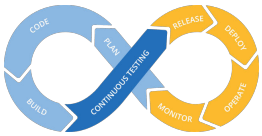


devops

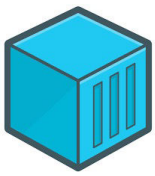
Git

SE
Agile

Kubernetes

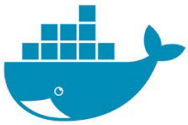
Advanced Software Development Methodologies

container



Selenium

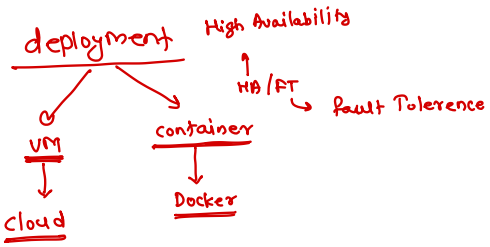
Jenkins



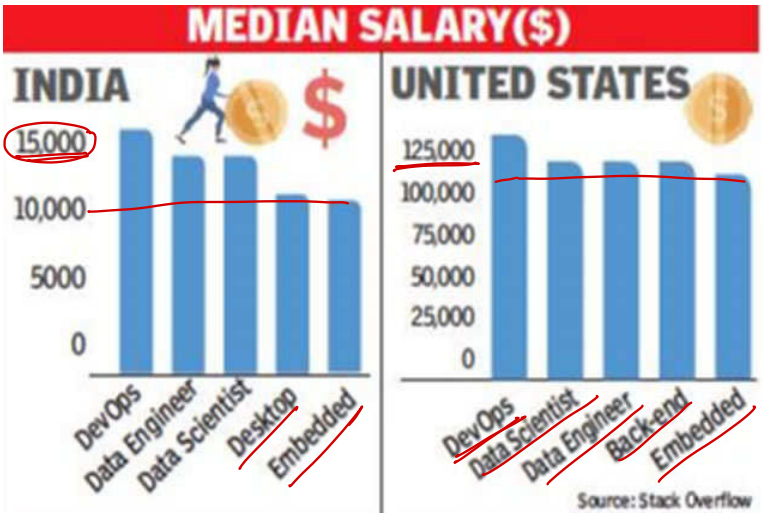
docker

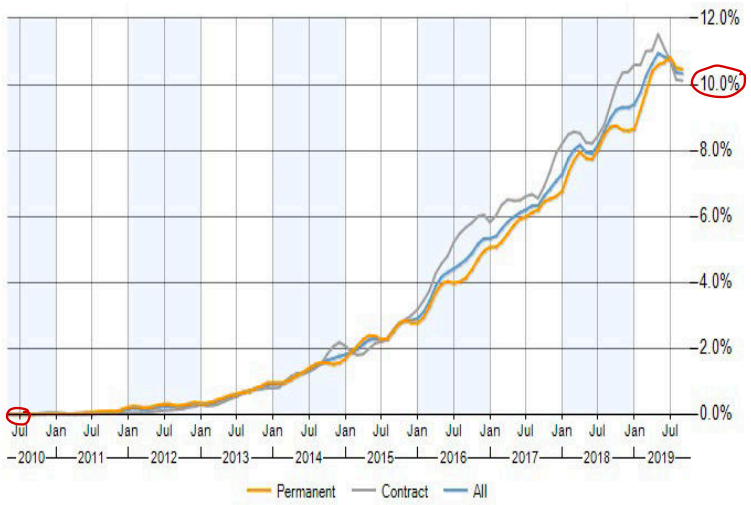
Contents

- Software Engineering
- SDLC VS Agile
- Application Testing
- Cloud Computing : AWS
- DevOps
- Source Control Management : Git → GitHub / GitLab
- Continuous Integration : Jenkins [CI/CD pipeline]
- Containerization : Docker
- Container Orchestration : Docker Swarm, Kubernetes
  - ↳ manage [create / restart / stop] containers



Why should you bother about DevOps?





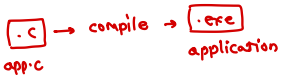
Pre-requisites

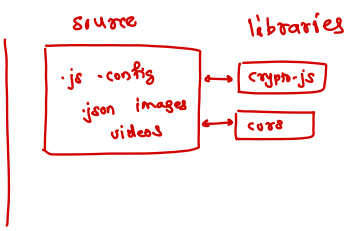
- Basic Linux knowledge
- Any Development Platforms → Node Js, Java
- Any language (preferred JS/Java)

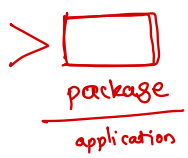
Software Engineering [SE]

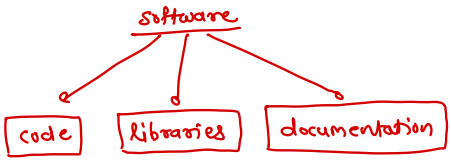
Introduction

- Software
  - Software is more than just a program code
  - A program is an executable code, which serves some computational purpose
  - Software is considered to be collection of executable programming code, associated libraries and documentations
  - Software, when made for a specific requirement is called software product
- Engineering
  - All about developing products using well defined principles, methods and procedures









What is Software Engineering?

- Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures
- The application of a systematic, disciplined , quantifiable approach to the development, operation and maintenance of software
- Establishment and use of sound engineering principles in order to obtain software that is reliable and work efficiently on real machines
- The process of developing a software product using software engineering principles and methods

Software Types

- System Software : operating System, Device Drivers
- Application Software : Calculator, Calendar , Email client
- Engineering/Scientific Software : Catia, Idea, AutoCAD,
- Embedded Software : IoT applications
- AI Software : Intrusion Detection System, Person detection in camera.
- Legacy Software : old application
- Web/Mobile Software : Websites , mobile apps
  e-commerce

Why SE is important

- Helps to build complex systems in timely manner
- Ensures high quality of software & testing
- Imposes discipline to work
- Minimizes software cost
- Decreases time

Software Evolution

- **S-type (static-type)** (fixed) — check if a number is prime
  - Works strictly according to the pre-defined specifications and solutions
  - Solution and method to achieve it can be understood immediately before coding starts
  - Least subjected to the changes
  - E.g. Calculator program for mathematical computation
- **P-type (practical-type)** → Dynamic
  - Software with a collection of different procedures
  - Is defined by exactly what procedures can do
  - The specification can be described and solutions are not obvious instantly
  - E.g. Gaming software

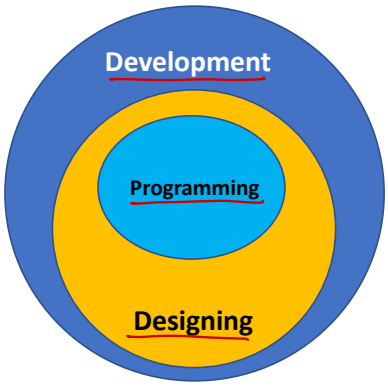
- **E-type (embedded-type)** → e-commerce → purchase → taxes → GST (9% + 9%.) / service tax
  - Works closely as the requirement of real-world environment
  - Has a high degree of evolution as there are various changes in laws, taxes etc. in the real world
  - E.g. online trading software

E-Type software evolution laws

- **Continuing change**
  - An E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful
- **Increasing complexity**
  - As software evolves, its complexity tends to increase unless work is done to maintain or reduce it
- **Conservation of familiarity** → keep the documentation
  - The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc. must be retained at any cost, to implement the changes in the system
- **Continuing growth**
  - In order for an E-type system intended to resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of the business
- **Reducing quality**
  - An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment
- **Feedback systems**
  - The E-type software systems constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.
- **Self-regulation**
  - E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
- **Organizational stability**
  - The average effective global activity rate in an evolving E-type system is invariant over the lifetime of the product.

Software Paradigms

- Refer to the methods and steps, which are taken while designing the software
- There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand
- These can be combined into various categories, though each of them is contained in one another
- Consists of
  - Requirement gathering
  - Software design
  - Programming



Characteristics of good software

- A software product can be judged by what it offers and how well it can be used
- Well-engineered and crafted software is expected to have the following characteristics
  1. Operational
  2. Transactional
  3. Maintenance

Operational

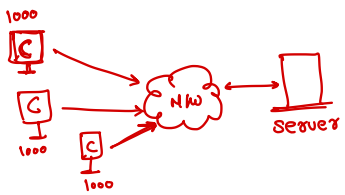
- This tells us how well software works in operations
- Can be measured on:
  - Budget
  - Usability
  - Efficiently
  - Correctness
  - Functionality
  - Dependability
  - Security : confidentiality [encryption]
  - Safety : parental control

Transitional

- This aspect is important when the software is moved from one platform to another
- Can be measured on
  - Portability
  - Interoperability
  - Reusability
  - Adaptability

Maintenance

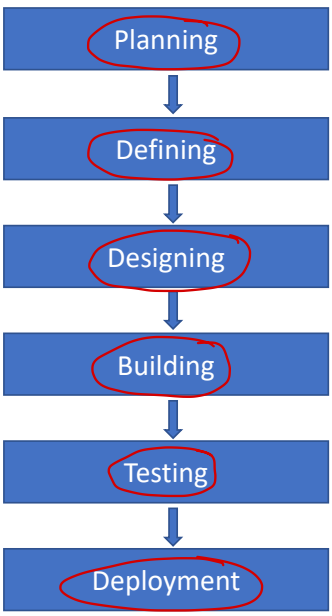
- Briefs about how well a software has the capabilities to maintain itself in the ever-changing environment
- Can be measured on
  - Modularity
  - Maintainability
  - Flexibility
  - Scalability



SDLC

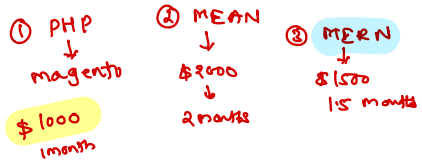
Overview

- Also called as Software Development Process
- Is a well-defined, structured sequence of stages in software engineering to develop the intended software product
- Is a framework defining tasks performed at each step in the software development process
- Aims to produce a high-quality software that
  - meets or exceeds customer expectations
  - reaches completion within times and cost estimates
- Consists of detailed plan (stages) of describing how to develop, test, deploy and maintain the software product



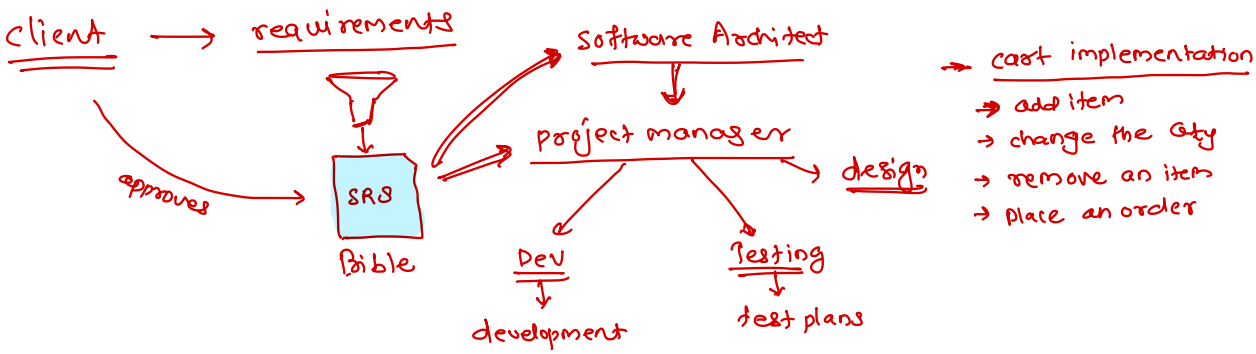
Planning and Requirement Analysis

- The most important and fundamental stage in SDLC
- It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry
- This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas
- Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage
- The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks



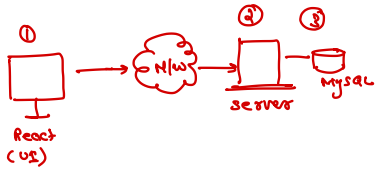
Defining Requirements

- Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts
- This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle



Designing the Product Architecture

- SRS is the reference for product architects to come out with the best architecture for the product to be developed
- Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification
- This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product
- A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any)
- The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS



Stack
- React - JS
- DB - MySQL + mongo
- Server - express
- deployment - cloud
- testing - Jest
- platform - NodeJs

Building or Developing the Product

- In this stage of SDLC the actual development starts and the product is built
- The programming code is generated as per DDS during this stage
- If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle
- Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code
- Different high level programming languages such as C, C++, Java, PHP etc. are used for coding
- The programming language is chosen with respect to the type of software being developed

