

# Index

<b>S.NO.</b>	<b>Content</b>	<b>Page No.</b>
1.	Azure Essential	1
2.	Azure Essentials Continuum	26
3.	Jira	74
4.	Prodigious Git	111
5.	Interstellar Git	144
6.	Continuous Integration	157
7.	Maven - Coalescing Pipeline	180
8.	Gradle - Fabricating Systems	213
9.	Continuous Integration with Jenkins	242
10.	Docker - Container Orcas	266
11.	Kubernetes K8s - The Basics	304
12.	Ansible - Automation Sibelius	318
13.	SonarQube	353

# Azure Essentials

Welcome to the course, **Azure Essentials!**

In this course, we will explore the broad range of ***services offered by Azure*** and deep dive into few of services that you can use for your applications.

**Azure Essentials** is the first of the series of courses on **Azure**. Have a great learning!

Please note that this course has been curated using the materials/resources received through our partnership with Microsoft. Hence, you could see that the content for this course has been taken from [Microsoft official sites](#).

## What is Microsoft Azure?

Microsoft Azure is a set of unified cloud services, which help IT professionals and developers to ***build, deploy and manage applications*** through the global network of Azure data centers.

## Overview of Azure Services

**Azure provides cloud services** that can be used to design and implement your customized cloud solution and infrastructure. They allow you to:

- Migrate on-premises datacenter to Azure cloud
- Deploy cloud-based applications
- Host workload in the Azure cloud
- Integrate Azure cloud services with an on-premises infrastructure

Azure cloud services can be categorized as ***Compute, Network, Data and Storage, App Services***, etc. These are few to name and there are much more to help with ***Identity and Access Management, Automation, Security, Availability***, etc.

# Azure Cloud Services

## Azure as IaaS (Infrastructure as a Service)

Allows the user to *access, manage and monitor the data centers*. Thus, giving *complete control* of the OS and the application platform stack to the developers.

- The virtual machine can be completely modified to meet business requirements.
- IaaS facilitates efficient *design time portability*. Hence an application can be migrated to Microsoft Azure without rework.
- IaaS allows a *quick transition of services to cloud*, which helps the vendors to offer services to their clients easily.

IaaS is perfect for the applications where complete control is required.

## Azure as PaaS (Platform as a Service)

The client is provided with the *platform to develop and deploy software*, without having to think about hardware and infrastructure. It takes care of most of the OS, servers and networking issues.

**PaaS is fast** with less hassle for developers; applications can go from idea to availability more quickly.

**PaaS is cost-effective** with lower upfront investment and less admin / management work for organizations.

**PaaS lowers risk** as platform is upto date with latest technology stack and tools for automation.

## Azure as SaaS (Software as a Service)

**Software as a service (SaaS)** allows users to connect to and use cloud-based apps over the Internet, such as **Office365**.

SaaS customers use the software running on the provider's infrastructure. SaaS is also referred as **software delivered over the web**.

### Advantages:

- Gained access to sophisticated enterprise applications
- Pay only for what you use
- Use free client software
- Mobilise your workforce easily

- Access app data from anywhere

## Azure Datacenters



Azure is backed by a *global network of data centers* that aims to meet global customer needs, ensure high application performance and maintain availability.

## Accessing Azure

Azure can be accessed and managed through:

- Azure Classic Portal
- Azure ARM portal
- Azure Resource Manager
- Client Tools like
  - PowerShell
  - Azure CLI

- Visual Studio with Azure SDK for .NET

- Azure Classic Portal

The screenshot shows the Microsoft Azure Classic Portal interface. At the top, there's a navigation bar with 'Microsoft Azure' on the left, a dropdown arrow, 'Check out the new portal' in blue, 'CREDIT STATUS' in orange, and a user account 'user@hotmail.com' with a profile icon on the right.

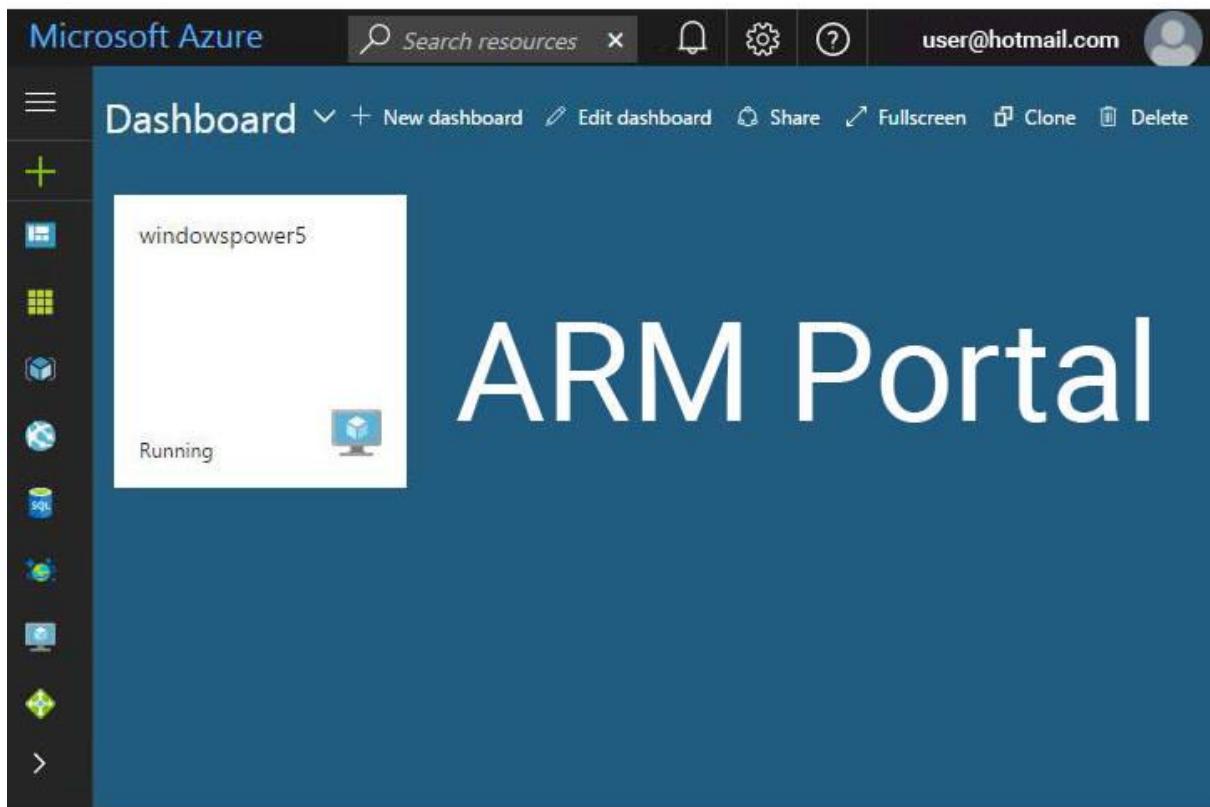
The main area is titled 'cloud services'. On the left, a sidebar lists 'ALL ITEMS', 'WEB APPS' (0), 'VIRTUAL MACHINES' (0), and 'CLOUD SERVICES' (1). Under 'CLOUD SERVICES', there are links for 'BATCH SERVICES' (0), 'SQL DATABASES' (0), 'STORAGE' (1), 'HDINSIGHT' (0), 'MEDIA SERVICES' (0), 'SERVICE BUS' (0), and 'MOBILE ENGAGEMENT' (0). A 'NEW' button with a plus sign is at the bottom of this sidebar.

In the center, a table displays a single row for a 'Cloud Service' named 'az-b48e'. The table columns are 'NAME', 'SERVICE STATUS', 'PRODUCTION', 'STAGING', 'SUBSCRIPTION', 'LOCATION', 'URL', and a search icon. The 'NAME' column shows 'az-b48e' with a right-pointing arrow. The 'SERVICE STATUS' column shows a checkmark and 'Created'. The 'LOCATION' column shows 'MSDN Platforms Southeast'. The 'URL' column shows 'http://az-b48e'. Below the table, there are 'SWAP' and 'DELETE' buttons, and a status indicator showing '1 !' (one warning).

# Classic Portal

- This was the first portal in Azure that was being used before the launch of Azure Resource Manager (ARM). It was based on the *Service Management model* and provides *limited Role-Based Access Control (RBAC) support*.

# Azure Resources Manager



**Azure Resources manager (ARM portal)** is now the default portal for Azure cloud services management. It supports new features like:

- Templates based deployments
- Role-based Access Control (RBAC)
- Customized dashboards to view key resources

## Client Tools

While Azure portals provide a GUI for managing your Azure subscriptions and services, in some scenarios, these portals may not offer the most optimal management capabilities.

For teams that want to ***perform service management in an automated fashion*** by using REST API and creating scripts for repetitive or cumbersome administrative tasks, Azure offers options like:

- **Azure PowerShell modules** - to run scripts from Windows.
- **Azure command-line interface (CLI)** - to run scripts on all operating systems like Windows, Linux, and macOS.

# Azure Network Services

**Microsoft Azure Network Services** offer the foundation for developing hybrid cloud solutions with the help of following essential resources.

- **Azure Virtual Network:** Isolated network within the Microsoft Azure cloud.
- **Azure Traffic Manager:** Controls how user traffic is distributed between geographies in cloud services.
- **Name Resolution Service:** For internal hostname resolution within a cloud service.
- **Azure ExpressRoute:** Extend on-premises networks into the virtual network over a dedicated private connection facilitated by a connectivity provider.
- **Application Gateway:** works at the application layer and acts as a reverse-proxy service, terminating the client connection and forwarding requests to back-end endpoints.

## What is Virtual Network?



Virtual Network, also known as a **VNet** constitutes a *logical boundary defined by a private IP address space* that you designate. You can distribute IP address space into one or more subnets. *This makes it functionally equivalent to on-premises networks.*

VNets are similar to AWS VPC (Virtual Private Cloud), offering various networking features like the *ability to customize inter-VM connectivity, Virtual Private Networks (VPN), access control, DNS, routing, and DHCP blocks*.

## Why Virtual Network?

Azure Virtual Network allows to **securely connect cloud infrastructure to your on-premises datacenter**.

- Virtual Networks allow to set up a virtual lab in the cloud by enabling connectivity to on-premises resources with the help of *Point-To-Site* and *Site-to-Site* VPN connections.
- Virtual Network also acts as a DHCP server, which allows configuring a *DNS server* to be leased out when a virtual machine is spun up in the cloud.

## VNet Capabilities

- **Isolation** - VNets are isolated from one another. One can create *separate VNets for development, testing, and production* that use the same CIDR address blocks.
- **Internet Connectivity** - By default, all Azure Virtual Machines (VM) and Cloud Services role instances are connected to a VNet and have access to the Internet.
- **VNet Connectivity** - VNet to VNet gateway needs to be configured to establish a connection between VNets.
- **On-premises Connectivity** - VNets can be connected to on-premises networks through *point to site, site to site*.

## VNet Capabilities...

- **Azure Resource Connectivity** - Azure resources such as Cloud Services and VMs can be connected to the same VNet. These *resources can connect to each other using private IP addresses*, even if they are on separate subnets.

*Azure offers default routing between subnets, VNets, and on-premises networks, thus avoiding the need to configure and manage routes.*

- **Traffic Filtering** - VM and Cloud Services role instance network traffic can be filtered outbound and inbound by destination IP address and port, source IP address and port, and protocol.

- **Routing** - Azure allows User-defined routes and BGP routes.
- **Load balancing and traffic direction** - Load balances traffic to servers.

## VNet Components - Subnets

A **subnet** is a range of IP addresses in the VNet. We can divide a VNet into multiple subnets for organization and security.

Additionally, we can configure VNet routing tables and Network Security Groups (NSG) to a subnet.

## VNet Components - IP Addresses

There are two types of IP addresses that can be assigned to an Azure resource:

- **Public IP Address** is used for internet/public-facing communication.
- **Private IP Address** is used for communication within a VNet, and when using VPN gateway or ExpressRoute.

Both Public and Private IP Address can be assigned through DHCP (Dynamic Host Configuration Protocol).

- **Dynamic IP** is allocated by default to the VM from the subnet via DHCP. When VM is started/stopped, the IP may be released/renewed based on the DHCP lease.
- **Static IP** can be allocated to a VM, which is only released when the VM is deleted.

## VNet Components - NSGs

Network Security Groups (NSGs) **allow or deny traffic** (through a rule base), to either a network interface or a subnet. **By default the outbound and inbound rules include an implied deny all**.

NSGs are **stateful**, meaning that the TCP sequence numbers are checked in addition to checking if the connection is already established.

## Network Services - Load balancing

Azure provides three different load balancing solutions:

- **Azure Traffic Manager**: DNS is used to direct traffic to the necessary destination. There are three destination selection methods - failover, performance or round robin.

- **Azure Load Balancer:** Performs L4 load balancing within a Virtual Network. Currently only supports round robin distribution.
- **Azure Application Gateway:** Performs L7 load balancing. Supports HTTP request based load balancing, SSL Termination, and cookie-based persistence.

## Network Services - DNS and Routing Tables

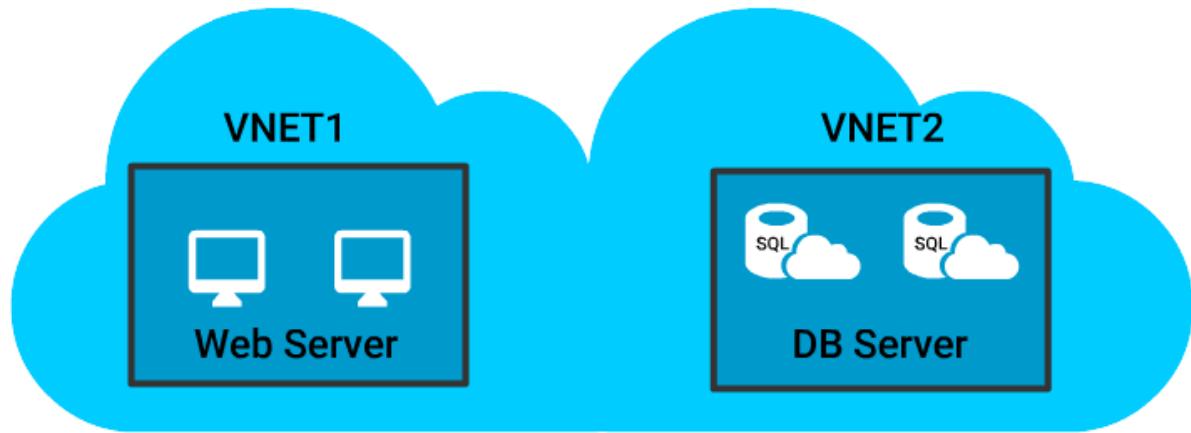
- **DNS name resolution** - Built-in (default) and support for custom (customer-owned) DNS.
- **Routing Tables** - Azure provides user defined routes and forced tunneling methods.

## Intersite Connectivity - Methods

There are two types of gateways.

- **VPN** - Traffic is encrypted within the endpoints by the following modes:
  - **Site-to-Site** - Traffic is secured using IPSEC/IKE between two VPN gateways, for example between Azure and an on-premise firewall.
  - **Point-to-Point** - Via a VPN client, a user connects to Azure, and traffic is encrypted using TLS (Transport Layer Security).
  - **VNet-to-VNet** - Traffic is secured between two Virtual Networks using IPSEC/IKE.
- **Express Route** - It provides a dedicated peered connection into Azure.

## Intersite Connectivity Detailed



- **VNet to VNet Connectivity** - VPN can be used to connect two or more Azure VNets. Such connections are termed VNet-to-VNet VPNs.
- **A Point-to-Site VPN** - connects a single computer to a VNet. To create this connection, you must *configure each on-premises computer* that you want to use, with the resources in the VNet.
- **A Site-to-Site VPN** - connects an on-premises network and all its computers to a VNet. To create this connection, you must configure a gateway and IP routing in the on-premises network. But it is *not necessary to configure individual on-premises computers*.
- **ExpressRoute Connectivity** - An ExpressRoute connection is a *dedicated server* that does not connect to the public Internet. By using ExpressRoute, you can *increase security, reliability, and bandwidth*.

# Azure Compute Services

Microsoft Azure Compute Services **offer the processing power for running cloud applications.**

The Microsoft Azure Compute Service can run many different kinds of applications. A principal goal of this platform, however, is to **support applications that have a substantial number of simultaneous users.**

## Compute Options in Azure

- **Virtual Machines** is an IaaS service that allows you to deploy and manage VMs inside a VNet.
- **App Service** is a managed service to host mobile app backends, web apps, RESTful APIs, or automated business processes.
- **Service Fabric** is known as a distributed systems platform that operates in numerous environments. Service Fabric is an orchestrator of microservices across a cluster of machines.
- **Azure Batch** is called a managed service for operating large-scale parallel and high-performance computing (HPC) applications.
- **Cloud Services** is a managed service for operating cloud applications and utilizes a PaaS hosting model.

## Resource Groups

Resource groups are **containers that are automatically created** for VMs, DBs, and other assets that are required for your solution or only the resources that you want to manage as a group.

They provide a way to **monitor, control access, provision and manage billing** for collections of assets that are required to run an application.

## Resource Groups - Points to Remember

- All the **resources in a group should share the same lifecycle** i.e. deploy, update, and delete them together.
- If a resource (e.g. DB server), needs to exist on a different deployment cycle, it should be in another resource group.
- **A resource can only exist in one resource group.** It can be added, moved and deleted from a resource group at any time.

- A resource group can include resources that *reside in different regions*.
- A resource group can be used to *control access*.
- A resource can *interact with resources in different resource groups*. (Scenarios where two resources are related but do not share the same lifecycle).

## What is Azure VM?

It is a **general-purpose computing environment** that lets you create, deploy, and manage VMs running in the Microsoft Azure cloud.

Azure VM's can be used in following ways:

- **Development and test** - Azure VMs provide a fast and effortless way to create a computer with particular configurations needed to code and test an application.
- **Extended datacenter** - VMs in an Azure virtual network can easily be connected to organization's network.
- **Applications in the cloud** - since the demand for an application can fluctuate, it might create economic sense to operate it on a VM in Azure. Thus one has to pay for additional VMs only when required and shut them down when they don't.

## Different Ways to Create VM

- **Azure CLI** - used to create and manage Azure resources from the command line or by using scripts.
- **Azure Portal** - provides a browser-based user interface for creating and configuring virtual machines and all related resources.
- **Azure PowerShell** - used to create and manage Azure resources from the PowerShell command line or scripts.
- **Resource Manager template** - a JSON file is used to define one or more resources to be deployed to a resource group and define the dependencies between the deployed resources. *This template can be used to deploy the resources consistently and repeatedly.*

## VM Size

The VM size is determined by the workload that you want to run. The size then determines factors such as processing power, memory, and storage capacity.

Following VM sizes are available:

- **A-series:** is basic with no load balancing or auto-scaling support.
- **D-series:** offers faster CPUs and local Hyper-V host SSD (temporary disk).
- **Dv2 series:** provides largest VMs with configuration up to 448 GB of RAM and 64 data disks. CPU is 35% faster than D-series.
- **DS, DSv2, and GS series:** Support for Premium Storage (SSD for operating system and data disks).

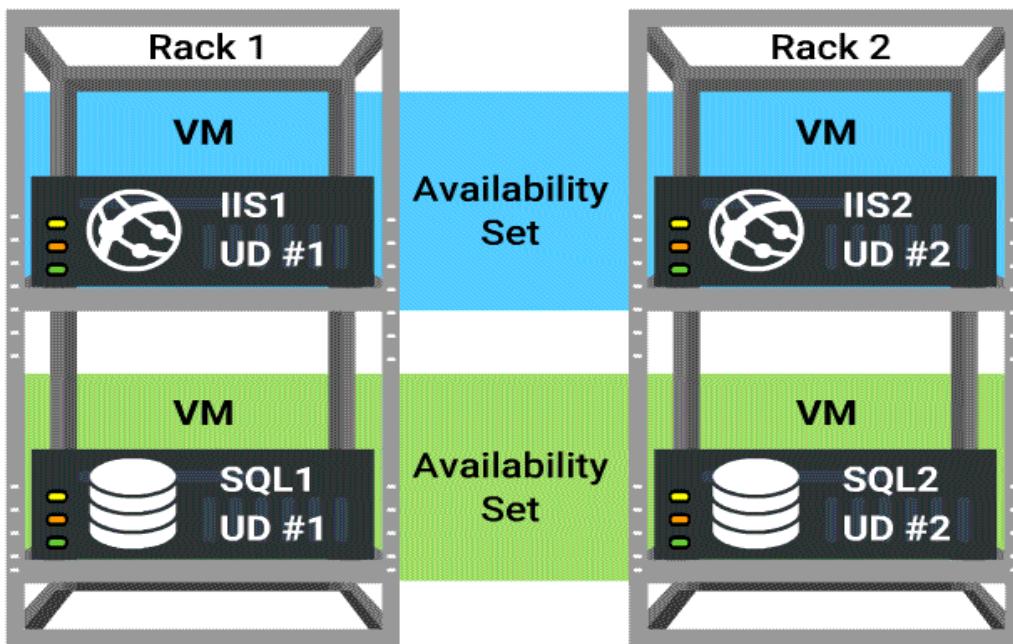
## VM Availability

To *ensure high availability* of an application, Azure places VMs into a logical grouping called an *Availability Set*.

When deployed with a service, Azure ensures that the VMs in the Availability set are arranged across Fault Domains on different Racks. In case of a maintenance event or failure of one fault domain, at least one VM keeps running.

Along with Load balancers, availability sets can provide up to 99.95% SLA for VMs.

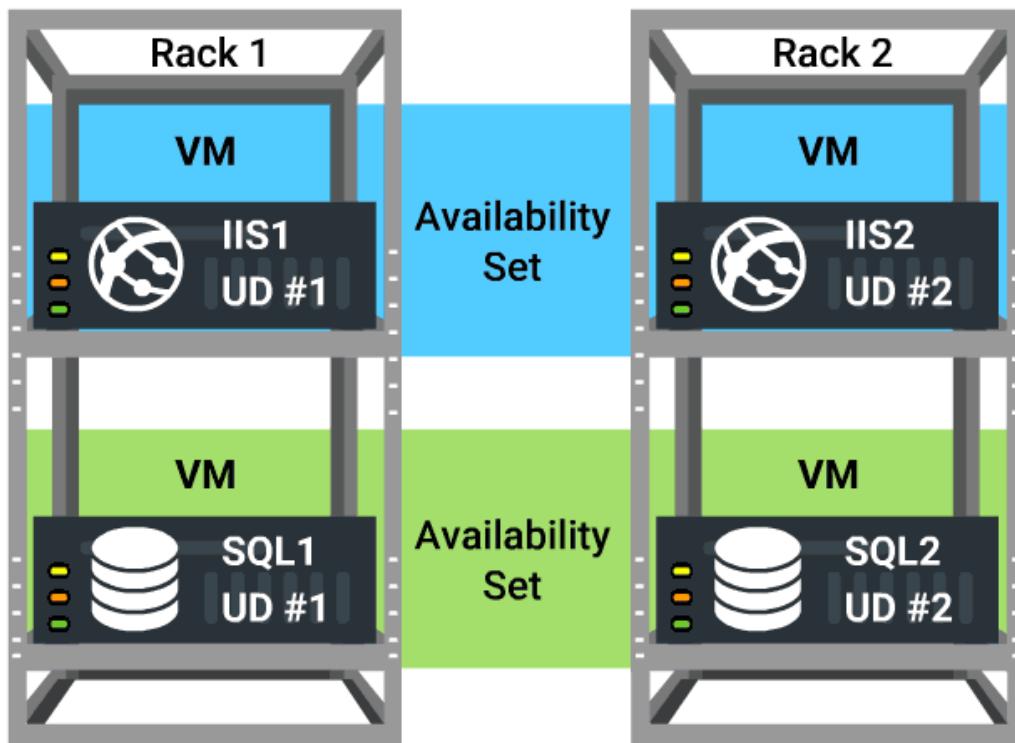
## VM Availability - Fault Domain



A fault domain is a set of hardware components (rack of resources like servers, power, etc.) that share a single point of failure. Web, worker and Virtual Machines are arranged in this hardware.

Azure deploys an application or service across multiple fault domains.

## VM Availability - Update Domain



Update domain in Azure means, that all physical servers in one update domain will get host updates like firmware, drivers and OS updates at the same time.

In the Illustration UD#1 is getting updated but the user can access the content from UD#2.

It provides Web or Worker role (within rack) instances with high availability by ensuring that only one of the Instances is down for an update at one time.

# VM Scaling

Scalability is known as the ability of a process, network, or system to accommodate fluctuating workload/demand.

- **Vertical scaling**, also known as scale up and scale down, *involves increasing or decreasing virtual machine (VM) sizes* in response to the workload, without creating additional VMs.
- **Horizontal scaling** also known as scaling out and in, *involves adding or removing instances of a resource*. The application continues operating without interruption as new resources are provisioned.
  - Once provisioning process is complete, the solution is deployed on the additional resources.
  - If demand drops, additional resources can be shut down cleanly and deallocated.

## Azure Storage Introduction

Microsoft Azure Storage is a highly scalable and robust storage solution for your applications.

## More about Azure Storage

- Is elastic and can *scale applications on demand*
- Uses auto-partitioning system that *automatically load balances* based on the traffic
- **Accessible** from any application, running on the cloud/desktop /on-premises server/mobile device
- Exposes data resources through **REST APIs**

## Azure Storage Services

Azure Storage offers four storage types.

- **Blob Storage** - for unstructured object data like images, videos, documents, etc.
- **Table Storage** - a NoSQL key attribute data store for structured datasets.
- **Queue Storage** - for storing a large number of messages. (Example: Creating backlog requests to be processed asynchronously OR for passing messages between various components.)

- ***File Storage*** - shared storage for sharing files across application components via File service REST API.

## Azure Storage Account

Azure Storage account is a secure account that provides access to Azure Storage Services. There are two types of Storage Accounts.

- ***General Purpose Storage Accounts*** that give access to blobs, tables, queues, files and Azure virtual machine disks.
- ***Blob Storage Accounts***, are specialized storage accounts for unstructured data as blobs. This type of account is recommended for applications that require just the ***block or append blob storage***.

## Storage Replication

Azure Storage Account offers multiple options to replicate data in a single data center / across facilities / across geographies to:

- **Ensure durability**
- **Higher availability**
- **Protect data** (against hardware failures or unforeseen catastrophes).

### Options available:

- Locally redundant storage
- Geo-redundant storage
- Zone-redundant storage
- Read-access geo-redundant storage

## StorSimple

StorSimple is a **Hybrid Cloud Storage Solution** to manage storage tasks between on-premise devices and cloud storage.

It has **physical arrays** for deployments in data centers and **virtual arrays** for smaller enterprise environments, which require network-attached storage (NAS).

Local storage gives the flexibility to retain part/entire data locally and could be useful for getting higher performance.

**StorSimple is ideal for applications requiring high performance and capacity.**

## Azure Backup

Azure Backup is a multi-tenant Azure service to back up and store data.

- Backup all your critical applications and data using the Azure Backup Agent.
- Azure ***Backup data is encrypted*** at the source, in transit, and at rest in the Azure.
- ***Configure the retention policy*** of backups as required (for example, 30 days, 99 years).
- ***Azure Site Recovery*** delivers a seamless portal experience for taking backups and operational recovery.

## Azure Database Services

Azure provides many Database services to cater to the wide range of needs of enterprises. Few them are:

- **Relational Database Services** - Azure provides SQL, MySQL, and PostgreSQL services as PaaS, making it easier to move existing workloads to Azure.
- **SQL Data Warehouse** - SQL based fully managed elastic data warehouse that can scale up and down as per demand.
- **Azure Redis Cache** - SaaS offering of Redis Cache that provides secure, dedicated cache for applications requiring low latency.

## Data Lake Store

Azure Data Lake Store is known as a **Highly Scalable Apache Hadoop file system** that can be used for enterprise-wide big data analytics workloads.

### Key Features:

- **Unlimited storage** and variety of data formats.
- Can **store data in native format** without the need for any transformation.
- Built for workloads requiring **massive read throughput** and analysis of large amounts of data.

- Can be accessed from Hadoop using the WebHDFS-compatible REST APIs.

## Azure Cosmos DB

Microsoft has developed Azure Cosmos DB to *support global distribution and horizontal scale*. It extends the idea of a *index-free database system*.

- Offers **turnkey distribution** by enabling seamless scaling based on user demand in any region or geography.
- Supports various new data types, making it flexible to **work as a graph database or key-value database**.
- **Experience Low latency**, owing to read and write from the nearest region.

## What are App Services?

App Service is a *Platform as a Service (PaaS)* that offers development framework to build and deploy mobile, web, logic, and API apps.

Create web and mobile apps for any platform or device. Azure runs these apps on *fully managed virtual machines (VMs)*, with users choice of shared VM resources or dedicated VMs.

Also, App Services allows connecting these apps to any SaaS or enterprise system within minutes and unlock the data.

## Key features and Capabilities of App Service

- **Multiple languages and frameworks** - supports ASP.NET, Node.js, Java, PHP, Python, etc.
- **DevOps optimization** - can set up continuous integration and deployment with Visual Studio Team Services, GitHub, or BitBucket.
- **Global scale with high availability** - can Scale up or down manually or automatically.
- **Security and compliance** - is ISO, SOC, and PCI compliant.
- **Application templates** - can choose from an extensive list of templates in the Azure Marketplace.
- **Visual Studio integration** - to streamline the work of creating, deploying, and debugging.

# Web App Services

Web App Services help in hosting your websites and web applications. **Main features include:**

- Support for .NET, Java, PHP, Node.js, and Python.
- High availability with auto-patching.
- Continuous deployment with Git, TFS, GitHub.
- Azure Marketplace based solutions that simplify the development and deployment.
- Multiple Deployment slots to run two or more versions of the same app (production and dev) concurrently on the same virtual machine.
- Azure WebJobs to execute background processes.
- Hybrid connections from web apps to access on-premises resources or VMs within an Azure virtual network.

# Mobile App Services

Mobile Apps is a highly scalable, globally available mobile application development platform for Enterprise Developers and System Integrators. With Mobile Apps you can:

- ***Build native and cross platform apps*** using native SDKs.
- Connect mobile apps to your enterprise on-premises or cloud resources.
- ***Build offline-ready apps with data sync*** - make your mobile workforce productive by building apps that work offline.
- ***Push Notifications to millions in seconds*** and engage your customers.

# Logic App Services

Use **Logic Apps** for automating business processes and integrating systems and data across clouds without writing code. Main features of Logic Apps:

- Visually create business processes and workflows
- Deliver integration capabilities in Web, Mobile, and API Apps
- Integrate with your SaaS and enterprise applications
- Automate business processes
- Connect to on-premises data

# API App Services

Azure API App Service offer a rich platform and ecosystem for ***building, consuming, and distributing APIs*** in the cloud and on-premises. Main features of this service are:

- Integrate with SaaS and enterprise applications
- Generate client proxies or APIs in language
- Automate versioning and deployment of API Apps
- Secure APIs with Single Sign-On, OAuth, and Active Directory
- Share APIs internally with organizational gallery

# Azure HD Insight

Azure HD Insight is widely used to deliver ***Hadoop as a service on top of the Azure platform***. It uses the ***Hortonworks Data Platform*** (HDP) Hadoop distribution.

Azure HD Insight gives **open-source analytic clusters** for Spark, Hive, Storm, Kafka, MapReduce, HBase, and R Server to deploy these big data technologies.

**Organizations can use HD Insight to:**

- Create Hadoop-powered big data solution and services
- Manage and monitor Hadoop clusters
- Analyze and report statistics on big data utilization and availability

# Azure Machine Learning

Azure Machine Learning is known as a **cloud predictive analytics service** that allows quick *\*creation and deployment of predictive models*.

It comes with ***ready-to-use library of algorithms***, to create models and deploy your predictive solution quickly.

Azure Machine Learning offers ***tools to model predictive analytics*** and offers a ***fully managed service*** for using predictive models as ready-to-consume web services.

# Azure Stream Analytics

Azure Stream Analytics strives to **gather knowledge structures from the continuous ordered streams of data** with real-time analysis.

The streams may comprise web searches, ATM transactions, sensor readings, phone conversations, social network data, or computer network traffic.

*It offers a ready-made solution for business requirement that involve handling large volumes of information and react very quickly to changes in data.*

## Azure Stream Analytics Usage

- **Connected devices** - Monitor and diagnose real-time data from connected devices to generate alerts, respond to events, or optimize operations.
- **Business operations** - Analyze real-time data to respond to dynamic environments to take immediate action.
- **Fraud detection** - Monitor financial transactions in real-time to detect fraudulent activity.
- **Website analytics** - Collect real-time metrics to gain instant insight into a website's usage patterns or application performance.

## Event Hubs

Azure Event Hubs is a **event ingestion service and scalable data streaming platform** that receives and processes numerous events per second.

Event Hubs processes and stores events, data, or telemetry created by distributed devices and software. Data transmitted to an event hub can be transformed and stored with the help of any real-time analytics provider or storage/batching adapters.

**Offering publish-subscribe capability with low latency and huge scale, Event Hubs acts as the "on ramp" for Big Data!**

## Azure Active Directory

Azure AD is known as an identity management solution that **offers access and identity services** for cloud resources. It is available as both on cloud and on-premises service. It helps to:

- Configure access to applications.
- Configure SSO to cloud-based SaaS applications.
- Manage users and groups.
- Provision users.
- Enable federation between organizations.

- Provide an identity management solution.
- Identify irregular sign-in activity.
- Multi-factor authentication.
- Extend existing on-premises AD implementations to Azure AD.

## Features of Azure AD



Azure AD includes a **full suite of IDM capabilities** such as,

### Access & Authentication

- Multi-factor authentication
- Device registration
- Role based access control

### Management

- Self-service password management

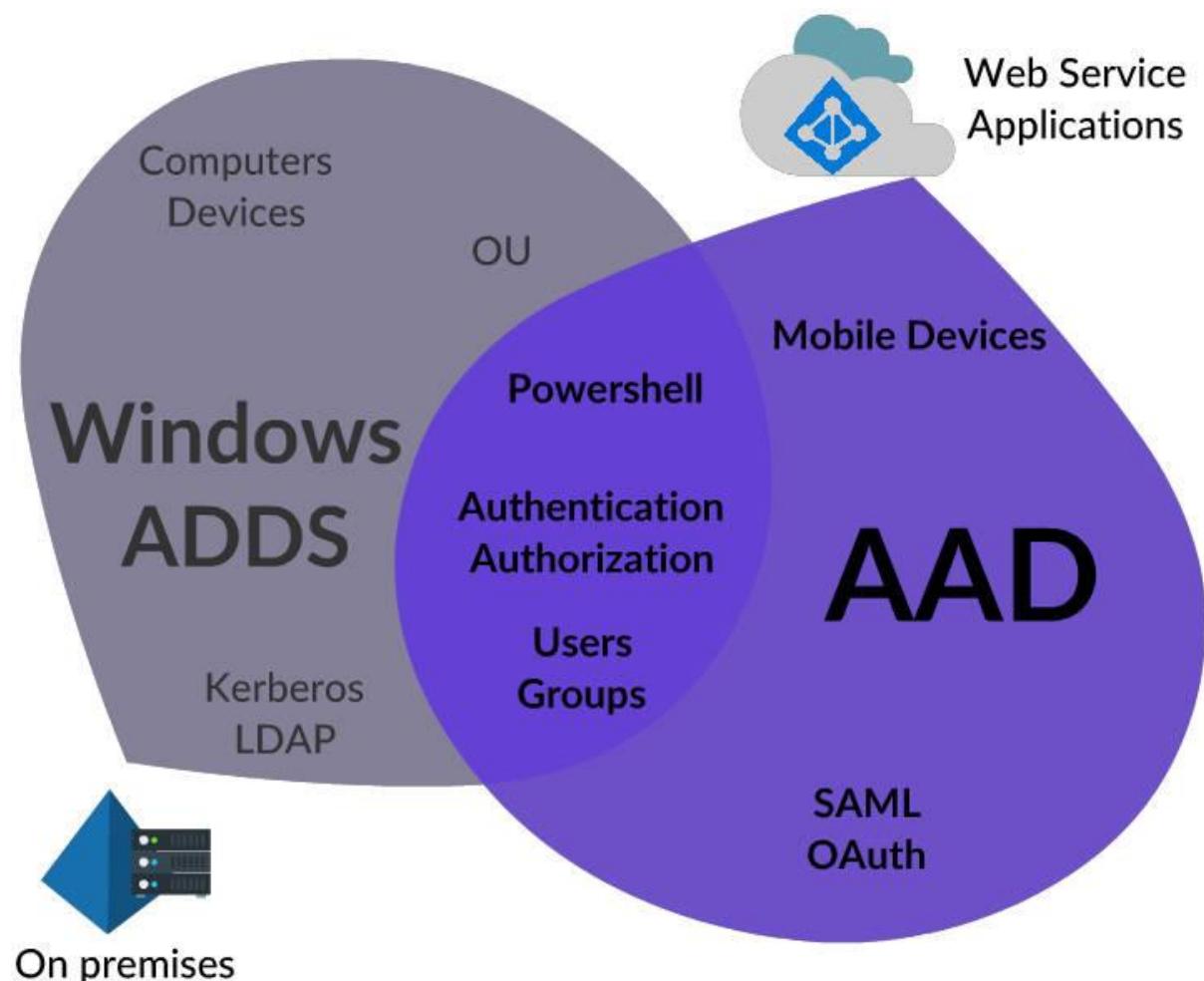
- Self-service group management
- Privileged account management

## Monitoring & Auditing

- Application usage monitoring
- Rich auditing
- Security monitoring and alerting

These capabilities can help **secure cloud-based applications, streamline IT processes, cut costs** and also help **assure corporate compliance goals are met.**

## Azure AD vs ADDS



Azure AD and Windows Server Active Directory (AD) are used for Authentication. But Azure AD differs in many aspects, such as;

- It doesn't have,
  1. Active Directory forest and Trust relations
  2. Organizational Units
  3. Group Policies
- It uses Open AD connect, O-Auth, WS-federation and SAML protocols for Authentication and Authorization.

## Azure AD Editions

Azure AD is available in free and paid editions such as,

- Free
- Basic
- Premium 1
- Premium 2

## Azure Security Center

Azure Security Center aids to **prevent, detect, and respond to threats** with increased visibility into and control over the security of Azure resources.

- Offers **integrated security monitoring and policy management** across Azure subscriptions.
- Aids in **detecting threats** that might otherwise go unnoticed.
- **Works with a broad ecosystem** of security solutions.

## Azure Key Vault

Azure Key Vault helps **safeguard cryptographic keys and secrets used by cloud applications and services**. It streamlines the key management process and enables to maintain control of keys.

By using Key Vault, you can **encrypt keys and secrets** (such as authentication keys, storage account keys, data encryption keys, etc.) by using keys that are protected by **Hardware Security Modules (HSMs)**.

Developers can create keys for testing and development in minutes, and then easily migrate them to production keys.

## Azure Essentials - Course Summary

Congratulations! You have come to the end of this course. By now, you must have got a good idea about various services offered by Azure for:

- Networking
- Computing
- Storage
- Database
- Analytics and IOT
- Few other important services

We will delve deeper into the internals of these services in a separate course.  
Keep Learning!

# Azure Essentials Continuum

## The Context

Welcome to the course, **Azure Essentials Continuum!**

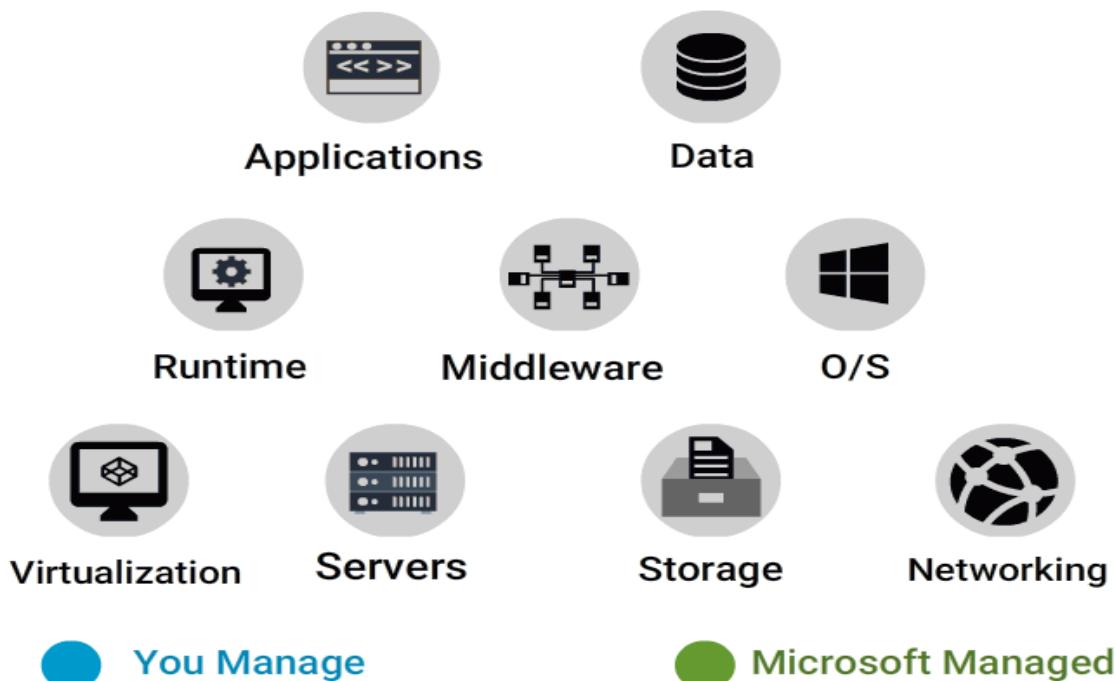
This is a sequel to the **Azure Essentials** course, and in this course, we will focus on the

- **Azure IaaS services** like Virtual Machines, Vnets and Storage
- **Azure PaaS services** like Web Apps and Azure SQL

This course will offer both conceptual understanding and **Hands-On lab exercises** for you to play around with.

Please note that this course has been curated using the materials/resources received through our partnership with Microsoft. Hence, you could see that the content for this course has been taken from [Microsoft official sites](#).

## Quick Refresher



Azure is Microsoft's cloud computing platform that enables individuals and organizations to create, deploy, and operate cloud-based applications and infrastructure.

It is highly scalable and flexible enough to integrate with your on-premises servers and data centers. Azure offers a number of services and resource offerings as:

- Infrastructure as a Service (**IaaS**)
- Platform as a Service (**PaaS**)

## Azure Services

Microsoft Azure provides cloud services for accomplishing various tasks and functions across the IT spectrum and those services can be organized into several broad categories.

- Compute and Networking Services
- Storage and Backup Services
- Identity and Access Management Services
- Application Services
- Data and Analytics Services
- Media and Content Delivery Services

## Azure Services: Compute, Storage, and Identity

### Compute and Networking Services

- **Virtual Machines**
- **Azure RemoteApp**
- **Azure Cloud Services**
- **Azure Virtual Networks**
- **Azure ExpressRoute**
- **Traffic Manager**

### Storage and Backup Services

- **Azure Storage**
- **Azure Import/Export Service**
- **Azure Backup**

- **Azure Site Recovery**

### **Identity and Access Management Services**

- **Azure Active Directory**
- **Azure Multi-Factor Authentication**

## **Azure Services: App, Data, and Media**

### **Application Services**

- **Azure App Services**
- **API Management**
- **Notification Hubs**
- **Event Hubs**

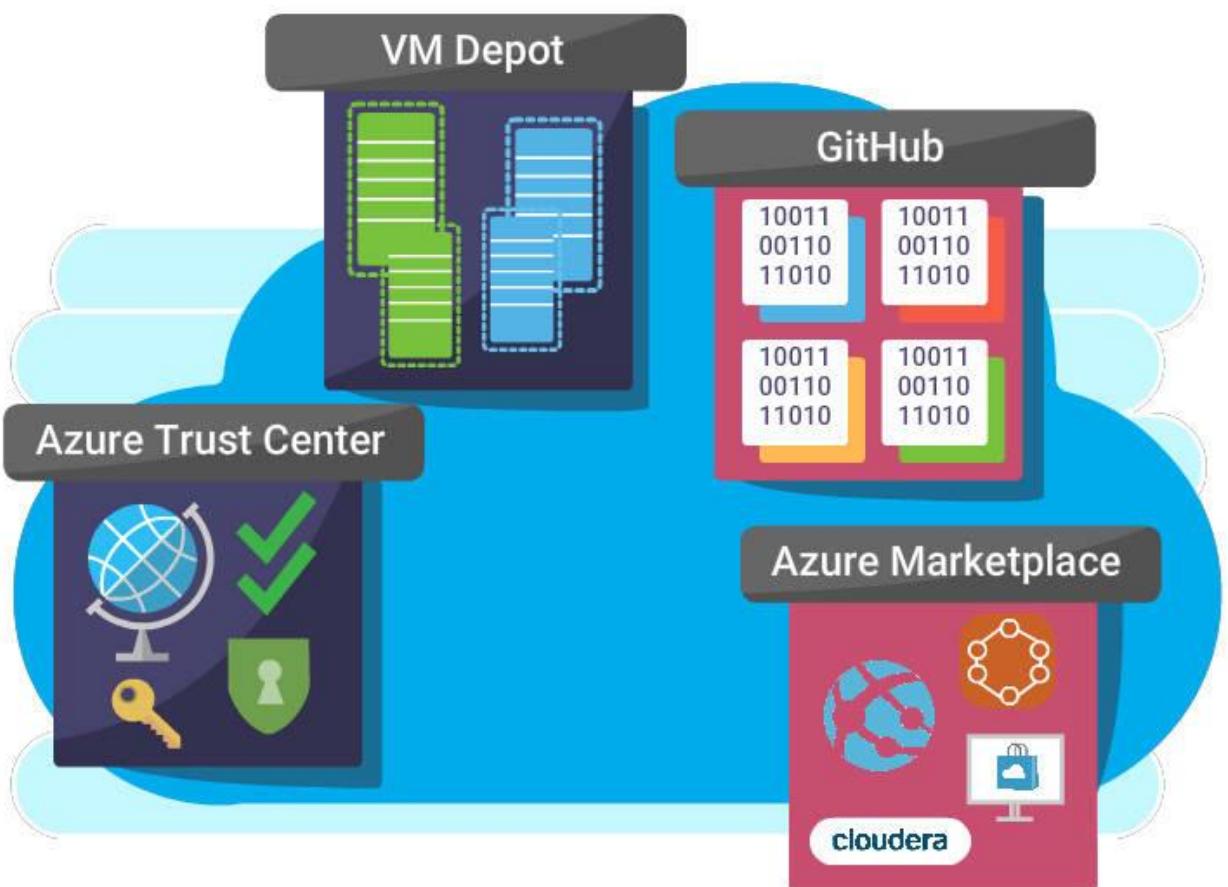
### **Data and Analytics Services**

- **SQL Database**
- **HDInsight®**
- **Azure Redis Cache**
- **Azure Machine Learning**
- **DocumentDB**
- **Azure Search**

### **Media and Content Delivery Services**

- **Azure Media Services**
- **Azure CDN**
- **Azure Service Bus**

# Azure Resources



Now, let's get familiar with some of the useful resources that will help you manage your Azure environment.

- **Azure Marketplace**
- **VM Depot**
- **GitHub**
- **Azure Trust Center**

## Azure Resources

**Azure Marketplace** - an online applications and services marketplace that offers,

- VM images and extensions
- APIs and Applications
- Machine Learning and Data services

**VM Depot** - a community-based catalog of open source virtual machine images that can be deployed directly from Azure.

**GitHub** - a web-based Git repository that is free to use for public and open source projects.

**Azure Trust Center** - offers guidelines for integrated security monitoring and policy management across Azure subscriptions. Also, it provides data security & data privacy guidelines essential to comply with regulatory controls.

## Azure Subscriptions

Azure is a subscription-based service. You need to sign up for using any service.

You can purchase Azure through a number of different licensing options based on your business need. There are no upfront costs and you only pay for what you use.

## Deploying Resources

Once you have your subscriptions in Azure, you will be able to deploy **Resources** like:

- **Virtual machine**
- **Storage account**
- **Virtual networks**
- **Services or any component** that you will be managing.

Azure offers two deployment models and multiple tools for you to be able to deploy and manage your resources.

# Azure Deployment Models

1. **Azure Service Management Model (ASM)** using Classic Portal. This was the first approach that was introduced by Microsoft. Here the resources are coupled and can be deployed using ASM PowerShell Module.
2. **Azure Resource Management Model (ARM)** using New Portal. The resources are decoupled and hence can be configured independently. JSON templates provide simple orchestration and rollback functions. They have their own ARM PowerShell Module as well.

**For example**, while deploying a VM - compute, Vnet and storage resources are coupled in ASM, and hence can not be configured independently. Whereas with ARM, these resources are can be configured independently.

ARM is the recommended model of deployment and in this course, we align to this model.

## Azure Resources Management

### Managing Resources

Azure offers a number tools that you can use to deploy, update or delete resources from your cloud solution. Resources can include virtual machines, storage accounts, virtual networks, services, or any component that you are managing.

Here are the tools at your disposal to manage resources,

- Azure PowerShell
- ARM and ASM Portals
- Azure CLI
- Azure Rest APIs
- Azure SDKs

**We will go in detail about these tools in this topic.**

# Azure PowerShell



PowerShell

Microsoft Azure

Azure PowerShell provides a set of cmdlets for managing your Azure resources and for Automating the scheduled task.

Azure PowerShell Module can be installed in several different ways,

- **From PowerShell Gallery**
- Using MSI installer from the **GitHub repository**
- Using **Microsoft Web Platform Installer**

In this course, we will cover the first two ways of installing PowerShell.

Installing Azure PowerShell from the PowerShell Gallery is most preferred

# Preparing to Install

Check if **PowerShellGet** is installed on your system using:

```
>Get-Module PowerShellGet -list | Select-Object Name,Version,Path
```

- (If PowerShellGet is not installed, download it by executing below command).

```
Install-Module -Name PowerShellGet -Force  
Exit
```

Once PowerShellGet is downloaded you are all set to Install PowerShell ARM or ASM modules. We will look into this, in next couple of cards.

Administrative Privilege is needed to install the Azure PowerShell and Scripting environment must be enabled.

## Install ARM Module

- You can install Azure Resource Manager (ARM) modules from the PowerShell Gallery using:

```
Install-Module AzureRM
```

- PowerShell gallery is not configured as a Trusted repository. Hence you will be prompted to “Allow installing modules from **PSGallery**”. Choose ‘Yes’ or ‘Yes to All’ to continue.
- Once installed, load the module in the PowerShell session using:

```
Import-Module AzureRM
```

- Validate the installation by Checking the version of Azure PowerShell using:

```
Get-Module AzureRM -list | Select-Object Name,Version,Path
```

# Install ASM module

- For Classic mode use following command to install Azure PowerShell,

```
Install-Module Azure
```

and

```
Import-Module Azure
```

- Validate the installation by Checking the version of Azure PowerShell using:

```
Get-Module Azure -list | Select-Object Name,Version,Path
```

## ARM and ASM Portals

- **Azure Service Management (ASM)** is the traditional way of provisioning Azure Resources. **ASM portal**, also called **Classic Portal** uses ASM API calls to provision and manage Azure resources.
- **Azure Resource Manager (ARM)** is the service that is now more widely used to provision resources on Azure. The **ARM Portal** also called the **New portal** uses ARM API calls to provision resources. Also, when you use tools like Azure PowerShell, you are invoking ARM API calls.

While Microsoft supports both these portals and API's, lot more features are being added to ARM on a regular basis. Hence it is recommended to use ARM - the new API's, and portal to provision and manage resources.

## Advantages of ARM

- **Resource Group** - ARM groups resources, making it much easier to manage them.
- **Smarter Provisioning** - ARM refers **ARM templates** to figure out resource requirement, dependencies and provisions resources optimally.

- **Role-based Access control(RBAC)** - gives greater control by configuring role-based access control at resource and resource group levels.
- **Tags** - can be applied to resources to logically organize all of the resources in the same subscription.
- **Billing** - In an organization, costs can be viewed for the entire group or for a group of resources sharing the same tag.

## ARM Templates

**ARM templates** are commonly used to automate deployment.

Resource groups and resource properties like the size of DB, type of storage account are defined in these **JSON documents**.

ARM inspects requirements in ARM templates, figures out resource dependencies and provisions resources in an order or even simultaneously when there is no direct dependency between certain resources.

- **GitHub Resource Manager** contains a number of [Azure Quickstart Templates](#) which can be used to build and deploy.
- Another option is to **create custom templates**, using [JSON Editor](#). **Visual Studio** or **Visual Studio Code** a lightweight, open-source editor can be used to create these templates. Visual Studio Code can be downloaded from [code.visualstudio.com](https://code.visualstudio.com).

# Azure CLI

## Command-Line Syntax Overview

Promt > azure <b>topic</b>	<b>verb</b>	<b>options</b>
account	download	username
account location	import	password
account affinity-group	list	dns-prefix
vm	show	vm-name
vm disk	delete	lb-port
vm endpoint	start	target-image-name
vm image	restrat	source-path
service	shutdown	disk-image-name
service cert	capture	size-in-gb
site	create	thumbprint
config	create-from	value
	attach	-json
	attach-new	-v
	detach	-vv
	browse	
	set	

**Azure CLI 2.0** is an open-source, cross-platform, shell-based command line interface for managing Azure resources.

You can use it in your browser with **Azure Cloud Shell**, or you can install it on macOS, Linux, and Windows and run it from the command line.

**Commands are structured as:**

```
>azure <topic> <verb> <options>  
or  
>az <topic> <verb> <options>
```

- Example to list the virtual machines within an account

```
>azure vm list
```

- Example to create a resource group named “MyRG” in the centerus region of Azure

```
>az group create -n MyRG -l centerus
```

## Azure REST APIs

One of the most powerful ways to manage Azure is via the Azure REST APIs. Representational State Transfer (REST) APIs are service endpoints that support sets of HTTP operations (methods) to manage your resources.

**It forms the connecting glue between your applications and Azure.**

### Test Drive Azure REST APIs

You can interact with the Azure REST APIs in a number of ways and methods, you can try out and experience interacting with the Azure REST APIs using <https://azure.github.io/projects/apis/>  
REST APIs adhere to the OpenAPI Specification (also known as Swagger 2.0).

## Azure SDKs

**Azure SDKs provide a framework which you can use to build, deploy and manage various solutions and services you may need on Azure.**

*Some Azure SDKs currently available for download are*

- .NET
- Java
- Node.js
- PHP
- Python
- Ruby
- GO

**Azure SDKs are downloadable**

**from <https://azure.microsoft.com/en-in/downloads/>**

*A series of SDKs available tailored for specific workloads or services are*

- IoT SDKs
- Media
- WebJobs

## Resource Management - Summary

By now you should have a fair idea about various tools you can use to manage resources on Azure.

Next, you will learn in detail about various Azure resources like Virtual Machine, Vnet, Storage, etc and try to install them on your Azure Subscription.

## Azure Virtual Machines

### Virtual Machines - F5

In the previous course **Azure Essentials** we had learned about various compute options offered by Azure. Also, we learned briefly about Virtual Machines and some of the key concepts like:

- Resource Groups
- Availability Set
- Auto Scaling
- VM creation Tools

Now, let us go further and understand VM sizes, supported OS and also create VM in your Azure Subscription.

## Azure VM

Azure Virtual Machines provides **Scalable Computing Environment** to host wide range Infra services and Applications. They offer **more control over the environment** than other compute options like App Service or Cloud Services.

Azure Virtual Machines follow **Infrastructure as a Service (IaaS)** model and lets you create VMs in the cloud. VM's are provisioned with:

- Compute
- Storage
- Networking capabilities
- Operating system

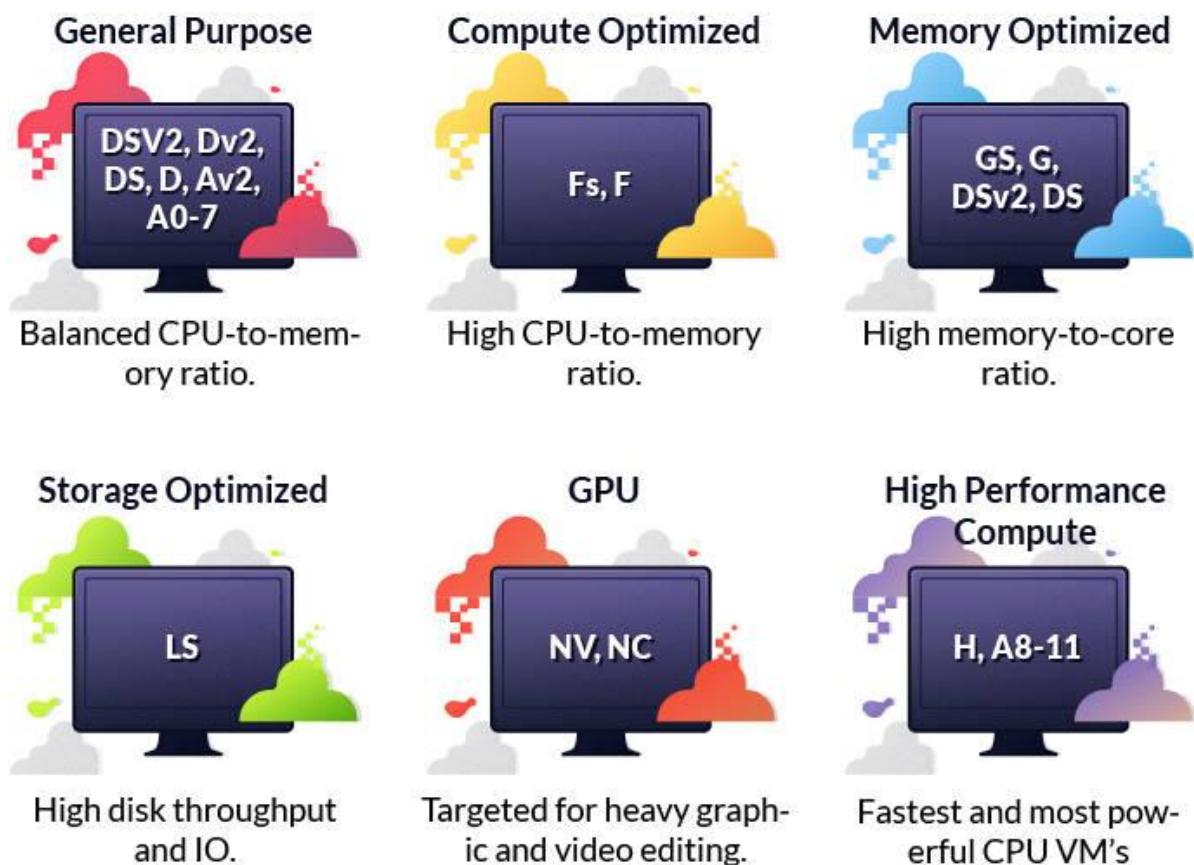
## When to use VMs

Azure VMs are best for workloads that:

- Require High availability
- Experience Unpredictable Growth
- Experience Sudden Spikes

Over next few cards, we will learn how to create Azure VMs.

## Virtual Machine Sizes



**The first Step to creating VM is to choose the right VM size.** Here are the various VM sizes that can cater to every kind of requirement.

- **General Purpose** - ideal for Testing and Development.
- **Compute Optimized** - recommended for medium traffic web servers and network appliances.
- **Memory Optimized** - used for relational DB servers, medium to large caches, and for in-memory analytics.
- **Storage Optimized** - used for Big Data, SQL, and NoSQL databases.
- **GPU** - Ideal for heavy graphic rendering and video editing.
- **High Performance Compute** - optimal for high-throughput network interfaces (RDMA).

## Azure Market Place



**Once the VM sizing is computed, OS images that are available in Azure Marketplace are used for provisioning a VM.**

Azure Marketplace provides a large image gallery, which includes:

- Recent operating system images of Windows Server, Linux, and SQL Server.
- You can also store your own images in Azure, by capturing an existing virtual machine and uploading the image.

## Supported OS for Azure VM

### Windows Server

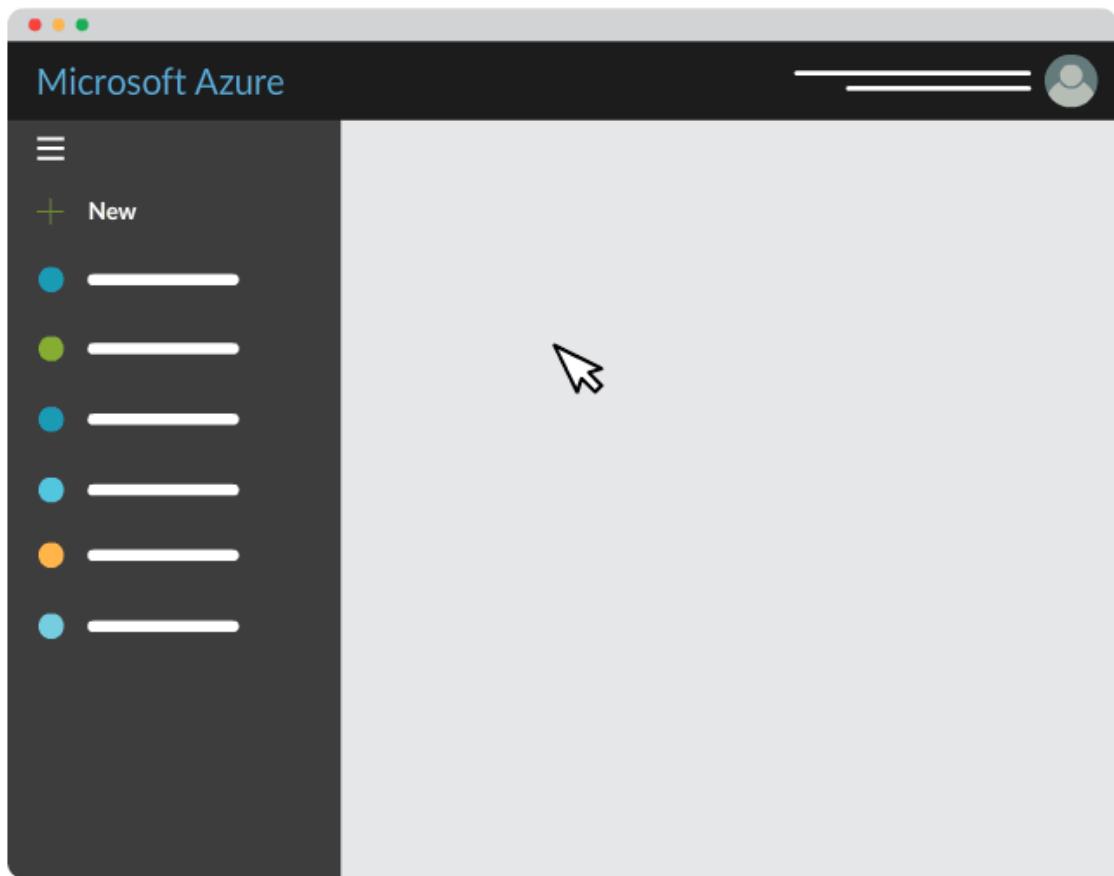
- Windows Server 2008 R2 and later versions are available in **Azure Market Place**.
- Windows Server 2003 and later versions are supported for deployment. However, OS images must be uploaded from On-Premises.

### Linux OS

- Azure supports many Linux distributions and versions including CentOS, Core OS, Debian, Oracle Linux, Red Hat Enterprise Linux, and Ubuntu.

Microsoft does not support OS that is past their End of Support date without a Custom Support Agreement (CSA). For example, Windows Server 2003/2003 R2.

# Creating Virtual Machines



Finally, now to create Azure VMs any of the following tools can be leveraged,

- Azure portal
- ARM Template
- Azure PowerShell
- Azure CLI

## Basic steps for deploying a virtual machine

- Select an image or disk to use for the new virtual machine from **Azure Market Place**.
- Provide **Required information** such as hostname, username, and password for the new virtual machine.

- Provide **Optional information** like domain membership, virtual networks, storage account, cloud service, and availability set.
- Go ahead and provision the machine.

## Azure Virtual Networks

### Azure Virtual Network

Once you create a VM, you will need to place it in a virtual network to receive IP address configurations and to connect to other VMs or other resources that you create in Azure.

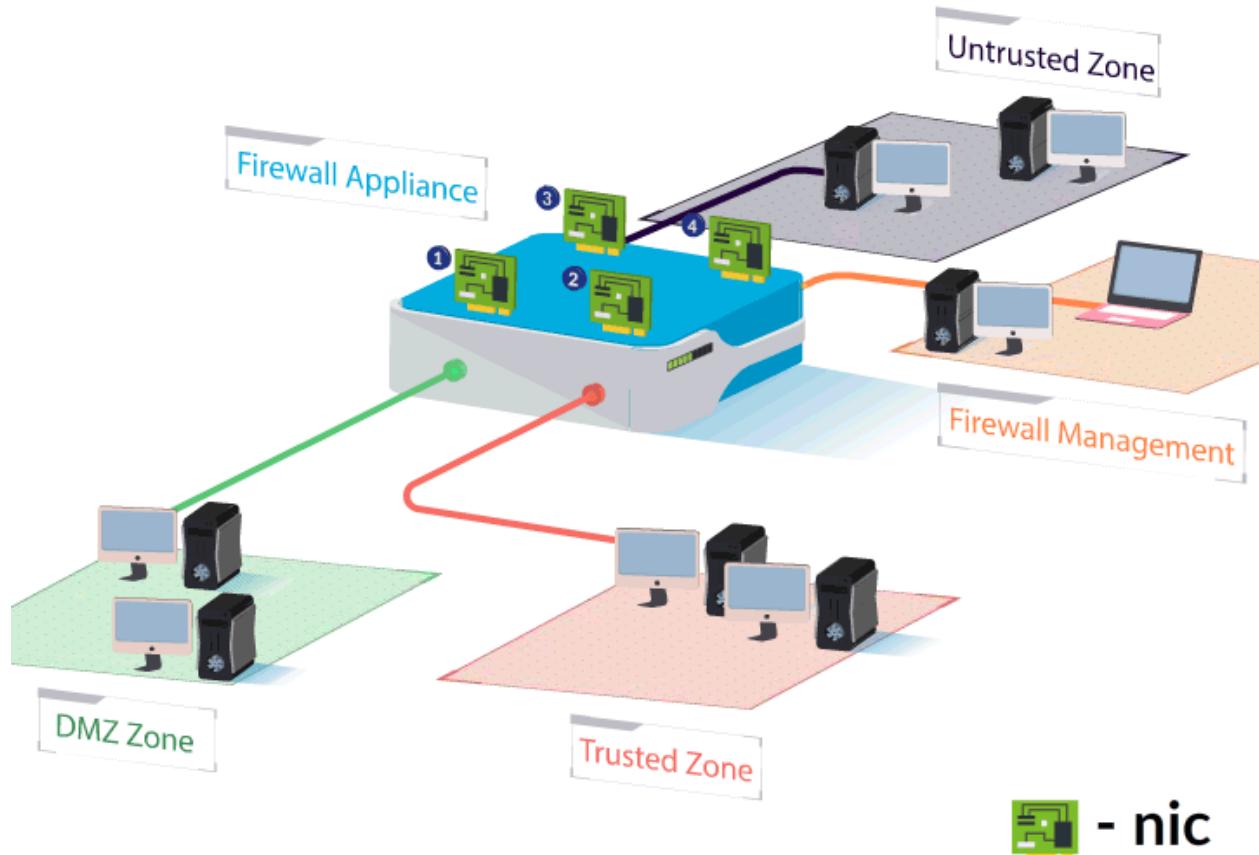
#### In this topic, you learn:

- Vnet Creation
- Multiple Network Interface Card(NIC) usage
- Network Security Group(NSG) and NSG Rules

You can create Azure network resources by using either the **ARM Portal, Classic Portal, Network Configuration File, Azure PowerShell module, Azure command-line interface (Azure CLI)**, or by using deployment templates.

# Multiple NICs in Virtual Machines

## Multiple NIC's



You can attach multiple network interface cards (NICs) to each of your VMs.

Multiple NICs are used for many network virtual appliances and WAN Optimization solutions, as it provides **high network traffic management capability**, including **isolation of traffic** between a front-end NIC and back-end NIC(s).

**Example** - Palo alto Firewall appliance which contains 4 NICs for,

- DMZ
- Trusted Zone
- Untrusted Zone
- Firewall Management

## Limitations of Multiple NICs

- All VMs in a **Availability set need to use either multi-NIC or single NIC**. There cannot be a mixture of multi-NIC VMs and single NIC VMs within an availability set.
- Once deployed, a **VM with single NIC cannot be configured with multi NICs** (and vice-versa), without deleting and re-creating it.

## Network Security Groups



VMs can have connectivity to the Internet when public IP address is assigned to the VMs or to the cloud service. Under such scenarios, **Network Security Group (NSG)** provides advanced security protection for the VMs.

NSGs contain inbound and outbound rules that specify whether the traffic is approved or denied.

*NSGs Rule can be applied at the following levels,*

- **NIC** (ARM deployment model)
- **VM** (classic deployment)
- **All VMs in a Subnet** (both deployment models)

## NSG - Example



Consider a requirement where **Application Server** should communicate with **Internet (User)** using **HTTP Protocol only**.

- To achieve this, **NSG1 - web-rule** can be set to allow **HTTP traffic** to the FrontEnd subnet.

Also, assume a requirement where **DataBase Server should not communicate with internet and it should get SQL traffic only from Application Server**.

For this, NSG2 - DB Rule can be set as

- **sql-rule** to allow **SQL traffic only** from the FrontEnd subnet.
- **web-rule** to deny all **internet bound traffic** from the BackEnd subnet.

## NSG Rule Properties

- **Name** - is a unique identifier for the rule.
- **Direction** - specifies whether the traffic is inbound or outbound.
- **Priority** - If multiple rules match the traffic, rules with higher priority apply.
- **Access** - specifies whether the traffic is allowed or denied.
- **Source IP address prefix** - identifies from where traffic originates.
- **Source port range** - specifies source ports.
- **Destination IP address prefix** - identifies the traffic destination IP range
- **Destination port range** - specifies destination ports
- **Protocol** - specifies a protocol that matches the rule.

## Things to Remember

- By default 100 NSGs can be created per region per subscription. This can be extended to 400 by contacting Azure support.
- A single NSG can have 200 rules which can be raised to 1000 by contacting Azure support
- Only one NSG can be applied to a VM, subnet, or NIC. However, the same NSG can be applied to multiple resources.

