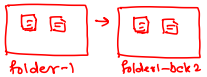
Source Code Management

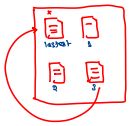
Why do we need Version Control System?

- Many people's version-control method of choice is to copy files into another directory
- This approach is very common because it is so simple
- But it is also incredibly error prone
- It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to
- To deal with this issue, programmers long ago developed VCS



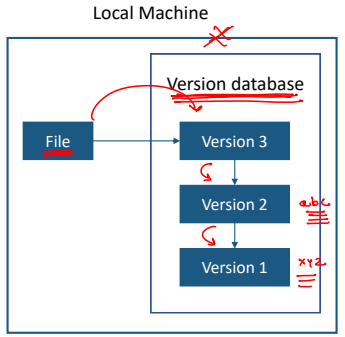
What is Version Control System?

- System that records changes to a file(s) over time so that you can recall specific versions later
- It allows you
  - to revert files to a previous state
  - to revert the entire project to a previous state
  - to compare changes over time
  - to see who last modified something that might be causing a problem
  - to see who introduced an issue and when
- Using a VCS also generally means that if you screw things up or lose files, you can easily recover



Local Version Control System

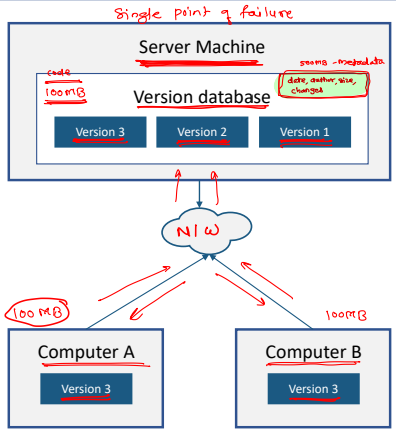
- Contains simple database that kept all the changes to files under revision control
- One of the more popular VCS tools was a system called RCS
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk
- It can then re-create what any file looked like at any point in time by adding up all the patches



Centralized Version Control Systems

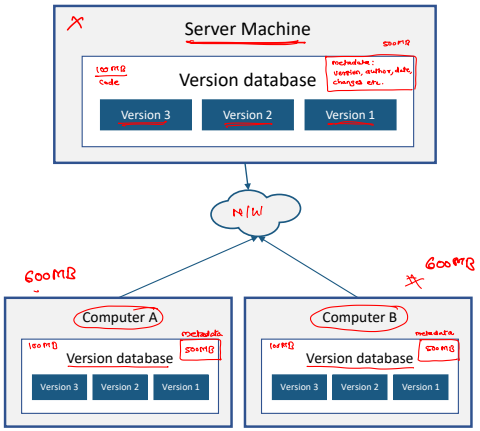
- People need to collaborate with developers on other systems
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place
- E.g. CVS, Subversion, and Perforce

disadvantages
- single point of failure
- not scalable



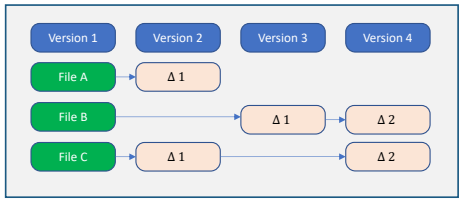
Distributed Version Control Systems

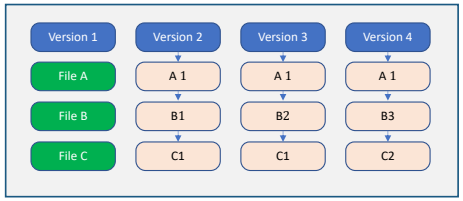
- Clients don't just check out the latest snapshot of the files, rather they fully mirror the repository
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- Every checkout is really a full backup of all the data
- Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project



What is Git?

- Git is one of the distributed version control systems
- The major difference between Git and any other VCS is the way Git thinks about its data
- Unlike other VCS tools, Git uses snapshots and not the differences
- Others think of the information they keep as a set of files and the changes made to each file over time
- Git thinks of its data more like a set of snapshots of a miniature filesystem
- Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot
- To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored





Git

Overview

- Git is a distributed revision control and source code management system
- Git was initially designed and developed by Linus Torvalds for Linux kernel development
- Git is a free software distributed under the terms of the GNU General Public License version 2

A little bit history about Git

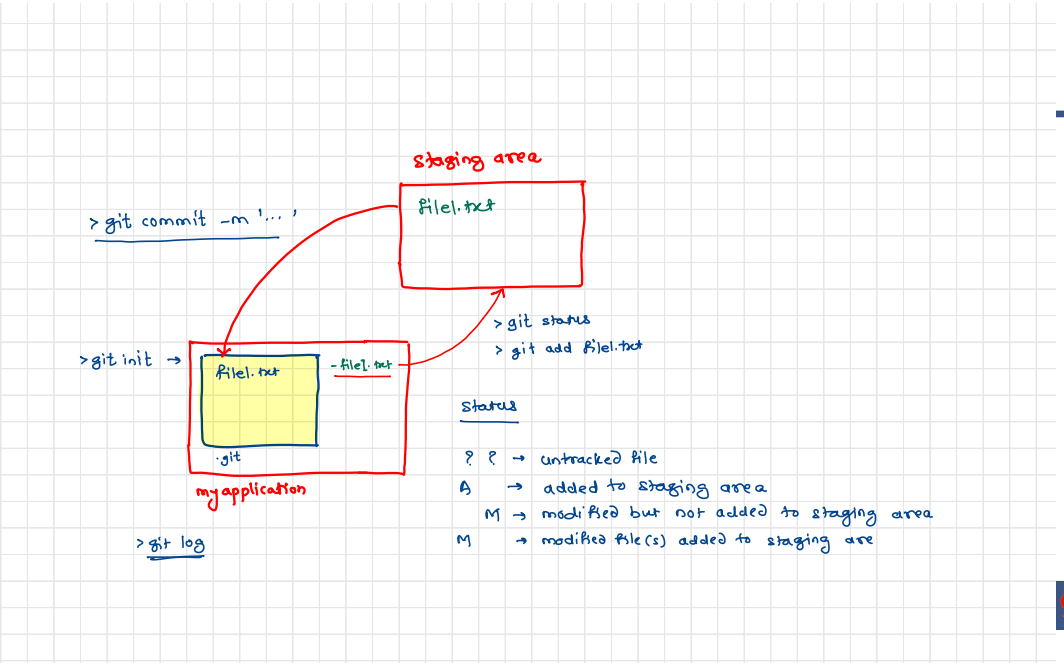
- The Linux kernel is an open source software project of very large scope
- From 1991–2002, changes to the software were passed around as patches and archived files
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper
- In 2005, the relationship with BitKeeper broken down and tool's free-of-charge status was revoked
- tool's free-of-charge status was revoked (and in particular Linus Torvalds) to develop their own tool based on some of the lessons they learned while using BitKeeper
- Some of the goals of the new system were
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

Characteristics

- Strong support for non-linear development (branching)
- Distributed development
- Compatibility with existent systems and protocols [https]
- Efficient handling of large projects
- Cryptographic authentication of history
- Toolkit-based design
- Pluggable merge strategies

Advantages

- Free and open source
- Fast and small
- Implicit backup
- Security
- No need of powerful hardware
- Easier branching



Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

- **Setup global properties**

> git config --global user.name <user name>
> git config --global user.email <user email>
> git config --global core.editor <editor>
> git config –global merge.tool vimdiff

Basic Commands

- **Initialize a repository**

> git init

- **Checking status**

> git status

- **Adding files to commit**

> git add .

- **Committing the changes**

> git commit –m '<log message>'

Basic Commands

- **Checking logs**

> git log

- **Checking difference**

> git diff

- **Moving item**

> git mv <source> <destination>

Terminologies

- Repository
  - Directory containing .git folder
- Object
  - Collection of key-value pairs
- Blobs (**B**inary **L**arge **Ob**ject)
  - Each version of a file is represented by blob
  - A blob holds the file data but doesn't contain any metadata about the file
  - It is a binary file, and in Git database, it is named as SHA1 hash of that file
  - In Git, files are not addressed by names. Everything is content-addressed
- Clone
  - Clone operation creates the instance of the repository
  - Clone operation not only checks out the working copy, but it also mirrors the complete repository
  - Users can perform many operations with this local repository
  - The only time networking gets involved is when the repository instances are being synchronized

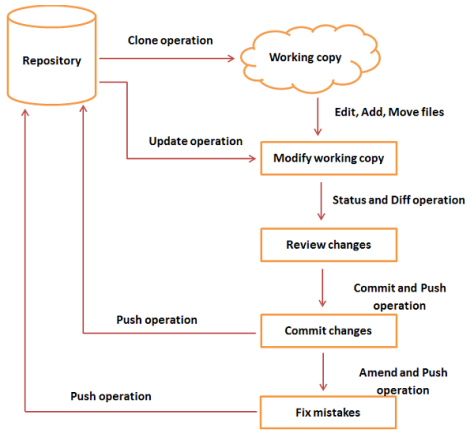
Terminologies

- Pull
  - Pull operation copies the changes from a remote repository instance to a local
  - The pull operation is used for synchronization between two repository instances
- Push
  - Push operation copies changes from a local repository instance to a remote
  - This is used to store the changes permanently into the Git repository
- HEAD
  - HEAD is a pointer, which always points to the latest commit in the branch
  - Whenever you make a commit, HEAD is updated with the latest commit
  - The heads of the branches are stored in **.git/refs/heads/** directory

Terminologies

- Commits
  - Commit holds the current state of the repository.
  - A commit is also named by **SHA1** hash
  - A commit object as a node of the linked list
  - Every commit object has a pointer to the parent commit object
  - From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit
- Branches
  - Branches are used to create another line of development
  - By default, Git has a master branch
  - Usually, a branch is created to work on a new feature
  - Once the feature is completed, it is merged back with the master branch and we delete the branch
  - Every branch is referenced by HEAD, which points to the latest commit in the branch
  - Whenever you make a commit, HEAD is updated with the latest commit

Life Cycle



Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

- **Setup global properties**

> git config --global user.name <user name>
> git config --global user.email <user email>
> git config --global core.editor <editor>
> git config –global merge.tool vimdiff

Basic Commands

- **Initialize a repository**

> git init

- **Checking status**

> git status

- **Adding files to commit**

> git add .