* Website → MERN / MEAN → JS / TS
  PHP
  .net → C#

* System apps → C / C++

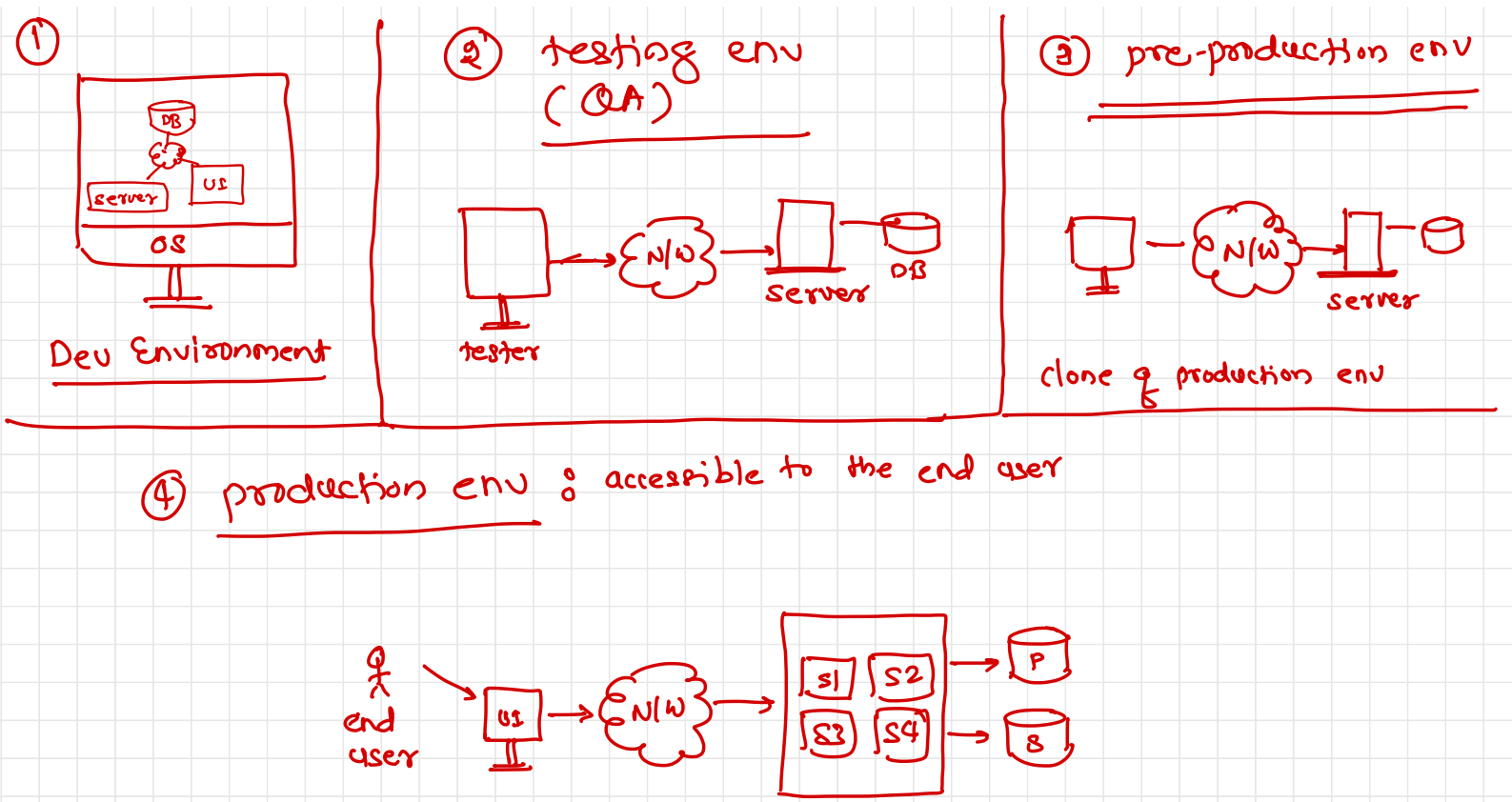
* Mobile apps
  - android - Java / Kotline
  → ios → Objective C / Swift
  } JS (React Native)

Testing the Product

- This stage is usually a subset of all the stages as in the modern SDLC models
- The testing activities are mostly involved in all the stages of SDLC
- However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS

Deployment in the Market and Maintenance

- Once the product is tested and ready to be deployed it is released formally in the appropriate market
- Sometimes product deployment happens in stages as per the business strategy of that organization
- The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing)
- Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment
- After the product is released in the market, its maintenance is done for the existing customer base

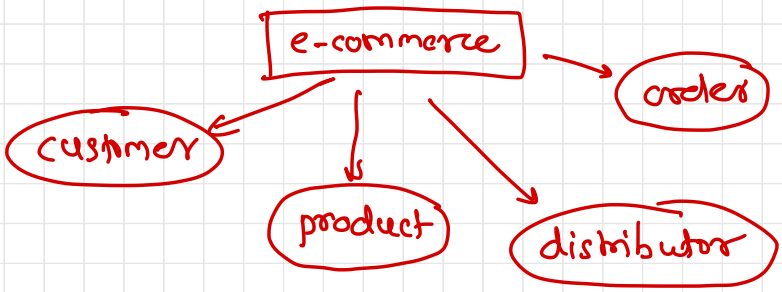


SDLC Models

- There are various software development life cycle models defined and designed which are followed during the software development process
- Also referred as Software Development Process Models
- Models
  - Waterfall Model
  - Iterative Model
  - Spiral Model
  - V-Model
  - Big Bang Model
  - Agile Model

e-commerce

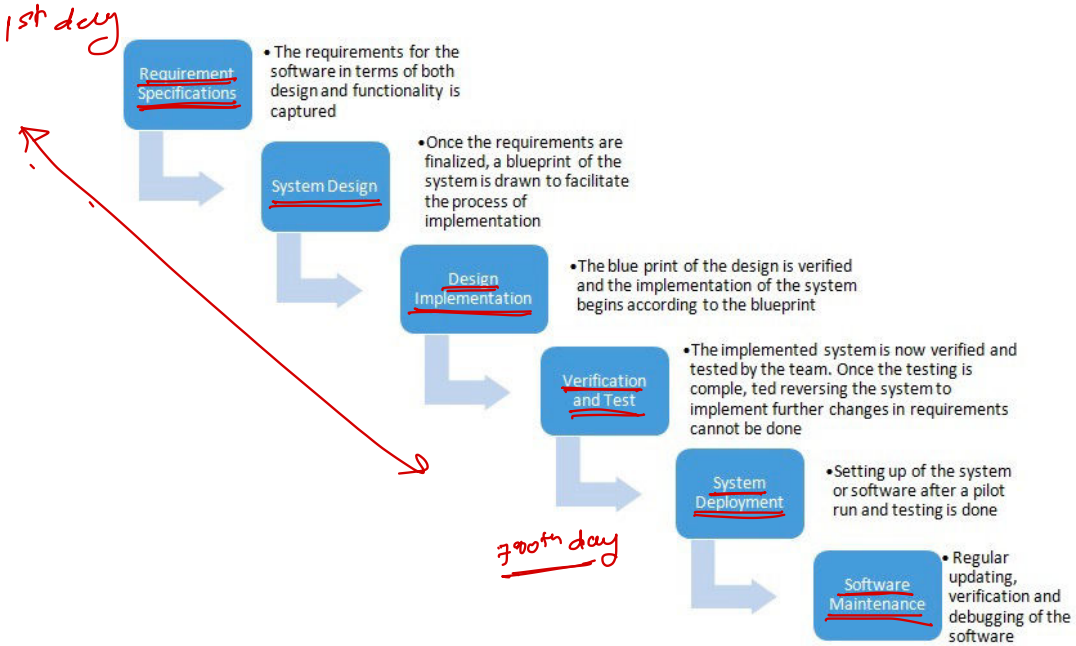
- product → add / update / delete / get / search ...
- order → place / get / history / search / cancel ...
- distributor → signup / signin / add products ...
- customer → signup / signin / changed password ...



Waterfall Model

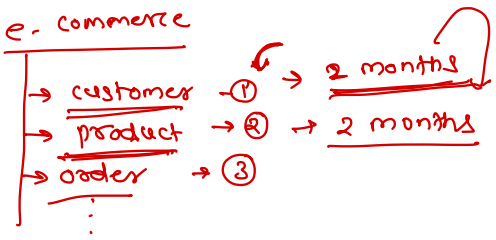
- **Requirement Specification**
- **Design**
- **Implementation**
- **Verification and Testing**
- **Deployment**
- **Maintenance**

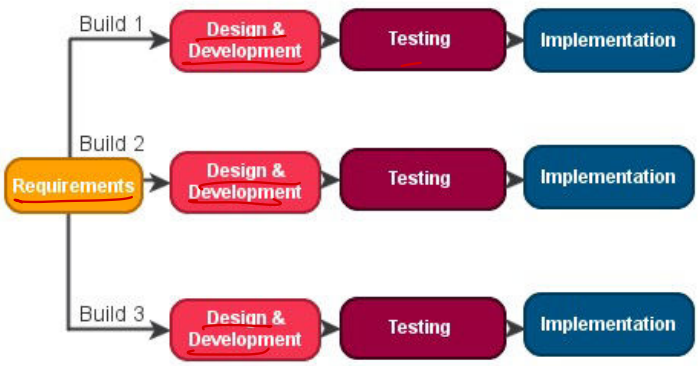
estimations
time : 2 years
budget : $1.5m
:



Iterative Model

- implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented

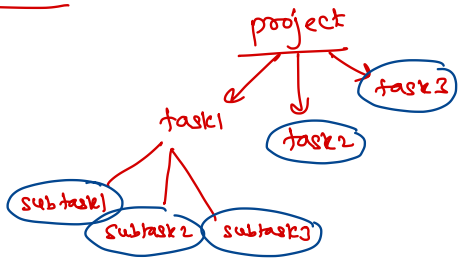




Agile Methodologies

Agile Methodologies

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements
- The tasks are divided to time boxes (small time frames) to deliver specific features for a release
- Iterative approach is taken and working software build is delivered after each iteration
- Each build is incremental in terms of features; the final build holds all the features required by the customer



Agile Manifesto

- **Individuals and interactions**
  - self-organization and motivation are important
- **Working software**
  - Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentations
- **Customer collaboration**
  - continuous customer interaction is very important to get proper product requirements
- **Responding to change**
  - focused on quick responses to change and continuous development

Agile Methodologies

- The most popular Agile methods include
  - Rational Unified Process
  - Scrum
  - Crystal Clear
  - Extreme Programming
  - Adaptive Software Development
  - Feature Driven Development
  - Dynamic Systems Development Method (DSDM)

e-commerce

- product
  1. add a product
  2. Edit a product
  3. delete a product
  4. Search product
  5. filter products
  6. get all products

- customer
  7. register
  8. sign in
  9. change password
  10. forgot password

- cart
  11. add an item
  12. remove item
  13. change quantities
  14. place an order

Sprint → 2 weeks : collection of tasks
80 hrs

sprint 1 → 2 weeks

1, 2, 3

→ 1 ✓
→ 2 ✓
→ 3 ✓

4, 5, 6

sprint 2 – 2 weeks

– 4 ✓
– 5 ✓
– 6 ✓

What is Scrum ?

task based approach

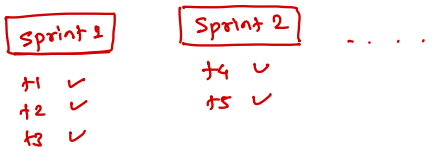
- Scrum isn't a process, it's a framework that facilitates processes amongst other things
- Is an agile way to manage a project
- Management framework with far reaching abilities to control and manage the iterations and increments in all project types
- One of the implementations of agile methodology
- Incremental builds are delivered to the customer in every two to three weeks time
- Ideally used in the project where the requirement is rapidly changing
- The framework is made up of a Scrum team, individual roles, events, artefacts, and rules

Scrum Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business people and developers must work together daily throughout the project
- Build projects around motivated individuals
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

Scrum Principles

- Working software is the primary measure of progress
- Agile processes promote sustainable development
- Continuous attention to technical excellence and good design enhances agility
- Simplicity, the art of maximizing the amount of work not done, is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly



Scrum Pillars

Transparency

All parts of the process should be transparent, open, and honest for everyone to see

Inspection

Everything that is worked on during a sprint can be inspected by the team to make sure that it is achieving what it needs to.

Adaptation

If a member of the Scrum team or a stakeholder notices that things aren't going according to plan, the team will need to change up what they're doing to fix this as quickly as possible

Scrum Values

Courage	Courage to do the right thing and work on tough problems
Focus	Focus on the sprint and its goal
Commitment	Commitment to the team and sprint goal
Respect	Respect, for each other by helping people to learn the things that you're good at, and not judging the things that others aren't good at
Openness	Be open and honest and let people know what you're struggling with challenges and problems that are stopping you from achieving success

How scrum pillars help us?

