified as business-critical and LOB applications require high availability.
- **Application integration**: This validates application performance with other on-premise applications and shared services.
- **Scalability/elasticity**: Application design supports scalability on cloud.
- **Compliance adherence**: This safeguards enterprise-level compliance, regulations, and governance for data moved/stored outside the enterprise's premises.
- **Return on investments**: This helps hosting applications on cloud to be more cost-effective for the enterprise.
- **Security**: The same level of security can be provided after migrating to cloud as:
    - Data security
    - Authentication
    - Authorization
- **Database compatibility**: The existing database is supported, and it is compatible with cloud. Here are ways to maintain application data while migrating an application to cloud:
    - Residing database on-premise
    - Creating database on VM
    - Windows database (PaaS)

Migration strategy/approach

Multiple factors are considered for the migration decision:

- **UI analysis**: UI interface to be analyzed for migration to cloud in the PaaS model. On-premise web-based applications and services can be mapped to cloud with re-engineering to be compatible with cloud services, the non-web applications at on-premise are to be rebuilt to move to cloud on as-per-need basis. In addition, third-party framework / class library to be compatible with cloud might require some modifications or may have to be re-written. With the IaaS model, the entire server image will be migrated with minimal code changes between the clouds.
- **Authentication and authorization**: The authentication mechanism in the application to be analyzed for compatibility such as using forms-based authentication and an access control service, or ACS with integration with enterprise on-premise active directory.
- **Interaction with other modules/applications**:
    - **Web services**: These can be hosted either as a web role or worker role, left as on-premise services, and can be exposed through APIs
    - **Native code**: A managed wrapper package can be created and deployed
    - **Third-party dependency**: These dependencies are validated to be consumed directly from web services
- **Diagnostics support**: This implements custom logging and saves the log information to storage tables:
    - Push event logs to diagnostics store
    - Push failed request logs to diagnostic store
    - Push performance counter data to diagnostics store
- **Miscellaneous**
    - **Message queues**: Subscriptions used for message publish and subscribe model
    - **Configuration changes**: Applications should not have any hardcoded physical disk or network access values
    - Check for any third-party library or content references
    - Replace static values and application states to handle scalability applications
    - **Application log**: This is the management of custom logs' capture and storage

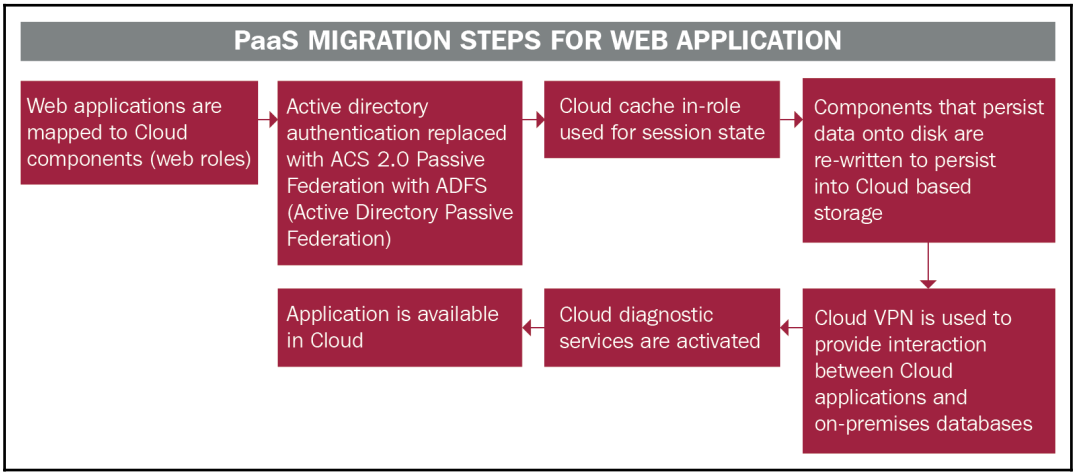
- **Data migration strategy**: Application migration strategy should go hand-in-hand with data migration strategy, as most of the applications are data-centric. Applications can store data onto disk, or into a database or network storage; however migrating applications from on-premise to Cloud demands that users do not see any discrepancy in their data.

  Cloud provides the flexibility to persist data in the same way they are stored; in the on-premise application. Cloud-hosted applications data can be saved in multiple ways:

  - Data from on-premise database to cloud-based database
  - Static content to cloud storage
  - Message queues to cloud queue storage / service bus queue

- **Migration execution**:
  - The web application migration from on-premise to cloud is planned component-wise in an incremental, independent fashion
  - The section that follows explains the migration process for the PaaS and IaaS options

- **Acronym Description**
  - **Access Control Service** (**ACS**)
  - **Active Directory Federation Service** (**ADFS**)
  - **Infrastructure as a Service** (**IaaS**)
  - **Platform as a Service** (**PaaS**)



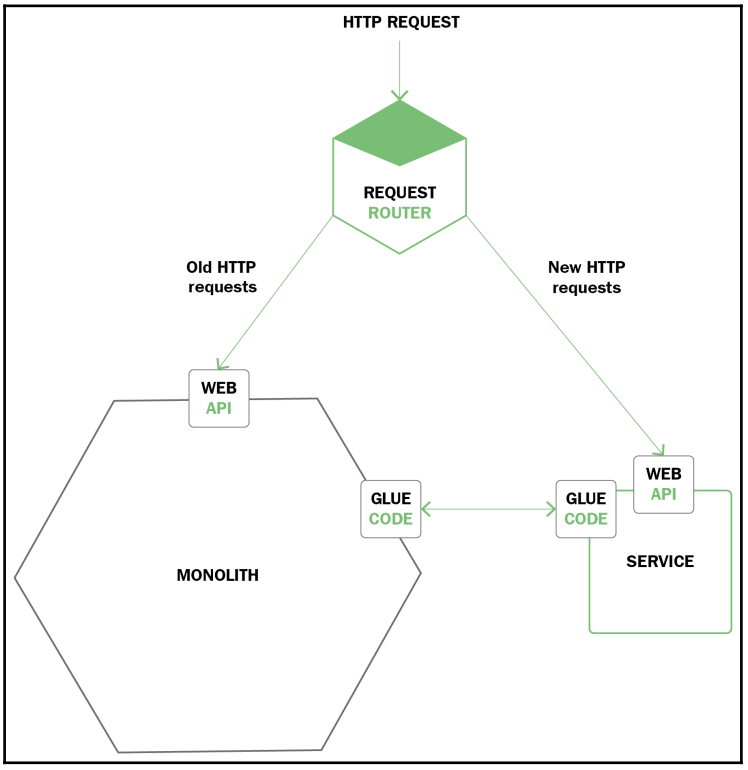
Migration to microservices - DevOps

DevOps principles and methodology are very appropriate for microservices while the latter migration involves architecture, API's, and code development. All of them would be under code version system, continuous integration, build, test systems till continuous deployment.

Refactoring to microservices from monolithic architecture can be pursued in multiple ways. Here are three prime strategies listed.

Strategy 1 - standalone microservice

Implementing new functionality to the monolithic application should be put up as new code in addition to a standalone microservice, instead of adding to monolith application making it more bulky. The following diagram shows the system architecture after applying this approach:



In the new architecture, both the new service and the legacy monolith co-exist. There are two communication components. A request router handles incoming (HTTP) requests similar to the API gateway; the router sends requests corresponding to new functionality to the new service routing the legacy requests to the monolith.

The glue code is the second component to integrate the service with the monolith. The service needs to access data to read and write to the data owned and processed by the monolith application, and the glue code could reside in the monolith application, service, or both.

There are three strategies to access the monolith's data by the service:

- Monolith applications remote API invoked
- Directly access the monolith's database
- The service maintains its own data copy to synchronize regularly with the monolith's database

The glue code is an important function to translate between the two different models. The glue code maintains its own pristine domain model. The glue code has to prevent its own model from getting polluted by the concepts of the legacy monolith's domain model; hence, it's also referred to as an anti-corruption layer.