- **Committing the changes**

> git commit –m ’<log message>’

Basic Commands

- **Checking logs**

> git log

- **Checking difference**

> git diff

- **Moving item**

> git mv <source> <destination>

Basic Commands

- **Rename item**

> git mv <old> <new>

- **Delete Item**

> git rm <item>

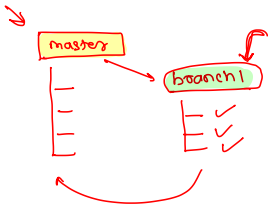
- **Remove unwanted changes**

> git checkout file

Branch

: collection of commits

- Allows another line of development
- A way to write code without affecting the rest of your team
- Generally used for feature development
- Once confirmed the feature is working you can merge the branch in the master branch and release the build to customers



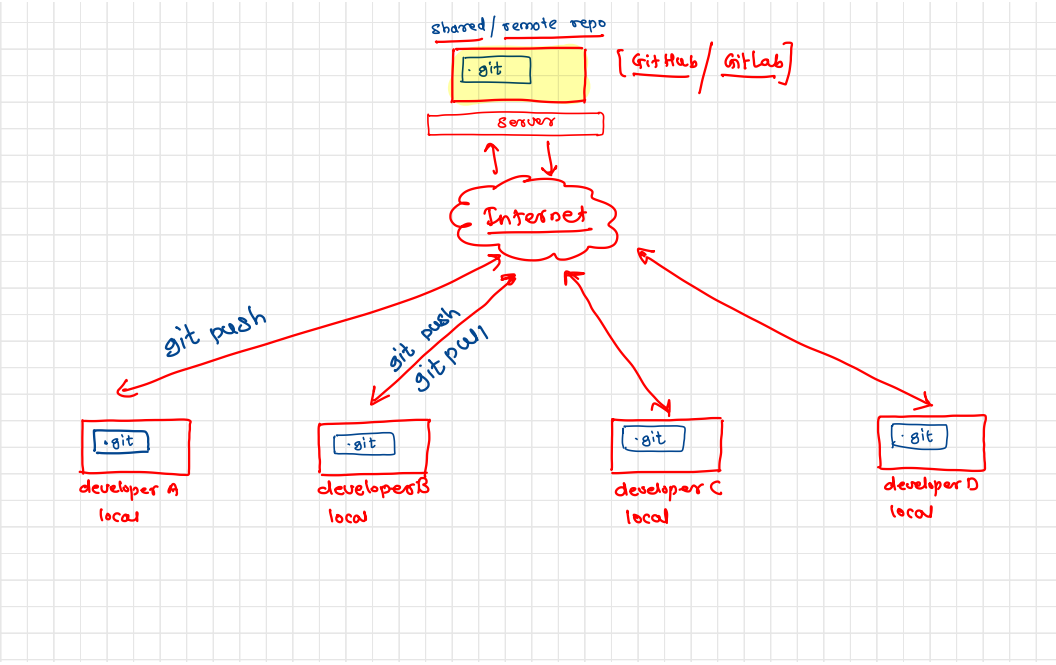
* master / main
  - contains
    - all latest changes
    - all working changes
    - non-crashing changes
    - stable code / tested code
  - clean code

Why is it required ?

- So that you can work independently
- There will not be any conflicts with main code
- You can keep unstable code separated from stable code
- You can manage different features keeping away the main line code and there wont be any impact of the features on the main code

Branch management commands

- **Create a branch**
  > git branch <branch name>
- **Checkout a branch**
  > git checkout <branch name>
- **Merge a branch**
  > git merge <branch name>
- **Delete a branch**
  > git branch –d <branch name>



GitHub

Overview

- GitHub is a web-based hosting service for version control using Git
- It provides access control and several collaboration features
  - bug tracking
  - feature requests
  - task management
  - wikis for every project
- Developer uses github for sharing repositories with other developers

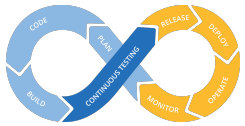
Workflow

- Create a project on GitHub
- Clone repository on the local machine
- Add/modify code locally
- Commit the code locally
- Push the code to the GitHub repository
- Allow other developers to get the code by using git pull operations

1. Create project on GitHub
2. git clone
3. change code
4. git status
5. git add .
6. git commit -m '...'
7. git push

Workflow commands

- **Add remote repository**
  > git remote add <name> <url>
- **Clone remote repository**
  > git clone <url>
- **Push the changes**
  > git push <name> <branch>
- **Pull the changes**
  > git pull



DevOps

Problems

- Managing and tracking changes in the code is difficult
- Incremental builds are difficult to manage, test and deploy
- Manual testing and deployment of various components/modules takes a lot of time
- Ensuring consistency, adaptability and scalability across environments is very difficult task
- Environment dependencies makes the project behave differently in different environments

Solutions to the problem

- Managing and tracking changes in the code is difficult: **SCM tools** [Git]
- Incremental builds are difficult to manage, test and deploy: **Jenkins** [CI/CD pipeline]
- Manual testing and deployment of various components/modules takes a lot of time: **Selenium**
- Ensuring consistency, adaptability and scalability across environments is very difficult task: **Puppet**
- Environment dependencies makes the project behave differently in different environments: **Docker**

SCM = git →

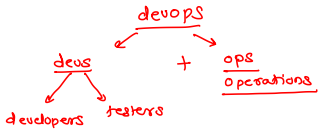
containers = Docker

orchestration = kubernetes

CI/CD = Jenkins

Overview

- DevOps is a combination of two words development and operations
- Promotes collaboration between Development and Operations Team to deploy code to production faster in an automated & repeatable way
- DevOps helps to increases an organization's speed to deliver applications and services
- It allows organizations to serve their customers better and compete more strongly in the market
- Can be defined as an alignment of development and IT operations with better communication and collaboration



Why DevOps is Needed?

- Before DevOps, the development and operation team worked in complete isolation
- Testing and Deployment were isolated activities done after design-build. Hence they consumed more time than actual build cycles.
- Without using DevOps, team members are spending a large amount of their time in testing, deploying, and designing instead of building the project.
- Manual code deployment leads to human errors in production
- Coding & operation teams have their separate timelines and are not in synch causing further delays

What is DevOps ?

- DevOps is not a goal but a never-ending process of continuous improvement
- It integrates Development and Operations teams
- It improves collaboration and productivity by
  - Automating infrastructure
  - Automating workflow
  - Continuously measuring application performance

Common misunderstanding

- DevOps is not a role, person or organization
- DevOps is not a separate team
- DevOps is not a product or a tool
- DevOps is not just writing scripts or implementing tools

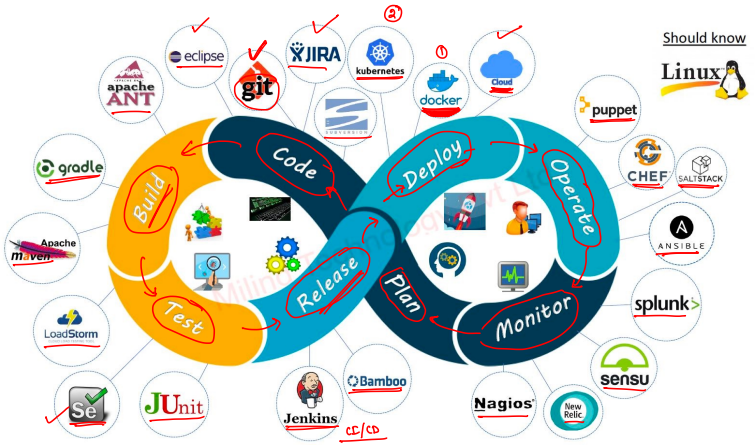
Reasons to use DevOps

- **Predictability:** DevOps offers significantly lower failure rate of new releases
- **Reproducibility:** Version everything so that earlier version can be restored anytime
- **Maintainability:** Effortless process of recovery in the event of a new release crashing or disabling the current system
- **Time to market:** DevOps reduces the time to market up to 50% through streamlined software delivery. This is particularly the case for digital and mobile applications
- **Greater Quality:** DevOps helps the team to provide improved quality of application development as it incorporates infrastructure issues
- **Reduced Risk:** DevOps incorporates security aspects in the software delivery lifecycle. It helps in reduction of defects across the lifecycle
- **Resiliency:** The Operational state of the software system is more stable, secure, and changes are auditable

Reasons to use DevOps

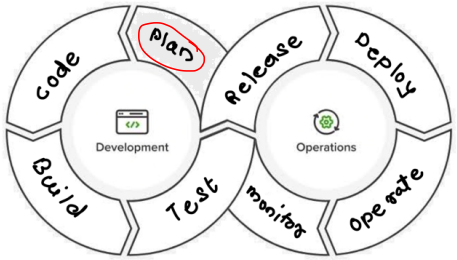
- **Cost Efficiency:** DevOps offers cost efficiency in the software development process which is always an aspiration of IT companies' management
- **Breaks larger code base into small pieces:** DevOps is based on the agile programming method. Therefore, it allows breaking larger code bases into smaller and manageable chunks

DevOps Lifecycle



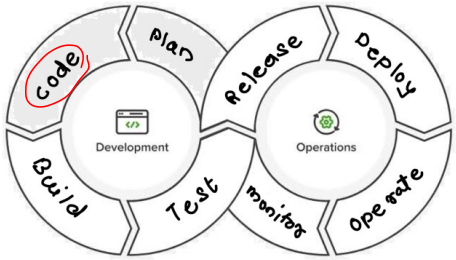
DevOps Lifecycle - Plan

- First stage of DevOps lifecycle where you plan, track, visualize and summarize your project before you start working on it
- Planning tools
  - Google sheet
  - Box
  - Dropbox
  - Trello
  - Jira
  - Planio



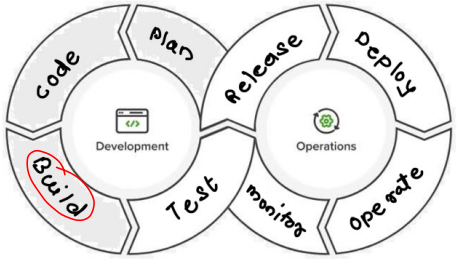
DevOps Lifecycle - Code

- Second stage where developer writes the code using favorite programming language
- Coding Tools
  - IDEs: Eclipse, Visual Studio etc.
  - SCM: Git, Subversion, CVS etc.
  - Package management: npm etc.