Self-organization

- This results in healthier shared ownership among the team members
- It is also an innovative and creative environment which is conducive to growth

Collaboration

- Essential principle which focuses collaborative work

Time-boxing

- Defines how time is a limiting constraint in Scrum method
- Daily Sprint planning and Review Meetings

Sprint

Iterative Development

- Emphasizes how to manage changes better and build products which satisfy customer needs
- Defines the organization's responsibilities regarding iterative development

Scrum Roles

Product Owner

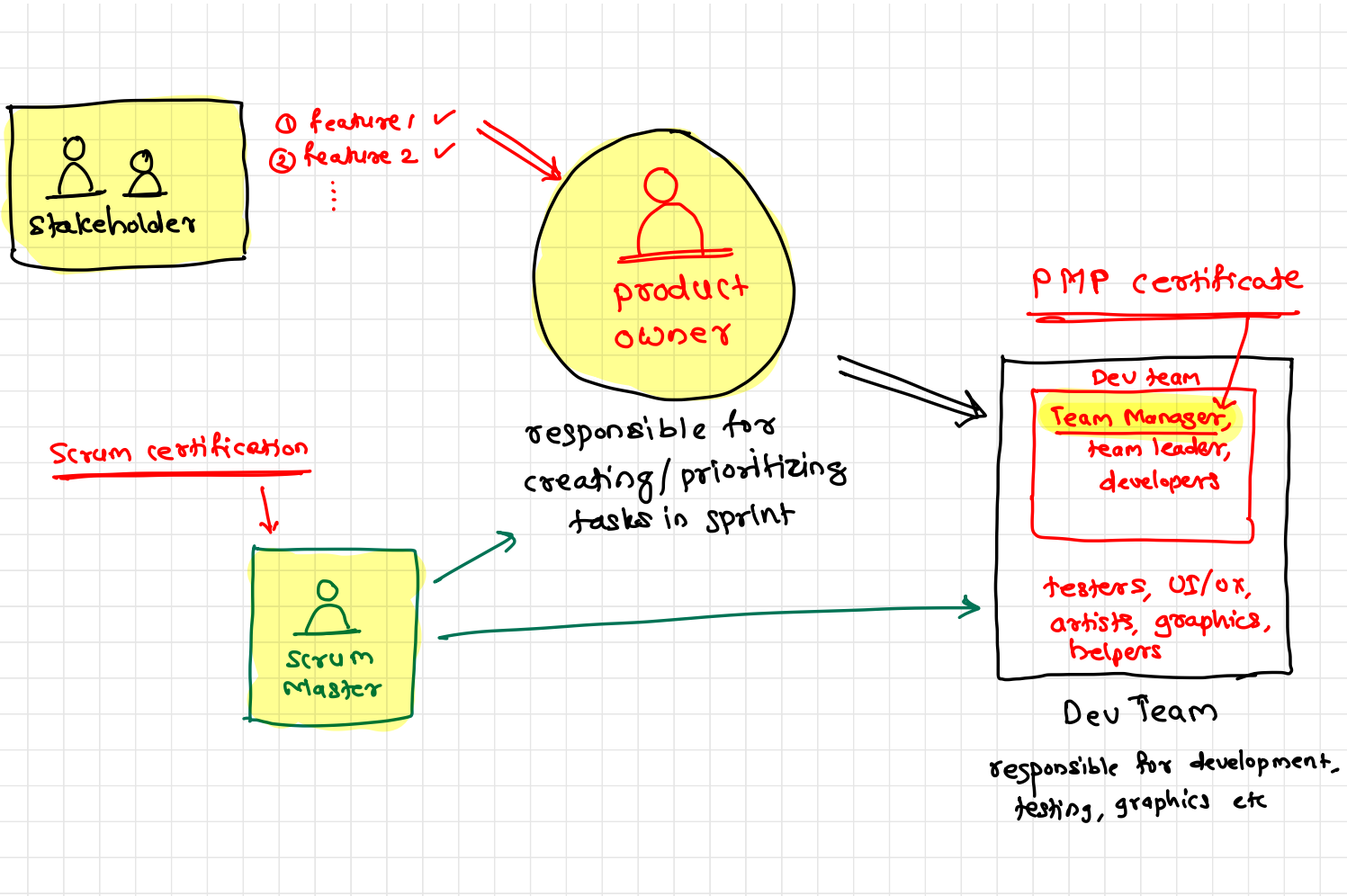
- Job to understand and engage with the stakeholders to understand what needs to be done and create that backlog
- Also need to prioritize that backlog

Scrum Master

- Helps the entire team achieve the scrum goals and work within scrum
- Support the product owner with their responsibilities in terms of managing the backlog as well as, supporting the development team

Dev Team

- The people who are creating the product or service and delivering done increments at the end of each sprint
- Includes developers, tester, writers, graphics artists and others



Scrum Artefacts

Product Backlog

collection of all tasks

Sprint Backlog

collection of selected tasks for the sprint

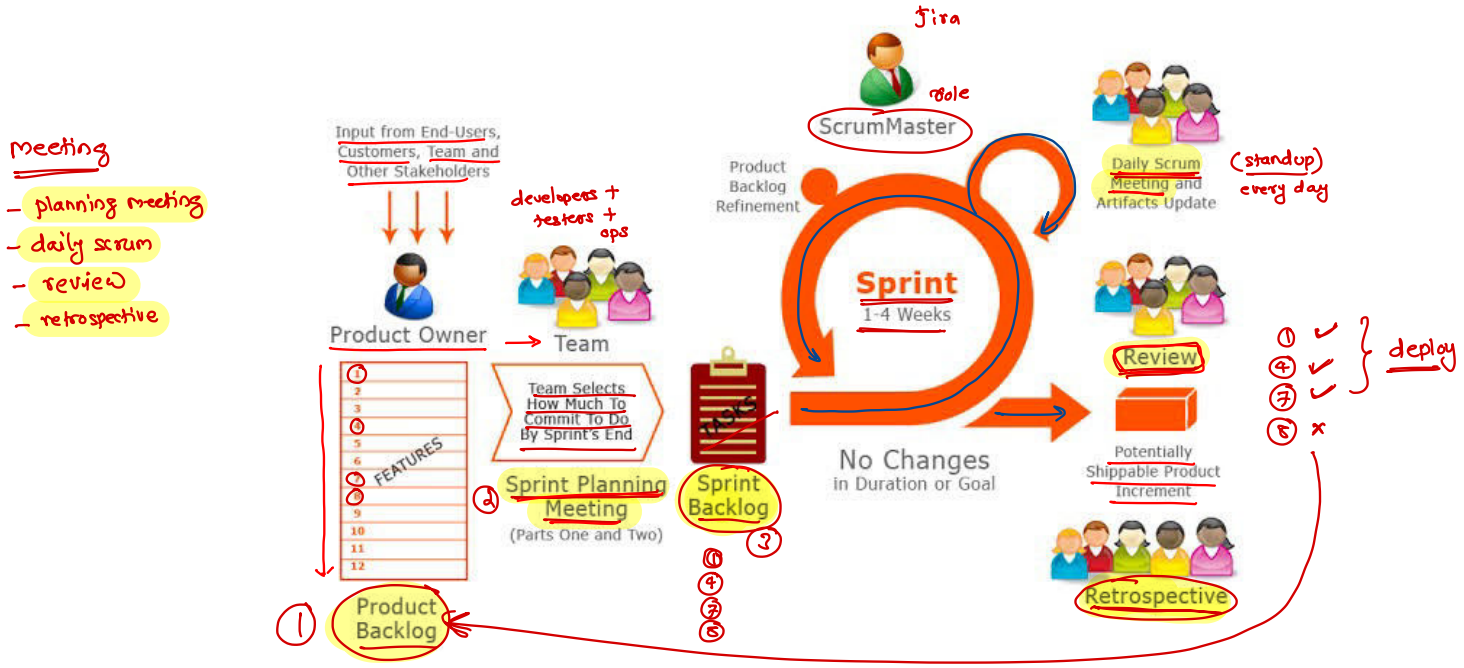
Increment

enhancement by adding features

Scrum Events

Sprint Planning	• Happens at the start of every sprint • it should probably be about between four and eight hours
Daily Scrum	• This is a very short time-boxed event • Usually only lasting no more than 15 minutes
Sprint Review	• Collaborative events to demo what has been achieved and to help keep everyone who's involved working together
Sprint Retrospective	• About continuous process and improvement and we need to take what we've learned into the next sprint planning session
Sprint	• It is really the beating heart of scrum and all the scrum events take place in it

Scrum Process

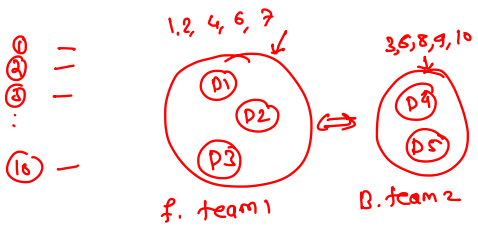


Agile vs traditional models

No	Agile Methodologies	Traditional Methodologies
1	Incremental value and risk management	Phased approach with an attempt to know everything at the start
2	Embracing change	Change prevention
3	Deliver early, fail early	Deliver at the end, fail at the end
4	Transparency	Detailed planning, stagnant control
5	Inspect and adapt	Meta solutions, tightly controlled procedures and final answers
6	Self managed	Command and control
7	Continual learning	Learning is secondary to the pressure of delivery

Agile - Advantages

- Very realistic approach to software development
- Promotes teamwork and cross training
- Functionality can be developed rapidly and demonstrated
- Resource requirements are minimum
- Suitable for fixed or changing requirements
- Delivers early partial working solutions
- Good model for environments that change steadily
- Minimal rules, documentation easily employed
- Enables concurrent development and delivery within an overall planned context
- Little or no planning required
- Easy to manage
- Gives flexibility to developers



Agile - Disadvantages

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Scrum tools

- Jira – https://www.atlassian.com/software/jira/
- Clarizen – https://www.clarizen.com/
- GitScrum – https://site.gitscrum.com/
- Vivify Scrum – https://www.vivifyscrum.com/
- Yodiz – https://www.yodiz.com/
- ScrumDo – https://www.scrumdo.com/
- Quickscrum – https://www.quickscrum.com/
- Manuscript – https://www.manuscript.com/
- Scrumwise – https://www.scrumwise.com/
- Axosoft – https://www.axosoft.com/

Agile Methodologies – Scrum Terminologies

- **Scrum:** a framework to support teams in complex product development
- **Scrum Board:** a physical board to visualize information for and by the Scrum Team, used to manage Sprint Backlog
- **Scrum Master:** the role within a Scrum Team accountable for guiding, coaching, teaching and assisting a Scrum Team and its environments in a proper understanding and use of Scrum
- **Scrum Team:** a self-organizing team consisting of a Product Owner, Development Team and Scrum Master
- **Self-organization:** the management principle that teams autonomously organize their work

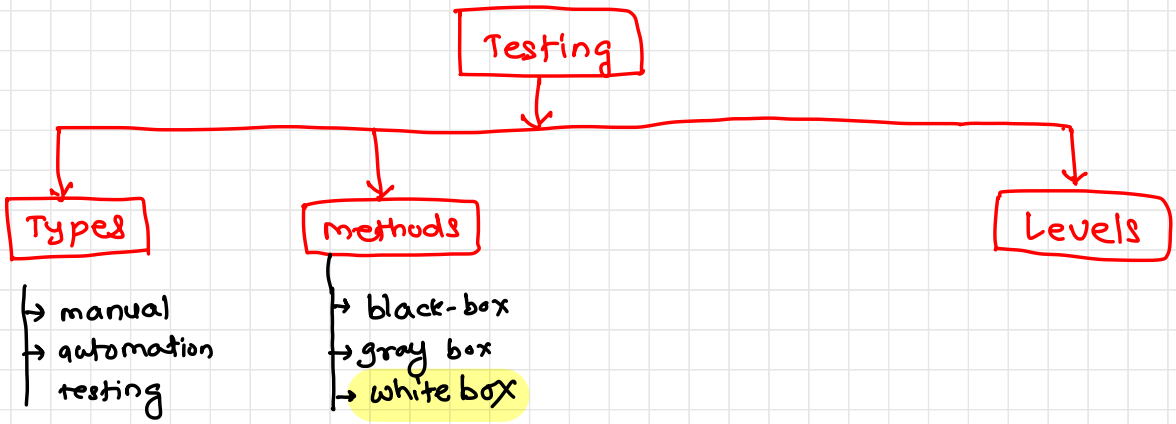
Agile Methodologies – Scrum Terminologies

- **Sprint:** time-boxed event of 30 days, or less, that serves as a container for the other Scrum events and activities. collection of tasks
- **Sprint Backlog:** an overview of the development work to realize a Sprint's goal, typically a forecast of functionality and the work needed to deliver that functionality.
- **Sprint Goal:** a short expression of the purpose of a Sprint, often a business problem that is addressed
- **Sprint Retrospective:** time-boxed event of 3 hours, or less, to end a Sprint to inspect the past Sprint and plan for improvements

Agile Methodologies – Scrum Terminologies

- **Sprint Review:** time-boxed event of 4 hours, or less, to conclude the development work of a Sprint
- **Stakeholder:** a person external to the Scrum Team with a specific interest in and knowledge of a product that is required for incremental discovery
- **Development Team:** the role within a Scrum Team accountable for managing, organizing and doing all development work
- **Daily Scrum:** daily time-boxed event of 15 minutes for the Development Team to re-plan the next day of development work during a Sprint

Testing



What is testing?