This method of implementing new functionality as a lightweight service offers the following benefits:

- The new functionality/service can be independently developed, deployed, and scaled, and not linked with the monolith
- The new services created extend the benefits of the microservices architecture
- It prevents the monolith from becoming bulkier and unmanageable

Strategy 2 - separate frontend and backend

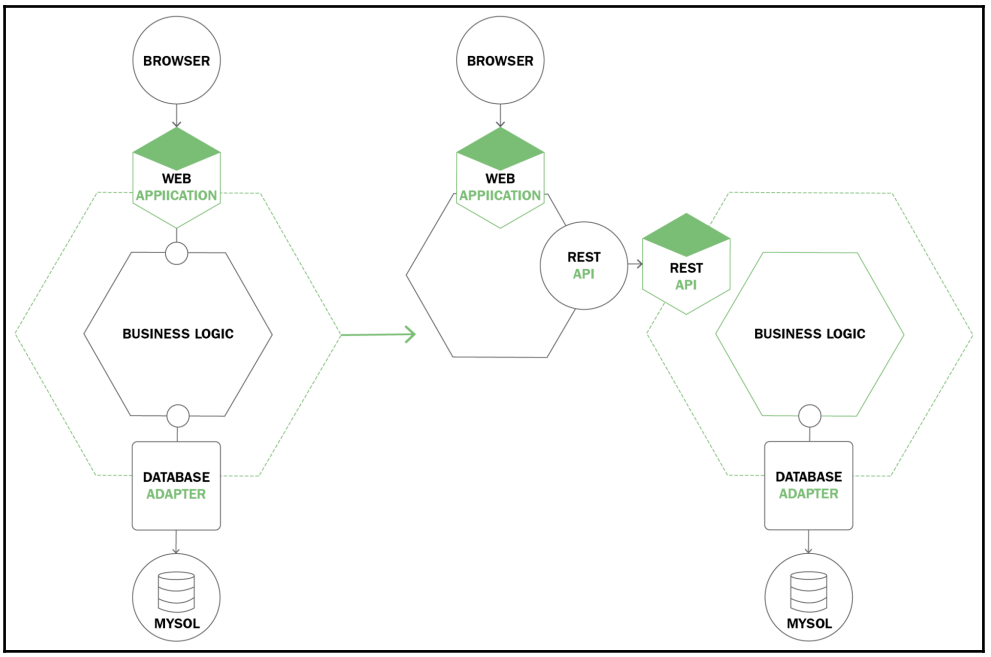
This strategy is to separate the presentation layer from the business logic and data access layers in the monolithic application.

An enterprise application comprises at least three different layers, as follows:

- **Presentation layer**: This is a sophisticated user interface with a substantial body of code; it handles HTTP requests with HTML-based web UI or a (REST) API, and so on.

- **Business logic layer**: This contains the core business rules components for the application.
- **Data-access layer**: This accesses infrastructure components of databases and message brokers

In the monolithic model, the division of roles and responsibilities between the layers is presentation logic and the business and data-access logic. The business tier encapsulates business-logic components with coarse-grained API's. This is a natural process to extend for microservices architecture by splitting the monolith into two segments. One segment contains the presentation layer and the other contains the business and data-access logic layer. Then, the presentation logic segment or application makes remote calls to the business logic application or segment, as shown in the following diagram for the architecture before and after the change.



The main benefits accrued are to enable, develop, deploy, and scale the two application segments independent of one another, such as the user interface from the presentation layer, so they can be iteratively and rapidly developed and tested; an other benefit of this approach is to expose a remote API for consumption by microservices.

In this case, all the three layers are bulky monolith components. This strategy is only a partial solution, so the next strategy will be explored.

Strategy 3 - extraction of services

Turning existing modules within the monolith into standalone microservices is the third refactoring strategy. In this process, every time a module is extracted and turned into a service, the monolith gradually shrinks. With adequate conversion of modules, the monolith ceases to be an issue, disappears entirely, or becomes small enough to be another service.

Prioritizing the modules for conversion to services

A few factors that will facilitate this are as follows:

- Initiating with a few modules that are easy to extract
- Identifying the modules that will provide maximum benefit
- Identifying modules that change frequently
- Ranking the modules by benefit or frequency of change
- Modules with unique resource requirements compared with other modules in a monolithic application, for example, those needing in-memory speed or computationally expensive algorithms, to be run on dedicated machines to scale the application quickly and easily
- Modules with existing coarse-grained boundaries are easier and cheaper to turn into services; for example, as a module that communicates with the rest of the application only via asynchronous messages

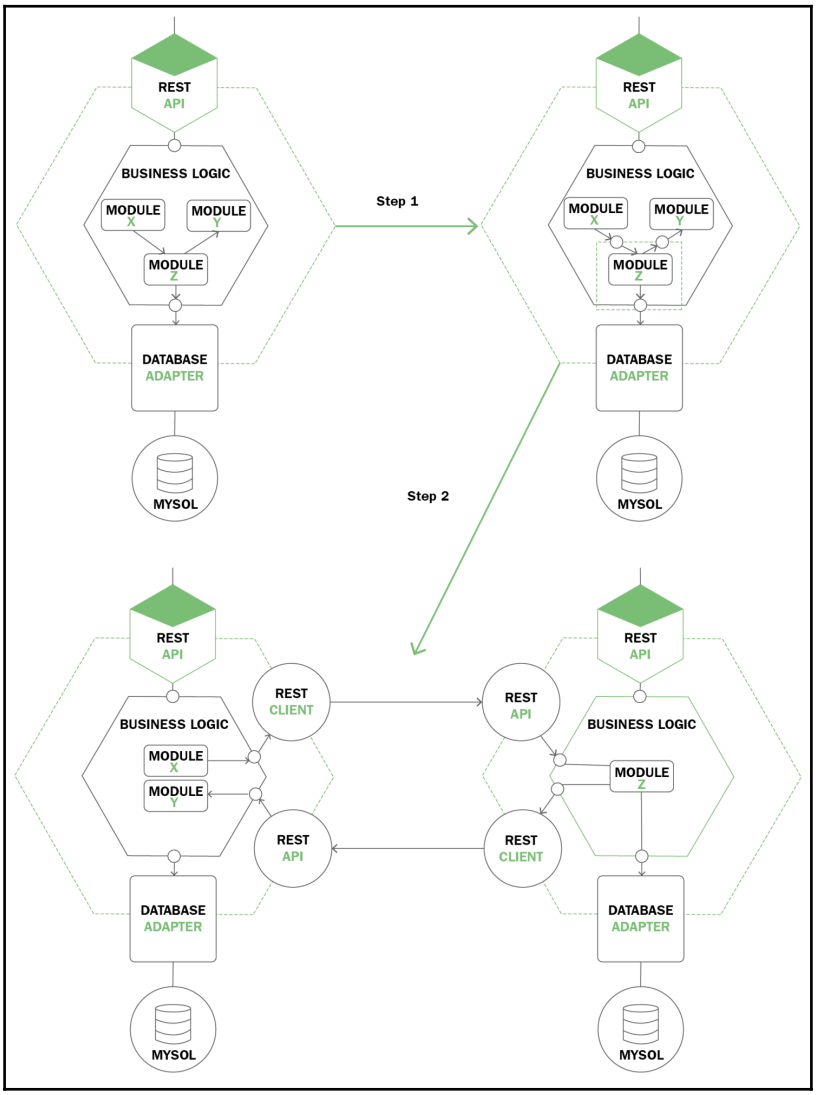
Adopting this approach, a complex monolithic application with tens or hundreds of modules can be conveniently extracted to microservices module after module, and thereafter the service can be developed and deployed independent of the monolith to accelerate development.

The process to extract a module

The initial phase to extract a module involves creating a coarse-grained interaction between the module and the monolith application, to be a bidirectional API for the data exchange between the monolith and the service. As per the tangled dependencies and fine-grained interaction patterns, it will be challenging to implement between the module and the rest of the application.

Domain model pattern-based business logic with numerous associations of domain model class dependencies can be altered with code changes. A coarse-grained module conversion into a free-standing service to communicate through **inter-process communication** (**IPC**) API enables the monolith and the service.

The architecture before, during, and after the refactoring is listed here:



In the initial architecture, as seen in the top left of the preceding figure, the input process flow is from **Module X** to **Module Z**. Then, **Module Y** uses inputs from **Module Z**.

Stage 1

- The refactoring stage defines a pair of coarse-grained APIs
- The initial interface is an inbound input from **Module X** to invoke **Module Z**
- The outbound interface from **Module Z** is consumed to invoke **Module Y**

Stage 2

- In this refactoring stage, the module is rebased as a standalone service.
- The inbound and outbound interactions are implemented using an IPC mechanism code-based service by combining **Module Z** with a microservice chassis framework that handles service discovery
- After extracting the module, the service can be developed, deployed, and scaled independent of the monolith and other services
- The service can be rewritten from scratch with API code as an anticorruption layer that translates between the two domain models to integrate the service with the monolith
- Every extracted service is an advance in direction of microservices; eventually the monolith will shrink over time with the increasing number of microservices

Apps modernization

Adoption of microservices is a form of application modernization for migrating an existing application into modern platforms. This is incrementally planned, and not attempted from scratch, by rewriting the code. As we have seen, the strategies involves separating the presentation layer components from the business and data access components converting existing modules into services, and implementing new features and functionality as microservices to improve the application agility and performance.