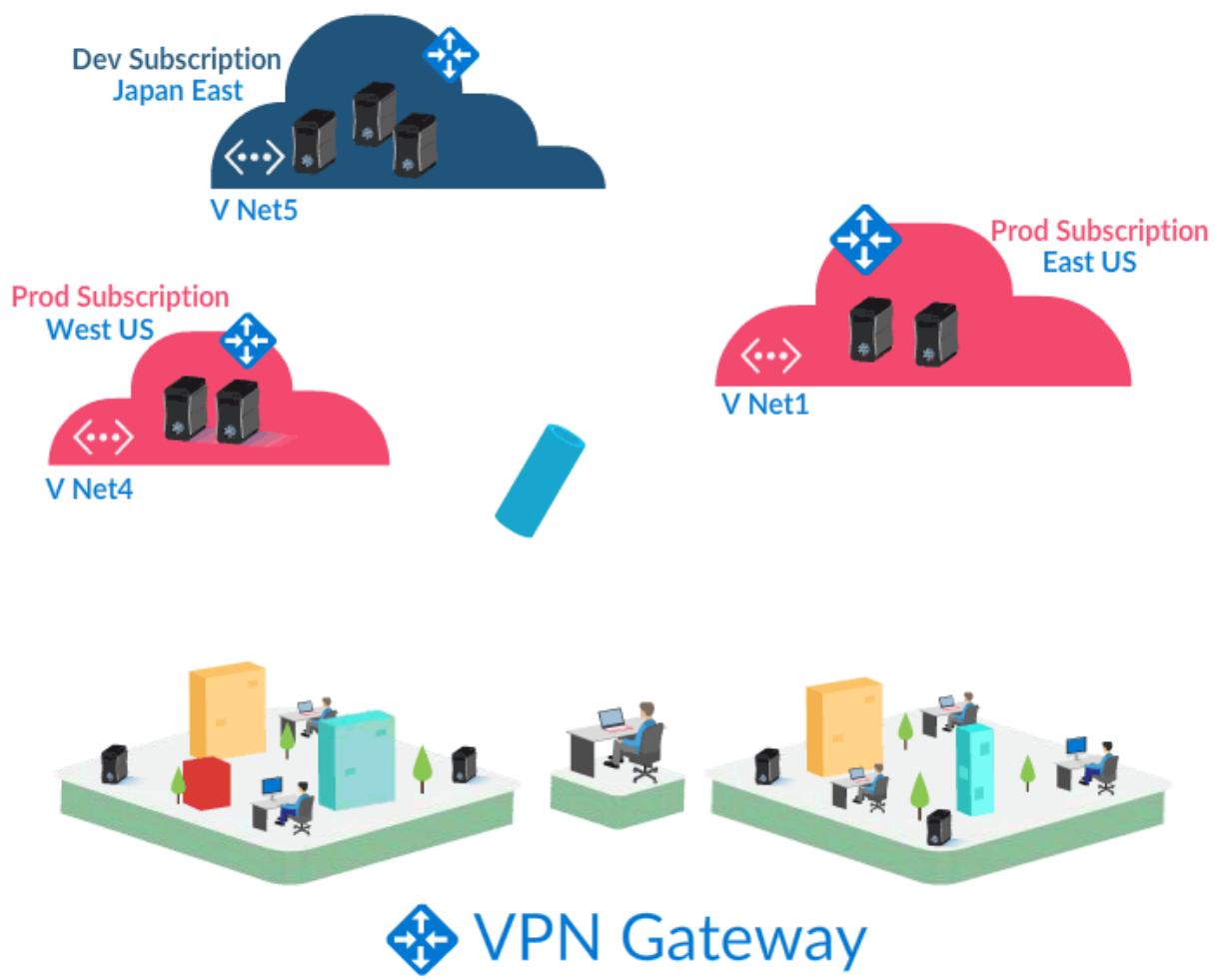
# Intersite Connectivity

## Intersite Connectivity Options

It is very common to come across a scenario where you have an On-Premise data center that needs to connect to resources deployed on Azure. There are four different options to handle such scenarios.

**Point to Site VPN** and **Site to Site VPN** were covered in detail in the **Azure Essential** course, So, now we will concentrate on **Vnet to Vnet Connectivity** and **Express Route**

## VPN Gateway



All 4 types of VPN connections require a **Virtual Network gateway** in the virtual network, which routes traffic to the on-premises computers.

The following VPN connections requires VPN Gateway:

- **Point-to-site**
- **Site-to-site**
- **VNet-to-Vnet** - Between different Azure Regions - Between different Azure Subscription
- **IaaS v1 VNet-to-IaaS v2 VNet**
- **Multisite**
- **ExpressRoute**

## Features of VPN Gateway

A VPN gateway is a type of **virtual network gateway** that sends encrypted traffic across Azure virtual networks and also from Azure virtual network to an on-premises location.

- Each virtual network can have **only one VPN gateway**.
- **Multiple connections** can be made with the same VPN gateway.
- When multi VPNs connect to the same VPN gateway, **all VPN tunnels share the bandwidth that is available for the gateway**.

## Vnet to Vnet Connectivity

VNet-to-VNet connectivity is similar to connecting a VNet to an on-premises site location, except that both ends of the connection are VNets.

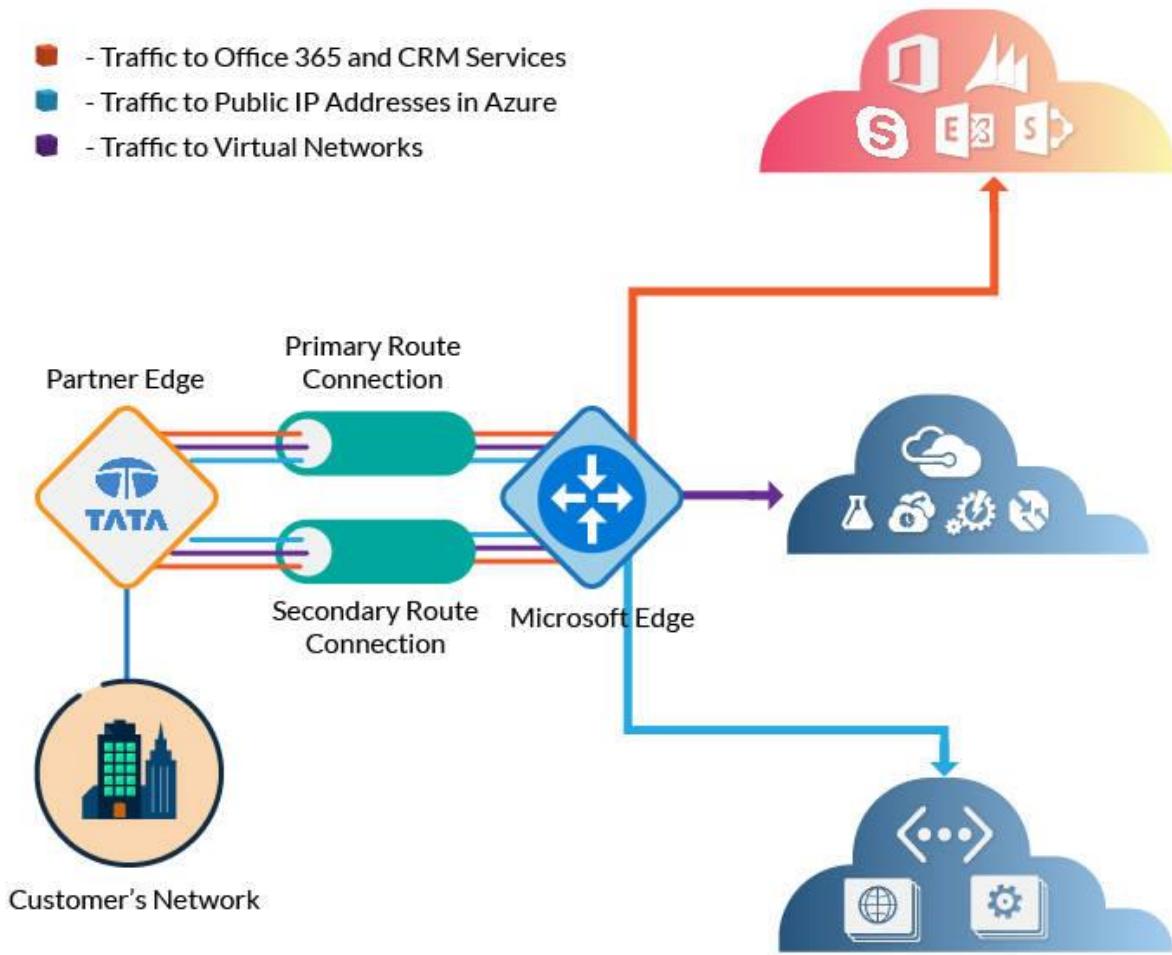
**VMs and cloud service components in each VNet can communicate as if they were on the same VNet.**

In VNet-to-VNet model, the connected VNets can be in the:

- **same or different regions**
- **same or different subscriptions**
- **same or different deployment models**

# Advantages of ExpressRoutes

- Traffic to Office 365 and CRM Services
- Traffic to Public IP Addresses in Azure
- Traffic to Virtual Networks



Typically ExpressRoute is used to establish the connections to Microsoft cloud services, such as Microsoft Azure, Office 365, and CRM Online. Also, it is used to transfer data between on-premises and Azure to achieve cost benefits.

Since they do not go over the public Internet, they are

- More Secure
- Highly Reliable
- Faster and Lower Latency

# Considerations for Intersite Connections

## VPN Tunnel

- Azure supports a maximum of 30 VPN tunnels per VPN gateway.
- Each point-to-site, site-to-site and VNet-to-VNet VPN counts as one of those VPN tunnels.
- Redundant tunnels are not supported.

## VPN Gateway

- A single VPN gateway can support up to 128 connections from client computers.
- All VPN tunnels to a virtual network share the available bandwidth on the Azure VPN gateway.

## Address spaces

- Address spaces must not overlap. Hence must be planned for virtual networks in Azure and on-premises networks.

# Azure Storage

## Azure Storage F5

In **Azure Essential** course, we briefly learned about various storage services, replication methods, and finally about the hybrid cloud storage solution ***StorSimple***.

In this course, you will learn:

- **Performance tiers of Storage Accounts**
- **Features of Premium Storage**
- **Virtual Machine Storage**
- **Azure Files storage**
- **Azure Blob or Unstructured Storage**
- **Storage Tools**

## Getting Started

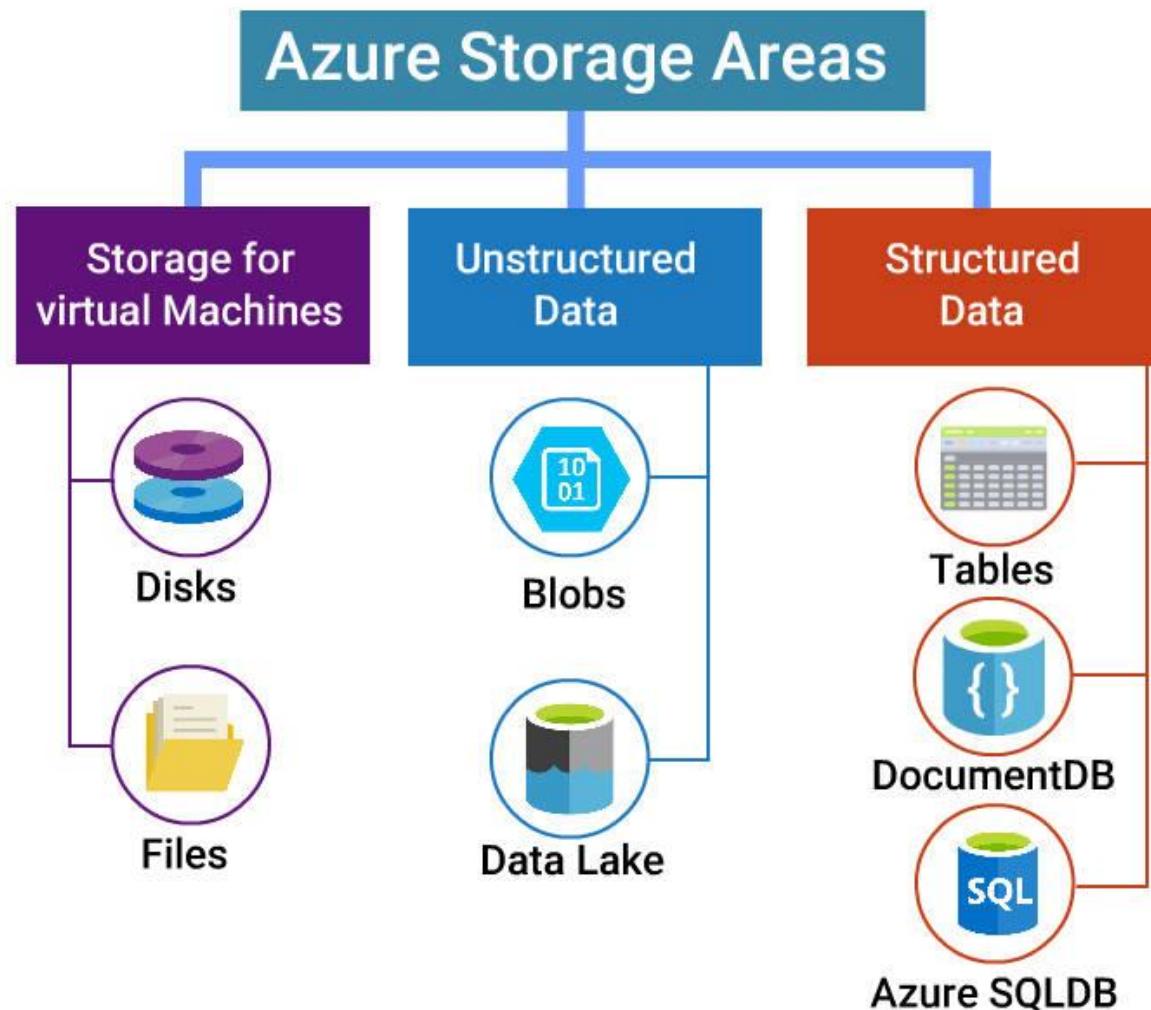
To use any of the Azure Storage services like Blob storage, File storage, and Queue storage, you will first create a storage account,

and then you can transfer data to/from a specific service in that storage account.

Once created, Azure Storage resources can be accessed by any language that can make HTTP/HTTPS requests. Additionally, Azure Storage offers programming libraries for several popular languages to simplify many aspects of working with Azure Storage.

We will learn about all of this in next few cards!

## Azure Storage Areas



**Before we get started with creating Azure Storage, let us understand various Azure Storage Areas and Accounts.**

Azure storage is broadly grouped into 3 categories:

- Storage for Virtual Machines

- Unstructured Data storage
- Structured Data storage

### **Storage for Virtual Machines:**

- **Disks** - Persistent block storage for Azure VMs.
- **Files** - Fully managed file share on the cloud.

### **Unstructured Data storage:**

- **Blobs** - Highly scalable, REST based Cloud Object Storage.
- **Data Lake Store** - Hadoop Distributed File System (HDFS) as a Storage.

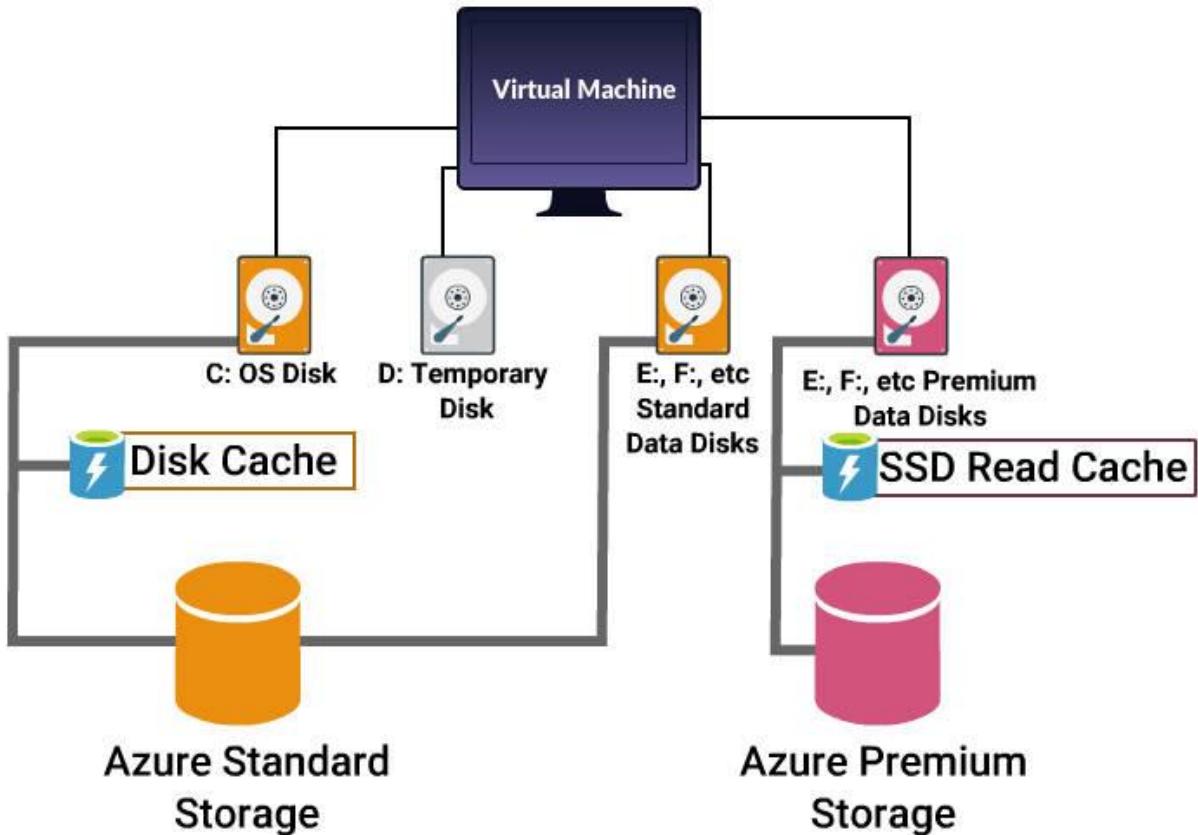
### **Structured Data storage:**

- **Azure SQL DB** - Fully managed Database-as-a-service built on SQL.
- **Tables** - Key Value, high scale, auto-scaling NoSQL store.
- **DocumentDB** - NoSQL document database service.

You will learn more about **Disk and File** storage in next couple of cards. Structured and Unstructured data storages are covered in separate topics.

# VM Disk Storage

## Virtual Machine Storage Architecture



Virtual machines in Azure use disks as a place to store **Operating system, Applications, and Data**.

- VM's also can have one or more data disks
- Standard Storage Account uses **Hard Disk Drive(HDD)** as VM disk.
- Premium Storage Account uses **Solid State Drive(SSD)** as VM disk.
- Temporary Disk is a **Non Persistent storage and uses SSD storage**.
- All disks are stored as **Virtual Hard Disk's (VHD)**, and the maximum capacity of the VHD is limited to **1023 GB**.

It is not recommended to store any data on Temporary Disk.

# VM Files storage

File storage offers shared storage using the standard SMB 3.0 protocol.

- It can be accessed as a mounted drive or Map network Drive as typical SMB share in Desktops.
- On-premises applications can access file data in a share via the **File storage API**.

**Common uses of File storage:**

- Applications that rely on file shares.
- Files like **Configuration files** that need to be accessed from multiple VMs.
- **Diagnostic data** like logs that can be written to a file share and processed later.
- **Tools and utilities** used by multiple developers.

# Storage Accounts

## Types of General Purpose Storage Account

- **Standard storage** - most widely used storage accounts that can be used for all types of data (tables, queues, files, blobs and VM disks).
- **Premium storage** - high-performance storage for page blobs, which are primarily used for VHD files.

## Performance tiers of Blob Storage Account

- **Hot access** - for files that are accessed frequently. You pay a higher cost for storage, but the cost of accessing the files is much lower. Example: **File Share**.
- **Cool access** - to store large amounts of rarely accessed data for lower cost. Example: **Backup Data**.

# Storage Accounts

## Types of General Purpose Storage Account

- **Standard storage** - most widely used storage accounts that can be used for all types of data (tables, queues, files, blobs and VM disks).
- **Premium storage** - high-performance storage for page blobs, which are primarily used for VHD files.

## Performance tiers of Blob Storage Account

- **Hot access** - for files that are accessed frequently. You pay a higher cost for storage, but the cost of accessing the files is much lower. Example: **File Share**.
- **Cool access** - to store large amounts of rarely accessed data for lower cost. Example: **Backup Data**.

## Storage Account Conversion

- Standard storage accounts are backed by magnetic drives (HDD) and provide the lowest cost per GB.
- Premium storage accounts are backed by solid state drives (SSD) and offer consistent low-latency performance.

Hence it is not possible to convert standard storage account to Premium Storage account or vice versa.

## Premium Storage



Microsoft recommends using Premium Storage for all VMs.

Premium storage has **high bandwidth with extremely low latency** and it offers less than 1ms read latency(cache). Also, premium storage disks for virtual machines support up to **64 TB of storage, 80,000 IOPS per VM and 50,000 IOPS per disk.**

- To improve total IOPS throughput we recommend striping across multiple disks and using SSD premium disks.
- Premium Storage is only supported on **Azure GS and DS series** of virtual machines.
- Premium Storage supports only **Locally Redundant Storage (LRS) Replication**.
- In Premium (SSD), the size of the VM disk is restricted to **128, 512, and 1023 GB.**

## Storage Access Tools

Azure Storage tools, make the life of a storage administrator much easier. Here are few of the most commonly used tools.

- **Azure Portal** and **Azure PowerShell**
- **Azure Storage Explorer** - a useful GUI tool to inspect and alter data in Azure Storage. It can be used to upload, download, and manage blobs, files, queues, and tables from any platform, anywhere.
- **AZ Copy** - a command-line utility to copy blob, and file data within a storage account or across accounts.
- **Azure Import/Export service** - to import or export large amounts of blob data to or from a storage account.

In this course we will learn about Azure Storage Explorer.

## Securing Storage

Azure Storage provides a comprehensive set of security capabilities which together enable developers to build secure applications.

## Data Security:

- **Data in transit** can be secured using **client-side encryption, HTTPS or SMB 3.0 protocol.**
- **Data at rest** can be secured using **Storage Service Encryption.**
- **OS and Data disks** used by VMs can be encrypted using **Azure Disk Encryption.**

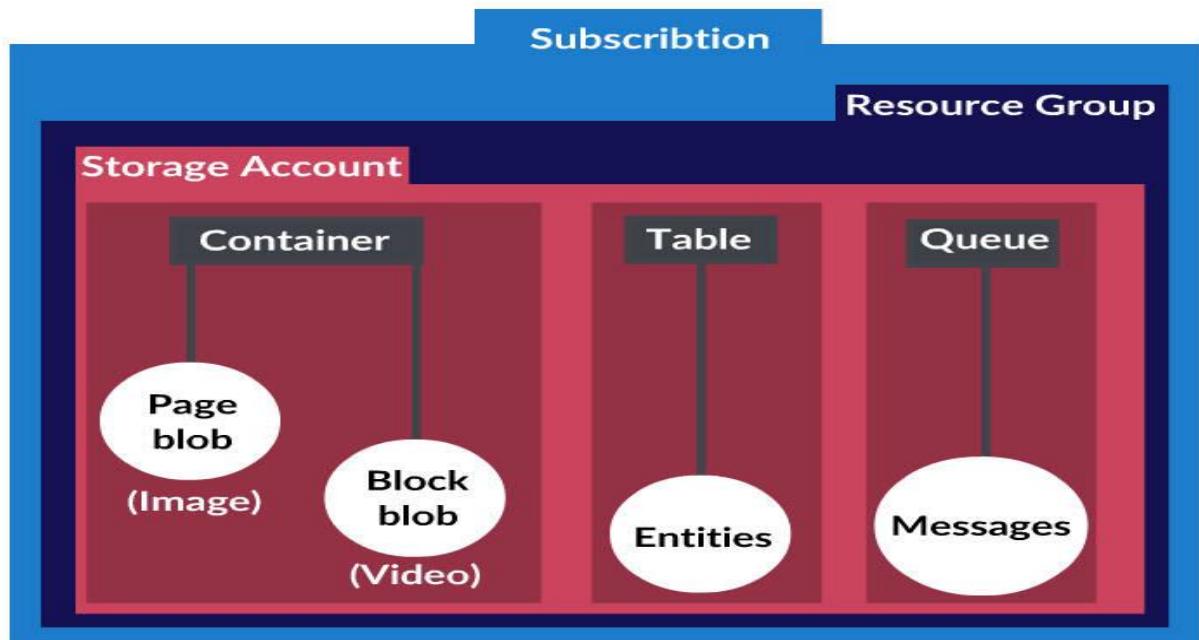
## Security Management:

- **Storage Access Policy**: define policies to grant and revoke access at a granular level, with a time limit.
- **Role-Based access control**: use default and custom defined roles to control access to the storage accounts
- **Audit and monitor authorization**: using request information available in storage analytics logs.

Storage Account Access keys and Shared Access Signatures (SAS) can be used to secure data access.

# Azure Blob Storage

## Blob Storage Overview



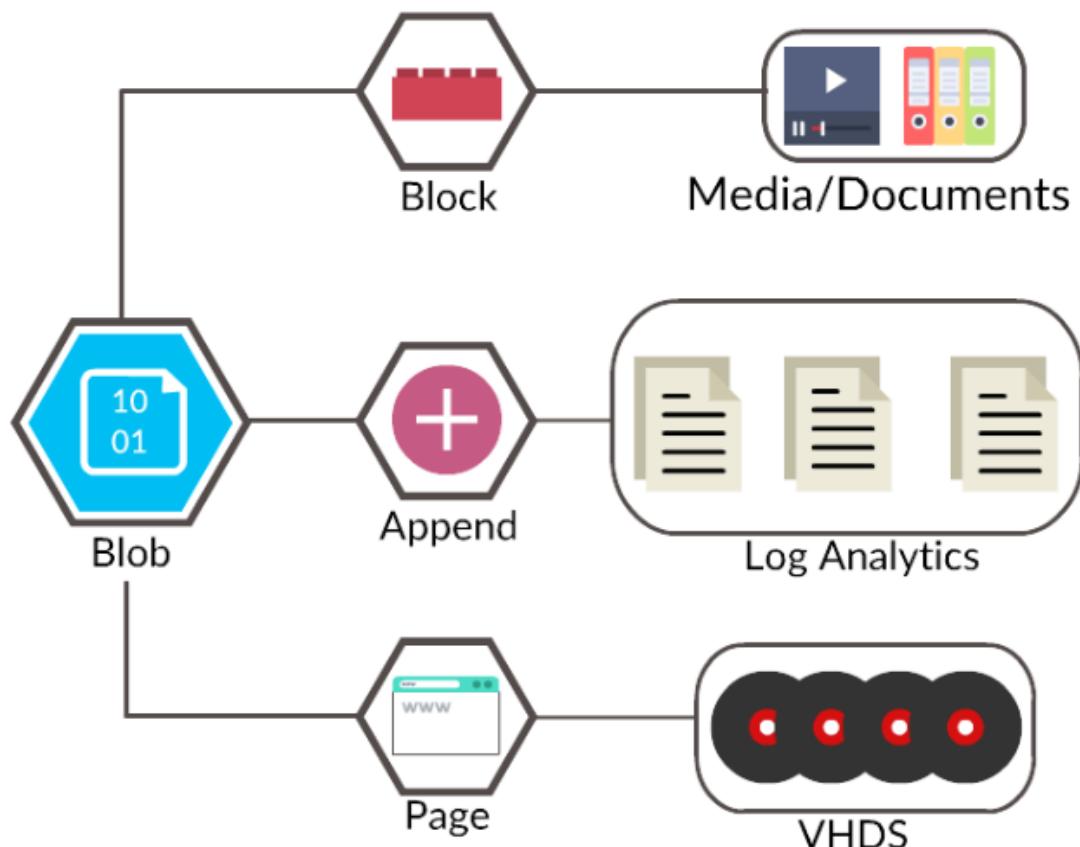
**Azure Blob storage is a service that stores Unstructured data in the cloud as blobs/objects.** All blobs must be in a container and a container can store an unlimited number of blobs.

### *Common uses of Blob storage:*

- Serving images or documents directly to a browser.
- Storing files for distributed access.
- Streaming videos and audios.
- Storing data for backup & restore, disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

Blob storage is also referred to as **Object storage**.

## Types of Blobs



Azure Storage offers three types of blobs: **Block blobs, Page blobs, and Append blobs.**

## Block blobs

- Suitable for **Sequential Read\Write** operations.
- Ideal for storing text or binary files, such as **documents and media files.**

## Append blobs

- Optimized for **Append** operations.
- Can be used for **logging scenarios.**

## Page blobs

- Optimized for **Random read/write** operations.
- Can be used for storing **VHD** files of Azure VM as **OS and Data disks.**

# Managing Blob Storage

Blob storage typically requires transacting **Huge amounts of data** from On-Premises to Azure and vice-versa.

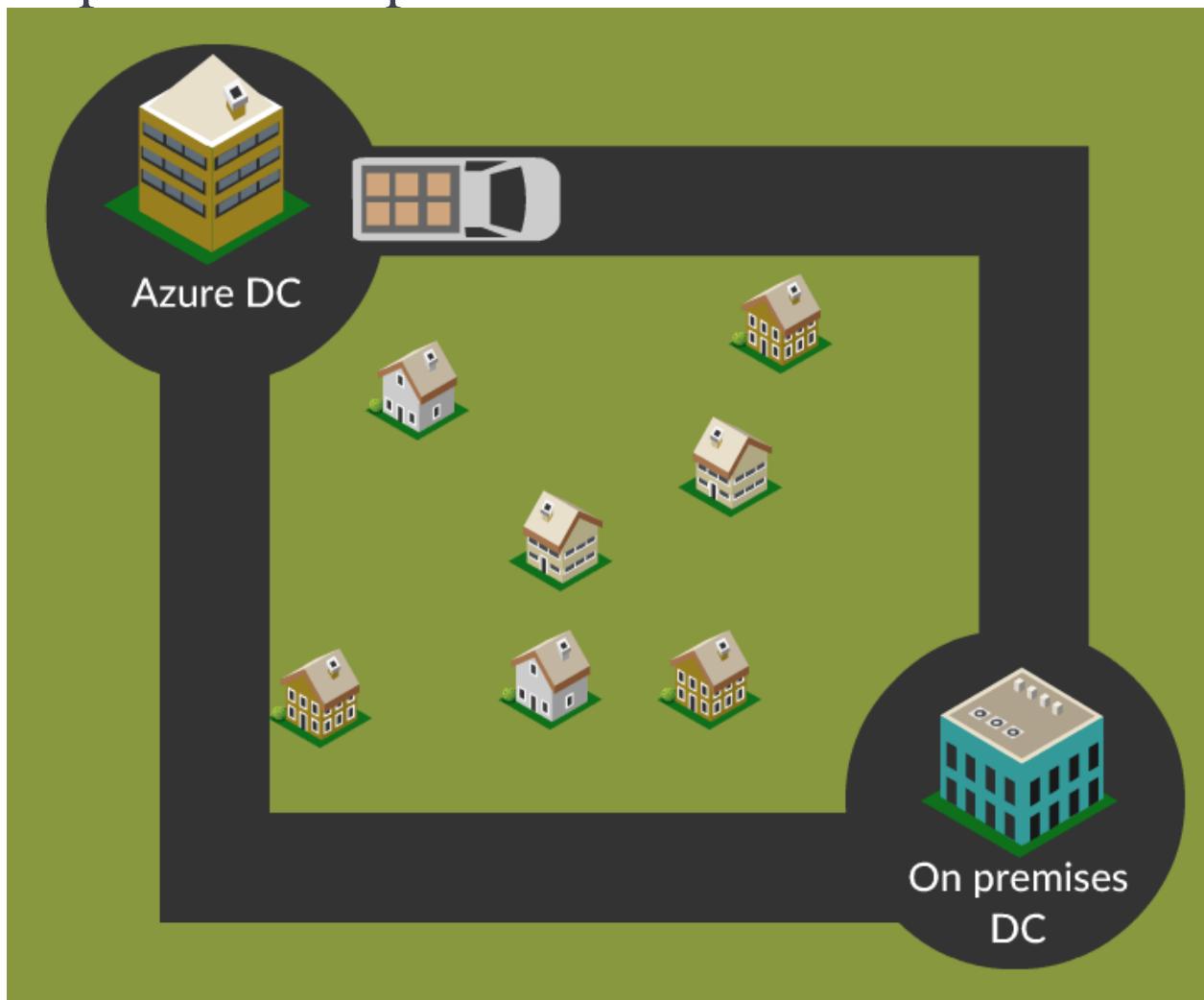
### Examples of Huge Data:

- **Large Virtual Hard Disks (VHDs)** - Using Upload and Download Commands
- **TBs of Backup Data** - Using Export and Import services

Generally uploading and downloading VHD files is done through **Azure PowerShell or Storage Explorer**. Azure PowerShell provides a very efficient way for moving these large files through following cmdlets:

- **Add-AzureRmVHD** - Uploads a VHD from an on-premises virtual machine to a blob storage in Azure.
- **Save-AzureRmVHD** - Saves downloaded VHD images locally.

# Import and Export Service



Import and Export is another service that is suitable for scenarios when several TBs of data needs to be transferred. Transferring such volume of data over the network is not feasible due to limited bandwidth and high network costs.

**Example:** Backup data to or from Azure.

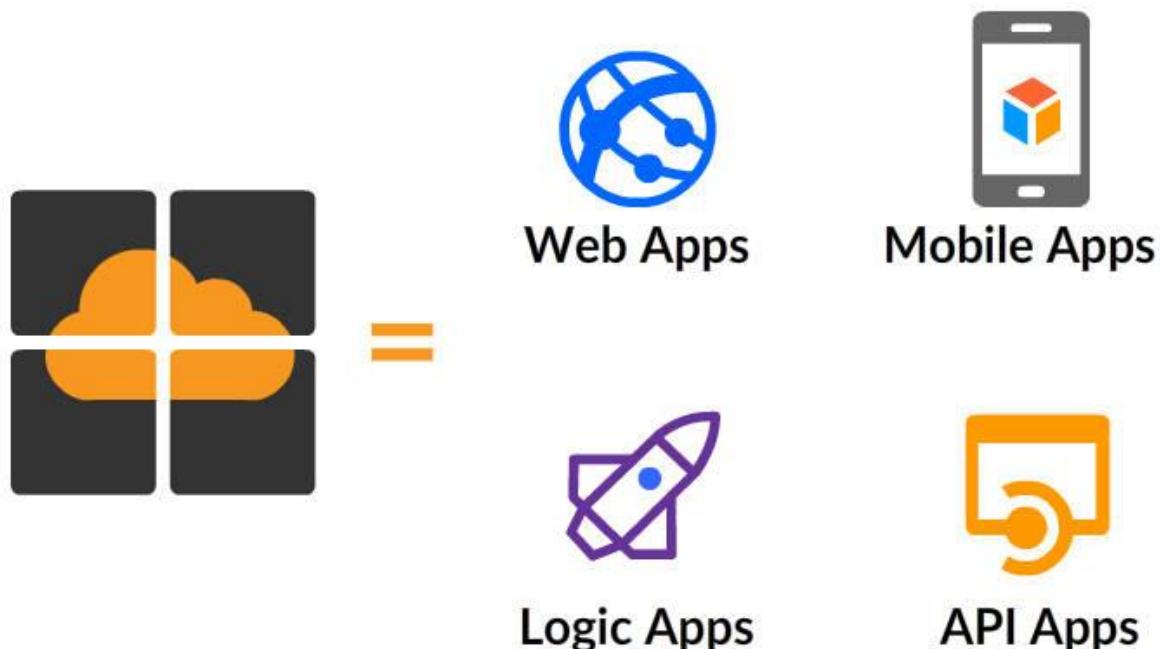
- **Import Service** - Securely transfers large amounts of data to Azure blob storage by shipping hard disk drives to an Azure DC.
- **Export Service** - Transfers data from Azure blob storage to hard disk drives and ship to the on-premises site.

### *Usage Scenarios:*

- Migrating data to the cloud.
- Content distribution to a different Datacenter.
- Backup and Recovery data.

## Azure App service

### Azure App Services - F5



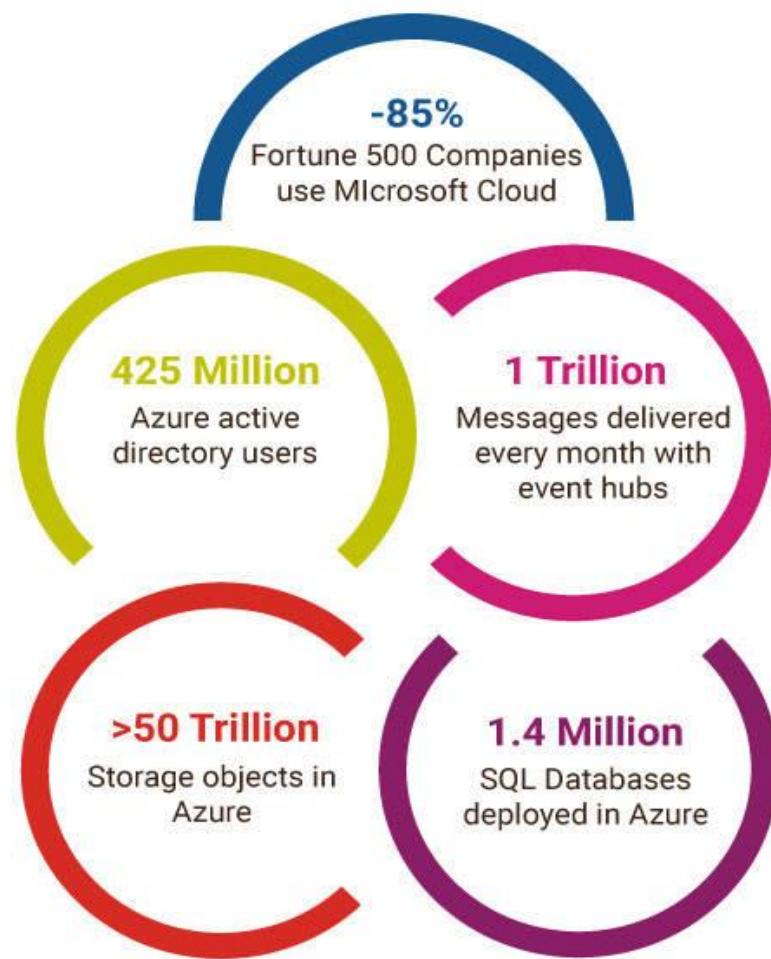
In **Azure Essentials** we briefly learned how Azure App Service enables us to easily create, Web, Mobile, Logic and API Apps.

- **Web apps:** web based applications that can scale with business requirements
- **Mobile Apps:** mobile applications that can run on any device

- **Logic apps**: For automating business processes and integrating systems and data across clouds without writing code.
- **API apps**: For hosting RESTful APIs that other services can leverage, such as in IoT scenarios

In this topic, we learn about Azure App Service Features, Plan and Environment.

## Azure Momentum



Global markets are now expecting:

- Deeper engagement with customers
- Faster times to market
- Scalability
- Availability
- Lower Costs

Azure App Services is gaining popularity and growing rapidly, as it helps teams meet these expectations. It offers flexibility, supports open source technologies and multiple languages for you to build your applications.

## App Deployment Guide

Responsibility	On-Prem	IaaS	PaaS	SaaS
Data classification & accountability	Cloud Customer	Cloud Customer	Cloud Customer	Cloud Customer
Client & end-point protection	Cloud Customer	Cloud Customer	Cloud Customer	Cloud Customer / Cloud Provider
Identity & access management	Cloud Customer	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Customer / Cloud Provider
Application level controls	Cloud Customer	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Provider
Network controls	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Provider	Cloud Provider
Host infrastructure	Cloud Customer	Cloud Customer / Cloud Provider	Cloud Provider	Cloud Provider
Physical security	Cloud Customer	Cloud Provider	Cloud Provider	Cloud Provider

■ Cloud Customer ■ Cloud Provider

Areas of responsibility and control that needs to be retained can guide in deciding if the application should be:

- On-premises
- On Azure virtual machines as part of an IaaS offering
- Part of a PaaS offering
- just consuming the end product in a SaaS offering

# Managing Azure Apps Services

## Management Tools

Like other Azure services, Azure App Service can also be managed by using the following tools,

- Azure PowerShell
- Azure Command Line Interface (CLI)
- REST APIs
- Templates
- ARM and ASM portals

In this topic, we will learn about **Locking Resources, Custom Domain name configuration, Site Extensions, and app deployment options.**

## Locking Resources

Azure App Services protects the resources using **Locks**.

- Locks can be applied to a **subscription, resource group, or a resource such as a web application.**
- It prevents **deleting or modification of resources** by other users.

### Lock Levels:

- **CannotDelete:** where authorized users can read and modify a resource, but they can't delete.
- **ReadOnly:** where authorized users can read from a resource, but they can not perform any actions on it.

Locks are different from Role Based Access control(RBAC).

## Custom Domain Name Configuration

When a Application is hosted on Azure, it can be accessed using a default domain name **<app name>.azurewebsites.net**.

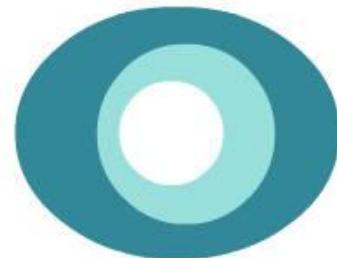
However, **it is preferred to access the application using our own URL**, such as <https://play.fresco.me>

Custom Domain name can be directly purchased through the **Azure App services portal** or one can carry forward their own Domain Name.

## Site Extensions with Apps



Application Insight



Newrelic



Jekyll



PHP Manager

**Site Extensions are used to extend functionality and provides ease management of Web Applications.**

- A full list of available site extensions is available on the Site Exnsions page, <https://www.siteextensions.net/>.
- It also provides pointers to project sites, licenses, owner details etc for each extension.

**Examples of Site Extensions:**

- **Application insights:** provides monitoring capabilities.

- **New Relic**: provides monitoring capabilities.
- **Php Manager**: tool for managing PHP installations.
- **Jekyll**: Adds support for Jekyll on a Web App.

## App Service Deployment Options

There are a number of different options available for deployment of web app services such as,

### Basic

- *FTP*
- *Web Deploy*

### Alternative

- *OneDrive/DropBox*
- *Kudu*

### Source Control / Continuous Deployment

- *Visual Studio Online*
- *Local Git*
- *GitHub*
- *BitBucket*

## Kudu based Deployments

Kudu is the engine behind source control based deployments into Azure App Service. Every Azure website has an associated 'scm' service site, which runs both Kudu and other Site Extensions.

### Accessing the Kudu service

If your website URL is <http://mysite.azurewebsites.net/>, the root URL of the Kudu service is <https://mysite.scm.azurewebsites.net>.

## Azure Data Services and Azure SQL

### Azure Database Services Intro

In **Azure Essentials** we learn about different types of Data services in Azure, such as,

- SQL Database
- SQL Data Warehouse
- Document DB
- Table Storage
- Redis Cache
- Data Factory
- Data Lake

In this course, we will focus on:

- **Difference between Azure SQL and SQL on Azure VM**
- **Architecture of Azure SQL and Service tiers**
- **Planning and Provisioning of Azure SQL**
- **Migrating SQL database to Azure**
- **HDInsight**

## Database On Cloud

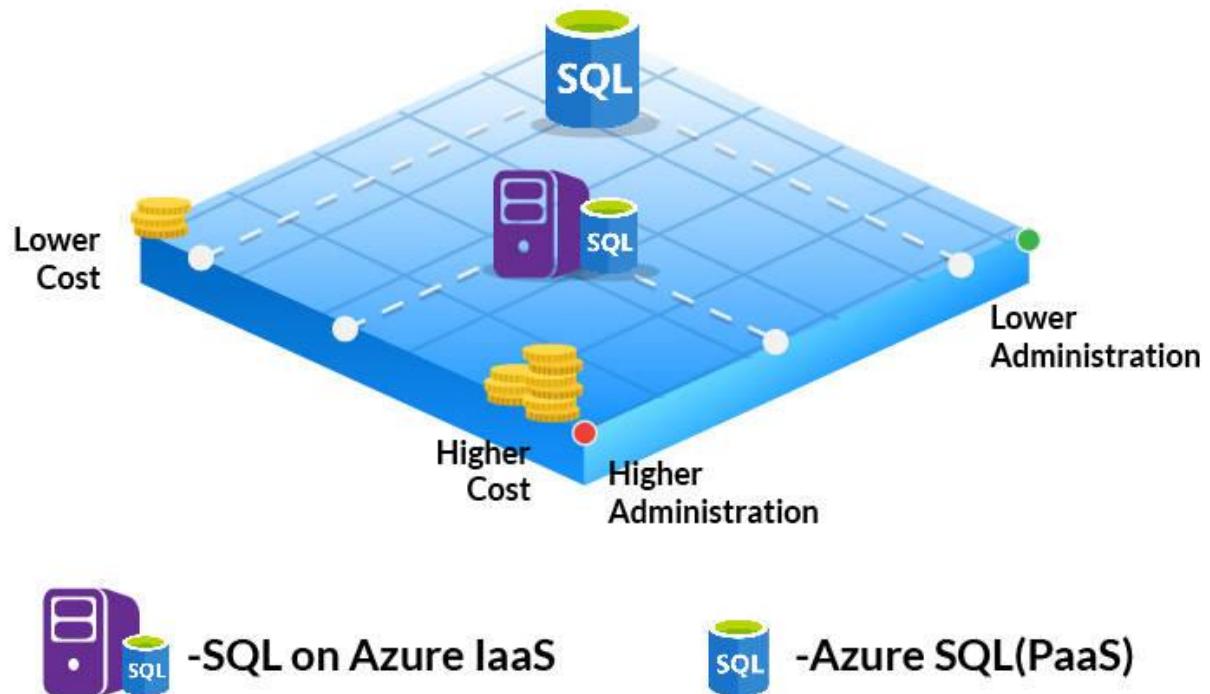
**One of the key decision points about where and how you want to run your SQL databases is the *administrative Vs Cost overheads*.**

There may be critical business requirements around data retention that must be adhered to in some scenarios. Sametime, there are instances where data is potentially suitable to be stored in the cloud either under a PaaS or an IaaS model.

Scenarios like these where there is a blend of answers, opting for a hybrid solution with some of the database needs to be met by using the cloud and other data being retained on-premises can help reduce management overhead and costs.

# Azure SQL (PaaS) Vs SQL Server (IaaS)

SQL in Azure IaaS V.M    Vs    Azure SQL (PaaS)

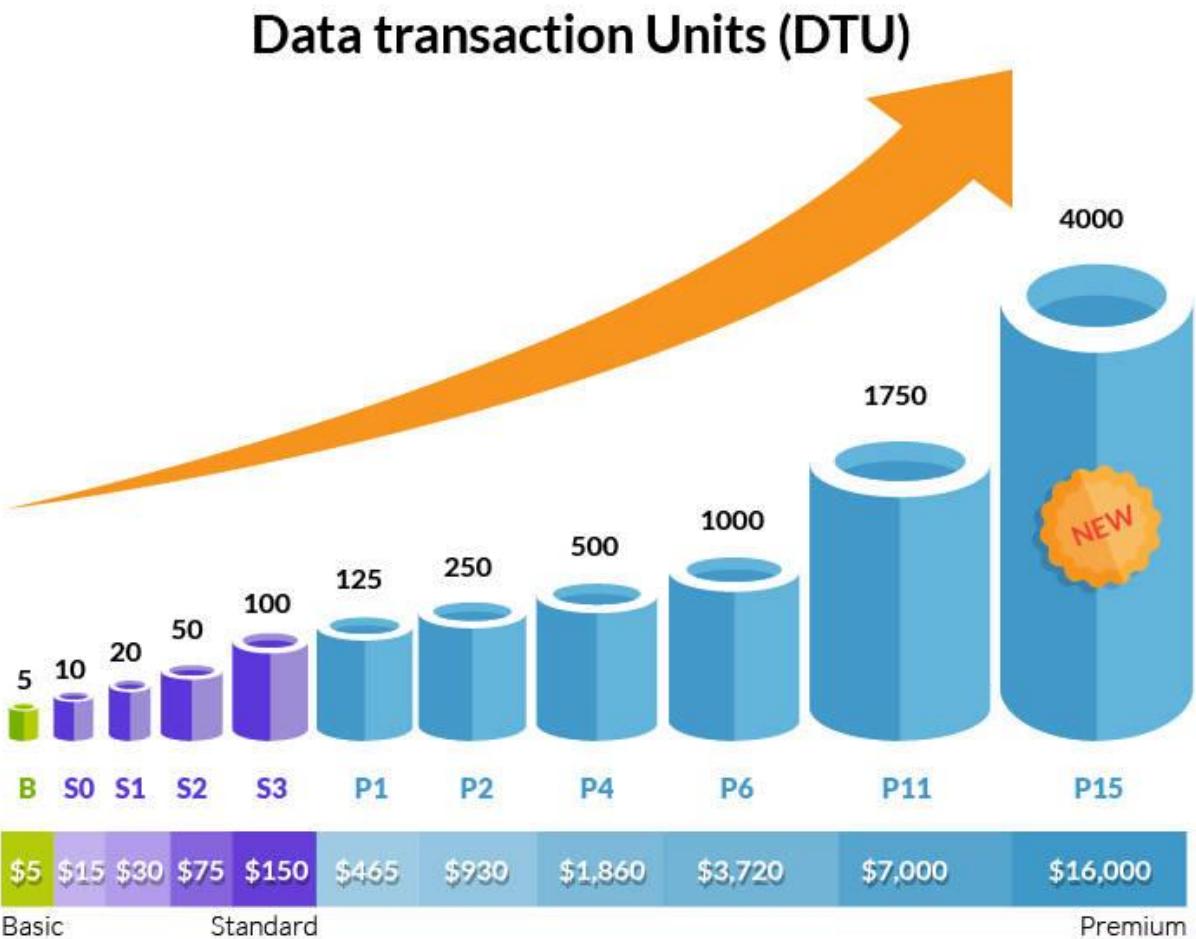


**Azure SQL Database (PaaS)**: is native to the cloud and is optimized for SaaS app development. It reduces overall costs to the minimum for provisioning and managing many databases as you do not have to manage any VMs, OS or DB software.

**SQL Server on Azure Virtual Machines (IaaS)**: is optimized for migrating existing applications to Azure or in hybrid deployments. It is a perfect choice when an organization already has IT resources available to maintain the VMs.

In general, these two SQL options are optimized for different purposes and needs to be determined based on the requirement.

# Database Transaction Unit



A DTU is a measure of the resources that are guaranteed to be available to a standalone Azure SQL database. **It is a measure that combines CPU, memory and I/O values.**

DTU is used to decide which Service Tier is suitable for your Database requirement.

**Larger the number, better the performance.** DTU provides a way to see the overall performance levels, need driving it and be able to relate that to cost.

An elastic DTU (eDTU) is a measure of the resources across a set of databases, called an elastic pool.

# DB Service Tiers

There are three different Service Tiers to accommodate a variety of workload requirements.

- **Basic** - Suitable for **small databases, and low volume needs**
- **Standard** - Suitable for most **cloud based apps**
- **Premium** - Suitable for **high transnational volumes with super critical workloads**

All these options provide an uptime SLA of 99.99% and hourly billing. Also, it is possible to change service tiers and performance levels dynamically.

## Migrating SQL DB

Quite often SQL DBs are migrated from On-Premises to Azure SQL.

### Pre-requisites of Migration

- Test for compatibility of the DB with Azure SQL
- Fix Compatability issues if found
- Perform migration

*Migration can be done in multiple ways, Acceptable Downtime decides the type of Migration.*

- For minimal downtime, use **SQL Server transactional replication** and replicate your data over the Network.
- When downtime is acceptable, use **Export to DAC package and ImportDAC package in Azure SQL**

## Data Analytics with HDInsight

**HDInsight** is a Microsoft managed Hadoop service running in Azure that provides a range of open source data analytics cluster models.

Microsoft makes the Hadoop components available in a distributed model in Azure where it manages the cluster. Making it easy to provision and manage them with high availability and reliability. It is available as a service under the Intelligence and Analytics grouping of services within Azure.

## HDInsight - Cluster Types

HDInsight currently provides several different optimized open source analytic cluster types:

- **Hadoop** - Petabyte scale processing with Hadoop components like Hive, Pig, Sqoop.
- **HBase** - Fast and scalable NoSQL Offering.
- **Storm** - Allows processing of infinite streams of data in real-time.
- **Spark** - Fast data analytics and cluster using in-memory processing.
- **Interactive Hive** (preview) - Enterprise Data Warehouse with in-memory analytics using Hive and Long Live and Process (LLAP)
- **R Server** - Terabyte scale, provides enterprise-grade R analytics used for machine learning models.
- **Kafka** (preview): High throughput, low latency, real-time streaming platform, typically used in streaming and IoT scenarios

## Azure Essentials Continuum - Course Summary

Congratulations! You have come to the end of this course. By now, you must have got a good idea about IaaS and PaaS services offered by Azure and various management tools,

- Compute
- High Availability and Scale set
- Virtual Networks
- Intersite Connectivity

- Storage Areas
- Azure PaaS
- Azure SQL
- Few other important services

All the concepts will be understood only if we do the hands-on „„  
Try out all the Hands-On in next topic.

We will go deep dive on these services in a separate courses. Keep Learning!

# JIRA

Introduction to JIRA

Installation and Configuration

Projects

Issues in JIRA

Workflows

Boards and Roadmaps

Search Features

Personalization and Reporting

Test Management

## Introduction to JIRA

Prelude

*Welcome to the course on JIRA!*

***Have you heard about JIRA before? Are you curious to know what JIRA is all about?***

Bingo! You are at the right place.

- This course will teach you the **fundamental features, usage, and advantages of JIRA**.

- It will also guide you on how to utilize the tool for **tracking and reporting bugs in various applications**.

## Significance of JIRA

*Nowadays many IT companies are adopting agile project methodologies to deliver custom software applications within less time. Agile project methodologies emphasize continuous planning, testing, and integration.*

*In an agile environment, QA specialists need to perform testing frequently and consistently. Hence specialized tools and frameworks are required to evaluate the software efficiently in an agile environment.*

- *The tool that was developed by considering the requirements of agile methodologies is Atlassian JIRA.*
- *In addition to the software testing, JIRA is also used as an issue tracking and project management tool.*

## Topics to Expect

You will be learning the following topics in this course.

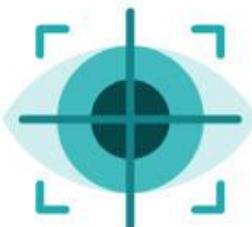
- **Introduction**
- **Installation and Configuration**
- **Projects**
- **Issues**
- **Workflows**
- **Roadmap and Boards**
- **Search Features**
- **Personalization and Reporting**
- **Test Management**

# What is JIRA?

JIRA is a popular **project management** and a **software testing tool**(User-acceptance testing and system testing) used by several testing, development, and technical support teams to complete a great deal of work quicker. **Atlassian Inc.**, an Australian company, developed JIRA.

- JIRA is a **commercial, multilingual** and a **platform-independent tool**.
- It supports **Oracle, SQL, MySQL** and **PostgreSQL** servers in the backend.
- It is integrated with **Hipchat, Bitbucket, Bamboo, Zephyr** and other developing and testing tools.

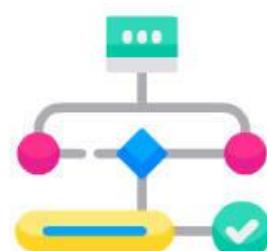
## Uses of JIRA



Tracking issues



Support



Workflow



Project management

## JIRA is used in

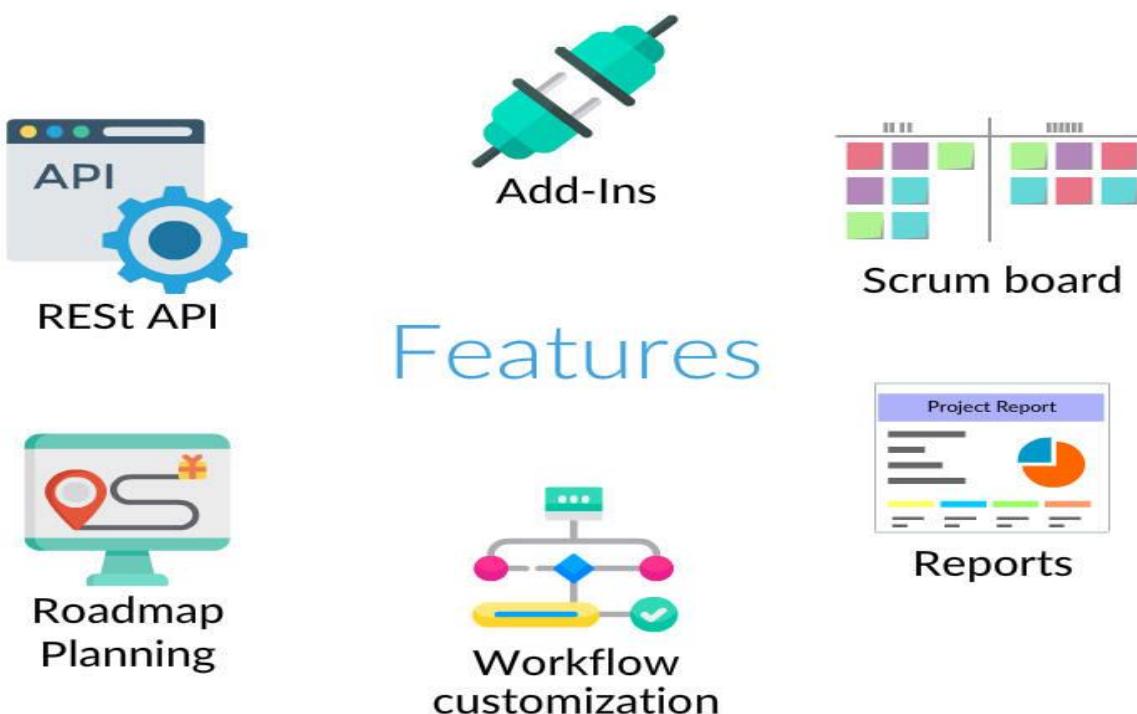
- Tracking issues, bugs, and change requests.
- Support, help desk, and customer services for creating tickets, tracking the resolution and status of the tickets.
- Task tracking, project management, and requirement management.
- Workflow and process management.

## Who Uses JIRA in a Project?

*JIRA is used by many. Following are the professionals who use JIRA.*

- **Scrum Masters** manage their projects and understand their team capacity.
- **Developers** log issues, resolve issues and create stories.
- **Testers** log issues and verify the issue resolutions.
- **Program managers** understand how effectively the issues are resolving by looking at the reports.
- **Product users** log bugs and feature requests.

## Features of JIRA



**The core features of JIRA are mentioned below.**

### **Boards**

- Boards provide an immediate snapshot of the project to the team. JIRA provides Scrum and Kanban boards.

### **Power Search**

- JIRA provides a powerful search functionality with basic quick and advanced features.
- JIRA supports the filter (saved search) options using JQL.

### **Reports**

- JIRA supports many reports to track progress specific timeframe, deadlines, individual's contribution, etc.

## **Features of JIRA**

### **Add-Ins**

- More than 100 add-ins are supported in JIRA to connect with different software to make work easy.

### **REST API**

- Interact with the JIRA Server applications remotely.

### **Workflow Customization**

- Provides an option to modify the workflow according to the project requirement.

### **Roadmap Planning**

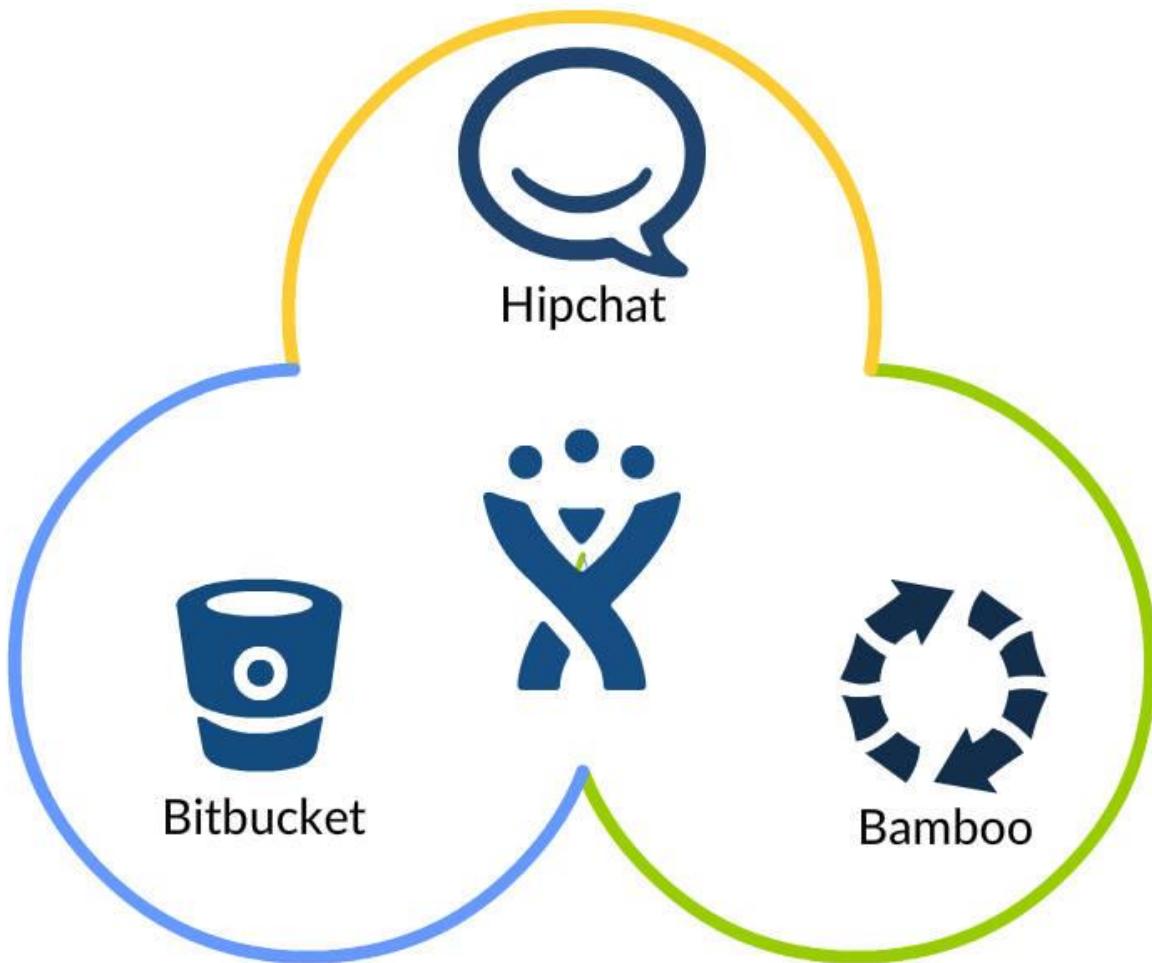
- Enables the team to set a vision for the product and deliver it on time to the customer.

## **Advantages of JIRA**

- Better visibility
- Better prioritization

- Increased productivity
- Stay connected on the go
- Extensible
- Usability

## Tool Integrations



Integrating JIRA with compatible tools helps your team with a wide range of information and functionality related to your work.

**Following are some of the tools integrated with JIRA.**

### Hipchat

- It is a chatting platform that allows you to discuss the project instantly, be notified of issue updates and can monitor from anywhere.

## Bamboo

- It seamlessly views and monitors the status of bamboo builds and deployment without leaving JIRA application projects.

## Bitbucket

- It is a version control repository hosting service that is web-based and used for code review. Bitbucket provides an unlimited number of private repositories.

# Installation and Configuration

## JIRA System Requirements

The following are the **system requirements** needed before installing **JIRA**.

- **Browsers** - Microsoft Internet Explorer (Windows), Microsoft Edge, Mozilla Firefox (all platforms), Google Chrome (Windows and Mac), Safari (Mac), and Mobile Safari (iOS)
- **Java Platforms** - Oracle JDK/JRE 1.8
- **Operating System** - Microsoft Windows and Linux
- **Application Server**
- **Databases** - Oracle, MySQL, PostgreSQL, Microsoft SQL Server, HyperSQL (inbuilt with JIRA)

## JIRA Installation on Ubuntu

1. Download the JIRA Linux 64-bit installer on your system by executing the below command.

```
sudo wget https://www.atlassian.com/software/jira/downloads/binary/atlassian-jira-6.4.10-x64.bin
```

2. Run the following commands to give the complete permission to the binary script and then execute it within its downloaded directory.

```
sudo chmod +x atlassian-jira-6.4.10-x64.bin  
sudo ./atlassian-jira-6.4.10-x64.bin
```

## JIRA Installation

*This step will install JIRA 6.4.10 on your computer.*

**OK [o, Enter], Cancel [c]**

**o**

Choose the appropriate installation or upgrade option.

*Choose one of the following:*

**Express Install (use default settings) [1], Custom Install (recommended for advanced users) [2, Enter], Upgrade an existing JIRA installation [3]**

**2**

**Where should JIRA 6.4.10 be installed?**

[/opt/atlassian/jira]

Default location for JIRA data [/var/atlassian/application-data/jira]

## JIRA Installation

Configure the ports that JIRA will use.

JIRA needs two TCP ports that are not utilized by any other applications on this system. You can access JIRA in the HTTP port through a browser. Then, the Control port is utilized for starting and shutting down JIRA.

**Utilize default ports (HTTP: 8080, Control: 8005) - Recommended [1, Enter], Set custom value for HTTP and Control ports [2]**

**2**

HTTP Port Number [8080]

Control Port Number [8005]

Note that JIRA can function in the background. JIRA can start automatically whenever the machine restarts, i.e., JIRA can run as a service.

Install JIRA as a Service?

**Yes [y, Enter], No [n]**

y

## JIRA Web Setup

- Start JIRA by executing the following command.

```
/etc/init.d/jira start
```

- JIRA can be accessed by visiting the below link.

```
http://<server_IP_address>:<Port_number>
```

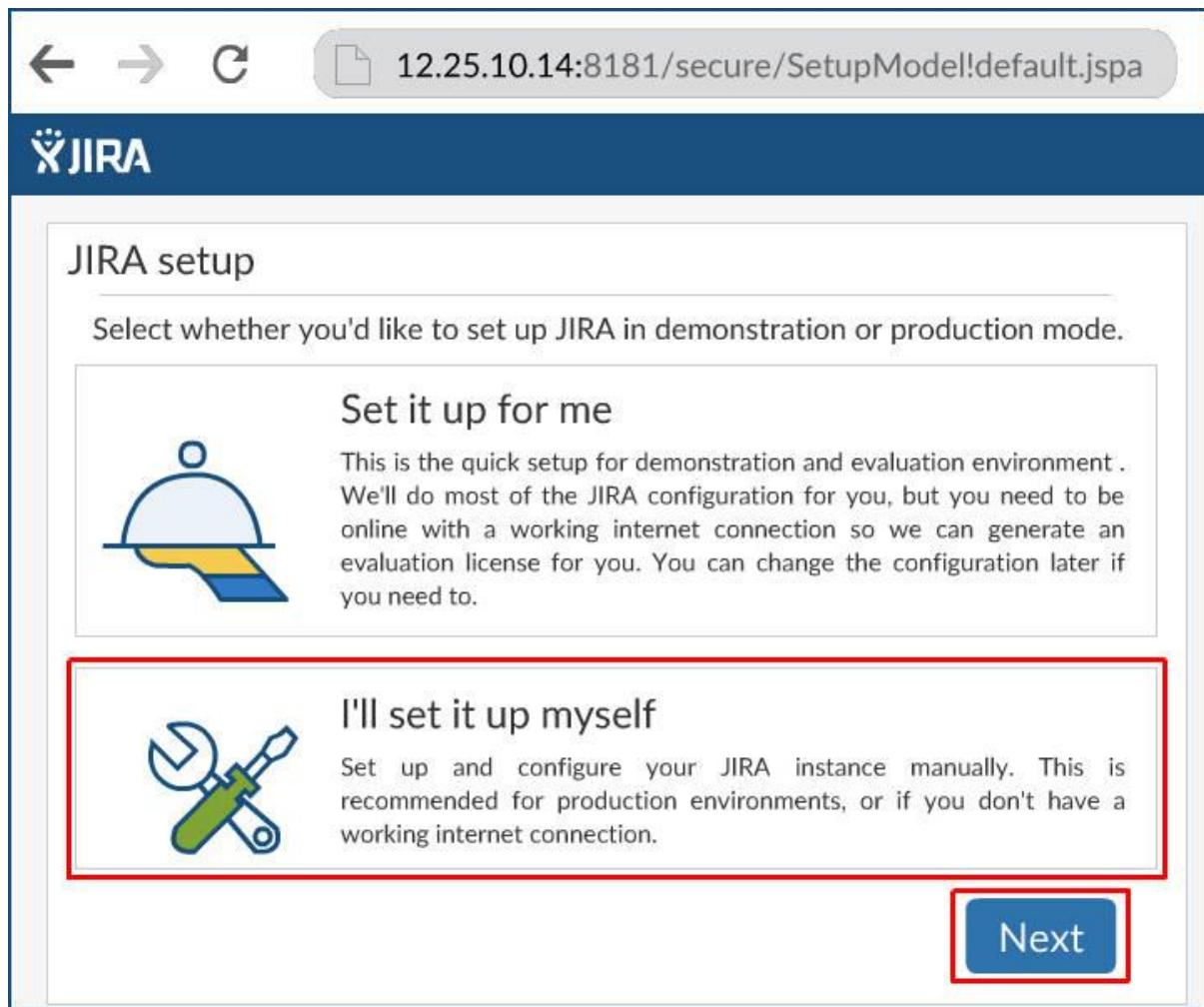
**server\_IP\_address:** IP address of the computer on which the JIRA is installed

**Port\_number:** HTTP port number specified during the JIRA installation

**For Example:**

```
http://127.0.0.1:8181
```

# Running the Setup Wizard in JIRA



The screenshot shows the JIRA setup wizard interface. At the top, there are navigation icons (back, forward, search) and a URL bar displaying "12.25.10.14:8181/secure/SetupModel!default.jspa". The main title is "JIRA setup". A sub-instruction says "Select whether you'd like to set up JIRA in demonstration or production mode." Two options are presented:

- Set it up for me**: This option is described as a quick setup for a demonstration or evaluation environment. It mentions that JIRA will be configured for you online with a working internet connection to generate an evaluation license.
- I'll set it up myself**: This option is described as setting up and configuring JIRA manually, recommended for production environments or if there's no working internet connection. It features a wrench and screwdriver icon.