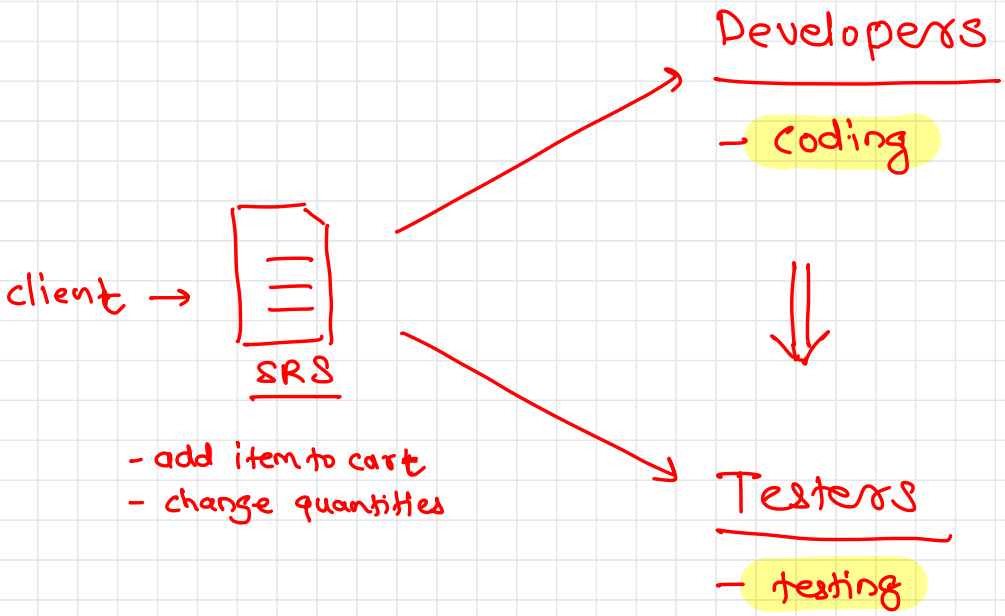
- process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not
- executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements
- A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item
- Testing will be done by
  - Software Developer : unit testing
  - Software Tester : testing
  - Project Lead/Manager : verification
  - End User : user acceptance test (UAT)

When to Start Testing?

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing
- Testing performed by a developer on completion of the code is also categorized as testing

When to Stop Testing?

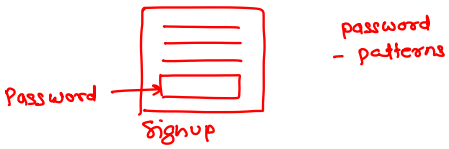
- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision



Verification vs Validation

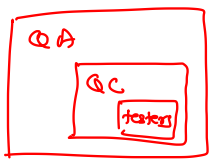
- Addresses the concern: "Are you building it right?"
- Ensures that the software system meets all the functionality
- Takes place first and includes the checking for documentation, code
- Done by developers
- It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software
- It is an objective process and no subjective decision should be needed to verify a software

- Addresses the concern: "Are you building the right thing?"
- Ensures that the functionalities meet the intended behaviour
- Validation occurs after verification and mainly involves the checking of the overall product
- Done by testers
- It has dynamic activities, as it includes executing the software against the requirements
- It is a subjective process and involves subjective decisions on how well a software works



Quality Assurance vs Quality Control

- QA includes activities that ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements
- Focuses on processes and procedures rather than conducting actual testing on the system
- Process-oriented activities
- Preventive activities
- It is a subset of Software Test Life Cycle (STLC)



- QC includes activities that ensure the verification of a developed software with respect to documented requirements
- Focuses on actual testing by executing the software with an aim to identify bug/defect through implementation of procedures and process
- Product-oriented activities
- It is a corrective process
- QC can be considered as the subset of Quality Assurance

Checking the testing process

- **Audit** – internal
  - It is a systematic process to determine how the actual testing process is conducted within an organization or a team
  - It is an independent examination of processes involved during the testing of a software
  - It is a review of documented processes that organizations implement and follow
  - Types: Legal Compliance Audit, Internal Audit, and System Audit
- **Inspection** : – external
  - Inspection is a formal evaluation technique in which software requirements, designs, or codes are examined in detail
  - Conducted by a person or a group other than the author to detect faults, violations of development standards, and other problems

Types

Manual Testing

- Manual testing includes testing a software manually, i.e., without using any automated tool or any script
- The tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug
- There are different stages for manual testing
  - unit testing
  - integration testing
  - system testing
  - user acceptance testing (UAT)
- Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing

Automation Testing

- also known as Test Automation
- Tester writes scripts and uses another software to test the product
- Involves automation of a manual process
- Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly
- Used to test
  - Regression
  - Performance
  - Stress point
- It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing

What to Automate?

- It is not possible to automate everything in a software
- The areas at which a user can make transactions such as
  - the login form or registration forms
  - any area where large number of users can access the software simultaneously
  - all GUI items
  - connections with databases
  - field validations

When to Automate?

- Large and critical projects
- Projects that require testing the same areas frequently
- Requirements not changing frequently
- Accessing the application for load and performance with many virtual users
- Stable software with respect to manual testing
- Availability of time

How to Automate?

- Identifying areas within a software for automation
- Selection of appropriate tool for test automation : selenium / Jest / Jasmin / pyunit / nunit
- Writing test scripts
- Development of test suits (collection of test cases)
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues

Software Testing Tools

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR

Methods

Black-Box Testing

- The technique of testing without having any knowledge of the interior workings of the application
- The tester is oblivious to the system architecture and does not have access to the source code
- Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon
- Advantages
  - Well suited and efficient for large code segments
  - Code access is not required
  - Clearly separates user's perspective from the developer's perspective through visibly defined roles
  - Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems
- Disadvantages
  - Limited coverage, since only a selected number of test scenarios is actually performed
  - Inefficient testing, due to the fact that the tester only has limited knowledge about an application
  - Blind coverage, since the tester cannot target specific code segments or errorprone areas

White-Box Testing

- Detailed investigation of internal logic and structure of the code
- Is also called **glass testing** or **open-box testing**
- A tester needs to know the internal workings of the code
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately
- Advantages
  - As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively
  - It helps in optimizing the code
  - Extra lines of code can be removed which can bring in hidden defects
  - Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing
- Disadvantages
  - Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
  - Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested
  - It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools

Grey-Box Testing

- A technique to test the application with having a limited knowledge of the internal workings of an application
- Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database
- Advantages
  - Offers combined benefits of black-box and white-box testing wherever possible
  - Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications
  - Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling
  - The test is done from the point of view of the user and not the designer
- Disadvantages
  - Since the access to source code is not available, the ability to go over the code and test coverage is limited
  - The tests can be redundant if the software designer has already run a test case
  - Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested

Black vs Grey vs White

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings need not be known	Requires limited knowledge of the internal workings	Requires full knowledge of the internal workings
Also known as closed-box testing, data-driven testing, or functional testing	Also known as translucent testing	Also known as clear-box testing, structural testing, or code-based testing
Performed by end-users and also by testers and developers	Performed by end-users and also by testers and developers	Normally done by testers and developers
Testing is based on external expectations	Testing is done on the basis of high-level database diagrams and DFDs	Tester can design test data accordingly
It is exhaustive and the least time-consuming	Partly time-consuming and exhaustive	The most exhaustive and time-consuming type of testing
Not suited for algorithm testing	Not suited for algorithm testing	Suited for algorithm testing
Only be done by trial-and-error method	Data domains and internal boundaries can be tested	Data domains and internal boundaries can be better tested

Levels

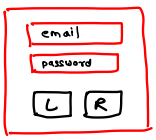
functional

non-functional

Functional Testing

SRS

- This is a type of black-box testing that is based on the specifications of the software that is to be tested
- The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for
- Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements
- Includes
  - Unit Testing
  - Integration Testing
  - System Testing
  - Regression Testing
  - Acceptance Testing
  - Alpha Testing
  - Beta Testing



complete system
all modules

Functional Testing – Steps

- The determination of the functionality that the intended application is meant to perform
- The creation of test data based on the specifications of the application
- The output based on the test data and the specifications of the application
- The writing of test scenarios and the execution of test cases
- The comparison of actual and expected results based on the executed test cases

test data

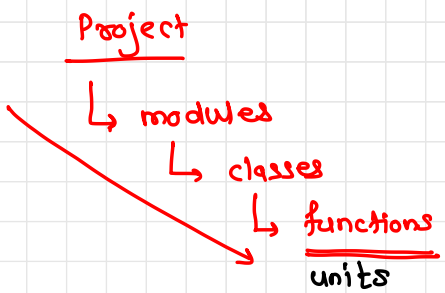
email
- happy path → +ve → working
- -ve testing → not working

test data	Expected	actual
amit	x	x
amit@	x	x
amit@test	x	✓
amit@test.com	✓	✓

password
- patterns

```
function add(p1, p2) {
    return p1 + p2
}
// 5, 5   ≠ 25  ⇒ -ve
// 10, 20 ⇒ 30  ⇒ +ve
const result = add(10,20)
if(result == 30) {cond} {
else { error }
```

```
function xyz()
{
    :
    add('20','20')
    ↳ 40x
      2020
```

Unit Testing : [Developer]

- Is performed by developers before the setup is handed over to the testing team to formally execute the test cases
- Unit testing is performed by the respective developers on the individual units of source code assigned areas
- The developers use test data that is different from the test data of the quality assurance team
- The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality
  - → functions
- Limitation
  - Testing cannot catch each and every bug in an application
  - It is impossible to evaluate every execution path in every software application
  - There is a limit to the number of scenarios and test data that a developer can use to verify a source code
  - After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units

unit testing

java → JUnit

javascript → Jest, Jasmin

c#(.Net) → NUnit

python → PyUnit

e-commerce

- product module
  - → add product ←
  - → delete ←
  - → update ←
  - → search ←
- order module
  - → place order → getting product
  - → delete
  - → get

Integration Testing

- Integration testing is defined as the testing of combined parts of an application to determine if they function correctly
- Integration testing can be done in two ways
  - Bottom-up integration testing
    - This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds
  - Top-down integration testing
    - In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter
- bottom-up testing is usually done first, followed by top-down testing

System Testing

- System testing tests the system as a whole
- Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards
- This type of testing is performed by a specialized testing team
- Importance
  - is the first step in the SDLC, where the application is tested as a whole
  - The application is tested thoroughly to verify that it meets the functional and technical specifications
  - The application is tested in an environment that is very close to the production environment where the application will be deployed
  - System testing enables us to test, verify, and validate both the business requirements as well as the application architecture

Regression Testing

- The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application
- Importance
  - Minimizes the gaps in testing when an application with changes made has to be tested
  - Testing the new changes to verify that the changes made did not affect any other area of the application
  - Test coverage is increased without compromising timelines
  - Increase speed to market the product

Acceptance Testing

- The most important type of testing, as it is conducted by the Quality Assurance Team who will verify whether the application meets the intended specifications and satisfies the client's requirement
- The QA team will have a set of pre-written scenarios and test cases that will be used to test the application
- Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application
- By performing acceptance tests on an application, the testing team will reduce how the application will perform in production.
- There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

- This test is the first stage of testing and will be performed amongst the teams (developer and QA teams)
- Unit testing, integration testing and system testing when combined together is known as alpha testing
- During this phase, the following aspects will be tested in the application
  - Spelling Mistakes
  - Broken Links
  - The Application will be tested on machines with the lowest specification to test loading times and any latency problems

Beta Testing

- This test is performed after alpha testing has been successfully performed
- In beta testing, a sample of the intended audience tests the application
- Beta testing is also known as **pre-release testing**
- Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release
- Users will install, run the application and send their feedback to the project team
- Typographical errors, confusing application flow, and even crashes
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users
- The more issues you fix that solve real user problems, the higher the quality of your application will be
- Having a higher-quality application when you release it to the general public will increase customer satisfaction

Non-Functional Testing

- Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc
- Includes
  - Performance testing
  - Usability Testing
  - Security Testing
  - portability testing

Performance Testing

- It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software
- There are different causes that contribute in lowering the performance of a software
  - Network delay
  - Client-side processing
  - Database transaction processing
  - Load balancing between servers
  - Data rendering : server sided rendering → Next.js → react
- Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects –
  - Speed (i.e. Response Time, data rendering and accessing)
  - Capacity : simultaneous access
  - Stability
  - Scalability
- Performance testing can be either qualitative or quantitative and can be divided into
  - Load testing
  - Stress testing

Load Testing

- It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data
- It can be done at both normal and peak load conditions
- This type of testing identifies the maximum capacity of software and its behavior at peak time
- Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader etc

Stress Testing

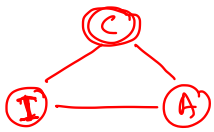
- Stress testing includes testing the behavior of a software under abnormal conditions
- For example, it may include taking away some resources or applying a load beyond the actual load limit
- This testing can be performed by testing different scenarios such as
  - Shutdown or restart of network ports randomly
  - Turning the database on or off
  - Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

- Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation
- Can be defined as
  - efficiency of use
  - learn-ability
  - memory-ability
  - errors/safety
  - Satisfaction
- UI testing can be considered as a sub-part of usability testing.