Architecture migration approach

Forrester Research and InfoWorld propose a four-tier engagement/architecture platform toward microservices. This architecture model adopts the changes in computing and penetration of mobile devices for application development.

The foremost consideration is to decide on microservices architecture and design the services interaction before optimizing.

The four-tier approach to microservices is broken down into the following different layers:

- **Client tier**: This is customer experience based on mobile clients and IoT
- **Delivery tier**: This optimizes user experience based on the device personalizing content by monitoring user choices
- **Aggregation tier**: This aggregates data from the services tier a long with data protocol translation
- **Services tier**: This is the usage of existing data services in-house, or external services, such as Twilio and Box

The biggest difference is the separation of the client tier; based on real-time interaction with users the layers underneath can be constantly changing.

The strategy to adopt microservices can be summarized as the following three steps:

- **Componentized**: Identifying components from existing applications and creating a microservices implementation on a pilot basis
- **Collaborate**: New processes and initiatives based on the lessons learned from the pilot stage to evaluate with stakeholders, programmers, and developers on the team
- **Connect**: Adopting microservices application to real-world scenarios

Data coupling

Microservices architecture is loosely coupled where the data is often communicated through APIs. A microservice could even run with small code, but be focused to manage a single task.

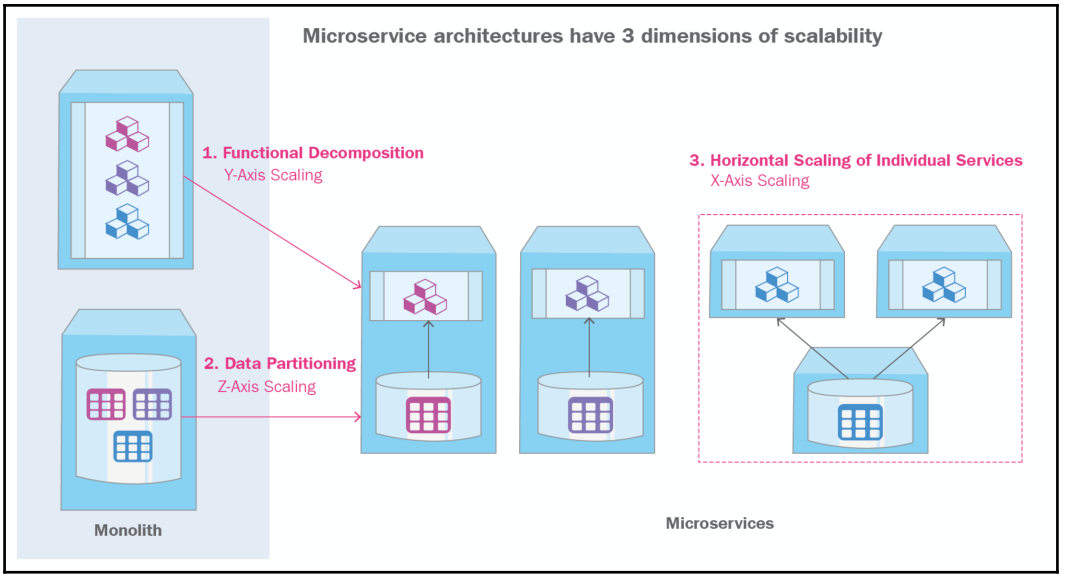
Loose coupling is based on following aspects:

- Scope boundary is defined and intelligence built-in.
- Intelligence is separated from the messaging function.
- Compatible with similar functions' microservices, and tolerance for a wide variety of modifications; changes are not forced or coordinated.
- Services are decoupled with APIs giving them freedom and flexibility. An API is a contract for services with specifications on what the services provide and also the way programs rely on them.

Microservices scalability

As per this listed architecture scalability model, these can include:

- Functional decomposition (Y-axis scaling)
- Data partitioning (X-axis scaling)
- Horizontal scaling of individual services (X-axis scaling)



Best practices for architectural and implementation considerations

Migrating to microservices is a strategy, and it needs step-by-step planning, as follows:

- Separating classes in a monolithic app
- Identifying classes that have CRUD-style interfaces with other business methods
- Identifying isolated classes with no dependencies on other classes, apart from the code needed to interact with external services, such as Memcache, Cloud Datastore, or Task Queue
- Identifying sections of code isolated from others with static code analysis tools
- Refactor code to remove unwanted dependencies, as circular dependencies are the most difficult to address
- As prerequisite to the move to microservices, refactor legacy codebase in production:
- Identify common areas for microservices as follows:
    - Account and user information
    - Authorization and session management
    - Configuration or preferences settings
    - Notifications and communications services
    - Photos and media, especially metadata
- The next steps for porting to microservices, post identification of classes set
    - Rollback option by retaining the existing code operational in the legacy application
    - New code repository creation, and copying the classes into it
    - Create HTTP API through view layer to provide the hook to format appropriately the response documents
    - New code creation as a separate application (`exampleapp.yaml`)
    - New microservice deployed as a service or separate project
    - Functional testing of the code
    - Data migration from the legacy app to the new microservice

- Legacy application modified to make use of the new microservice application
- Altered legacy application to be deployed
- Ensure adequate verifications in functionality and performance
- Dead code if any from the legacy application is removed

Domain modeling

At the heart of designing coherent and loosely coupled microservices is domain modeling, to ensure isolation and insulation of microservices and their reusability. Each application's microservices should be adequately isolated from runtime side effects, and also insulated from changes during the implementation of other microservices in the system. Proper domain modeling helps avoid the modeling shortcomings in the system based on technological or organizational boundaries, resulting in separation of services for data, business logic, presentation logic, and so on.

An example could be extraction of promotions services from a monolithic e-commerce system to be consumed by various clients using mobile Web, iOS, or Android apps. So, the domain model of *promotions,* along with its state entities and logic, are to be insulated, and not polluted with cross-domain logic or entities from other domains of the system, such as products, customers, or orders.

Service size

The right service size for a microservice based on the *Single Responsibility Principle* is a driving force for independent operation, and testing advocates as small a service size as appropriate, possibly similar to Unix-based utilities for small code bases that are easy to maintain and upgrade.

Depending on product type, and based on different business logic, encapsulating can become overwhelming, so the better approach is to consider adding more boundaries inside the product domains and create further services. Another option is to consider the time to replace the microservice with a new implementation or technology and accordingly plan for size reworking.

Testing

Testing plays an important role while a monolithic system is progressively transformed into a microservice-enabled system; integration testing of the services with the monolithic system to be executed and also business operations spanning the pre-existing monolithic system continue to perform with new microservices systems. The system-provided consumer-driven contracts can be translated into test cases for testing the new microservices, in which case automated tests ensure expectations from the system are met.

A few automated test suites are:

- `Pact`, a consumer-driven contract testing library, which creates a reusable test environment to deploy a test copy of the entire system for testing the microservices
- Using an automation tool such as Docker compose, which can containerize the entire system in the form of Docker containers orchestrated to quickly deploy the test infrastructure of the system to perform integration tests locally

Service discovery

A service discovery system enables services to know of each other while accomplishing a business function. Each service refers to an external registry with details of the other services. This can be accomplished with environment variables for a small number of services; service discovery for more sophisticated systems rely commonly on Consul and Apache Zookeeper.

Deployment

The deployment of each microservice should be self-enabled through a runtime container or by embedding a container in itself similar to how a JVM-based microservice Tomcat can container be embedded eliminating the need for a standalone web application server. At any instance, a number of microservices of the similar type (reference scale cube X-axis scaling) exist to allow more reliable handling of requests. A software load balancer to act as a service registry is included in implementations for failover and transparent balancing of requests, such as Netflix Eureka.

Build and release pipeline

Continuous integration and deployment pipelines are highly valuable for implementing microservice-based systems with on-demand, build and release, exclusive pipeline for each microservice reducing the cost of building and releasing the whole application.

Rolling upgrades (or blue-green deployment) should be part of release practices, which means if its green the upgrade is successful, otherwise a rollback strategy should be implemented. This can also be achieved by maintaining concurrent versions of the same microservice running in the production environment at any point in time (existing versions along with new builds) to quickly swap as needed. In this scenario, an active percentage of the user load can be routed to the new microservice version to test its operation, while gradually phasing out the older versions. This builds redundancy in the system against failed changes in a microservice crippling the system.

Feature flags

Microservice patterns include feature flags. It's a configuration parameter added to the system to allow toggling on or off to a feature. This pattern in the system facilitates to trigger the use of the requisite microservice for the feature associated with the flag option (say, turned on). For example, the same feature can coexist both in a new microservice and production, the traffic can be routed with the feature flag; this enables delivery teams to expedite the build cycle time.