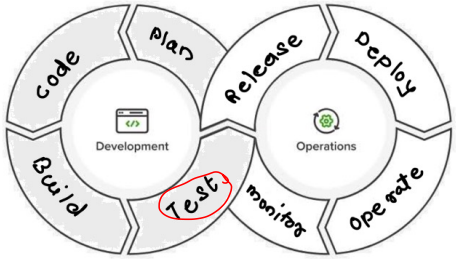
DevOps Lifecycle -Build

- Integrating the required libraries
- Compiling the source code
- Create deployable packages
- Build tools
  - Maven
  - Gradle
  - Ant



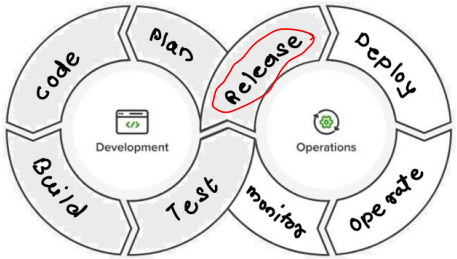
DevOps Lifecycle - Test

- Process of executing automated tests
- The goal here is to get the feedback about the changes as quickly as possible
- Testing tools
  - JMeter
  - Selenium
  - JUnit
  - QUnit
  - NUnit
  - Appium



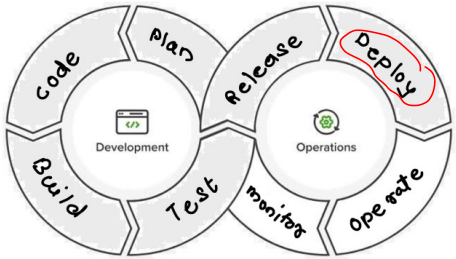
DevOps Lifecycle - Release

- This phase helps to integrate code into a shared repository using which you can detect and locate errors quickly and easily
- Release tools
  - Jenkins
  - Travis CI
  - Bamboo
  - GitLab CI



DevOps Lifecycle - Deploy

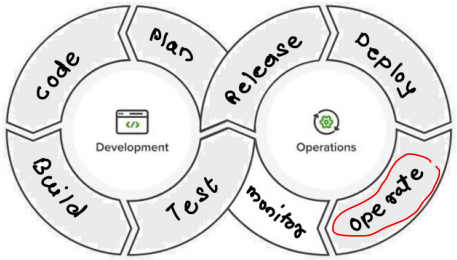
- Manage and maintain development and deployment of software systems and server in any computational environment
- Deployment tools
  - Docker
  - Kubernetes
  - Virtual Machines
- Configuration management tools
  - Puppet
  - Chef
  - Ansible



DevOps Lifecycle - Operate

- This stage where the updated system gets operated
- Operating Tools
  - Puppet
  - Chef
  - Ansible

environment creation
- dev
- QA
- staging
- pre-production
- production



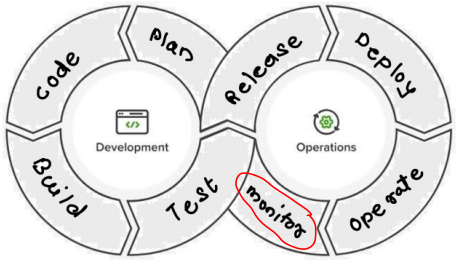
Certifications

- Linux : RHCSA $150 / RHCE $200 , Linux+ $250
- cloud : AWS : 45 certificates :- Associate $150 + professional $300 → architect, developer, sysops
  - Gcp : 20 certificates
  - Azure : 18 certificates
- Docker : DCA $200
- Kubernetes : CKA $300, CKAD $300, CKS $200
- Jenkins : CJE $110
- Ansible : Red Hat $100
- Puppet & chef : Red Hat $100
- networking : CCNA $150, MCSE $99, Network+ $289
- hardware : A+ $289
- security : CISSP $749, Security+ $386

DevOps Lifecycle - Monitor

- It ensures that the application is performing as expected and the environment is stable
- It quickly determines when a service is unavailable and understand the underlying causes
  - health check
  - logging
- Monitoring tools
  - Nagios
  - Sensu
  - Splunk
  - DataDog

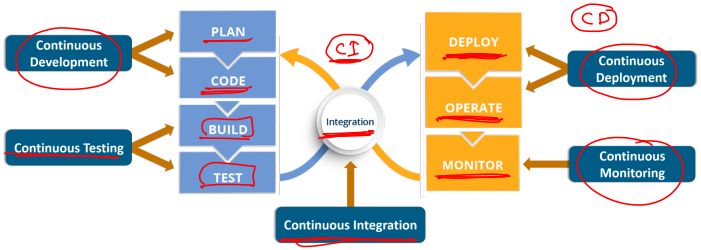
new relic



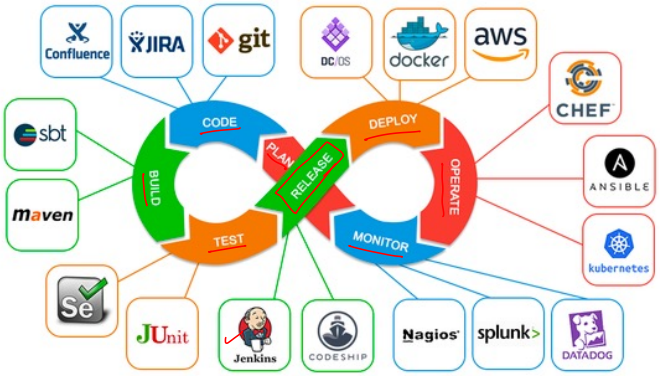
DevOps Terminologies

- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Continuous Monitoring

Continuous learning :)

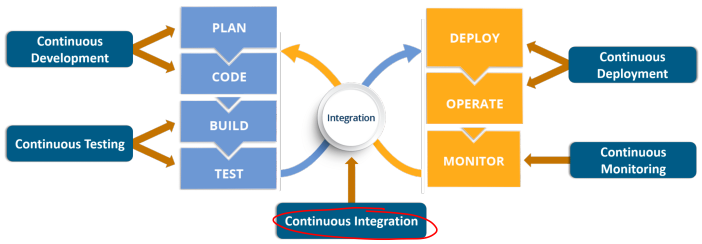


DevOps Lifecycle



DevOps Terminologies

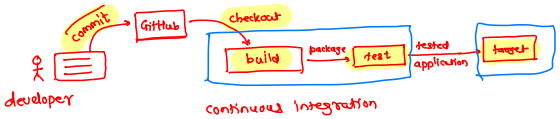
- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Continuous Monitoring



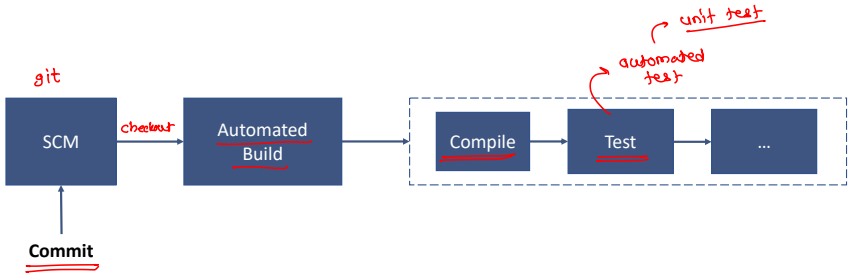
Continuous Integration

Overview

- It is the process of automating the building and testing of code, each time developer commits changes to the version control system
- CI is necessary to bring out issues encountered during the integration as early as possible
- CI requires developers to have frequent builds
- The common practice is that whenever a code commit occurs, a build should be triggered



Continuous Integration

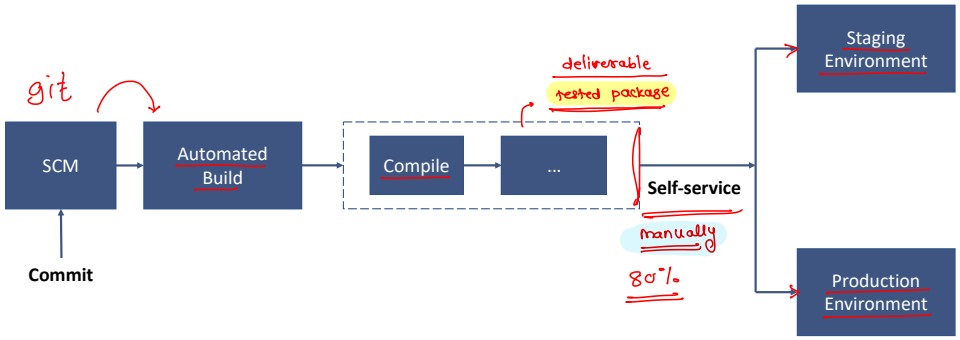


CI Best Practices

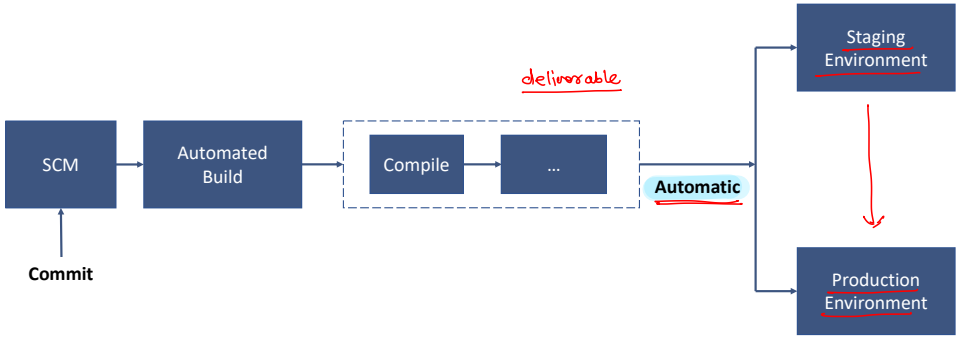
- Frequent commits
- No long running branches
- Automated test execution
- Fix broken builds

master / main
- latest code
- stable code
- tested code

Continuous Delivery

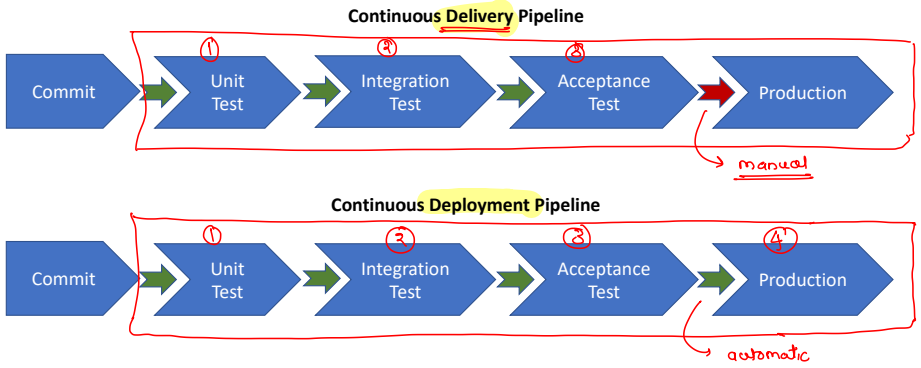


Continuous Deployment



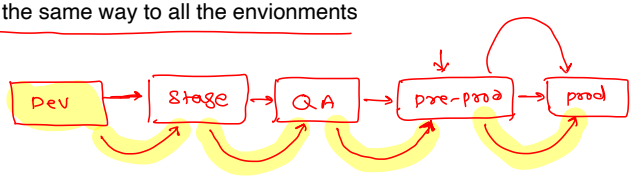
CI/CD Pipeline

pipeline = collection of stages



CD Best Practices

- Version control all configuration
- Stop the line immediately for a failure
- Build your binaries only once
- Deploy the same way to all the envionments