Security Testing

- Testing a software in order to identify any flaws and gaps from security and vulnerability point of view.
- Involves
  - Confidentiality : confidential data is secure [encryption]
  - Integrity : Authenticated user can access the data
  - Authentication : validating user by comparing username/email & password
  - Availability : data is available for intended user at any condition [RAID, backup]
  - Authorization : authenticated user has enough permission to access a feature
  - Non-repudiation : tokens
  - Input checking and validation : SQL injection x
  - SQL insertion attacks



Portability Testing

- Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well
- Following are the strategies that can be used for portability testing
  - Transferring an installed software from one computer to another.
  - Building executable to run the software on different platforms
- Portability testing can be considered as one of the sub-parts of system testing
- Computer hardware, operating systems, and browsers are the major focus of portability testing

Selenium

Overview

- Selenium is an open-source and a portable automated software testing tool for testing web application
- It has capabilities to operate across different browsers and operating systems
- Selenium is not just a single tool but a set of tools that helps testers to automate web-based applications more efficiently
- Tools
  - Selenium IDE
  - Selenium RC (Remote Control)
  - Selenium WebDriver
  - Selenium Grid

languages
- JS
- Java
- python
- C#
- C++

Advantages

- Selenium is an open-source tool
- Can be extended for various technologies that expose DOM
- Has capabilities to execute scripts across different browsers
- Can execute scripts on various operating systems
- Supports mobile devices
- Executes tests within the browser, so focus is NOT required while script execution is in progress
- Can execute tests in parallel with the use of Selenium Grids

Disadvantages

- Supports only web based applications
- No feature such as Object Repository/Recovery Scenario
- Cannot access controls within the browser
- No default test report generation
- For parameterization, users has to rely on the programming language

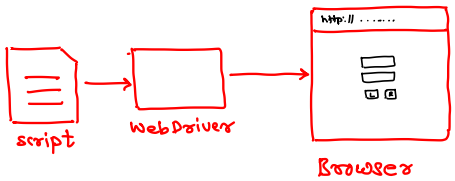
Locators

- Element Locators help Selenium to identify the HTML element the command refers to
- Types
  - **identifier =** id Select the element with the specified "id" attribute and if there is no match, select the first element whose @name attribute is id.
  - **id =** id Select the element with the specified "id" attribute.
  - **name =** name Select the first element with the specified "name" attribute
  - **dom =** javascriptExpression Selenium finds an element by evaluating the specified string that allows us to traverse through the HTML Document Object Model using JavaScript. Users cannot return a value but can evaluate as an expression in the block
  - **xpath =** xpathExpression Locate an element using an XPath expression.
  - **link =** textPattern Select the link element (within anchor tags) which contains text matching the specified pattern
  - **css =** cssSelectorSyntax Select the element using css selector

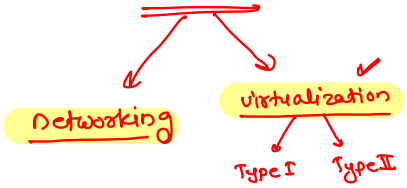
WebDriver

- WebDriver is a tool for automating testing web applications
- It is popularly known as Selenium 2.0
- Interacts directly with the browser and uses the browser's engine to control it
- It is faster, as it interacts directly with the browser
- It can support the headless execution

no browser GUI



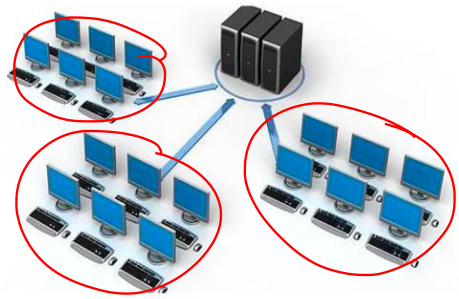
Cloud Computing



Networking

What is network ?

- It is the interconnection of multiple devices, generally termed as Hosts connected using multiple paths
- The purpose of network is: sending/receiving data or media
- It involves various devices like hubs, switches, routers etc.

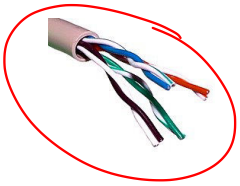


Wired network

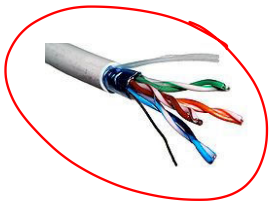
- The network build by connecting devices together using wires/cables as a medium to transfer the data
- Cables
  - Coaxial cable
  - Twisted pairs cables
  - Fiber optics

Co-axial

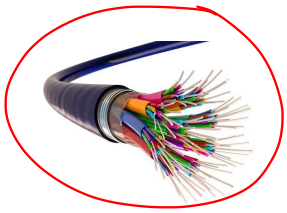
UTP



STP



fo



Wireless network

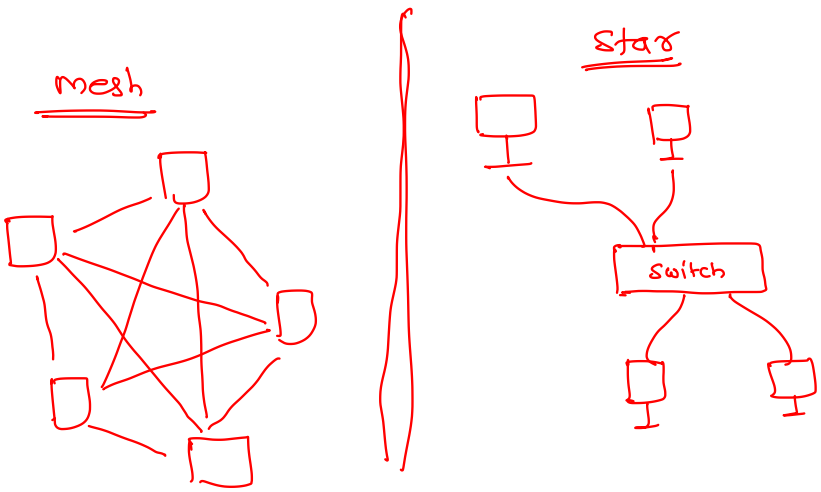
- The network build by connecting devices together using air as a medium to transfer the data
- EM Waves are used to transfer data from sender to receiver

Network Types

- Personal Area Network (PAN)
  - Smallest network which is very personal to the user
  - E.g. BlueTooth
- Local Area Network (LAN)
  - Spans across building(s) and operated under single administrative system
  - E.g. company, school network
  - Technologies: TokenRing or Ethernet
- Metropolitan Area Network (MAN)
  - Spans across cities
  - E.g. cable network
  - Technologies: high speed fiber optics
- Wide Area Network (WAN)
  - Spans across countries
  - Technologies: ATM, Frame Relay

What is a network topology ?

- Physical arrangement of computers is known as topology
- Famous topologies
  - Bus
  - Ring
  - Token Ring
  - Star → wired / wireless
  - Mesh — wireless



ISO OSI model

- Conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology
- Goal is the interoperability of diverse communication systems with standard communication protocols
- Layered architecture having 7 layers
  - Application
  - Presentation
  - Session
  - Transport
  - Network
  - Data Link
  - Physical

Application Layer

- Specifies interface methods used by hosts in a communications network
- Contains communication protocols
  - **HTTP [80]**: Hyper Text Transfer Protocol
  - **HTTPs [443]**: Secure Hyper Text Transfer Protocol
  - **FTP [20, 21]**: File Transfer Protocol
  - **SFTP [115]**: Simple FTP
  - **DNS [53]**: Domain Name Service
  - **NFS [1023]**: Network File System
  - **POP3 [110]**: Post Office Protocol
  - **SMTP [25]**: Simple Mail Transfer Protocol
  - **SSH [22]**: Secure Shell
  - **LDAP [389]**: Lightweight Directory Access Protocol

Presentation Layer

- Serves as the data translator for the network
- Also known as syntax layer
- Responsible for
  - Translation
  - Compression/Decompression
  - Encoding/Decoding
  - Encryption/Decryption

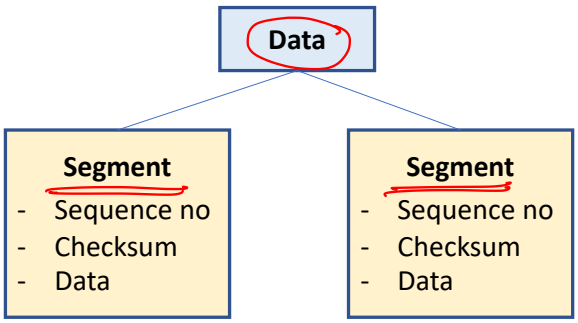
Session Layer

- Provides mechanism for opening, closing and managing session between processes
- Communication sessions consist of requests and responses that occur between applications
- Protocols
  - **ASP**: AppleTalk Session Protocol
  - **ADSP**: AppleTalk Data Stream Protocol
  - **NetBIOS**: Network BIOS
  - **PAP**: Password Authentication Protocol
  - **PPTP**: Point to Point Tunnelling Protocol
  - **RPC**: Remote Procedure Call
  - **SCP**: Session Control Protocol
  - **SDP**: Socket Direct Protocol

VPN

Transport Layer

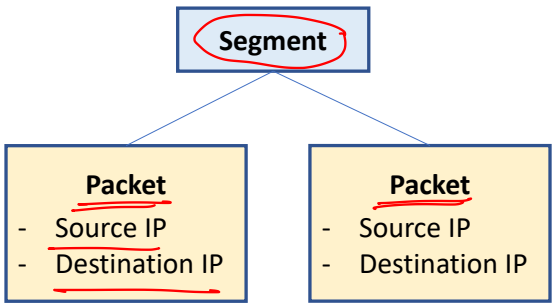
- Provide host-to-host communication services for applications
- Creates Segment (data unit) containing
  - Sequence number
  - Checksum
  - Port number
- Protocols
  - TCP
    - Connection oriented protocol
    - Provides: Flow Control, Error checking
    - Guarantees data delivery
    - Slower than UDP
    - E.g. WWW, HTTP
  - UDP
    - Connectionless protocol
    - Does not provide flow control
    - Does not guarantee data delivery
    - Faster than TCP
    - E.g. streaming, online games



Network Layer (Routing)

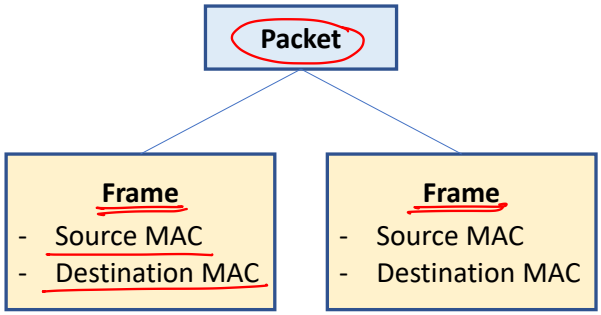
- Responsible for packet forwarding including routing through intermediate routers
- Responsible for splitting segment into packets containing
  - Source IP address
  - Destination IP address
- Protocols
  - **IP**: Internet Protocol
  - **IPX**: Internetwork Packet Exchange
  - **IPSec**: Internet Protocol Security
  - **EGP**: Exterior Gateway Protocol

Device : Router



Data Link Layer

- Transfers data between
  - adjacent network nodes in a wide area network (WAN) or
  - between nodes on the same local area network (LAN) segment
- Encapsulates packet into Frames containing
  - Source MAC Address
  - Destination MAC Address
- Sublayers
  - Logical Link Layer
  - Media Access Control Layer



Device : Switch

Data Link Layer: Logical Link Layer

- The uppermost sublayer multiplexes protocols running at the top of data link layer, and optionally provides flow control, acknowledgment, and error notification
- Provides addressing and control of the data link
- Services
  - Error control (automatic repeat request, ARQ)
  - Flow control [Data-link-layer flow control is not used in LAN protocols such as Ethernet, but in modems and wireless networks]

Data Link Layer: Media Access Control Layer

- Refers to the sublayer that determines who is allowed to access the media at any one time (CSMA/CD)
- Determines where one frame of data ends and the next one starts (frame synchronization)
- Frame synchronization uses: time based, character counting, byte stuffing and bit stuffing.
- Services
  - Multiple access protocols for channel-access control,
    - CSMA/CD protocols for collision detection and re-transmission in Ethernet networks
    - CSMA/CA protocol for collision avoidance in wireless networks
  - Physical addressing (MAC addressing)
  - LAN switching (packet switching), including MAC filtering, Spanning Tree Protocol (STP) and Shortest Path Bridging (SPB)
  - Data packet queuing or scheduling

Physical Layer

- Consists of the electronic circuit transmission technologies of a network
- Fundamental layer underlying the higher level functions in a network which provides means of transmitting raw bits rather than logical packets or segments
- The bitstream may be grouped into code words or symbols and converted to a physical signal that is transmitted over a transmission medium
- Translates logical communications requests from the data link layer into hardware-specific operations to cause transmission or reception of electronic signals
- Services
  - Modulation/Demodulation
  - Multiplexing
- Consists of
  - Cables/wires
  - Devices like hub, repeaters etc.