Developer productivity with microservices adoption

Monolithic architectures allow quick turnaround of new business features on a tight schedule, whereas the overall system is small. However, as the system becomes bulkier, both development and operations are cumbersome.

Building new features or systems with the microservices' first approach is complicated, with many moving parts. Although it demands strong disciplines around architecture and automation, investing is rewarding in terms of creating an environment for teams to build microservices quickly and cleanly. One method is to create a standard boilerplate project that encapsulates key microservice design principles with project structure, test automation, integration with monitoring and instrumentation infrastructures, patterns of circuit breakers and timeouts, documentation hooks, API frameworks, and so on. These kinds of project templates help focus on building business functionality in a microservices-based distributed environment rather than scaffolding and glue code. A few interesting approaches are projects such as Dropwizard, Spring Boot, and Netflix Karyon, with choices based on architecture and developer skill level.

Monitoring and operations

Monitoring of coexisting features in both monolithic systems and microservices enables better visibility of the performance gains of implementing the new microservice implementation. This requires collecting statistics for comprehensive monitoring of performance of systems and resources to improve confidence in pursuing further migration.

Organizational considerations

Organizational changes are the most challenging components of migrating from monolithic systems to microservices, such as building service teams to own all aspects of their microservices. To embrace more collective code, ownership requires creating multidisciplinary units with developers, testers, and operations staff who care about software craftsmanship.

DevOps for data science

Data science projects involve majorly the multiple streams of roles performing different functions of big data engineers, data scientists, and operations teams. For data engineers, primary activities include ETL, preparing data sets for analysis, and coding for the models developed by data scientists into scripts. Data scientists are involved in developing the models, evaluating different algorithms and models based on sample test data and validating them with real data.

In this silo-working scenario, the team output may be confined to Poc's and not extend to big projects. However, even the minor tasks require overlap of skills as well as multiple interactions and design sessions with big data engineers, data scientists, and operations teams.

DevOps can bridge the gap between the streams for collaborative working by adopting best practices:

- **Tools and platforms evolution**: Engineers and data scientists should continuously evaluate and contribute to the creation of new tools and open source project, and base lining Apache Spark/Hadoop ecosystem to be stable and user-friendly for day-to-day operations.
- **Cross-skills education**: Data scientists should collaboratively consider the realistic, rational, and practical possibility rather than be confined to writing code with abstractions, such as the time duration for query and extracted data suffices into the storage system.
- **Process improvement**: DevOps is the way forward not just limited tools implementation such as writing Ansible scripts or installing Jenkins. DevOps should aid in invent of new tools of self-service for enhanced productivity and reduce hand-off between teams.

The DevOps continuous analytics environment

With maturity of continuous analytics and self-service, data scientist ownership extends from the original idea all the way to production for the data project. Being autonomous helps us to dedicate more time to producing actual insights.

Data scientists operate through the following multiple phases:

- Start with the original business idea and work on data exploration and preparation
- Invest in model development
- Deployment and validation of the environment
- Deployment to production: with proper tools, they are equipped to run complete iteration multiple times a day themselves, with quicker turnaround

Big data engineers focus on developing and contributing to tools, such as Spark, scalability, and storage optimization, enabling streaming architectures, and providing APIs and DSLs needed for the data scientist.

Product engineers can build smart applications for business users based on bundled services of analytical models developed by data scientists.

Each person is working with their own abstractions and contributing to the overall success in this collaborative mode.

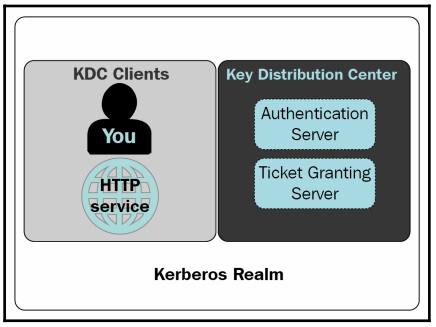
DevOps for authentication and security

Data security with Kerberos is listed as the the following process

- An authentication protocol
- Tickets to authenticate
- Avoid locally storing passwords or sending them over the internet
- A trusted third-party validation
- Symmetric-key cryptography

Kerberos realm

Kerberos realm is based on policy management definitions; it encompasses all that is available to access such as clients, services, hosts, and a **Key Distribution Center** (**KDC**) (Authentication Server and the **Ticket Granting Server** (**TGS**)). Proof of identity is user/password credentials encrypted with a secret key for the particular service requested, and **single sign-on** (**SSO**) authenticates ticket created with a new login or with a cache on the system.



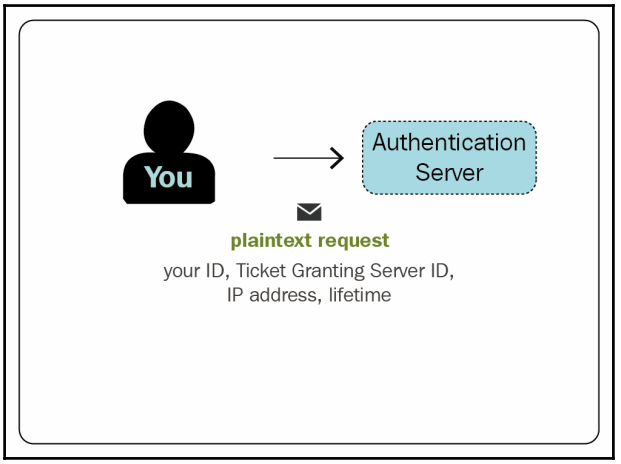
Accessing request to a service or host happens through the following interactions:

- The authentication server
- The ticket granting server
- The service or host machine needed to access
- KDC stores all of the secret keys for user machines and services in its database
- The secret keys are passwords along with the hash algorithms
- A key is generated during initial setup and memorized on the service/host machine
- Secret keys are all stored in the KDC database, based on symmetric-key cryptography
- Kerberos can also use public-key cryptography instead of symmetrical key encryption

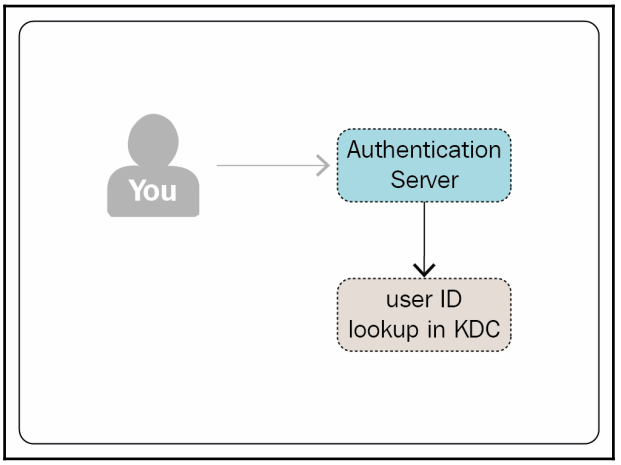
The user and the authentication server

The service request authentication between user and server is as follows:

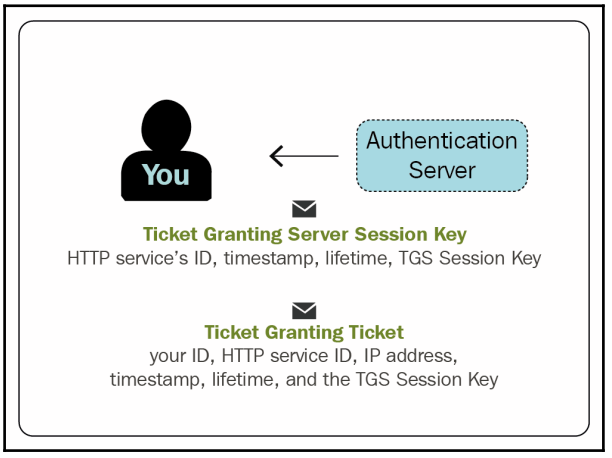
1. Access to an HTTP service needs authentication with the server through TGT from the host terminal:



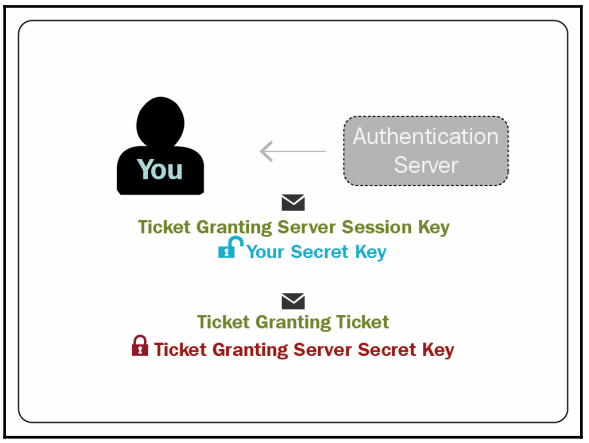
2. The authentication server will check validity in the KDC database, but not for the credentials:



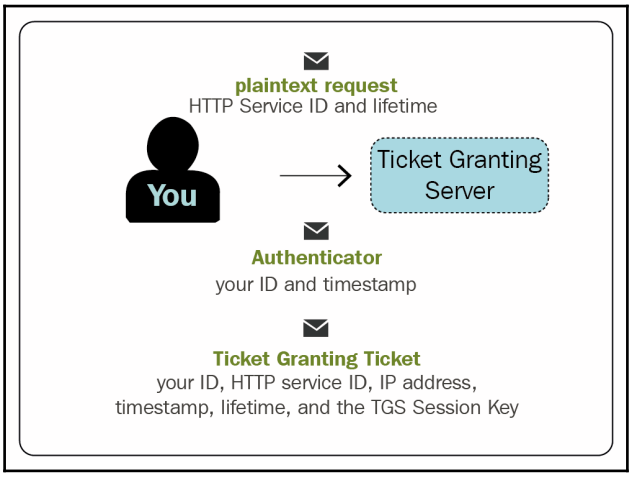
3. Once the validity is established, a session key is created between the user and the **Ticket Granting Server** (**TGS**):



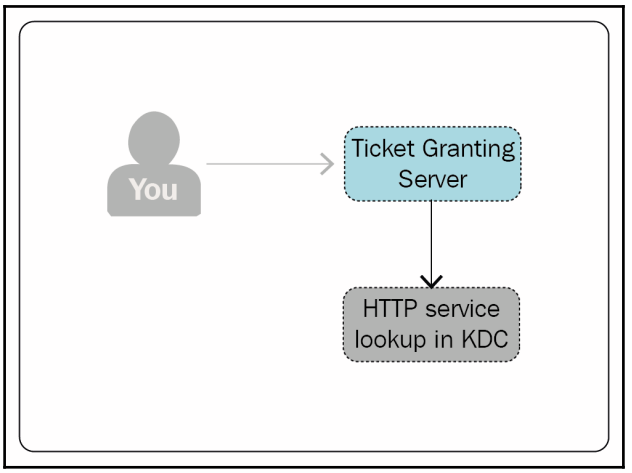
4. A client secret key (password) validates and authorizes:



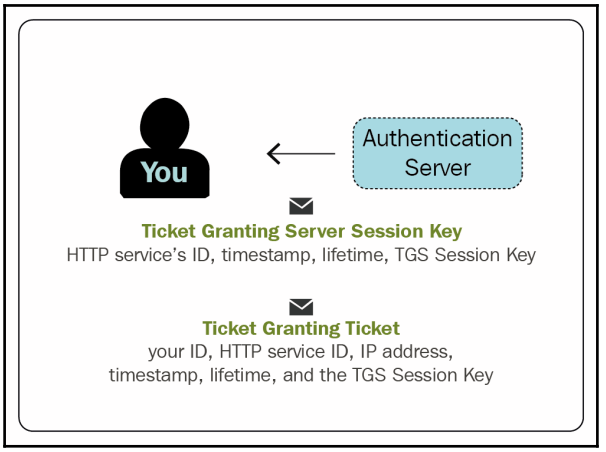
5. HTTP service requests also follow the similar process of encrypted authentication:



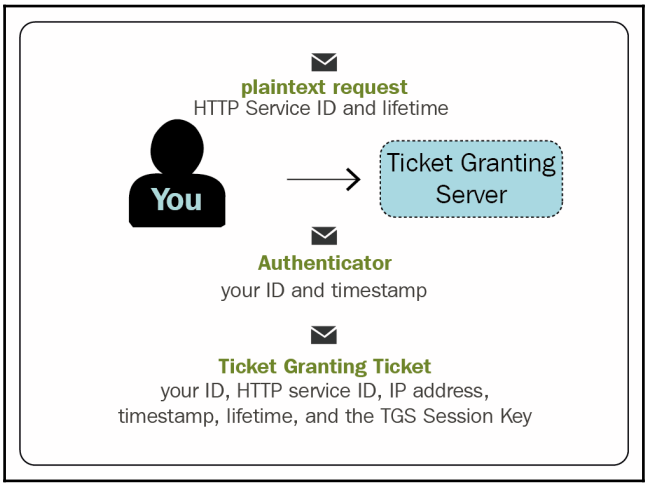
6. The TGS will check the KDC database for HTTP service availability:



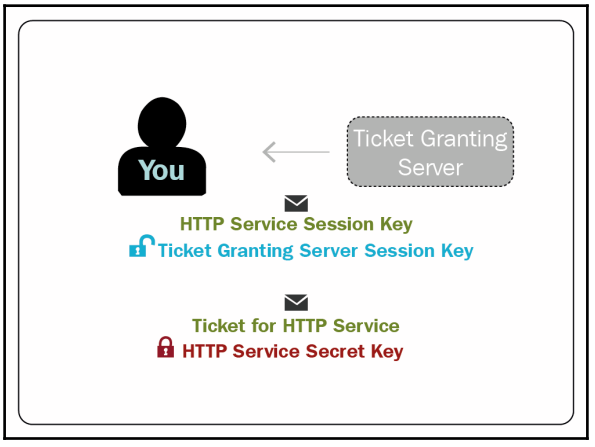
7. To render the web service page, TGS decrypts the TGT with its secret key:



8. The TGS randomly generates the HTTP service session key, prepares the HTTP service ticket, and encrypts it with the HTTP service secret key:



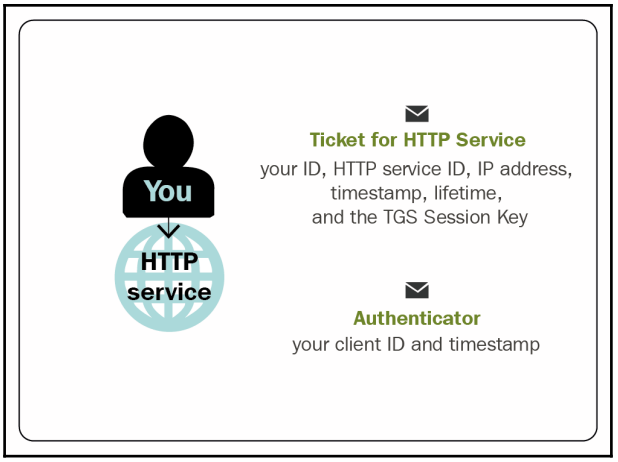
9. TGS sends the encrypted HTTP service ticket; however, the machine waits for the HTTP service secret key:



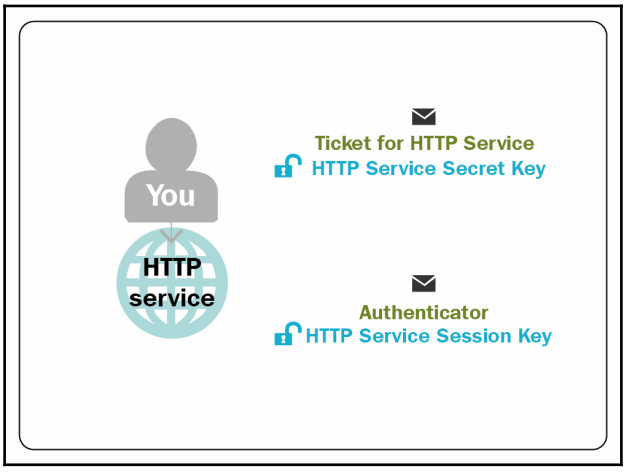
Client and the HTTP service

The service request between HTTP service and client node is as following:

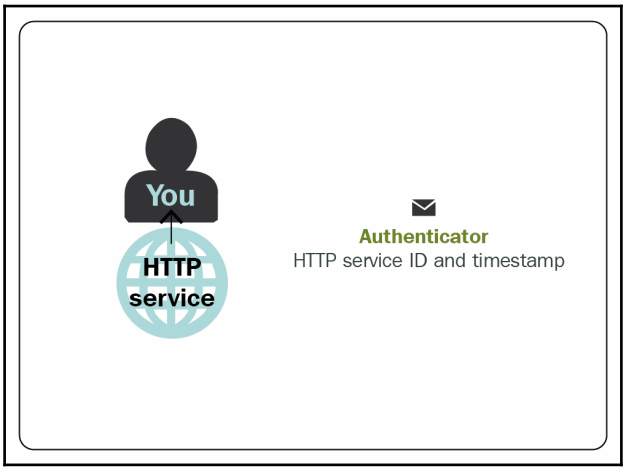
1. To access the HTTP service, the client machine prepares another authenticator message and is encrypted with the HTTP service session key:



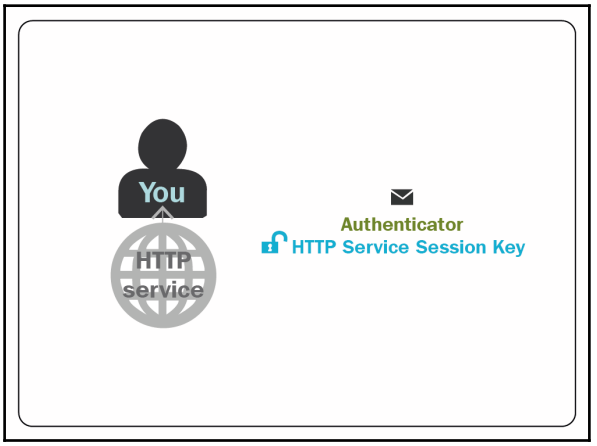
2. The HTTP service then decrypts the ticket with its secret key to obtain the HTTP service session key and decrypt the authenticator message is shared:



3. Similar to the TGS and the HTTP server sends an authenticator message encrypted with the HTTP service session key:



4. The client machine reads the authenticator message by decrypting with the cached HTTP service session key:



5. Future requests use the cached HTTP service ticket, so long as it has not expired as defined within the lifetime attribute:

6. The security authentication process involves multiple servers, policy, and credentials management, which when enabled with DevOps are highly effective for automated upgrades, deployments, and so on.

DevOps for IoT systems

Feedback drives the value. The fundamental aspect of DevOps is to gain insights into performance and usage quickly, which translates to feedback for further refinement to design and automate delivery pipelines to improve the quality of our software delivery, and to drive the most value for our customers and business. Closer customer contacts to solicit feedback and usage metrics analysis, followed by continuous improvement, helps agile organizations respond promptly to emerging requirements.

In IoT, we gather feedback about device performance and usage that can help us with:

- Preventive maintenance by predicting the need for repair based on actual usage characteristics, which reduces the duration of periodic maintenance and improves overall maintenance.
- Automatically adjusting settings to improve energy consumption and to improve the quality of the service
- Insights gained from devices that can improve the behaviors of all the connected devices.
- Improvement to products and services over time, based on customers' unique needs and actions.
- Consumer IoT; constant feedback on devices wearable devices can help improve their performance such as battery life span, for example, smartphones, watches, fridges, and cars.
- Industrial IoT; consumer, and industrial IoT drive value through the feedback loop. Businesses can thus ensure a reliable, compelling, and profitable offering that requires the operator and manufacturer have intimate knowledge of device performance based on metrics; for example, embedded microprocessors in your thermostat, sprinkler systems, and so on.
- Vital to the business value chain is listening to the feedback from customers, systems, and devices. DevOps agility helps smart organizations address IoT software challenges.

IoT is more than hardware and connectivity. It needs software to be able to develop and update it quickly while ensuring its performance securely and efficiently. Continuous delivery and continuous deployment are only feasible through automation on a wider scale. Mixing agile practices into DevOps accelerates development cycles with incremental builds, and frequent releases of small batches of code makes for faster and safer releases.

Security by design

The new feedback mechanism comes at a cost in terms of security; connected products, and systems now expose additional risk from hackers and malware. Safety-critical systems are to be developed with *Secure by design* methodology. For example, in mobile devices, **trusted platform modules** (**TPMs**) provide a restrictive area within the device to handle things, such as encryption, certificates, and keys. This impacts battery power, considered a non-functional requirement in DevOps parlance. Security is not siloed; it's part of the development cycle, in the similar way that product, quality, and performance are in-built. Addressing these significant concerns means organizations that use modern development practices are well set up to cope with the challenges of IoT.

DevOps adoption is more than science; it is an art for the digital transformation journey in the light of DevOps adoption to systems of big data, Cloud, microservices, data sciences, authentication security, and IoT.

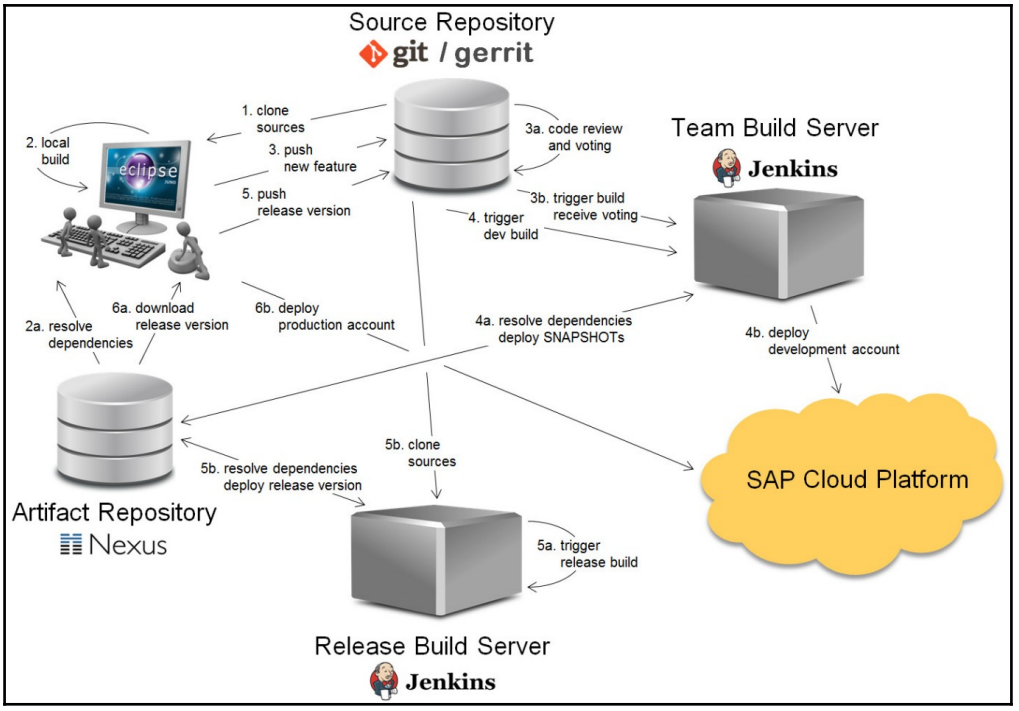
Summary

In this chapter, we learned about the digital transformation journey with DevOps adoption for complex enterprise systems of big data migration, cloud migration, microservices migration, data science, authentication systems, and Internet of Things.

11

DevOps Adoption by ERP Systems

The following is one of the high-profile adoptions of the ERP system (SAP) for the DevOps process at each stage of its cycle:



Reference: https://cloudplatform.sap.com/capabilities/devops.html#

As we can see from the preceding process, on mature DevOps adoption, replacing even the traditional SAP life cycle manager for the following activities involves:

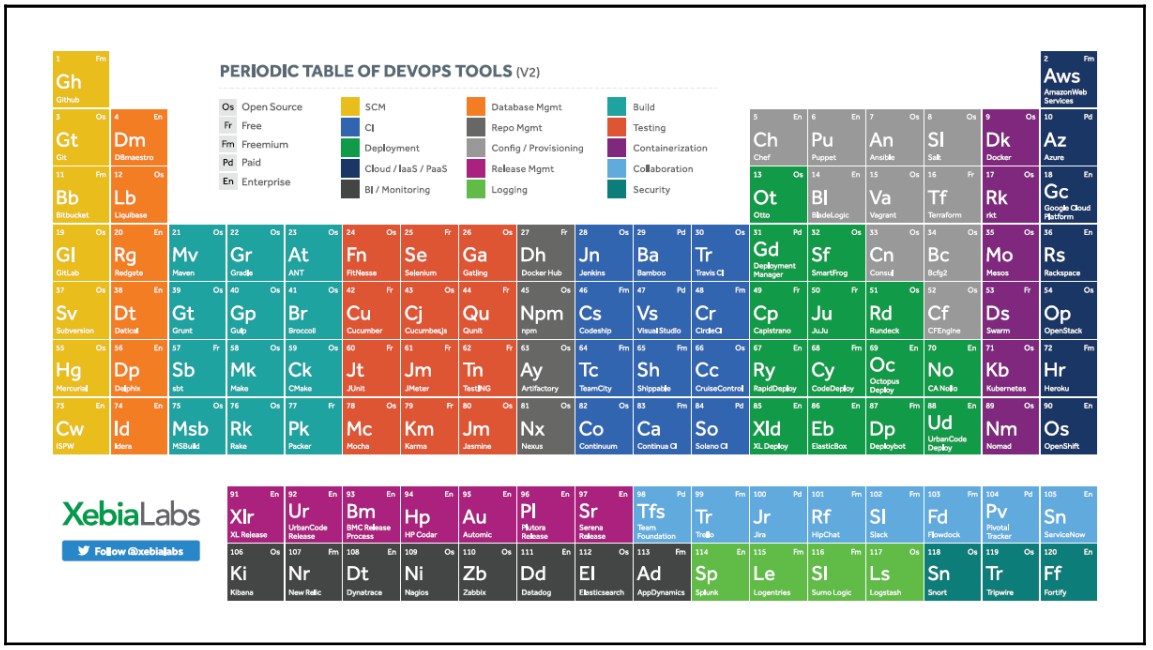
- Eclipse integrated development environment (IDE)-based local development
- Git/Gerrit is used for code repository
- Nexus is used for artifact repository
- Jenkins for Release Build Server for continuous integration and deployment
- SAP Cloud platform is adopted for production deployments