Importance

- Improves product quality
  - Improves the product quality by running the various unit test cases every time developer commits changes
- Increase productivity
  - Automating build of code saves a lot of time, thereby increasing productivity
  - Developer can utilize the time more to develop the code
- Reduces risk
  - Eliminates the potential human errors by automating test

Popular CI tools

TeamCity

Bamboo

GitLab CI

Travis CI

Jenkins

What is Jenkins ?

- Jenkins is a powerful application that allows continuous integration and continuous delivery of projects
- It is a free and open source application that can handle any kind of build or continuous integration

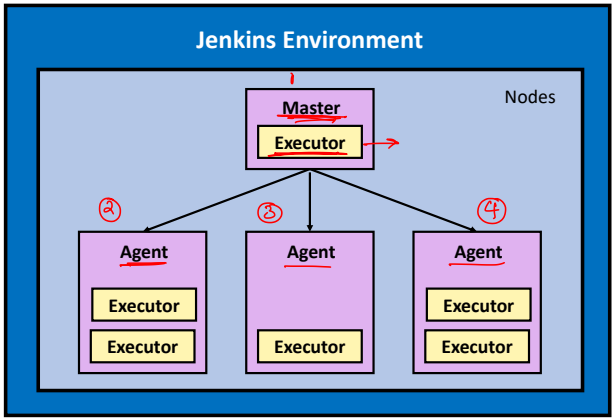
Where is it came from ?

- It was first started as project Hudson at Sun Microsystems in 2004 and was first released in Feb 2005
- In 2010, Oracle acquired Sun Microsystems
- In 2011, Oracle created fork of Hudson as Jenkins, since when these two projects exist as two independent projects
- On April 20, 2016 version 2 was released with the *Pipeline* plugin enabled by default

Features

- Easy installation on different operating systems
- Supports pipelines as code that uses **domain-specific language** (**DSL**) to model application delivery pipelines as code
- Easily extensible with the use of third-party plugins
- Easy to configure the setup environment in the user interface
- Master slave architecture supports distributed builds to reduce the load on CI servers
- Build scheduling based on cron expressions
- Shell and Windows command execution that makes any command-line tool integration in the pipeline very easy
- Notification support related to build status

Jenkins Environment



Terminologies

- **Node**
  - Node is the generic term that is used in Jenkins to mean any system that can run Jenkins jobs
  - This covers both masters and agents, and is sometimes used in place of those terms
  - Furthermore, a node might be a container, such as one for Docker
- **Master**
  - A Jenkins *master* is the primary controlling system for a Jenkins instance
  - It has complete access to all Jenkins configuration and options and the full list of jobs
  - It is the default location for executing jobs if another system is not specified
  - Master node must be present in Jenkins installation
- **Agent**
  - Is also known as Jenkins slave
  - This refers to any non-master system
  - The idea is that these systems are managed by the master system and allocated as needed, or as specified, to handle processing the individual jobs

Terminologies

- **Executor**
  - It is a slot in which to run a job on a node/agent
  - A node can have zero or more executors
  - The number of executors defines how many concurrent jobs can be run on that node
  - When the master funnels jobs to a particular node, there must be an available executor slot in order for the job to be processed immediately. Otherwise, it will wait until an executor becomes available.

Jenkins Pipeline

What is Jenkins pipeline ?

- Jenkins is, fundamentally, an automation engine which supports a number of automation patterns
- Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines
- It is a suite of plugins which supports implementing and integrating continuous delivery pipelines
- A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers
- Every change to your software (committed in source control) goes through a complex process on its way to being released
- This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment

Pipeline Features

- **Code**
  - Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable**
  - Pipelines can survive both planned and unplanned restarts of the Jenkins master.
- **Pausable**
  - Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile**
  - Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible**
  - The Pipeline plugin supports custom extensions to its DSL [1] and multiple options for integration with other plugins.

Pipeline concepts

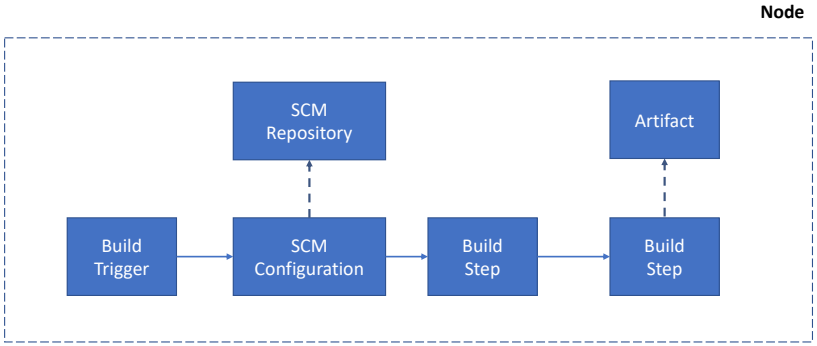
- **Pipeline**
  - A Pipeline is a user-defined model of a CD pipeline
  - A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it
  - **pipeline** block is used to create a pipeline
- **Node**
  - A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline
  - **node** block is used to define a node which can be used while executing job
- **Stage**
  - A **stage** block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. Build, Test, Deploy stages), which is used by many plugins to visualize or present Jenkins Pipeline status
- **Step**
  - A step in the process
  - A single task, fundamentally, a step tells Jenkins what to do at a particular point in time
  - **step** block can be used to create a step

Job

- Also known as Project or Item or Task
- It represents the steps used to build the code
- To create a new job, use option “new item”
- Project in Jenkins has different types
  - Freestyle Project
  - Pipeline
  - Multi-configuration Project
  - Folder
  - GitHub Organization
  - Multibranch Pipeline
- Demo
  - Create a freestyle project to print “hello world”

Build

- Execution of an automated task or multiple tasks
- Jenkins will run the job to create a build



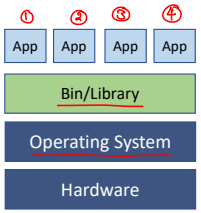
What is Kubernetes ?

- Portable, extensible, open-source platform for managing containerized workloads and services
- Facilitates both declarative configuration and automation
- It has a large, rapidly growing ecosystem
- Kubernetes services, support, and tools are widely available
- The name Kubernetes originates from Greek, meaning helmsman or pilot
- Google open-sourced the Kubernetes project in 2014

Sunbeam Infotech www.sunbeaminfo.com

Traditional Deployment

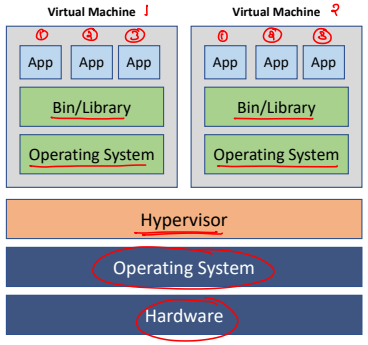
- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform
- A solution for this would be to run each application on a different physical server
- But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers



Sunbeam Infotech www.sunbeaminfo.com

Virtualized Deployment

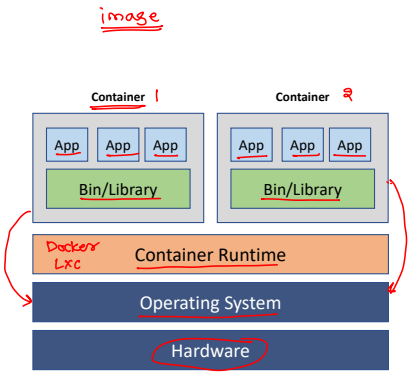
- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application
- Virtualization allows better utilization of resources in a physical server and allows better scalability because
  - an application can be added or updated easily
  - reduces hardware costs
- With virtualization you can present a set of physical resources as a cluster of disposable virtual machines
- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware



Sunbeam Infotech www.sunbeaminfo.com

Container deployment

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications
- Therefore, containers are considered lightweight
- Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions



Container benefits

- Increased ease and efficiency of container image creation compared to VM image use
- Continuous development, integration, and deployment
- Dev and Ops separation of concerns
- Observability not only surfaces OS-level information and metrics, but also application health and other signals
- Cloud and OS distribution portability
- Application-centric management:
- Loosely coupled, distributed, elastic, liberated micro-services
- Resource isolation: predictable application performance

What Kubernetes provide?

- **Service discovery and load balancing**
  - Kubernetes can expose a container using the DNS name or using their own IP address
  - If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable
- **Storage orchestration**
  - Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more
- **Automated rollouts and rollbacks**
  - You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate
- **Automatic bin packing**
  - You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks
  - You tell Kubernetes how much CPU and memory (RAM) each container needs
  - Kubernetes can fit containers onto your nodes to make the best use of your resources

What Kubernetes provide?