Addressing Modes: MAC Address

- Used to identify NIC uniquely
- Consists of 6 bytes [48 bits]
- First 3 bytes represents manufacturer
- Next 3 bytes represents NIC's unique address

windows: ipconfig /all

Linux/mac : ifconfig

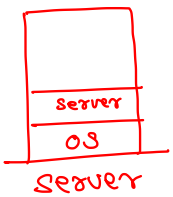
Addressing Modes: IP Address

- Used to identify every device uniquely
- Set by operating system running on the device
- Can be written in
  - Decimal: 192.168.100.10
  - Binary: 11000000.10101000.01100100.00001010
- Types
  - Private: used to communicate with other devices in local network
  - Public: used to communicate with other devices over internet
- Versions
  - IPv4
    - 32 bit [4 bytes] address
    - Classful and Classless addressing
  - IPv6
    - 128 bit address

Virtualization

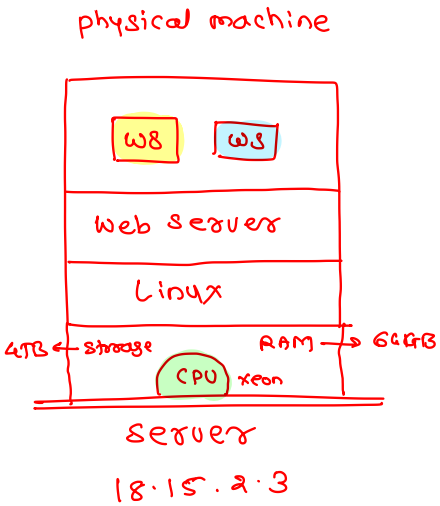
Software Deployment

- Software deployment defines a process of making the software available for the users
- E.g. a web site can be available for the end users, when it is hosted on a machine which can be accessed from anywhere in the world



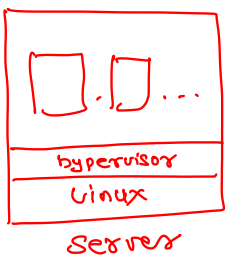
Traditional Deployment

- 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



What is virtualization

- Virtualization is the creation of a virtual -- rather than actual -- version of something, such as an operating system (OS), a server, a storage device or network resources
- Virtualization uses software that simulates hardware functionality in order to create a virtual system
- This practice allows IT organizations to operate multiple operating systems, more than one virtual system and various applications on a single server
- Types
  - Network virtualization
  - Storage virtualization
  - Data virtualization
  - Desktop virtualization
  - Application virtualization
  - Hardware virtualization



Network Virtualization

- Network virtualization takes the available resources on a network and breaks the bandwidth into discrete channels
- Admins can secure each channel separately, and they can assign and reassign channels to specific devices in real time
- The promise of network virtualization is to improve networks' speed, availability and security, and it's particularly useful for networks that must support unpredictable usage bursts

Storage Virtualization

- Storage virtualization is the pooling of physical storage from multiple network storage devices into what appears to be a single storage device that is managed from a central console
- Storage virtualization is commonly used in storage area networks
- Applications can use storage without having any concern for where it resides, what technical interface it provides, how it has been implemented, which platform it uses and how much of it is available
- Benefits
  - Makes the remote storage devices appear local
  - Multiple smaller volumes appear as a single large volume
  - Data is spread over multiple physical disks to improve reliability and performance
  - All operating systems use the same storage device
  - Provided high availability, disaster recovery, improved performance and sharing

Data virtualization

- Data virtualization is the process of aggregating data from different sources of information to develop a single, logical and virtual view of information so that it can be accessed by front-end solutions such as applications, dashboards and portals without having to know the data's exact storage location
- The process of data virtualization involves abstracting, transforming, federating and delivering data from disparate sources
- The main goal of data virtualization technology is to provide a single point of access to the data by aggregating it from a wide range of data sources
- Benefits
  - Abstraction of technical aspects of stored data like APIs, Language, Location, Storage structure
  - Provides an ability to connect multiple data sources from a single location
  - Provides an ability to combine the data result sets across multiple sources (also known as data federation)
  - Provides an ability to deliver the data as requested by users

Desktop virtualization

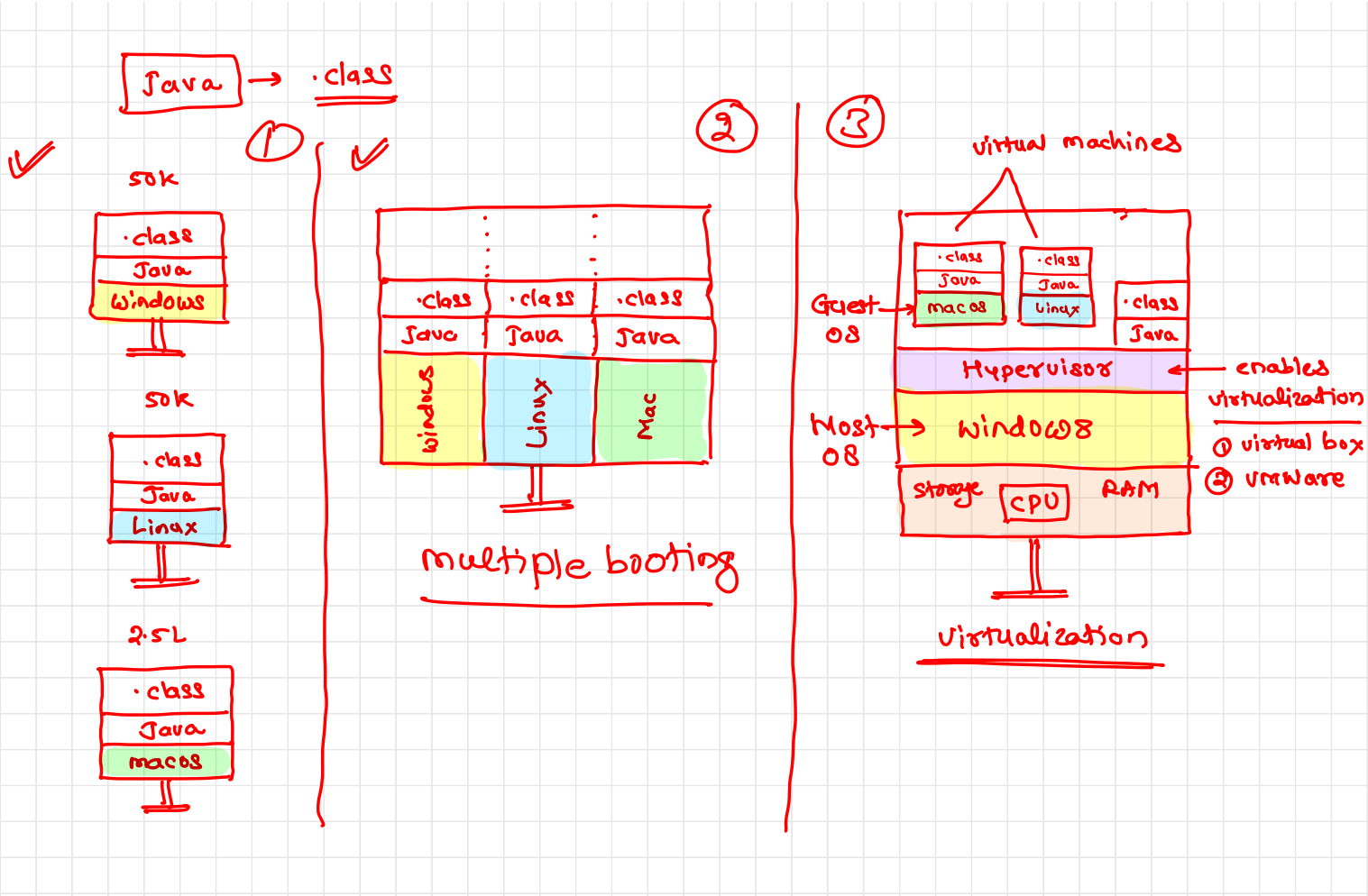
- With desktop virtualization, the goal is to isolate a desktop OS from the endpoint that employees use to access it
- It provides an ability to connect to the desktop from remote site
- When multiple users connect to a shared desktop, as is the case with Microsoft Remote Desktop Services, it's known as shared hosted desktop virtualization

Application Virtualization

- With application virtualization, an app runs separately from the device that accesses it
- Application virtualization makes it possible for IT admins to install, patch and update only one version of an app rather than performing the same management tasks multiple times

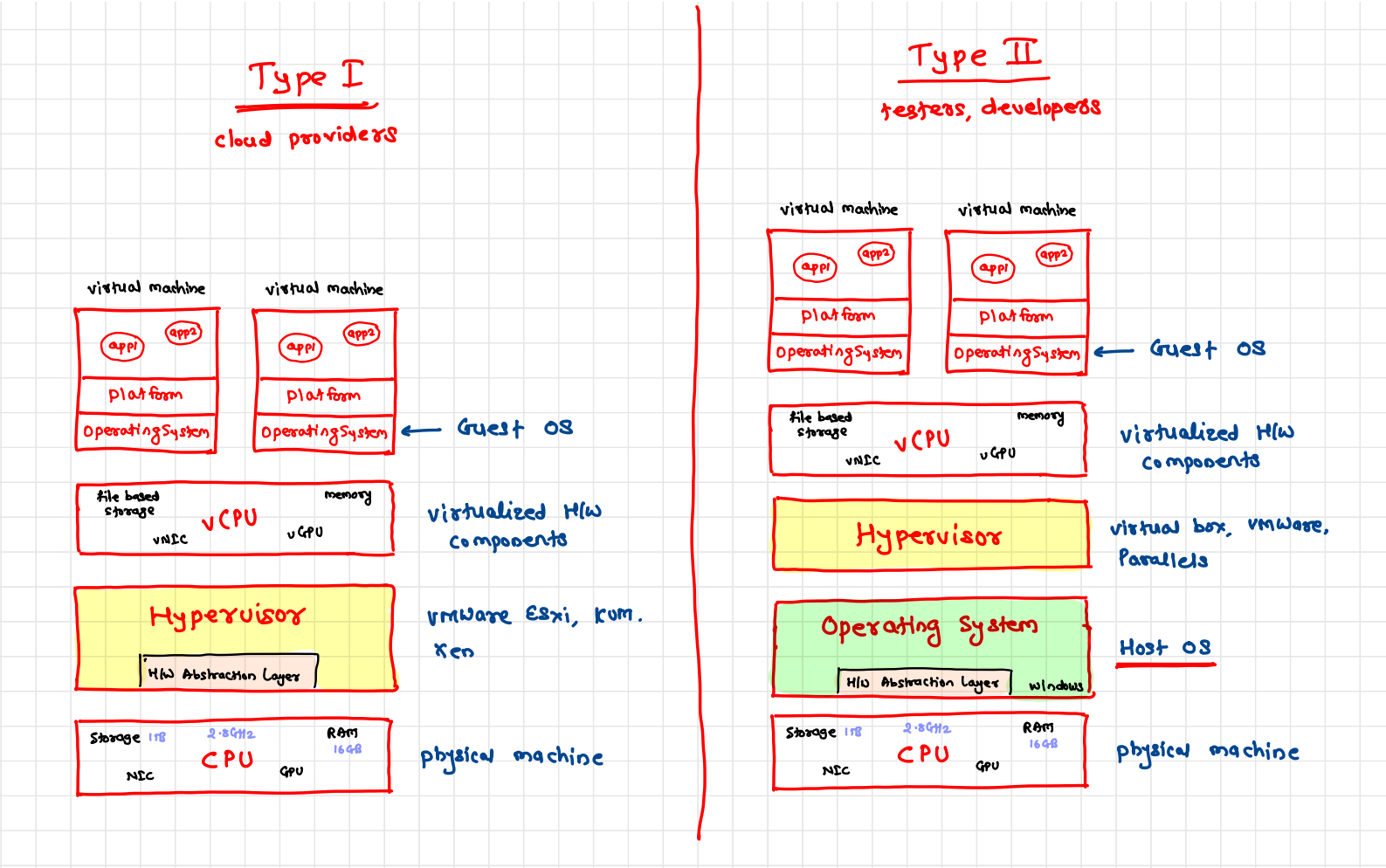
Hardware Virtualization

- Hardware virtualization or platform virtualization refers to the creation of a virtual machine that acts like a real computer with an operating system
- The process of masking the hardware resources like
  - CPU
  - Storage
  - Memory
- For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine
- The process of creating Machines