Users benefit from the agility and customization of features by adopting the open source tools and DevOps process. The procurement and maintenance costs of the SAP life cycle manager are saved for the organizations.

12

DevOps Periodic Table

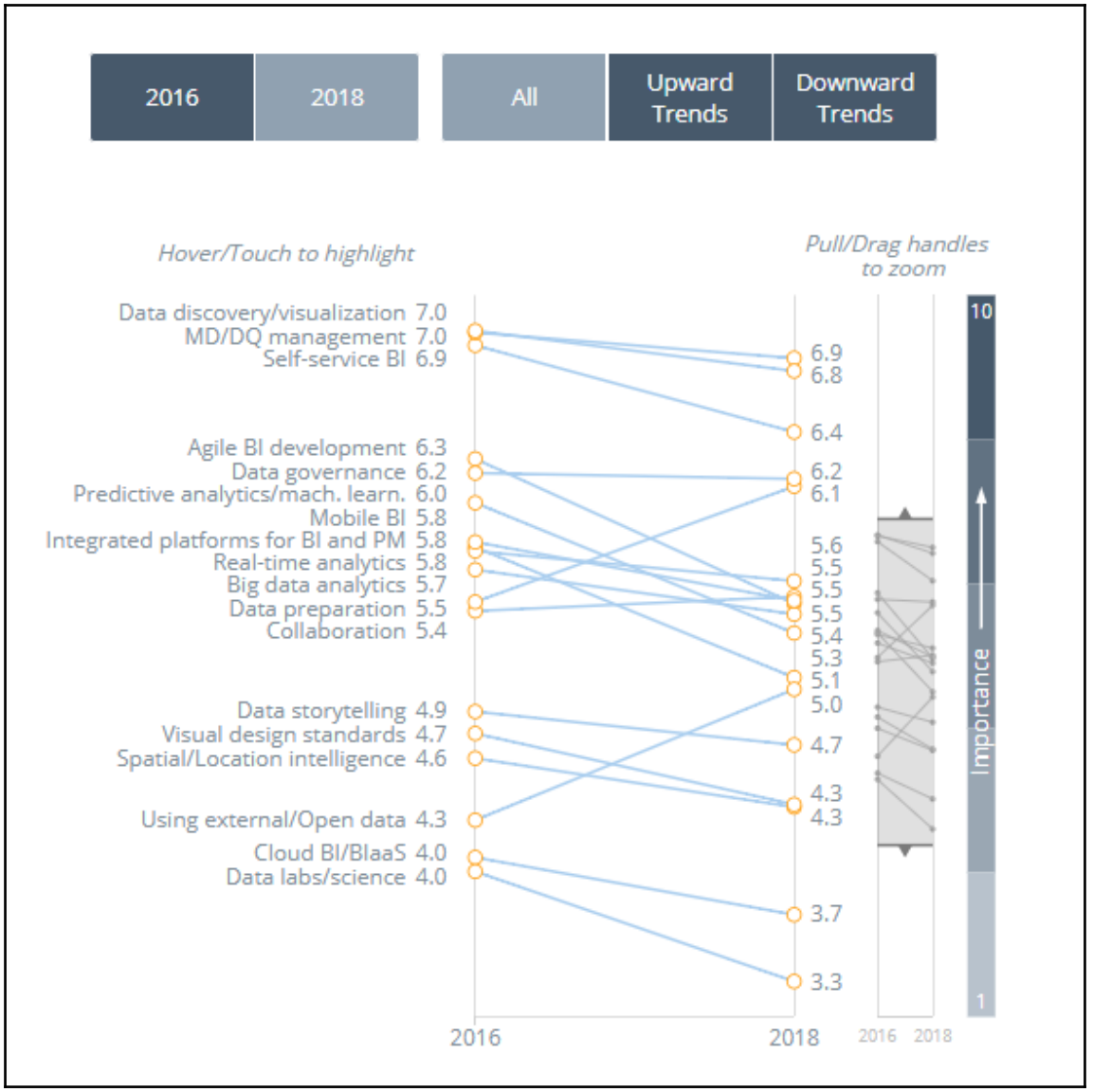
DevOps comprises of multiple overlapping phases, many tool offering are available for each service area, and many times the tools cover multiple phases. The entire set of tools and their offerings are listed in periodic table for simplicity and convenience:



Ref: https://xebialabs.com/periodic-table-of-devops-tools/

13
Business Intelligence Trends

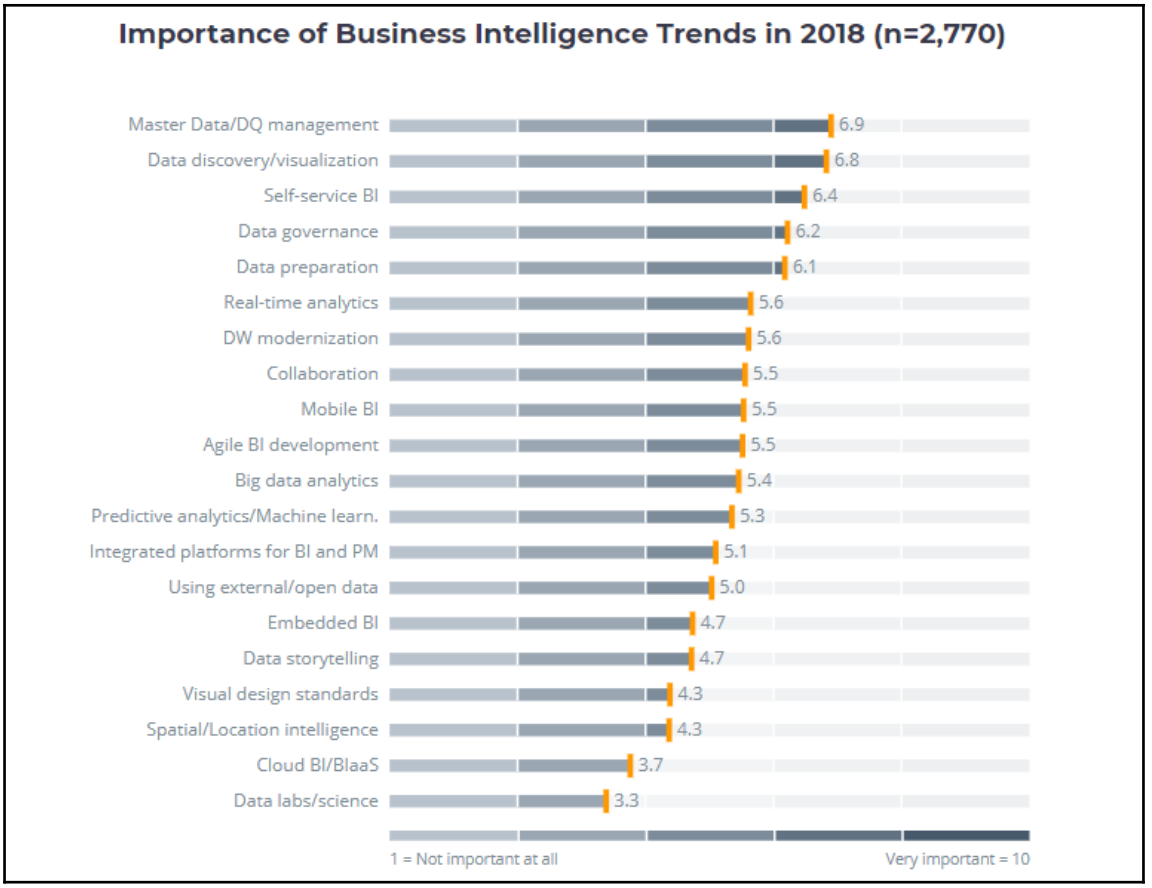
Business Intelligence trends variation and analysis is presented as following:



Ref: https://bi-survey.com/top-business-intelligence-trends

As we can see from preceding picture data preparation and usage of open and external data is considered to be high priority of consideration in the industry.