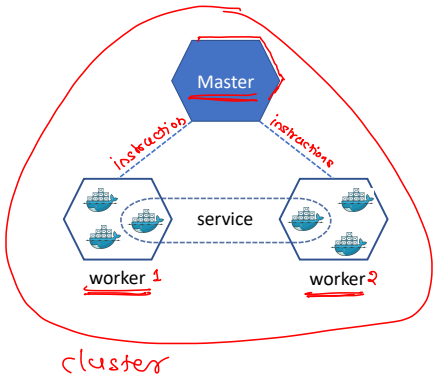
- **Self-healing**
  - Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve
- **Secret and configuration management**
  - Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys
  - You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration

What Kubernetes is not

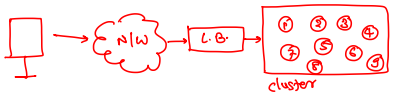
- Does not limit the types of applications supported
- Does not deploy source code and does not build your application → CI/CD pipeline
- Does not provide application-level services as built-in services
- Does not dictate logging, monitoring, or alerting solutions
- Does not provide nor mandate a configuration language/system
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems

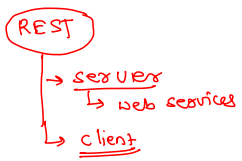
Kubernetes Cluster

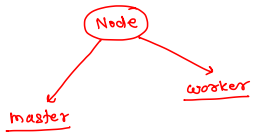
- When you deploy Kubernetes, you get a cluster.
- A cluster is a set of machines (nodes), that run containerized applications managed by Kubernetes
- A cluster has at least one worker node and at least one master node
- The worker node(s) host the pods that are the components of the application
- The master node(s) manages the worker nodes and the pods in the cluster
- Multiple master nodes are used to provide a cluster with failover and high availability

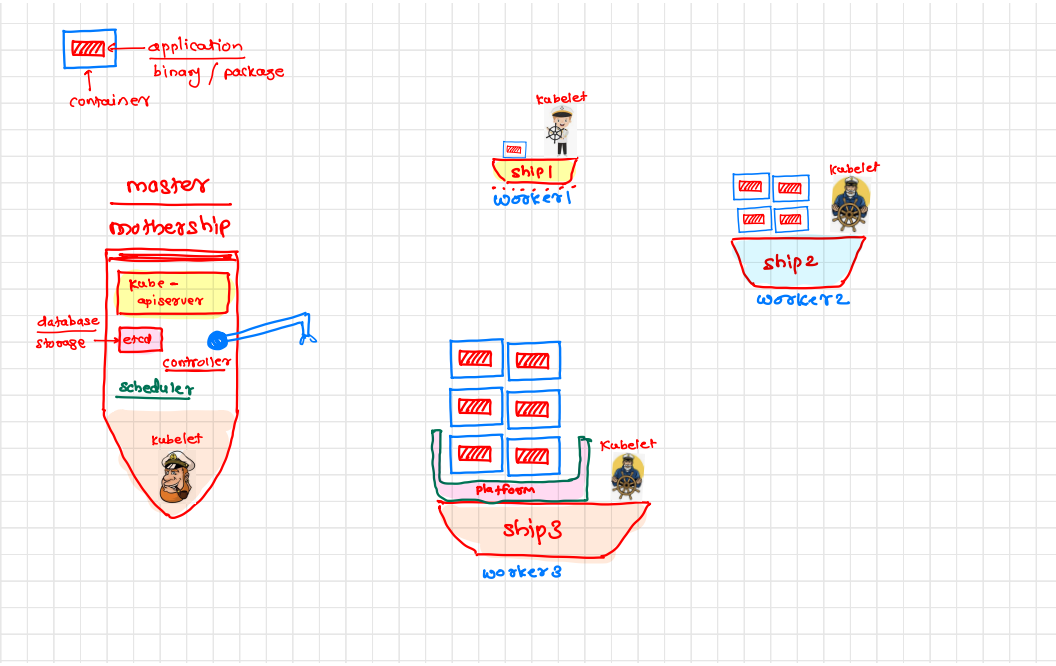


Kubernetes Architecture

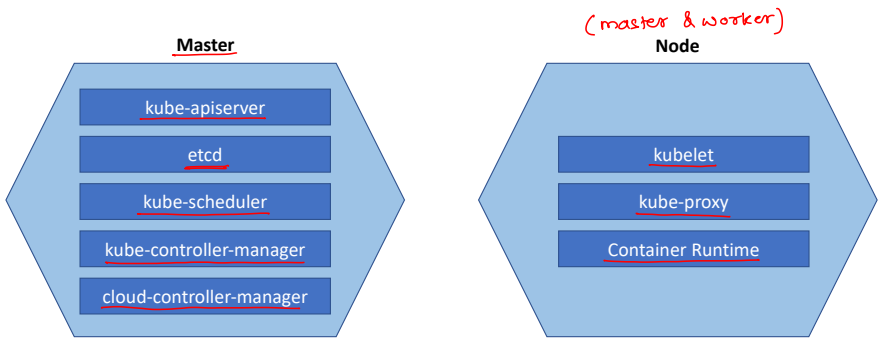








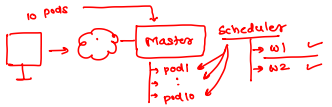
Kubernetes Components



Master Components

- Master components make global decisions about the and they detect and respond to cluster events
- Master components can be run on any machine in the cluster

1. **kube-apiserver**
   - The API server is a component that exposes the Kubernetes API
   - The API server is the front end for the Kubernetes
2. **etcd**
   - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
3. **kube-scheduler**
   - Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on



Master Components

4. **kube-controller-manager**
   - Component on the master that runs controllers
   - Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process
   - Types
     - Node Controller: Responsible for noticing and responding when nodes go down.
     - Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system
     - Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods)
     - Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces
5. **cloud-controller-manager**
   - Runs controllers that interact with the underlying cloud providers
   - The cloud-controller-manager binary is an alpha feature introduced in Kubernetes release 1.6

Node Components

- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment
- **kubelet**
  - An agent that runs on each node in the cluster
  - It makes sure that containers are running in a pod
- **kube-proxy**
  - Network proxy that runs on each node in your cluster, implementing part of the Kubernetes service concept
  - kube-proxy maintains network rules on nodes
  - These network rules allow network communication to your Pods from network sessions inside or outside of your cluster
- **Container Runtime**
  - The container runtime is the software that is responsible for running containers
  - Kubernetes supports several container runtimes: Docker, containerd, rktlet, cri-o etc.

Create Cluster

- Use following commands on both master and worker nodes

> sudo apt-get update && sudo apt-get install -y apt-transport-https curl

> curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

> cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/kubernetes-xenial main EOF

> sudo apt-get update
> sudo apt-get install -y kubelet kubeadm kubectl
> sudo apt-mark hold kubelet kubeadm kubectl

Initialize Cluster Master Node

- Execute following commands on master node

> kubeadm init --apiserver-advertise-address=<ip-address> --pod-network-cidr=10.244.0.0/16

> mkdir -p $HOME/.kube

> sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

> sudo chown $(id -u):$(id -g) $HOME/.kube/config

- Install pod network add-on

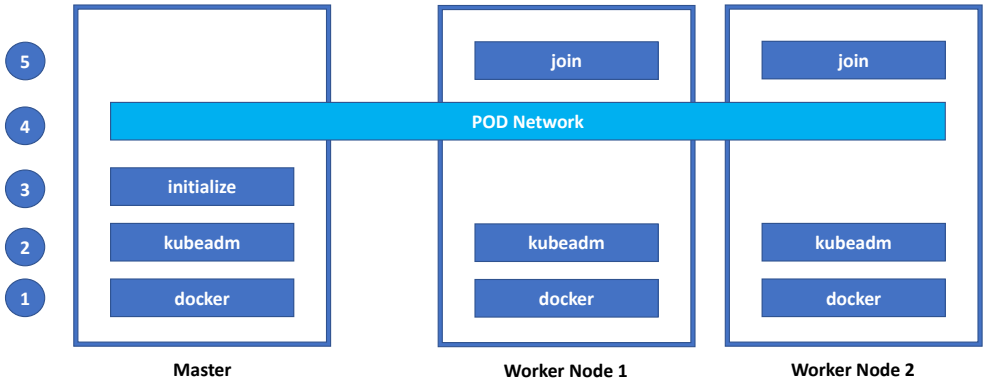
> kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml

Add worker nodes

- Execute following command on every worker node

> kubeadm join --token <token> <control-plane-host>:<control-plane-port> --discovery-token-ca-cert-hash sha256:<hash>

Steps to install Kubernetes

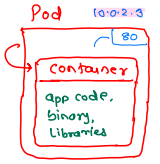


Kubernetes Objects

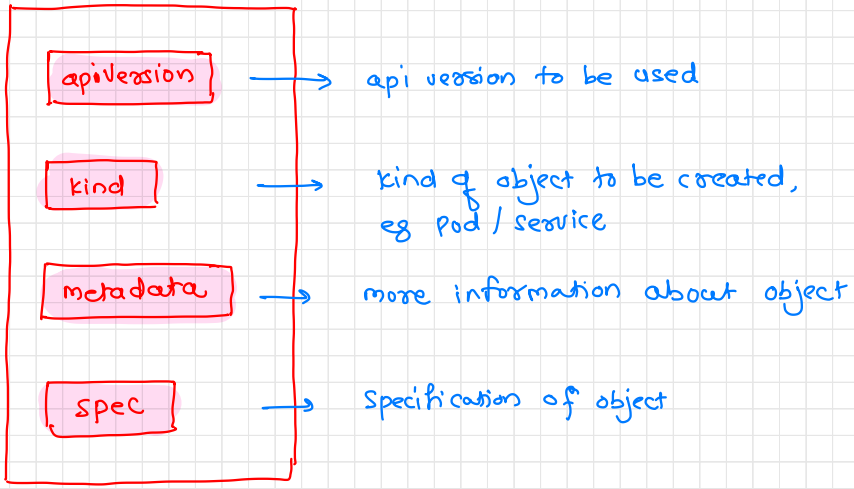
- The basic Kubernetes objects include
  - Pod
  - Service
  - Volume
  - Namespace
- Kubernetes also contains higher-level abstractions build upon the basic objects
  - Deployment
  - DaemonSet
  - StatefulSet
  - ReplicaSet
  - Job