A large red rectangular box highlights the "I'll set it up myself" option. A blue "Next" button is located at the bottom right of the page.

Select **I'll set it up myself** and then click **Next**.

# JIRA Database Setup

## Welcome

Follow these steps to setup JRA



Database  Build In(for evaluation or demonstration)

Connection  My Own Database (recommended for production environment)

Build in database can be migrated to a database of your own later.

Learn more about connecting JIRA to a database.

[Next](#)

Now you get a page to set up the database. You need to select the database setup of the two available options on whether to choose the built-in database or choose your own.

- **External database** is used for production instance.
- **Built-in setup** is used for evaluation and demonstration purpose.

We will set up the built-in database for performing and hands-on on Katacoda environment.

# Application Properties to JIRA

**Set Up Application Properties**

---

**Existing data?** You can import your data from another installed or hosted JIRA server instead of completing this setup process.

Application Title	<input type="text" value="LinOxide"/>
	The name of this installation.
Mode	<input checked="" type="radio"/> <b>Public</b> The name of this installation.
	<input type="radio"/> <b>Private</b> Only administration can create new users.
Base URL	<input type="text" value="http://17.25.10.17:8181"/>
	The base URL for this installation of JIRA. All links created will be prefixed by this URL.

---

**Next**

Set up the **application properties** to JIRA as

- Enter the **application title** in the text box. This title will be displayed on the JIRA login page and the dashboard.
  - Select if the issue tracker has to run in **private mode or public mode**.
  - Specify the **base URL** that you will be using to access JIRA. You can only configure JIRA to respond to one URL.
- Click **Next** to continue.

# Customizing JIRA

How do you want to use JIRA?

I want to use JIRA for project tracking

We'll install JIRA for you.



I want to use JIRA for software development

We'll install JIRA and JIRA Agile to enable agile software development workflows and task boards.



I want to use JIRA for IT requests

We'll install JIRA and JIRA Service Desk to help make IT request tracking easier for end users and your IT help desk.



**Next**

Now customize JIRA by selecting one of the three options as per your requirement and click **Next**.

# License Key and Administrator Account

## Specify your license key

You need a license key to set up JIRA. Enter your license or generate a new license key below. You need an account at [my.atlassian.com](https://my.atlassian.com) to generate a license key.

- I have a JIRA key    I have an account but no key  
 I don't have an account

Please enter your license key

Server B1

Your License Key

Next

- Add a license key to use JIRA.
- To obtain the license key, you have to log on to the Atlassian account.
- You will be provided with the three options. Select the option accordingly and add the license key.
- Set up the account for administrator by entering the administrator details.

Finally set up email notifications according to your needs and click on finish.

**Note:** Trial version of JIRA is available for only 30 days.

Now, the JIRA instance is ready for use. You will be learning about the projects in the next section.

# Projects

## JIRA Project

A **JIRA project** is a collection of issues. Teams use the project to

- Coordinate the development of a product
- Track a project
- Manage a help desk and many more.

A JIRA project can be configured and then customized to suit the needs of the user and the team.

## User Management in JIRA

A user is the one who can access the JIRA application.

**User management** helps you to

- Create, edit and remove a user
- Assign users to groups, project roles, and applications
- Monitor a user's activity
- Prevent automatic login
- Manage password policy

## Project Roles in JIRA

**Project roles** are a flexible way to associate users and groups with a particular project. The default roles in JIRA are

- Administrators
- Developers
- Users

**To create a project role:**

- Click the cog icon present on the top right corner.
- Select **System > Roles** (under security menu)
- Enter the Role **Name** and **Description** in the text boxes, and click on **Add Project Role**.

# Managing Project Roles in JIRA

JIRA allows you to add particular people to the specific roles in the project. This can be done by

- Clicking the **cog icon** available on the top right corner.
- Choose **Projects** from the dropdown list.
- Choose the project to which you wish to add the users to the specific roles.
- Choose **Roles** in the left menu.
- Hover over the **Users column** for the project role that you want to add.
- Click **Add Users** button.
- Type the name of the user and click **Update**.

You can also add groups to the specific roles using the above procedure.

## Components

**Components** are the subsections of a project. They are the generic containers for issues.

- Used to classify the issues in a project into smaller groups.
- Managed by only by the project admins and JIRA admins.

Components add structure to the projects by breaking them into teams, features, modules, sub-projects and many more.

You can also generate the reports and collect statistics using components.

## Versions

**Versions** is the point-in-time for a project.

They help you

- Organize the work by giving milestones to aim for
- Schedule and organize the releases.

After creating a version and assigning issues to it, you can use it to filter the information in many reports.

You can also assign the issues in the project to a specific version, and build up the needed work to complete the version.

## Archiving a Project

**Archiving a project** helps you to preserve the data of the project that is deleted from the JIRA instance, for future auditing purpose.

A project can be archived in two ways.

- **Online archiving**
  - With online archiving, you can **hide the project** or make it read-only.
- **Offline archiving**
  - With offline archiving, you have an option to **restore the deleted project**.

*You will be learning about the issues in a project in the next section.*

## Issues in JIRA

### Issue Fields

The following are some of the important fields of an issue.

#### **Issue type:**

- JIRA is used to track the issues of different types.
- Bug, Improvement, New Feature, Task, and Custom Issue are some of the default issue types present in JIRA.

#### **Priority:**

- Defines the importance in relation to other issues.
- Highest, High, Medium, Low, and Lowest are some of the default priorities present in JIRA.

# Issue Fields

## Assignee:

- The person under whose name the issue is currently assigned.

## Reporter:

- A person who has entered the issue into the system.

## Due Date:

- Date by which the issue is scheduled to be completed.

## Time tracking:

## It shows

- The **Original estimate** of the time required to complete an issue.
- The **Remaining estimate** of the time required to resolve an issue.

# Issue Types

Since JIRA handles a lot of tasks, many types of issues are present to identify the work and categorize the issue.

**The following are some of the issue types present in JIRA.**

## Bug

- An issue that prevents the function of the product.

## Epic

- A big user story that requires to be broken down.

## New Feature

- A new feature of the product that needs to be developed.

## Task

- A task that needs to be completed to achieve the team's goal.

## Improvement

- An improvement to the existing task.

# Editing an Issue

**Edit Issue project permission** is required for the issues relevant project to edit an issue.

An issue can be edited by

- Clicking on the **Issues** button in the header menu.
- Select the **My Open Issues**.
- Select the issue to be edited.
- Click **Edit** button.

## Commenting on an issue

You can add a comment while working with an issue. Comment option is available at the bottom of the issue screen under the **Activity** section.

- If you want to mention any user in the comment, you need to use @ symbol followed by the user's name.

# Cloning, Linking, and Moving an Issue

## Cloning an issue

- **Cloning** an issue lets you to quickly create a duplicate of that issue within the same project.

## Linking an Issue

- **Linking** an issue allows you to create a connection between two existing issues. Two issues can either be on the same server or the different servers of JIRA.

## Moving an Issue

- **Moving** an issue helps you to move an issue from one project to the other.

# Documenting the Work Done on an Issue

Once the issue is opened, you see a **Time Tracking** option on the bottom right of the view issue screen. This shows the original, remaining and the logged time of an issue.

Enter the time spent on the issue by clicking **More** button and selecting **Log Work**.

- You need to specify the time tracking details using W, D, H and M that specifies the week, days, hours, and minutes

You can modify the status of the work by clicking the current status of the workflow which is shown on the top, and click on **Update**. The issue status gets reflected immediately.

## Security Controls on Issue

Setting the **security level** on an issue restricts the access of that issue only to people who are the members of the chosen security level.

You can set the security levels by

- Adding a **Security Scheme**
- Adding a **Security Level**
- Adding **Users or Groups** to the security level
- **Associating security scheme** to the project

## Security Controls on Issue

### Adding a Security Scheme, Security Level and members

- Go to the Administrators menu (cog icon) and click **Issues**.
- Scroll down to the bottom and select **Issue Security Schemes**.
- Click **Add Issue Security Scheme**
- Fill in the requested details and click **Add**
- Once the security scheme is added, you find the **Security Levels** under the **Operations** column to add a security level.
- Add the members to the security level by clicking the **Add** button under the **Operations** column.

## **Associating the Security Scheme to the project**

- Select the project.
- Choose the Issue Security from the left menu.
- Click **Actions > Select a Scheme**.

## **Importing Issues Using a CSV file**

You will be able to create issues in bulk using a **Comma-Separated Value File (CSV)**.

First, you need to create a CSV file with the issue fields.

### **CSV file structure**

Summary, Description, Status

"Login fails", "This is on a new line", Open

### **CSV file can be uploaded in two ways.**

1. By selecting the **Issues** drop-down menu from the system dashboard and choosing **Import Issues from CSV**.
2. Go to the **System Administration menu > Import/Export section > External System Import > Import from Comma-Separated Values (CSV)**.

## **Integration of Issues to bitbucket**

### **By integrating JIRA issues with bitbucket, you can**

- Automatically link code to JIRA.
- Automatically link issues and keep your team up to date.
- Get context on the work and the team without switching the applications.
- Evaluate the status of a JIRA issue development at a glance.

# Workflows

## Introduction to Workflows

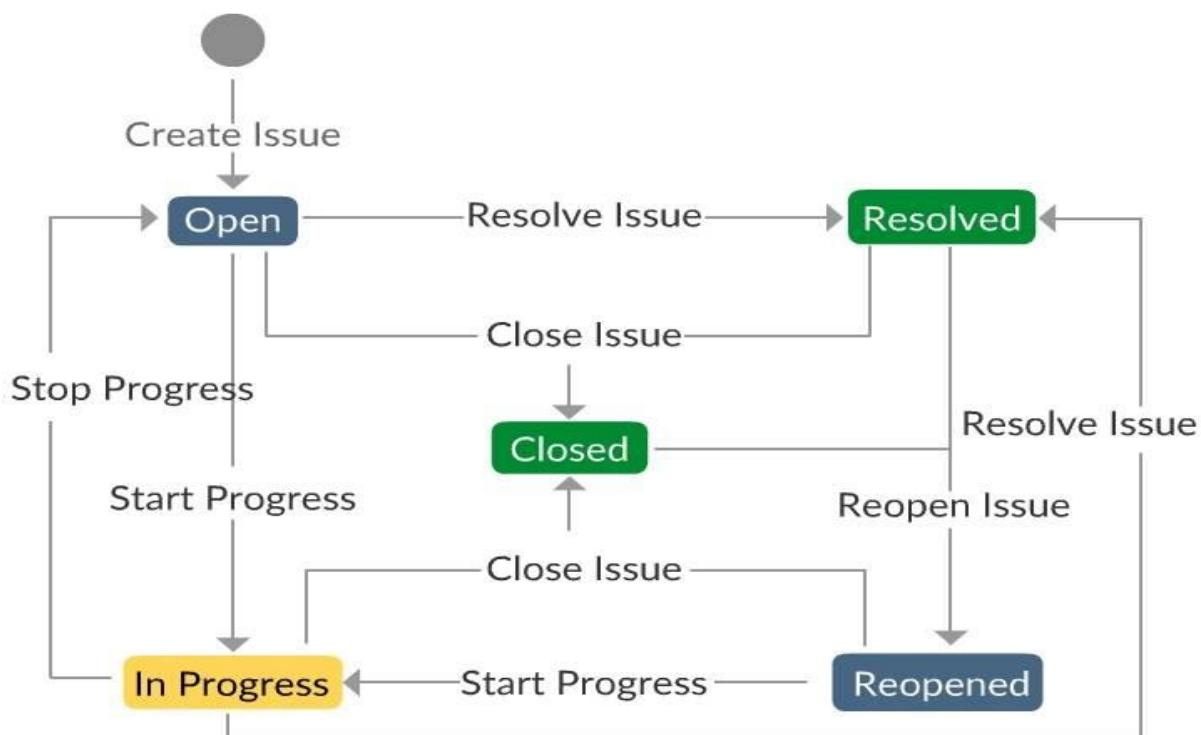
A team has a certain business process that it follows for taking work item, bug or an idea from beginning to end. **Workflow** represents that process. The work item has definite states that represent the process status at a given point of time.

**For example**, Open, In Progress, Resolved, Reopened and Closed represent the states of the Workflow. To proceed from one status to another status, work item moves along a transition.

If the transitions are applied with conditions, only a team member with certain roles or if they include a comment, or if they have approval, can only execute them.

In JIRA, Workflow **tracks the lifecycle of an issue**.

## JIRA Workflows



A **JIRA workflow** is a combination of **statuses** and **transitions** that an issue moves through during its lifecycle. It typically represents the processes within an organization. There are some default workflows present in JIRA. These can be copied and edited according to your requirement.

*An example of JIRA workflow is shown in the above image.*

## Statuses And Transitions

### Statuses

- Status indicates the state of an issue at a particular point in the workflow.
- An issue can be in only one status at a given point in time.
- You have an option to specify the properties for the statuses.

### Transitions

- A transition is a link between two statuses that allows an issue to move one status to another.
- It is a one-way link. So if you want to move the status back and forth, you need to create two transitions.

## Active and Inactive Workflows

### Inactive Workflow

- The workflow that is currently **not used** by any projects is known as **Inactive workflow**.
- Workflow steps and transitions can be edited directly, as no current issues will be transitioning through an inactive workflow.

### Active Workflow

- The workflow that is currently **used** by one or more projects is known as **Active workflow**.
- To edit an active workflow, JIRA first creates a draft of it, that you can then modify as you see fit and finally publish the draft.

# Custom Event

A **custom event** is used to generate an email notification from a particular workflow transition post function.

## Configuring a notification for a custom event by

- Adding the custom event to the system.
- Configuring the notification scheme to send an email when the custom event is fired.
- Configuring the workflow transition post function to fire the custom event.

# Validators in JIRA Workflow

A **validator** checks whether the certain input to the transition is valid, before performing the transition.

## Configuring Validators in JIRA Workflow

- Just like the post function, you need to select the transition that needs to be validated.
- Click **Add Validator**.
- Select the validator and click **Add**.
- Select **Permission** on the add parameters to validators page and click **Add**.

# Boards and Roadmaps

## Roadmaps

A **roadmap** is a strategy or a high-level plan visualized on a timeline. Roadmaps are about vision, overview, priorities, highlight, shared understanding, expectations, etc.

Product owners use roadmaps to estimate the **future product functionality** and when **new features** will be released.

### A **roadmap** consists of:

- A timeline with the project divided into stages
- High-level tasks

- Workstreams
- Project goals
- Risks
- Dependencies
- Key Events.

## JIRA Boards

A board displays the issues from one or more projects. It gives a flexible way of managing, viewing, and reporting the progressing work.

A single project can have multiple boards or a combination of Scrum and Kanban boards.

Two popular boards present in JIRA are

- **Scrum board** and
- **Kanban board.**

## Scrum Board

A **Scrum board** was created using the Scrum framework. A Scrum Board is a tool that helps Teams make Sprint Backlog items visible. The board can take many physical and virtual forms but it performs the same function regardless of how it looks. The board is updated by the Team and shows all items that need to be completed for the current Sprint.

The backlog of a Scrum board shows the issues for your project grouped into a backlog and sprints. In the Scrum backlog, you can create and update issues, drag and drop issues to rank them, or assign them to sprints, epics, or versions, manage epics, and more.

### Scrum boards have

- **Plan mode:** Includes moving an issue from backlogs and giving a time estimate for each issue.
- **Active sprint mode:** Allows you to move the issues across the statuses.

# Kanban Board

- **Kanban**, in contrast to the Scrum board, permits users to begin work without any structured plan and does not even have a planning mode.
- The same column-based interface is used as Scrum active sprint for tracking the task status, without the ability to organize sprints.
- Instead of dealing with a portion of issues in a project, the Kanban board deals with all the issues.

## Search Features

### Basic Search in JIRA

The screenshot shows the JIRA web interface. At the top, there is a navigation bar with links for Dashboards, Project, Issues, Agile, Test, and Create. Below the navigation bar, on the left, is a sidebar with a link to the "Zephyr Dashboard". The main content area is titled "Sample Dashboard" and contains a section titled "Two Dimensional Filter Statistics by Assignee". This section includes a table with three columns: Issue Type, Assignee, and Unassigned. The data in the table is as follows:

Issue Type	Assignee	Unassigned
Task	5	0
Test	1	1
Test Case	1	1

Below the table, there is a summary: "Total Unique Issues: 7 2". At the bottom of the dashboard section, it says "Filtered by: Filter... Showing 3 of 3 stati".

*The generic search offers a user-friendly interface that permits you to define complex queries.*

Basic search is performed by navigating to **Issues > Search for Issues > Basic**.

*There are two ways to perform a basic search.*

- **Search against a specified text**
- **Search against a specified field**

**For example:**

If the name of the issue is "**To test the bug**", you just need to type either "**test**" or "**bug**" in the search box.

Combining the above two ways will narrow your search results.

## Quick Search in JIRA

A **quick search box** is located at the top right of your screen. It can be accessed from any page in JIRA.

*A quick search can be done by typing the key to the issue in the quick search box. It takes you straight to the issue page.*

**For example:**

If you want to search for the second issue in the software development project(key = SD), you just need to type SD-2 in the quick search box. It takes you directly to the SD-2 issue page.

# Advanced Search in JIRA

The screenshot shows the JIRA navigation bar with options like Dashboards, Project, Issues, Agile, Test, and Create. Below the navigation bar, there are two cards: "Zephyr Dashboard" and "Sample Dashboard". The "Sample Dashboard" card displays a table titled "Two Dimensional Filter Statistics by Assignee". The table has columns for Issue Type, Assignee, and Unassigned. The data is as follows:

Issue Type	Assignee	Unassigned
Task	5	0
Test	1	1
Test Case	1	1
<b>Total Unique Issues:</b>	<b>7</b>	<b>2</b>

Below the table, it says "Filtered by: Filter..." and "Showing 3 of 3 stati".

An **advanced search** allows you to use structured queries to search for JIRA issues.

- Advanced search is performed by opening the issue navigator (**Issues > Search for issues > Advanced**).
- **JIRA Query Language(JQL)** is used to construct queries for advanced search.

## Advanced Query for a single word

**Syntax:** field = field value

**For example:** priority = Low

# Advanced Search in JIRA

## **Keywords used in query syntax:**

In JQL a keyword is a word or phrase that does something. It joins the two or more clauses together to form a more complex query.

- The most commonly used keyword are AND, OR, NOT, EMPTY, NO and ORDER BY.

**Syntax:** field = field name keyword field = field name

**For example:** priority = Low AND Type = Task

You can save the advanced queries by clicking the **Save as** option on the top.

A field with no value can be found by using the Empty or NO keyword.

**For example:** due date = null

## **Filters in Issue Navigator**

The screenshot shows the Jira Issue Navigator interface. At the top, there's a navigation bar with links for Dashboards, Project, Issues, Agile, Test, and Create. Below the navigation bar, there's a search bar and a 'Save as' button. On the left, a sidebar titled 'FILTERS' contains links for New filters, Find filters, My Open Issues, Reported by Me, Recently Viewed, All Issues, and Favorite Filters (with a link to my summary). The main area has two columns: 'Order by' and 'Details'. The 'Order by' column lists issues SD-9 and SD-7. The 'Details' column shows the following information for the first issue:

Type:	<input checked="" type="checkbox"/> Task
Status:	PURCH
Priority:	↑ Highest
Resolution:	Unresolved
Affects Version/s:	1.0
Fix Version/s:	None
Labels:	None

*If you find yourself searching for JIRA Issues daily, you will benefit immensely by setting up the filters that make your work faster.*

Once you run a query in JQL and get a result that you like, click **Save as** button to save the query as a filter for later use.

**Filters** can be described as the **saved searches**.

Filters are used to

- Share and email the search results to the other coworkers.
- Create a list of favorite filters.
- Display the search results in a dashboard gadget.

## JQL Queries in Filters

*A query has the basic three parameters.*

- Field
- Operator
- value

*These parameters are linked with the selected keyword.*

**Examples:**

1. project = Software Development AND type = Task

*Query results all the issues from the Software Development Project, which are Tasks.*

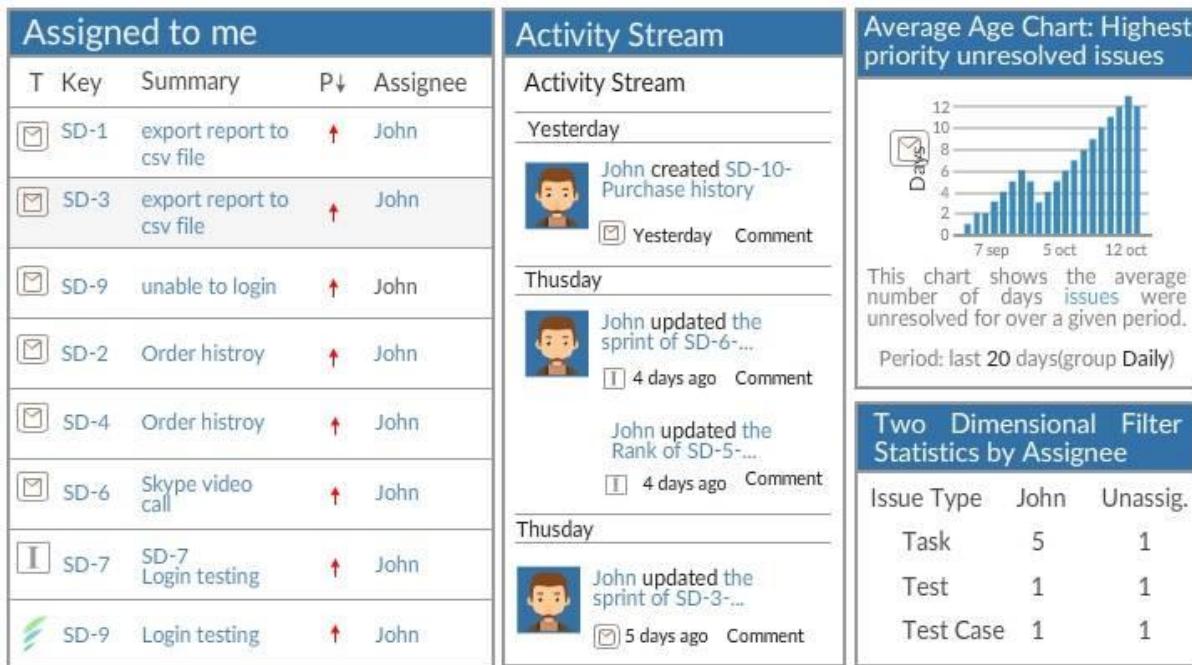
2. project = Software development AND type = Bug AND assignee = Tom Developer AND reporter = John Smith.

*Query results the bugs assigned to Tom Developer and reported to John Smith from the Software Development project.*

# Personalization and Reporting

## Introduction to JIRA Dashboard

### Sample Dashboard



**JIRA Dashboard** is the main display that you see after logging into JIRA.

### JIRA Dashboard helps

- To organize your projects, assignments, and achievements in different charts.
- The team to monitor the status of the project live on TVs around the workspace.

A sample dashboard is shown in the above image.

# Creating a JIRA Dashboard

You can create and customize the dashboard to display the information you need.

## Creating a Dashboard.

- Click the **Dashboards > Manage Dashboards > Create New Dashboard**.
- **Name** your dashboard and give the **description** so that your team knows when to use.
- Fill the remaining fields and click **Add**.

## Dashboard Gadgets

*Gadgets allows you to customize an information that appears on the Dashboard in JIRA applications. Once the dashboard is created, you can set the layout and add the required gadgets to the dashboard. You also have the filters option to increase the visibility of the results on the gadget.*

The following are some of the gadgets present in JIRA.

- **Activity Stream:** Displays all the activities done in your instance.
- **Assigned to me:** Shows all the unresolved issues assigned to you.
- **Average age chart:** Tells you the average age of the unresolved problems.
- **Two-dimensional filter statistics by assignee:** Displays the data based on a specified issue filter.
- **Filter results:** Shows the issues of the specified issue filter.

## Generating a JIRA Report

*JIRA provides reports that will show statistics for projects, versions, users and other issue fields. Reports can be accessed directly from the project overview summary.*

It helps you to track and analyze the teamwork throughout the project. JIRA provides various reports of different categories like Agile, Issue analysis, Forecast, and management, etc.

**Following are some of the reports available in JIRA.**

- Burndown report
- Sprint report
- Time tracking report
- Version workload report
- Recently created issues report

## Getting JIRA Report from Market Place

In addition to reports offered by JIRA, you can also add the report to JIRA from the **Atlassian Marketplace** where you find the add-on that provides you with the additional reporting functionality.

### **Adding Reports to JIRA from Atlassian Marketplace:**

- Type **marketplace.atlassian.com** in the browser.
- Select the add-on and download the JAR file.
- Select **Add-ons** from the administration section of JIRA.
- Click the **manage add-ons** from the left-hand panel.
- Click **Upload add-on** and click continue to add the report.

## Test Management

### Intro to Test Management

Many teams are utilizing JIRA for test case management so that the development and testing can stay in one system.

#### *How to set up JIRA for test case management?*

Separate test case functionality is not present in JIRA. You need to integrate with another testing tool to perform testing. One of the tools that you can integrate with JIRA is **Zephyr**.

### Zephyr Add-On in JIRA

**Zephyr** in JIRA is a native application that comes with JIRA. It brings the quality test management capabilities to JIRA project.

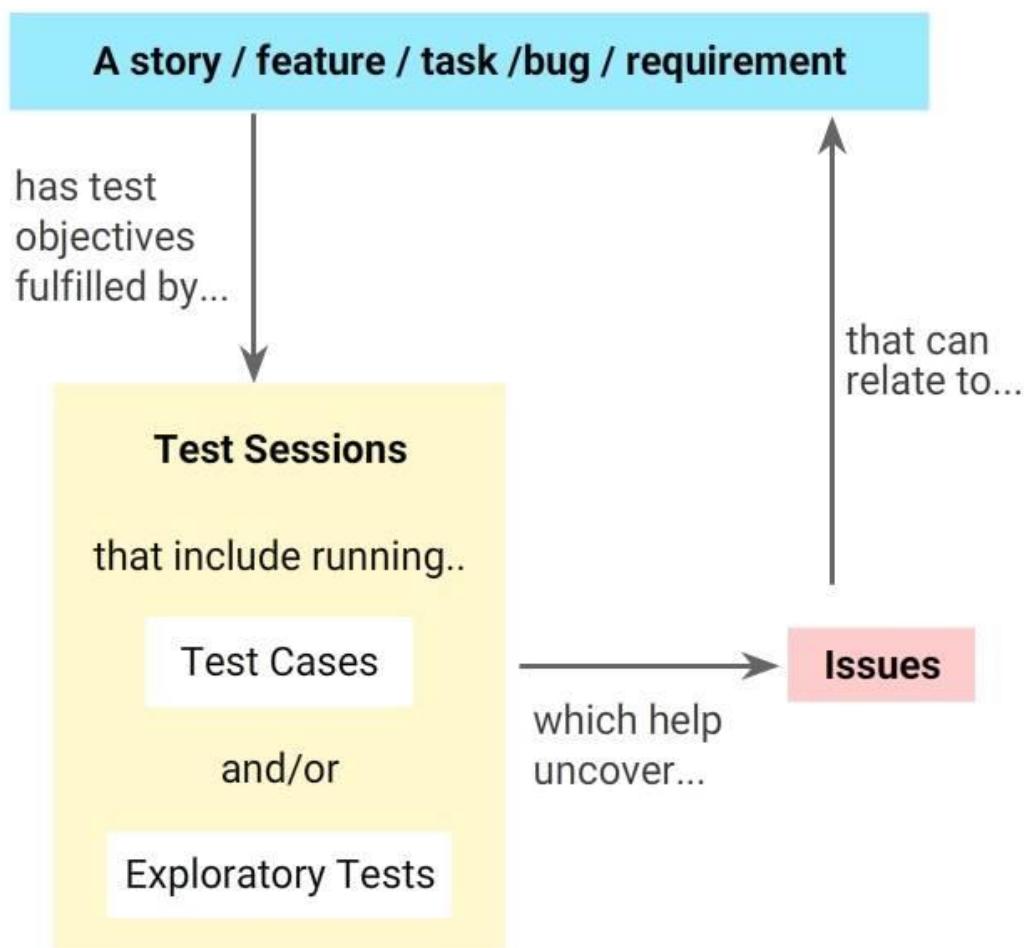
#### **Features of Zephyr**

1. Provides the testing functionality within JIRA.
2. Track quality metrics.

*When Zephyr is integrated with JIRA, you can*

- Write, modify and view test cases
- Create test cycles
- Add test cases to test cycles
- Assign the test executions
- Execute the test cycles
- View the test process

## Test Session



**Test sessions** helps you to carry out more than one test in a single session. They give detail information on who, how, when and why the test was done.

*Test sessions are useful for*

- Planning

- Executing
- Tracking manual testing.

## Hands-on Scenario

Note : After launching the hands-on, click on the credentials button on the top right to type something on the terminal.

### Installation steps

1. sudo wget <https://www.atlassian.com/software/jira/downloads/binary/atlassian-jira-core-8.5.7-x64.bin>
2. sudo chmod a+x atlassian-jira-core-8.5.7-x64.bin
3. sudo ./atlassian-jira-core-8.5.7-x64.bin
4. sudo /etc/init.d/jira start

Once the installation is complete, access Jira instance on 8080 port using the below link.

<http://server-IP:8080>

### Jira Instance Set up

1. Set up JIRA by selecting the **I'll set it up myself** option.
2. Set up the database by selecting the **Built-in setup** option.
3. Enter the title mode as public and the base URL. On the next page it asks you How do you want to use JIRA? Choose Jira and click on next
4. Specify License key - Choose the options accordingly and apply the License **Note** : If you are unable to connect to Atlassian, sign up or log in to the account using <https://my.atlassian.com/license/evaluation> to generate a license key
5. Set up the administrator account by creating the Atlassian account and entering the license key.

### Add Zephyr for Jira - Test Management

1. Click the **cog** icon.
2. Choose **Manage apps** from the drop-down list.
3. Search for **Zephyr for Jira - Test Management** and install it. Once Zephyr is integrated with JIRA, the 'Test' tab appears on the top menu.

**Note** : If you face any error while applying license follow the below steps :

1. Generate a license by clicking on this [link](#)
2. Choose **Try it free > Server** > Generate Zephyr License
3. Copy the generated license in this path (cog icon > Atlassian Market Place > Manage Apps > ZEPHYR FOR JIRA > License > Update License)

## **Hands-on**

### **1.Create Project **PlayIssues****

Projects > Create Projects > Enter PlayIssues > Click on Submit

### **2.Configure **Test** Issue Type to the created Project**

Settings > Projects > Select Project > Issue Types > Actions > Edit Issue Type > Drag and drop the available issue types to the current scheme > Click on save

### **3.Create component **Testing****

Projects > Select project > Component > Enter the required details and click on Create

### **4.Create Version 1.0**

Projects > Select project > Versions > Enter the required details and click on Create

### **5.Create and configure the following fields using custom screens by clicking the **cog** icon > **Issues > Screens**.**

- Summary
- Issue Type
- Fix Version/s
- Component/s
- Description
- Priority
- Due Date
- Assignee
- Reporter
- Attachment

### **6.Create Test in Jira**

### **7.Write Test cases for the created test**

8.Create a Test Cycle by clicking **Tests > Plan Test Cycle > Create New Cycle.**

9.Enter details in the required fields, and click **Save**.

10.Add test cases to the test cycle by clicking the icon > **Add Tests**.

11.Execute test cases in the test cycle by clicking the (Execute test) icon.

12.Create a dashboard named **Play-Issues** by  
Clicking **Dashboards > Manage Dashboards > Create New Dashboard**. Enter details in the fields, and click *Add\**.

# Prodigious Git

Introduction to Version Control System

Git Basics

Git History

Collaborate Using Git

Building New Features Using Branches

Merging

Conflicts

Undo Changes

Delete Files

Run Through

Prodigious Git - Final Assessment

Introduction to Version Control System

## Developer's Dilemma

Imagine you and your friend are working on a dashboard feature.

How would you:

- **Find the changes** made by your friend?
- Get the changes made by your friend, and **add new features** to it?
- **Undo the changes** made by your friend a month ago as they are identified as bugs now?

If you are wondering, keep reading to know how Git can help you!

## Course Overview

In this course, you will learn:

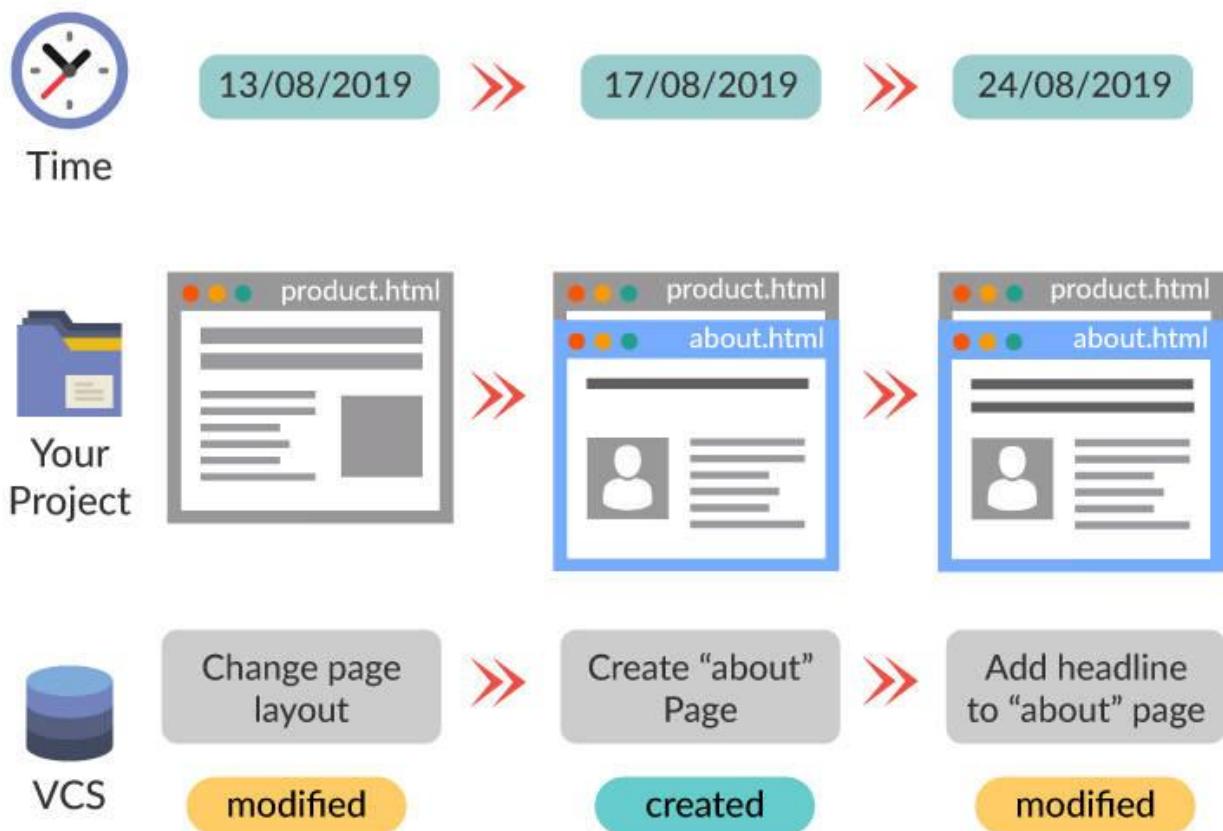
- Distributed Version Control basics
- Basic Git commands
- Core concepts of Git like:

- Creating new features without affecting the current working version (Branching)
- Collaborating with other developers (Clone, Push, Rebase and Merge Conflicts)
- Saving new changes temporarily (Stashing)
- Selecting particular changes from others (Cherry-pick)

By the end of this course, you will be able to use Git.

Let's get started!

## What is Version Control?



**Version Control** is a system which **records the changes made to a file** so that you can recall a specific version later.

By using Version Control in your project, you can easily find out:

- **What** are the changes made
- **Which** files are changed
- **When** were the changes made
- **Who** has made the changes

## Why Use a Version Control System?

**In all software projects, the source code is the *crown jewel* that has to be protected**

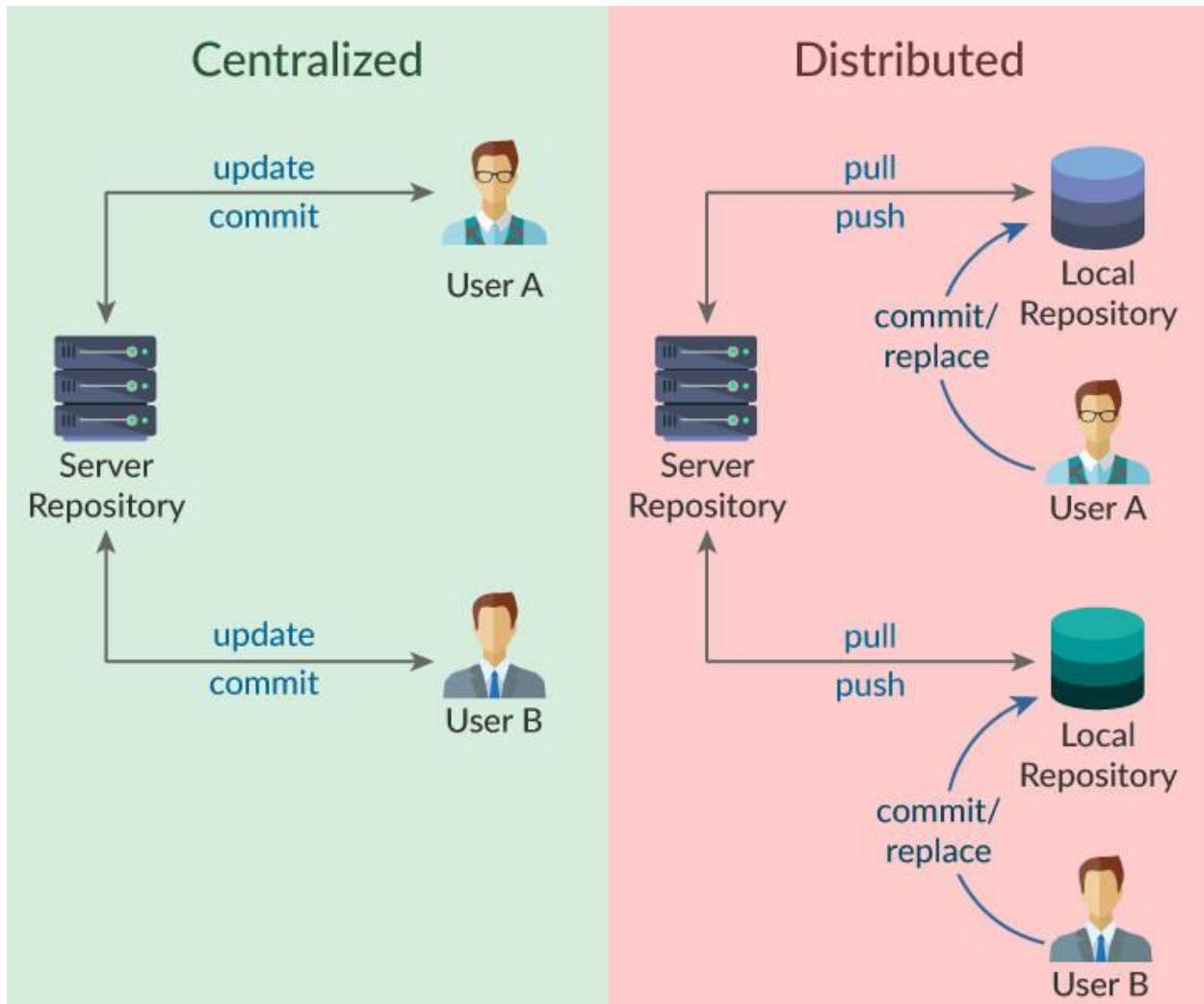
A Version Control System (VCS) gives you many super-powers that help you track changes and secure your *crown jewel* such as:

- **Collaborating with big teams** smoothly to develop code
- **Reverting the code** to the bug-free version in case any minor change in the new version introduces a bug
- Providing **backup** when the central server crashes
- Rendering an option to **time-travel your project** and go to a specific version

## Types of Version Control Systems

- **Local**: It allows you to copy files into another directory and rename it (For example, project1.1). This method is error-prone and introduces redundancy.
- **Centralized**: All version files are present in a single central server. For example, CVS, SVN, and Perforce.
- **Distributed**: All changes are available in the server as well as in local machines. For example, Git and Mercurial.

# Centralized vs. Distributed Version Control Systems



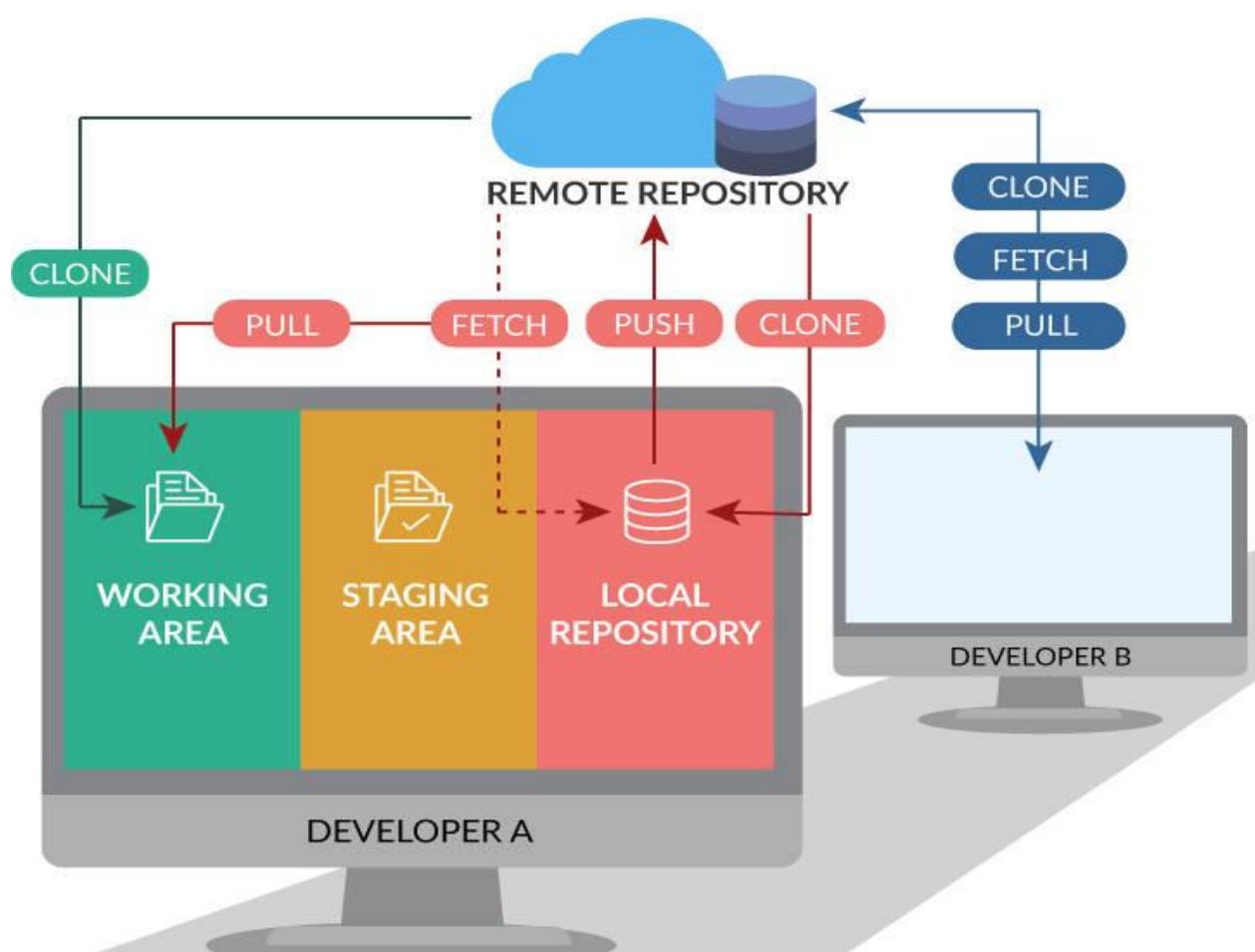
Centralized	Distributed
You can keep changes only in the server	You can keep changes locally (commit) as well
Changes can be merged in the server (remote) alone	Changes can be merged locally as well as remotely
Work gets interrupted if the server is down	The project is present with all the team members to work locally

**Git** is the widely popular Distributed Version Control System which you will be learning in this course.

Are you ready?

Let's *Git Set, Go!*

## Cheat Sheet



- `git clone`: Get the complete project from remote to your local machine
- `git pull origin <branch_name>`: Get the new changes from remote branch to local branch
- `git push origin <branch_name>`: Send your local branch changes to the remote branch
- `git remote add <name> <url>`: Add a new remote repo link to your local repo
- `git remote -v`: List all the remote repo URLs linked to your local repo

image: git collaboration commands

## Git Basics

### How to Install Git?

For Windows

[Windows Download Link](#)

For Mac

[Mac Download Link](#)

For Linux

Type the following commands from the Shell.

```
$ sudo apt-get update  
$ sudo apt-get install git
```

After successful installation of Git, you can configure your user name and e-mail ID using the following commands.

```
$ git config --global user.name "First Last"  
$ git config --global user.email "myemail@domain.com"
```

## Git Terminologies

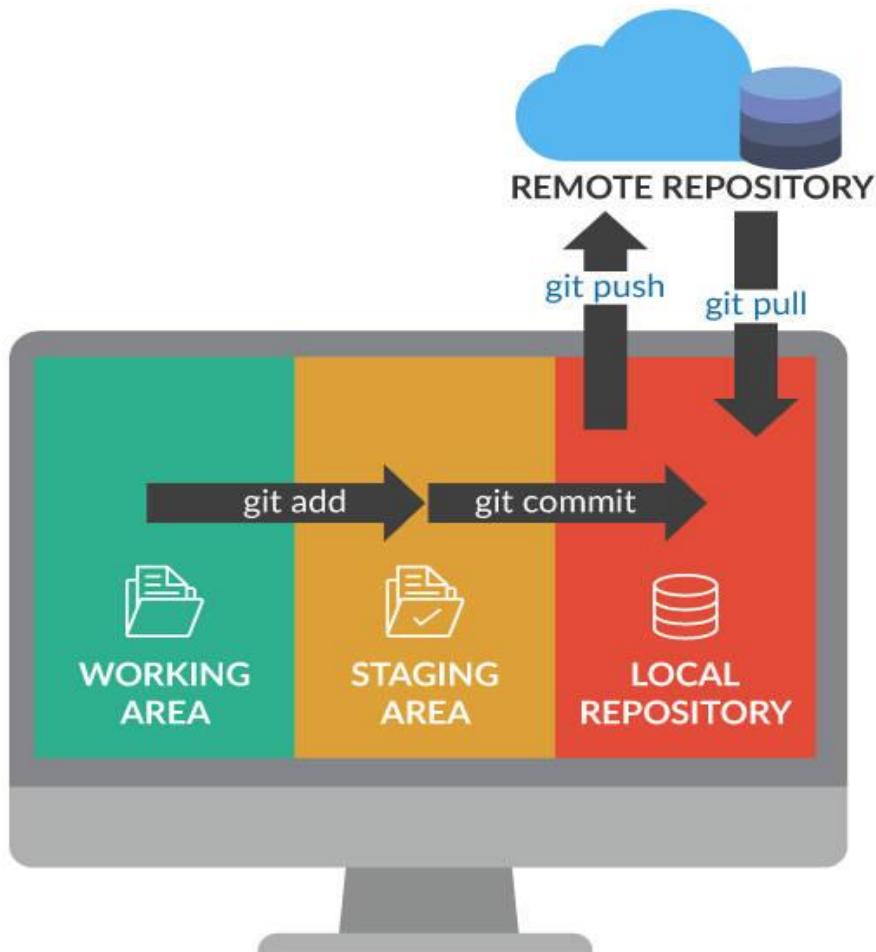
Before starting with the basics, let's explore a few terminologies:

- **Git Repository**: A directory with **.git folder**, where all the contents are tracked for changes.
- **Remote**: It refers to **a server** where the project code is present. For example, Github and Gitlab.
- **Commit**: It is similar to version. When you make changes in a file, you need to **commit the changes** in order **to save and create its new version**, which will create a unique commit hash (like version number).
- **Origin**: It is a variable where Git stores the URL of your remote repository. For example, origin => www.github.com/username/myrepo

# Git Basic Commands

- **git init** adds `.git` folder and **initializes the current folder to track its changes**
- **git status** displays the current state of the staging area and the working directory, that is, which files are added/removed/modified
- **git diff** **shows the exact changes** with line and column number
- **git add** adds the changes to the staging area. If you have added a new file, this command **starts tracking** the file for modifications.
- **git commit** will **save all the changes** with a unique hash number in the local repository
- **git push** sends the changes to the remote repository (server)

## Stages in Git



The following are the **three stages** in the Git workflow:

- **Working Area:** You can **edit files** using your favorite editor/Integrated Development Environment (IDE).

- **Staging Area:** You have made the changes and **added the changes to Git**. You can **still make changes** here. (From the analogy explained in the next card) It is like taking an item out of the box, where the box is the staging area. (`git add`)
- **Local Repository:** You have finalized the changes and **committed** them with a new hash and proper message. (`git commit`)

**Remote Repository:** You can now **push the changes to online platforms** like Github or Gitlab from where others can collaborate. (`git push`)

## Git Stages: Analogy

Git works in three stages known as **The Three Trees**: Working Area, Staging Area, and Local Repository, which means Git maintains three states of a file. To understand this better, **let us take an analogy of a box.**

Assume you are **packing items in the house in different boxes** and labeling them to identify it later.

- Get a table and keep all the items that you want to pack underneath. (`git init`)
- Now you might select specific kitchen items, dust them, and club similar items (like spoons) together. (*doing changes - Working area*)
- Add the items that are ready to the box. (`git add` - Staged)
- Seal the box and add a label - 'Kitchen Items'. (`git commit` - Committed)

After `git commit`, a unique hash is created and the changes are saved.

## Ignore or Keep

If you **do not want Git to track any file/directory**, you can add the file/directory to `.gitignore` file. To track an empty directory, you need to add `.gitkeep` to that empty directory, as Git ignores them by default.

## Git History

# Git History

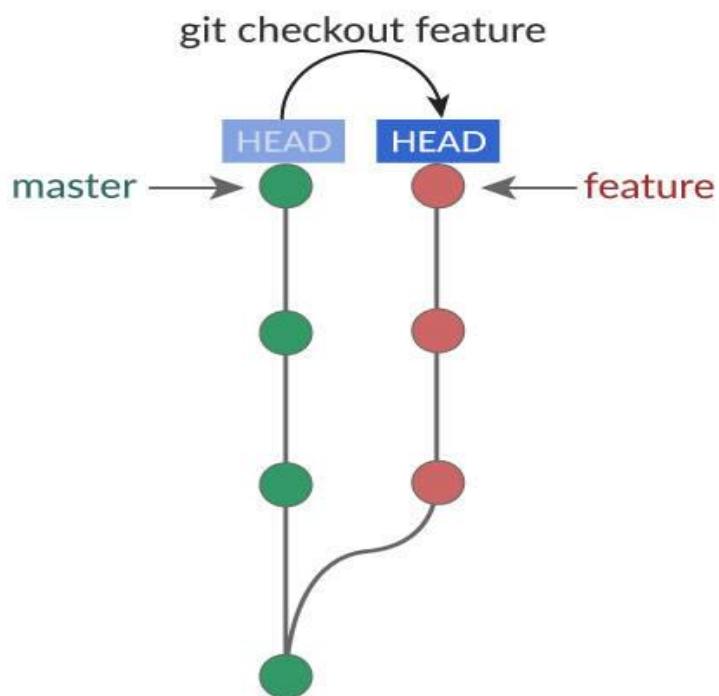
With multiple developers pushing new commits and changes, you might want to view the history of everything that happened with the repository.

Git provides the following commands, which allows you to **read and review your repo's history**:

- `git log`
- `git show`
- `git diff`

Before learning more about these tools, **you must first understand the HEAD and Dot operators**.

## What is HEAD in Git?

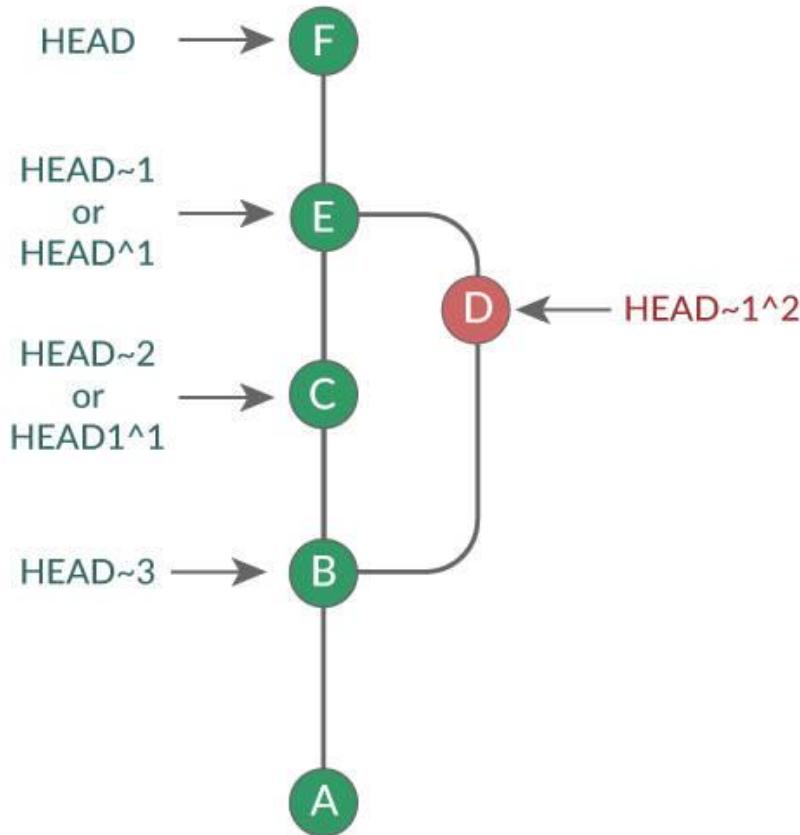


**HEAD** is a reference variable that always **points to the tip of your current branch, that is, recent commit of your current branch.**

**HEAD** can be used with the following symbols to refer to other commits:

- Tilde symbol (~): Used to point to the **previous commits from base HEAD**
- Caret symbol (^): Used to point to the **immediate parent commit** from the current referenced commit

## Using HEAD with ~ and ^



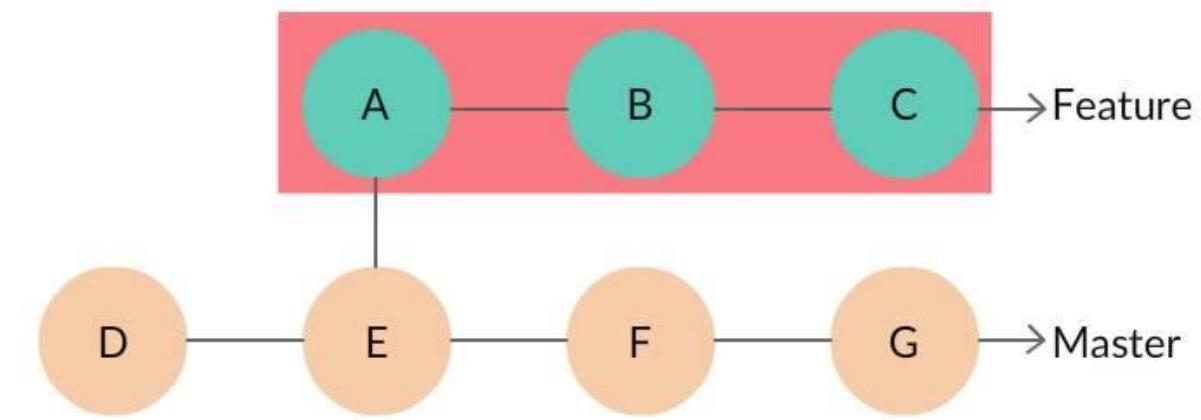
You can use both **tilde** and **caret** symbols in combination with **HEAD** to refer specific commit:

- **HEAD** means (the reference to) the **recent commit of current branch -> F**

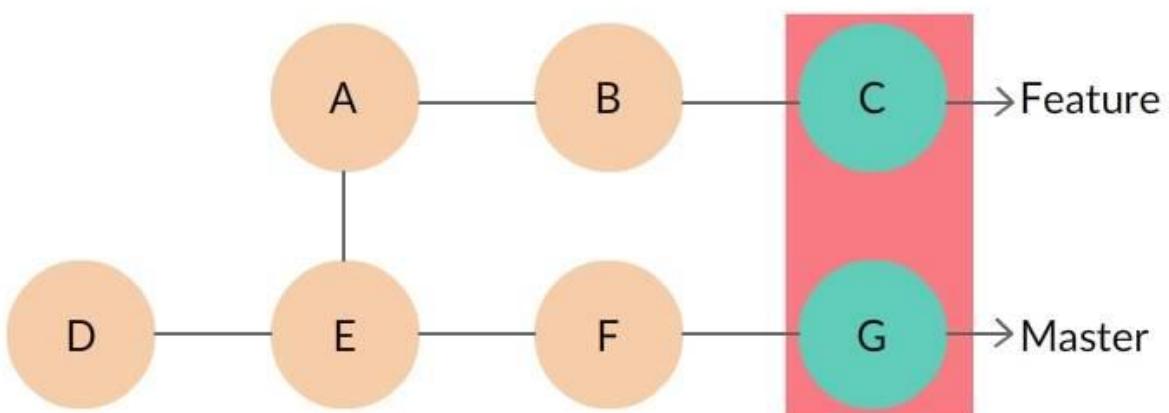
- **HEAD^1** means **First parent of F -> E**
- **HEAD^2** means **Second parent of F -> error** as there is *only one immediate* parent commit
- **HEAD~1** means (the reference to) **one commit before HEAD -> E**
- **HEAD~1^1** means **First parent of E -> C**
- **HEAD~1^2** means **Second parent of E -> D**
- **HEAD~1^3** means **Third parent of E -> error**
- **HEAD~3** means (the reference to) **three commits before HEAD -> B**

## Dot Operators

**Triple Dot Operator**



**Double Dot Operator**



*Double Dot Operator*

- It is the default operator in git diff
- **git diff master..feature** or **git diff master feature** command will display all the **differences from G to C** (that is, including **F and G**)

### *Triple Dot Operator*

- It shows the difference between master and feature branch **starting at the last common commit E**.
- `git diff master...feature` command's output would be the difference in *feature* branch (that is, only **A, B, and C**)

## Git Commands to View History

### Git Log

**Git log** command shows the list of commits in the current branch. You can use it in the following ways:

- `git log -2` displays the history of **last two commits**
- `git log commit_id` shows the history **starting from commit\_id**
- `git log filename` displays the list of commits for the file

### Flags

You can enhance the output of `git log` command using these optional flags:

- `--oneline`: Fits the log output to a single line
- `--decorate`: Adds a symbolic pointer to the output
- `--graph`: Gives a graphical representation to the log output
- `--grep=<pattern>`: Filters the log output and displays the output which matches the specified pattern

For example, `git log --oneline`

### Git Show

`git show` is a versatile command which is similar to `git log`, but with few additional features.

You can pass different arguments like:

- `commit_id`
- `tree`

It shows the author's name, timestamp, commit message, and difference from the previous commit.

For commit range `git show --oneline HEAD~2..HEAD`, it gets resolved, and each commit is displayed individually.

## Git Diff

`git diff` function takes two input datasets and **outputs the changes** between them.

Datasets can be commits, branch, files and more.

`git diff HEAD~1 HEAD` shows the difference between two commits.

Let us take a sample `git diff` output as follows.

```
diff --git a/sample.txt b/sample.txt
index 8d3g6cv..f94e50c 574641
--- a/sample.txt
+++ b/sample.txt
@@ -1 +1 @@
-this is a git diff test example
+this is a diff example
```

- The line `diff --git a/sample.txt b/sample.txt` displays the input file for git diff
- `--- a/sample.txt +++ b/sample.txt`

The changes from a/sample.txt are marked with --- and changes from b/sample.txt are marked with +++ symbol.

## Cheat Sheet

- `git log -p`: Prints full details of each commit
- `git log --grep-reflog=<pattern>`: Shows the list of commits when commit message matches regular expression pattern
- `git log --follow ./path/to/filename`: Shows the history for the current file
- `git show`: Outputs content changes of the specified commit
- `git diff --color-words`: Output has only the color-coded words that have changed
- `git diff --staged`: Shows the file differences between staging and the last committed version
- `git diff .path/to/file`: Shows changes in a file compared to the previous commit

## Collaborate Using Git

### Local to Remote

In the previous topic, whatever you have learned was mostly limited to your local machine. To collaborate with other developers, you need to **push** your work to the remote repository, and vice-versa, you need to **pull** others' work from remote to contribute your work to the project.

In Git, **remote** is a repository on a server where all your team members can place the code to collaborate

### Remote URL Types

You can keep your project code in remote servers like **GitHub**, **GitLab** or on a self-hosted server. Git sets a **default name for your remote URL as origin**

Remote URL can be one of the two types:

- **HTTPS URL** like `https://github.com/user/repo.git`: You can clone/push using your user name and password
- **SSH URL**, like `git@github.com:user/repo.git`: You need to configure SSH keys in Github/Gitlab account and local machine

Keep reading to find out the Git commands used to collaborate with other developers.

### Git Clone

To get source code of an already existing project from remote repo (For example, Github), you can use `git clone <url>` command.

For example, `git clone https://github.com/facebook/react.git`

This command **downloads the complete project**, all branches, commits and logs from the given remote URL (react repo here) to your local machine.

### Git Pull

Your teammate has pushed the changes to the project's remote repo where you are also working. You can now **pull the changes to your local machine using any one of the following commands**.

- `git pull` is the convenient shortcut key to fetch and merge the content.

- `git pull <remote_name> <branch_name>`
- `git fetch` command downloads the remote content to your local repo, **without changing your code changes.**
- `git fetch <remote_name> <branch_name>` fetches the content from that specific branch in remote to your current working area
- `git merge` command merges the fetched remote content to the local working tree.
- `git merge <remote_name>/<branch_name>` merges the content to the specified branch.

## Git Push

To keep your **changes and work in remote repo**, you need to **push the branch** using the command `git push <remote_name> <branch_name>`

Git push takes two arguments, namely:

- `<remote_name>`
- `<branch_name>`

For example, `git push origin master`, where:

- `origin` will contain the remote URL
- `master` is the branch that is pushed (We shall discuss branches later in this course)

## Working with Existing Projects

Peter is new to the project. For a particular task, he created ten new files in his local machine. His technical lead said there is a common repo where all the team members place their code. He asked Peter to push his files to the same repo. What should Peter do?

In such a scenario, you can **connect your local repo with an existing remote repo** using `git remote add` command.

# Git Remote

The syntax to link your local repo with remote repo is:

```
git remote add <remote_name> <remote_url>
```

It takes two arguments, namely:

- `<remote_name>`, let us take default name **origin**
- `<remote_url>`, let us take https\_url `https://github.com/play/repo.git`

For example: `git remote add origin https://github.com/play/repo.git`

Note: Your local repository can be linked to multiple remote repositories as `git remote add origin1 <url>`, `git remote add origin2 <url>`

## Emergency Tip



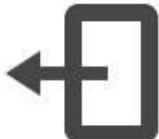
In case of fire



1. git commit



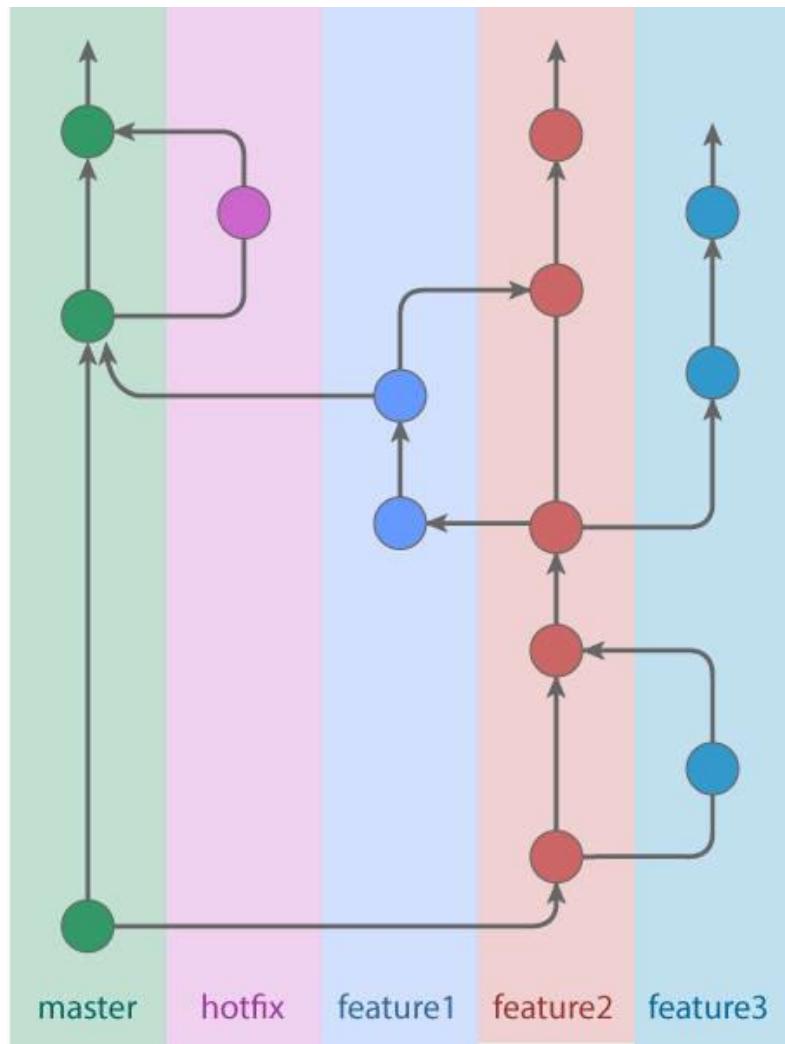
2. git push



3. leave building

## Building New Features Using Branches

### What is Branch in Git?



A **Branch** is a **copy of your complete project**, where you can add new changes and develop new features. Technically, it is a collection of commits.

When you create a **new Git project**, it has a branch called **master** by default. Even in the projects you clone from Github/Gitlab, a master branch is always present. **Master branch usually has the stable and working version of your application**, and hence it **can be modified only by a few access-controlled users**.

### How to Build Features?

To build new features without affecting the current working code, you need to:

- **Create new branch** from the master `git branch <branchname>`. Here you will write code for the new feature.
- **Merge** the feature branch with the master (or other branch where you want it to be). You can merge two branches locally or in remote.

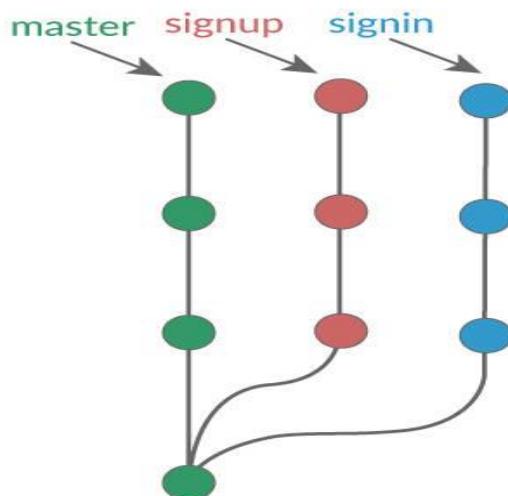
Keep reading to learn more about branches in Git.

## Branch Operations

You can do the following with branch:

- **Creating** new branch: `git checkout -b <branch-name>`
- **Pushing** branch from local to remote repo: `git push origin <branch-name>`
- **Renaming** branch:
  - Renaming local branch: `git branch -m old-name new-name`
  - Renaming remote branch: `git push origin :old-name new-name`
- **Deleting** branch:
  - Deleting local branch: `git branch -d <branch-name>`
  - Deleting remote branch: `git push origin -d <branch-name>`

## New Branches For New Features



•

- In a project, two developers need to **build sign-in and sign-up pages at the same time**. How can they do that without affecting the existing application?
- They can **create new branches for each new feature**, such as `signin` and `signup` branches **and work on the features parallelly**.

## Building New Feature: Signin

For a new feature to be built:

- **Create a new branch** from the master branch: `git checkout -b signin`
- **Add your new code** in the new feature branch: `git add -A`
- **Commit your changes** once done: `git commit -m "add signin page"`
- **Push your changes** to the remote: `git push origin signin`

As your sign-in feature is ready in a different branch, now you can **merge it with the master branch**.

Note: Merging will be discussed in the next topic.