Following picture depicts that data discovery, preparation, governance, and visualization along with MDM/DQ are the focus areas trending for 2018:

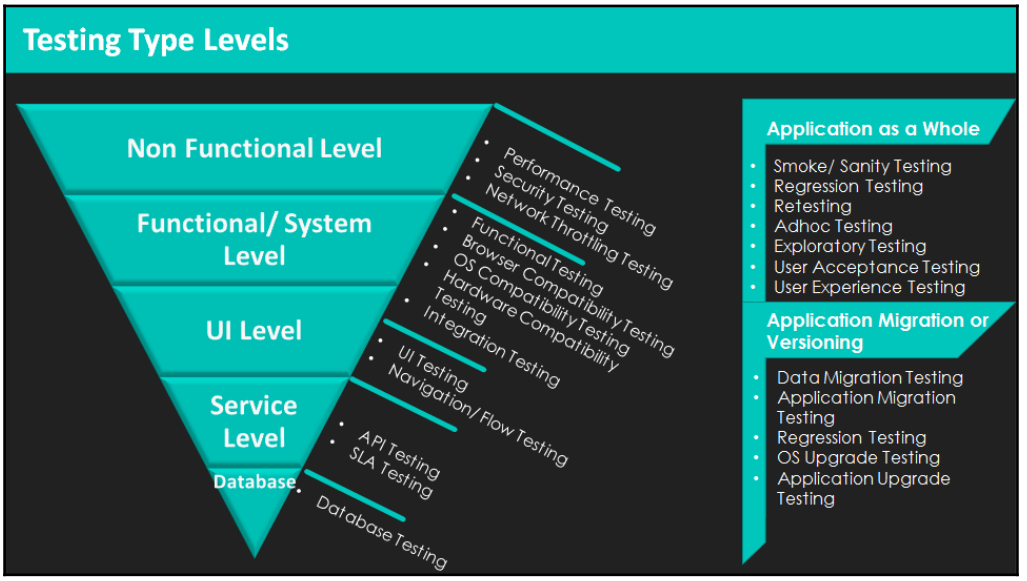


Ref: https://bi-survey.com/top-business-intelligence-trends

14

Testing Types and Levels

An end-to-end testing is critical aspect of application life cycle. Its understanding is vital to DevOps to automate as many segments as possible. The following listing covers the entire spectrum of testing variants:



- **Functional testing**: Verifying the functionality of the application based on business rules and requirements.
- **API testing**: Verifying the JSON request and response using Postman for various services. Using Postman, creating and automating test cases.
- **SLA testing**: Verifying the request and response times of different services/ pages and comparing them with SLA's metrics provided by the development team.
- **UI testing**: Comparing the UI elements / graphics based on the Wire Frames or Creatives provided by the design team.
- **Compatibility testing**:
    - **OS compatibility testing**: Verifying the application in different OS platforms, such as Windows, Mac, and Linux, and with different Mobile OS (iOS, Android, BlackBerry, Windows, Symbian, and so on).
    - **Browser compatibility testing**: Verifying the application in different Browser platforms, such as Firefox, Chrome, IE, Safari (iOS), and Chrome (Android).
    - **Hardware compatibility testing:** Verifying the application in different hardware requirements.
- **DB testing**: Verifying the data is modified/created or deleted based on the operations performed on the UI side (frontend).
- **Migration testing**: Verifying the user's data is migrated properly after rewriting the site from v1.0 to v2.0.
- **Integration testing**: Verifying the API services after testing and integrating with all services.
- **Performance testing**: Verifying the application performance on various work loads and also testing memory leaks using Jmeter.
- **Security testing**: Verifying the application's vulnerabilities using VERA Code.
- **User experience testing:** Verifying the application accessibility and user friendliness.
- **Localization testing**: Verifying the content of the application in different languages (for example, Chinese, Japanese, Spanish, French, Italian, Portuguese, and so on).
- **Smoke/Sanity testing**: Verifying the application basic functionalities before performing any major tests on the new builds.

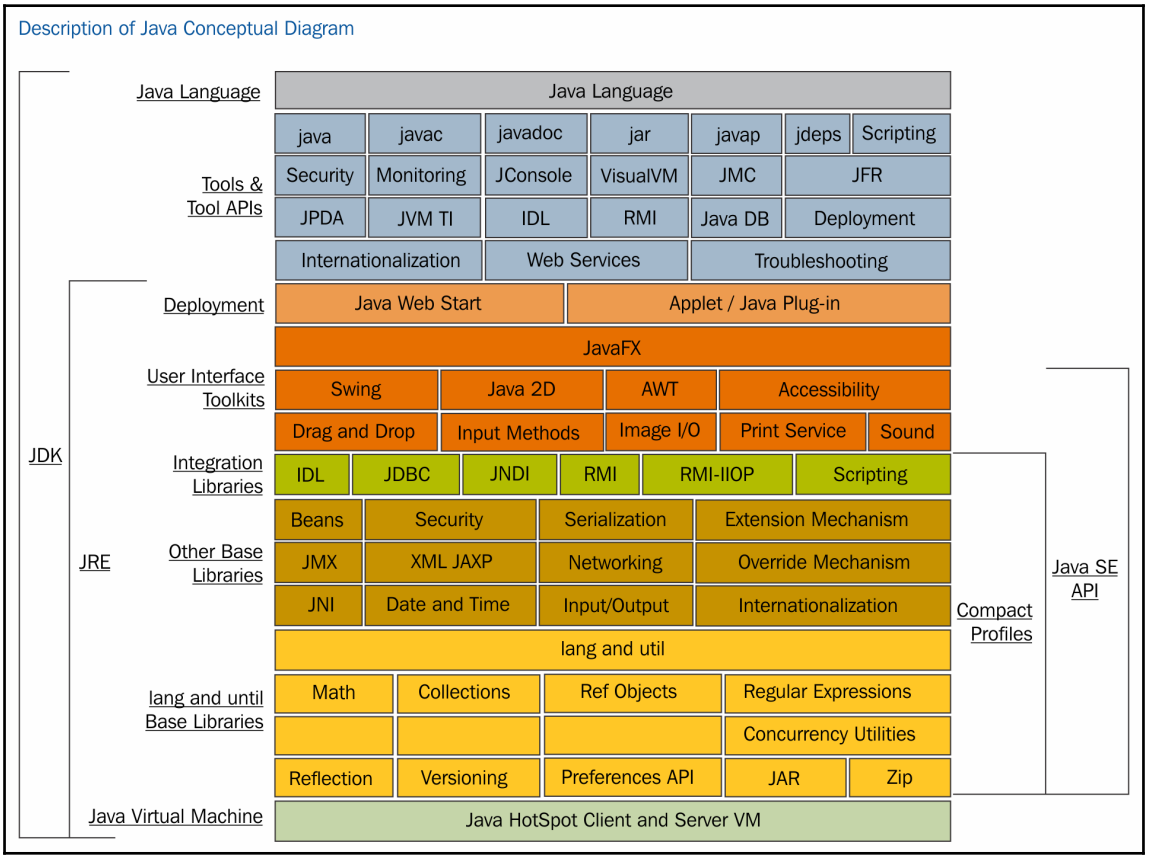
- **Regression testing**: Verifying affected areas when there is any bug fix or new feature/functionality implemented.
- **Ad hoc testing**: Verifying the application functionality randomly without knowing about the application behavior.
- **Exploratory testing**: Verifying the application functionality randomly by knowing the application behavior.

15
Java Platform SE 8

Oracle has two products that implement **Java Platform Standard Edition** (**Java SE**) 8, **Java SE Development Kit** (**JDK**) 8 and **Java SE Runtime Environment** (**JRE**) 8.

JDK 8 is a superset of JRE 8, and contains everything that is in JRE 8, plus tools such as the compilers and debuggers necessary for developing applets and applications. JRE 8 provides the libraries, the **Java Virtual Machine** (**JVM**), and other components to run applets and applications written in the Java programming language. Note that the JRE includes components not required by the Java SE specification, including both standard and non-standard Java components.

The following conceptual diagram illustrates the components of Oracle's Java SE products:



Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y