Pod

- A Pod is the basic execution unit of a Kubernetes application
- The smallest and simplest unit in the Kubernetes object model that you create or deploy
- A Pod represents processes running on your Cluster
- Pod represents a unit of deployment
- A Pod encapsulates
  - application's container (or, in some cases, multiple containers)
  - storage resources
  - a unique network IP
  - options that govern how the container(s) should run

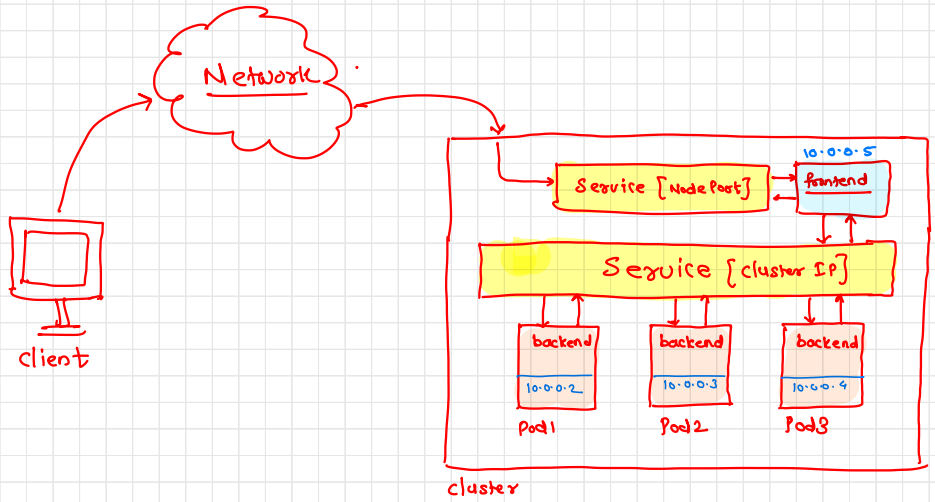


a pod may run one or more containers within it.



YAML to create Pod

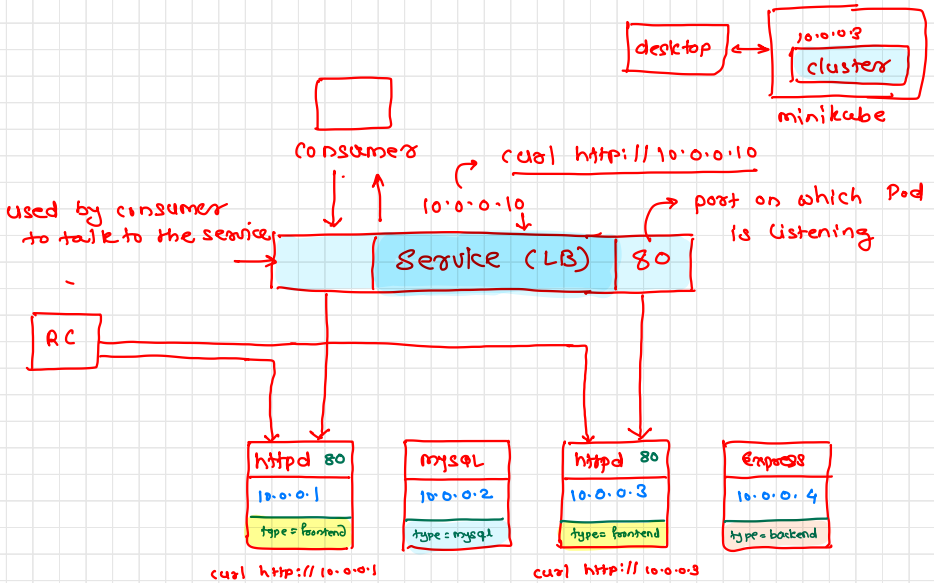
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: httpd
```

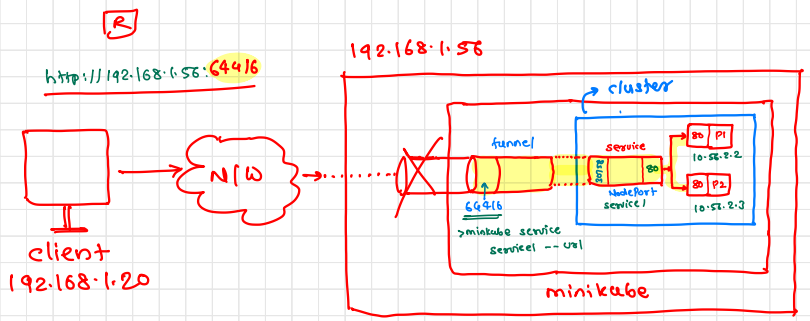


Service

- An abstract way to expose an application running on a set of Pods as a network service
- Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service)
- Service Types
  1. ClusterIP → available within the same cluster
     - Exposes the Service on a cluster-internal IP
     - Choosing this value makes the Service only reachable from within the cluster
  - LoadBalancer → cloud
    - Used for load balancing the containers
  2. NodePort

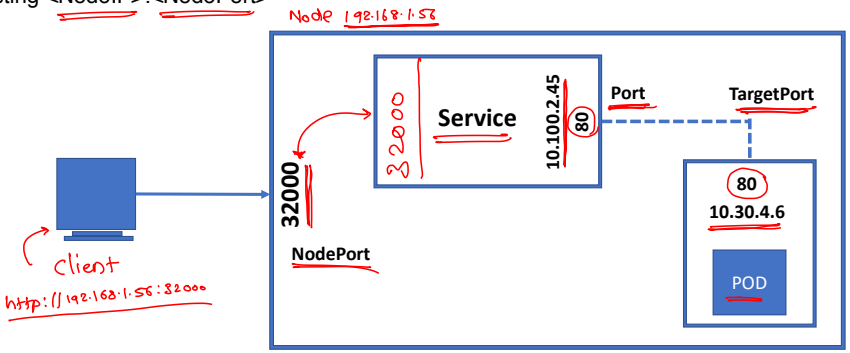
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```



Service Type: NodePort

- Exposes the Service on each Node's IP at a static port (the NodePort)
- You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>



Replication Controller

- A *ReplicationController* ensures that a specified number of pod replicas are running at any one time
- In other words, a ReplicationController makes sure that a pod or a homogeneous set of pods is always up and available
- If there are too many pods, the ReplicationController terminates the extra pods
- If there are too few, the ReplicationController starts more pods
- Unlike manually created pods, the pods maintained by a ReplicationController are automatically replaced if they fail, are deleted, or are terminated

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```


Deployment

- A Deployment provides declarative updates for Pods and ReplicaSets
- You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate
- You can use deployment for
  - Rolling out ReplicaSet
  - Declaring new state of Pods
  - Rolling back to earlier deployment version
  - Scaling up deployment policies
  - Cleaning up existing ReplicaSet

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website-deployment
spec:
  selector:
    matchLabels:
      app: website
  replicas: 10
  template:
    metadata:
      name: website-pod
      labels:
        app: website
    spec:
      containers:
      - name: website-container
        image: pythoncpp/test_website
        ports:
        - containerPort: 80
```

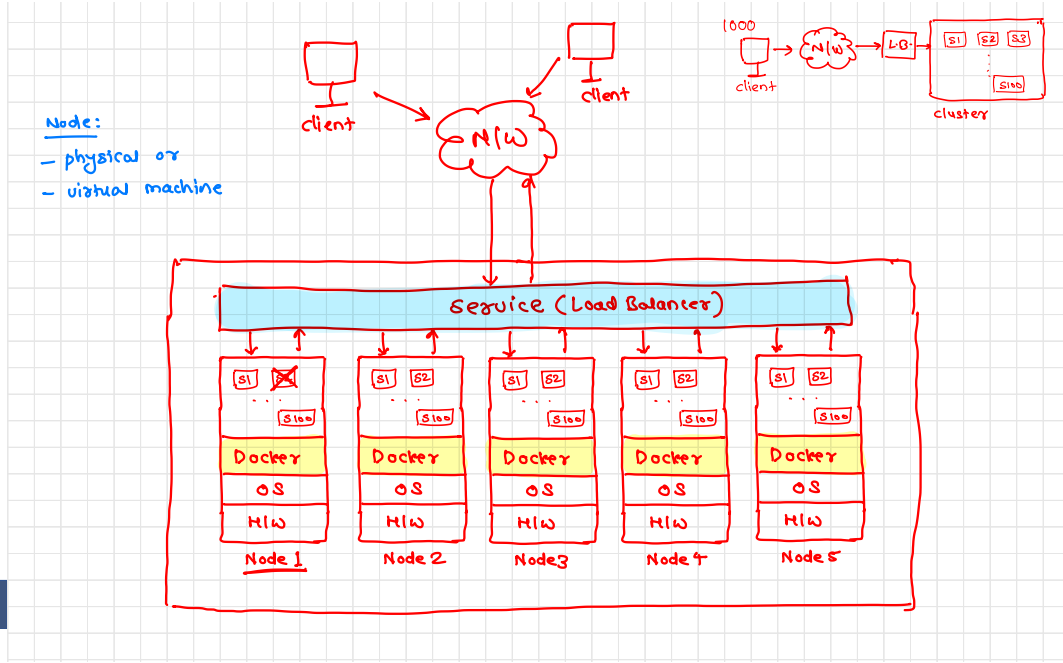

Volume

- On-disk files in a Container are ephemeral, which presents some problems for non-trivial applications when running in Containers
- Problems
  - When a Container crashes, kubelet will restart it, but the files will be lost
  - When running Containers together in a Pod it is often necessary to share files between those Containers
- The Kubernetes Volume abstraction solves both of these problems
- A volume outlives any Containers that run within the Pod, and data is preserved across Container restarts

Namespace

- Namespaces are intended for use in environments with many users spread across multiple teams, or projects
- Namespaces provide a scope for names
- Names of resources need to be unique within a namespace, but not across namespaces
- Namespaces can not be nested inside one another and each Kubernetes resource can only be in one namespace
- Namespaces are a way to divide cluster resources between multiple users

Container Orchestration



Overview

- Container orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments
  - ↑ starting / restarting / deleting
- Software teams use container orchestration to control and automate many tasks
  - Provisioning and deployment of containers
  - Redundancy and availability of containers
  - Scaling up or removing containers to spread application load evenly across host infrastructure
  - Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies
  - Allocation of resources between containers
  - External exposure of services running in a container with the outside world
  - Load balancing of service discovery between containers
  - Health monitoring of containers and hosts
  - Configuration of an application in relation to the containers running it