## Tip on Branching

To see the **difference between two branches** (master branch and test branch), you can execute the following command

`git diff master test`

## Cheat Sheet

- `git branch -a`: Lists all the branches
- `git branch -d <branch-name>`: Deletes the branch in local repo
- `git checkout -b`: Creates a branch and switches to it
- `git checkout <branch-name>`: Switches to the provided branch

## Merging

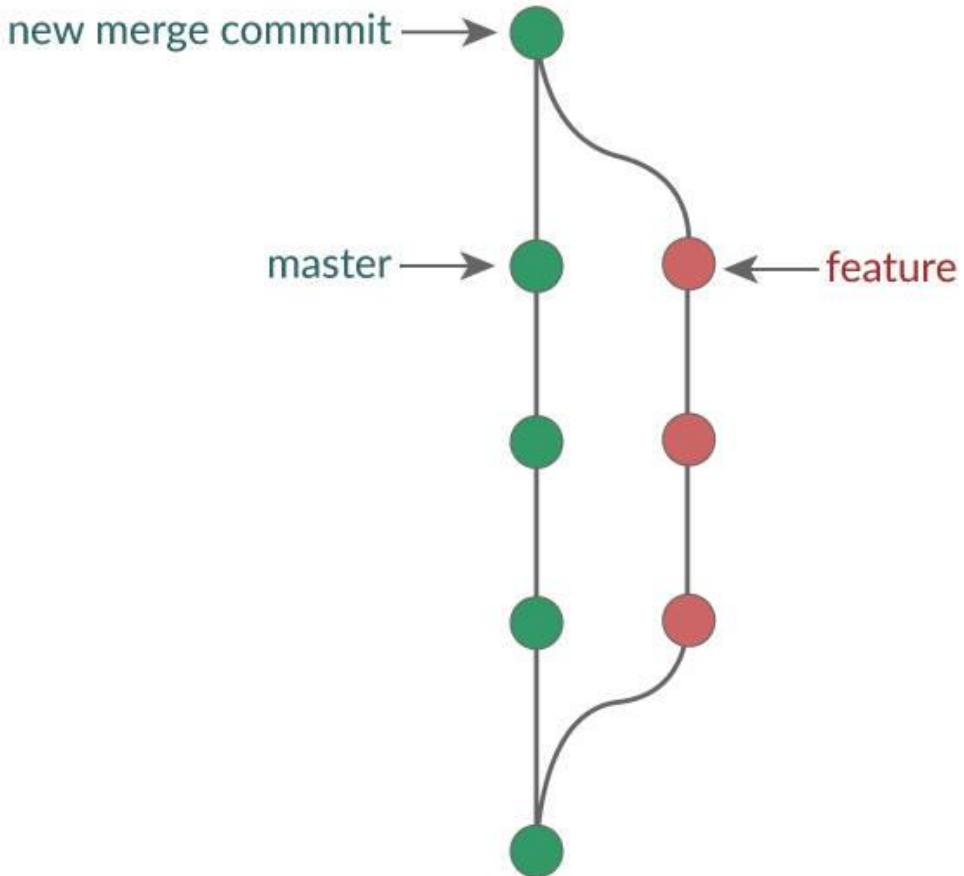
## Integrating Changes

Once you have developed your feature in a separate branch, you need to integrate the feature and master branch. You can do that using one of the following two commands:

- merge
- rebase

## Merging

### What is Git Merge?



Merge is an operation to integrate changes from one branch to another branch by adding a new commit

### Merging Feature Branch: Signin to Master

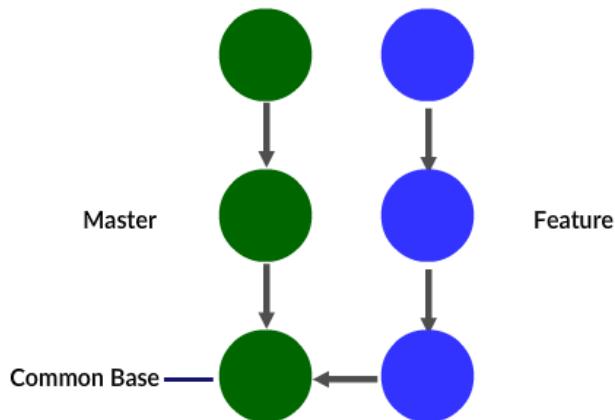
You can merge the signin branch with the master in two ways:

- In your local and push master: `git merge master signin && git push origin master`
- In remote server: You need to create a new Pull request(or Merge request) from `signin` branch to `master` branch in Github portal

Note: Even if you merge your feature branch with your local master, you will be unable to push your code to remote if you do not have appropriate access (like admin access).

## Rebase

### What is Git Rebase?



Rebase is an operation to integrate changes from one branch to another. It is an alternative to the `Merge` command.

If you are re-basing `feature` branch with `master`, the command will remove the original commits on `feature`, bring all the commits from master to feature and add the original commits on top of it.

- This **changes the commit hashes** and re-writes the history (as shown in gif in a different color).
- This gives **linear flow of development** when you look at the history of repo.

## Merge vs. Rebase

While **merge** and **rebase** are both used to integrate changes of two branches, let us see how they differ and which one you can use.

	Merge	Rebase
What it does?	<b>Creates new commit</b> while integrating changes from one branch to another	<b>Does not create any new commit</b> while integrating changes from one branch to another
Safe or not?	It is <b>safe in nature</b> , as it does not re-write the history of feature branch	It is <b>destructive in nature</b> , if you rebase feature branch with latest changes in master, your feature branch commit hashes changes.
When to use?	<b>When your feature branch is in remote</b> and someone else is creating new branch on top of your feature branch. Use <b>git merge</b> to get the changes from master to feature.	<b>When your feature branch is local</b> and changing history will not affect others. Use <b>git rebase</b> to get new changes from master to your feature branch.

## Cheat Sheet

- **git merge <branch>**: Merges <branch> to current branch
- **git rebase <base>**: Rebases the current branch onto the base (branch name, commit ID, tag)
- **git rebase -i <base>**: Performs interactive rebase. Launches editor where you can specify command on each commit to transform it.

- `git rebase --abort`: Cancels rebasing and goes back to the previous commit
- `git rebase --continue`: Continues rebasing after resolving the rebase conflicts
- `git pull --rebase`: Fetches the remote copy and rebases it into the current branch of the local copy

## Conflicts

### The Conflict

Suppose you have planned to go for a movie with your family at 8 PM tomorrow. And **at the same time**, your best friend has asked you to join for dinner.

What will you do? Confused?

Well, this is what happens to Git, **conflict arises when two developers change the same line of code**. Git fails to understand which line to keep and which one to delete. So it asks you to resolve the conflict.

## Undo Changes

### Managing Changes

When you make a mistake in Word, what do you do? Ctrl+Z, right?

You cannot undo your mistakes in Git as you do in Word. But don't worry, Git gives you the following.

**Set of commands to undo your changes:**

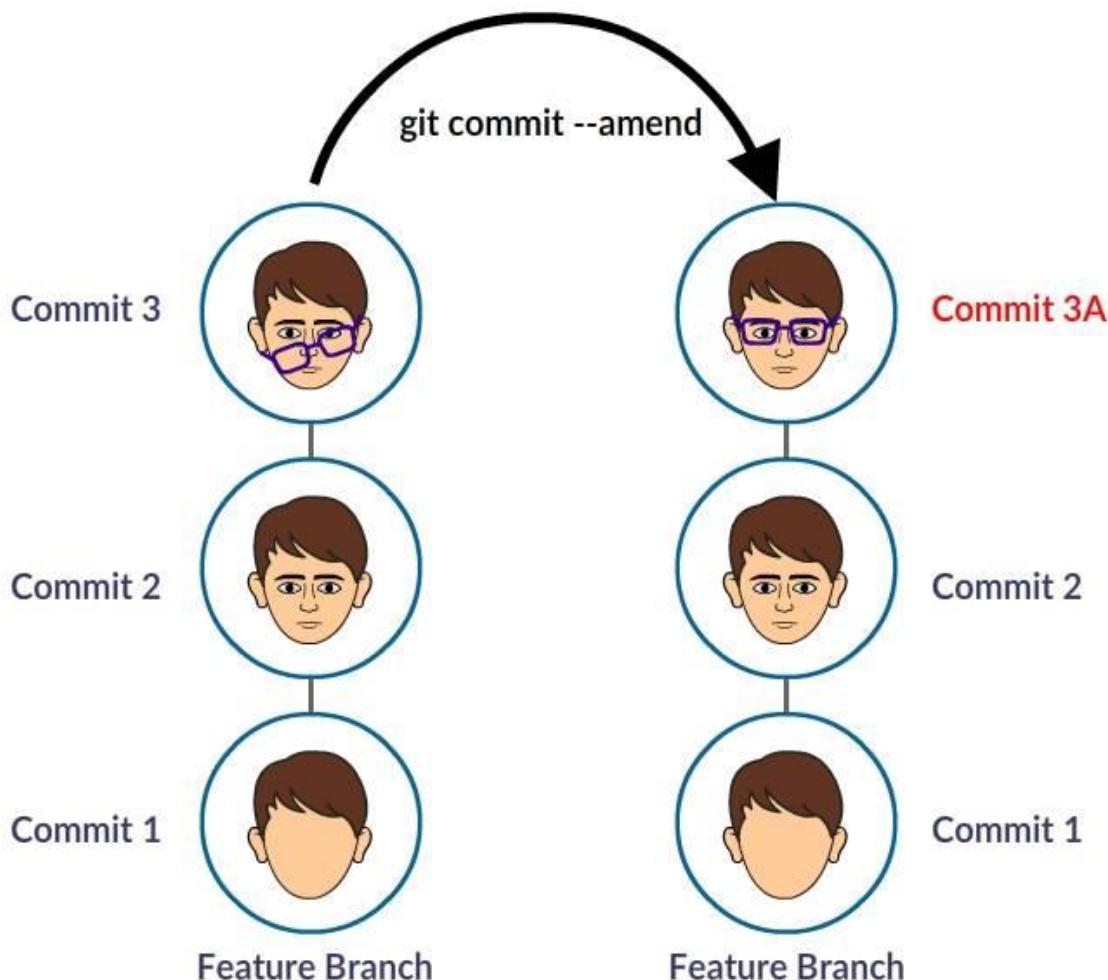
- amend
- checkout
- reset
- revert

**Set of commands to undo/remove your files:**

- rm
- clean

You will be learning more about each of them in this topic.

## Git Amend



**git commit --amend** command is used to fix your previous commit where you do not want to add a new commit. You can:

- Correct your typos in commit messages
- Add small changes in the file that you missed adding in the last commit

This command **re-writes history** by changing the commit-hash and the old commit will no longer be on the branch. Be cautious and avoid amending on public commits (commits you have already pushed to remote branch). **Amending a public commit might create merge conflicts** for other developers.

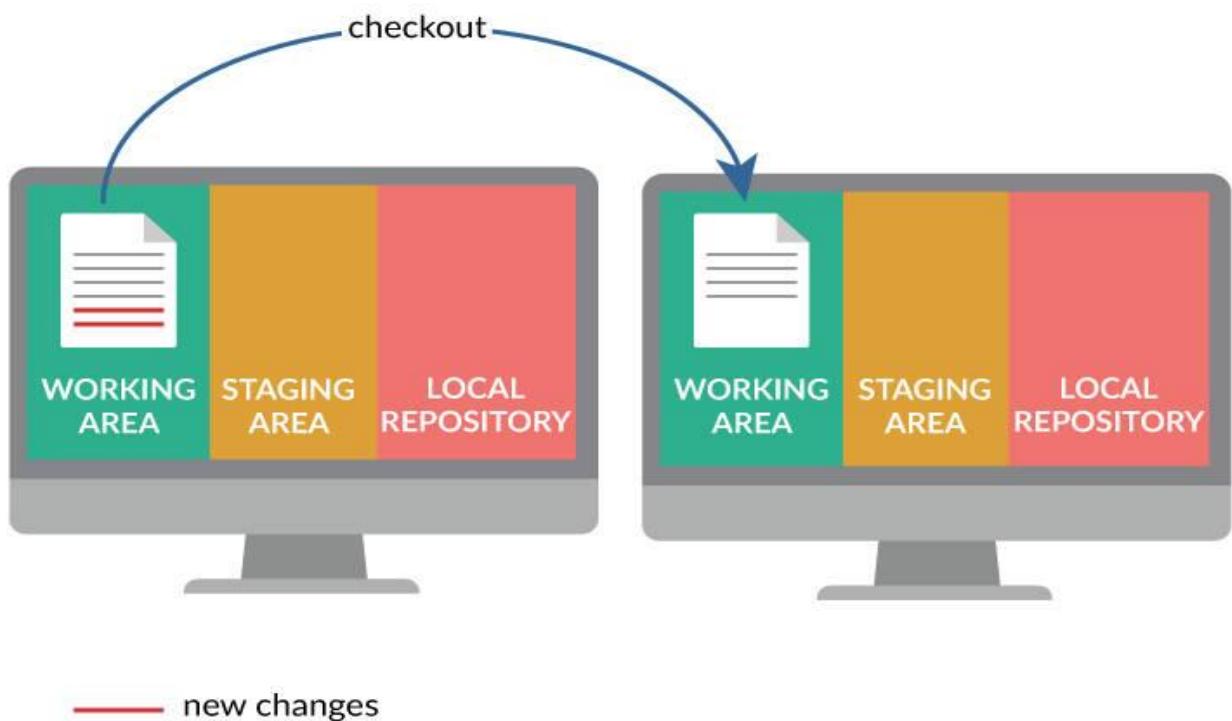
# Git Checkout

## Git Checkout

Checkout is used to **switch**. You can switch between branches, commits, and files. Here you will learn how to use this command to undo changes.

You can apply **checkout** command on:

- Working file
- Commit
- **Checkout Files**

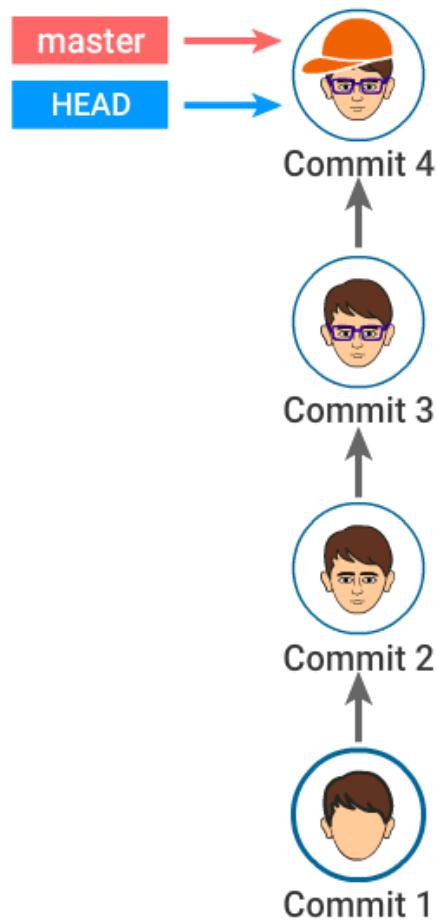


- *Oh no.. what am I doing? What is happening? I want to delete everything and start anew..*
- If this is what you feel, you can use **checkout** command to **discard all your changes from your working area** and **switch** to the last committed version of your file. This

command will help you restart your work as if nothing happened.

- Example: `git checkout filename`
- Note: This **command will only discard the changes in the working area**. If you have already staged the changes, you need to use `reset` command (as discussed later).

## Checkout Commits



*Why is the app behaving like this? Which commit has the bug?*

While debugging the application, if this thought crossed your mind, then you can use `git checkout` command to **switch to a specific commit**.

For example: `git checkout commitSHA`

- `commitSHA` is the commit hash to identify it uniquely. You can find this using `git log`
- This command **creates a detached head**, meaning, this will **give you a temporary branch** to work and debug.
- Avoid creating any new commits here, as this is a temporary branch

## Git Reset

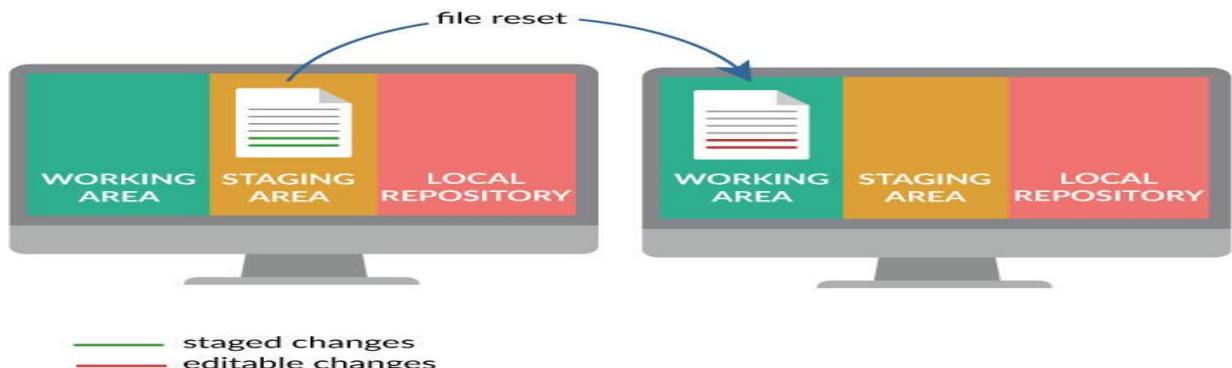
### Git Reset

**Reset**, as the name says, is used to **re-set** your work to a specific point in time (a commit). It is a powerful and versatile command, using which you can even **undo the changes that you have already committed**.

Imagine your **Git history as a timeline**, then you can quickly jump to a specific time in the past and reset your work as if nothing happened.

Next, you will learn how to **apply reset command on**:

- staged files
- commits
- **Reset Files**



- If you have staged your changes and forgot to add something, you can reset the file, so the **file is moved from staging to working area** where you can make the required changes.
- For example: `git reset filename`

## Reset on Commits: Types

You tracked a bug in your project which is due to a rogue commit (commit 3 in the gif).

You can move to the previous commit where the app is working fine (commit 2 in the gif). Did you wonder what will happen to the changes in those two commits that you are skipping? You can use one of the following **three flags**, which decides in which stage the changes should move:

- **--soft**: This **moves your commit changes into staging area** and does not affect your current working area.
- **--hard**: This **deletes all the commit changes. Be cautious with this flag**. You might lose your changes as this flag resets both staging area and working directory to match the <commit>.
- **--mixed**: This is the default operating mode, where your commit changes are moved to working area.

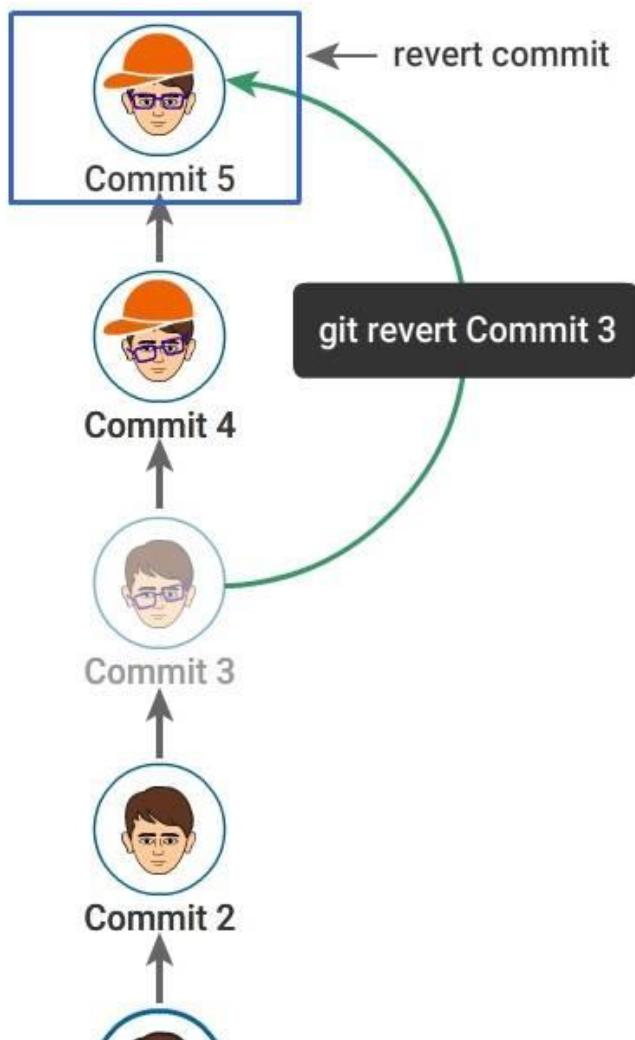
## Resetting Branch Commits: Example

Example: `git reset commitSHA`

- This executes `git reset --mixed commitSHA` by default, moving all the changes to the working directory
- You can find SHA code of your commit (where you want to move) using `git log`
- `reset` command adds a new commit, re-writing your history
- **Avoid** reset command, **if the branch** you want to reset **is public** (in remote), and somebody else is building a new feature on top of yours. Instead, you can use `revert`.

# Git Revert

## Git Revert

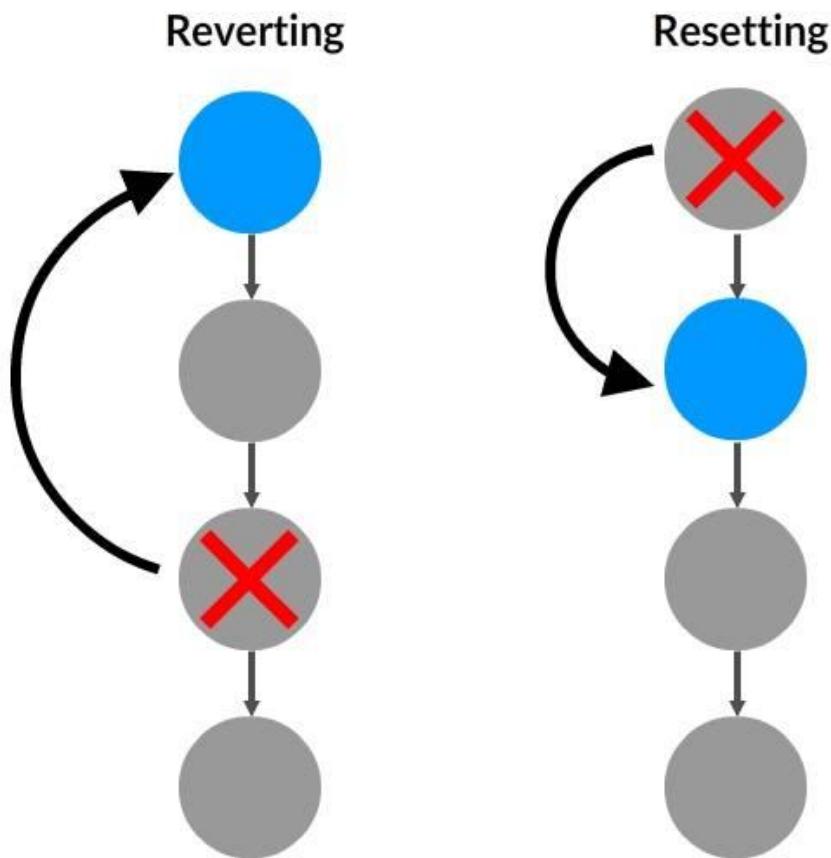


You tracked a bug in your project which was due to a rogue commit.

`git revert` is another command to undo changes from an old commit, similar to reset. However, **git revert inverses the changes from that old commit and creates a new revert commit**, instead of deleting the old commit.

This helps prevent Git from losing history.

# Revert vs. Reset Commits



While both reset and revert can undo your commit changes, the following are a few differences.

Revert	Reset
Does not delete commits and preserves history	Deletes commit and changes history

If other developers are using your branch to build new features, prefer `revert` to undo changes on that branch as it might prevent any conflicts

Prefer `reset` if the branch is only in local

## Delete Files

# Delete Files

Just like changes (in a file), you might also want to undo/remove files. Depending on its status, whether Git is tracking the file or not, you can use the following commands to remove them:

- `git rm`
- `git clean`

## Git Rm

**git rm** is used to delete any tracked file from your repository. Files from both the staging area and the working directory can be removed using the same.

These changes will not persist until a new commit is added, which in turn creates a new commit history.

You can get back your deleted files using `git reset` and `git checkout`.

## Restore Deleted Files

*How to restore a deleted file ?*

- First, find the commit ID where the file was deleted: `git rev-list -n 1 HEAD -- filename`
- Then checkout to that commit ID to get back the file `git checkout deletingcommitid^ -- filename`

## Git Clean

**git clean** command undoes files from your repo. However, it stands unique from other undo operations like checkout, reset and revert, as it **primarily focuses on untracked files**.

- `git clean` is undoable
- Git clean makes hard file deletion possible, similar to the Unix `rm` command.
- It is good to execute `git clean -n` command to perform a dry run, which helps to know the list of files to be removed.

# Cheat Sheet

- `git revert <commit>`: Creates a new commit that undoes all changes made in the commit and applies it to the current branch
- `git reset <file>`: Moves file changes from staging area to working directory.
- `git reset <commit>`: Moves current branch's HEAD tip to the old <commit>. All the changes in the commits (that you skipped to move to the old commit) are moved to the working area.
- `git commit --amend`: Adds staged changes to the last commit and allows for editing the old commit message
- `git rm --cached(file_name)`: Untracks the current file
- `git checkout <commit>`: Switches the HEAD to the provided commit

## Run Through

## Golden Rules of Git

- Create a new repository for every new project
- Create a new branch for every new feature
- Let branches be temporary, that is, delete the merged branch
- Write a good descriptive commit message, where:
  - The first line should not be more than 75 characters
  - `Fix`, `Feature`, or `Update` is present at the start of your commit message
- Use Pull Request (PR) to merge your code with master.

## Course Summary

In this course, we have learned about Git. We have understood the following concepts

- Basic commands to add and commit files
- Pushing and Pulling from Remote Repo
- Fixing merge conflicts
- Creating Branches
- Concept of Rebase
- Viewing commit history and making changes to the previous commits

# Interstellar Git

## Course Overview

In this course, you will learn how to:

- Move commits between branches
- Save changes temporarily
- Add a subproject
- Find the author of a line
- Find the commit which introduced a bug
- Retrieve deleted commits
- Automate using the script

By the end of this course, you will be able to manage your projects more efficiently and handle any scenarios using these advanced commands in Git.

Let's get started!

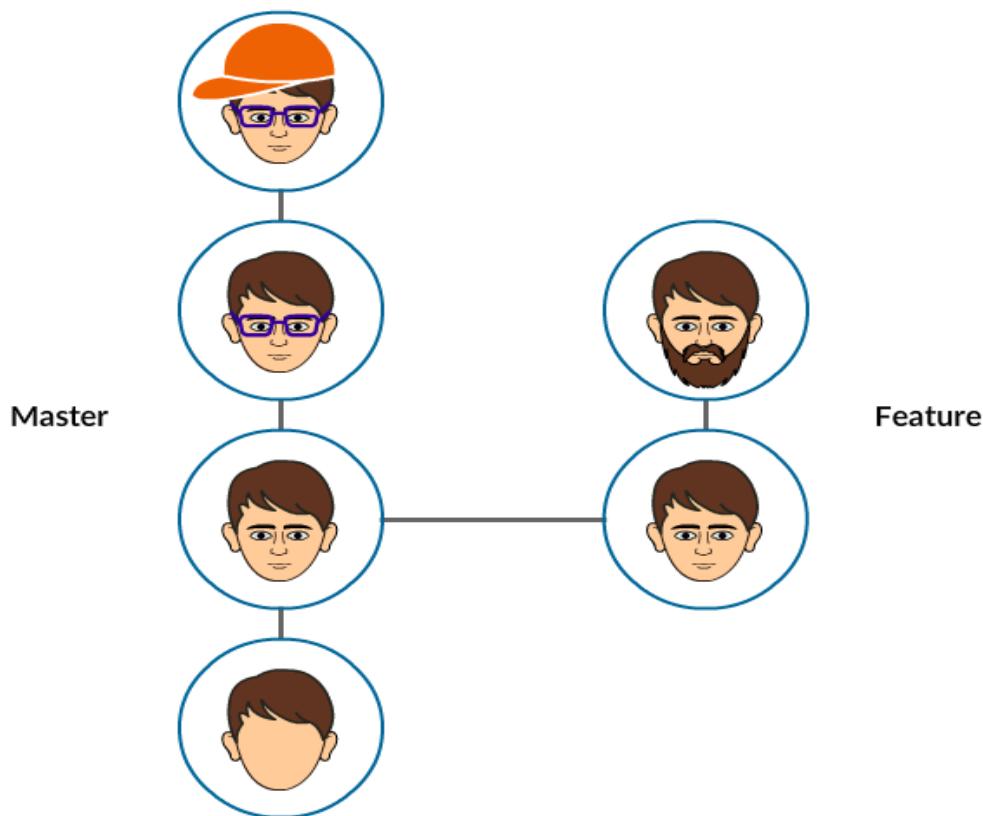
## Moving Commits

*Oh no, this commit/fix belongs to feature-A. I committed the changes in the feature-B branch.*

**How will you get your changes from feature-B to feature-A branch?**

It's simple! **Use Cherry-Pick!**

# What is Cherry-Pick?



With **Cherry-Pick**, you can **copy the selected commits from one branch (which contains the fixes) and append them to a different branch.**

For example, `git cherry-pick commitSHA`

As shown in the GIF, the **feature** branch needs the **third commit from the master branch to complete the feature**. So it cherry-picks the specific commit and adds it at the end of the feature branch as a new commit.

## Production Bug

You are in the middle of an important feature upgrade. Suddenly, all hell breaks loose as production crashes. You have to give a hotfix ASAP!

How would you switch between the two tasks? The feature upgrade cannot go along with the hotfix. And, pulling your hair out is not a solution!

You can use `stash` commands in such scenarios.

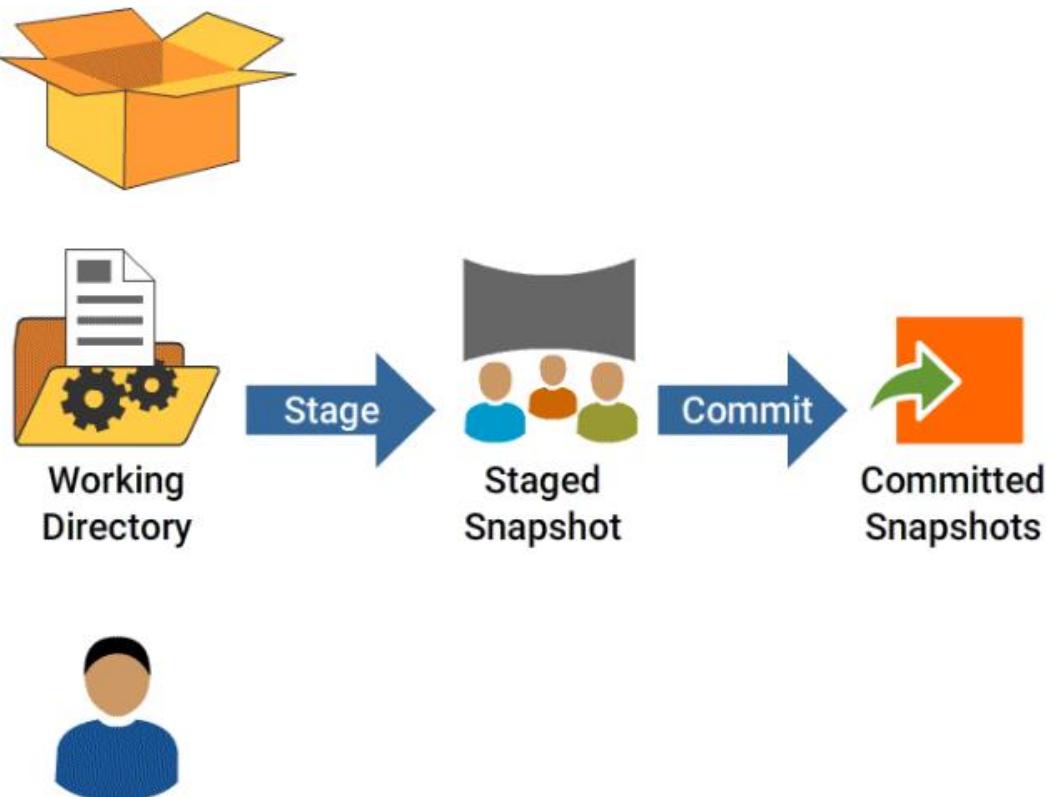
## What is Git Stash?

**Git stash** temporarily saves the changes you have made in your working directory so that you can switch your focus to something else. In this case, switching from feature branch to the hotfix branch to fix a production bug.

Later, when production is stable, you can re-apply your uncommitted stashed changes.

### Example

- `git stash`: Saves your current changes
- `git stash pop`: Gets your changes from `stashed list` and applies in the working area



**Git stash** temporarily saves the changes you have made in your working directory so that you can switch your focus to something else. In this case, switching from feature branch to the hotfix branch to fix a production bug.

Later, when production is stable, you can re-apply your uncommitted stashed changes.

## Example

- `git stash`: Saves your current changes
- `git stash pop`: Gets your changes from `stashed list` and applies in the working area

## When to Use Git Stash?

- Before checking out to a different branch (Why?)
- Before pulling the latest remote changes (Why?)
- Before merging and rebasing with another branch (Why?)

## Stash Tips

- Stashes are not transferred to the server when you push
- By default, Git will not stash changes made to untracked or ignored files
- Adding the `-u` option (or `--include-untracked`) tells `git stash` to stash your untracked files

## Tagging

- **Tag** allows you to capture a reference point in your project history, such as release versions
- An *annotated tag* contains additional information such as name, message, and email of the person who created the tag
- A *lightweight tag* points to just a commit hash

## Cheat Sheet

- `git stash`: Saves all the modified files temporarily
- `git stash list`: Lists all the stashed sets

- `git stash apply`: Applies the latest stashed content
- `git stash pop`: Applies the latest stashed set and drops it
- `git tag`: Adds a tag to a commit
- `git tag -a -m`: Adds an annotated tag with a message
- `git push tag name`: Pushes tag to the remote repo

## Using Third-Party Library

Consider you are working on a project that creates a man from a boy. Instead of writing the code from scratch, you found a **third-party library**, called **mustache repo**, which can add the required feature to your project.

The possible challenges are:

1. How will you add the third-party library in your repo?
2. How will you get further updates (of the third-party library) in the future?

## Git Submodule Introduction

**Git submodule** will allow you to add a **vendor library** to your project and get their future updates instantly.

To add a Git submodule use:

- **Syntax:** `git submodule add <URL of vendor library>`
- **Example:** `git submodule add https://github.com/doctrine/some-library.git`

## Benefits of Submodule

Following are few of the most popular **benefits** that **submodule** offers:

- You can re-use other's work instead of doing everything from scratch.
- If many projects of yours are using the same feature, then you can keep that feature separately and access it as a submodule within all your projects.

- **Customize vendor's libraries** to your needs without affecting the library itself and easily share the customizations **with other developers**.
- **Maintain separate commits** for your main project and the vendor's libraries.

## Git Submodule - Cheat Sheet

- `git submodule add <URL>`: Adds a submodule to the project.
- `git submodule status`: View status (working, staging, or indexed files) of all the submodules.
- `git submodule update`: Updates submodules after switching branches.
- `git submodule update --init`: After cloning a new repo, if you need to add submodules to it from .gitmodules file, use this command.
- `git submodule update --init --recursive`: If the submodules inside a newly cloned repo are nested, then use this.
- `git submodule update --remote`: Pulls all changes in the submodules.

## Git Blame

### Find the Accused

Consider there are five boys. They go on a trip in search of monsters. One night, they get stuck in an old palace, and they decide to stay there for the night.

The next morning, one of the boys was found shot dead. There is a gun beside him. How to **find the killer**?

Yes! You are right! Just by **checking the fingerprint** on the gun!

Your team has collaboratively written the code. In a particular file, a buggy line was introduced, which affects the entire working of the software.

Using Git, how to find:

- Who introduced that line?
- In which commit the buggy line was introduced?
- When was it introduced?

# Introduction to Git Blame

`git blame` helps you in such circumstances.

**Syntax:** `git blame <filename>`

**Example:** `git blame index.html`

## Git Blame - Cheat Sheet

`git blame <options> <filename>`

**Options:**

- `-L` to filter based on line numbers
- `-s` to suppress the author's name and timestamp from the output
- `-e` to show the author's email instead of the author's name
- `-f` to show the filename in the original commit
- `-n` to show the line number in the original commit

**Example:** To find the author of lines 5 to 7 in the file Readme.md, one can use:

`git blame -L 5,7 Readme.md`

# Git Bisect

## Identify the Overweighing Matchbox

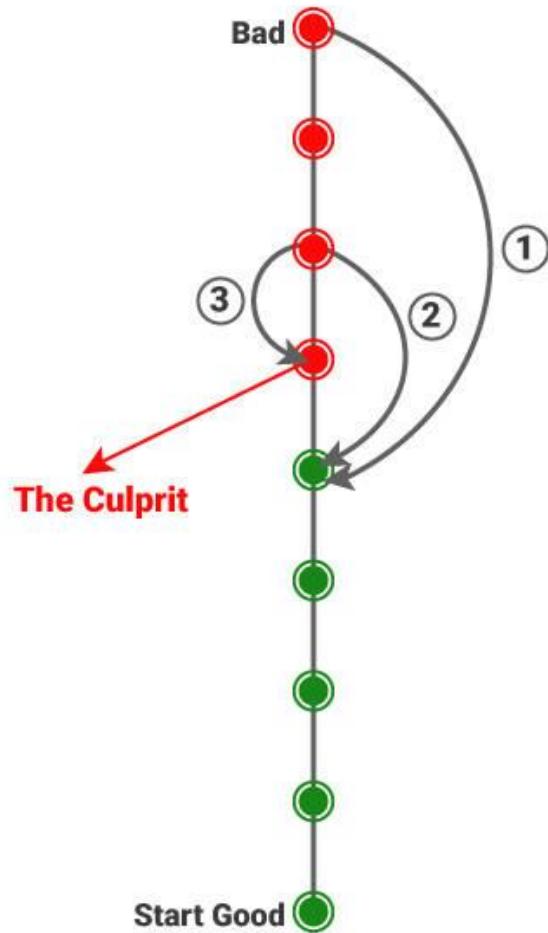
Consider a scenario where there are 100 matchboxes which weigh 10 grams each. It is found that one matchbox weighs 12 grams. Your task is to find that matchbox in the least number of attempts and time. You are provided with a balancing machine with two plates.

Now, what would you do?

If you perform a linear search, then the number of attempts will be huge. Hence you can perform a binary search, find the wrongly packed box and replace it with a new one.

This is what Git Bisect does!

# Why Git Bisect?



Whenever you find buggy behavior or an unknown error, you need to find the commit that introduced it.

In that case, you may not know the exact file name. This is where **Git Bisect** comes into the picture.

Git Bisect will:

1. Perform a binary search in the commits
2. Allow us to check it manually
3. Allow us to declare its status as good or bad

When this process is repeated, the commit which introduced the bug will be found.

# Git Bisect Tips

- You can use `git bisect log` to find the flow of Git Bisect, that is, to see what has been done so far.
- If you find something wrong, then you can use `git bisect reset`.

## Git Bisect - Cheat Sheet

### Examples:

- `git bisect start` - To start Git Bisect
- `git bisect good` - To declare that the commit is good
- `git bisect bad` - To declare that the commit has an error

# Git Hooks

## Automating Tasks

Consider you just returned home after a long day at work. You are exhausted. On entering your home, you find the `fan and lights turned on automatically`. Wouldn't you be happy?

You joined a project where all the developers check the following **before pushing their code**:

- Commit **messages** should follow project standards
- Every file should have an empty line at the end
- **Test cases** should pass
- **Linting errors** should be checked

Similar to the following simulation, wouldn't it be exciting and comfortable if all these checks are automated just like your home?

This is where **Git Hooks** comes into the picture!

## Introduction - Git Hooks

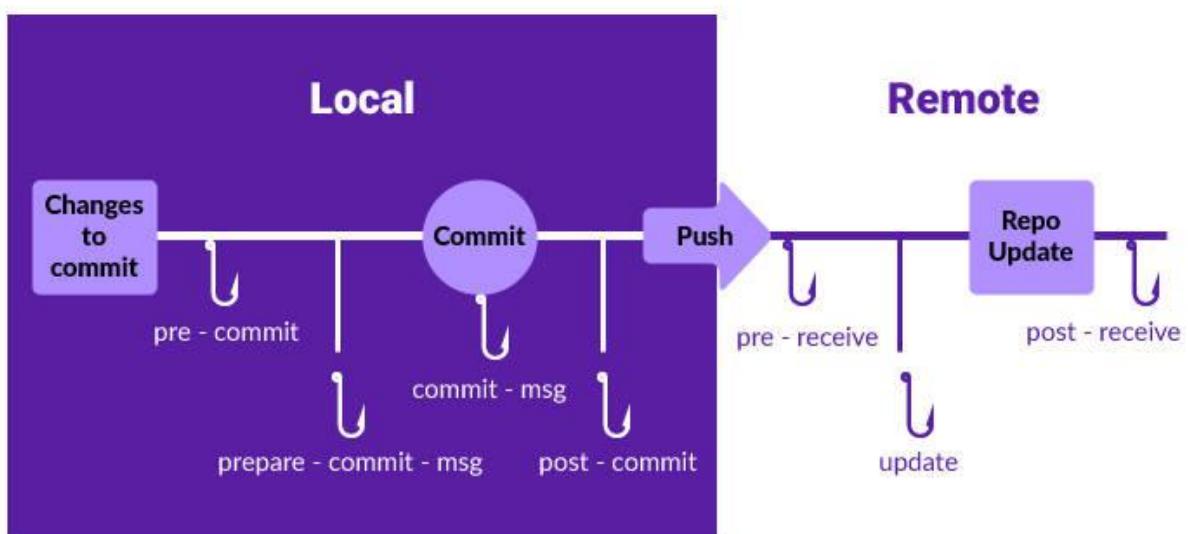
Git Hooks are **shell scripts** that get triggered when we perform a specific action in Git.

- You can set up a Git Hook that prevents you from committing when there is a problem
- Automating frequent tasks will reduce errors. For instance, if we want to check the trailing spaces in our commit message, we can

write hooks script, which gets triggered automatically whenever we commit.

- You can find all your hooks in the path `<project-dir>/ .git/hooks/`

## Few Git Hooks Files



The above image shows a quick summary of **Git Hooks** that you can trigger when you make any changes in your local machine or the remote repository.

## Grouping Git Hooks

Based on the Git operations, any one of the following **git hooks** will be triggered. These hooks can be divided into the following two groups.

**CLIENT-SIDE**

## Operations

like **commiting, merging, rebasing, squashing** and **checkout** will trigger the client-side hooks.

### Examples:

- Committing workflow hooks
  - pre-commit
  - prepare-commit-msg
  - commit-msg
  - post-commit
- Email workflow hooks
  - applypatch-msg
  - pre-applypatch
  - post-applypatch
- Other client hooks
  - pre-rebase
  - post-rewrite
  - post-checkout
  - post-merge

## SERVER-SIDE

Network operations such as receiving pushed commits will trigger the server-side hooks.

### Examples:

- update
- pre-receive
- post-receive

## Points to be Noted

1. The hooks exist as simple text files in your `.git/hooks` directory.
2. You can write hooks in any **scripting language** like Python, Ruby, and so on.
3. The Script **filename should match the hooks' names**.  
Example: If you want to send an email after each new commit, you need to write a POST-COMMIT script in a new file called `post-commit` or rename the corresponding sample file.
4. Hooks are **not within version control**. You can, however, use **Grunt** and **Symlinks** to manage hooks.

# Git Reflog

## Why Git Reflog?

Let us imagine that you checked out to your new feature branch, made some changes, and committed it.

Unfortunately, you force deleted that branch using `git branch -D branch_name` before merging it with the master.

Wouldn't it be nice if you had the **superpower to travel back in time** and retrieve those deleted commits?

## Learn Git Reflog

- Git reflog has the superpower to **track your head**.
- The difference between log and reflog is that:
  - `git log` will track every commit that you make and record it as a snapshot at a particular time, whereas `git reflog` will keep track of commits that are made as well as the commits that are discarded.
- This is provided in a rolling buffer for 30 days
- The `git reflog` command will list down the logs whenever the HEAD changes like the branch was created, cloned, checked-out, renamed, or any commits made on the branch.

## Example

```
$ git reflog
fc3d0e7 (HEAD -> branchA) HEAD@{0}: commit: Deleted
d01d36d (HEAD -> branchA, master) HEAD@{0}: checkout: moving from master to branchA
d01d36d (HEAD -> branchA, master) HEAD@{1}: commit: File Created
8c2419a HEAD@{2}: reset: moving to HEAD
```

Note that, `HEAD@{1}` shows you moved to branchA from the master branch.

## Interstellar Git Summary

You started the Journey by exploring the **Prodigious Git** course and then completed the **Interstellar Git**.

Let us have a quick recap of the concepts covered in this course:

1. `git cherry-pick` (Moving commits between branches)
2. `git stash` (Saving changes temporarily)

3. `git submodules` (Adding a subproject)
4. `git Blame` (Finding the author of a line)
5. `git Bisect` (Finding the commit which introduced a bug)
6. `git reflog` (Retrieving deleted commits)
7. `git hooks` (Automating things using script)

Hope you enjoyed the course!

*Git Checkout Bye!*

# Continuous Integration

You can develop good software regardless of the technology you are working on by following certain fundamental practices:

- **Plan** for code changes
- **Perform** the code changes, then compile and test
- **Check** the results
- **Act** on the results

Adapt **Continuous Integration** to help you with seamless software development.

## What is Continuous Integration?

*The earlier you detect defects, the cheaper they are to fix - David Farley*

**Continuous Integration (CI)** is a **software development practice**, where developers will **commit** (integrate) their code changes to a shared repository frequently. Such commits must trigger automated builds and tests, **enabling quick verification of the changes at an early stage of the development cycle**, rather than waiting to detect bugs after the code is completely developed.

## Why Continuous Integration?

*Continuous Integration does not get rid of bugs, but it does make it easier to find and remove.*

When developers cultivate the habit of **integrating** their **code changes regularly**:

- **Changes** will be typically **small**
- **Errors** can be **detected quickly**
- **Pointing** out the **change** that introduced an **error** can be **done quickly**

# How to Make a Process Continuous?

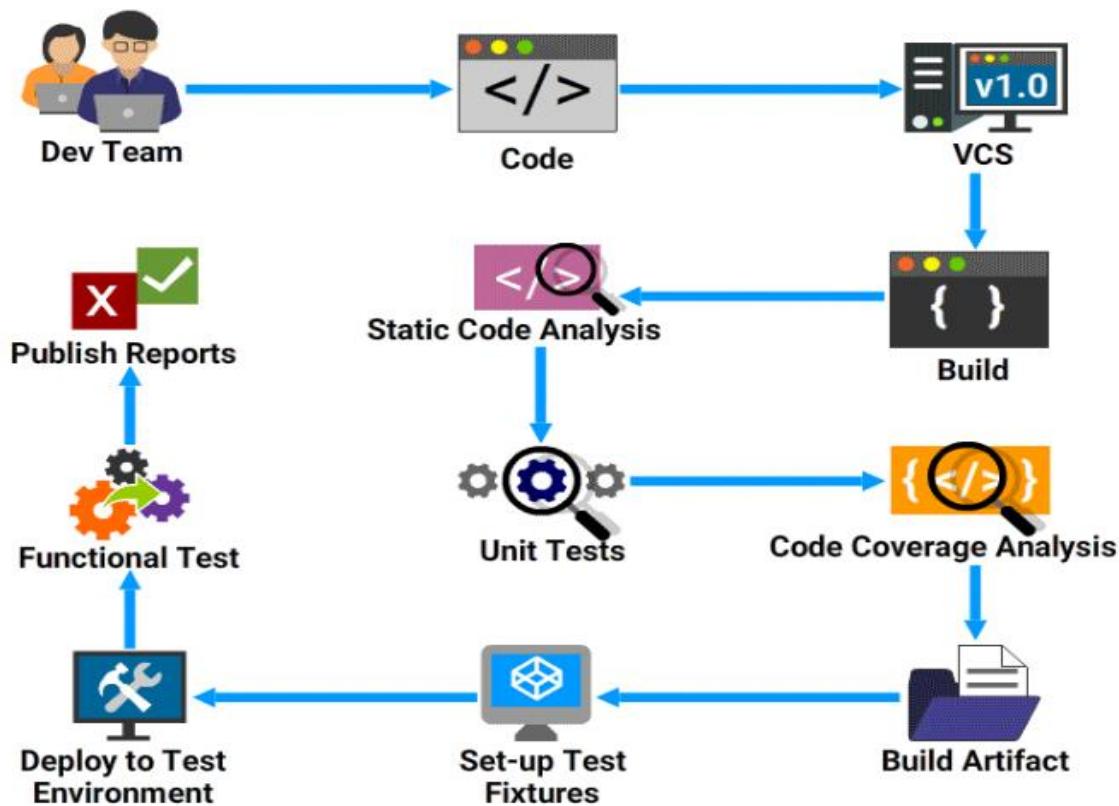
As a first step towards adopting Continuous Integration, identify and make a software development process continuous.

## Continuous Integration - Services

Continuous Integration includes the following:

- *Source Code Control*
- *Code Compile*
- *Integrate Database Changes*
- *Run Tests*
- *Code Inspection*

## Continuous Integration - Workflow



## Version Control

# Source Code Control

**Source code control** is the heart of Continuous Integration. Source code must be managed using a **Version Control System** (VCS).

The different types are:

- *Local*
- *Centralized*
- *Distributed*

## Source Code Control - Types

### **Local Version Control System:**

- Codebase is maintained locally

### **Centralized Version Control System** (CVCS):

- Code resides on a *central base*
- Developers:
  - create *work branches*
  - perform *changes* on the work branch
  - *publish* changes to the *central base*
- SVN is an example of CVCS

### **Distributed Version Control System** (DVCS):

- Developers:
  - *clone the central base* into their local machines
  - create *work branch* from the *local base*
  - perform the *changes* in the work branch
  - *merge* the changes in the *work branch* to *local base*
  - *synchronize* the *local base* with the *central base*
- Git is an example of DVCS

# Version Control Branch Types

All the currently existing version control tools support **branches or codeline**.

**Branch:** An **independent line of work** that **stems out** from a **central codebase**.

Types:

- The **Main branch** is known as **trunk, mainline or master**
- The **Release branch** is used for bug fixes post user release
- The **Work Branch** is used by developers for development changes

Every branch should have an owner.

The owner must define the branch policy (when a code should be checked in).

## Mainline Branch

- The **Mainline** branch must be *stable always* so that the code is in *ready to deploy* state.
- Ready to deploy implies that the **code** has *successfully passed tests* like integration, regression, and so on.
- Code in mainline is **deployed to the user** or production environments.

## Release Branch

- **Change** done on the release branch must *flow back to the mainline*.
- The Release branch should *never receive a change from the mainline*.
- It **must be closed** after a **release** from the mainline.

## Work Branch

- Work or development branch is where the developer *compiles the code, integrates and runs tests.*
- *Stable changes* in work branches are *published to the mainline.*

What if your team is *implementing multiple changes in parallel on the work branch?*

## How to Handle Parallel Changes?

Publishing to the mainline is easy when only one change is implemented at a time.

Consider a scenario when 2 developers are working on the same work branch:

- Developers A and B are working on different changes simultaneously.
- Developer A has partially completed the changes and merges it with the work branch.
- Developer B completes the changes fully and merges it with the work branch.
- Then, developer B publishes the changes to the mainline.
- Accidentally, B ends up publishing the partially done changes of A to the mainline.
- Publishing an incomplete code to the mainline is a violation of the mainline branch policy.
- If an issue arises later, it will be hard to say whose change introduced it.

*What can be done now?*

## How to Handle Parallel Changes?

*Team collaboration* plays a crucial role in such scenarios.

- Either developer B must wait until both changes have been completed and then publish to the mainline.

- Or prioritize the changes by publishing the top priority change first to the mainline. Then, merge the other change with local version control.

**Avoid multiple parallel changes in the same work branch.**

## Merging Work Branch Conflicts

*How to handle conflicts during code merge?*

- Developer A is using a library variable in the code changes.
- Developer B completely removes references to the library and publishes it to the mainline.

Now, developer A needs to discover the conflicting change before proceeding further.

To detect conflicts at an early stage:

- *Merge down the code from the work branch to your development region as often as possible*
- *Check-in changes to the work branch frequently*

What if other teams are working on separate work branches that end up publishing to the mainline?

## Merging Mainline Conflicts

Consider the scenario with two teams:

- Each team has their own work branch.
- Each team will publish their changes to the mainline independently.
- There is a possibility of a new change on the mainline, which may conflict with the other team's code.

Pick up the library example discussed previously (replacing developers with teams):

- Team A removes the use of the library and publishes the change to the mainline.
- Team B continues to use the library.

*How to handle the merge issue in this case?*

## Merging Mainline Conflicts Contd.

- *Merge* down changes from the **mainline** to the **work branch**, ideally, *every day*.
- The **team that discovers the conflict** is **responsible for sorting** out the conflict.
- **Publish** changes from the work branch to the mainline *regularly*.

The team that checks in the changes first is the winner.

## Release Branch

*A high priority bug got detected post user-release. What has to be done now?*

- *Create a release branch* from the mainline based on the time it was released.
- *Fix the bug* on the release branch.
- *Merge the changes* from the **release branch with the mainline**.

## Version Control - Summary

Now that you have learnt briefly about version control, how does CI fit in here?

- Merging or *committing changes frequently is a top practice*, that *lays as a backbone of CI*.
- *Version control helps* a developer *to plan and execute changes seamlessly*.

## Uses of Branching

Branching helps in **parallel development**. Work can be **done** on two or more work streams **simultaneously**, without affecting one another.

- **Physical:** Branches created for files, subsystems, and components.
- **Functional:** Branches created for features, logical changes, bugfixes, and enhancements.

- ***Environmental***: Branches created for build and runtime platforms such as compilers, libraries, hardware, operating systems, and so on.
- ***Organizational***: Branches created for activities/tasks, subprojects, roles, and groups.
- ***Procedural***: Branches created to support policies, processes, and states.

## Branching Techniques

The next few cards will help you understand the various branching techniques that teams will adapt based on their need.

- ***Branch by Feature***
- ***Branch by Release***
- ***Branch by Team***
- ***Branch by Abstraction***

All the 3 types of branches (`mainline`, `work` and `release`) discussed in the ***version control*** section must be used at appropriate places in each of these techniques.

### Branch by Feature

- *Created to work simultaneously on features* or user stories.
- *Mainline is kept* in a *releasable* state.
- A *feature* is *developed* on a *separate branch*.
- It must be *merged into the mainline after it is tested*.

### Branch by Feature - Practices

- *Merge mainline* onto every branch *daily*.
- *Branches must be short-lived* (few days).
- The number of *active branches at any time* must *depend on the number of features* that are developed.
- *New branch should not be created until the previous branch is merged with mainline*.

- **Refactorings** (changing code without changing the behavior) must be *merged immediately* to minimize conflicts.

## Branch by Release

- **Code is developed on the mainline.**
- **Branch** is *created* when a **feature is complete** and ready for release.
- Release **testing and validation** are **done** on this branch.
- **Only bug fixes are done** on this branch and merged back with the mainline.
- **No new branches** must be *created off the release branch.*
- Branches for later releases must always be created off the mainline.

## Branch by Team

- **Used** in a large **team that works on functionally independent areas.**
- The mainline must be in a releasable state.
- **Branch** is *created* for **every team**.
- **Merged** with the mainline only **when the branch is stable.**
- **Merge done to the mainline** from any given branch must be *published to every other branch.*

## Branch by Team - Practices

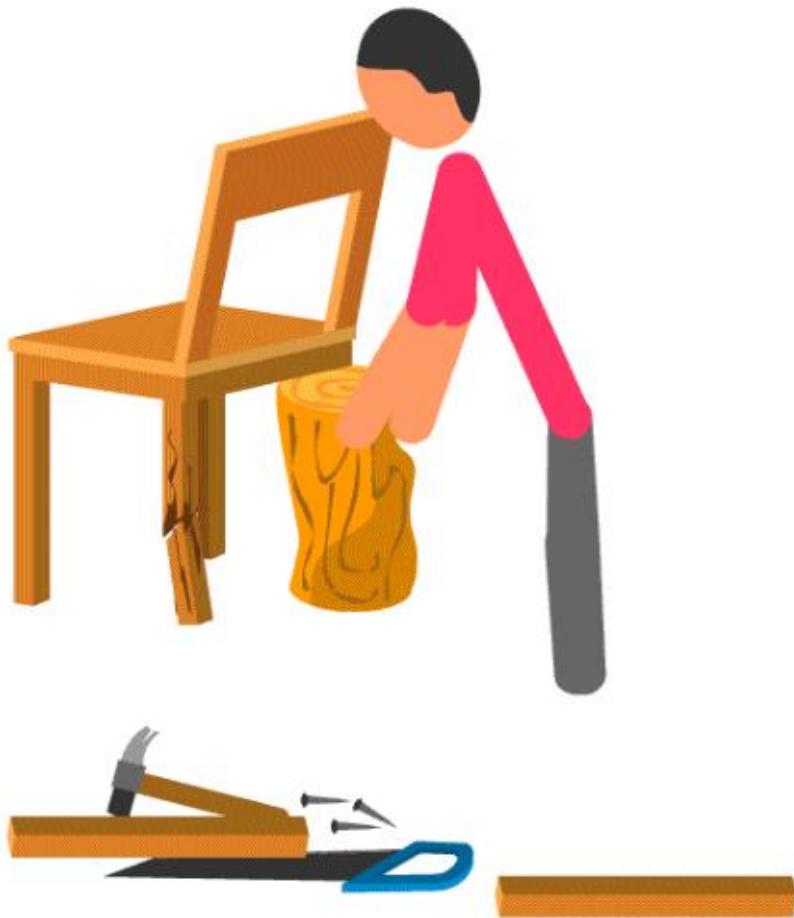
- **Create small teams** with each team working on its own branch.
- **Publish changes from the mainline** to every branch **daily**.
- **Run unit and acceptance tests** for every check-in done to the branch.
- **Run all tests** (including integration tests) on the mainline every time a branch is merged with it.
- **On discovering a bug** after merging with the mainline:
  - **Perform changes** in the **team branch** and stabilize before merge (or)

- *Create a new branch for bug fixes.*

## Branch by Abstraction

- Trunk (mainline) based development.
- Mainline is always stable and ready for deployment.
- Used for making large-scale changes incrementally.

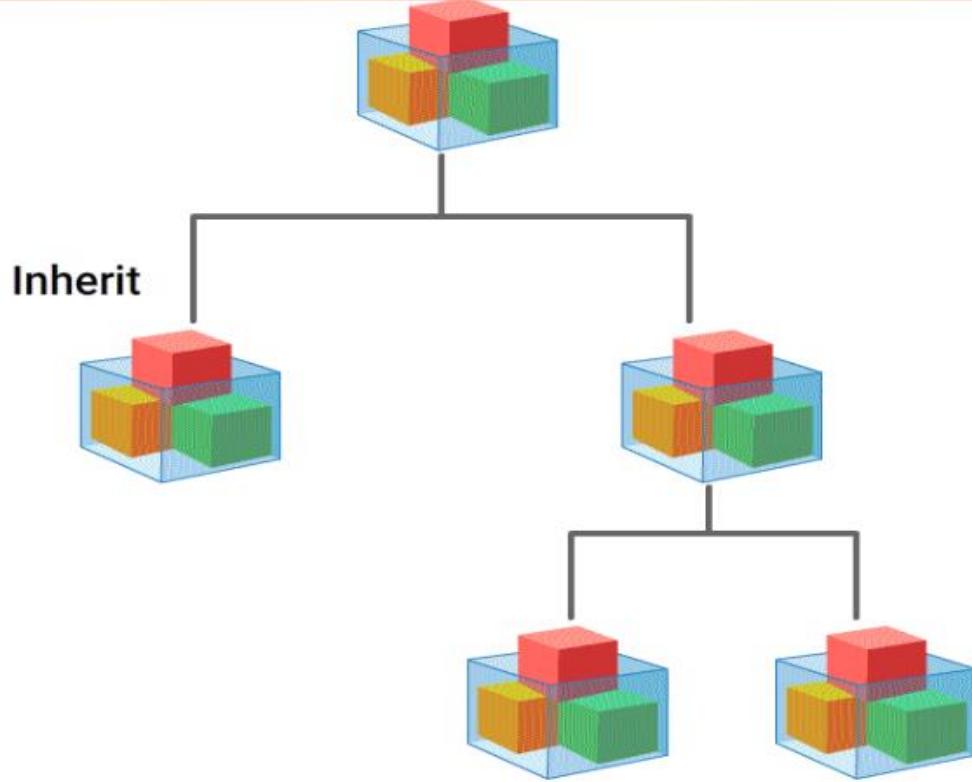
## How it Works?



- *Create an abstraction layer* around the code to be changed.
- *Change the application to use the abstraction layer.*
- The application interacts with the code through the layer.
- *Create the new code.*
- *Reroute application interaction* to the new code through the layer.
- Once changes are complete and stable, *remove the layer.*

# Stream-Based Version Control

---



- Developers *develop code in their own workspaces*.
- *Changes* are *promoted* to streams *once* they are *ready*.
- *Branch is replaced by streams*.
- A *change* applied to a stream will be *automatically inherited by the downstream streams*.
- Merge problems addressed by this automatic inheritance.

## Stream-Based Version Control Contd.

*Stream-based version control is helpful* in the following scenarios:

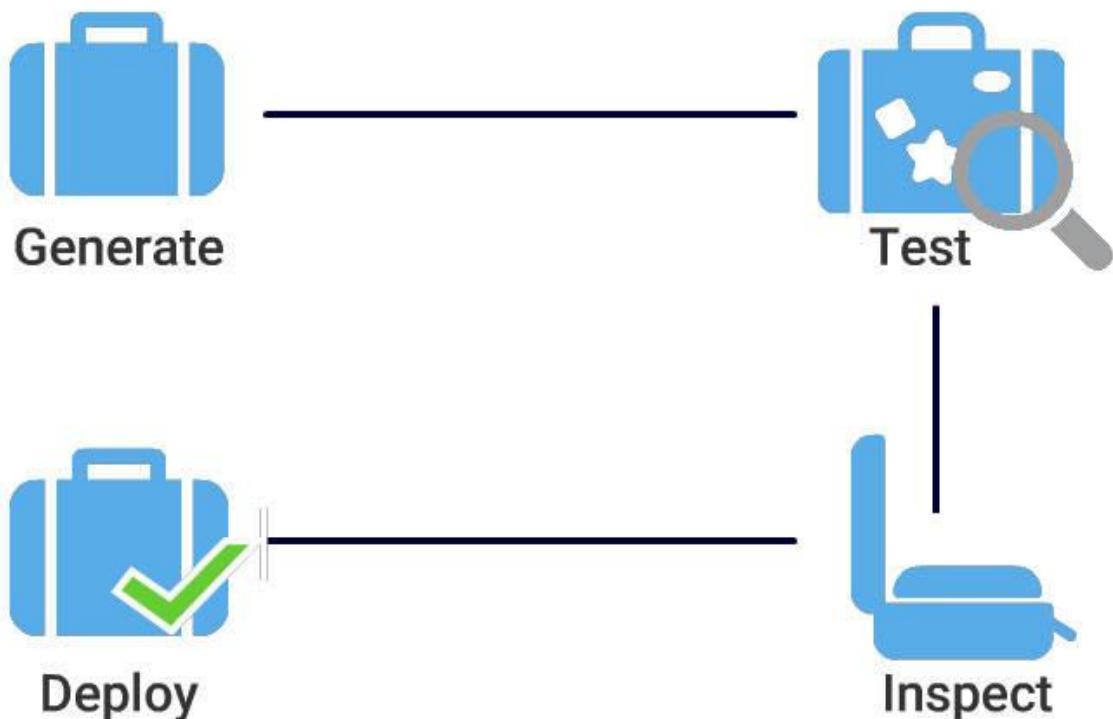
- *Applying a bugfix* to several versions of the application.
- *Adding a new version of a third-party library* to the codebase.

How it is done:

- *Promote the changes* in your stream *to the common ancestor* of all the streams that need the change.
- *IBM Clearcase* and *AccuRev* are popular *stream-based version control* systems.

## Build Process

### What is a Build?



## Build

Activities performed to **generate**, **test**, **inspect** and **deploy** software.

## Central Repository

- All the assets required to build like **library**, **DLLs**, **configuration files** are centralized into the central repository.

To maintain the central repository effectively:

- ***Use a consistent directory structure*** in the central repository, which enables you to retrieve only the required files.

- Create folders for design, requirement, implementation, testing, and so on.
- **Fetch** the required files from the *implementation folder* for *integration build*.

## Build Scripts

As soon as a *change is moved into version control*, it is a recommended CI practice to *build immediately*.

- *Create build scripts* (using tools like Maven, Ant)
- *Execute build scripts* from an IDE or command line or CI tool
- *Build scripts should not be dependent on the IDE*, which means it must be executable from the CI tool if used

Use CI tools to *automate* the *trigger* and *execute* the *build scripts* on detecting a change.

## Build Types

Build is of 3 types, namely:

- Private Build
- Integration Build
- Release Build

## Integration Build

*Integrates* changes committed on the *work branch* with the *mainline*.

- Ideal to *run integration build on a dedicated machine*.
- *Code compilation, unit test, component test, system* and *performance test* along with *inspections* are executed as part of this build.

## Release Build

- *Deploys* code to the production or *end-user*.
- *Includes* extensive *load and performance tests* along with *user acceptance tests*.
- *Triggered by code changes* on the *mainline* or release branch.

## Build Mechanism

- *On-Demand*: Build initiated manually.
- *Scheduled*: Build triggered based on time.
- *Poll for Changes*: Build runs after a change is detected by a CI tool.
- *Event-Driven*: Build triggered by version control tool based on a change.

## Build for any Environment

In order to enable software deployment to different environments, say, for testing purposes, the code must be deployed to various environments.

- *Build scripts should remain the same*.
- *Configuration files* (like .properties or .include) *helps in differentiating the environments*.

## Run Fast Builds

*Rapid feedback* is a *key factor of CI*.

If the build runs for a longer duration, it may delay the reporting and fixing of errors. So *stage the builds* (split builds logically).

Create separate build scripts to:

- *Fetch changes from version control and compile*.
- *Run unit tests*.
- *Run automated processes* like integrating the database changes, component tests or system tests, and code inspection.

*Do incremental builds* (compile only the components changed)

when *compilation takes a longer duration*. But, use it judiciously.

# Integrate Database Changes

Any time a database is changed, such as:

- New objects are created
- Existing objects are altered
- Objects are dropped or removed

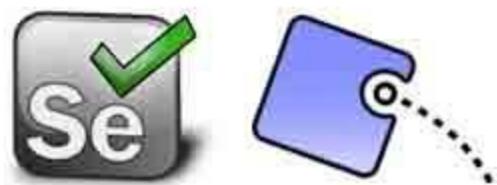
build the components that are using the database.

As a recommended practice:

- *Incorporate database integration as part of the build*
- *Use a local sandbox* to test database changes
- *Share the artifacts*, like scripts to create, modify, delete, schema, and so on, using a central repository.

## Test, Inspection and Metrics

### Test Types



You might have come across this quote: *A code that cannot be tested is flawed.*

As an important CI practice, you need to **execute** :

- **Unit test** (using tools like JUnit, NUnit, PHPUnit, as appropriate).

# Test Types

- **Integration or Component test:** To verify how certain changes interact with the rest of the system (using tools like JUnit, NUnit, DBUnit).
- **System test:** To completely test a software (using tools like JWebunit).
- **Functional test:** To test the software functionality from a user perspective (tools like Selenium are used).

**Automate the tests using the tools provided.**

# Testing Strategy

- **Categorize** your *tests* (unit, system, component).
- **Create test scripts** and include them as part of the build.
- **Ensure** proper *test code coverage*.
- **Schedule the build based on the test category.** Different intervals need to be planned for slow-running tests.
- **Prioritize** and run *faster tests*.

# Inspection Activities

**Code review** plays a crucial role in maintaining the overall quality of a codebase. This must be part of the build.

- **Leverage automated inspectors** like JavaNCSS or CCMetrics to *identify the piece of the code that is highly complex*.

The complexity of the code is determined by **Cyclomatic Complexity Number (CCN)**.

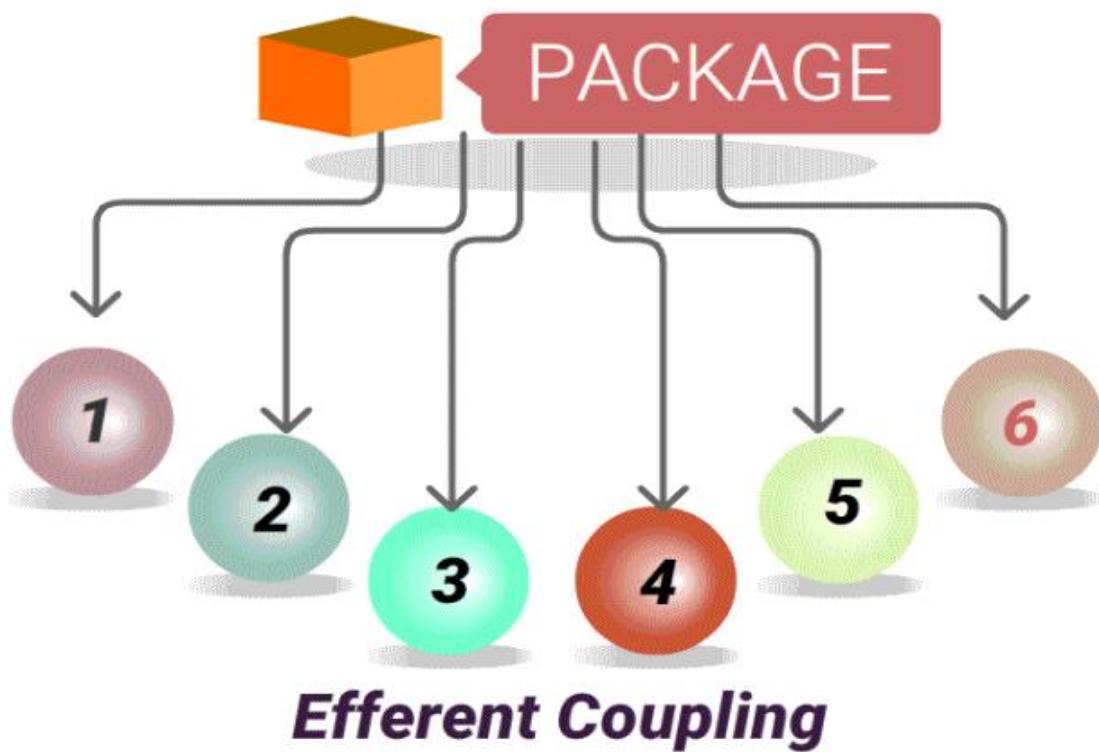
It is the measure of:

- Number of linearly independent paths (or)
- Number of ways present to traverse a piece of code.

It determines the minimum number of test inputs required to check all the ways of executing a program (done using a control flow graph).

- *Report coding standards violation* (using PMD, FxCop, and so on).
- *Identify the amount of duplicate code* (using tools like Simian, CPD).
- *Assess code coverage*, identify the percentage of code executed on running a test (using tools like NCover, Cobertura, Clover).
- *Determine if a package is highly dependent on other packages*, measured using *Afferent and Efferent coupling* .

## Afferent and Efferent Coupling



*Coupling* is a *measure of dependencies*.

### Afferent Coupling

- *Who depends on you*
- Measure of how many other packages use a specific package.
- Incoming dependencies

## Efferent Coupling

- *Who do you depend on*
- Measure of how many different packages are used by a specific package
- Outgoing dependencies

## Build Metrics

**Build metrics** plays a crucial role in helping **reduce the build duration**. The following are some important build metrics:

- **Compilation time**: *Time taken to compile the software*, compares with the past compile times.
- **Number of Source Lines of Code (SLOC)**: *System's size or size of what* has to be *compiled*.
- **Number and types of inspections**: *Number of different inspections performed*.
- **Build Repair Rate**: *Time taken to repair a build failure*.
- **Test execution time**: *Time taken to perform testing* at each level like unit, component, and system.
- **Inspection time**: *Time taken to perform the inspections*.
- **Deployment time**: *Time taken to deploy the software*.
- **Database rebuild time**: *Time taken to rebuild the database*.