Virtual Machine

- A virtual machine is the emulated equivalent of a computer system that runs on top of another system
- Virtual machines may have access to any number of resources
  - Computing power - through hardware-assisted but limited access to the host machine's CPU
  - Memory - one or more physical or virtual disk devices for storage
  - A virtual or real network interfaces
  - Any devices such as
    - video cards,
    - USB devices,
    - other hardware that are shared with the virtual machine
- If the virtual machine is stored on a virtual disk, this is often referred to as a disk image

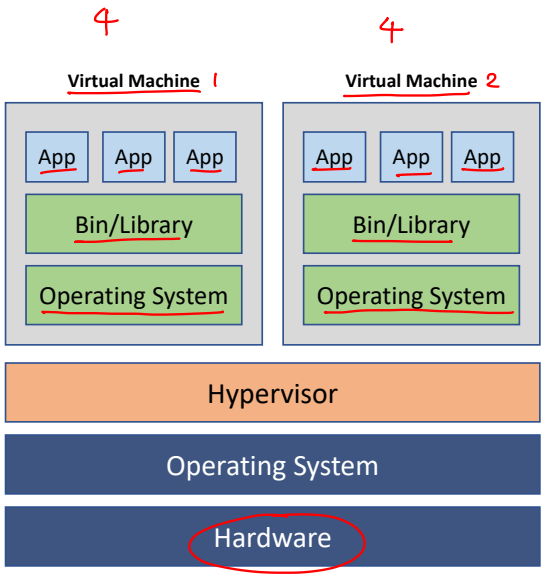


Types of hardware virtualization

- Type I
  - A Type 1 hypervisor runs directly on the host machine's physical hardware, and it's referred to as a bare-metal hypervisor
  - It doesn't have to load an underlying OS first
  - With direct access to the underlying hardware and no other software, it is more efficient and provides better performance
  - It is best suited for enterprise computing or data centers
  - E.g. VMware ESXi, Microsoft Hyper-V server and open source KVM
- Type II
  - A Type 2 hypervisor is typically installed on top of an existing OS, and it's called a hosted hypervisor
  - It relies on the host machine's pre-existing OS to manage calls to CPU, memory, storage and network resources
  - E.g. VMware Fusion, Oracle VM VirtualBox, Oracle VM Server for x86, Oracle Solaris Zones, Parallels and VMware Workstation

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



Advantages of virtualization

- **Lower costs**
  - Virtualization reduces the amount of hardware servers necessary within a company and data center
  - This lowers the overall cost of buying and maintaining large amounts of hardware
- **Easier disaster recovery**
  - Disaster recovery is very simple in a virtualized environment
  - Regular snapshots provide up-to-date data, allowing virtual machines to be feasibly backed up and recovered
  - Even in an emergency, a virtual machine can be migrated to a new location within minutes
- **Easier testing**
  - Testing is less complicated in a virtual environment
  - Even if a large mistake is made, the test does not need to stop and go back to the beginning
  - It can simply return to the previous snapshot and proceed with the test.

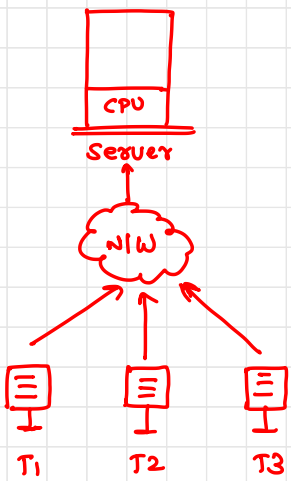
Advantages of virtualization

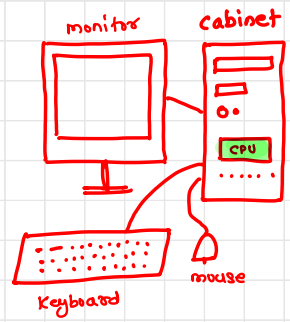
- **Quicker backups**
  - Backups can be taken of both the virtual server and the virtual machine
  - Automatic snapshots are taken throughout the day to guarantee that all data is up-to-date
  - Furthermore, the virtual machines can be easily migrated between each other and efficiently redeployed
- **Improved productivity**
  - Fewer physical resources results in less time spent managing and maintaining the servers
  - Tasks that can take days or weeks in a physical environment can be done in minutes
  - This allows staff members to spend the majority of their time on more productive tasks, such as raising revenue and fostering business initiatives

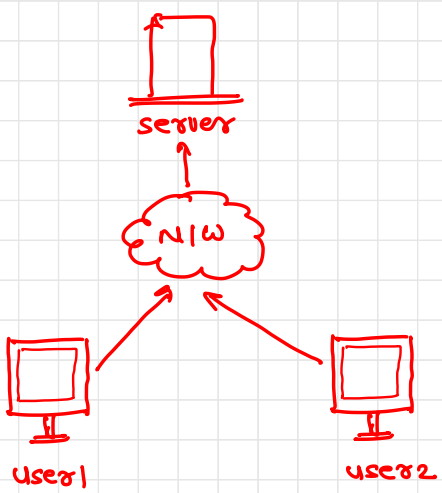
Cloud

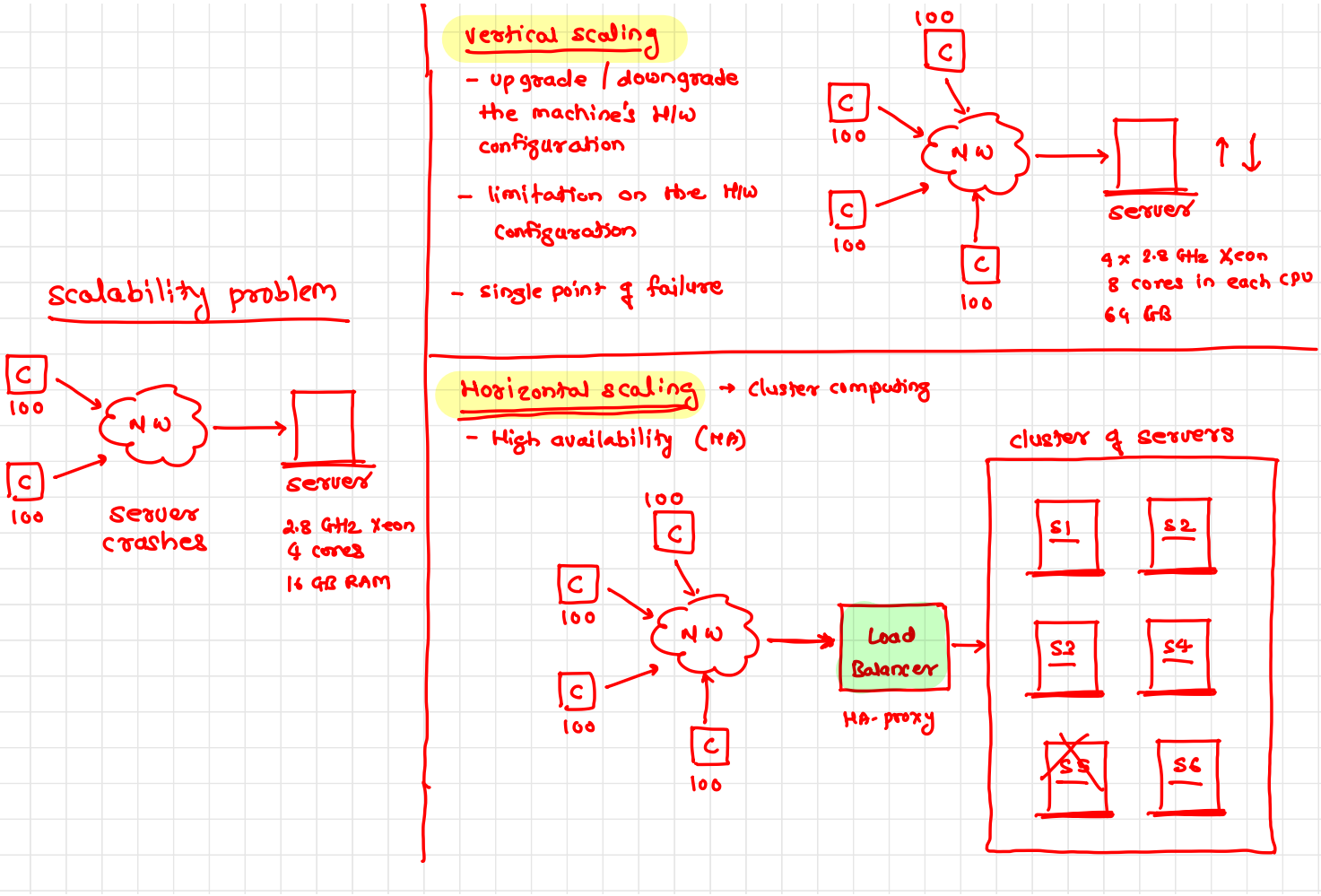
Computing Model

- Desktop computing
- Client-Server computing
- Cluster computing
- Grid computing
- Cloud Computing









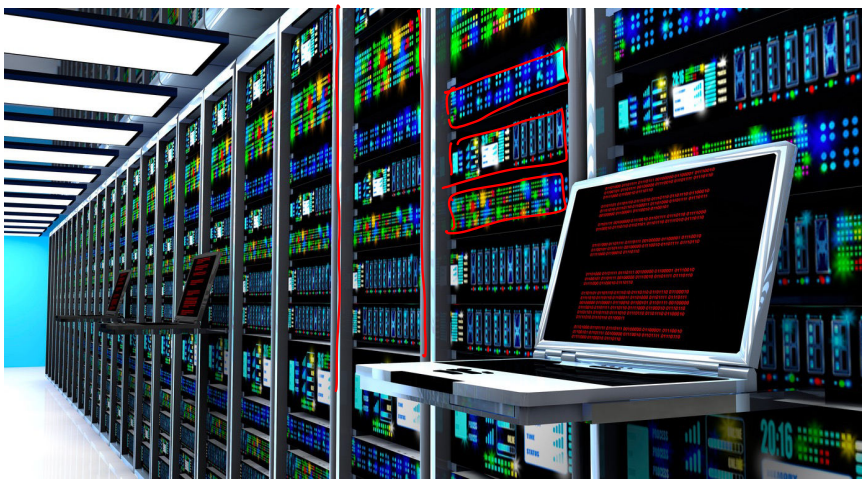
What is cloud computing ?