Orchestration Tools

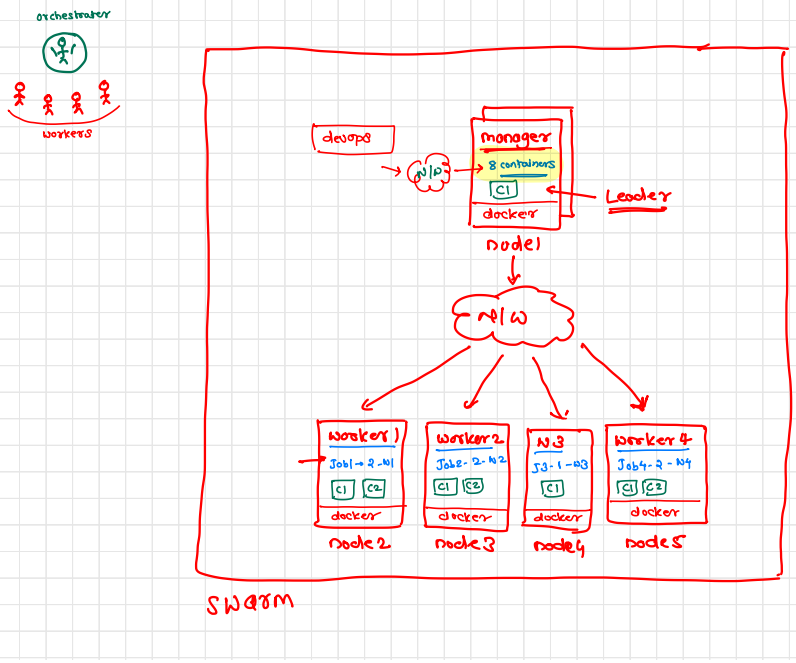
- Docker Swarm
- Kubernetes
- Mesos
- Marathon

Docker Swarm

Overview

- Docker Swarm is a container orchestration engine
- It takes multiple Docker Engines running on different hosts and lets you use them together
- The usage is simple: declare your applications as stacks of services, and let Docker handle the rest
- Services can be anything from application instances to databases
  - frontend | backend | database

cluster / docker swarm



What is a swarm?

- A swarm consists of multiple Docker hosts which run in **swarm mode**
- A given Docker host can be a manager, a worker, or perform both roles
- When you create a service, you define its optimal state [no of containers]
- Docker works to maintain that desired state
  - For instance, if a worker node becomes unavailable, Docker schedules that node's tasks on other nodes
- A *task* is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container
- When Docker is running in swarm mode, you can still run standalone containers on any of the Docker hosts participating in the swarm, as well as swarm services
- A key difference between standalone containers and swarm services is that only swarm managers can manage a swarm, while standalone containers can be started on any daemon

Features

- Cluster management integrated with Docker Engine
- Decentralized design
- Declarative service model
- Scaling
- Desired state reconciliation
- Multi-host networking
- Service discovery
- Load balancing
- Secure by default
- Rolling updates

Nodes

- A **node** is an instance of the Docker engine participating in the swarm
- You can run one or more nodes on a single physical computer or cloud server
- To deploy your application to a swarm, you submit a service definition to a **manager node**
- **Manager Node**
  - The manager node dispatches units of work called tasks to worker nodes
  - Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm
  - Manager nodes elect a single leader to conduct orchestration tasks
- **Worker nodes**
  - Worker nodes receive and execute tasks dispatched from manager nodes
  - An agent runs on each worker node and reports on the tasks assigned to it
  - The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker

Services and tasks

- **Service**
  - A service is the definition of the tasks to execute on the manager or worker nodes
  - It is the central structure of the swarm system and the primary root of user interaction with the swarm
  - When you create a service, you specify which container image to use and which commands to execute inside running containers
- **Task**
  - A task carries a Docker container and the commands to run inside the container
  - It is the atomic scheduling unit of swarm
  - Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale
  - Once a task is assigned to a node, it cannot move to another node
  - It can only run on the assigned node or fail

Swarm Setup

- **Create swarm**

  > docker swarm init --advertise-addr <MANAGER-IP>

- **Get current status of swarm**

  > docker info

- **Get the list of nodes**

  > docker node ls

Swarm Setup

- **Get token (on manager node)**
  > docker swarm join-token worker

- **Add node (on worker node)**
  > docker swarm join --token <token>

Swarm Service

- **Deploy a service**
  > docker service create --replicas <no> --name <name> -p <ports> <image> <command>

- **Get running services**
  > docker service ls

- **Inspect service**
  > docker service inspect <service>

- **Get the nodes running service**
  > docker service ps <service>

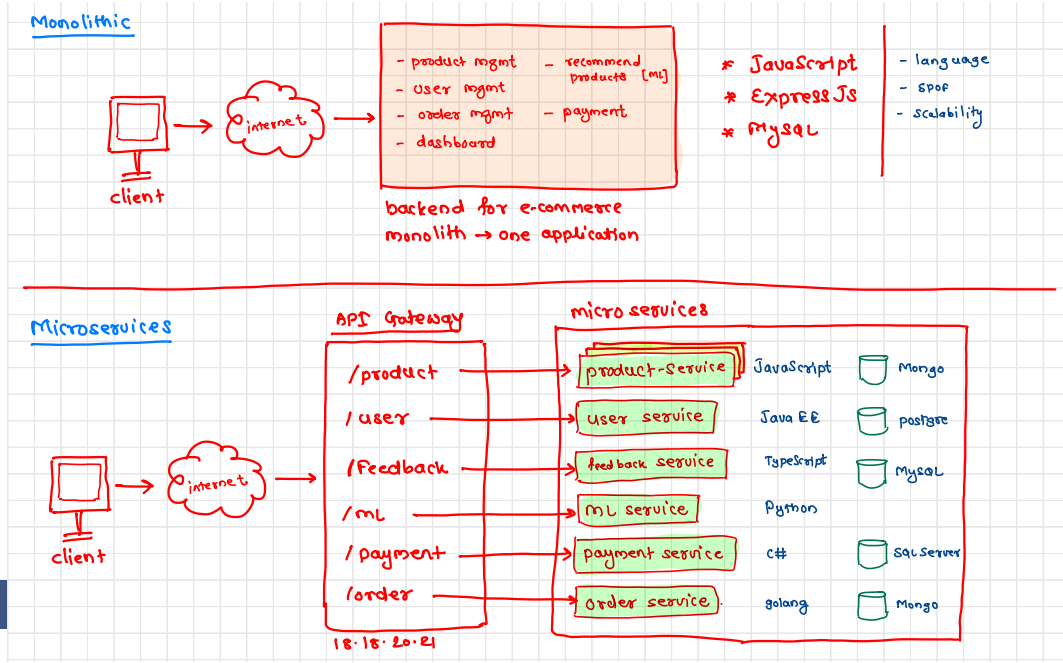
Swarm Service

- **Scale service**
  > docker service scale <service>=<scale>

- **Update service**
  > docker service update --image <image> <service>

- **Delete service**
  > docker service rm <service>

Microservices



Overview

- Distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations
- Is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities

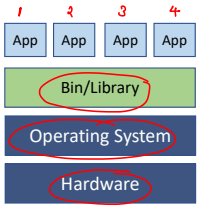
Benefits

- Easier to Build and Maintain Apps
- Improved Productivity and Speed
- Code for different services can be written in different languages
- Services can be deployed and then redeployed independently without compromising the integrity of an application
- Better fault isolation; if one microservice fails, the others will continue to work
- Easy integration and automatic deployment; using tools like Jenkins
- The microservice architecture enables continuous delivery.
- Easy to understand since they represent a small piece of functionality, and easy to modify for developers thus they can help a new team member become productive quickly
- Scalability and reusability, as well as efficiency
- Components can be spread across multiple servers or even multiple data centers
- Work very well with containers, such as Docker

Containerization

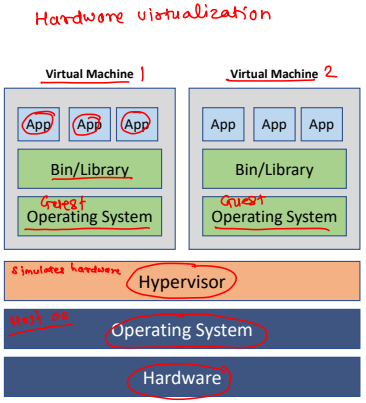
Traditional Deployment

- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform
- A solution for this would be to run each application on a different physical server
- But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers



Virtualized Deployment

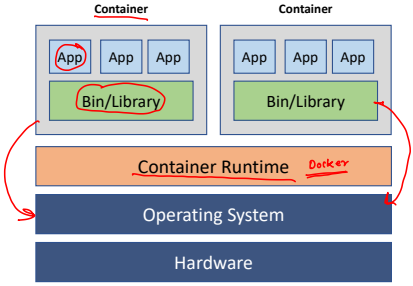
- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application
- Virtualization allows better utilization of resources in a physical server and allows better scalability because
  - an application can be added or updated easily
  - reduces hardware costs
- With virtualization you can present a set of physical resources as a cluster of disposable virtual machines
- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware

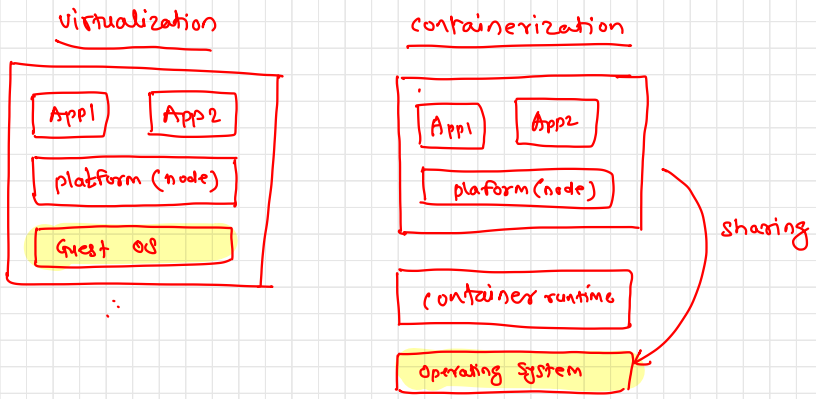


Container deployment

container - lightweight VM

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications
- Therefore, containers are considered lightweight
- Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions





Containerization

- Lightweight alternative or companion to a virtual machine
- Involves encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure
- Allows developers to create and deploy applications faster and more securely

Containers vs Virtual machines

Virtual Machine	Container
Hardware level virtualization	OS virtualization
Heavyweight (bigger in size)	Lightweight (smaller in size)
Slow provisioning	Real-time and fast provisioning
Limited Performance	Native performance
Fully isolated	Process-level isolation
More secure	Less secure
Each VM has separate OS	Each container can share OS resources
Boots in minutes	Boots in seconds
Pre-configured VMs are difficult to find and manage	Pre-built containers are readily available
Can be easily moved to new OS	Containers are destroyed and recreated
Creating VM takes longer time	Containers can be created in seconds

Advantages

- **Portability**
  - A container creates an executable package of software that is abstracted away from (not tied to or dependent upon) the host operating system, and hence, is portable and able to run uniformly and consistently across any platform or cloud
- **Agility**
  - The open source Docker Engine for running containers started the industry standard for containers with simple developer tools and a universal packaging approach that works on all operating systems
- **Speed**
  - Containers are often referred to as “lightweight,”
  - Meaning they share the machine’s operating system (OS) kernel and are not bogged down with this extra overhead
- **Fault isolation**
  - Each containerized application is isolated and operates independently of others
  - The failure of one container does not affect the continued operation of any other containers

Advantages

- **Efficiency**
  - Software running in containerized environments shares the machine's OS kernel, and application layers within a container can be shared across containers
  - Thus, containers are inherently smaller in capacity than a VM and require less start-up time
- **Ease of management**
  - A container orchestration platform automates the installation, scaling, and management of containerized workloads and services
  - Container orchestration platforms can ease management tasks such as scaling containerized apps, rolling out new versions of apps, and providing monitoring, logging and debugging, among other functions
- **Security**
  - The isolation of applications as containers inherently prevents the invasion of malicious code from affecting other containers or the host system
  - Additionally, security permissions can be defined to automatically block unwanted components from entering containers or limit communications with unnecessary resources

Popular container platforms

- Linux Containers (LXC)
- Docker
- Windows Server
- CoreOS rkt

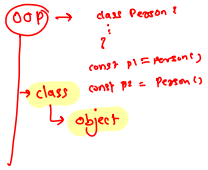
Docker

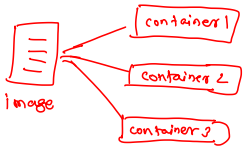
Overview

- Docker is software to create, manage and orchestrate containers
- It supports all major Operating Systems
- Docker Inc, started by Solomon Hykes, is behind the docker tool
- Docker Inc started as PasS provider called as dotCloud which was using the Linux Containers behind the screen to run contianers
- In 2013, the dotCloud became Docker Inc
- It comes in two editions
  - Enterprise Edition (EE)
  - Community Edition (CE)

Images

- Object that contains an OS filesystem and an application
- You can think of it as a class in Object Oriented Programming language
- Docker provides various pre-built images on Docker Hub





Commands related to images

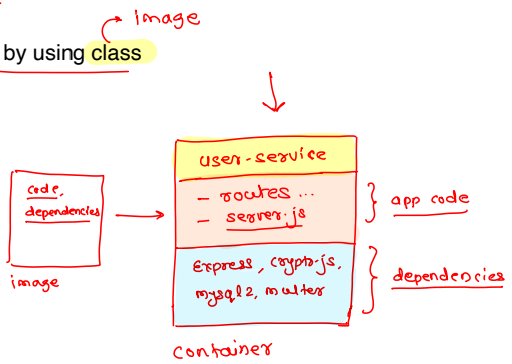
- **List all the images**
  > docker image ls
- **Download an image from docker hub**
  > docker image pull <image name>
- **Get the details of selected image**
  > docker image inspect <image name>
- **Delete an image**
  > docker image rm <image name>

Commands related to images

- **Push the image to docker hub**
  > docker image push <image name>
- **Tag an image**
  > docker image tag <image name> <tag>
- **Build an image with Dockerfile**
  > docker image build <Dockerfile>

Containers

- It is created using docker image
- You can think of container as an object created by using class
- It consists of
  - Your application code
  - Dependencies
  - Networking
  - Volumes



Commands related to containers

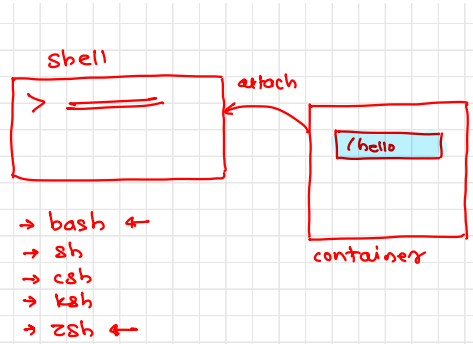
- **List the running containers**
  > docker container ls
- **List all the containers (including stopped)**
  > docker container ls –a
- **Create and start container**
  > docker container run <image>
- **Start a stopped/created container**
  > docker container start <container>

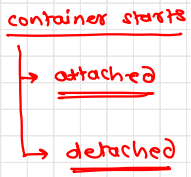
Commands related to containers

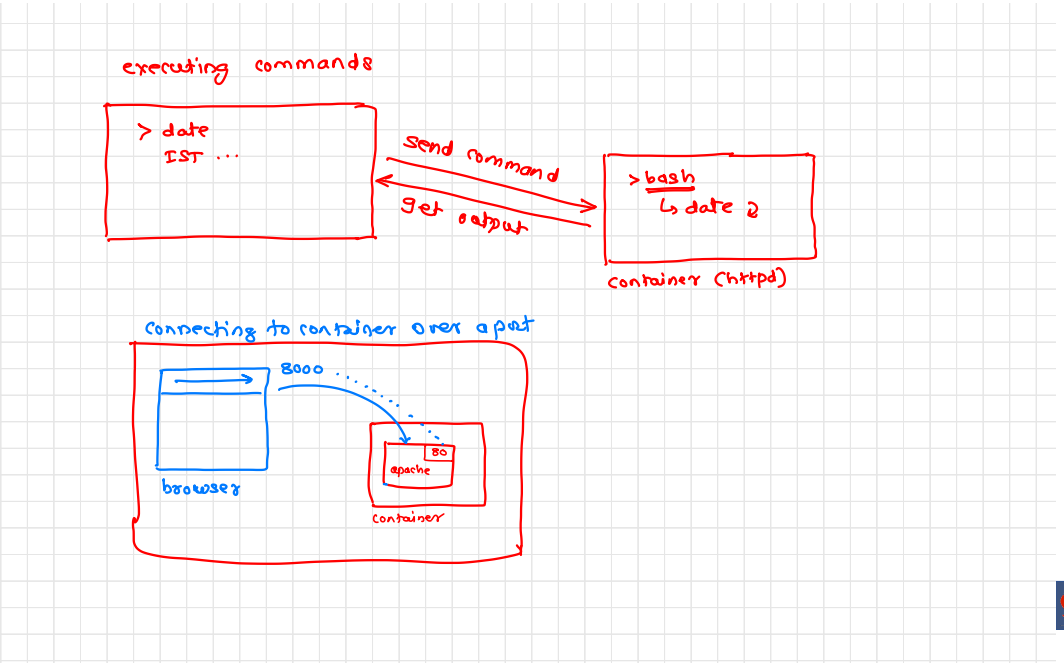
- **Stop a container**
  > docker container stop <container>
- **Remove a container**
  > docker container rm <container>
- **Get the stats of selected container**
  > docker container stats <container>
- **Execute a command in a container**
  > docker container exec <container>

Commands related to containers

- **Create an image from current state of a container**
  > docker container commit <container>
- **Get the processes running in the container**
  > docker container top <container>







Docker Compose

Microservices

- Distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations
- Is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities

Docker Compose

- Compose is a tool for defining and running multi-container Docker applications
- With Compose, you use a YAML file to configure your application's services
- Then, with a single command, you create and start all the services from your configuration

Sunbeam Infotech www.sunbeaminfo.com

Features

- Manages multiple services easily
- Multiple isolated environments on a single host
- Only recreate containers that have changed
- Variables and moving a composition between environments

Installation

- Run this command to download the current stable release of Docker Compose

  > sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

- Apply executable permissions to the binary:

  > sudo chmod +x /usr/local/bin/docker-compose

Start using docker compose

- Docker compose uses docker-compose.yml
- Following is sample docker-compose file

```
version: '3'
services:
  web:
    build: .
    ports:
      - "9090: 80"
```


Build and run the application

- To run the application use

  > docker-compose up

- To stop the containers

  > docker-compose stop

- To remove the containers

  > docker-compose down

YAML

Overview

- YAML is the abbreviated form of “YAML Ain’t markup language”
- It is a data serialization language which is designed to be human -friendly and works well with other programming languages for everyday tasks
- It is useful to manage data and includes Unicode printable characters

Features

- Matches native data structures of agile methodology and its languages such as Perl, Python, PHP, Ruby and JavaScript
- YAML data is portable between programming languages
- Includes data consistent data model
- Easily readable by humans
- Supports one-direction processing
- Ease of implementation and usage

Basics

- YAML is case sensitive
- The files should have **.yaml** or **.yml** as the extension
- YAML does not allow the use of tabs while creating YAML files; spaces are allowed instead
- Comment starts with #
- Comments must be separated from other tokens by whitespaces.

Scalars

- Scalars in YAML are written in block format using a literal type
- E.g.
  - Integer
    - 20
    - 40
  - String
    - Steve
    - “Jobs”
    - ‘USA’
  - Float
    - 4.5
    - 1.23015e+3

Mapping

- Represents key-value pair
- The value can be identified by using unique key
- Key and value are separated by using colon (:)
- E.g.
  - name: person1
  - address: “India”
  - phone: +9145434345
  - age: 40
  - hobbies:
  - - reading
  - - playing

Sequence

- Represents list of values
- Must be written on separate lines using dash and space
- Please note that space after dash is mandatory
- E.g.
  - # pet animals
  - - cat
  - - dog

  - # programming languages
  - - C
  - - C++
  - - Java

Sequence

- Sequence may contains complex objects
- E.g.
  - products:
    - - title: product 1
    - price: 100
    - description: good product
    - - title: product 2
    - price: 300
    - description: useful product