## Analyze Metrics

*Capture and analyze the metrics* to **determine** what **improvement** must be done to **reduce the build duration**.

The following are such improvement outcomes:

- *Improve test performance*
- *Streamline integration builds*
- *Improve inspection performance*
- *Distribute integration builds*

## Broken Build

If any *activity* that is performed as *part of the build fails*, then *build* is considered **broken**.

- *Do not commit changes* of a broken build to the work branch or mainline.
- *Fix broken builds* immediately.
- *Do not check out changes* related to a broken build.

## Summary

You have learnt about:

- Build
- Activities that are part of a build
- Build types and mechanisms
- Build practices like run fast builds and so on
- Tests
- Metrics

## Risk Mitigation and Best Practices

### Risks Addressed by CI

*Following CI* effectively and efficiently helps *reduce* the following *risks* :

- *Lack of deployable software*
- *Late discovery of defects*
- *Low quality software*
- *Lack of visibility on project health*

## Best Practices

To adapt CI in your project, the following are the important *practices to be followed diligently* :

- *Using a version control system*
- *Committing code frequently* to the work branch and mainline

- *Automating developer tests*
- *Running private builds*
- *Ensuring all the tests and inspections run successfully*
- *Refraining from committing a broken build*
- *Fixing broken build on priority*
- *Avoiding checking out a code related to broken build*

## Continuous Integration Tools

### Criteria to Evaluate CI Tools

*A craftsman who wishes to practice his craft well must sharpen his tools* - Chinese Proverb

Selecting a CI tool is highly dependent on the environment, size, and functionality of the project.

Evaluate the required tool, based on its:

- **Functionality** (Essential and Extendable)
- **Compatibility with your environment** (supports current build configuration and existing version control, compiles the code language)
- **Reliability**
- **Longevity** (choose a tool with a healthy user base and established development group)
- **Usability** (easier to configure and use)

## Build Tool: Essential Functionality

- Code compilation
- Component packaging
- Program execution
- File manipulation

# Build Tool: Extended Functionality

- Development test execution
- Integration with Version control tool
- Document generation
- Deployment functionality
- Code quality analysis
- Extensibility to integrate plugins
- Multi-platform build support

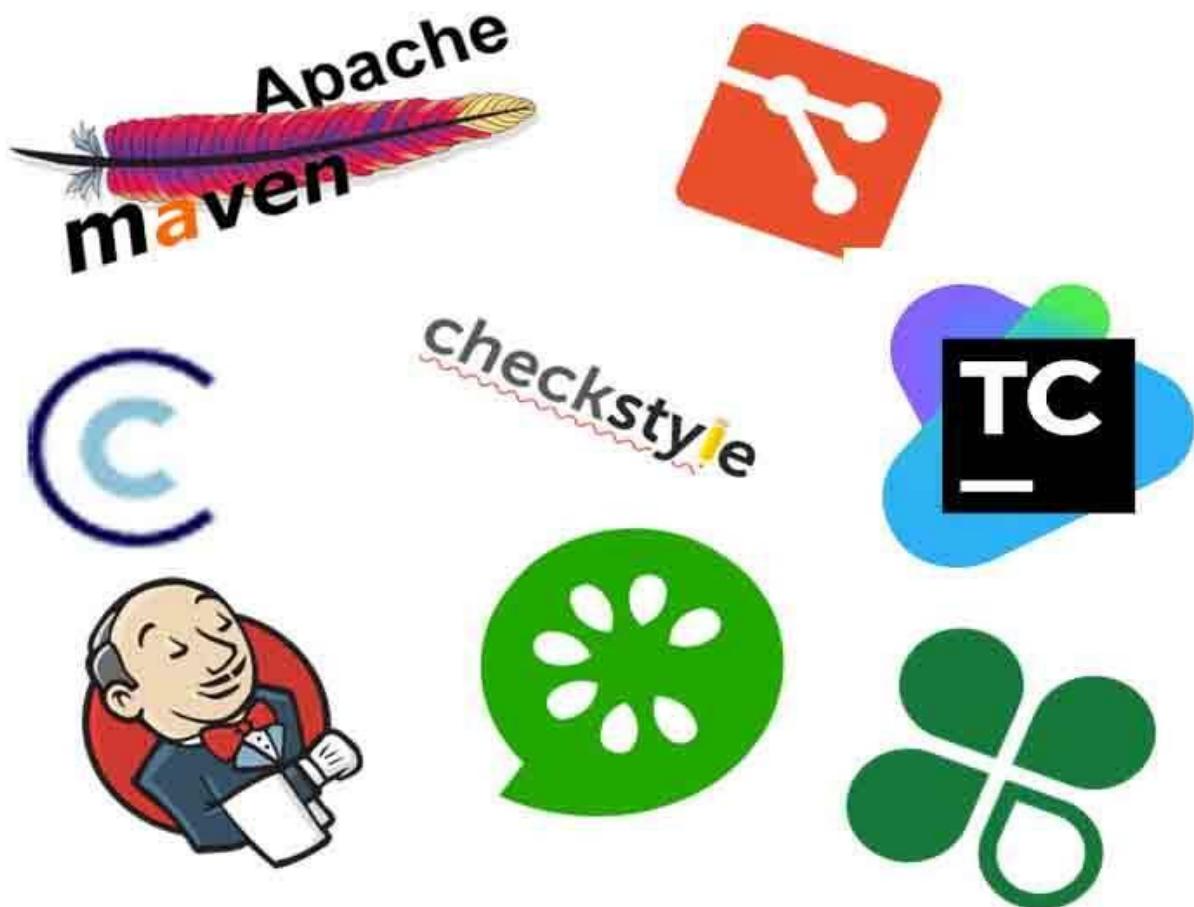
# Build Scheduler: Essential Functionality

- Build Execution
- Version control integration
- Build tool integration
- Feedback
- Build labeling

# Build Scheduler: Extended Functionality

- Inter-project dependencies
- User Interface
- Artifact publication

# Continuous Integration Tools



**Version Control** - GitHub, Subversion

**Java Build** - Ant, Maven

**.Net Build** - NAnt, MSBuild

**Java build scheduler** - Jenkins, CruiseControl

**Static code analysis** - SonarQube, Checkstyle, PMD

**Code Coverage** - JCov, Clover, Serenity, Cobertura

**Unit Testing** - NUnit, JUnit

**Functional Testing** - Cucumber, Selenium

**Artifact Repository** - Nexus, Artifactory

## Continuous Integration – Conclusion

To summarize, we have learnt the following in this course:

- *CI and its practices*
- *Use and features of version control*
- *Build Processes*
- *Metrics associated with build*
- *Ways to evaluate and select a CI tool*
- *Examples of CI tools*

# Maven - Coalescing Pipeline

## Introduction to Maven

**Maven** is a very popular **build automation tool** used for **Java** projects.

In this course, you will learn:

- Core concepts of Maven
- Significance of POM file
- Build lifecycle of Maven
- Phases of Maven
- Goals of Maven
- Usage of various Maven commands
- Maven plugin types - Junit Surefire
- Integration of Maven with Jenkins and JMeter.

## What is Maven?



Maven is a comprehensive **build management tool** that helps developers in performing the following tasks related to any project.

- Compiling the source code.
- Running the test.
- Packaging the code into jar files.
- Performing related activities such as,

**Create websites > Upload build result > Generate reports**

- Deploying the final product.

## Why Maven?

Often while working on a **Java application**, you might be handling numerous jar files.

***You will have to handle proper dependency, version inclusions, building, publishing as well as deploying the app.***

One of the efficient and hassle-free ways to deal with these activities is to use a **automated build tool** like **Maven**.

## Downloading Maven

You can set up **Maven** on any OS environment. Follow the subsequent steps for **Maven on Linux / Mac**:

- **Download** apache-maven latest version
- **Extract** *apache-maven-3.5.0-bin.tar.gz* file to, say, */usr/local/apache-maven*

## Setup Maven environment

- Open terminal and set path as

```
export MAVEN_HOME=/usr/local/apache-maven/apache-maven-3.5.0  
export MAVEN=$MAVEN_HOME/bin
```

- Add maven bin directory path to system path

```
export PATH=$MAVEN:$PATH
```

- Verify Maven Installation using

```
$ mvn –version or $mvn --version or $mvn -v
```

## Maven Core Concepts

Maven works around the following core concepts:

- **Project Object Model (POM)** is the XML representation of the project where all the dependencies and configuration details are stored. POM plays a major role in ensuring that all the project resource references are maintained.
- **Build Life Cycles, Phases, Goals** - Maven build process is composed of many build life cycles. Each life cycle has one or more phases. Each phase has one or many associated goals.

## Maven Core Concepts (continued)

- **Dependencies and Repositories** - Dependencies are external JAR files required for the project to work. Maven downloads these dependencies into the local, central or remote repository.
- **Build Plugin** - Adding plugins to the POM file allows us to add new custom actions to be done during the build process.
- **Build Profile** - Projects can be built differently by using different build profiles

## POM File

Each project has a corresponding POM file that is located in the root directory.

Whenever a goal (a specific build task) has to be executed, Maven looks for the configuration details in POM.

- Project details can be represented in the form of XML file called **pom.xml**.
- It is the fundamental unit that contains information about the project.
- It holds all the resources required for a build such as source code location, test source, dependency details such as external or internal dependency etc.

## Unique Identifier for POM file

**POM.xml** takes **minimal coordinate attributes** as inputs for the project as **groupId:artifactId:version** (alias **GAV**).

POM stores the information such as the **location of the source code** and **records any external dependencies**. It describes what needs to be built as part of the project.

### Sample POM file:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.fresco.play</groupId>
  <artifactId>first-mvn-project</artifactId>
  <version>1.0-SNAPSHOT</version>
</project>
```

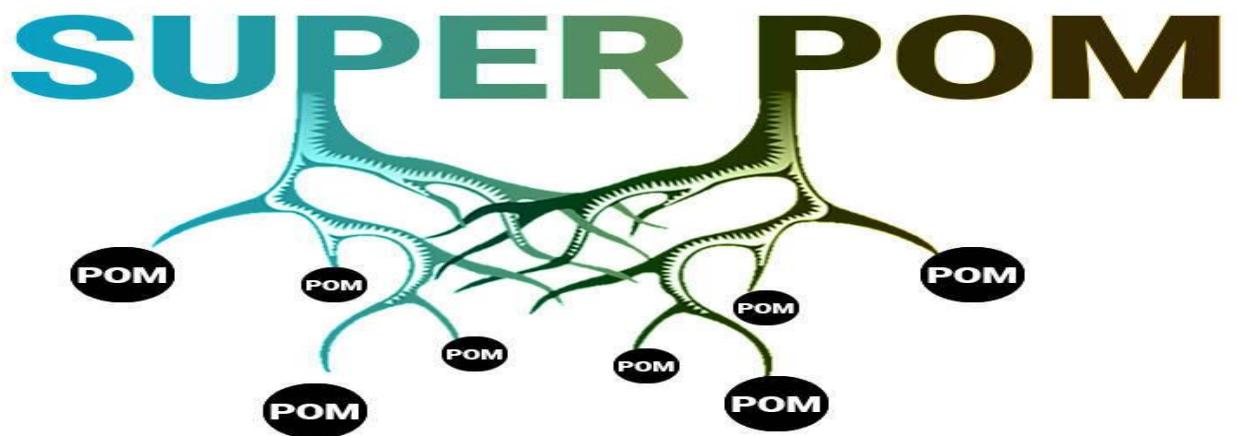
## POM.xml with Additional Inputs

In addition to minimal inputs, some of the additional elements can also be added such as **Packaging** (jar, war etc.), maven project **name**, **url**, **dependencies**, **scope** (compile, provided, runtime, test, system) etc.

### Sample pom.xml file with additional elements:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.fresco.play</groupId>
  <artifactId>first-maven-project</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>first-maven-project</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

## Super POM



POMs can also inherit properties from a single POM called **Super POM**.

Super POM is a **view-only POM** to see the entire attributes of all the dependencies spanning across multiple POMs. *By default, all POMs will inherit Super POM settings.* However, these settings can be overridden if necessary.

Quite often, it is possible that a project might have many sub-projects. In such cases, each subproject will have its own **pom.xml** file, while, the project might have a separate parent POM.

*This way the project can be built together with sub-projects or the sub-projects can be built independently.*

## Super POM View

Now, let us try to generate a view of Super POM by using a command **mvn help:effective-pom**. For this:

- Check the current directory by using **pwd** and change the directory to where pom.xml file exists.

**Example:** `cd first-maven-project`.

- You can do an **ls** to ensure that you are in the same directory as that of pom.xml.
- Run the command **mvn help:effective-pom** to view **Super POM**.

## Setting Dependencies

One of the key benefits of using Maven is its **effective dependency management mechanism**.

As the project complexity grows, the number of Jar files and other external APIs that might be needed for the project to consume/interact might grow. This might lead to difficulties in managing such dependencies along with appropriate versions.

With Maven, all dependencies of your project are maintained in a **single pom.xml file**. Maven takes care of downloading these dependencies into the local repositories and makes them available for the project.

## Sample Dependency Element

Following is a sample dependency element **junit** added in pom.xml.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.2</version>
    <scope>test</scope>
</dependency>
```

## Maven Repositories

*All the dependencies are held and maintained in a repository in Maven. These repositories are the directories of packaged jar files.*

Whenever Maven searches for dependency addition, it looks at local repository followed by central repository, then the remote repository.

- A **local repository** refers to the repository on a developer's computer.
- A **central repository** is the one that Maven community provides.
- A **remote repository** could be one on the web server from where the dependencies can be downloaded.

## Get a New Dependency

There are many repositories that hold some of the commonly used Maven dependencies. [search.maven.org](http://search.maven.org) is the recommended site for getting the needed ones.

Let us search for **csv file** maven dependency in [search.maven.org](http://search.maven.org). You could click on the needed version to get the dependency information as shown below. The same can be copied to the dependencies element in **pom.xml**.

```
<dependency>
  <groupId>com.opencsv</groupId>
  <artifactId>opencsv</artifactId>
  <version>3.9</version>
</dependency>
```

This dependency will be downloaded and kept at **MAVEN\_REPOSITORY\_ROOT=.m2/repository/ in Linux**

## External Dependencies

At times, the jar files to be added as project dependency might reside outside the Maven repository. These can also be added as an external dependency in **POM.xml** file as shown in the subsequent example.

```
<dependency>
  <groupId>mydependency</groupId>
  <artifactId>mydependency</artifactId>
  <scope>system</scope>
  <version>1.0</version>
  <systemPath>${basedir}\war\WEB-INF\lib\frescoplay-gems.jar</systemPath>
</dependency>
```

## Dependency Ranges

Till now, we were defining the version for dependency uniquely.

Let us see how we can specify dependency version for

- **as greater than Range**
- **in between Range**

# Dependencies Version as Greater than Range.

If user depends on a version of **JUnit** which is **higher than 3.8**.

- Dependency Range for JUnit will be specified as: **JUnit > 3.8**
- We can define using Exclusive quantifiers boundary, denoted as **(, ]**

## Sample Dependency View

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>(3.8,]</version>
  <scope>test</scope>
</dependency>
```

# Dependencies Version in Between Range

If user depends on any version of **JUnit**, which is greater than or equals to 3.8 but less than 4.0 .

- Dependency Range for Junit will be specified as: **JUnit 3.8 to JUnit 4.0**
- You can define it using Exclusive quantifies boundary denoted as **[, )**

## Sample Dependency View

```
<dependency>
  <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  <version>[3.8,4.0)</version>
  <scope>test</scope>
</dependency>
```

## Build Life Cycle

# Maven Build Life Cycle

Maven comes with a default life cycle. *The practice for building and distributing a project (artifact) is defined comprehensively in the build life cycle.*

- **Build life cycle** constitutes different **build phases**.
- Each **build phase** has its own **build goals**.
- A **goal** is a **unit of work** in the build process.

## Built-in Build Lifecycles

Maven has three built-in build lifecycles:

- **default** - supports project deployment (compiling and packaging)
- **clean** - supports project cleaning (removal of previous jar files, temp files, source files etc.)
- **site** - supports project's site documentation.

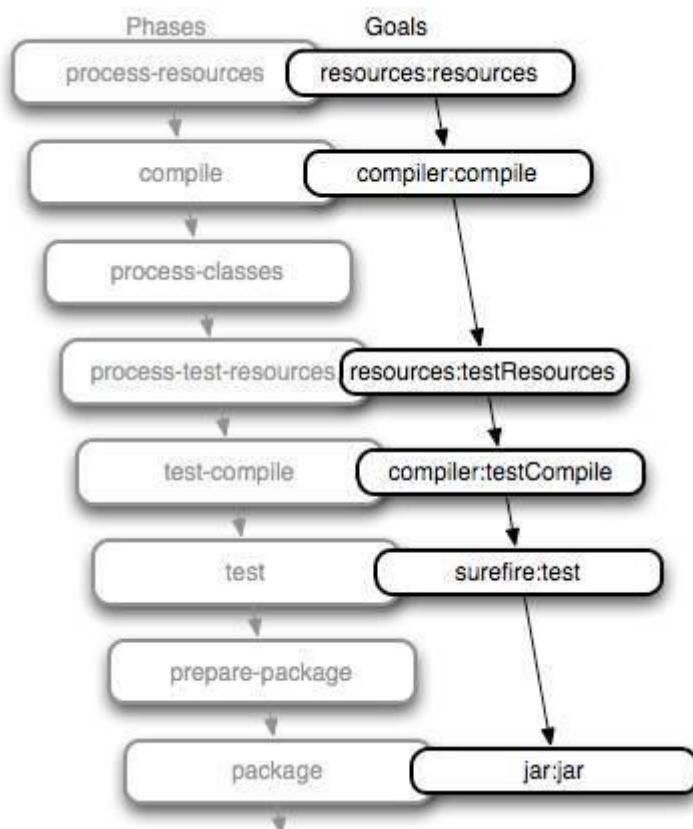
## Build Life Cycle Phases



Following are the default phases of each build life cycle:

- **validate** - checks if the project is correct and all information is available.
- **compile** - compiles the source code in binary artifacts.
- **test** - executes the tests.
- **package** - takes the compiled code and packages it as, a **war** or **jar** or an **ear** file.
- **integration-test** - takes the packaged result and executes additional tests required for packaging.
- **verify** - checks if the package is valid.
- **install** - installs the result of the package phase into the local Maven repository.
- **deploy** - deploys the package to a target, i.e. a remote repository.

## Build Goals



Some phases have goals bound to them by default. All relevant goals associated with a phase are executed during this process.

A **goal** is relevant for a phase if the Maven plug-in or the pom binds this goal to the corresponding life cycle phase.

**For example** - By running the command **compiler:compile**, Maven gets the goal of compiling application sources done by using the parameters specified in **POM.xml**.

## Maven Plugins



A Maven Plugin comprises of one or more goals. As referred earlier, a goal is a **unit of work**. It is to execute a specific task and perform certain operations for the project.

**Custom actions** can be included in the build process with the help of Maven Plugins.

*Goals can be executed independently or as part of a larger build.* The Goal is executed based on the information found in the POM of the project.

## First Maven Project

### Check Maven Installation

- Let us check Maven installation on Terminal by typing **mvn -version**. You should able to see Maven version, its home path, Java version, Java home path and few other default settings.
- To get Maven path, you may try **whereis mvn**.

### Create First Maven Project

- You will be prompted with series of options to select while generating these templates. Some of mandatory options to respond are:
  - Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 962: **hit enter**.
  - Choose org.apache.maven.archetypes:maven-archetype-quickstart version: `\*\* choose the version to include. Default is the latest version of Maven.
  - Define value for property 'groupId': **give a unique group id, say, com.fresco.play**.

### First maven - Contd

- Define value for property 'artifactId': \*\*give first-maven-project\*\*.
- Define value for property 'version' 1.0-SNAPSHOT: \*\*give `1.0-SNAPSHOT`\*\* Snapshot indicates that the project is in development.
- Define value for property 'package' com.fresco.play: \*\*provide the suggested name i.e., `com.fresco.play`\*\*.

- Now a summary of all the options keyed in as part of the project creation is shown. Confirm the same by giving **Y**.
- The build is successful with **BUILD SUCCESS** message.

# Verify Project Creation

- Go to the Maven project location and find your newly created project **first-maven-project**.
- The project structure would contain `src` and `pom.xml` files. This is a standard structure that Maven creates.
- `src` folder holds the `com.fresco.play.App.java` file and `com.fresco.play.AppTest.java` files, which are the source code and test code files.

*Project creation can be considered successful if you see this kind of a structure.*

## Project Source Code

- Let us add some code, say, the **Hello World** application code into `App.java` file.

```
public class App {  
    public static void main(String[] args) {  
        // Prints "Hello, World" to the terminal window.  
        System.out.println("Hello, World");  
    }  
}
```

- Let us change the directory to **first-maven-project** by executing `cd first-maven-project`.
- Do a `ls` and verify the current folder list. You should have `pom.xml` and `src`.

# Compile the Source Code

It is time to compile the code. Type **mvn compile**.

If the compilation is successful, **BUILD SUCCESS** message appears.

## JAR File Creation

Now, assuming you have finished coding your Java project and compiled the same, it is time to create a JAR file.

- The first step for creating JAR file is to ensure cleansing any previous build artifacts before packaging them as a Maven project. This can be done using **mvn clean** which removes all the previously generated files from ./target folder.
- **mvn package** would create a new package at ./target folder.

Alternately, we could combine cleaning and package creation as one step by typing **mvn clean package**. You would get a new JAR file created.

## Running Jar File

- To run the jar file, copy the jar build path and execute `java -cp /home/scrapbook/tutorial/test/target/test-1.0-SNAPSHOT.jar com.fresco.play.App` in terminal.
- It will print **Hello World!**

## Running the Test

To run the test phase of Maven life cycle, use **mvn test** instead of running a full build packaging.

## Environment Variables

Environment variables can be set and read in Maven.

- **To set the variable**, say, M2\_HOME path, type `export M2_HOME=/usr/share/maven`.

- **To verify the path**, type echo \$M2\_HOME. If /usr/share/maven is displayed as the path, then the environment variable is properly set.
- These variables can be read in <build> phase in maven **pom.xml** by pre-fixing the environment variable with env property name as shown.

## Setting Environment Variable

Sample plugins with an environment variable:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-antrun-plugin</artifactId>
      <version>1.1</version>
      <executions>
        <execution>
          <phase>validate</phase>
          <goals>
            <goal>run</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <tasks>
          <echo>Displaying environment variables</echo>
          <echo>[JAVA PATH] ${env.JAVA_HOME}</echo>
          <echo>[M2_HOME] ${env.M2_HOME}</echo>
        </tasks>
      </configuration>
    </execution>
  </executions>
```

```
</plugin>
</plugins>
</build>
```

## Program Execution with Maven

Now, to make your Java program work with Maven, incorporate exec-maven-plugin in pom.xml file.

This will eliminate the need for using jar with java path to run the source code. Java path gets configured in Maven itself by including exec-maven-plugin in pom.xml.

Once the pom file is ready with the exec maven plugin, source code can be executed using **mvn exec:java**.

*Sample pom file with maven-compile and exec-maven plugin after <Dependencies></Dependencies> tag:*

```
<build>
  <sourceDirectory>src</sourceDirectory>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.3</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>1.2.1</version>
```

```
<configuration>
    <mainClass>com.fresco.play.App</mainClass>
</configuration>
</plugin>
</plugins>
</build>
```

## Creating Web Archives

### Create WAR Project



It will create the collection of war plugins and package all to the classes, artifacts, dependencies and resources of the web application.

Let us try creating war file using maven, using our traditional approach,

- First create templates like we created for jar file. Type **mvn archetype:generate**
- Choose a number or apply filter (format: [groupID:]artifactId, case sensitive contains): **choose the type of war, here we are choosing. maven-archetype-quickstart**
- Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): **choose the archetype. org.apache.maven.archetypes:maven-archetype-quickstart version.**
- Choose **org.apache.maven.archetypes:maven-archetype-quickstart version: choose the latest version, 9.**
- Define value for property 'groupId': **give a unique group id, say, com.fresco.play**
- Define value for property 'artifactId': **give first-war-project**
- Define value for property 'version' 1.0-SNAPSHOT: **give 1.0-SNAPSHOT**
- Define value for property 'package' com.fresco.play: **provide the suggested name i.e., com.fresco.play**

Confirm by selection Y: **Y**

- The build is successful with **BUILD SUCCESS** message. It is time to create your own first maven project.

## Verify and Compile WAR Project

- **first-war-project** will be created with **pom.xml** file and **src** folder.
- Open **pom.xml** file and verify that packaging value is changed to war.
- In **src >main > web app** path is create.
- Change the folder path by entering **cd my-war-project**.
- To compile type **mvn compile**.

# Create WAR File

We can create and use war file in a different way.

- Using war type in the project package.
- Using `war:war` goal.

## Using war type in the project package.

- First, we need to compile the project, type `mvn compile`.
- Enter `mvn clean package`.
- It will generate the war file in the target folder.

## Using war:war goal to create war file.

- Enter `mvn compile war:war`. It will compile and generate war file together.
- It will create generate war in the target folder `target/second-war-project.war`

# Single Command to Create WAR Project

Create War file by passing all the dependencies value together.

- Enter `mvn archetype:generate -DgroupId=com.fresco.play -DartifactId=First-WebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false` in terminal.
- It will generate war file.

## Verify the war file:

- It should have `pom.xml` and `src` file.
- `src` file should have `main` file with `resources` and `webapp`.

## Create the war file:

- Change directory with `cd First-WebApp`
- Try yourself using `war:war` goal method and package method.

# Creating EAR Project



Try creating EAR file yourself with the following hints-

- Enter `archetype:generate` command.
- To filter ear type files enter `ear`.
- Select the type of `ear` file you want.
- Enter the latest version of EAR.
- Enter `groupId` as `fresco.play.com`.

-Type `artifactId` as `first-ear-project`.

- Enter `version: : 1.0-SNAPSHOT`.
- Enter `package` as `com.fresco.play`.
- Confirm project by selecting `Y`.

# Verify and Create EAR file

Verify the EAR project.

- `pom.xml` file should have a package as ear

Create EAR file.

- Change directory to the current project.
- Compile the project
- Create the ear file.

# Compile the Source Code



There are two types of plugins

- **Build Plugin** - Used while executing the build. They are included in `<build>` element of the POM.
- **Reporting Plugin** - Used while generating the site. They are included in `<Reporting>` element of the POM.

Some of the frequently used Maven Plugins:

- `clean` - To clear target after the build.
- `compiler` - To compile the Java source code.
- `jar` - To generate Java jar file.

- **war** - To generate Java war file.
- **surefire** - To run Junit test and generate a report.

## Plugin in POM.xml

Following is the sample code for compiling plugin in **pom.xml** used in one of the previous learning cards:

```
<plugins>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.3</version>
    <configuration>
      <source>1.8</source>
      <target>1.8</target>
    </configuration>
  </plugin>
</plugins>
```

You could recollect the usage of **exec-maven-plugin** during source code execution.

## Surefire Plugin

One of the important plugin Maven offers is **Surefire Plugin**. This is *used for generating unit test reports* for your project.

- This plugin is very helpful to identify the unit tests that are failing during the build process.
- Once the issue related to unit test failure is addressed, Maven can once more run all the unit tests to ensure everything else is working fine with no impact by the changes done to address the previous failures.

The unit test report generated by this plugin is called **surefire report**.

Unit test reports are generated using command **mvn surefire-report:report**.

## Junit Surefire Plugin

We will learn about **JUnit Surefire** plugin usage. To start with, you need to add JUnit as the dependency in your project.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

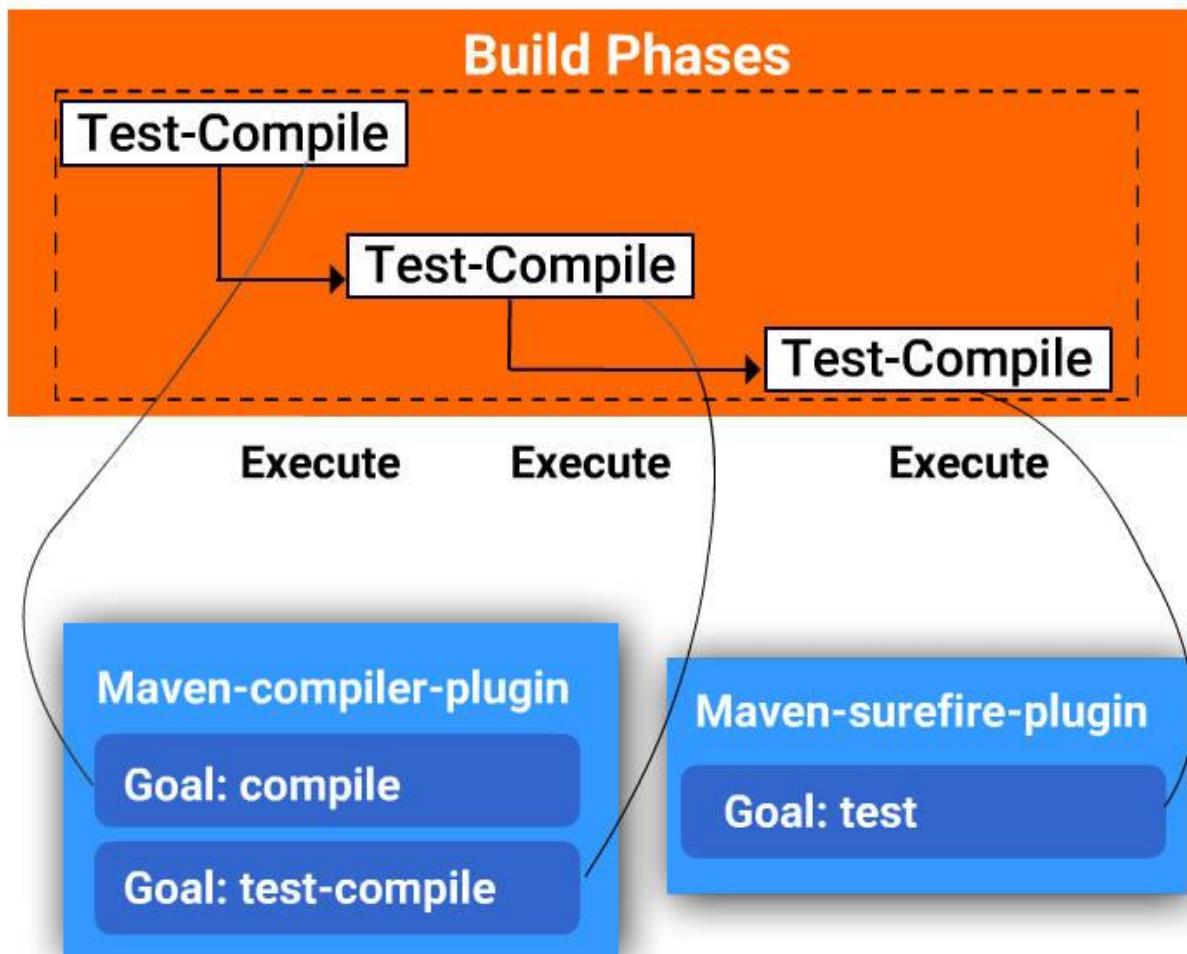
## Surefire with JUnit Versions

Surefire supports different generations of **JUnit**. Some of the commonly used versions are

- JUnit 3.8.x
- JUnit 4.x - Enables serial execution of the tests
- JUnit 4.7 - Enables parallel execution of tests

Appropriate generation of JUnit can be used with Surefire based on JUnit version being used in the project.

# Specifying a Provider Manually



To manually force a serial or parallel execution of tests, following code can be included in `pom.xml`:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.20</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.maven.surefire</groupId>
```

```
<artifactId>surefire-junit47</artifactId>
<version>2.20</version>
</dependency>
</dependencies>
</plugin>
```

In the earlier versions of surefire, ie., prior to 2.7 version, all the tests would get executed without proper checks on the validity of the tests. There would be no notification about the invalid tests.

To address this issue, the build can be run with - **Dsurefire.junit4.upgradecheck** to perform and notify on the invalid ones.

## Parallel Test Execution

Junit 4.7 onwards, you can perform parallel execution of tests. This can be done by passing **parallel** parameter and change **threadCount** attribute.

### Sample plugin for parallel execution:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.20</version>
  <configuration>
    <parallel>methods</parallel>
    <threadCount>10</threadCount>
  </configuration>
</plugin>
```

# Skipping a Test

To skip the test execution, use command **mvn install -DskipTests**. The same can be achieved by using the plugin and set the property of **skipTests** as **True**.

## Sample dependency:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.20</version>
  <configuration>
    <skipTests>true</skipTests>
  </configuration>
</plugin>
```

# Skipping Compilation of a Test

To skip the compilation of the test, use \*\*`mvn install -Dmaven.test.skip=true`

Skipping by Default`\*\*.

Following plugin code helps to achieve the same:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.20</version>
  <configuration>
    <skipTests>${skipTests}</skipTests>
  </configuration>
</plugin>
```

# Skipping Compilation of a Test

To skip the compilation of the test, use \*\*`mvn install -Dmaven.test.skip=true`\*\*

Skipping by Default`\*\*.

Following plugin code helps to achieve the same:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.20</version>
    <configuration>
        <skipTests>${skipTests}</skipTests>
    </configuration>
</plugin>
```

## Maven Jenkins Integration

### Jenkins Configuration



- *Kindly visit our Jenkins course before starting this topic.*

We have covered step by step [Jenkins setup and configuration](#) in that course.

- In this course, we will be covering ***Maven integration with Jenkins***. We assume that user had configured and set up Jenkins.
- Let us start Jenkins server and configure it for Maven.

## Configure Jenkins with Git

Now if we have our code in GitHub repository and want to have it in Jenkins for execution. For this:

- Click [Manage Jenkins](#) link from the Jenkins GUI.
- Click [manage plugins](#), then select the available tab by searching for [GitHub](#) from the search box.
- Select the [GitHub](#) plugins for installation and click [download](#) button.
- Select [Configure System](#) from the Manage Jenkins page.
- Verify that [JDK](#) version and [JAVA\\_HOME](#) path is set.
- In Git installation, set [Path](#) to [git executable](#).
- Set [Maven Home](#) path in the plugins.
- Click [Save](#).

## Set Git Repository in Jenkins

We need to use Git repository in the Jenkins.

- Select [New Item](#) from the Jenkins GUI.
- Enter the [Item](#) name as [FrescoPlayTest](#)(Name can be of your choice).
- Go to Maven project and select the [maven project](#) radio button.

- Click **OK** button to confirm maven project selection.
- Select **Git** from **Source Code Management**.
- Set the **Repository URL** in this box.
- Click **Advance button** twice, and you will get a checkbox option.
- Select **Skip internal** tag. (If it is not checked, Jenkins will create separate tags in the repository.)
- Scroll down to **Build section**, select **Invoke top-level**, then Maven targets from the drop-down list.
- Set Goal as **compile**. Now we have to set POM path.
- Go to Git and find your folder where **POM.xml** file exists. (e.g. App/pom.xml)
- Click **Save** button.

## Run Maven Project from Jenkins

- Now you can click **build now** tab.
- In Build History, right click and go to the **Console Output**.
- It will execute the script.

### JMeter and Maven Integration

#### Create JMeter Script



Open JMeter GUI. Now, we need to create a **Thread Group**.

1. Go to **Test** right click -> Select **Add-> select Threads (Users)-> Thread User** -> select **Thread Group**.
2. Enter **Thread Group Name**.
3. Enter number of **Threads** (users).
4. Enter **Ramp-Up Period**(in Seconds).
5. Enter **Loop count**.

Now we can create actual test script or manual record the process.

## Setting up JMeter Reports

### Add an **HTTP Request** element.

1. Go to **Thread Group-> Select Add -> Select Sampler->click on HTTP Request**.
2. On **HTTP Request** page enters valid **URL** in **Server Name** or **IP box**.
3. Set **Port**.

### Create **View Result in Table**:

1. Go to **Thread Group-> Select Add -> Add Select Listener-> View Result in Table**.

Now, we have created sample test.

## Integration JMeter Script with Maven

Take **JMeter script** and build it with Maven. We need **JMeter Maven plugin**.

- Create a Maven project for JMeter.
- Add JMeter Maven plugin in <build> element of pom.xml file.

```
<build>
  <plugins>
    <plugin>
```

```

<groupId>com.lazerycode.jmeter</groupId>
<artifactId>jmeter-maven-plugin</artifactId>
<version>2.1.0</version>
<executions>
  <execution>
    <id>jmeter-tests</id>
    <goals>
      <goal>jmeter</goal>
    </goals>
  </execution>
</executions>
</plugin>
</plugins>
</build>

```

- Include your JMeter script to Maven project class under src/test/App.
- Change directory to the project from terminal and enter mvn verify.
- If the build is proper, it will give us success message as Build Successful.
- The current executed result will be stored under target/App/result

## Include View - Result Plugins

- To get a result in the form of a graph, use JMeter Graph Maven Plugins.
- Add this plugin in same pom.xml after JMeter plugins details.

```

<plugin>
  <groupId>de.codecentric</groupId>
  <artifactId>jmeter-graph-maven-plugin</artifactId>

```

```

<version>0.1.0</version>

<configuration>
    <inputFile>${project.build.directory}/jmeter/results/20170319-BlazeDe
mo Home Page.jtl</inputFile>
    <graphs>
        <graph>
            <pluginType>ResponseTimesOverTime</pluginType>
            <width>800</width>
            <height>600</height>
            <outputFile>${project.build.directory}/jmeter/results/Blaze
Demo Request.png</outputFile>
        </graph>
    </graphs>
</configuration>
</plugin>

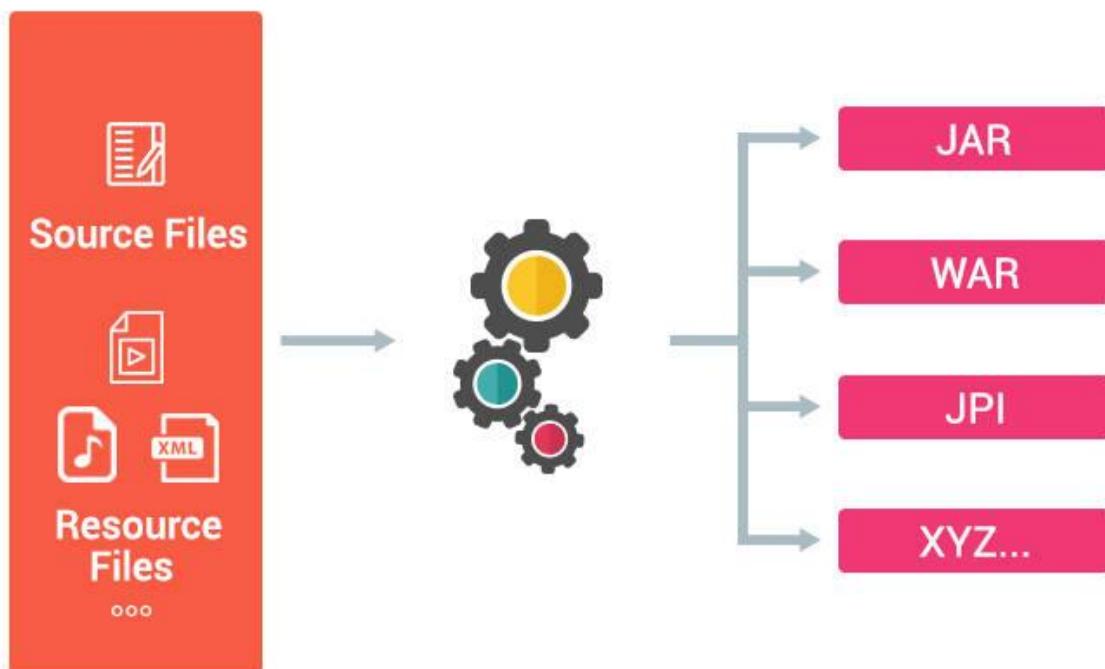
```

- After this, we need to make some changes in the script, open the JMeter GUI and in **Thread group** for **Loop Count**, increase parameter value to 100. This will help us to execute Thread request 100 times.
- To get the output, type **mvn jmeter-graph:create-graph** in the terminal.

# Gradle - Fabricating Systems

## What is Gradle?

What is a Build Tool?



Before you start with **Gradle**, it will be helpful to get comfortable with build tools first.

**Build tools** are programs that automate the process of turning up source code into executable applications.

Example: Ant, Maven, Gradle, and NAnt

They are responsible for:

- Compiling code resources
- Managing dependencies

- Generating documentation
- Running test scenarios
- Publishing the application

Gradle is a **declarative build tool** for Java projects.

It combines the best features of other build tools.

Gradle helps to automate:

- Compiling
- Testing
- Packaging
- Deployment of software or projects.

Android Studio, Eclipse IDE, IntelliJ IDEA, and NetBeans use Gradle build system.

## Gradle - Features

Gradle is an **open-source project**, and it is licensed under the Apache Software License (ASL). Following are some of the features:

- Utilizes a **Domain Specific Language (DSL)** based on **Groovy** or **kotlin** to declare builds.
- With **Deep API**, Gradle allows to monitor and configure the build script's execution behavior.
- Manages **dependencies** and **Structures** the build.
- Supports Ant tasks and projects.
- Uses **Maven and Ivy repositories** to publish or fetch dependencies.
- Supports **incremental builds** by executing only the necessary tasks in a build.
- Increases productivity.
- Supports **multi-project builds**.
- Extends **easy migration** of builds.
- The **Gradle Wrapper** allows to execute Gradle builds on machines where Gradle is not installed.

# Advantages of Gradle over Maven and Ant

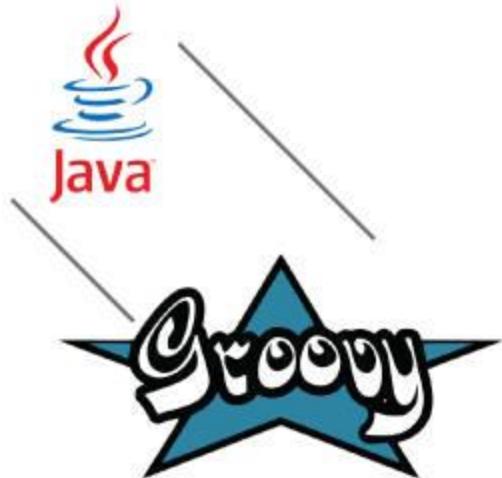


- Gradle build files are less verbose as they are written in **Groovy**, while others are in XML .
- **Stronger dependency management** than others ,especially its ability to dynamically replace project dependencies with external ones and vise versa

The following features are exclusively for Gradle :

- Supports **incremental builds**
- Provides **compiler daemon** which compiling faster
- Provides **Incremental compilations for Java classes**
- Supports **Compile avoidance for Java**
- Provides **version conflict resolution**

# Why Groovy?



Gradle's build scripts are written in **Groovy, not XML**. Do you want to know why?

**Groovy:**

- Provides **greater transparency** for Java people.
- The **base syntax, type system, and package structure** of Groovy are the same as **Java**.
- Overcomes all **Java limitations**
- **Reduces the size of a build script**
- **Groovy** is far more readable
- The **reduced line of code** implies a reduction in time.

## Set up Gradle

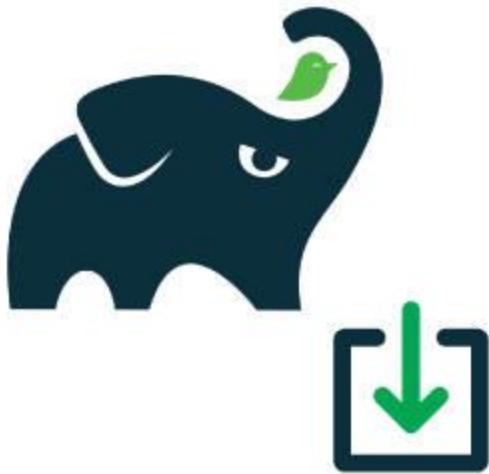
### Gradle in Local Machine - Prerequisite

To setup Gradle on your local machine, ensure that Java JDK or JRE, version 7 or higher is installed on the computer. To check, run the subsequent command on the command line:

`java -version`

Gradle carries its Groovy library. Therefore, **Groovy does not require to be installed explicitly**. If Groovy is installed, Gradle ignores that.

# Download and Install Gradle



- Download a Gradle distribution from the Gradle portal - <https://gradle.org/install/>.
- Extract the zip file to the desired folder.
- Set Environment variables – `JAVA_HOME` pointing to JDK, `GRADLE_HOME` pointing to Gradle. Add `GRADLE_HOME/bin` to `PATH` environment variable.
- To check if Gradle is correctly installed, type `gradle -v`. The output shows the Gradle version and also the local environment configuration (Groovy, JVM version, and OS).

## Installation Reference

More details on installing and configuring Gradle in different OS is given in the link below:

<https://gradle.org/install/>

## Basics of Gradle Scripts

### Introduction to Gradle Projects and Tasks

- Each Gradle build consists of one or more `projects`. Example: generating a jar file, deploy our application to a production environment, etc.

- Every project comprises one or more **tasks**.
- **Task** is a basic self-contained unit of work. Example: compile a java file, print hello, copy files from one location to another.
- A task can depend on one or more tasks.

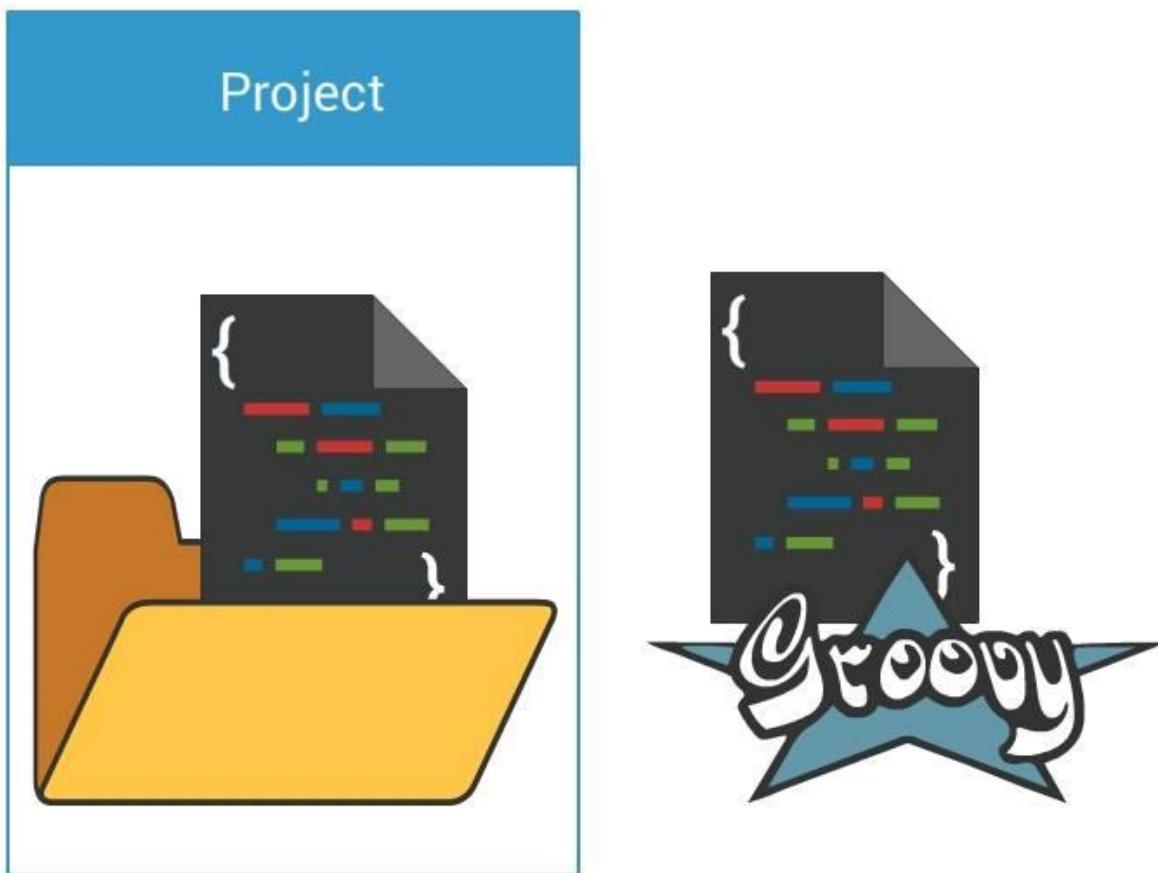
## Gradle Configuration Files



The Gradle build has the following **three** configuration files:

- **The Gradle build script (build.gradle)** specifies a project and its tasks.
- **The Gradle properties file (gradle.properties)** is used to configure the properties of the build.
- **The Gradle Settings file (settings.gradle)** is optional in a build which has only one project. If Gradle build has more than one project, it is necessary as it explains which projects engage to the build. Each multi-project build should include a settings file in the project hierarchy's root project.

# build.gradle



- **build.gradle** is the file located in the root folder of the project.
- This file is the **build script** for the project.
- It defines a project and its tasks.
- This can be written in **Groovy or Kotlin**.

## Custom Tasks

Let's see how to execute a simple task which prints 'Hello World'.

In the project folder, create a file **build.gradle** and add below code :

```
task hello {  
    doLast {  
        println 'Hello world!'  
    }  
}
```

With this code, we define a `hello` task. The task prints the words "Hello world!" to the console.

In the command line, type – `gradle -q hello` for checking the output.  
**Output of gradle -q hello**

```
> gradle -q hello  
Hello world!
```

## Custom Task in Action

You may use Gradle installed locally in your machine to carry out hands-on exercises from here on.

For checking the Gradle installation use `gradle -version` command.

Now, create `build.gradle` file using `cat` command: `cat > build.gradle`  
Enter the file contents and then press `Ctrl + C` to exit and return to the command prompt

Type `gradle -q <name_of_task>` for checking the output.

## One More Example

### `build.gradle`

```
task hello {  
    doLast {  
        String text = 'Hello world!'  
        println "Original: " + text  
        println "Upper case: " + text.toUpperCase()  
    }  
}
```

### **Output of gradle -q hello**

```
> gradle -q hello  
  
Original: Hello world!  
  
Upper case: HELLO WORLD!
```

# Skipping Tasks

Gradle provides several ways to skip the execution of a task.

- `onlyIf()` method is used to attach a predicate to a task.
- The task's actions are executed only if the predicate evaluates to true.
- The predicate is implemented as a closure.

## **build.gradle**

```
task hello {  
    doLast {  
        println 'hello world'  
    }  
}  
  
hello.onlyIf { !project.hasProperty('skip_hello') }
```

## Output of **gradle hello -Pskip\_hello**

```
>gradle hello -Pskip_hello  
:hello SKIPPED
```

# Task Dependencies

- Declare tasks that rely on different tasks.
- The tasks could be on the same project or different projects.

## **build.gradle**

```
task hello {  
    doLast {  
        String text = 'Hello world!'  
        println "Original: " + text  
        println "Upper case: " + text.toUpperCase()  
    }  
}  
  
task greeting(dependsOn: hello) {  
    doLast {  
        println 'This is executed after hello!'  
    }  
}
```

- Here, **greeting** task depends on **hello**.
- **hello** task executes first and then **greeting**.

## Output of gradle -q greeting :

```
> gradle -q greeting
Original: Hello world!
Upper case: HELLO WORLD!
This is executed after hello!
```

- Tasks also depend on tasks that could not be defined (yet).

## Task Properties

New properties can be added to a task. These properties can be read and set like a predefined task property.

In the following sample, to add a property named myProperty, set ext.myProperty to a value.

### Example: Adding extra properties to a task

```
task hello{
ext.myProperty = "Hello world!"
}

task greeting{
doLast {
    println hello.myProperty
}
}
```

## Output of gradle -q greeting

```
> gradle -q greeting
```

```
Hello world!
```

## Default Tasks

Gradle allows defining one or more default tasks that are executed if no other tasks are specified.

To set the default tasks, use the method `defaultTasks`. In the following build script, the tasks **hello** and **greeting** are made default tasks:

### Defining a default task:

```
defaultTasks 'hello', 'greeting'
```

```
task hello{
    doLast {
        println 'Default Hello!'
    }
}

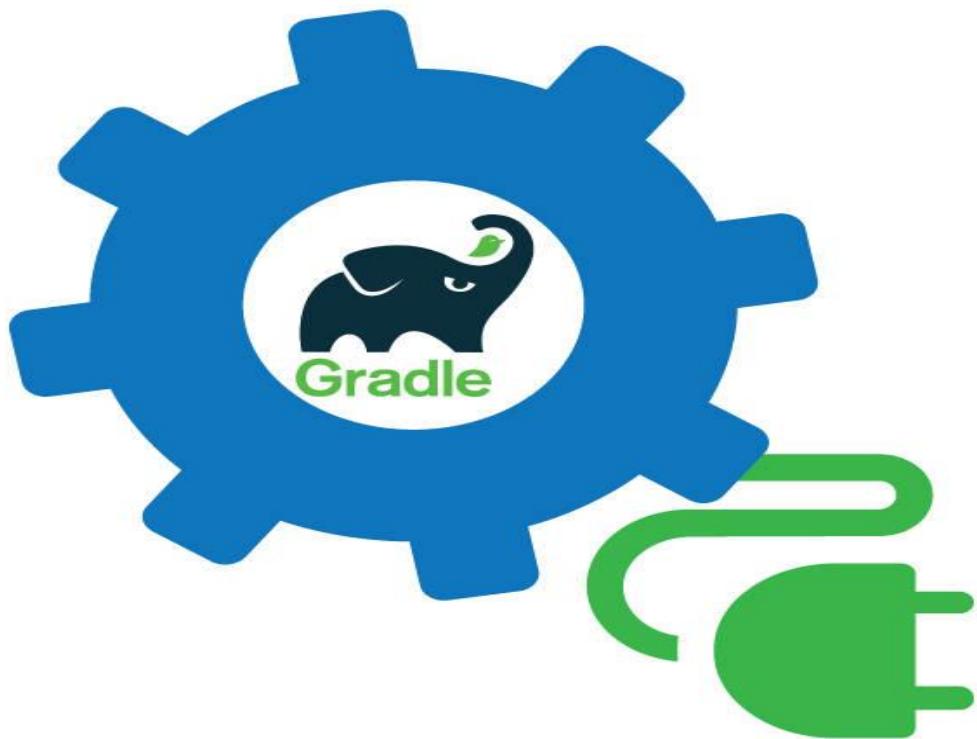
task greeting{
    doLast {
        println 'Default Greeting!'
    }
}
```

### **Output of gradle -q**

```
> gradle -q
Default Hello!
Default Greeting!
```

## Gradle Plugins

### Gradle Plugins



- A Gradle plugin works as an **extension to Gradle**. It extends the project's capabilities when applied to a project.
- Gradle comes with several plug-ins, and you can create custom plug-ins. *Example:* **Java plug-in** adds tasks to a project that permits to create a JAR file, run unit tests and compile Java source code.
- A plug-in is included in a build.gradle file with the `apply plugin:'pluginname'` statement.

**Example:** The entry `apply plugin:'com.android.application'` makes the Android plug-in available for a Gradle build.

- Gradle also provides a registry for plug-ins via <https://plugins.gradle.org/>

## Applying plugins

- Plugins are applied using the `Project.apply()` method.
- The same plugin can be applied multiple times.

# Types of Plugin

## Script plugins

- They are additional build scripts.
- They follow a declarative approach to manipulate the build.
- They are applied from a script on the local filesystem or at a remote location.

## Example: Applying a script plugin

```
apply from: 'other.gradle'
```

## Binary plugins

- They are classes that implement the plugin interface.
- They follow a programmatic approach to manipulate the build.
- They occupy within a build script/project hierarchy/externally in a plugin jar.

## Example: Applying a binary plugin

```
apply plugin: 'java'
```

- Applied using a unique plugin id (in the above case - 'java')

## Java Plugin



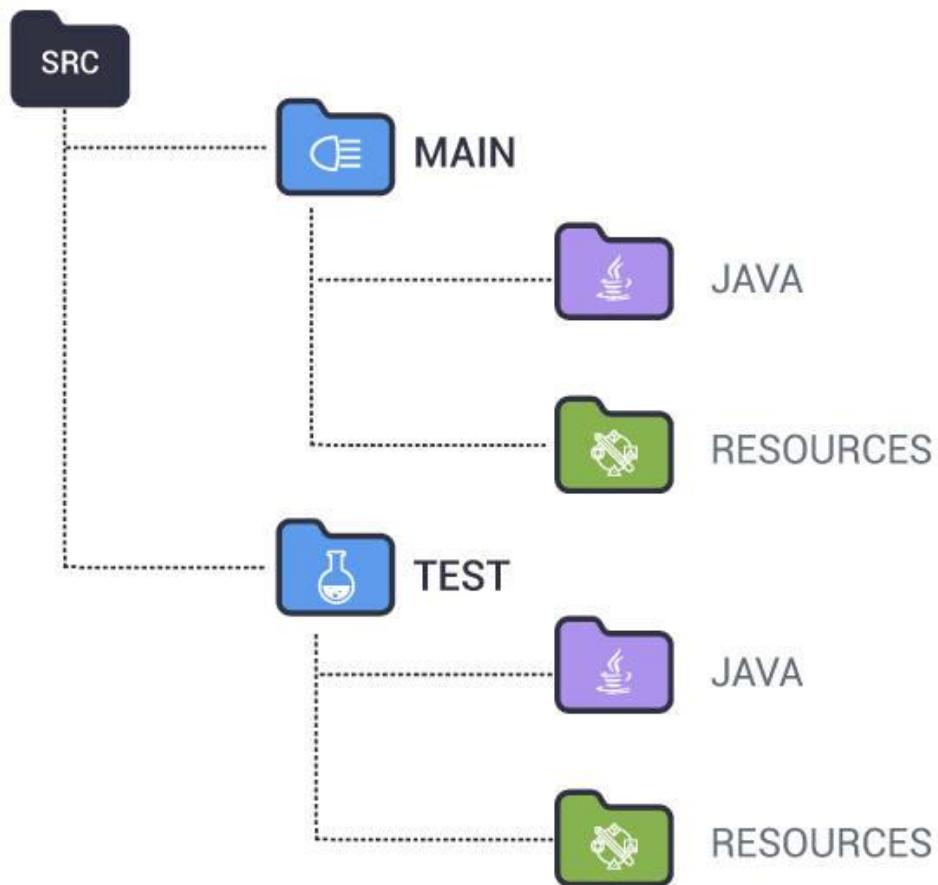
- The Java plug-in offers tasks to **create a JAR file, create Javadoc, run unit tests, and compile Java source code.**
- It acts as the basis for many of the other Gradle plugins.
- To work with the Java plugin:
  - Create a Java source file
  - Use the tasks of the plugin to build the source file.

### Example: Using the Java plugin

```
apply plugin: 'java'
```

The above code is included in the build script to use Java plugin. By default, the plugin searches for Java source file in the directory `src/main/java`, relative to the project directory.

## Source Sets



A **source set** is considered as a group of source files (Java source files and resource files) that can be compiled and executed together.

The Java plugin defines two standard source sets - **main** and **test**.

- `src/main/java` contains the *Java production source code*.
- `src/test/java` contains the *Java test source code*.
- Non-Java source files that are included in the JAR file are placed in `src/main/resources`.

- Non-Java source files needed for testing are placed in `src/test/resources`.

## Build Tasks Added by Java Plugin

```
$ gradle build
```

- To start the build, type `gradle build` on the command line.
- The figure given here is the output while running the command `gradle build`.
- `build` is one of the tasks the Java plugin adds to the project. Here, some tasks are marked with the message UP-TO-DATE, which means that the task has been skipped.
- The `build` task compiles the code, runs tests, and assembles the JAR file, in the correct order.
- Once the `build` is successful, open the project directory, and you will see a build folder created with the following contents:

- **classes** - contains the compiled classes.
- **libs** - contains the assembled JAR file.
- **tmp** - contains temporary generated files, such as a manifest file.

## Customizing the Project

### Changing properties and adding a JAR header

The subsequent code snippet shows the way to include a header attribute to the JAR manifest and to configure default values in the build script.

#### Customization of MANIFEST.MF

```
version = '1.0'
sourceCompatibility = 1.7
jar {
    manifest {
        attributes 'Main-Class': 'HelloGradle'
    }
}
```

- A **version number** for the project is specified, and the Java source compatibility is indicated.
- The **jar task** automatically adds a manifest file to the JAR file it creates.
- Jar task's manifest property is used to add new entries to the manifest file. Here, a *Main-Class header* is added to jar file's manifest.

### Changing the project default layout

```
sourceSets {
    main {
        java.srcDirs=['src']
    }
}
```

- In the above code, we have told Gradle where the source code is.
- The Location of source code is at the root level, in the source directory (src -name of directory).

# Configuring and Using External Dependencies

## Defining external dependencies for the build script

```
dependencies {  
    compile 'com.google.code.gson:gson:2.8.0'  
}
```

Here, we have told Gradle, there is a **dependency**.

- The dependency for the compile configuration is defined.
- There are three parts for the name of dependency:

name of the organization that created the dependency: name of the dependency itself:version number

## Defining the repository

```
repositories {  
    mavenCentral()  
}
```

Here, we have told Gradle, **the location of the repositories** for the dependencies.

# Publishing the JAR file

In Gradle, JAR files are published to repositories.

In the example below, JAR file is published to a local directory.

It can also be published to a remote location or multiple locations.

## Example: Publishing the JAR file

```
uploadArchives {  
    repositories {  
        flatDir {  
            dirs 'repos'  
        }  
    }  
}
```

To publish the JAR file, run **gradle uploadArchives**.

# Build Init Plugin



- The Gradle Build Init plugin is used to:
  - create brand new projects of different types
  - convert existing builds to Gradle builds.
- It is an automatically applied plugin.
- To use the plugin, execute the task named `init`, where the Gradle build is to be created. Without new parameters, this task creates a Gradle project that comprises `settings.gradle` file, `gradle wrapper files`, and `build.gradle`.

## init with --type parameter

- The `init` supports different build setup types.
- The type is specified by supplying a `--type` argument value.

Example: To create a Java library project, execute:

```
gradle init --type java-library.
```

- If a `--type` parameter is not supplied, Gradle will attempt to infer the type from the environment.
- If the type cannot be inferred, the type “basic” will be used.

## java-application

The **java-application** build init type must be explicitly specified.

## **Features:**

- Uses the **application** plugin to produce a command-line application implemented using Java.
- Uses the **jcenter** dependency repository.
- Uses **JUnit** for testing.
- Features **directories** in the general locations for source code.
- Contains a **sample class and unit test**, if there are no existing source or test files.

To use a different test framework, execute:

```
gradle init --type java-application --test-framework testing`
```

## **java-library**

The **java library** must be explicitly specified.

### **Features:**

- Uses the **java** plugin to produce a library Jar.
- Uses the **jcenter** dependency repository.
- Uses **JUnit** for testing.
- Has **directories** in the conventional locations for source code.
- Contains a **sample class and unit test**, if there are no existing source or test files.

To use a different test framework, execute:

```
gradle init --type java-library --test-framework testing`
```

## **Creating an Eclipse Project**

Another plugin should be added to the build file to create Eclipse-specific descriptor files (.project).

### **Example: Eclipse plugin**

```
apply plugin: 'eclipse'
```

Here, Gradle eclipse command is executed to generate Eclipse project files.

# Integration with Jenkins

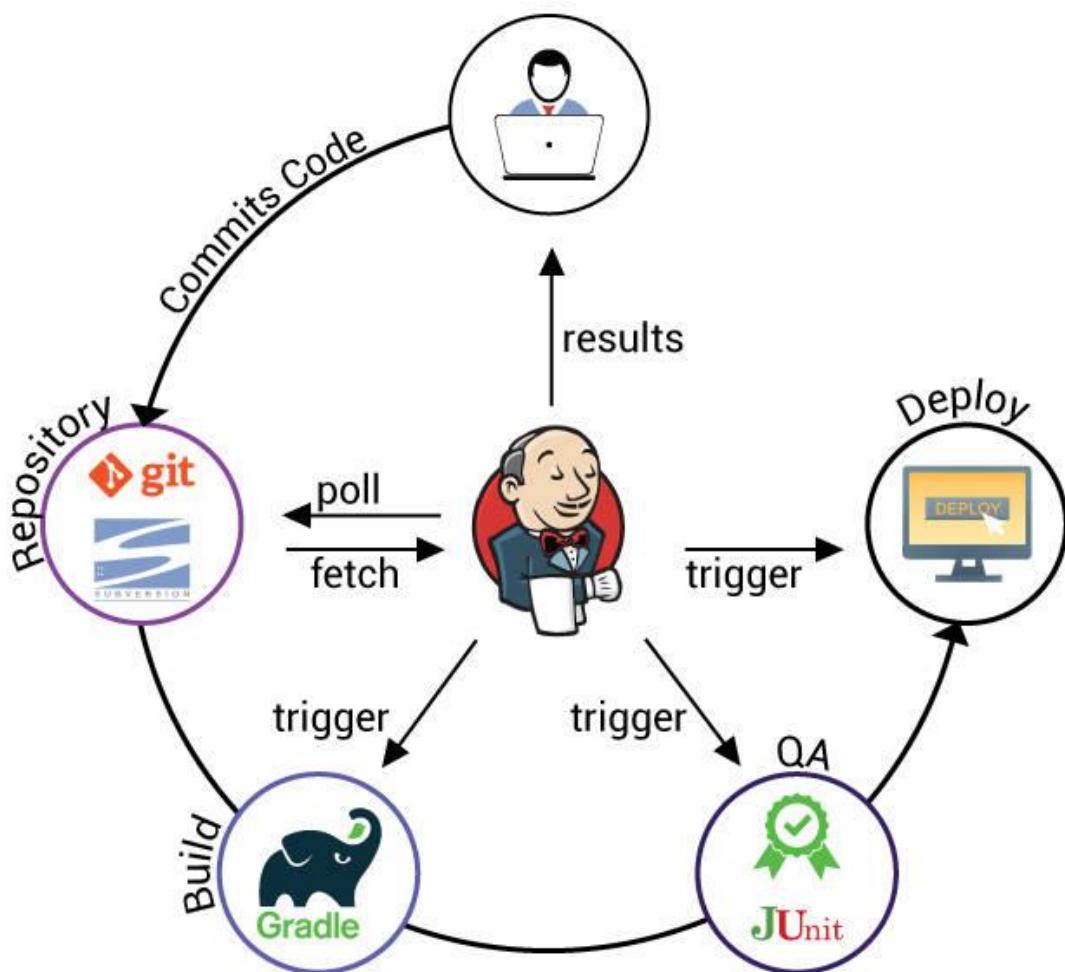
Jenkins



- One of the favorite open-source **continuous integration** tool.
- Helps to automate software build.
- Performs software build using build tools such as Maven, Gradle, and Ant.
- Offers a variety of plugins supporting build, deployment, and automation of projects.
- Supports different repositories like SVN and Git.

You can visit the course [Continuous Integration with Jenkins](#) for a better understanding.

# Workflow in Jenkins



- Developer commits the code into the repository.
- Jenkins detects the changes.
- Triggers to build the source via Gradle.
- Initiates quality checks.
- Prepares to deploy.

# Jenkins Installation



- Download [jenkins.war](http://jenkins-ci.org/) from <http://jenkins-ci.org/>.
- After the download, it can be deployed in a container such as Tomcat, or via the command line with `java -jar jenkins*.war`.
- If it is started locally, open a browser and type the following URL: <http://localhost:8080/> and Jenkins welcome page will be displayed.

# Jenkins Configuration

A screenshot of a web browser displaying the Jenkins dashboard at [localhost:8080](http://localhost:8080). The title bar shows "Dashboard [Jenkins]". The main content area features the Jenkins logo and the text "Welcome to Jenkins!". It includes a call-to-action button "Please [create new jobs](#) to get started." On the left sidebar, there are links for "New Item", "People", "Build History", "Manage Jenkins", and "Credentials". Below these are sections for "Build Queue" (No builds in the queue) and "Build Executor Status" (1 Idle, 2 Idle).

Configure the following plugins for Jenkins:

- Gradle Plugin
- Git Plugin (required if Git is used as a repository)

The Jenkins welcome page is shown above.

- Click **Manage Jenkins** on the left-hand side vertical menu.
- A list of different categories will be displayed.
- Click **Manage Plugins**. Four tabs will be displayed.
- Go to the **Available** tab and filter (top right) for **Gradle Plugin**.
- Select the plugin and click **Download now** and install it.

This adds Gradle build execution capability to the Jenkins server.

- Next, configure JDK, Gradle, and Git with Jenkins.

To configure these settings:

- Open Jenkins URL.
- Click \*\*Manage Jenkins\*\* and click Configure System.
- Enter the path to JDK and save the settings.

## Create the First Gradle Build Job

**Jobs** are the unit of execution in Jenkins. A build job can perform **compilation, run automated tests, package and deployment related tasks**.

Steps:

- Go to Jenkins home page and click **Create new jobs**.
- Choose a category of jobs (Freestyle project) that can be created in Jenkins.
- Type a name for the project and then click OK.

In the next page, the following details have to be configured:

- **Source code management location** to download the project.
- **Build step** for the project.
- **Schedule** the Build task (daily, hourly, after every commit, etc.).
- Put in any **post build action** to perform.

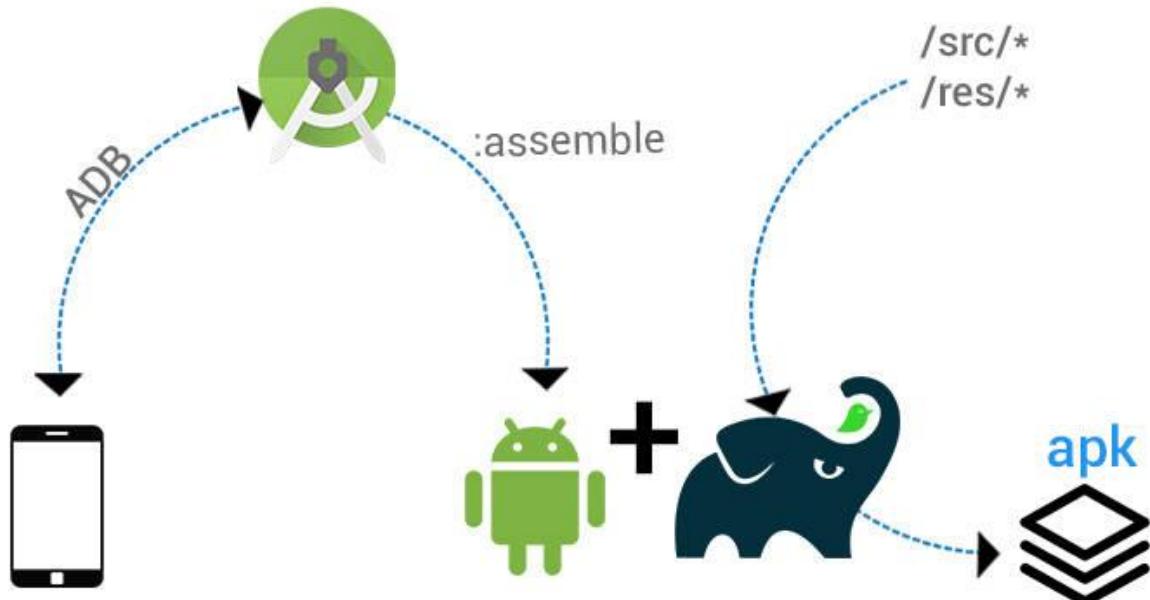
After saving the configurations, the project will be displayed on the dashboard.

## Execute Job

- Click the <name\_of\_project> job on the Jenkins home page. It gets navigated to the job console at [http://localhost:8080/job/name\\_of\\_project/](http://localhost:8080/job/name_of_project/).
- On the job console, click the **Build Now** option (left-hand side) to execute the job manually.

## Building Android Apps with Gradle

### Linking with Android Studio



- Gradle is regarded as the build tool for **Android Studio**.
- Android Studio assigns the entire process of building android apps to Gradle.
- Gradle doesn't know anything about Android. Thus **Android Gradle plugin** is leveraged to bridge the gap.
- Gradle offers to create variants of apps easily like,
  - debug and release builds
  - paid versus free versions

## Creating Android Project

Name	Type	Size
.gradle	File folder	
.idea	File folder	
app	File folder	
build	File folder	
gradle	File folder	
.gitignore	Text Document	1 KB
AndroidSampleApp.iml	IML File	1 KB
build.gradle	GRADLE File	1 KB
gradle.properties	PROPERTIES File	1 KB
gradlew	File	5 KB
gradlew.bat	Windows Batch File	3 KB
local.properties	PROPERTIES File	1 KB
settings.gradle	GRADLE File	1 KB

When the Android Studio creates a project, the **directory structure** should look like the figure given.

- In the project home directory, there are `build.gradle` and `settings.gradle` files. i.e., Android Studio has created a multi-project build structure.
- Android Studio creates :
  - one `build.gradle` for the parent project
  - individual `build.gradle` files for the subprojects.
  - creates `settings.gradle` file that includes all the subprojects.
- Android Studio also adds **Gradle Wrapper**. Thus, Android project can be built on a machine where Gradle is not installed.

## Creating Android Project

Name	Type	Size
 app		
 build	File folder	
 libs	File folder	
 src	File folder	
 .gitignore	Text Document	1 KB
 app.iml	IML File	8 KB
 build.gradle	GRADLE File	1 KB
 proguard-rules.pro	PRO File	1 KB

The actual Android application is in the `app` directory, and its content is shown in the figure.

**Source directory:** `src/main/java`

**Test directory:** `src/androidTest/java`

## Building Android Project with Gradle

Android Studio has automatically generated two build files for the project:

- one in the **root folder** of the project
- other in the **app directory**.

The `build.gradle` file of the app folder is used to build the Android application.

This `build.gradle` file has the following content:

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "ch10.androidsampleapp"
        minSdkVersion 15
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.1.1'
}
```

With these configurations, the application is ready to get build with Gradle.

## buildTypes

- The **buildTypes** configuration is used to define types or environments of build, such as debug, release, QA.
- By default, both the debug and release versions of the Android project are in the **build/outputs/apk** directory.

You can customize both build and release build types and also extend the build types by adding your own build types, as follows:

```
buildTypes {  
    release {  
        minifyEnabled false  
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
    }  
  
    staginginitWith(buildTypes.release)  
    staging {  
        debuggable true  
    }  
}
```

Here, one more build type **staging** is added and configured it to be a copy of the release build type and added **debugable true**.

## Gradle Summary

### Sample Build File

Let's have a glimpse of build file **build.gradle** :

```
apply plugin: 'java'  
apply plugin: 'eclipse'  
  
sourceCompatibility = 1.7  
version = '1.0'  
jar {  
    manifest {  
        attributes 'Implementation-Title': 'Gradle Quickstart',  
                  'Implementation-Version': version  
    }  
}
```

```

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'commons-collections', name: 'commons-collections', version: '3
    .2.2'
    testCompile group: 'junit', name: 'junit', version: '4.+'
}

test {
    systemProperties 'property': 'value'
}

uploadArchives {
    repositories {
        flatDir {
            dirs 'repos'
        }
    }
}

```

## Gradle Course Summary

**Congrats! You have completed your learning on the Gradle basics.**

In this course, you read about one of the most popular Build Automation System - **Gradle** and covered in-depth about:

- How to install Gradle
- Create Gradle script
- Gradle plugins especially Java plugins
- Build init plugins
- Integrating Jenkins with Gradle
- Building Android apps with Gradle

# Continuous Integration with Jenkins

## Introduction to CI

Will you be pleased to discover an error, a show stopper, at a crucial stage in your software?

Obviously, the answer is **No!**

What steps can be taken to detect and fix coding issues at an early stage of the development cycle?

**Continuous Integration (CI)** can help in providing a solution.

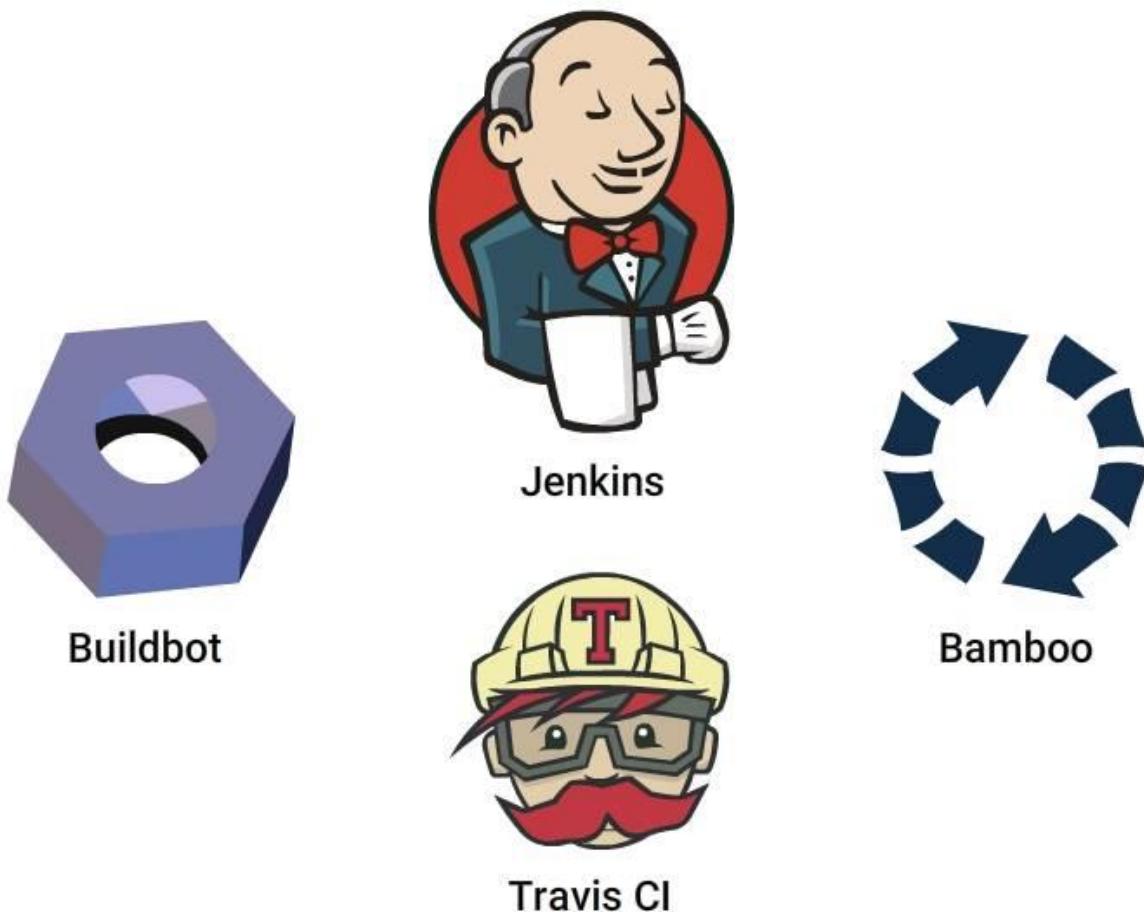
## What is CI?



According to [Martin Fowler](#),

**Continuous Integration** is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

## How to Enable CI?



You need a tool to create a **CI** enabled environment. [Jenkins](#), [Travis CI](#), [Bamboo](#), [Buildbot](#) are different tools available to enable CI.

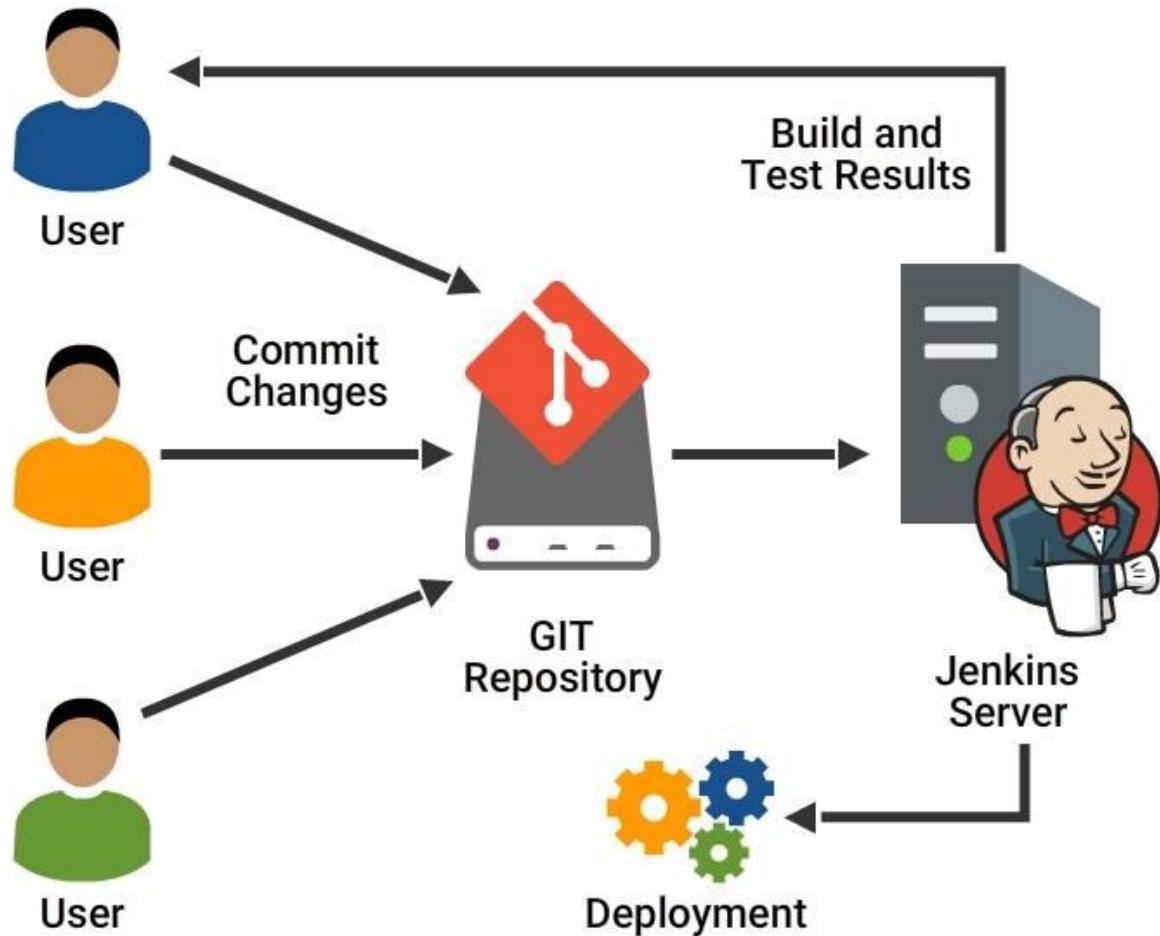
In this course, you will be learning about **Jenkins**, a widely used open source CI tool, written in [Java](#).

## Minutiae - 2

Jenkins was earlier referred to as Hudson

### Introduction to Jenkins

What does Jenkins do?



- *Compiles and builds* the code
- *Runs* an internal shell or command line script
- *Starts execution* of the integration tests
- *Monitor* execution of tasks
- *Stops build* in case of failure
- *Notify user* on the build status
- *Deploy* in test or production environments

# Features of Jenkins

- *Easy to install*
- *Easy to configure* various tasks
- *Rich plugin ecosystem* - Integrates with a variety of build, test, deploy, reporting tools
- *Permanent links* - Jenkins provides direct links to the latest or failed build, which can be used for easy communication
- *Extensibility* - Customize Jenkins to suit your needs
- *Distributed builds* - Jenkins can distribute build, test jobs to multiple computers with different operating systems
- *File fingerprinting* - Manages dependencies
- *Email integration*- Emails the build status

## Difference made by Jenkins

### **Pre Jenkins:**

- Source code was completely built and then tested
- Bugs identified during testing in the source code, should be fixed and then re-tested
- Slows the software delivery, as the entire process is manual

### **Post Jenkins :**

- Once code change is committed, Jenkins automatically takes care of the build , test and reporting of results.

## Minutiae - 3

Jenkins was primarily developed by Kohsuke Kawaguchi.

## Installing and Configuring Jenkins

### Pre-Requisites to Install Jenkins



Recommended minimum configuration for installing Jenkins:

- *locally* - Java 8, 256MB RAM and > 1 GB free disk space
- *small team* - Java 8, >1 GB RAM and > 50 GB free disk space

Since all the builds take place on the Jenkins machine , system should have **enough disk space** for **build storage**.

Jenkins can be installed on **Windows, Ubuntu/Debian, Red Hat, Fedora/CentOS, Mac OS, X openSUSE**

## Installing Jenkins

Jenkins can be started from command line or can run on a web application server.

- Download the jenkins.war file from [Jenkins](#)
- Start Jenkins directly from Command line with java-jar jenkins.war.
- On Successful completion , Jenkins can be accessed locally from <http://localhost:8080/>.

### ***To run it from Tomcat server***

- Put the .war file into the webapps directory.
- ***Start Tomcat***, Jenkins installation will be available on (<http://localhost:8080/jenkins>).

## **Launching Jenkins**

On first launch of ***Jenkins***, you will need to complete few configuration steps -

- Obtain a generated value which is the super admin password
- Proceed as per instructions on the screen
- Proceed to install **suggested plugins**
- Set up **admin user** when prompted or choose to continue with the generated value

***Jenkins is now ready for use!***

## **Configuring Jenkins**

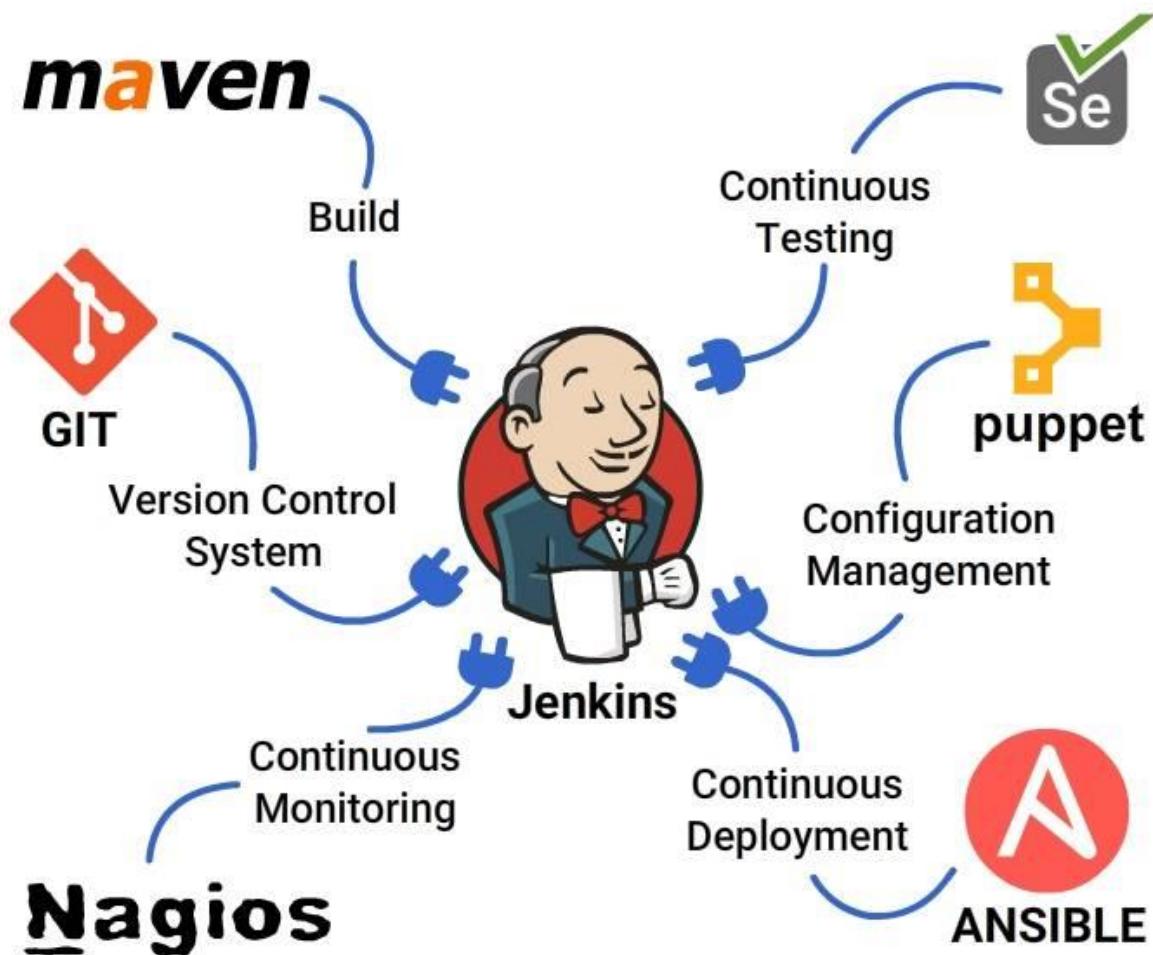
*Jenkins installation is now complete. What are the next steps?*

- Select the Jenkins home directory (prefer location with enough free space)
- Decide number of concurrent job executions to be allowed on **Jenkins** machine
- Add custom environment variables.
- Mention SMTP server, user email suffix in the email notification section.
- Configure the location of JDK installation.
- To build Maven applications configure the location of Maven Home.

You can perform these tasks by selecting ***Configure System under Manage Jenkins***.

## Plugin Management

# Extending Jenkins Functionality



- Once Jenkins is installed, it is time to configure it, to fit your needs.
- Jenkins has relatively few abilities, but it aids the s/w developers by providing a variety of plugins.

Plugins are add-ons that allow Jenkins to interact with many other softwares

- The exact plugins you install depends on the nature of your project.

# Plugins in Jenkins

[Jenkins Plugins Index](#) provides you the various plugin options such as:

- **Source Control:** Git, SVN, Mercurial
- **Testing :** Selenium, Windmill
- **Triggers:** Jabber, Directory watchers
- **Artifact:** To copy components between projects like Amazon S3, SCP
- **Code Analysis:** To parse the code with tools like CheckStyle, Findbugs, PMD
- **Build Tools:** In large projects use a build manager such as *Maven* or *Ant*.
- **Reporting:** Jenkins provides its own reports. It can be extended using tools like **Static Analysis Collector** that collects the different analysis results and shows it in a combined trend graph.

Plugins can be configured via the **Manage Plugins under Manage Jenkins**.

## Creating a Jenkins Project

### Set up a simple project



For setting up a new project in **Jenkins** , following sections are to be planned and configured as required:

- **SCM** - Associate with a version control server
- **Triggering Build** - Control when Jenkins will perform builds by Polling, Periodic or Build based on other projects
- **Execution of scripts, Ant and Maven targets**
- **Archiving the artifacts**
- **Recording and publishing** build and test results
- **Email notifications**

Next few cards, will help you in defining these sections.

## Create a new project

Now you will learn, how to create a simple build project.

- Select *New Item* from Jenkins dashboard
- Type the *project name*
- Select **Freestyle** project (freestyle is the most configurable and flexible option, easy to setup!)
- Click ok.

## Configure Job Notifications

In the job notifications section, you can choose to -

- Prevent build of your project, when the dependent upstream or downstream job is in the queue or is building.

Notifications plugin like *Tikal* will expand the available notification options.

- On installing Tikal , you can send job status in JSON and XML formats
- Set up the notification endpoint section with the required details.

# Configure Build Triggers

After source code location is defined , you need to **configure Jenkins** to check for code changes, so that, build is triggered automatically.

Various options to trigger the **builds** are:

- Build whenever a **SNAPSHOT** dependency is built
- **Trigger** builds remotely (e.g., from scripts)
- Build after other **projects** are built
- Build **periodically** (Runs on CRON job)
- Poll **SCM** (Runs on CRON job)

## How to define CRON expression?

Jenkins schedules are configured using the CRON syntax.

It consists of **five fields** separated by white space, indicating respectively the minute (0–59), hour (0–23), day of the month (1–31), month (1–12) and the day of the week (0–7, with 0 and 7 being Sunday).

star is a **wildcard** character which accepts any valid value for that field.

- “\* \* \* \* \*” means every minute of every hour of every day.
- “\* 9-17 \* \* \*” means every minute of every day, between 9am and 5pm.

There are other convenient short-hands, such as “**@daily**” and “**@hourly**”.

# Adding a Build Step

The screenshot shows the Jenkins configuration interface for a job named 'Jenkins Demo'. At the top, there are tabs for 'Terminal', 'Dashboard', and a plus sign icon. Below the tabs, the Jenkins logo is displayed. The main area shows the job's schedule as 'H/5 \* \* \* \*' (every 5 minutes). A note below the schedule indicates it would last have run at Tuesday, April 25, 2017 11:15:28 AM GMT and would next run at Tuesday, April 25, 2017 11:15:28 AM GMT. There is also an 'Ignore post-commit hooks' checkbox. In the 'Build' section, a dropdown menu titled 'Add build step' is open, showing options: 'Execute Windows batch command' (which is highlighted with a blue box and has an arrow pointing to it), 'Execute shell', 'Invoke Ant', and 'Invoke top-level Maven targets'.

Now you have learnt how to setup the code location and build frequency. Next step is to configure the build action.

In the build section, select *Add Build step* and opt for the required build option.

- **For simple Java build** - Select *Execute Windows Batch command* and enter the script in the command window.
- **For Maven build** - Select *Invoke top-level Maven targets*. Enter clean package, clean install or clean test as appropriate in the *Goals* field .

Hope you know - **clean package** will delete any previous build artifacts, compile code, run unit tests and generates a JAR file.

# Configure Post Build Activities

Once your code is built, the results should be displayed for you to check and act. Jenkins does a great job of displaying test results and trends. Some of those are :

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- Deploy artifacts to maven repository
- Record fingerprints of files to track usage
- Email Notification

Install post-build script plugin to help you execute scripts after build completion.

# Notifications

The screenshot shows the Jenkins configuration interface for a project named "Jenkins Demo". The top navigation bar includes "Terminal", "Dashboard", and a "+" icon. The main header features the Jenkins logo and a search bar. The breadcrumb navigation shows "Jenkins > Jenkins Demo > configuration".

The configuration page has a "Build" section. Under "Execute Windows batch command", the "Command" field contains "javac sample.java". A dropdown menu titled "Add post-build action" lists several options: "Aggregate downstream test results", "Archive the artifacts", "Build other projects", "Publish JUnit test result report", "Publish Javadoc", "Record fingerprints of files to track usage", "Git Publisher", and "E-mail Notification". An "Apply" button is visible at the bottom of this section.

Now you have setup everything that is required for a build job :  
Detect code change, Trigger build , Record results.  
*But, will you not require an automatic notification of the build status?*

How about an email notification?

Jenkins provides an *Email Notification* section. Mention the recipients under your project configuration.

## Execution of the Jenkins Project

### Executing the build job

Once a project is created successfully in Jenkins:

- Build job is displayed on the Jenkins dashboard
- Build starts automatically based on the build trigger settings
- To run the job manually select *Build Now*
- Progress of the build is displayed in the *Build History* section
- Once build completes, click on the build number
- Select the console output to see the details of the build.

### Build job status

Build Status is indicated in two ways :

- A **weather icon** (on the home page dashboard) - shows you a record of multiple builds
- A **colored ball** (on the individual project page) - shows you status of a single build

Status has corresponding tooltips with explanations , when you hover over it.

### Job Status

Job health	Description
	<b>No recent builds failed</b>
	<b>20-40% of recent builds failed</b>
	<b>40-60% of recent builds failed</b>
	<b>60-80% of recent builds failed</b>
	<b>All recent builds failed</b>

# Project Status

Status of the build	Description
	<b>Failed</b>
	<b>Unstable</b>
	<b>Success</b>
	<b>Pending/Disabled/Aborted</b>

## CI pipeline with Jenkins, Git, Tomcat

### Setting up Jenkins CI Pipeline

In the Katacoda exercise, you would have learnt how to use jenkins and docker in the CI pipeline.

Below are steps to setup Jenkins CI pipeline from Git to Tomcat.

#### ***Pre-Requisites***

- Jenkins, Maven, Tomcat should be up and running in your machine
- Source code should be made available in [Git](#)

#### ***Plugins to install***

- Deploy to Container , Git

#### ***Jenkins Jobs to be created-***

- Create a **Pull** job that detects code changes in Git
- Create a **Build** job that is triggered **after** the **Pull** job
- Create a **Deploy** job to push **results** to tomcat.

## Setting up Deploy Job

You should be able to create the first two jobs easily. Below steps will help you to create the **deploy** job.

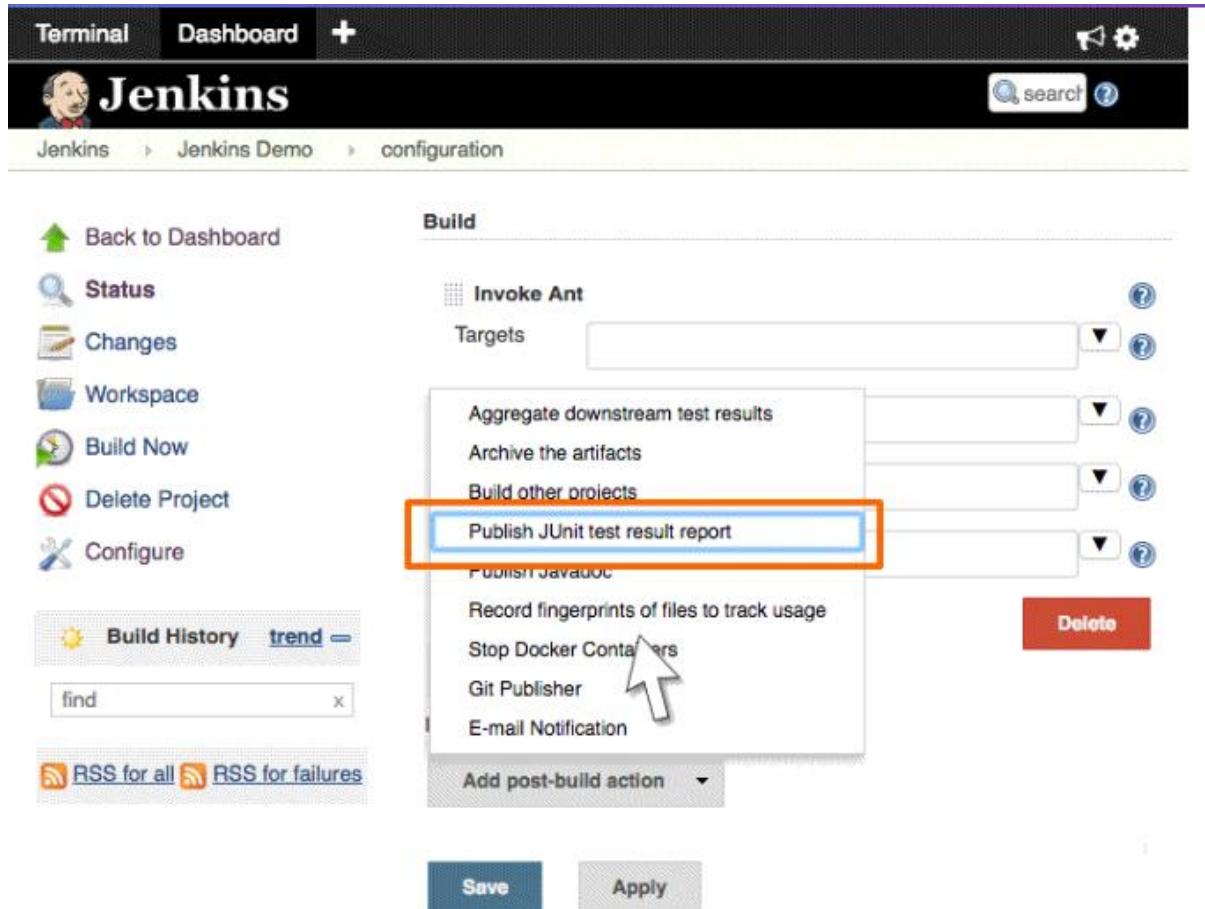
***In the Post-build Actions section:***

- Select deploy war/ear to a container
- Mention the \*.war file
- Mention **Context path**- where your application will be published in Tomcat
- Select Tomcat version in the Container dropdown
- Enter credentials of user who can access Tomcat
- Mention **Tomcat URL** to reach your tomcat instance

On changing your code in **Git**, you will see these jobs will trigger automatically on Jenkins.

## Invoking Test from Jenkins project

### Steps to invoke unit test



Jenkins provides a host of plugins for unit testing such as Junit and Mstest for .Net unit tests..

#### *If you choose to use the Junit test*

- Select project and choose the configure option
- In Add Build step , choose action to invoke Ant.
- Select *Advanced* and enter the location of build.xml file
- In Add post-build, select *Publish Junit test results*.
- In Test Report XMLs field, enter location of result xml files produced by executing Junit test cases
- Save and then build.

# Installing Hudson Selenium Plugin



*Do you want Jenkins to decide automatically whether a build is ready to proceed to the next stage ?*

## Use Automated tests

For this, install *Hudson Selenium Plugin* from *Manage Plugins* section.

## Steps to invoke automated test

The screenshot shows the Jenkins configuration interface for a project named "Jenkins Demo". The left sidebar contains links like "Back to Dashboard", "Status", "Changes", "Workspace", "Build Now", "Delete Project", and "Configure". The main area shows "Build Triggers" with options for Git, Subversion, and various periodic triggers. Under the "Build" section, a dropdown menu is open, showing options such as "Execute Windows batch command", "Execute shell", "Invoke Ant", "Invoke top-level Maven targets", and "SeleniumHQ htmlSuite Run". The "SeleniumHQ htmlSuite Run" option is highlighted with a red box and has a mouse cursor pointing at it.

- Once plugin is installed successfully, select your project, click on the configure option.
- In the Add Build Step section, select the Selenium SuiteRun option.
- Add the necessary details for tests like browser, suitefile, resultfile and Save.

*Now you are ready to build and get it going...*

### **Security Management in Jenkins**

**How Secure is Jenkins ?**



***Jenkins is open to every one.*** So, anyone can access Jenkins and ***perform*** all ***available tasks*** that were discussed previously.

*Hence it is recommended to secure Jenkins.*

It provides settings for security and role management - useful for controlling access and defining user actions.

### **Setting Admin User**

- Select **Configure Global Security** under *Manage Jenkins*.
- In the next window - select **Enable Security**.
- Choose **jenkins own user database** to maintain your own user database.
- For a central administrator to define other users in the system, *unselect* **Allow users to signup** and save.

- Then, set up the admin user.

## Configuring User Access

As an admin user, you can now set up other *users* .

- Select **Manage Users** under **Manage Jenkins** to add other users
- Now you need to setup **authorizations**
- Select **Matrix based Security** under **Configure Global Security**
- If the **user** is not in the user group list, enter the **user** and add to the **list**
- Define the **relevant authorizations** and **save**

*Jenkins security setup is now complete.*

## Distributed Build

How to distribute work load ?



What if, as part of a *large project*, you need to **build codes** and **report results** regularly ?

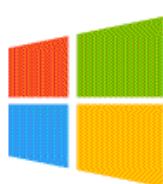
*Running all the builds on a centralised machine would not be the best option.*

What if, you need to automatically **test** the **code** in **various environments** ?

*Different environments are to be set up in separate machines.*

*Jenkins distributed architecture* is the solution in such scenarios.

## Master-Slave architecture



Slave



GIT



Master



Slave



Slave



Slave

Jenkins uses **Master - Slave architecture** for managing distributed builds. In master node, you need to install Jenkins. While it can also execute build jobs, it handles all related tasks for build system -

- Schedule build job
- Dispatch build job to slaves
- Monitor the slave
- Report the build results

# Master-Slave Communication

*Slave node offloads work from the master.*

- It can run on various operating systems.
- Executes build jobs dispatched by master.
- Jenkins need not be fully installed on slave.

To operate, master and slave will establish bi-directional communication link like TCP/IP

## How to Setup Slave node ?

To setup slave node:

- Select Manage Node under Mange Jenkins and Click New Node
- Select the dumb slave or permanent agent option and Click Ok
- In the next window, enter details like the IP address and the user credentials
- Select the launch method and then Save.

## How to Configure Slave node ?

On completion of the steps mentioned in the previous card, the new node machine will be online and ready .

To execute the project in the slave node:

- Select 'Restrict where this job can be run' option under your 'project configuration'
- Mention the slave name in Label expression field.

## Jenkins Backup

### How to Backup Jenkins ?

The screenshot shows the Jenkins interface with the 'Backup manager' page selected. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'Credentials'. Below that is a 'Build Executor Status' section for 'master' showing 1 Idle and 2 Idle. The main content area is titled 'Backup config files' and contains 'Backup configuration' settings. It includes fields for 'Hudson root directory /var/jenkins\_home', 'Backup directory' (with a question mark icon), 'Format' set to 'tar.gz', 'File name template' with a placeholder 'backup...@date@.@extension@', 'Custom exclusions' (empty), and three optional checkboxes: 'Verbose mode', 'Configuration files (all) only' (which has a large red arrow pointing to it), and 'No shutdown'. Under 'Backup content', there are three checkboxes: 'Backup job workspace', 'Backup maven artifacts archives', and 'Backup fingerprints'. A 'Save' button at the bottom is highlighted with a red box.

Taking project backups are crucial in software delivery process.

In Jenkins, all **build logs**, **archives**, **configuration settings** are stored under the **Jenkins home directory**.

**To backup the home directory :**

- You can **manually copy** it to another location OR **Install backup plugin**
- If you choose to use the **backup plugin**, you need to trigger backup manually
- If **full** backup of **home directory** is **not needed**, use **ThinBackup** plugin to backup specific details, like **jobs**, **configuration**, **build history**.

## Best Practices and Conclusion

### Using Jenkins - Best Practices

Few *recommendations* for effective *utilization* of *Jenkins*

- *Secure* Jenkins.
- *Choose* the correct plugins based on your project need.
- *Schedule* jobs ensuring not all jobs are executed at the same time.
- *Monitor* disk space and clear or archive data to maintain space.
- *Break down jobs* to allow reuse of generic jobs.
- *Backup* is required for configurations and activity logs.

### Jenkins - Course Summary

We hope you have learnt basic concepts on Jenkins .

You should be able to create a simple Jenkins project.

To summarize, what you have learnt:

- Features of Jenkins
- Jenkins Installation
- Plugin Management
- Creation and Configuration of a simple project
- Jenkins Security
- Distributed Build
- Jenkins Backup

# Docker - Container Orcas

## Docker Course Introduction

In this course, you will learn the

- Basic concepts of Containers
- Docker Engine
- Docker architecture and workflow
- Docker end-to-end flow with example

Let us begin the course with the basic concepts of Virtualization (Virtual Machines) and Containerization (Containers) and then dive into the concepts of Docker.

## What is Virtualization?

**Virtualization** is a creation of logical object version in place of an actual version.

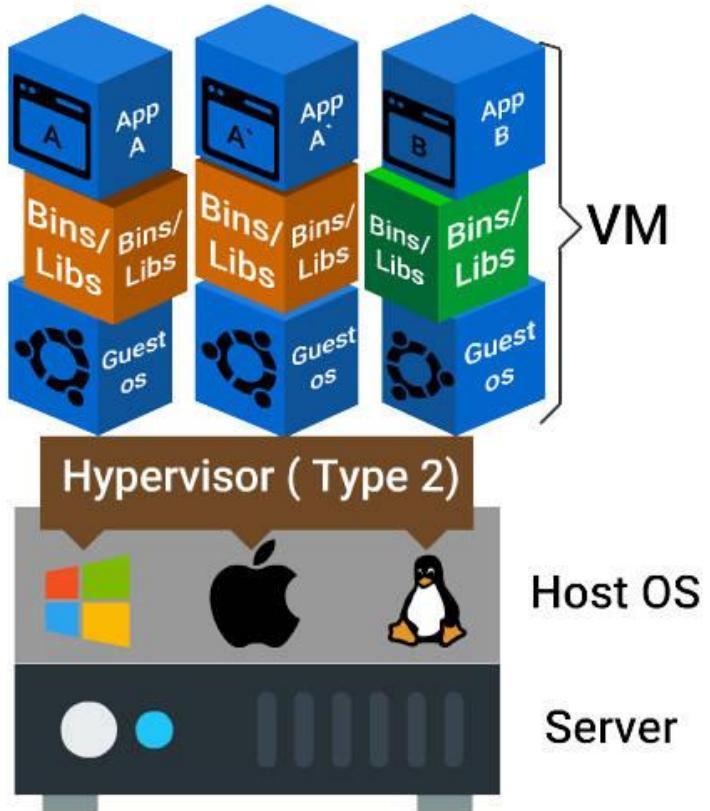
Few examples are virtual computer hardware platform, virtual storage, and virtual LANs.

**Hardware virtualization** means creating **virtual machine** that acts like a real physical computer with an OS.

For example, a **virtual machine** (VM) hosted on a computer with **Microsoft Windows** may behave like **Ubuntu** and Ubuntu supported software may run on the VM.

# Virtual Machines

---



This block diagram explains virtual machine configuration and how applications are deployed on VMs.

## Need for a Virtual Machine

Virtual machine setup has the following benefits.

- **Multiple operating systems** can be hosted on the same machine simultaneously with complete isolation.
- **Multiple VMs** can be deployed on the same physical box. This reduces the total number of physical machines.
- **Easy maintenance**, app provisioning, and quick recovery.

# Problems with Virtual Machine

- A lot of **wastage of resources** like ram, processor, disk space due to fixed space slicing for every application deployed. Hence this is not ideal for a large scale application developed using micro services.
- **Inconsistent computing environment** across the software delivery life cycle (Prod/Dev/QA).
- **Hardware failures** like malfunctioning and power supply loss will **stop all working servers** since many servers run on a single physical server.

## What are Containers?

**Containers are multiple isolated services** that are run on a single control host (underlying infrastructure) and they access a single kernel.

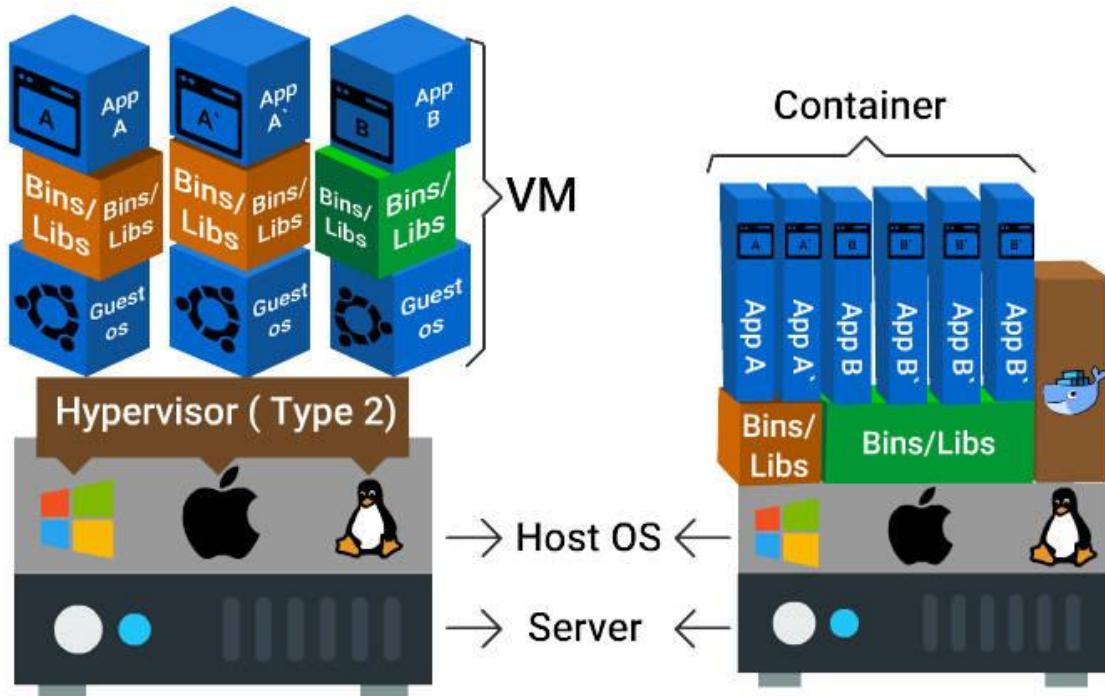
**Container based virtualization is an OS-level virtualization** method for deploying and running distributed applications without launching an entire VM for each application.

They **isolate applications** from one another.

# Virtual Machines vs Containers

## VMs vs. Containers

---



This block diagram clearly depicts the difference between VMs and Containers on how an application is deployed.

## Container - Benefits

- Improved portability
- Better performance
- Optimum RAM/disk space/cloud utilization
- Suited for agile environment
- Facilitates approaches such as micro services, continuous integration, and delivery.

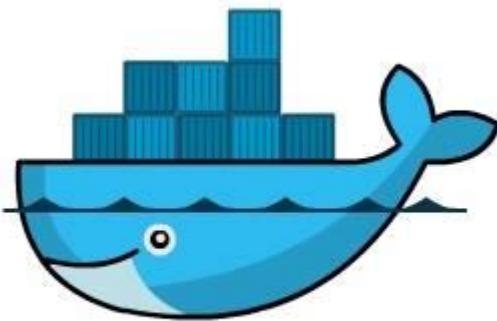
- Eliminates environment inconsistencies

## Benefits from Container Orchestration Tools

Container Orchestration tools like Docker Swarm, Amazon ECS, and Azure Container Service:

- Facilitate auto deployment
  - Scale application easily
  - Quickly push application from one environment to another
  - Enable automated rollbacks and backups
- Support

## Docker - Introduction



Docker is a tool intended to make the process of **creating, deploying and running applications** easier by using **container-based virtualization technology**.

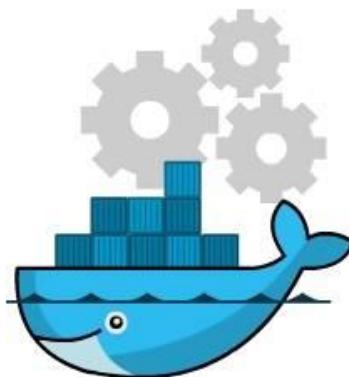
Docker is an **open source container technology** that provisions far more apps running on the same old servers compared to traditional VMs.

# Docker Engine

Docker engine is the Docker core component that is responsible for creating Docker Images and running them as services.

Let us learn in detail about Docker Images in the next topic.

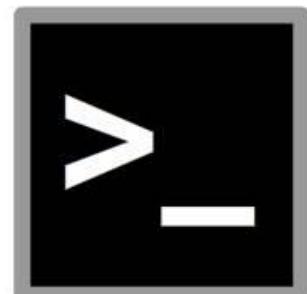
## Docker Core Components



Docker Daemon



REST API



Docker Client

Docker Engine Core Components:

- **Docker Daemon**

Continuous running program (daemon process) that manages the service and other docker objects tied to it.

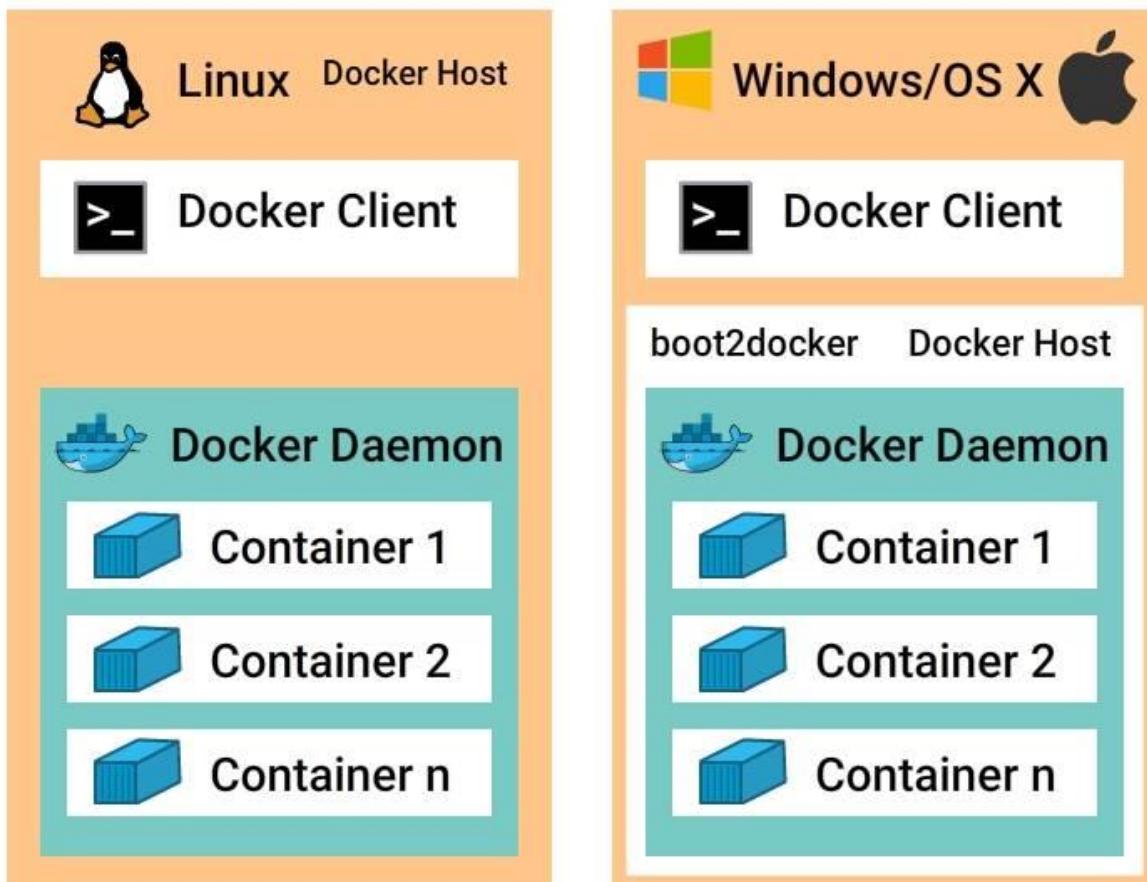
- **REST API**

Specifies **interfaces**, that programs can utilize to speak to the daemon and direct it what to do.

- **Docker Client**

CLI is utilized to interact with the daemon (docker command).

## Docker on Linux and non-Linux Kernel



### Docker hosted on Linux:

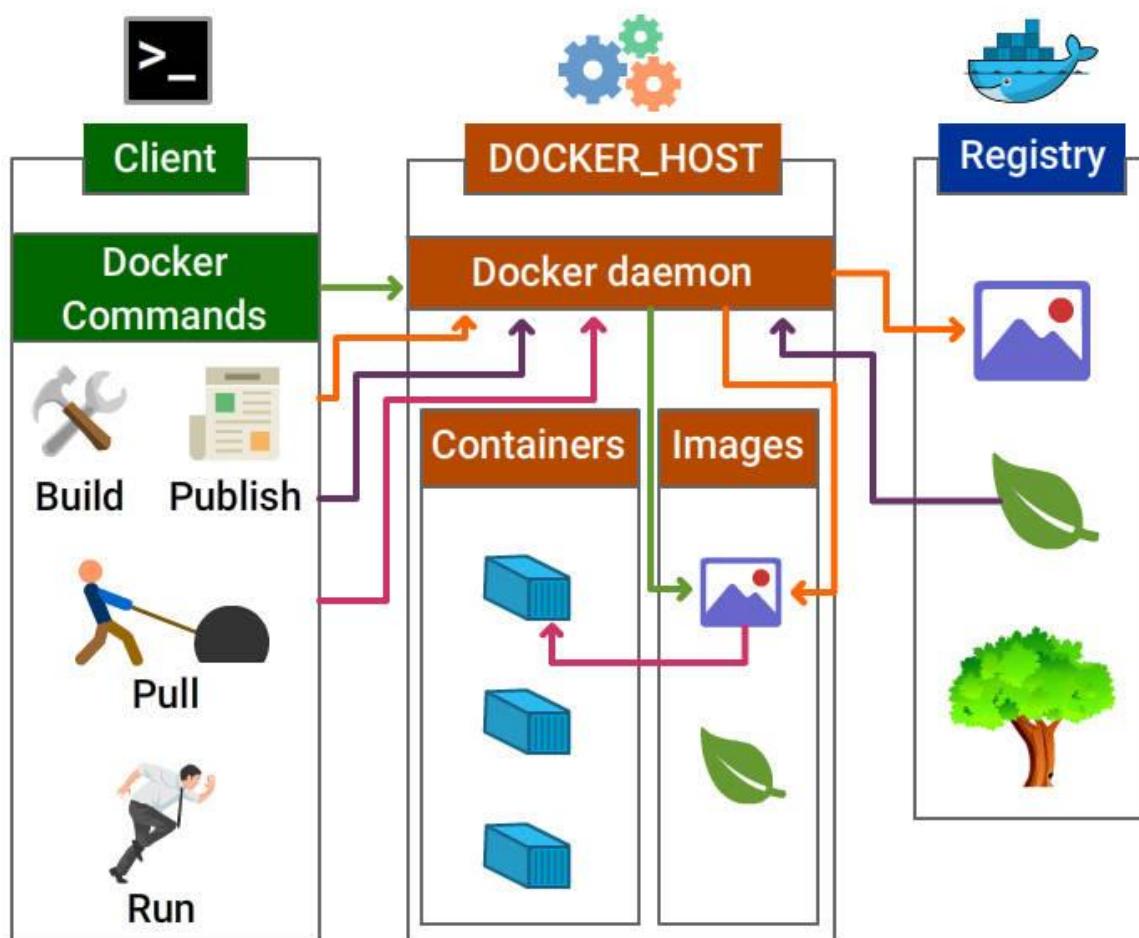
This requires just a Docker client and Docker daemon.

### Docker hosted on non-Linux:

- Docker desktop for mac uses [HyperKit](#) VM, which handles virtualization.

- Docker desktop for windows uses **Microsoft Hyper-V** to manage virtualization.
- 

## Docker Architecture



### Docker Architecture Block Diagram

Let us discuss in detail on the Docker components and workflow process in the upcoming cards.

# Docker Components

Docker components include

- Docker daemon
- Docker client
- Docker Objects
  - Images
  - Containers / Services
  - Network
  - Volumes
- Docker Registry

## Docker Daemon and Docker Client

### Docker Daemon

Docker daemon is the docker process that receives requests from docker client and is responsible for **managing docker objects** such as containers, images, networks, and volumes.

### Docker Client

Docker client **communicates with the docker daemon** through Rest API calls. A docker client can send a request to many docker daemons.

## Docker Image

Docker image is the collection of **all files, libraries, binaries and other dependencies** forming an executable software application, which can run everywhere without glitches.

An image is an **inert, immutable** file.

Here are the points to be noted about Docker Image:

- Docker image is **read-only**, i.e., the image and **its content cannot be altered**.
- Although the alteration is not allowed in Docker, we are allowed to **add the new layer with the changes**.
- After having many alterations, a docker image may be visualized as **several layers one above another**.

## Parent and Child Images

The layering concept in docker images leads to the addition of required capabilities efficiently by adding a new layer to the existing one resulting in a new image.

Hence, **the image has the parent-child relationship where the original image is termed as the base image upon which several child images are added**.

## Docker Service / Container

**Containers are run-time instances of Docker images that can be run using the Docker run command.**

The fundamental purpose of Docker is to run containers.

You can run a docker image to create as many docker containers as you want.

## Docker Container Lifecycle

This example explains the container life cycle stages and traversal from one stage to another.

## Docker Network

The concept of networking in Docker comes into account when working with Docker in a real time scenario at a large scale.

**Docker Networking** helps us to share data across various containers.

Host and containers in Docker are tied with **1:N relationship**, which means one host can command multiple containers.

## Modes of Networking

Various modes for networking is all about how we manage connections between containers.

- Bridge mode Networking
- Host Mode Networking
- Container Mode Networking
- No Networking

## Docker File

Dockerfile is a **script**, formed of different **arguments** and **commands** (instructions) listed successively to automatically execute actions on a base image to form or create a new one.

A Docker File is a simple text file with instructions on [how to build your images](#).

## Docker Registry

Docker Registry (Docker Repository) is a **storage house** for the Docker Images. It can be accessed publicly or privately by developers across the world.

- Docker images can be **sent to registry** by using [docker push](#) subcommand.
- Docker images can be **downloaded** from the registry using [docker pull](#) subcommand.

Following are the places where Docker registry can be hosted:

1. **Docker Hub**
2. **AWS Container Registry**
3. **Google Container Registry** and lot more

# Docker Hub

**Docker hub is one of the repositories of images which can be accessed at [index.docker.io](https://index.docker.io).**

Docker Hub is the official repository by Docker development community. Any third party images can be pulled from the repository.

e.g.:

```
docker pull thedockerbook/helloworld
```

## Docker Storage

### Storage drivers

Container layer contains a **very thin writable layer**, unlike images that have read only layers.

Each container has its storage layer whereas they **share the read only image layer** across containers in the same host.

Docker uses **storage drivers** that will manage the data using copy-on-write mechanism.

### Copy-on-write mechanism

Docker engine **does not copy the whole image** when we try to launch it. Instead, it uses Copy-on-write mechanism by which it uses a single copy of shared data until the data within the image is modified.

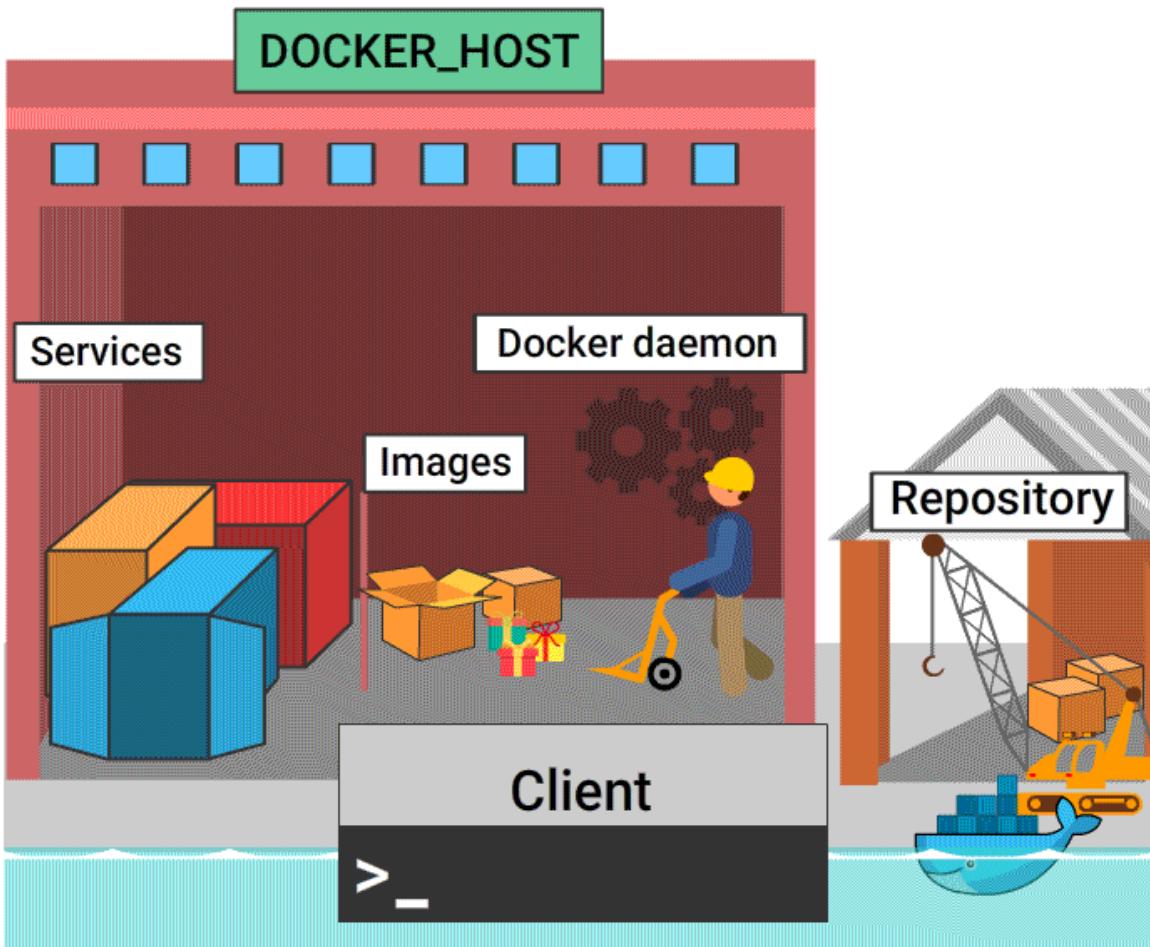
This saves a lot of disk volume, and the startup process is quick.

### Volumes

Volume is a **directory mounted** in the container that is created using docker command.

They are used to share data between containers by using the same volume across various containers.

## Docker Workflow



Docker workflow includes the following components:

- Docker Image - *Read only template that stores the application and environment.*
- Docker Container - *Runtime instance of a docker image*
- Docker registry - *Public and private repositories to store images*
- Docker File - *Automates Image construction*
- Docker file Compose - **Compose** is a tool that can be used to manage multiple containers containing different applications.

## Docker Workflow - In Detail

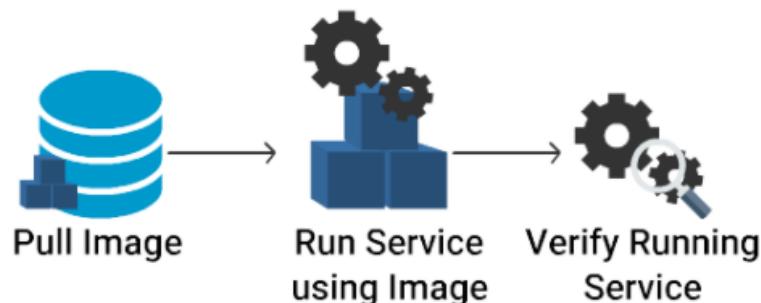
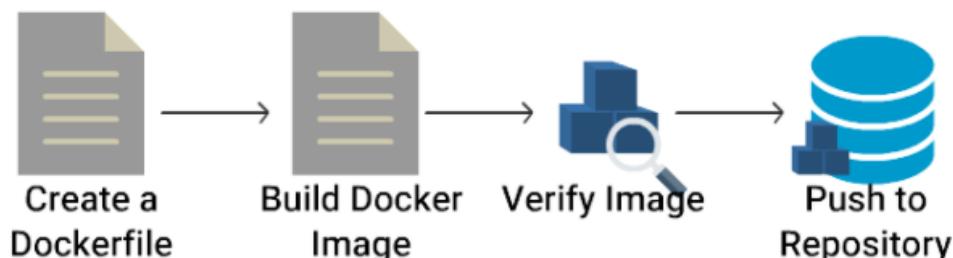
### Part 1

- Create a **docker file** that includes details on the base and child images to be built.

- Build the docker file using `docker build` command and `tag` a name to the image.
- Verify if the image is built successfully using `docker images` command and Run `docker inspect` command to view the complete details of the image.
- Now the image is ready, push the same to the image repository using `docker push` command.

## Part 2

- Pull the newly created image from the repository using `docker pull` command.
- Run the image using `docker run` command or using Dockerfile `compose`.
- Now you can verify the running container using `docker ps` command.



---

## Practise Docker Commands

Now that you have understand about docker architecture and its components, it is time to practice some of the docker commands.

For this purpose, you can have docker installed on your local machine.

### Docker Basic Commands

Here are few **basic docker commands**.

Check the version of Docker:

```
docker version
```

Check the detailed information on the running/stopped containers:

```
docker info
```

Docker images can be downloaded from **Docker hub using docker commands**.

Lets **pull an image from docker hub** using pull command.

Download a image from docker hub

```
docker pull <<image name>>
```

```
e.g. docker pull nginx
```

### Docker Commands - Images

Verify the downloaded docker images:

```
docker images ↵
```

View all the commands that were run with an image via a container.

```
docker history <<Image Name>> ↵
```

e.g. docker history nginx

## Remove Docker Images

```
docker rmi <<Image Name>> ↵
```

e.g. docker rmi nginx

Download and run an image in docker container using run command

```
docker run <<Image Name>> ↵
```

e.g. docker run --name nginxservice -d nginx

*--name --> to specify a name for the running service. In this example, it is nginxservice*

*-d --> to run the service in the background*

## Docker Version Tag

You must have noticed that on pulling the image from the Registry, the tagged version of the image is displayed:

```
> docker pull hello-world  
Using default tag: latest
```

For example, `docker pull busybox:1.24` will download the corresponding version:

## Docker Search

We can also search for the images in the Docker Hub registry by using `docker search` subcommand. Let us search for `ubuntu` images in the Docker Hub, and limit the search result only to 20 because we have more than 2000 images on Ubuntu:

```
docker search ubuntu | head -20
```

# Docker Commands - Container

Now lets run list of commands on the service/ container

List running containers

```
docker ps
```

Know the IP address of the running container:

```
docker inspect <Container Name>
```

e.g. docker inspect nginxservice

Print the stats for a running Container

```
docker stats <<Container Name>> ↵
```

e.g. docker stats nginxservice

Pause the processes in a running container

```
docker pause <<Container Name >> ↵
```

e.g. docker pause nginxservice

Unpause the processes in a running container

```
docker unpause <<Container Name >> ↵
```

e.g. docker unpause nginxservice

Kill the processes in a running container

```
docker kill <<Container Name >> ↵
```

e.g. docker kill nginxservice

Start the same container:

```
docker start <<Container Name>> ↵
```

e.g. docker start nginxservice

Stop the running container

```
docker stop <<Container Name >> ↵  
e.g. docker stop nginxservice
```

List all containers (This includes containers in all states):

We will be able to see the container we just stopped listed here.

```
docker ps -a
```

Delete a container:

```
docker rm <<Container Name >> ↵  
e.g. docker rm nginxservice
```

To remove all stopped containers:

```
docker container prune
```

**Note: Instead of using the Container Name, all the above commands can be executed with the container id as well.**

Export a container

```
docker export <<Container Name>> <<file_Name>>.tar ↵  
e.g.
```

Lets run a service using docker run command.

```
docker run --name newnginxservice -d nginx
```

```
docker export newnginxservice > test.tar
```

Import a container

```
docker import <<Remote URL/Image Name.tar>> ↵  
e.g. docker import test.tar
```

## Docker daemon Commands

Stop Docker daemon process

```
service docker stop
```

## Start Docker daemon process

```
service docker start
```

You may not be able to try these 2 commands since you would not have access to root on Katacoda playground.

## Diagnose Run Issues

In case you are having a problem with downloading the images and running them, please follow these steps to check whether the docker service is running on your system or not:

- Check the running status of docker:

```
service docker status
```

- Restart Docker service in your system:

```
service docker restart
```

## Let Us Install it!

**Looks like you are excited to dive into the Docker World! Aren't you?**

**Let us quickly explain you the procedure to install Docker in your system and get a quick hands-on with the basics.**

## Docker Installation

Prerequisite for installation of Docker on Linux are:

- 64-bit architecture Linux
- Linux kernel must be 3.10 or later

**Here are the steps for installing the community edition in Ubuntu 16.04:**

- Add the GPG key for the official Docker repository to the system:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- Now add the Docker repository to APT sources:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

1. Now update the package database with the Docker packages:

```
$ sudo apt-get update
```

2. Check for the policy:

```
3. $ apt-cache policy docker-ce
```

4. Install Docker:

```
5. $ sudo apt-get install -y docker-ce
```

6. Check the installed version of Docker:

```
$ sudo docker --version
```

**Congratulations!** You have successfully installed Docker version 17.03.0 community edition.

## Installation Using Automated Script

**Are you feeling this procedure lengthy? There is a shortcut to this process.**

Just run the below command to install Docker

- **curl command**

```
$ sudo curl -sSL https://get.docker.io/ | sh
```

# Uninstall Docker CE

This command is used to uninstall Docker CE package in Ubuntu machine.

```
$ sudo apt-get purge docker-ce
```

## Docker Example - Step by Step

### Hands-on scenario

***Welcome to the Docker challenge, your task is to follow the below steps and complete them successfully.***

\*Perform the following actions described below by executing the respective commands. Open terminal and execute the commands. \*

1. Check the version of Docker.
2. Check the detailed information on the running/stopped containers.
3. Download tomcat:latest image from docker hub.
4. Verify the downloaded docker images.
5. View all the commands that were run on the tomcat:latest image (check docker image history).
6. Remove tomcat:latest docker image.
7. Download and run an nginx:latest image in the docker container using the run command, name the container nginxservice.
8. Pull busybox:1.24 docker image.
9. Search for ubuntu images in the Docker hub, and limit the search result to 20.
10. List all containers.
11. Identify the IP address of the running container 'nginxservice'.
12. Print the stats for a running container 'nginxservice'.
13. Pause the processes in a running container 'nginxservice'.
14. Start the processes in a running container 'nginxservice'.
15. Kill the processes in a running container 'nginxservice'.
16. Start the same container 'nginxservice'.

17. Stop the running container 'nginxservice'.
18. List all containers (including the containers in all states).
19. Export the container 'nginxservice' and name the tar as 'test.tar'.
20. Delete the container 'nginxservice'.
21. Remove all stopped containers by pruning them.
22. Import the container 'test.tar'

- Create an nginx Dockerfile with index page that returns the string 'Welcome to fresco'
- Build the image and name it 'nginximage', run the image and expose it on port 80 and name it 'nginxcontainer'
- Start a container registry image using following command `docker run -d -p 5000:5000 --restart=always --name registry registry:2`
- Write the following content to /etc/docker/daemon.json file to avoid http security error. Replace the string with registry container ip.

```
{
  "insecure-registries" : ["replace-with-registry-ip:5000"]
}
```

- Restart docker service using the command 'sudo service docker restart'
- Push the nginximage image to registry. (Hint: need to tag the docker image with the registry ip and port)

## Ocean in the Capsule!

In this course, you studied the following topics:

- Docker Architecture
- About Dockerfiles, Containers, and Images
- Running and playing with containers
- Creating and pushing Docker images
- Export and Import of Images

# Kubernetes K8s - The Basics

## Prelude

### Getting started

Before getting started with this course there are a few concepts that you need to know like Containerization, Docker and Docker Swarm.

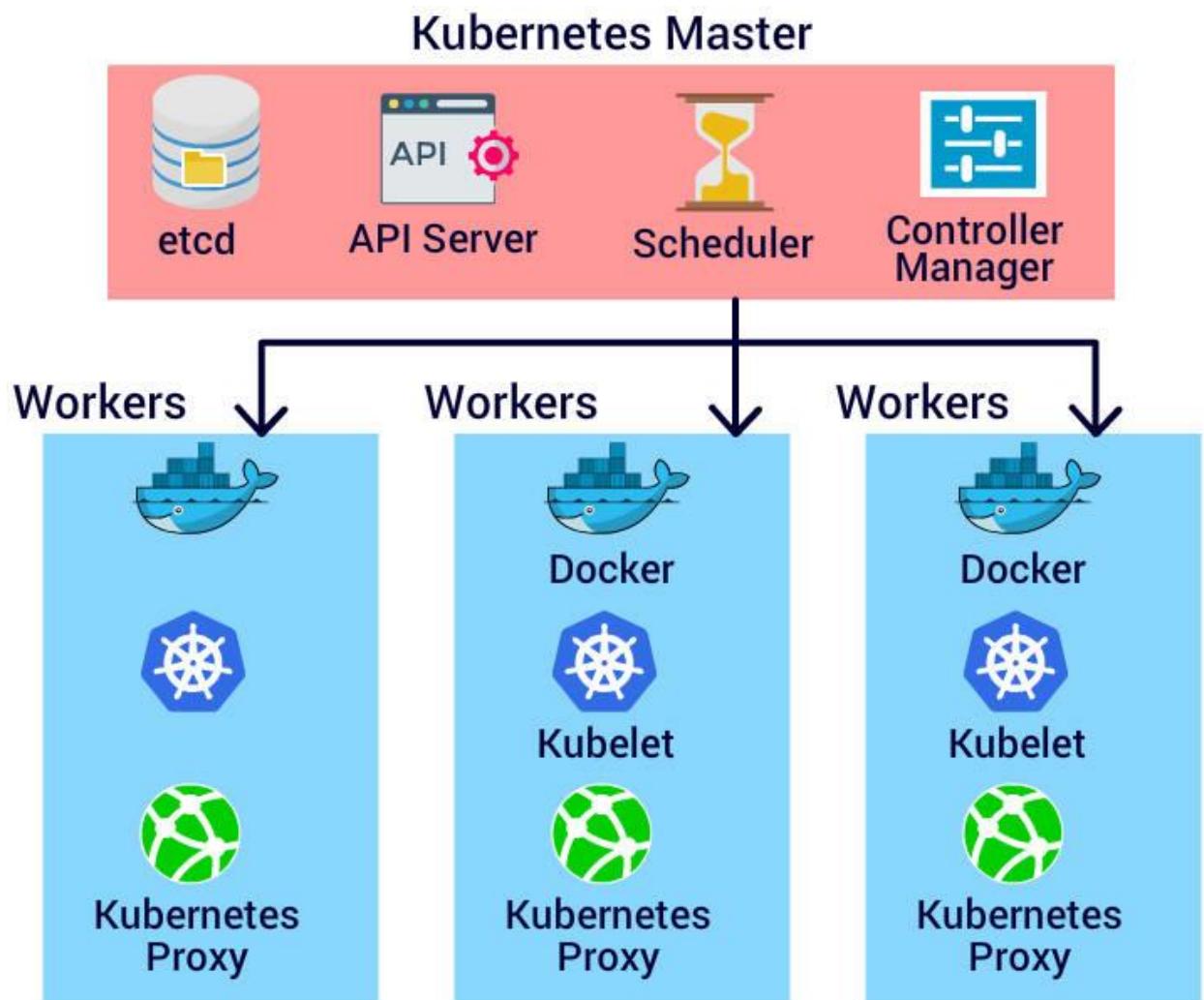
If you aren't familiar with any of these , visit the courses below and you are all set to learn Kubernetes.

- [Docker-container Orcas](#)
- [Docker-Magneta](#)
- [Docker Swarm](#)

### What is Minikube?

- **Minikube** is a light weight version of Kubernetes that can run on a local machine or a local VM.
- It is used for learning, evaluation and development purpose only.
- Minikube has an actual Kubernetes code, hence it runs all the Kubernetes CLI commands.
- Minikube **cluster** has only one Master and one worker node.

### Kubernetes Architecture



The basic components of **Kubernetes** architecture are **Master node**, several **Worker Nodes** and key value store called **etcd** .

A Cluster in Kubernetes is a group of computers(physical or virtual) connected together to perform a task. The cluster consists of one or more Master nodes that control the Worker nodes .

## App Management commands

**autoscale** - This command creates an autoscaler that automatically decides the number of pods that run in a kubernetes cluster.

**create** - The create command is used to create Kubernetes objects from the corresponding YAML or JSON file.

**delete** - Deletes Objects by file name, stdin, resource and names.

**edit** - Edits a resource from the default editor.

**get** - The get command gets specific resources or objects from the Kubernetes and displays them

Ex: `kubectl get deployments` This command will fetch all the deployments in Kubernetes cluster.

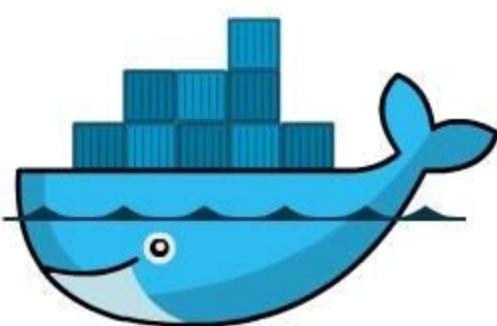
## Containerization

**Containerization** is a **virtualization** technique in which the application and all its dependencies are packed into a single isolated package called a **container** and can be run on the host OS, virtual machines and on cloud.

*Need for Containerization*

Every business depends on software and every software needs its own set of dependencies. Containerization helps to bundle the applications and its dependencies into an isolated package that can run on any platform or OS.

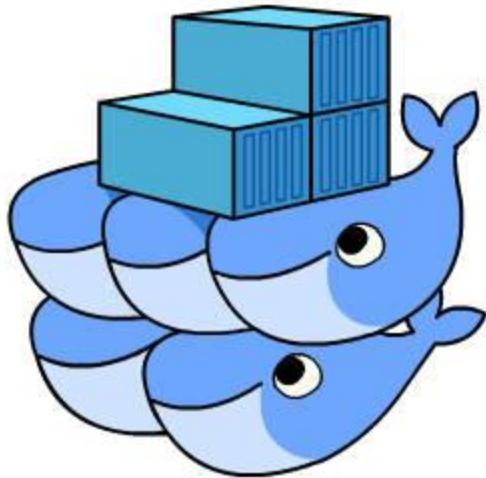
## Dockers



There are many container systems available out there to **create**, **run** and **deploy** containers but **dockers** is the ubiquitous leader. It is so popular that it has become the synonym for containers. Some other famous container systems are

- rkt
- LXC(Linux containers)

# Docker Swarm



**Docker Swarm** is an inbuilt docker container orchestration and management tool. It is used for creating the node clusters, scaling up the application and rolling changes and updates.

## **kube-scheduler**

kube-scheduler checks for newly created pods that haven't been assigned to any worker nodes and assigns them to worker nodes, it schedules work to the Worker nodes. It considers many factors for scheduling such as

- resource requirement and availability
- data locality
- hardware/software policy constraints
- deadlines

## **kube-controller-manager**

It manages the Kubernetes cluster and helps maintain the desired state of the cluster. It is not a single component, it is made of other controllers such as node controller, Replication controller, end-points controller, service-account and token controller.

# Container Orchestration

**Kubernetes** is a Container Orchestration System but what is **container orchestration** and why do you even need it.

### *Need for Container Orchestration*

A useful application depends on other applications/functionalities to complete a task. It's good to keep logically distinct applications/functionalities separate , it helps in developing, testing , scaling and deployment. But these applications need to interact with each other to become a useful software system. This is where **container orchestration** comes into picture, it runs and manages multiple containers containing different applications or different functionalities of a single application and establishes communication between them.

## What is Kubernetes ?



- Kubernetes is an open source **container orchestration** platform, used for deploying, scaling, updating and managing containerized applications.
- It was developed by Google and is now maintained by CNCF(Cloud Native Computing Foundation).

## What's K8s?

**K8s** is a short form for **Kubernetes**. In **K8s** the **8** represents the eight letters **ubernete** of the word **Kubernetes**.

## K8s v/s Docker Swarm

### Kubernetes vs. Docker Swarm

FEATURES	Kubernetes
Installation & Cluster configuration	Complicated & time consuming
GUI	GUI available
Scalability	Scaling up is slow compared to Swarm; but guarantees stronger cluster state
Load Balancing	Load balancing requires manual service configuration
Updates & Rollbacks	Process scheduling to maintain services while updating
Data Volumes	Only shared with containers in same Pod
Logging & Monitoring	Inbuilt logging & monitoring tools

Why use **Kubernetes** when **Docker** already has an inbuilt container orchestration tool called **DockerSwarm**? Well, the biggest advantage of using **Kubernetes** is that it was developed by **Google** and is the most popular **Container Orchestration** system.

# Getting Started with Minikube

## Installing Minikube

Now let us install Minikube on your local Linux machine. but before installing here is a list of prerequisites for Minikube to run on your local system.

- Virtual Box must be installed (If you are not able to install first activate virtualization in the system BIOS and try)
- Kubectl CLI tool must be installed in your system to run the Kubernetes CLI commands
- Dockers must be installed to create containers.

## Running Kubectl and Minikube

To see if kubectl CLI tool is installed in your system and to check the version just type the following.

```
$ kubectl version
```

To check if minikube is installed in your system type

```
$ minikube version
```

## Installing Kubectl

Before installing **Minikube** on your local machine and running Kubernetes you have to install **kubectl CLI tool**.

The steps for installation are:

```
$sudo apt-get update  
$sudo apt-get install apt-transport-https  
$curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -  
$sudo touch /etc/apt/sources.list.d/kubernetes.list  
$echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list  
$sudo apt-get update  
$sudo apt-get install -y kubectl
```

# Installing Minikube

Minikube is a lightweight version of Kubernetes that can run on your local machine. The minikube kubernetes cluster consists of only one master and node. The steps to install minikube are :

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.28.1/minikube-linux-amd64  
chmod +x minikube  
sudo mv minikube /usr/local/bin/
```

## Kubernetes-Architecture

### The Master Components

Every Kubernetes cluster contains at least one Master node. The Master node manages the cluster , it controls the Worker nodes in the cluster and makes decisions regarding scheduling, scaling and updating the application.

You can communicate to the master node via kubectl CLI or the APIserver.

The different components of a Master node are

- Kube-apiserver
- etcd key-value store
- kube-scheduler
- kube-controller-manager

**NOTE:** Kubernetes master can run on any machine but all master components must run on the same machine and no nodes should run in that machine.

### Master Components

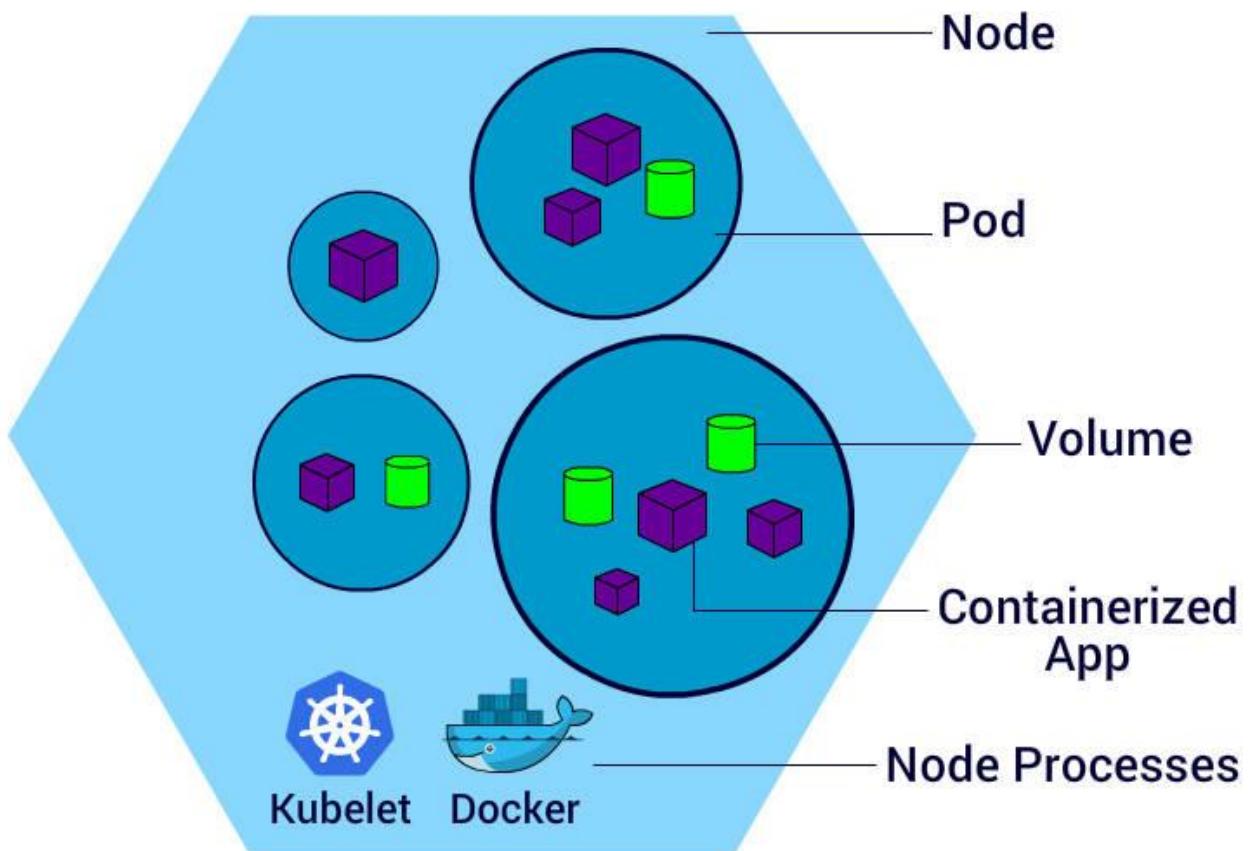
#### **kube-apiserver**

The kube-apiserver connects to the Kubernetes API and helps to perform all the administrative tasks given by the user and stores the cluster state in etcd key-value store after all the executions are done . It is the front end on Kube control plane. It is scaled horizontally i.e it scales by increasing the number of instances.

### etcd

etcd is a highly reliable and distributed key-value store which is used to store the data regarding cluster state. It can be part of the master node or can be external in which case the master node connects to it.

## Worker Node Components



Every Kubernetes cluster has a number of worker nodes. These worker nodes are also called as **nodes**. The nodes are controlled by the Master and run the application or a part of an application using **pods**.

The different node components are

- kubelet
- kube-proxy
- container runtime

**kubelet** is a worker node component that runs on every worker node in a cluster and is used to communicate with the master node. It runs containers inside a **pod** according to **pod-spec**.

A pod-spec is YAML or JSON file that contains information regarding which containers should be run in the **pod**.

**Pods** are group of similar type of containers, you will learn more about them later.

A kubelet interacts with the container-runtime with the help of CRI(container runtime interface). A CRI consists of two parts Image Service and Runtime Service. The Image service deals with the container images and the runtime service deals with the running of containers in a pod.

- **kube-proxy** It is a kubernetes network proxy service that runs on every node, it is used to connect the application to the external world/environment. Instead of directly connecting to the pods to interact with the application **Services** are used.

A **service** groups related pods and is used for TCP and UDP load balancing.

- **Container Runtime** Every container must have a container runtime, it is used to run and maintain containers in a node. Container runtime are tools or software that are used to create and run containers. Eg: dockers and rkt

## Nodes in Kubernetes

Nodes are the worker machines in the kubernetes cluster that are controlled and managed by the **Master**. Nodes can be local machines or the VMs. Nodes contain **Pods**.The Services provided by the nodes are

- Node Status
- Management
- API objects

**Node Status** contains the information about the node such as

1.**Addresses**: The addresses contain the

\*\*Hostname\*\*: The hostname provided by the node's host kernel.

**External IP**: the IP address that is used to communicate with the node outside of its cluster.

**Internal IP**: the IP address that is used for communication within the cluster.

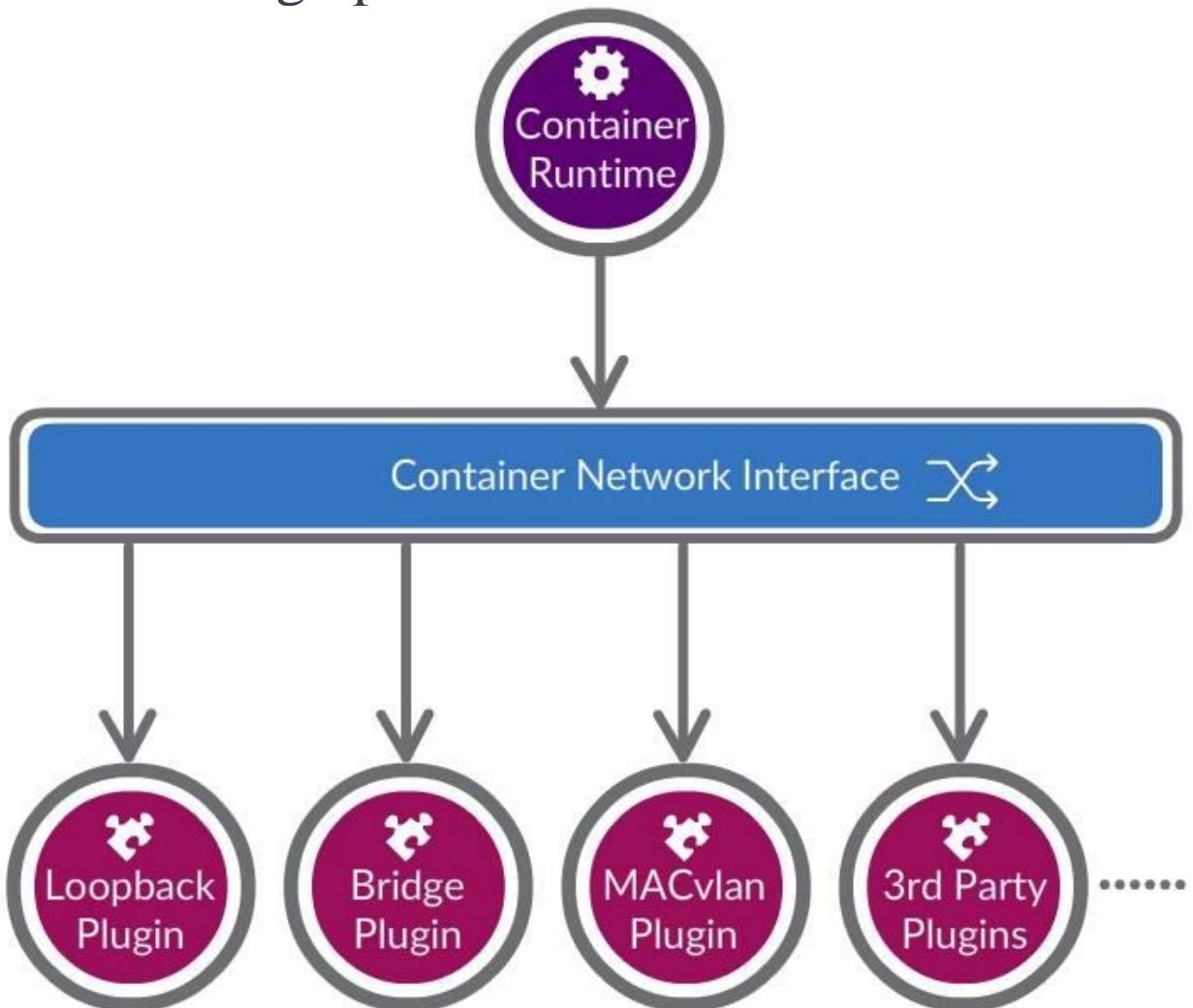
2. **Condition**: The condition field is used to describe the status of the running nodes.
3. **Capacity**: The capacity field describes the resources available to the nodes and the maximum number of pods that can be scheduled onto the node.
4. **Info**: This field provides the generic information about the node, such as kernel version, Kubernetes version (kubelet and kube-proxy version), Docker version (if used), OS name. The information is gathered by Kubelet from the node.

## Node Management

**Management**: Nodes are not directly created by Kubernetes, they are externally created or are already existing in the local machine. So Kubernetes just creates an object representing the node. After creation, Kubernetes checks whether the node is valid or not based on the **metadata.name** field. If the node is valid it is eligible to run **Pods** and other cluster activities.

**API Object**: Node is the top level resource in the Kubernetes REST API.

# Networking specifications



There are two types of container networking specifications

- CNM(Container Network Model)
- CNI(Container NEtwork Interface).

Kubernetes uses the CNI specification for container networking and communication.

The CNI assigns IP address to the pods.

# Communications

## *Inter Container communication*

Container runtime along with the host OS creates an isolated network entity called as **network namespace** for every container. Containers inside Pod share the network namespace and communicate via localhost.

## *Inter Pod Communication*

For pods to communicate with each other within the same node and with pods in different nodes, kubernetes doesn't allow Network address Translation(NAT). This is achieved by

- creating routable pods and nodes, using the Google Kubernetes Engine.
- Using Software Defined Networking, like Flannel, Weave, etc.

## Kubectl commands

### Kubectl Commands

kubectl is a Command Line Interface tool for running Kubernetes to create clusters and deploy applications. Before running kubectl commands kubectl must be installed in the local machine.

The basic commands of kubectl CLI

- **version** - Print the client and server version information for the current context
- **config** - Modify kubeconfig files using subcommands and seeing the connection details

### kubectl commands

The most important and frequently used commands in kubectl are the **run** and **create**

**run** - Creates a deployment or job to manage the created container(s).

**expose** - Searches for a deployment, service, replica set, replication controller or pod by name and uses the selector for that resource as the selector for a new service on the specified port. A deployment or replica set will be exposed as a service only if its selector is

convertible to a selector that service supports, i.e. when the selector contains only the matchLabels component.

## App Management Commands cont

`label` - Update the labels of Kubernetes Objects

`replace` - Replace a resource by filename or stdin. It uses YAML or JSON files to replace the existing resources

`rolling-update` - Perform a rolling update of the given ReplicationController.

`rollout` - Manage the rollout of a resource.

Valid resource types include:

deployments

daemonsets

statefulsets

`scale` - Set a new size for a Deployment, ReplicaSet, Replication Controller, or StatefulSet. Scale also allows to specify the conditions when the scaling should take place.

`set` - Configure application resources. These commands help you make changes to existing application resources.

## Working with Apps

`attach` - is used to attach the object to a process that is already running inside an existing container.

`cp` - Copy files and directories to and from containers.

`describe` - Describes the specified Kubernetes resource.

`exec` - Execute a command in a container.

`logs` - Print the logs for a container in a pod or specified resource. If the pod has only one container, the container name is optional.

`proxy` - Creates a proxy server or application-level gateway between the external world and Kubernetes API server.

`top` - Display Resource (CPU/Memory/Storage) usage.

## Cluster management commands

`api-versions` - Print the supported API versions on the server, in the form of "group/version"

**cluster-info** - Display addresses of the master and services with label kubernetes.io/cluster-service=true To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

**drain** - Drain node of all its resources like pods, services,etc for maintenance.

## The Dashboard

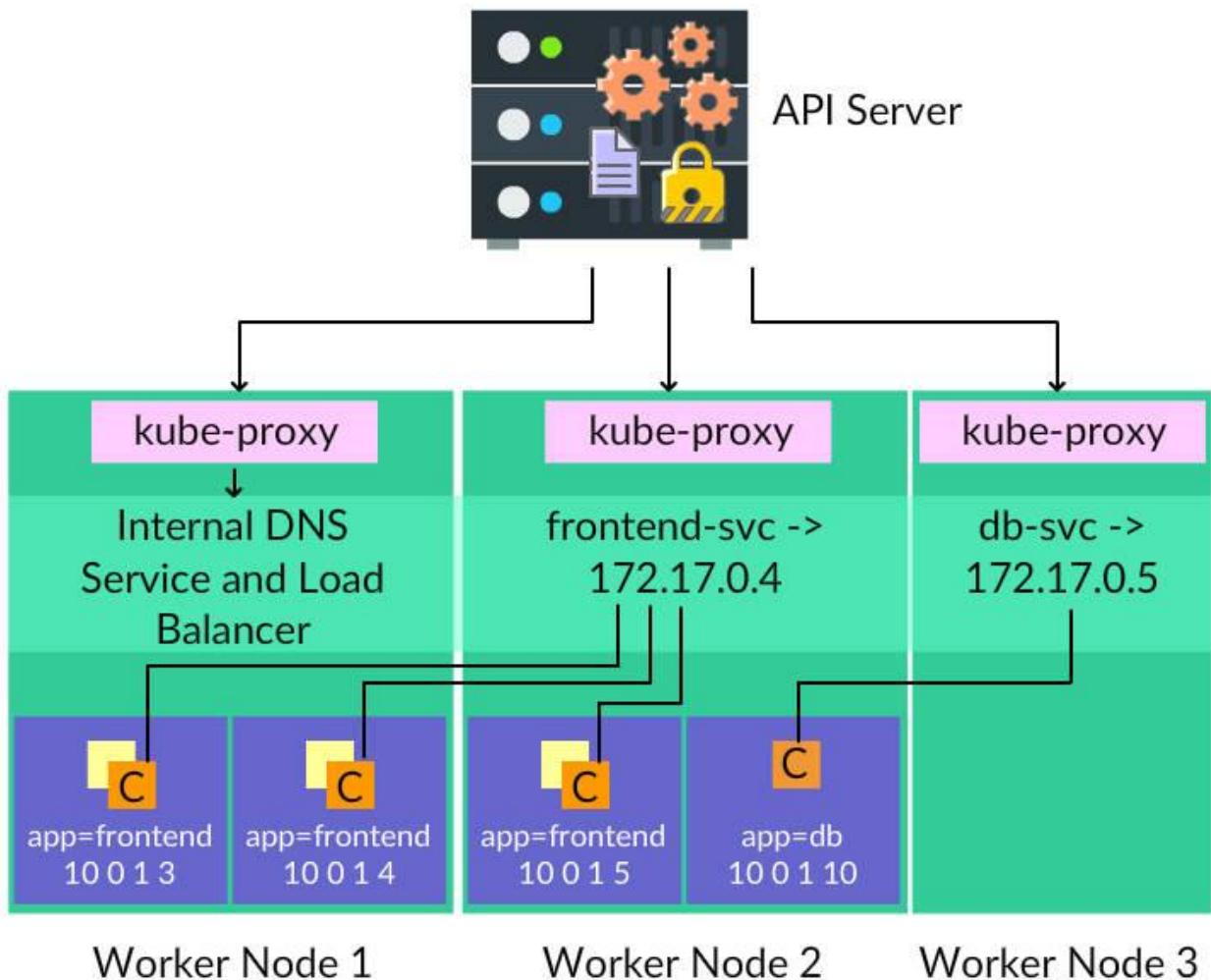
The screenshot shows the Kubernetes Dashboard's Overview page. On the left, there's a sidebar with navigation links: Cluster, Namespaces (selected), Nodes, Persistent Volumes, Roles, Storage Classes, Namespaces (dropdown set to default), Overview (selected), Workloads, Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, and Replication Controllers. The main content area has two sections: 'Discovery and Load Balancing' and 'Config and Storage'. Under 'Discovery and Load Balancing', there's a 'Services' table with one entry: kubernetes (component: ap, provider: kuber.., Cluster IP: 10.96.0.1, Internal endpoints: kubernetes:443 T, External endpoints: kubernetes:0 TCF). Under 'Config and Storage', there's a 'Secrets' table with one entry: default-token-pdf9q (Type: kubernetes.io/service-account-token, Age: 7 minutes).

- The Kubernetes Dashboard is GUI(Graphical User Interface) tool which is used to access Kubernetes, in our case the Minikube version of Kubernetes.
- It is used to interact with Kubernetes API server and to interact with clusters in Kubernetes.
- To open the Kubernetes dashboard execute the following command in Minikube terminal

```
$ minikube dashboard
```

The dashboard looks as shown in the image

## Kubectl-proxy



- The `kubectl proxy` command authenticates the Kubernetes API server on the Master node.
- By using this command we can access the dashboard from the localhost browser from the following  
url: `http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard:/proxy/#!/overview?namespace=default.`

# Kubernetes Objects

## Kubernetes Objects

Objects are used to maintain the desired state of a cluster.

**Object Fields:** Objects have two fields

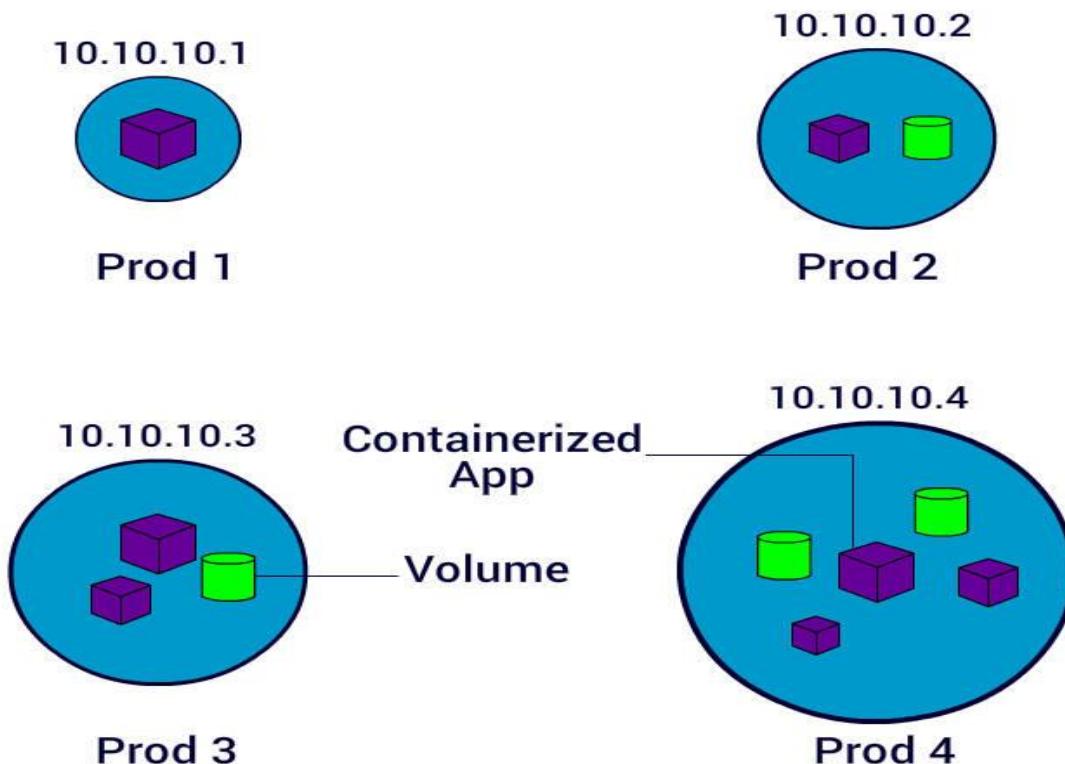
- Specs: Define the desired state of the object.
- Status: Defines the actual state of the object.

### Describing Object:

- Objects are described using a `.yaml` or `.json` file.
- It contains the following requirements
  - apiversion.
  - kind/type of object
  - metadata
  - spec, the desired state of the object
- `$kubectl create` command is used to create an object.

Every Object has an **UID**, a system generated string that uniquely identifies the object.

## Pods



**Pods** are logical group of similar type of containers. It is the smallest unit of deployment in Kubernetes.

- Pods do not have a well-defined lifecycle, they are recreated to maintain the desired state.
- Every container outside the pod has a unique IP address.

Pods allow data sharing and communication within the containers as they have a shared network namespace. Pods are transient in nature and cannot replicate automatically so we use controller manager components like replication controller with them.

## Services

**Service** in Kubernetes is used to defines a logical set of Pods and a policy by which to access them. Services are used for communications among different pods. Service is defined using **YAML** or **JSON** file. The set of Pods targeted by a Service is usually determined by a **LabelSelector**. We will learn more about **Labels** and **Selectors** later.

Services are used to expose the pods outside the cluster. Services are exposed in different ways using different ServiceSpecs such as

- ClusterIP
- NodePort
- Load Balancer
- ExternalName

## Kube-proxy

- Every Worker node runs a daemon process called **kube-proxy**
- The kube - proxy keeps track of API server of the master node for addition and deletion of Service endpoints.
- For every Service created in a worker node, the kube-proxy configures the IP tables rules to handle the traffic for its ClusterIP and forwards it to service endpoints
- When the service is removed, kube-proxy removes the iptables rules on all worker nodes as well.

# Service Discovery

Service discovery at runtime is an important concept because all communications from external world to Kubernetes occurs through Services.

- **Environment Variables:** When the Pod starts on a worker node, the kubelet adds a set of variables called Environment Variables in the Pod for all active Services. For example, if we have an active Service called redis-master, having Cluster-IP 172.10.0.12 and exposing Port -7307, the environment variables created will be as shown

```
REDIS_MASTER_SERVICE_HOST=172.10.0.12
REDIS_MASTER_SERVICE_PORT=7307
REDIS_MASTER_PORT=tcp://172.10.0.12:7307
REDIS_MASTER_PORT_7307_TCP=tcp://172.10.0.12:7307
REDIS_MASTER_PORT_7307_TCP_PROTO=tcp
REDIS_MASTER_PORT_7307_TCP_PORT=7307
REDIS_MASTER_PORT_7307_TCP_ADDR=172.10.0.12
```

- **DNS :** There are addon components in Kubernetes called DNS which creates and manages the DNS names for all the Services created. The format looks like `my-svc.my-namespace.svc.cluster.local`.

## Types of Services

In the previous card the different type of specs for Service creation has been discussed. In this card the types of Services are defined.

- **Cluster IP:** Cluster IP is the default Service type, in this type of Service there is virtual IP that can be used for communication only within the cluster.
- **NodePort:** In NodePort Service there is a virtual IP address along with a port number in the range of 30000-32767, that maps to the respective Service from all the worker nodes.

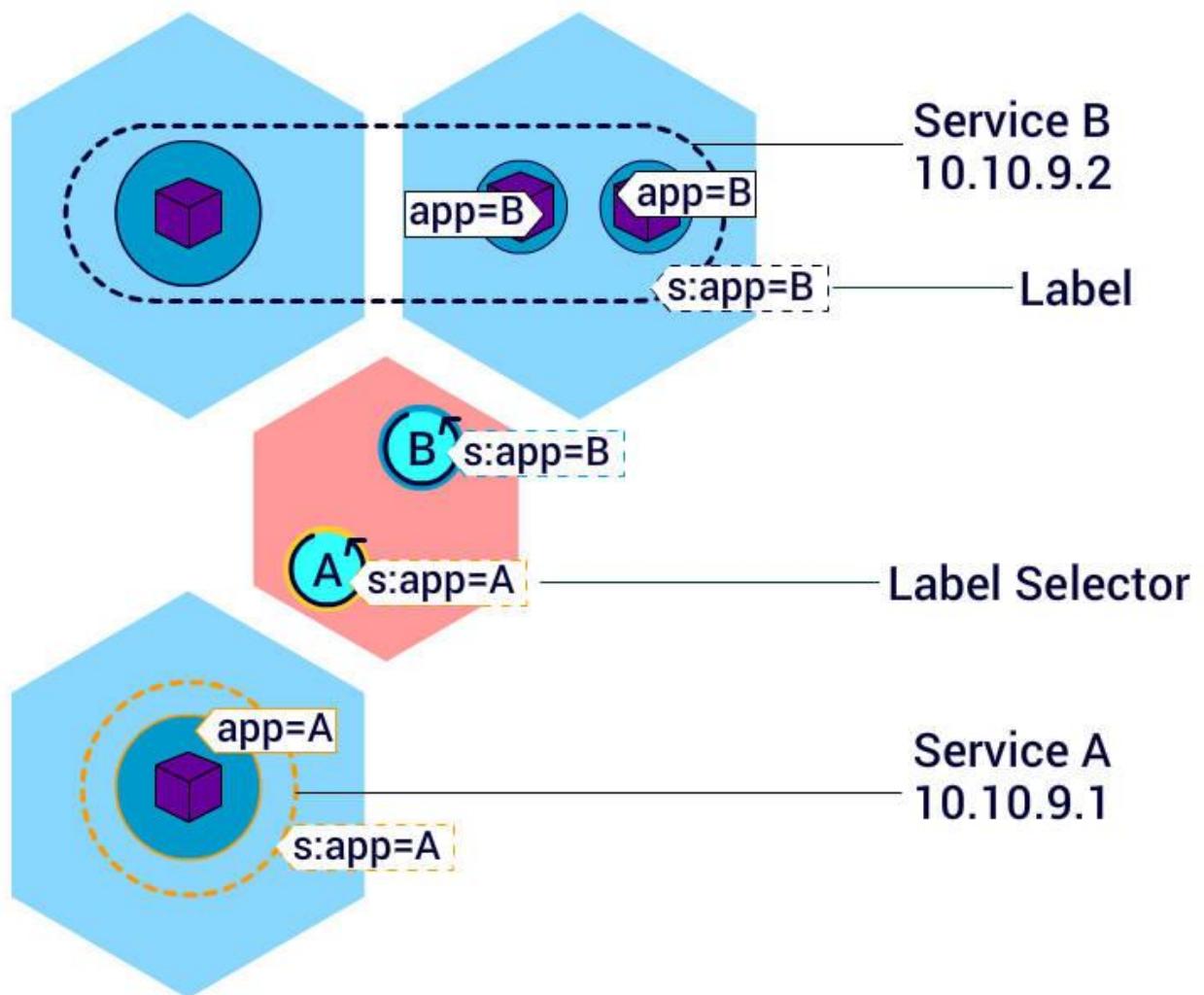
A default port number is selected by the Kubernetes while creating this type of Service. NodePort type Service is used when we want to communicate with the external world.

- **Load Balancer:** This type of Service uses external load balancer, the Cluster IP and NodePort for this type of Service is

automatically created and the load balancing is done by external load balancer.

- **External IP:** This type of Services are mapped to External IP address if it can route worker nodes. The External IPs are not managed by Kubernetes.

## Labels



**Labels** are key-value pair, that are used to organise and select attributes of an object. Ex

```
"metadata":{  
  "labels":{  
    "key1":"value1"  
    "key2":"value2"  
  }}}
```

Labels can be used to map organisation structure onto the Kubernetes Object. Ex:

```
"environment":"dev","environment":"qa"
```

## Selectors

**Selectors** are used to identify a sub-set of Objects. The kubernetes API supports two types of selectors.

- equality based
- set based

Empty label selectors select all the objects in the collection. A null label selector selects no objects

**Equality-based requirements** are used for identifying sub-sets of objects based on keys and values. The `=, ==, !=` are used for equality based selectors.

```
Ex: environment = production
```

**Set based Requirements** are used for filtering objects based on a set of values. There are three kinds of operators used `in, not in and exists`

```
Ex: environment in (production, dev)
```

## Replication Controller

- Replication controller is a part of the controller- manager and it is used to produce replicas of the pods because pods are transient in nature and cannot replicate them selves.
- If the number of Replicas are less they generate more replicas of the pods to maintain the desired state.
- If the number of replicas are more they are destroyed

## Deployment Objects and Deployment controller

### *Deployment Object*

- A deployment object is used to define the desired state of an application.

- Deployments can create new replica sets, delete other deployments and update deployments.

### *Deployment Controller*

- Deployment controller is a part controller-manager of the Master node.
- It is used to create and update pods and replica sets.
- Deployment Controller converts the actual state to desired state

## Names and Namespaces

**Names** are the client provided name given to a kubernetes Object in resource URL.

Ex; /api/v1/pods/example-pods/

**UID:** A system-generated string uniquely identifies a kubernetes Object.

**Namespaces:** Namespaces are virtual clusters within physical clusters. They are used when users are spread across multiple teams or projects. The names inside a namespace must be unique but can be similar across namespaces.

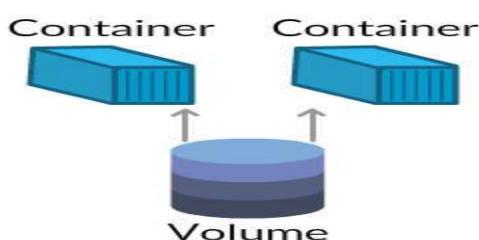
## Creating and Deleting Namespaces

Namepaces are created using YAML files. Ex:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <insert-namespace-name-here>
```

the `$kubectl create -f <namespace-name-here>` command is run in CLI to create namespace. \*

## What are Volumes?



Volumes are the persistent state data that can be stored on-disk or on other containers.

*Why use Volumes in Kubernetes?*

- You know that Pods are transient in nature and when the containers in a pod crash all the data inside them are deleted and new containers are created. The Data inside them is lost, to overcome this problem **Volumes** are used.
- Volumes can store data on a storage medium or in other containers.
- Volumes are placed inside the Pods and are connected to the containers.

## Persistent Volumes

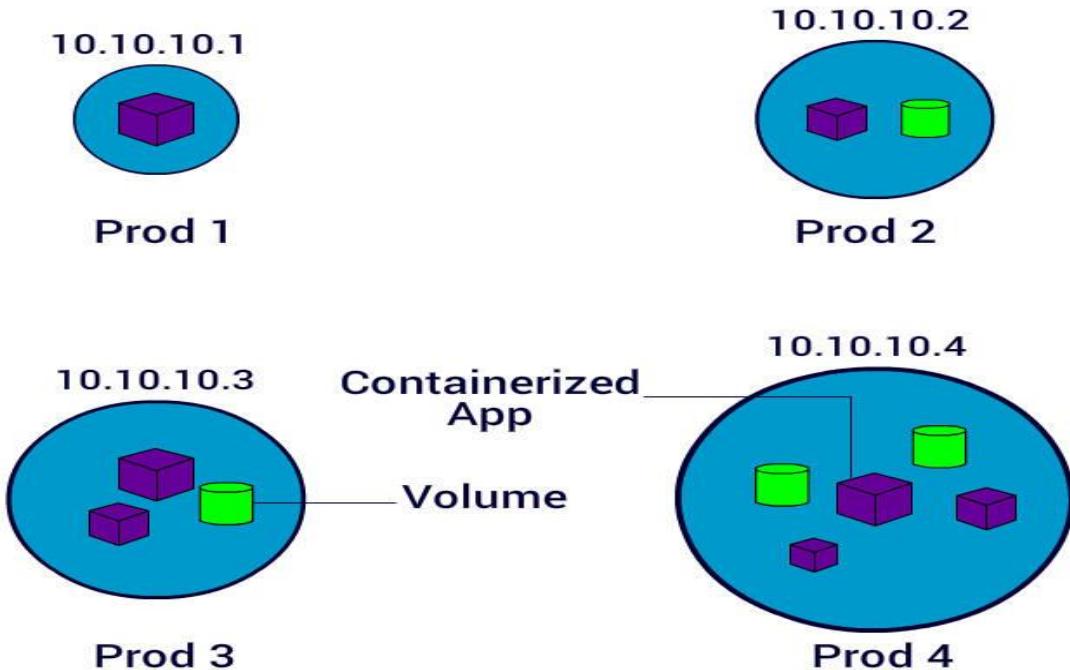
- A PersistentVolume (PV) is storage space created in the cluster by the administrator.
- It is similar to other cluster resources like the nodes. PVs are volume plugins like Volumes, but unlike volumes inside a container, they have a lifecycle independent of the lifecycle of the pod.
- The API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

**PersistentVolumeClaim (PVC)** are storage requests created by the user. Users request for Persistent Volumes based on their requirements. Once a suitable PersistentVolume is found, it is bound to a PersistentVolumeClaim.

Like Pods consume node resources, the PVCs consume Persistent Volume resources. Pods can request specific levels of resources (CPU and Memory).

# Workloads

## Pods



- A pod is the smallest unit of Kubernetes that can be deployed and managed. A pod is a collection of similar kind of containers that share storage and network and specification how to run them. A pod's contents are always co-located and co-scheduled and run in a shared context.
- Containers inside a pod share the same IP address and port space and can communicate with each other via **localhost**. They can also communicate with each other using standard inter-process communications like semaphores or POSIX shared memory. Containers in different pods have distinct IP addresses and cannot communicate by IPC without special configuration. These containers usually communicate with each other via **Pod IP addresses**.

## Pod Lifecycle

Pods are transient, pods are created, assigned a unique ID (UID), and scheduled to nodes where they remain until termination or deletion. If a node dies, the pods scheduled to that node are scheduled for deletion, after a timeout period. A pod is never rescheduled to a node

instead, a new pod with a new UID is assigned to the node. the following are the terms used to define a Pod Lifecycle.

- Pod Phase
- Pod Condition
- Container Probes
- Pod Lifetime

## Pod Lifecycle Terms

**Pod Phase:** The Pod phase is a simple yet high-level summary of where the pod is in its lifecycle. The different values of Pod phase are Running, Succeeded, failed and Unknown.

**Pod Condition:** is an array through which the Pod has to pass. Each element in **Pod Condition** has six possible fields.

1. `lastProbeTime`:
2. `lastTransitionTime`:
3. `message`:
4. `reason`:
5. `status`:
6. `type`

- PodScheduled
- Ready
- Initialized
- Unscheduled
- ContainersReady

**Container Probe** is a health check performed by the `kubelet` on a container. Handler is used to perform the check. There are three types of handlers

1. ExecAction:
2. TCPSocketAction
3. HTTPGetAction

**Pod Lifetime:** Pods are not deleted and destroyed automatically unless they are destroyed manually or by a controller but there is an exception to this when the pod with a phase succeeded or failed for more than a specific duration it will expire and automatically destroyed.

# Deployment and Deployment Controller

A Deployment controller is used to create and update Pods and ReplicaSets.

Deployment controller changes the actual state to the desired state as described in the deployment object. It can create new ReplicaSets, remove existing deployments and create new deployments.

**Deployment Controller uses:**

- Creating deployments for new Replicasets
- Declaring desired states of the pods
- Rolling back deployment to an earlier version
- Scaling up the deployment
- To pause or stop deployments
- To check the status of deployment
- Clearing old ReplicaSets

**Creating Deployments:** A deployment is created using a YAML file and then running the following command on Terminal.

```
$kubectl create -f <file_name>.yaml
```

To see the created deployments

```
$kubectl get deployments
```

## ConfigMaps

- ConfigMaps are used to define the configuration of a container image.
- They are used to separate the configuration details from the container image, so that you do not have to create multiple images for similar applications, instead you can create multiple ConfigMaps.
- ConfigMaps can be used by Pods and replication controllers to create multiple containers.
- There are two ways in which configMaps can be created
  - from literals
  - from files

## Types of ConfigMaps

*Creating configMaps files using literals*

To create configMaps using literals we use the `kubectl` CLI. Execute the following command to create a configMap.

```
$ kubectl create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2  
configmap "my-config" created
```

You can get the configuration information by executing the following command

```
$ kubectl get configmaps my-config -o yaml
```

*Creating the configMap using YAML file*

This method of creating a configMap is more preferred because it gives you full control to change the different components of the configMap. The YAML file structure looks like this

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: imageName  
data:  
  image data
```

To create the configMap file create a yaml file and run the `kubectl create` command Ex: `$ kubectl create -f imageName-configmap.yaml`

## Secrets

- Secrets are volume spaces that are used to store sensitive information of the application.
- They are created using the `kubectl create` command

```
$ kubectl create secret generic userpassword --from-literal=password=loginpassword
```

This creates a secret called userpassword that stores the login password.

- Secrets can be created manually by using YAML files and encoding the information. -Ex:

```
$ echo loginpassword | base64  
bG9naW5wYXNzd29yZAo=
```

The YAML file

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: userpassword
type: Opaque
data:
  password: bXlzcWxwYXNzd29yZAo=
```

## Deploying Application

### Hands-on Scenario

#### *Minikube installation*

```
sudo apt install docker.io -y
sudo systemctl unmask docker
sudo service docker restart
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb
sudo dpkg -i minikube_latest_amd64.deb
rm -rf minikube_latest_amd64.deb
```

*In this hands on you are going to start your kubernetes journey by getting your hands dirty with few basics*

#### **Environment Setup**

Check whether docker & minikube are properly installed and configured.

- Start Minikube and execute this command to sync host docker with minikube docker `minikube -p minikube docker-env` and `eval $(minikube docker-env)`

#### Step - 1

- Create a pod object using kubectl run command with google's sample image: `gcr.io/google-samples/kubernetes-bootcamp:v1` and expose it on port 8080, name the pod as firstapp.
- Check if the pod creation is successful by running the command: `kubectl get pod firstapp`

#### Step - 2

- Expose the application to the local VM by creating a Service object of type NodePort.
- Check if the service is created by running the command: `kubectl get svc firstapp`

### Step - 3

- Create another deployment using a 'YAML' file, create a deployment.yaml file that contains information of the number of replicas and the images to be used. Use an nginx image to deploy. Name the deployment as 'nginx'
- Check if the deployment is created by running the command: kubectl get deployment nginx

### Step - 4

- Create a NodePort type service using a 'YAML' file, create a service.yaml file that contains information of the type of service and the port numbers. Name the Service nginx-svc and use port 30080 for nodePort.
- Check if the service is created by running the command: kubectl get svc nginx-svc

### Step - 5

- Use kubectl exec command to get inside the running nginx pod and write the string 'Welcome to fresco nginx pod' to /usr/share/nginx/html/index.html file

## Summary

In this course you learned about

- The basics of Container Orchestration
- what is Kubernetes and how to install kubectl and MiniKube on local machine
- The architecture of Kubernetes and its different components
- The basic kubectl commands used to create objects
- How to deploy a simple application using kubernetes.

## What's next?



In the next course, you will learn more about

- Networking in Kubernetes, load balancing and scaling in Kubernetes
- What are stateful applications and how to deploy them?
- How to deploy Multi-tier application
- Monitoring and logging in Kubernetes and how to go serverless using kubeless framework.

# Ansible - Automation Sibelius

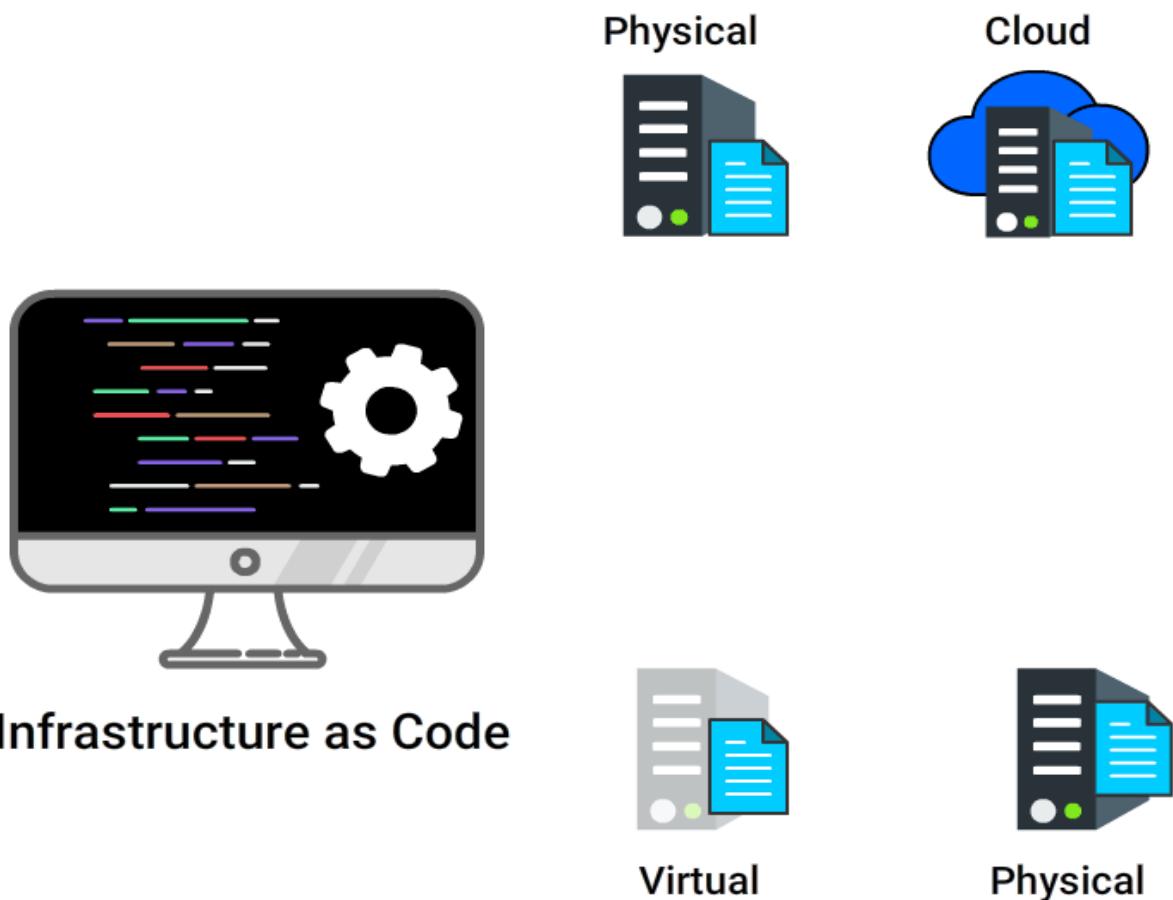
## Introduction to Ansible What are You Going to Learn?

We hope you have gained a good understanding of what is Infrastructure as a code. In this course, you will learn about

- Ansible and its benefits
- Ansible in Infrastructure as code (IaC)
- YAML format
- Few Ad\_Hoc commands
- Different parts of Ansible
- Role of Modules in Ansible
- How to write Playbooks
- How to Control flow of execution in Ansible
- Few points about configuration file (ansible.cfg)
- How to setup environment in local machine and online playground

By the end of this course, you will be in a position to write a few **ad-hoc commands** and **Playbook** of your own.

# What Is Infrastructure As Code?

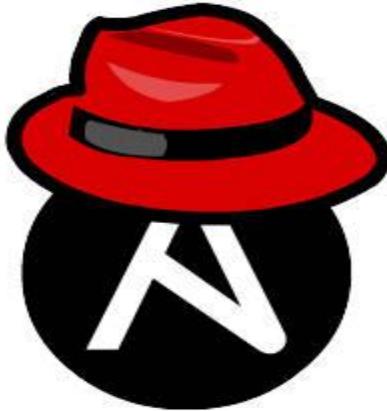


IaC means writing code for infrastructure i.e. your **systems and devices**, which are used to run Softwares, are to be treated as software and **defined using code**(which can be done using a high level or descriptive language).

*For example:* version control, testing, small deployments, use of design patterns etc.

**Configuration Management tools** are used to accomplish IaC.

# Why are You Here?



**Infrastructure as Code** can be achieved using **Ansible**, which is one of the popular **Configuration Management tools**. In this course, we will have you explore various capabilities of Ansible.

## What is Configuration Management?

**Configuration Management** is a process of establishing, tracking and controlling the **current design** and **build state** of the system (*software versions*).

It also ensures that **past records of system state** is easily and accurately accessible which helps in project management, audit purposes, debugging etc.

## Before Configuration Management

Consider you are planning for a **New Year Special Sale** in your e-commerce site. You need to

- **Scale up** your servers
- Then **configure** them (and all other old servers) for special new year sale
- This whole process would take **lot of effort and time**
- What if new configurations did not work as expected? Then you will have to **roll back to previous stable version**, which will add more work and subtract the profits and potential customers while in downtime.

# Configuration Management Tools



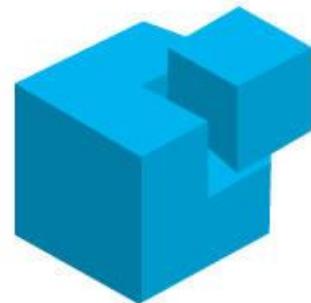
Ansible



Chef



Puppet



Saltstack

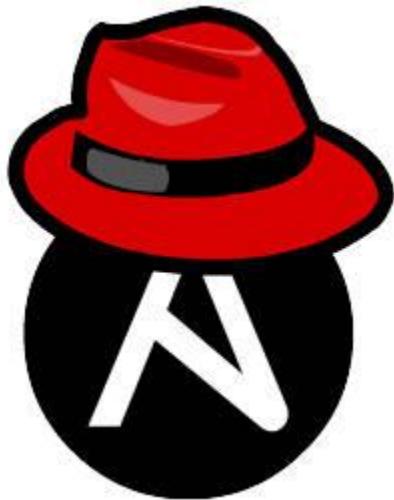
You need some kind of Configuration Management tool that can automate these tasks:

- *Roll back to stable version with zero downtime*
- *Provide you with constant computing environment throughout SDLC*
- *Automatically scale up or down depending upon traffic*

Some of the popular Configuration Management tools are: **Ansible**, **Puppet**, **Chef** and **Saltstack**.

You will now learn more about Ansible in upcoming cards.

# What is Ansible?



**Ansible** is an **open source software**, first released by Michael DeHaan in 2012 and owned by Red Hat.

It is used to **automate**

- *configuration management*
- *application deployment*
- *software provisioning* and other **IT needs**

## Benefits of Ansible

- **Simple:** Very easy to *install, setup* and *learn*. Written in **YAML** file which is pretty much like reading English.
- **Agentless:** Do not need to install any agents on target nodes.
- **Powerful:** It can model any complex IT workflow as it has *1100+ modules*
- **Efficient:** You can *customize modules*, using any programming language
- **Secure:** Uses SSH for connection

*Wondering about YAML language?? Read on to discover*

### Introduction to YAML Language

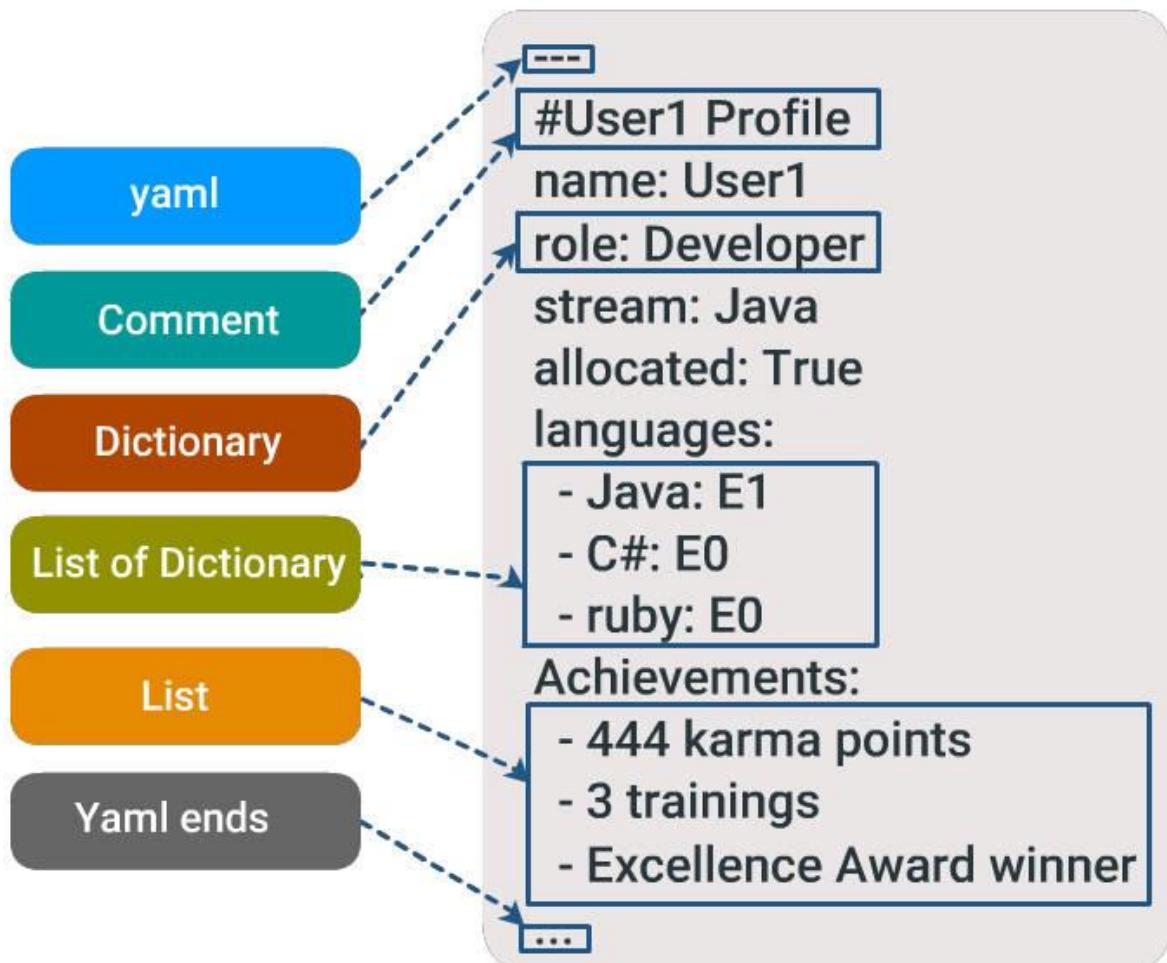
## What is YAML?

**YAML** stands for **Yet Another Markup Language**. It is a data serialization language just like JSON.

## **Why YAML, when we already have JSON or XML?**

- YAML files are easy to read and write for humans, similar to English.

## **Foundation of YAML**



- YAML files should end as `.yaml` or `.yml`
- Begins with `---` and ends with `...`
- `#` defines comment

## **YAML is Case and Indentation Sensitive**

- Members of a **list** should be at the same **indentation level** starting with a **dash(-)** and **space**.
- Each item in the list is a **key: value pair** (**colon must be followed by a space**), called as **dictionary**.
- At each level, exactly **two spaces are used for indentation**. Using tabs is not recommended here.

## Boolean Values

Variables can be defined in YAML files as shown:

```
stream: Java
```

```
allocated: true
```

**Variables can be assigned boolean values in different ways as shown:**

```
allocated: yes
```

```
allocated: no
```

```
allocated: True
```

```
allocated: TRUE
```

```
allocated: false
```

## Data Structures

Complicated data structures are possible in YAML.

You can define **lists having dictionaries**, **dictionaries having lists** or **a mix of both**.

In the following example,

- `name` and `job` are dictionaries. `Skill` is a **list of dictionaries**.
- `David` and `Amy` are **lists having dictionaries**

```
# Employee records
```

```
- David:
```

```
  name: David Moore
```

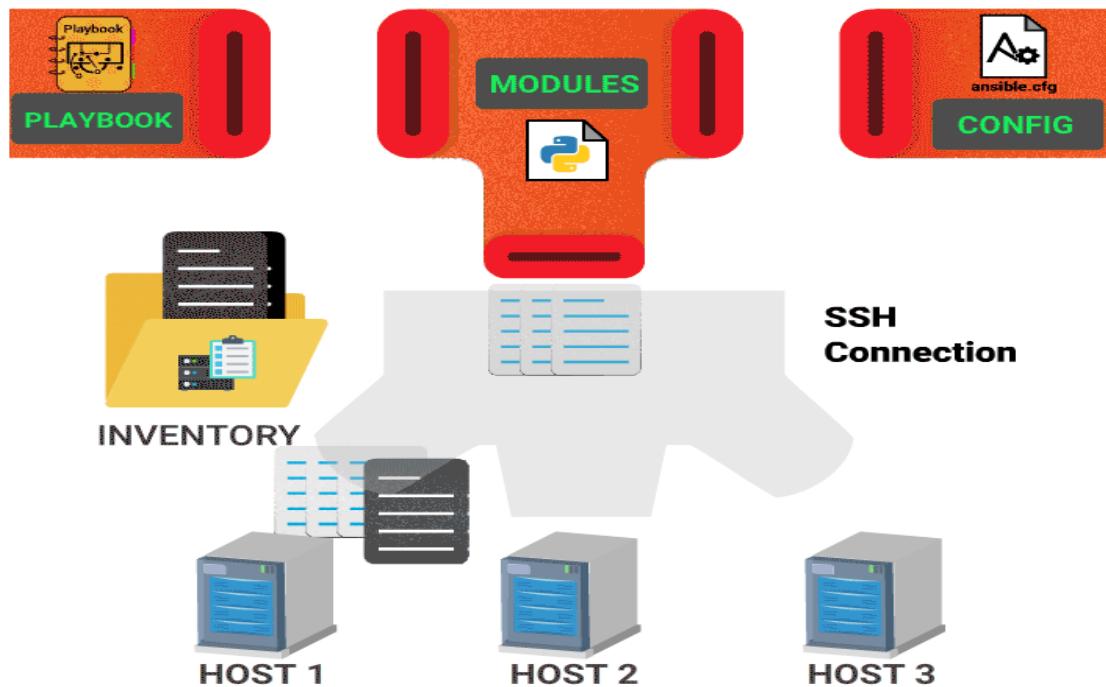
```
  job: Developer
```

```
  skills:
```

- python
  - sql
  - java
- Amy:
- name: Amy Brown
- job: Developer
- skills:
- angular
  - redux
  - react

## Parts of Ansible

### How Ansible Works?



*As already discussed, Ansible is a configuration management tool, based on push-based architecture to automate configuration of your hosts to achieve a **desired state**.*

Following are the components of **Ansible architecture**

- **Inventory** - Defines the list of target hosts
- **Playbook (YAML file)** - Defines list of tasks
- **Module** - A python code invoked from tasks and executed on hosts
- **Control Machine** - Takes playbook and executes each task on particular group of hosts

## Playbook

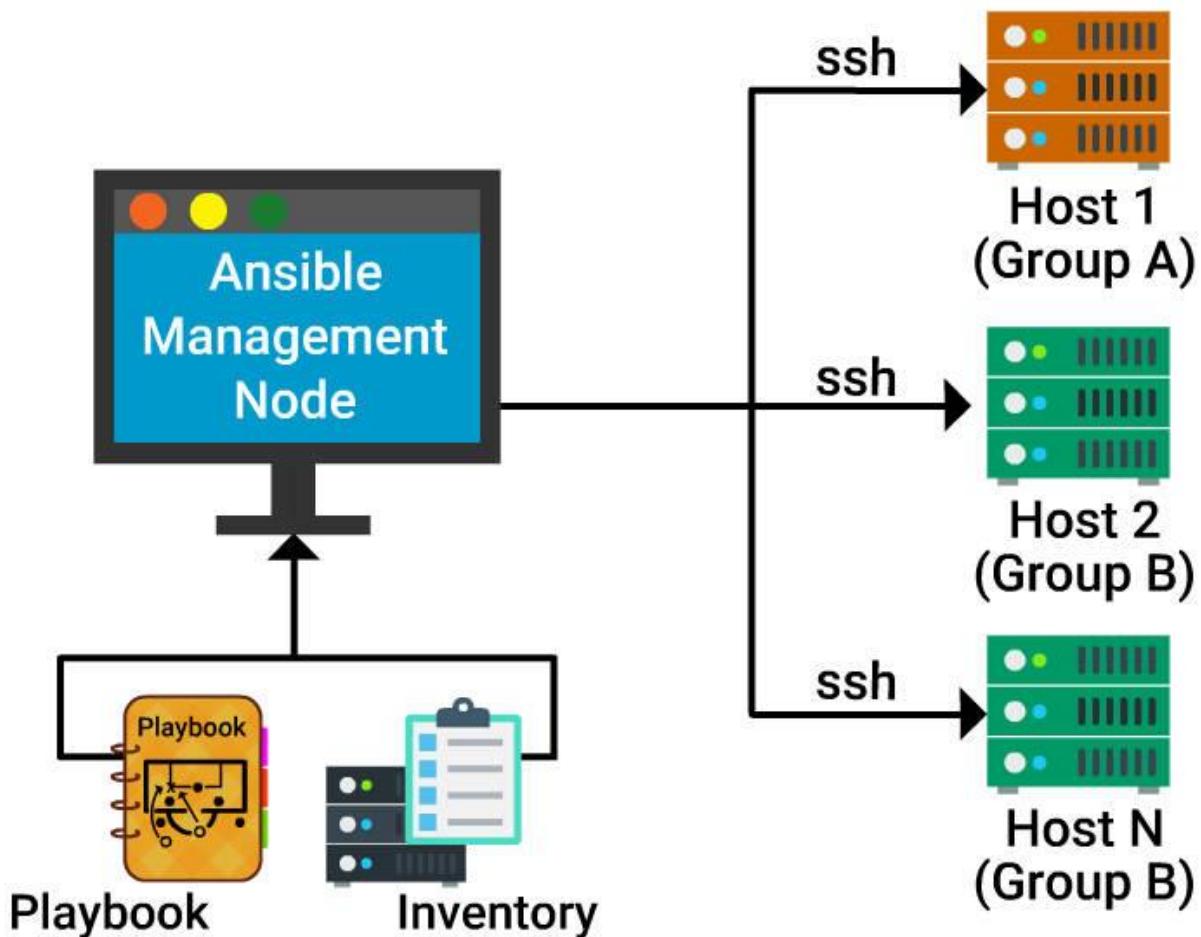


A **Playbook** is a file that defines the desired state of your system.

It contains **plays**, which has a list of **tasks** to run in sequence against a list of **hosts**.

- A **play** is set of **tasks**, grouped together to achieve an objective
- A **task** is an **instruction** you give to Ansible.
- They are written in **YAML** format, a **data serialization language**, that we discussed in previous cards.

## Inventory

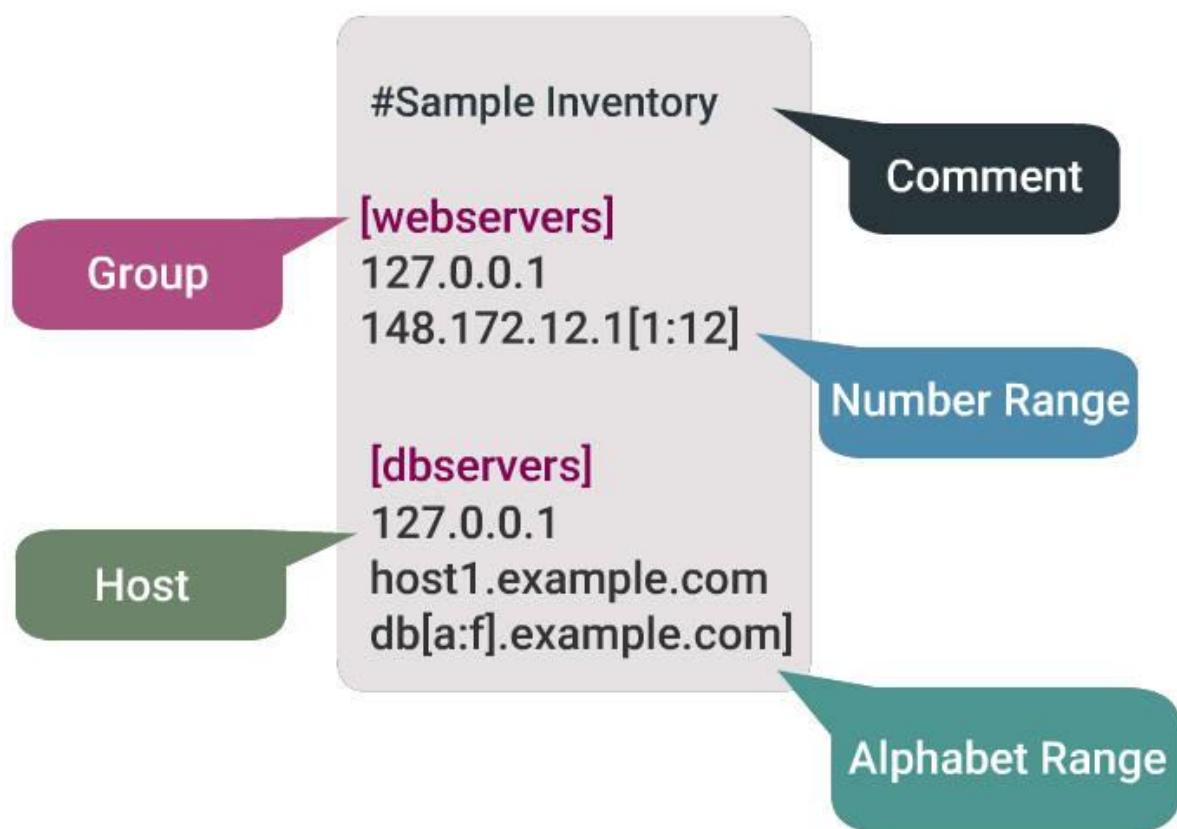


*So Ansible captured the desired state through Playbook, but how would ansible know which machines it should configure through **Inventory**?*

**The Inventory file** in Ansible contains the list of all hosts (target systems/servers) that need to be configured. You can also group hosts under different names as shown:

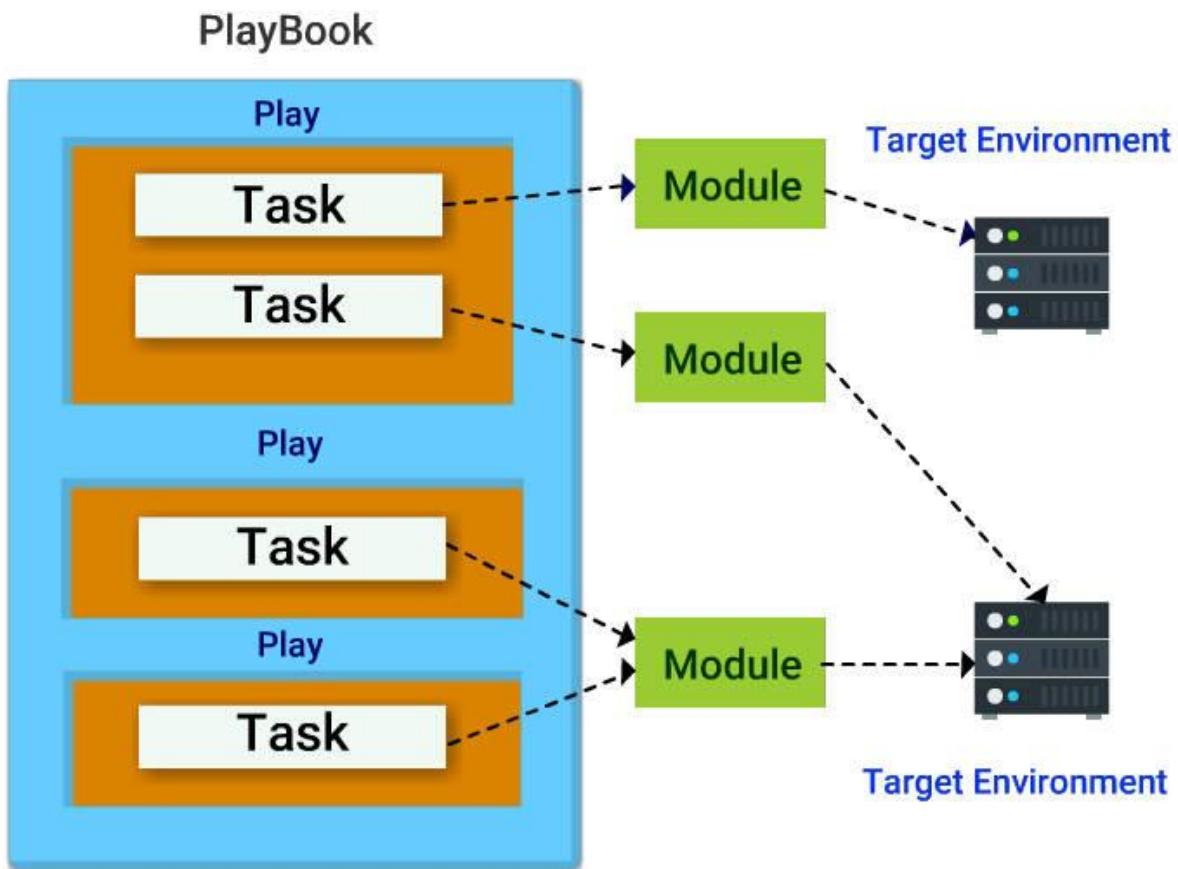
```
[group A]  
Host 1  
[group B]  
Host 2  
Host N
```

## More on Inventory



- Default location of Inventory File: `/etc/ansible/hosts`
- You can define an Inventory file **statically** (`.ini` file) or **dynamically** (`.json` file) to Ansible.

# Modules Perform the Action



Modules are a piece of code that gets executed when you run playbook. You use them to **describe the state you want the host to be in**.

Each **task** in play is made of **module** and **arguments**.

## Ad-Hoc Commands

### Ad-Hoc Keywords

Before hopping into Ad-Hoc commands, let us first learn **Ansible keywords**:

- **ansible**: This is a tool that allows you to **run a single task at a time**.  
\$ ansible <host-pattern> [-m module\_name] [-a args] [options]

- **ansible-playbook**: This is the tool used to run ansible playbook  
`$ ansible-playbook <filename.yml> ... [options]`

## Ad-Hoc Keywords

- **ansible-console**: This is a REPL using which you can run ad-hoc commands on chosen inventories.  
`$ ansible-console <host-pattern> [-m module_name] [-a args] [options]`
- **ansible-pull**: This inverts the default push architecture of Ansible into a pull architecture, which has near-limitless scaling potential.  
`ansible-pull -U URL [options] [ <filename.yml> ]`
- **ansible-doc**: Displays data on modules installed in Ansible libraries.  
`$ ansible-doc [-M module_path] [-l] [-s] [module...]`

## Ad-Hoc Keywords

- **ansible-vault**: Using this you can encrypt any structured data file used by Ansible.  
`$ ansible-vault [create|decrypt|edit|encrypt|rekey] [--help] [options] file_name`
- **ansible-galaxy**: This is a shared repository for Ansible roles. This ansible-galaxy command can be utilized to manage these roles, or to create a skeleton framework for the roles to be uploaded to Galaxy.  
`$ ansible-galaxy [delete|import|info|init|install|list|login|remove|search|setup] [--help] [options]`

## Short Hands in Ansible

- **-a**: This tells the arguments to pass to the module
- **-m**: Execute the module
- **-b**: Use privilege escalation (become)
- **-i**: The path to the inventory, which defaults to /etc/ansible/hosts
- **--version**: Show program version number
- **--help**: Shows help message

# Running Your First Ad-hoc Command

An **ad-hoc command** is a single statement to complete a particular task. For example: consider you want to check if you could connect to your hosts.

Enter the following command:

```
ansible group1 -i myhosts -m ping
```

The above statement is a **single task** to ping target host and return pong if the connection is successful.

- `ansible` is a **keyword** you need to write before running any ad-hoc command
- `group1` is the group name of the **list of hosts**
- `-m` means **module**, this is followed by the **module name ping**, which will be executed to achieve the task

To know more about each module you can try: `ansible-doc ping`.

# Copy a File to the Servers

You can use **copy module** to copy a file from your **control machine to host** as shown:

```
$ touch test.txt
```

- This will create a sample file which could be used to copy

```
$ ansible host01 -i myhosts -m copy -a "src=test.txt dest=/tmp/"
```

- `copy` the file `test.txt` **from your control machine** (where ansible is installed) **to all the hosts** defined in `myhosts` inventory group
- `-a` means **arguments** of that module (*here copy module*)
- `src` is **attribute of copy module** that defines the **source path** of file or directory **on control machine**

Similarly, to fetch a file from Host to your Control Machine, you can use **fetch module**. You may use `ansible-doc fetch` to know about it.

# Encrypting Your File

As you just created a `test.txt` file, let us now encrypt the same using **ansible-vault keyword**.

- `$ ansible-vault encrypt test.txt`: encrypts the file.

This asks for a password to be set. Give a password and confirm it.

- `ansible-vault edit test.txt`: to edit the file and add some content.

This opens `vi editor`. Type some text, then save it(:wq)

- `cat test.txt`: to view the content inside.

*Observe the output carefully*

- `ansible-vault decrypt test.txt`: to decrypt the file, use the password set during encryption
- `cat test.txt`: now observe the output

## Create Directories and Files

You can use `file module` to create files and directories, manage their permissions and ownership as shown:

```
ansible host01 -i myhosts -m file -a "dest=/tmp/test mode=644 state=directory"
```

- This will create **directory** `/tmp/test` on all the `host01` of *myhosts group*
- `mode` defines permission of file/directory
- `state` can take value: file, directory, link, absent, etc

You can set the `state` to **absent** to delete a file or directory to delete it:

```
$ ansible host01 -i myhosts -m file -a "dest=/tmp/test state=absent"
```

## Automating with Ansible

*Till now you were executing each task (using Ad-hoc command) to create a folder, copy a file, encrypt or decrypt a file, etc.*

**What if you need to execute, say, \*\*500 tasks\*\* to configure a server?**

**Do not worry, as Ansible got you.**

You just need to define a `Playbook` that Ansible can play with, and have a popcorn watching Ansible in action.

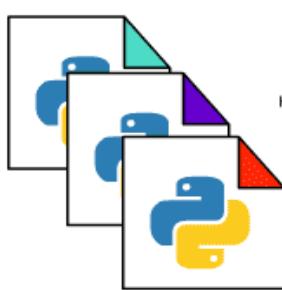
*Read on to find out what Playbook is.*

## Modules - The Engine of Ansible

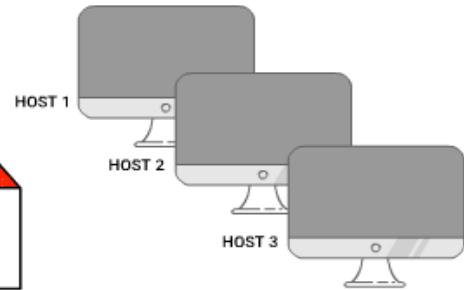
### Modules



**TASKS**



**MODULES**



**HOSTS**

Modules are **blocks of Python code** that gets **executed on remote hosts** when you run each task in Playbook.

Each module is built for a particular task and **you can use arguments to change the behavior of module**.

### Things You Should Know About Modules

- Also referred as **task plugins** or **library plugins**
- **Default location** for Ansible modules is `/usr/share/ansible`
- Take arguments in **key=value** format (`state=stopped`)
- Returns data in **JSON** format

- **Modules should be idempotent**, meaning they should not do any changes if current state of system matches with the desired final state
- To access **list of all installed modules** using command line: `ansible-doc -l`
- To see the **documentation of particular module** using command line: `ansible-doc yum` where yum is the module name
- You can **run modules from the command line or include them in Playbook**.
- Ansible allows you to **write your own module** (this you will learn later in advanced courses of Ansible).

*Let us now go through some standard modules: `apt`, `yum`, `shell`, `command`, and `template`.*

## APT Module

**APT (Advanced Package Tool)** is a command-line tool used to easily manage (install, remove, search, etc.) packages on Ubuntu/Debian based Linux systems.

**Debian Based OS:** Ubuntu, Kali Linux, SteamOS and much more.

```
$ ansible host01 -i myhosts -m apt -a "name=sudo state=latest"
```

```
#This is how you write in Playbook
```

```
- name: Upgrade sudo to the latest version

  apt:
    name: sudo
    state: latest
```

## YUM Module

**YUM (Yellowdog Updater Modified) is a command-line tool used to easily manage (install, remove, search, etc.) packages on RPM (Red Hat Package Manager) based Linux systems.**

**Red Hat Based OS:** Fedora, Qubes OS, CentOS and much more.

```
#This is how you write in Playbook
```

```
- name: upgrade all packages
```

```
yum:
```

```
  name: sudo
```

```
  state: latest
```

## shell Module

In **shell** module, the command name is followed by arguments, which **runs** on remote hosts through a shell(/bin/sh).

You **can use** various operations(|,<,> etc) and environment variable(#HOME).

**ansible host01 -i myhosts -m shell -a "echo \$TERM":** This displays the terminal name of host machine

```
#This is how you write in Playbook
```

```
- name: Execute the command in remote shell
```

```
  shell: echo $TERM
```

## command Module

In **command** module, the command name is followed by **arguments**, which does not run on remote hosts through a shell(/bin/sh).

You **cannot use** various operations(|,<,> etc) and environment variable(#HOME).

- **Make a directory in remote host:** `ansible host01 -i myhosts -m command -a "mkdir folder1"`
- **Check files or folders in remote host:** `ansible host01 -i myhosts -m command -a "ls"`

*Now to check files or folders in your terminal use `ls` and observe the output. As you can see, using `command`, you can execute tasks on remote host.*

## command Everytime

**Most of the Ansible modules are idempotent.**

But `command` module does not exhibit this property as this **runs every time you run playbook**. Hence you will always find the `changed` status on running the same Playbook again and again. Consider you wrote a task to copy a file to remote hosts using `command` module.

**Ansible `command` module will copy the same file the number of times you run the Playbook.**

*Had this been idempotent; Ansible will not copy from the second time, as the file is already present (current state = desired state).*

## command can be Idempotent

To overcome this, you can use `creates` or `remove` parameter, where you define the filename/pattern.

- `creates`: if filename **exists**, task will not run
- `remove`: if filename **does not exist**, task will not run

```
#This is how you write in Playbook
```

```
- name: copy a file, but do not copy if the file already exists
```

```
  command: cp /home/dist/file1.txt /usr/someFolder/ creates=file1.txt
```

# template Module

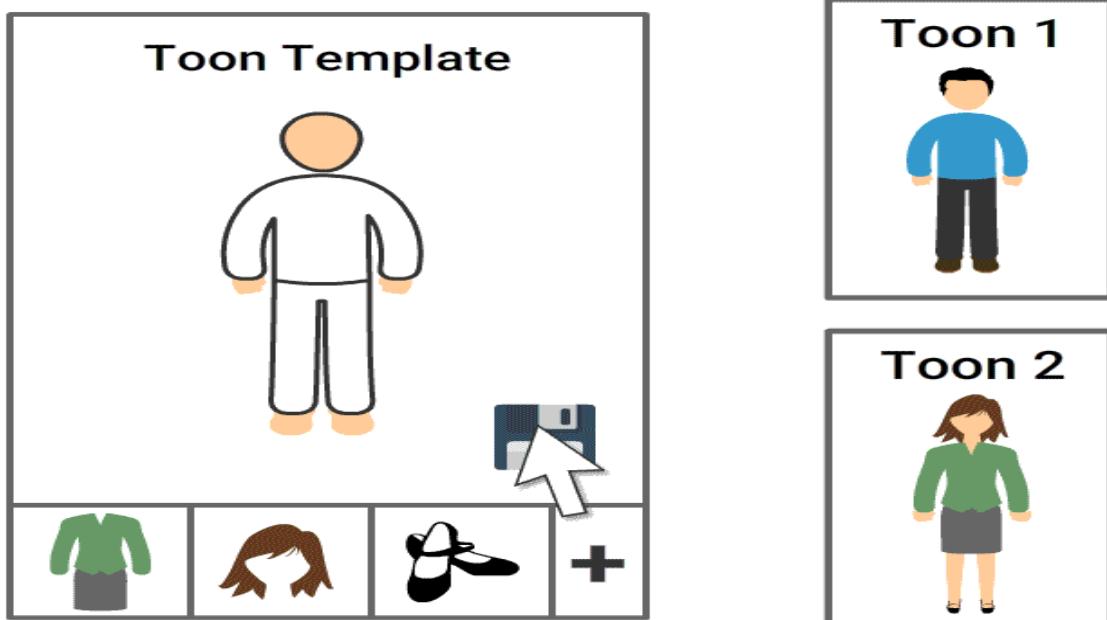
Ansible **template** is a file having a defined structure with some variables and expressions, which can be replaced with the corresponding values to generate new configuration file.

- Template files are **written in Jinja2 language (.j2)** (*Read on to know more about Jinja2*)
- Generally, in Ansible, these files are copied to remote hosts in **JSON or YAML format**

There are basically two files involved to define templates in Ansible:

- **playbook(YAML file)**: here you substitute variables in template file with values
- **template(Jinja2 file)**: here you define the template file in Jinja2 format

*Let us take the example of the human body as a template. Depending on the various elements (footwear, dress, hair) you could name the template as a man or a woman.*



## file: sample-playbook.yml

This is where your template and variables are merged.

Let us consider another example of **Payslip as a template**, where a structure of payslip is pre-defined. Depending on the values being passed new payslip is generated for each **name**.

```
#This is how you write in Playbook
- hosts: all
  vars:
    quarter: false,
    salary : 30000,
    extra : 10000,
    names: ["John","David"]
  tasks:
    - name: Ansible Template
      template:
        src: ../templates/sample-template.j2
        dest: /home/sample-template
```

- **src**: defines path where your template file is kept.
- **dest**: path where you want to copy your file (mostly JSON or YAML formatted file).

## file: sample-template.j2

This is where you define your template.

```
{% for name in names %}
Hi {{name}}!
{% if quarter -%}
  Your pay cheque for this month is {{ salary + extra }}.
{% else -%}
  Your pay cheque for this month is {{ salary }}.
{% - endif %}
{% endfor %}
```

- variable names are given within double curly braces **{}{ }{ }**

# Hands-on Template

You can test your Jinja2 templates [online](#) with the steps as explained.

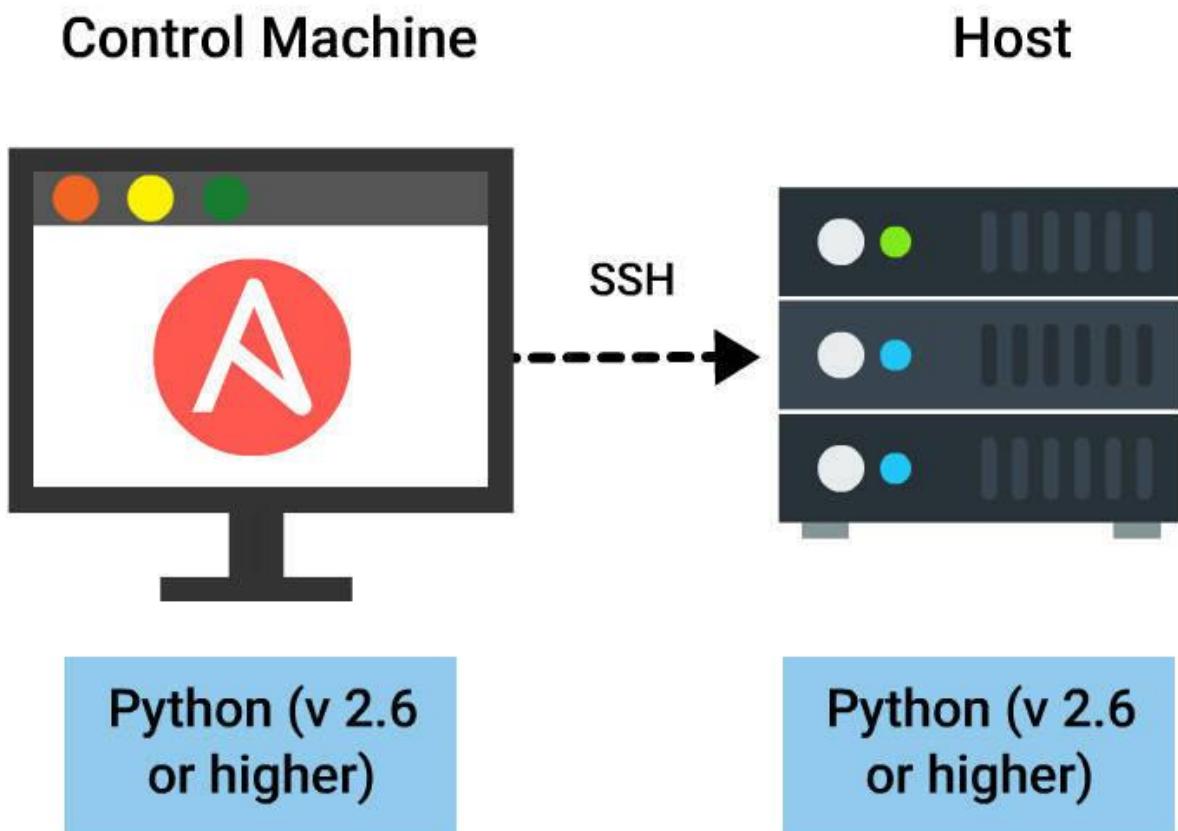
- Copy `sample-template.j2` file and paste in **Template** box
- Copy **variables** from `sample-playbook.yml` and paste in **Values** box
- Click on **Convert**

## Try It Out - Template

Try to make a template to display **Students mark sheet** that displays `name of student`, marks in `Physics`, `Chemistry` and `Maths`. Provide relevant values via variables.

## Set-Up Environment for Ansible

### Environment at a Glance



## Control Machine Requirements

- Linux/Mac OS (Windows not supported)
- Python (2.6 or later version) should be installed.
- Install Ansible.

## Host Requirements

- Only Python needs to be installed.
- Uses SSH to connect to control machine.
- SFTP is used else scp can be configured.

## Step By Step

*You need to follow these steps to setup environment on your system:*

1. Install [\*\*VirtualBox\*\*](#).
2. Install [\*\*Vagrant\*\*](#). [\*\*Vagrant\*\*](#) enables virtual software development environment.
3. Install [\*\*Ansible\*\*](#).

## Install Vagrant

### Download and install Vagrant

- **Identify the OS** you want to install in your virtual system. Accordingly, you need to find the [\*\*Vagrant Box\*\*](#) (*let us consider Ubuntu on both our control and host machines*).
- **Create a folder on your local machine**, to keep your Vagrant Boxes.

Now, **open your terminal** and move to the path where you created your folder for Vagrant boxes. Then run the following commands:

- `$ vagrant init ubuntu/trusty64`: Initialises vagrant file inside the folder
- `$ vagrant up --provider VirtualBox`: will download necessary files of your virtual machine
- `$ vagrant ssh`: will make secure connection with virtual machine

## Install Ansible

*Once you are done with Virtual Box and Vagrant setup, you only need to run one last command in your terminal to install Ansible on the control machine.*

On Ubuntu/Debian/Linux Mint

```
$ sudo apt-get install ansible
```

On RHEL/CentOS/Fedora

```
$ sudo yum install epel-release  
$ sudo yum install ansible
```

To verify if Ansible is successfully installed or not: `ansible --version`

You can follow [Ansible Official Docs](#) for detailed instructions.

## Playbook Explained

## Writing Playbook

Let us now write a Playbook to print **Hello World**. You can follow the steps parallelly.

## Step By Step

- **Open** any environment
- Enter **commands given in Step 2 of 6**. This makes an SSH connection to your host.
- Open Playbook using the **vi editor** (eg: vi demo.yml).
- **Write** your playbook (given in next card).
- **Press Escape** key on your keyboard.
- **Save and close editor** using :wq(w: write, q: quit).
- **Run your playbook:** `$ ansible-playbook -i myhosts demo.yml`.

## Sample Playbook 1 - "demo.yml"

```
---  
- name: this play displays "hello world"  
  hosts: all  
  tasks:  
    - name: displaying "hello world"  
      shell: echo "hello world"  
    - name: displaying wishes for the day  
      shell: echo "have a good day"
```

## Playbook Step 1 - Name and Hosts

---

- name: this play displays "hello world"

hosts: all

- A Playbook always starts with three dashes ---.
- **name** tells the name of the play.
- **hosts** tell the list of hosts on which this play will be played.

## Playbook Step 2 - Tasks

tasks:

- name: displaying "hello world"

  shell: echo "hello world"

- name: displaying wishes for the day

  shell: echo "have a good day"

- **name** is optional but is always recommended as it improves readability
- **shell: echo "hello world"** is a single task. This executes **shell module** and calls its **echo argument** to display the message written

## Playbook Step 3 - Run Your Playbook

Type **ansible-playbook -i myhosts demo.yml** from the terminal to run your Playbook.

- **ansible-playbook** is the command to execute your Playbook
- **-i myhosts** tell the inventory name is *myhosts*
- **demo.yml** is the playbook that needs to be executed

## Output Of "demo.yml"

This is how your Playbook would run

```

PLAY [this play displays "hello world"] ****
*****
GATHERING FACTS ****
*****
ok: [host01]
TASK: [displaying "hello world"] ****
*****
changed: [host01]
TASK: [displaying wishes for the day] ****
*****
changed: [host01]
PLAY RECAP ****
*****
host01 : ok=3    changed=2   unreachable=0   failed=0

```

## Sample Playbook 2 - Install Apache

**Let us now take another example to [install apache in your hosts](#).**

This is how your Playbook would look:

```

---
- name: install apache
  hosts: all
  sudo: yes
  tasks:
    - name: install apache2
      apt: name=apache2 update_cache=yes state=latest

```

*Save and run this playbook as [apache.yml](#) in Katacoda and check the output.*

**Please note that for installing in RHEL/CentOS/Fedora,[yum module](#) is used instead of [apt](#).**

## Restructuring Playbook

You can define the same playbook as:

```

---
- name: install apache
  hosts: all
  sudo: yes
  tasks:

```

```
- name: install apache2
  apt:
    name: apache2
    update_cache: yes
    state: latest
```

- observe colon `:` is used while structuring arguments vertically, whereas equal to sign `=` is used while structuring arguments horizontally

*Both ways of structuring your playbook is fine.*

- This vertical structuring of arguments is not a `list`, as list starts with dash sign `-`.

Here `name`, `update_cache` and `state` are arguments of module `apt`, hence they do not start with `-`.

## Handlers

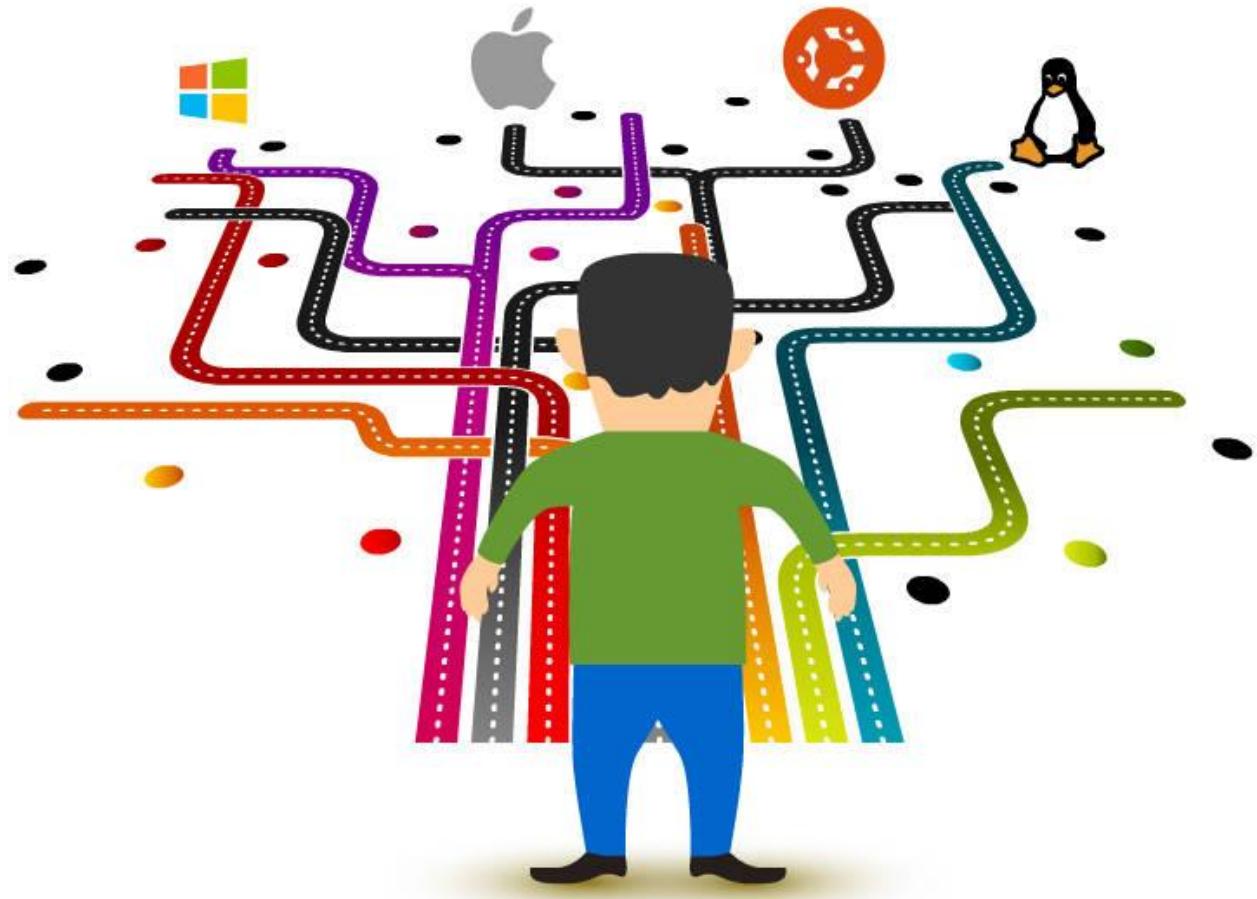
Handlers are special tasks that run only on certain triggers like `notify` keyword.

- Handlers run mostly at the `end of the play`.
- Handlers `run only once` even if you run the playbook multiple times.

This is how a handler would look:

```
---
- name: install apache
.....
tasks:
  - name: install apache2
.....
notify:
  - start Apache
.....
handlers:
  - name: start Apache
.....
```

## The Decision Conditionals In Ansible



*Recollect while reading the section on setting up your environment or while installing an application , you need different modules (`apt` and `yum`) to execute task.*

*What will you do, if you have to configure 50 such servers with different OS and requirements?*

This is where **conditionals can be used to decide and control the execution flow** in Ansible.

## Conditionals - When Clause

**When** clause in Ansible is a raw **jinja2 expression** that defines the condition which will be evaluated for TRUE or FALSE.

tasks:

```
- name: "shutdown Debian flavored systems"
  command: /sbin/shutdown -t now
  when: ansible_os_family == "Debian"
```

## Conditionals - When Clause

- You can **group conditions using parenthesis** 0

tasks:

```
- name: "shutdown CentOS 6 and Debian 7 systems"
  command: /sbin/shutdown -t now
  when: (ansible_distribution == "CentOS" and ansible_distribution_major_version == "6") or
        (ansible_distribution == "Debian" and ansible_distribution_major_version == "7")+
```

- You can **define multiple conditions**, where all of them should be true to execute the tasks

tasks:

```
- name: "shut down CentOS 6 systems"
  command: /sbin/shutdown -t now
  when:
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "6"
```

# Looping in Ansible

## LOOPS



If you need to repeat the same task with different items (loop through), you can use `with_items` clause

For example, consider you want to add several users

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "developer"
  with_items:
    - raj
    - david
    - john
    - lauren
```

# Looping Through The Inventory

You can loop through the Inventory file to *list all hosts or hosts from a play*, using `with_items` with the `play_hosts` or `groups` variables,

```
# show all the hosts in the inventory
- debug:
  msg: "{{ item }}"
  with_items:
    - "{{ groups['all'] }}"

# show all the hosts in the current play
- debug:
  msg: "{{ item }}"
  with_items:
    - "{{ play_hosts }}"
```

## Sample Playbook 3 - Conditions And Loop

This Playbook will add Java Packages to different systems (handling Ubuntu/Debian OS)

```
- name: debian | ubuntu | add java ppa repo
  apt_repository:
    repo=ppa:webupd8team/java
    state=present
  become: yes
  when: ansible_distribution == 'Ubuntu'
- name: debian | ensure the webupd8 launchpad apt repository is present
  apt_repository:
    repo="{{ item }} http://ppa.launchpad.net/webupd8team/java/ubuntu trusty
main"
    update_cache=yes
    state=present
  with_items:
    - deb
    - deb-src
  become: yes
  when: ansible_distribution == 'Debian'
```

## Configuring Your Master

### ansible.cfg

You told how your hosts should behave (via Playbook). But how do you tell Ansible (your control machine) should behave? through ansible.cfg

ansible.cfg is a configuration file that defines how Ansible should behave.

It tells

- how to establish an ssh connection
- duration of ssh connection with the host
- how to run the playbook
- where to log errors that might occur while playing playbook on hosts etc.

## Ansible Configuration Settings

Your ansible.cfg file will have the following settings:

- [defaults]
- [privilege\_escalation]
- [paramiko\_connection]
- [ssh\_connection]
- [accelerate]

You will now learn the frequently used settings in upcoming cards. For detailed understanding, you may refer to the [Ansible Official Docs](#).

### [defaults]

poll\_interval

For asynchronous tasks in Ansible, this tells the frequency of checking the status of task completion. The default value is 15 seconds; which implies, for every 15 sec, it will check if the task is completed.

poll\_interval = 15

## `sudo_user`

This is the default user to sudo. If `--sudo-user` is not specified in an Ansible playbook, the default is the most logical: ‘root’:

`sudo_user = root`

## [defaults]

### `ask_pass`

This tells **if Ansible Playbook should ask for a password** by default. If SSH key is used for authentication, the **default behavior is no**.

`ask_pass = True`

### `ask_sudo_pass`

Just like `ask_pass`, this asks for **sudo password** by default while sudoing.

`ask_sudo_pass = True`

### `remote_port`

If your systems did not define an alternate value in Inventory for SSH port, this sets the default value to 22.

`remote_port = 22`

## [privilege\_escalation]

\*For some tasks to execute in Ansible, you require administrative access to the system.

The settings under [privilege\_escalation] will escalate your privileges.\*  
`become`

If set to true or yes, you **activate privilege escalation**.

*Default behavior is no.*

`become=True`

### `become_method`

You can define the **method for privilege escalation**.

*Default is sudo, other options are su, pbrun, pfexec, doas, ksu.*

`become_method=sudo`

## [privilege\_escalation]

### `become_user`

This allows you to **become the user over privilege escalation**.

*Default is root.*

`become_user=root`

`become_ask_pass`

Asks the **password for privilege escalation**, the *default is False*.

`become_ask_pass=False`

## Other Settings

`accelerate_timeout`

This setting will **close the socket connection** if no data is received from the client for the defined time duration.

*default: accelerate\_timeout = 30*

`record_host_keys`

If the setting is `True`(default value), this will **record new hosts on the user's host file**, provided **host key checking is enabled**. Setting it to `False`, the performance will improve and which is recommended when **host key checking is disabled**.

`record_host_keys=False`

`pipelining`

If enabled, the **number of SSH connections** required for execution of a module on the remote server is **reduced**, improving the performance significantly.

*By default this is disabled: pipelining = False.*

## Write a Play Book

### Expected Output - Try It Out

On executing `ansible-playbook -i myhosts demo.yml`, you can expect the output as shown:

```
PLAY [play1 install and start nginx] ****
*****
GATHERING FACTS ****
*****
ok: [host01]
TASK: [Installs nginx web server] ****
*****
changed: [host01]
TASK: [start nginx] ****
*****
ok: [host01]
```

```

PLAY [play2 install PostgreSQL] ****
*****
GATHERING FACTS ****
*****
ok: [host01]
TASK: [Installing PostgreSQL] ****
*****
changed: [host01]
TASK: [ensure PostgreSQL is running] ****
*****
ok: [host01]
PLAY RECAP ****
*****
host01 : ok=6    changed=2   unreachable=0   failed=0

```

## Ansible Is Idempotent

**Try executing the same playbook for the second time and observe the output. Thus, Ansible is **Idempotent**.**

*An **Idempotent** operation results to the same output, if executed multiple times (no changes).*

## Summary

Hope you enjoyed learning Ansible.

**Let us list down the takeaways from this course:**

- **Configuration Management** and different tools
- **YAML** language
- **Ansible**: what it is and how it works
- **Key parts** of Ansible
- **ansible.cfg**: To configure your control machine
- **Ad-Hoc commands**
- **Sample Playbooks** to display "hello world" and install "Apache"
- **Local Environment** for Ansible (*using Virtual Box and Vagrant*)
- **Online Environment** for Ansible (*Katacoda*)

# SonarQube

## Introducing SonarQube

### Prologue

*Welcome to the course on SonarQube (formerly known as Sonar).*

You are probably wondering what SonarQube is all about? So read on to find it out.

#### **Where can you use it?**

- SonarQube can be used in any project to inspect the code. It supports more than 25+ languages.

#### **How is it different from other analyzing tools?**

- Only tool in the market that supports a leak approach as a practice to code quality.

### Preface to Sonar

#### **Why to use Sonar?**

- With SonarQube, a developer has the proprietorship of his/her code.
- It ensures fast, efficient report generation and processing across multiple fields of analysis.
- It detects tricky issues, supports multiple languages, DevOps integrations, and centralize quality.

#### **Is it difficult to operate?**

- No, it's very easy to handle, and it gives you the result without keeping in concern how many projects you have or how complex your reporting requirements are!
- The results are shown in a very fancy UI having charts, graphs, and red-blue light.

# Topics to Expect

*In this course, you will come across these topics.*

- What is Sonar?
- Installation of SonarQube
- Features of SonarQube
- Types of Issues
- What are the rules in SonarQube?
- Importance of Code Viewer
- Administration
- How to scan the code?

*So without much delay, let's start.*

## What is Sonar?

**SonarQube** (formerly known as **Sonar**) is an open source platform that allows you to track and improve the quality of source code.

- It's a code analysis tool for **continuous inspection** of the code quality.
- One of the most important aspects in analyzing code is how much of your source code is being tested? (code coverage)
- For this, Sonar unites with the most popular open source code coverage tool (**JaCoCo, Cobertura, Emma**)
- By default, it uses the JaCoCo engine.

## JaCoCo

**Now a question arises, why JaCoCo is selected among the three?**

- **JaCoCo** is continuously developing and improving.
- It's fast.
- It's the only tool that analyses bytecode in a flash.
- It has its own Java agent for code analysis.

## Installation of SonarQube

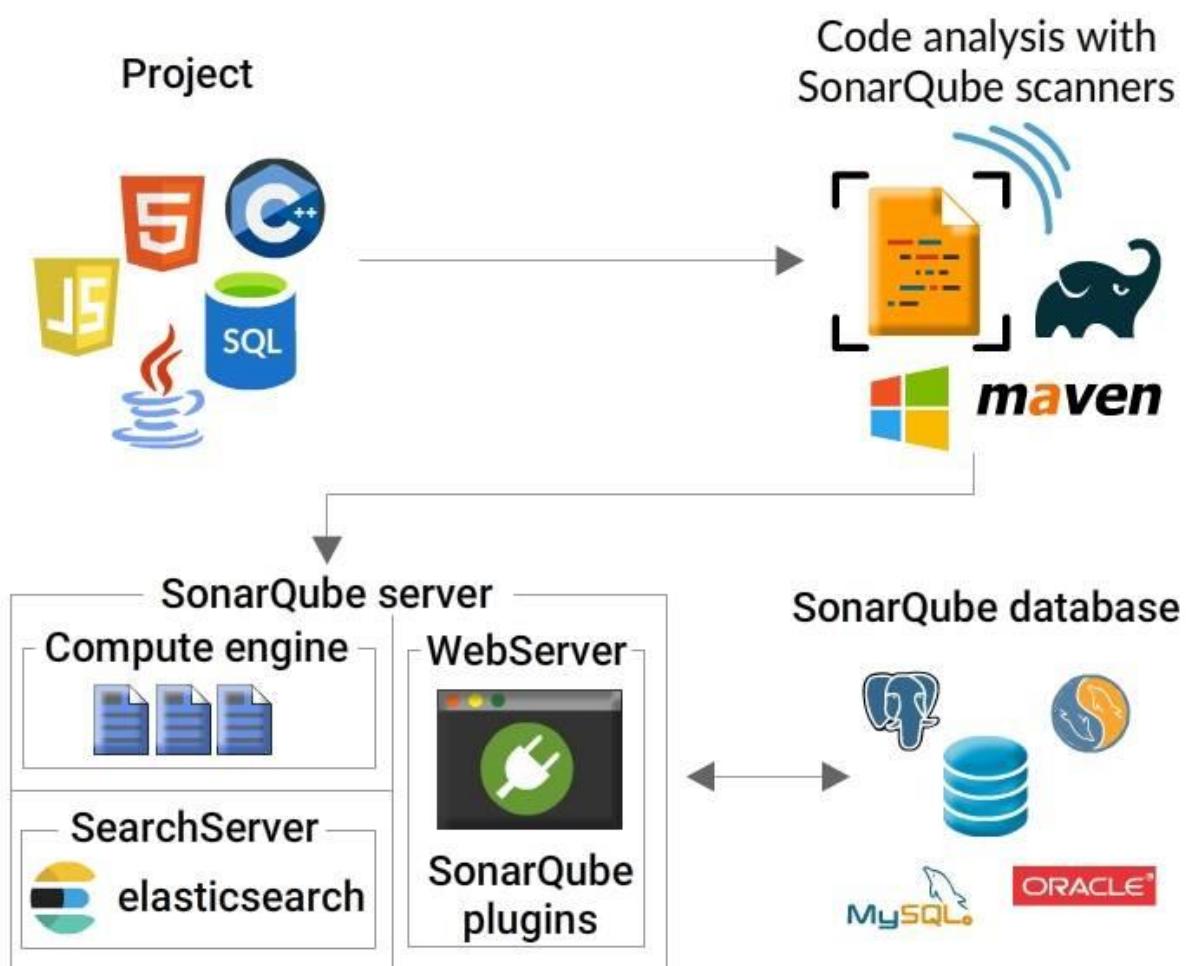
**SonarQube** incorporates with **Eclipse**, **Visual Studio**, and **IntelliJ ID** development environment through SonarLint plugins or else you can install in your local machine.

## Steps:

1. Install SonarQube.
2. Unzip the file.
3. Move it to the desired directory (e.g., opt).
4. Go to the directory, open the terminal, then run

```
/opt/sonarqube/bin/[OS]/sonar.sh console
```

## Architecture



The SonarQube platform constitutes four processes:

### *1. SonarQube Server*

It is comprised of 3 main processes viz.,

1. **Web Server** - to look for quality snapshots and configures the SonarQube instances.
2. **Search Server** - to get results from UI.
3. **Compute Engine Server** - to save the analysis report in the SonarQube Database.

## Architecture

### *2. SonarQube DataBase*

1. The configuration of the SonarQube instance (security, plugins settings, etc.)
2. The quality snapshots of projects, views, etc.

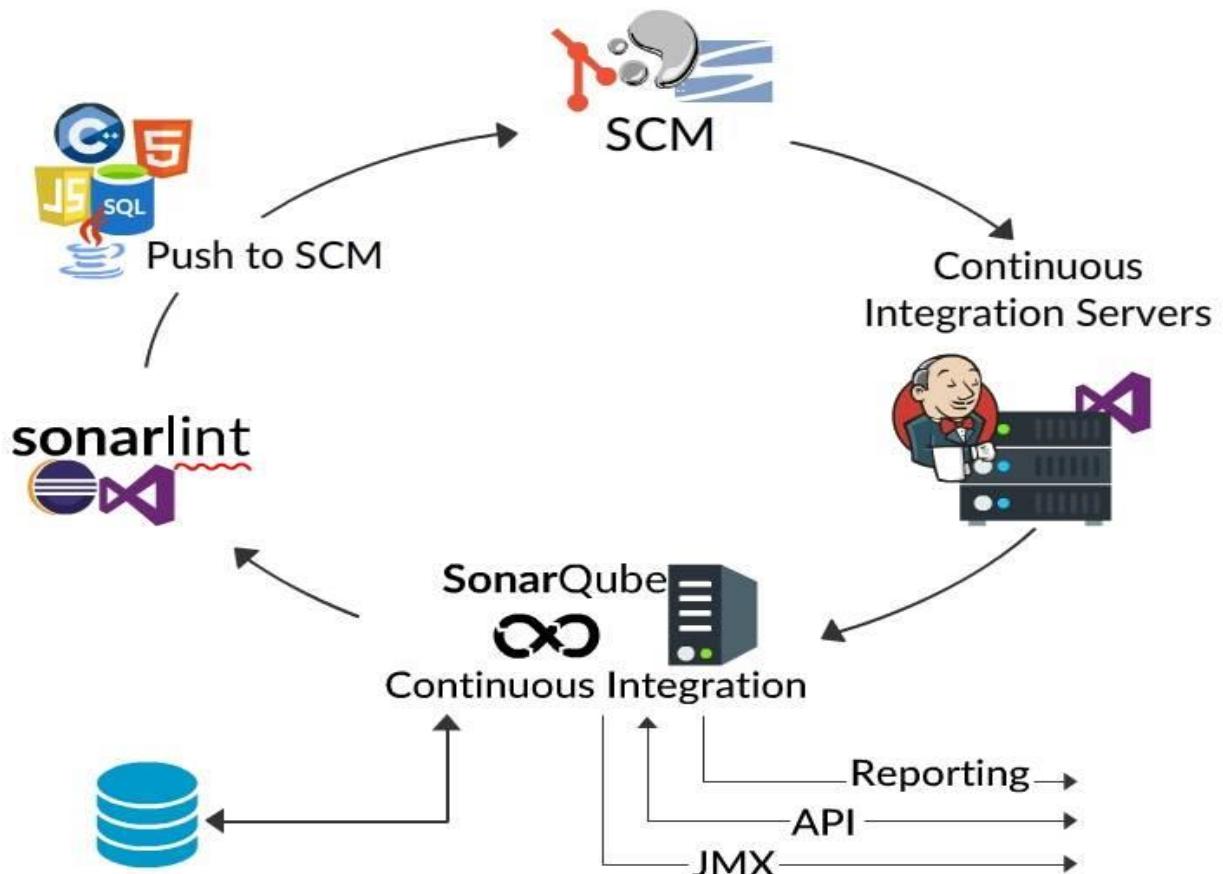
### *3. SonarQube Plugins*

Installed on the server.

### *4. SonarQube Scanners*

Running on your Build / Continuous Integration Servers to analyze projects.

## Integration



- SonarQube integrates with other Application Lifecycle Management (ALM) tools.
- Developers use SonarLint plugin in their IDEs and to do the local analysis.
- Developers push their code into their favorite SCM: git, etc.

## Integration

- The Continuous Integration Server triggers an automatic build and the execution of the SonarQube Scanner required to run the SonarQube analysis.
- The analysis report is sent to the SonarQube Server for processing.
- The results are stored in the SonarQube Database.
- Developers review, comment, and challenge their issues to manage and reduce their Technical Debt through the SonarQube UI.
- Managers receive Reports from the analysis.
- Ops use APIs to automate configuration and extract data from SonarQube.
- JMX is being used by Ops to monitor SonarQube Server.

## Requirements

### Prerequisite

The essential requirement to run SonarQube is Java, it should be installed in your system.

Hardware Requirements:

- The SonarQube server needs 2GB of RAM to run efficiently.
- Hard drives that have excellent read and write performance as for searching the server should be up and run to a large amount of I/O will be done.
- SonarQube JAVA analyzer is compatible with any kind of Java source files despite the version. But SonarQube analysis and the SonarQube Server require the specific version of JVM.
- It is recommended to use Oracle JRE 8 and OpenJDK 8.
- And in databases, you can use PostgreSQL 8.x/9.x 1, Microsoft SQL Server-2014/2016, and MySQL 5.6/5.7.

# Getting Started

Steps to start with SonarQube:

1. Download the [latest](#) version of Sonar.
2. Unzip the file: `unzip /home/<yourusername>/Downloads/sonarqube-6.7.4.zip.`
3. Go to the file: `/home/<yourusername>/Downloads/sonarqube-6.7.4/conf/sonar.properties`  
(To make changes in this file, refer to the next card).
4. Now move the `sonar.properties` file to `"/etc/sonarqube"` folder.
5. Start the SonarQube server:

```
/etc/sonarqube/bin/[OS]/sonar.sh console
```

6. Then log into <http://localhost:9000> with System Administrator credentials.

For more information, you can check [here](#).

## sonar.properties File

You need to make some changes and configure Sonar properties in [sonar.properties File](#).

```
sonar.jdbc.url=jdbc:sqlserver://localhost;databaseName=sonar  
sonar.jdbc.username=sonarqube  
sonar.jdbc.password=mypassword
```

To set up [server on port 9000](#), add the below configurations:

```
sonar.web.host=127.0.0.1  
sonar.web.context=/sonar  
sonar.web.port=9000
```

Now you can start the service,

```
sudo /opt/sonar/bin/linux-x86-64/sonar.sh start
```

Replace the start keyword with [stop](#) to [shutdown](#) the server.

For more information, refer [configuration of database](#).

## Installing Database

SonarQube has an inbuilt database available along with installation that supports small projects. This database can be made use of by executing the following commands.

```
wget https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-7.1.zip
```

```
unzip sonarqube-7.1.zip  
mv sonarqube-7.1 /opt/sonar
```

Alternately, you could also install the database of your choice depending on the project needs. Following are the set of commands given as an example to install MySQL and make it as a SonarQube DB.

After installing MySQL, run `mysql -u root -p`. Username and password would be set at the time of installation.

Let's create a sonar database by executing these commands.

```
CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8_general_ci;  
CREATE USER 'sonar' IDENTIFIED BY 'sonar';  
GRANT ALL ON sonar.* TO 'sonar'@'%' IDENTIFIED BY 'sonar';  
GRANT ALL ON sonar.* TO 'sonar'@'localhost' IDENTIFIED BY 'sonar';  
FLUSH PRIVILEGES;
```

## Analysis of Project

### Overview

After the installation of SonarQube, you can install an analyzer to analyze a project. So install the Sonar Scanner (analyzer of SonarQube) and configure according to your needs.

- For Gradle Project-Scanner for Gradle
- For MSBuild- Scanner for MSBuild
- For Jenkin Project-Scanner for Jenkins
- For Maven Project-Scanner for Maven
- For Ant-Scanner for Ant
- For Azure Devops-Scanner for Azure

### What Analysis Produces?

**SonarQube** performs analysis on more than 20 languages. The result of the analysis depends on issues and measures.

- The analysis will vary from language to language.

- A static analysis of source code is performed on all languages (.class file in java, .dll files in C#).
- A dynamic analysis is done on certain languages.

## Files SonarQube will Analyze

*The files that are being recognized by the analyzer will be loaded by default.*

For example, if we have plugins like SONARJAVA, SONARJS, then only .js and .java files will be taken into consideration.

## Behind the Scene During Analysis

Sonar Scanner is taken as a **default** scanner to analyze code with SonarQube.

- During analysis, the server requests data.
- Then the files are being analyzed, and then the analyzed data is sent to the server.
- The analysis of the code is done sequentially and are being queued.
- The updates sometimes take a long time, and you can see the in-progress icon in your SonarQube project.

There are some [background tasks](#) after the analysis of reports.

## Configuration of Parameters

There are three types of analysis parameters.

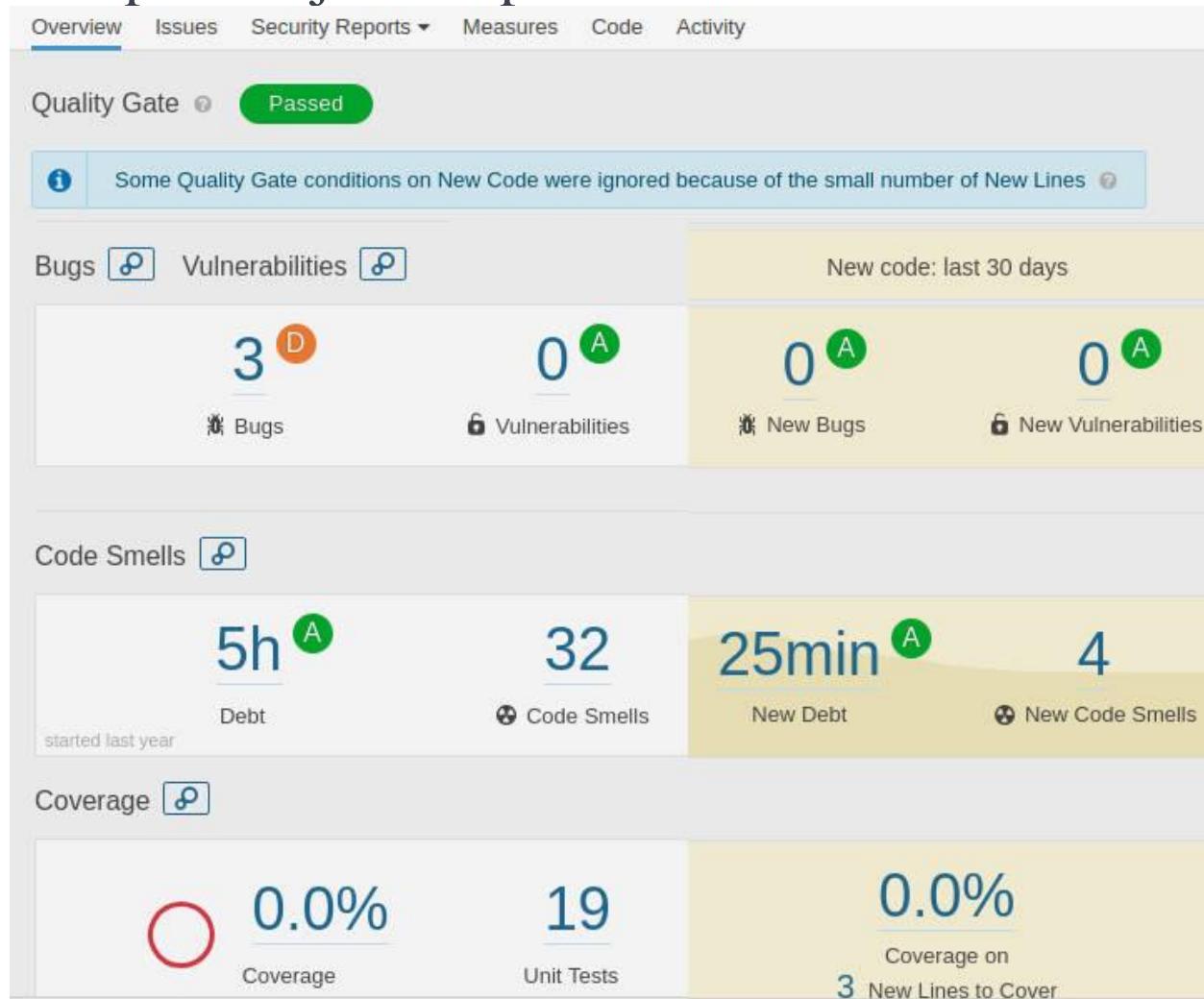
1. **Global analysis parameters**
2. **Project analysis parameters**
3. **CommandLine Parameters**

Global analysis parameters are set in the UI (Administration > Configuration > General Settings).

Project analysis parameters are determined in the project analysis configuration file and override global analysis parameters.

Command line parameters are set while launching an analysis.

# Sample Project Report



## Sonar Scanner

### Why Sonar Scanner?

Sonar is the code quality platform.

- It is a **server**.
- **SonarRunner** is the old name of **Sonar Scanner**. It analyses the **code**.
- The result of code analysis is issues and measures.
- For all languages, static code analysis of source code is done.
- For only a few languages, static analysis of compiled code is done.
- Dynamic analysis is done in certain languages.

**Note:** For a maven-project, you do not need Sonar Scanner.

For more details on Sonar Scanner installation, you can go through this [link](#).

# Creating a Database

- Following is the SQLite3 command that is used to create a new database.

```
sqlite3 DatabaseName.db
```

**Note:** Database name should be unique.

- You can view the databases that you have created by running **.databases** command in terminal.
- You can get out from the sqlite3 through **.quit** command.
- You can also export the whole database into a file using **.dump** command.
- Following is the command that you should use to export a db to a file.

```
sqlite3 DatabaseName.dump > FileName
```

# Installation of Sonar-Scanner

- Download [sonar-scanner](#).
- Extract the zip file and move it to opt directory.

```
cd opt/sonarscanner/conf/sonar-scanner.properties
```

Uncomment the following line:

```
#----- Default SonarQube server  
#sonar.host.url=http://localhost:9000
```

Then add the directory to your path:

```
vi ~/.bashrc  
export PATH=$PATH:/opt/sonarscanner/bin
```

Go to the project directory and run the following in command line:

```
source ~/.bashrc  
sonar-scanner
```

# Use of Sonar Scanner

You need to create a configuration file in the root directory of the project namely,

->**sonar-project.properties**.

```
sonar.projectKey=project_name  
sonar.projectName=project_name  
sonar.projectVersion=1.0  
sonar.sources=.
```

Then open the command prompt in your project base directory and run sonar scanner.

```
sonar-scanner
```

NOTE: Project Key should be unique.

## Analysis of Java Project

- Installation of Sonar Server.
- Install SonarJava (other plugins if you want). By default, SonarJava is given.

Analysis: **For Maven Project:** (use SonarQube Scanner).

Execute the command from the root directory of the project.

```
mvn sonar:sonar
```

**For Gradle projects:** declare the org.sonarqube plugin in build.gradle file

Execute the command from the root directory of the project.

```
./gradlew sonarqube -Dsonar.host.url=[SonarQube URL]
```

# Plugins

## Plugins

Sonar is not just restricted to Java-based projects. It also supports a wide variety of projects and tools using a list of plugins supported to extend the code analysis, defects hunting, and reporting functionalities.

SonarQube installs available [plugins](#) from Marketplace automatically if the SonarQube is on the open internet. However, in case of lack of internet, needed plugins can also be installed manually.

As an example, the best plugin for java code analysis is **SONARJAVA**. To test Java code, you need certain [test libraries](#) properties. You can use additional java-specific plugins such as [Cobertura](#) or Android Lint.

## Hands-on scenario

You are tasked to work on SonarQube where you need to scan a sample application using sonar scanner. Follow the below steps to complete your task.

- You are provided with the Readme file on desktop and refer that to access sonarqube.
- Learn to install sonar scanner in the environment provided to you.
- Download a sample project and scan it using sonar scanner to generate coverage report and analyse the results.

## Features of Sonar

### Features of Sonar

To work smoothly with this tool, you should know some important features of Sonar.

- Leak
- Quality Gate
- Quality Profile
- Issues

You will get to understand more about these features in the forthcoming cards.

### Leak

In a project, the code review comes much later in the development, and by then the collaborator (developer) would be moving on to develop newer versions of their project(s).

The best and convenient way to address code quality issues is to fix the **leak** ie., focussing on the parts of the code that is developed after the earlier stable release.

## Quality Gate

In Sonar, you can define various rules and checks that determine the code quality. In case of any violation against these rules, it is necessary to be alerted immediately. For instance, we want coverage for new code to be more than 80%, or else we will get warnings. In this, we have many metrics like duplicate lines, maintainability rating, reliability rating, etc.

**Quality Gate** is the best way to enforce quality policies. You can define as many Quality Gates as many applications since quality requirement will vary for every application.

The quality gate "**Sonar way**" is provided by SonarSource is built-in, read-only and activated by default.

Three information measures allow you to enforce a given **Rating of Reliability, Security, and Maintainability**. All of these three are recommended and comes as a part of default Quality Gate.

## More on Quality Gate

The Quality Gate status can be seen at the top of [Project Page](#).

- You can also subscribe to quality gate status to get the notification.
- Quality Gates can be accessed by any user and can view every aspect of a quality gate.
- To make any changes users must be granted the Administrator Quality Profiles and Gates permission.
- A project administrator can choose which quality gates his/her project is associated with. See [Project Settings](#) for more.

## More on Quality Gate

- To manage quality gates, go to Quality Gates (option in the menu bar). Each Quality Gate condition is a combination of :

1. measure

2. period: Value (to date) or Leak (differential value over the Leak period)
3. comparison operator
4. warning value (optional)
5. error value (optional)

## Quality Profile

- SonarQube works on profiles.
- The set of coding rules according to different criteria is defined through the Quality Profile related to the project.
- The **Quality Profile service** is main to SonarQube, as it's where you define requirements by defining **rules**.
- To manage Quality Profiles, go to Quality Profiles (top bar), where you'll find profiles grouped by language. To know more about Quality Profile, you can go through this [link](#).

## Handling of Issues

When SonarQube needs to handle issues?

- Whenever the code breaks the rule
- Whenever the piece of code disobeys the rules you have set for the project in Quality Profiles. SonarQube will raise an issue.
- Now when the issue is raised, the developer requires to correct the code. In this video, you will see how SonarQube helps in handling the issues

## Rules

### Introduction to Rules

There are three basic types of rules:-

1. **Reliability** rules-related to bugs in a project.
2. **Maintainability** rules are enforced to detect code smells (code smell is an indication that there are chances of bugs, failure in future).
3. **Security** rules - related to vulnerability and security hotspots.

Except for security rules, others are expected to have zero or no false positive rules. To avoid this security rules they are given a different standard. As an issue is raised it is passed on to a human auditor to handle it.

# Introduction to Rules

You can click on the top "Rules" menu item to see rules. By default, you will see all the available rules.

**Language**: the language to which a rule applies.

**Type**: Bug, Vulnerability or Code Smell rules.

**Tag**:- it is possible to add tags to rules

**Repository**:- the engine that contributes rules to SonarQube.

**Default Severity**: the original severity of the rule - as defined by the plugin that contributes to this rule.

**Status**:- rules can have 3 different statuses(Beta, Deprecated, Ready).

**Template**:- display rule templates

**Quality Profile**:- Quality Profiles are collections of rules to apply during an analysis..

## More on Rules

The following two actions are available only if you have the right permissions.

**Add/Remove Tags**: It is possible to add existing tags on the rule or to create new ones.

**Note** that some rules have built-in tags that you cannot remove.

**Extend Description**: Extending rule descriptions is useful to users as you come to know how your organization is using a particular rule for instance.

**Note** that the extension will be available to non-admin users as a normal part of the rule details.

## Creation of Rules

SonarQube already has some minimal number of rules. You can activate/deactivate rules according to your need.

To create a custom rule you need following details:-

1. Name
2. Key
3. Description
4. Type
5. Severity

6. Status
7. factoryMethod

You can have a clear view by looking into the image.

## Adding Rules

There are two ways to extend coding rules:

1. [Coding rules using Java via a SonarQube plugin](#)
2. [Adding XPath rules via SonarQube web interface](#)

## Tags

- **Tags** differentiate between rules and issues.
- Rules raised tags and then its inherited by issues.
- Some tags are language specific.

To get more information on Tags you can go through this [link](#).

## Issues

### Statuses of an Issue

Issues go through five lifecycles:

1. **Open** - set by SonarQube when a new issue is being encountered.
2. **Confirmed** - manually being set to indicate that the issue is valid.
3. **Resolved** - manually being set to show that the issue will be closed in the next analysis.
4. **Reopened** - placed automatically by SonarQube when a resolved issue hasn't been corrected.
5. **Closed** - automatically being closed by SonarQube.

## Resolutions

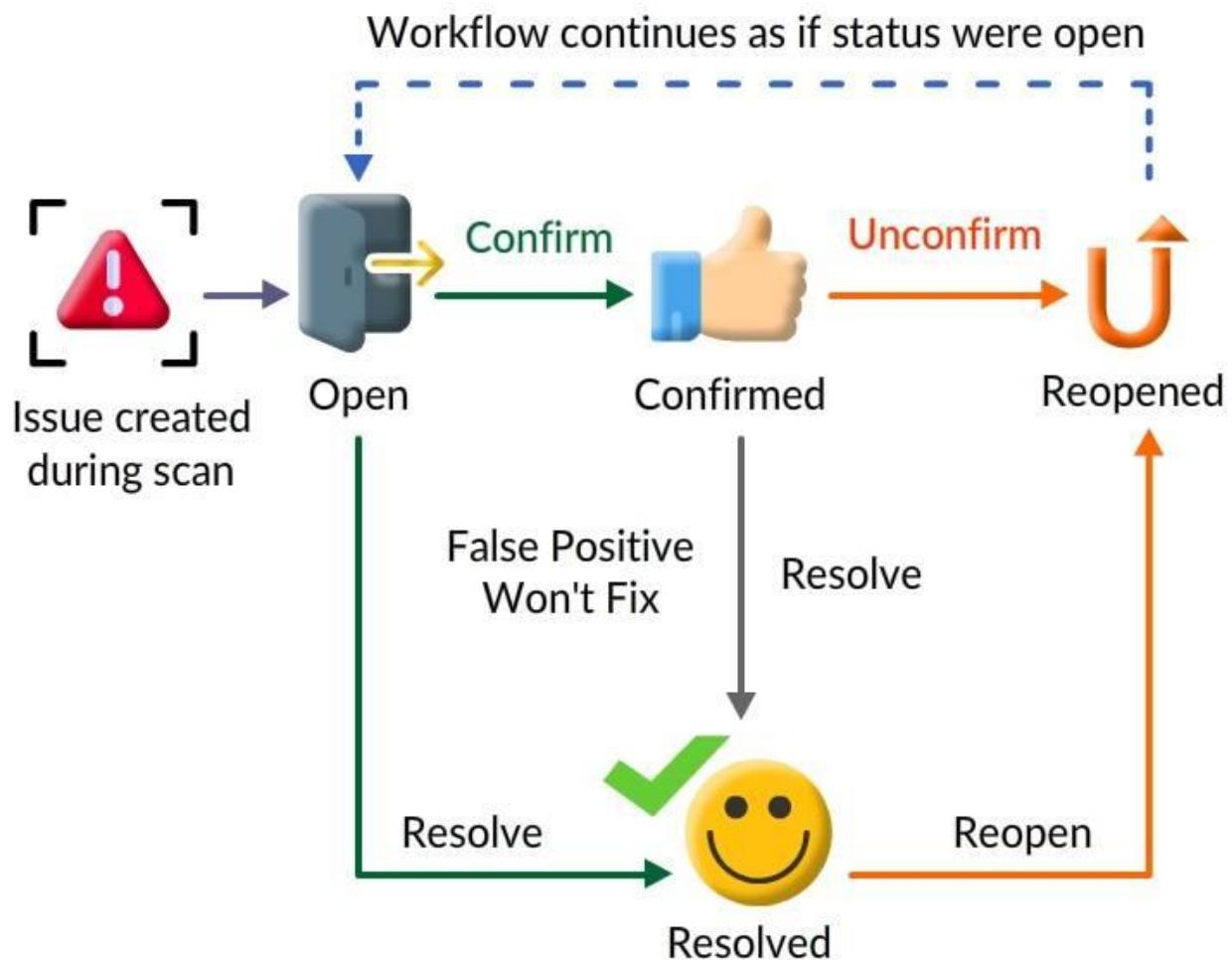
When an issue is *closed* it will have one of the two resolutions:

- **Fixed** - placed automatically when a subsequent analysis shows that the issue has been fixed.
- **Removed** - placed automatically when either the related coding rule or the file is no longer available. The rule may not be available because it has been removed from the profile or because the underlying plugin has been uninstalled. The file will not be available because it has been deleted from the project, moved to a different location or being renamed.

*Resolved issues* will have one of the two resolutions:

- **False Positive** - manually being set.
- **Won't Fix** - manually being set.

## Issue Workflow



When are issues automatically **closed**?

- When issue properly fixed (status: **fixed**).
- When issue no longer exists as the related rule has been removed (status: **removed**).
- When are issues automatically reopened?
- When in subsequent analysis, the issue that was resolved still exist.

# Gravity of Issues

Each issue has one of the five cases.

1. **BLOCKER**- The bug that has a major impact on the behavior of the application in production as in memory leak and unclosed JDBC connection.
2. **CRITICAL**- The bug which represents security flaw as in empty catch block, SQL Injection.
3. **MAJOR**- The flaw which has a great impact in developer productivity as in duplicated blocks, unused parameters.
4. **MINOR**- Quality flaws which have a lesser impact as in switch cases must have minimum of three cases.
5. **Info**- Its neither a bug nor a flaw, just a finding.

## Issues - Rules

*Issues come due to violation of rules, and rules are collected in profiles. Only certain users can edit profiles, but every user can view them.*

## Code Viewer

### Code Viewer

- It's an important part of SonarQube. It shows the files (source code and test files).
- The statistics of the file:
  1. **Lines of Code**
  2. **Bugs**
  3. **Vulnerabilities**
  4. **Code Smells**
  5. **Coverage**
  6. **Duplications**

# A Note to Code Viewer

It's advisable that every member in the team should be a user in the SonarQube so that they can uniquely review the code and every member can look through the review. If not accepted then can be added manually.

The code viewer has two facets:

- Current File
- Pinned File - you can adjust the pinned window.

## Current File

Let's discuss the layout of the file. It comprises of two parts:

1. **Header**
2. **Source Code**

The **Header** is displayed in the top bar. It consists of four blocks. The high-level statistics come under the header section( **lines**, **issues**, **coverage**, and **duplications**).

If any particular block is not required, it won't be shown.

The **source code** consists of SCM data, etc.

## Coverage

In **Java** projects, by using **JaCaCo** we can get **coverage per test**, which includes:

- Displaying the number of tests that will cover a specific line.
- Listing those tests.
- Navigating to the test files that define those tests.

## Duplications

The duplications metric tells the duplication in the code.

**Duplicated blocks**: The no of duplicated blocks of lines.

A block can be considered as duplicated if:

- **Non-Java project**: There should be 100 duplicated tokens. These should be divided on:-

1. 30 lines of code for COBOL.
  2. 20 lines of code for ABAP.
  3. 10 lines of code for another language.
- **Java Project:**
    - There should be minimum 10 duplicated statements whatever may be the number of tokens and lines.

**Duplicated lines(%)** - Density of duplication=(duplicated Lines/Lines )\*100.

## Administration

### What will You Learn?

In this topic, you will learn about the administration in SonarQube.

- Custom Metrics.
- Security.
- Authentication.
- LDAP Integration
- Server Log Management

### Custom Metrics

- It gathers the majority of measures in an automated way but there are some measures which are required to be injected manually.
- Before creating a metric, you need to save your measure.
- You need to log in as a system administrator.
- Go to **Administration > Configuration > Custom Metrics**
- Then to add a measure, sign in as project administrator **Administration > Custom Measures**.
- You can edit, delete an existing measure or enter a new one.

### Security

SonarQube manages **security** i.e. **authentication and authorization**. Security covers up two main areas:

1. It manages the access rights to components, pieces of information.
2. It enables customization for users.

Some examples are:

- Making a particular project inaccessible to anonymous users.
- Source code can be accessed by a selective user.
- Administering project privileges.

## Authentication

Log-in as System Administrator, then **Administration > Configuration > General Settings > Security**.

This mechanism can be handled by:

- via SONARQUBE built-in
- via external identity providers example GitHub.
- via HTTP headers.

The administrator can manage token's on user behalf via **Administration > Security > Users**.

You can click on user column and get to know the existing token of the user.

Default Admin credentials:

- **Login:** admin
- **Password:** admin

## LDAP Integration

**LDAP (Lightweight Directory Access Protocol)** is a client/server protocol that is used to access and manage directory information. It reads and edits directories over IP networks and runs directly over TCP/IP.

- You can configure SONARQUBE to LDAP server by configuring the correct values in `$SONARQUBE_HOME/conf/sonar.properties` file.
- Group (static) Mapping is supported not roles. Steps for the setup:
- You have to configure it by editing the changes in properties file.
- You need to restart the SonarQube server and check the log file.
- Login.

You can go through the [documentation](#) of JAVA parameters if its time out while running SonarQube analysis.

## Server Log Management

Server Side logging will be controlled according to the properties you have set in sonar.properties file.

One SonarQube process has four log files.

The properties that should be considered for server-side log level are:

- `sonar.log.level.app`- for process(Bootstrap,wrapper) of SONARQUBE.
- `sonar.log.level.web`- WebServer
- `sonar.log.level.ce`- ComputeEngineServer
- `sonar.log.level.es` - SearchServer
- `sonar.log.maxFiles`- the maximum number of files
- `sonar.log.rollingPolicy`- for control log rolling

(if `sonar.log.rollingPolicy=none`, the property is ignored)

## Sonar - Course Summary

### Course Summary

*Finally, you have arrived at the end of the course. In this course, you have learned:*

- How the issues can be checked parallelly developing the code.
- How to use Sonarqube.
- How it helps in maintaining a coding standard.
- All the features of Sonar.
- To resolve issues.
- To have ownership of your own code(to improve your own skills and quality of the code).
- How sonarqube provide additional value and professional support(reduces risk during deployment).