- The practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer.
- Is the delivery of on-demand computing resources – everything from data centers over the internet on a pay for use basis
- Cloud computing is an umbrella term used to refer to Internet based development and services

network

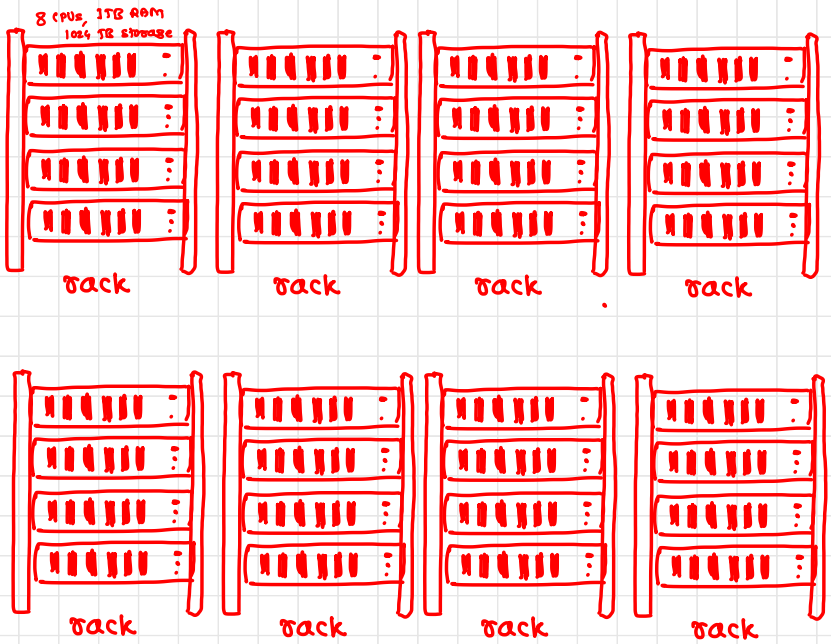
What is Data Center ?

- Where your IT devices and applications are located
- For a non-technical person it is the cloud where the user's files/data is stored
- Components
  - Servers
  - Security
  - WAN
  - Storage
  - File Sharing



server
- rack server
- tower server

cloud provider
- region
  - availability zone
    - data center
      - building
        - floors
          - rooms
            - rack rows
              - racks
                - servers
                  - cpu
                  - RAM
                  - storage



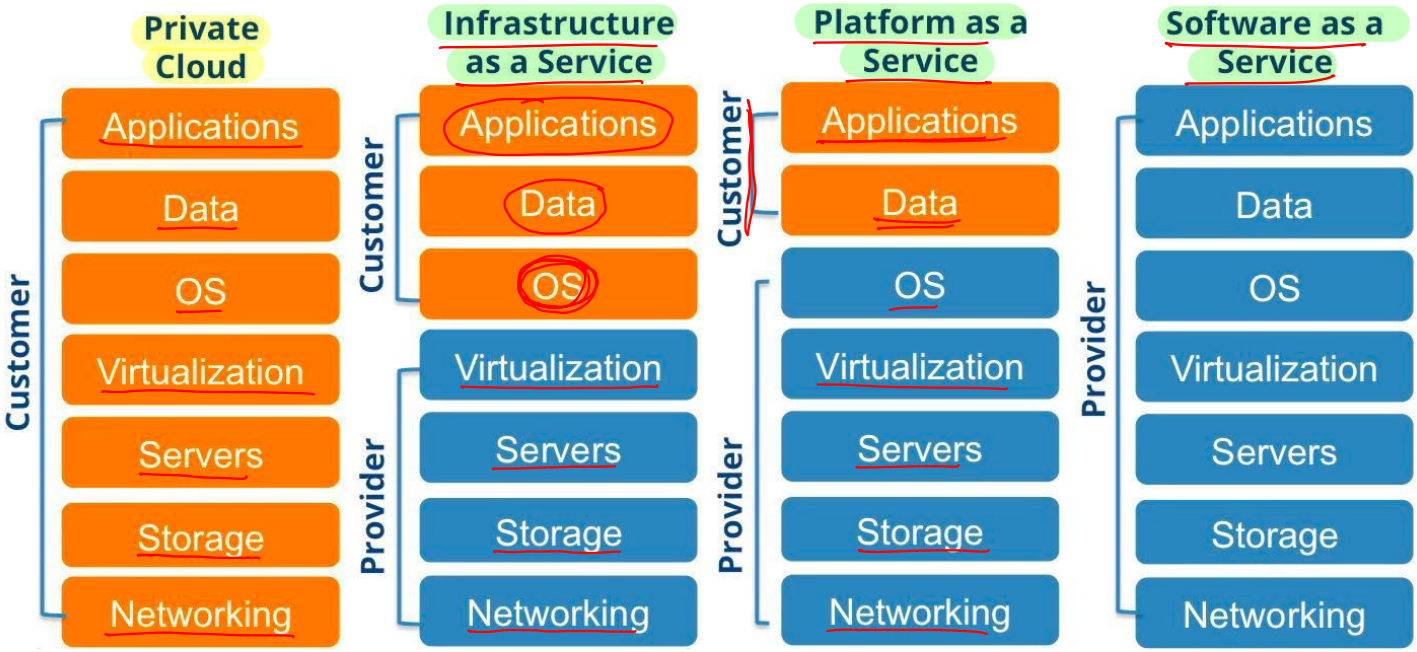
What is Virtualization ?

- Refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources
- Types
  - Type I
  - Type II
    - Containerization

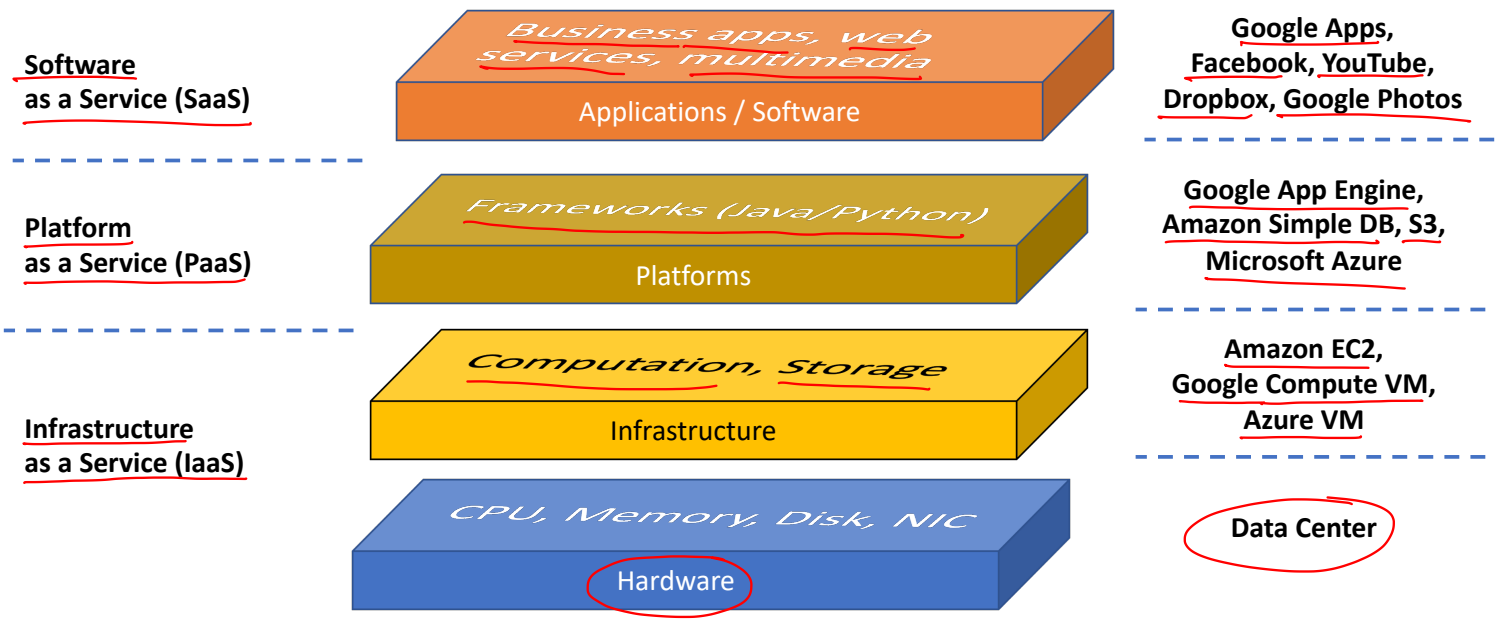
Terminologies

- Scalability
  - refers to the idea of a system in which every application or piece of infrastructure can be expanded to handle increased load
- Elasticity
  - the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible
- Availability
  - refers to the ability of a user to access information or resources in a specified location and in the correct format
- Information Assurance
  - availability, integrity, authentication, confidentiality and nonrepudiation
- On-demand service
  - A model by which a customer can purchase cloud services as needed

Service Models



Service Models



Service Models: IaaS

- Infrastructure as a Service
- Allocates virtualized computing resources to the user through the internet
- IaaS is completely provisioned and managed over the internet
- helps the users to avoid the cost and complexity of purchasing and managing their own physical servers
- Every resource of IaaS is offered as an individual service component and the users only have to use the particular one they need
- The cloud service provider manages the IaaS infrastructure while the users can concentrate on installing, configuring and managing their software
- Generally meant for operations team to setup the required infrastructure
- Benefits
  - Time and cost savings: more installation and maintenance of IT hardware in-house,
  - Better flexibility: On-demand hardware resources that can be tailored to your needs,
  - Remote access and resource management.

Service Models: PaaS

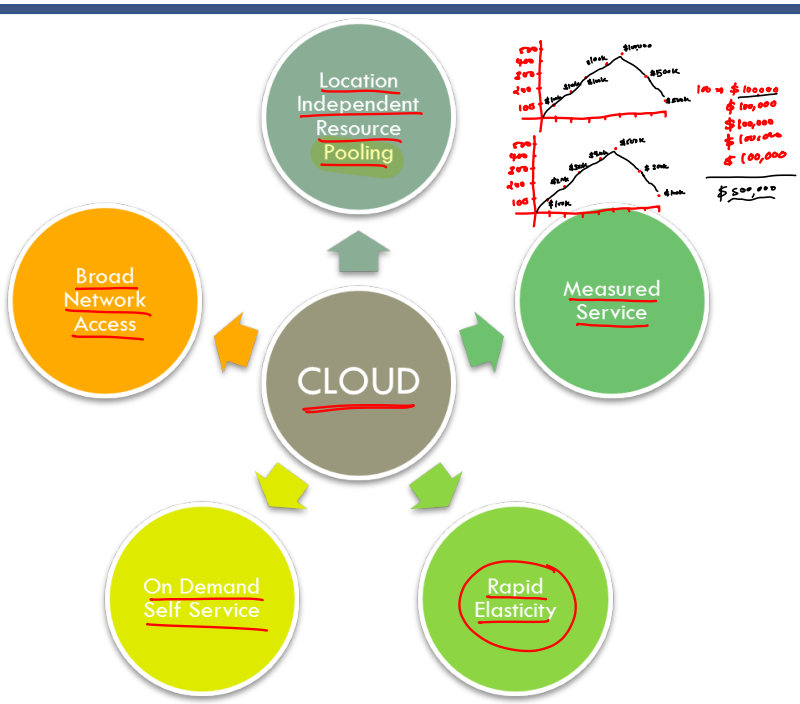
- Provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app
- Generally meant for developers
- Benefits
  - Mastering the installation and development of software applications
  - Time saving and flexibility for development projects: no need to manage the implementation of the platform, instant production
  - Data security: You control the distribution, protection, and backup of your business data

Service Models: SaaS

- Software as a Service
- Software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet
- User wont know which computer or operating system or infrastructure is used to host the software
- Generally meant for end user
- Benefits
  - You are entirely free from the infrastructure management and aligning software environment: no installation or software maintenance
  - You benefit from automatic updates with the guarantee that all users have the same software version
  - It enables easy and quicker testing of new software solutions.

Cloud Computing Characteristics

- Rapid Elasticity
- On Demand Self Service
- Broad Network Access
- Location Independent Resource Sharing (pooling)
- Measured Services



Cloud Deployment Models: Public

- Supports all users who want to make use of a computing resource, such as hardware (OS, CPU, memory, storage) or software (application server, database) on a subscription basis
- Most common uses of public clouds are for application development and testing, tasks such as file-sharing, and e-mail service
- Requires internet to access the resources

Cloud Deployment Models: Private

- Typically infrastructure used by a single organization
- Such infrastructure may be managed by the organization itself to support various user groups, or it could be managed by a service provider that takes care of it either on-site or off-site
- Private clouds are more expensive than public clouds due to the capital expenditure involved in acquiring and maintaining them
- However, private clouds are better able to address the security and privacy concerns of organizations

Cloud Deployment Models: Hybrid

- Organization makes use of interconnected private and public cloud infrastructure
- Many organizations make use of this model when they need to scale up their IT infrastructure rapidly, such as when leveraging public clouds to supplement the capacity available within a private cloud
- For example, if an online retailer needs more computing resources to run its Web applications during the holiday season it may attain those resources via public clouds.

Cloud Services

- Compute: used to create the Virtual Machine [CPU]
- Storage: used to provide the storage
- Database: RDBMS + No SQL
- Security and Identity Management
- Media Services
- Machine Learning
- Cost Management
- Application Integration

Advantages

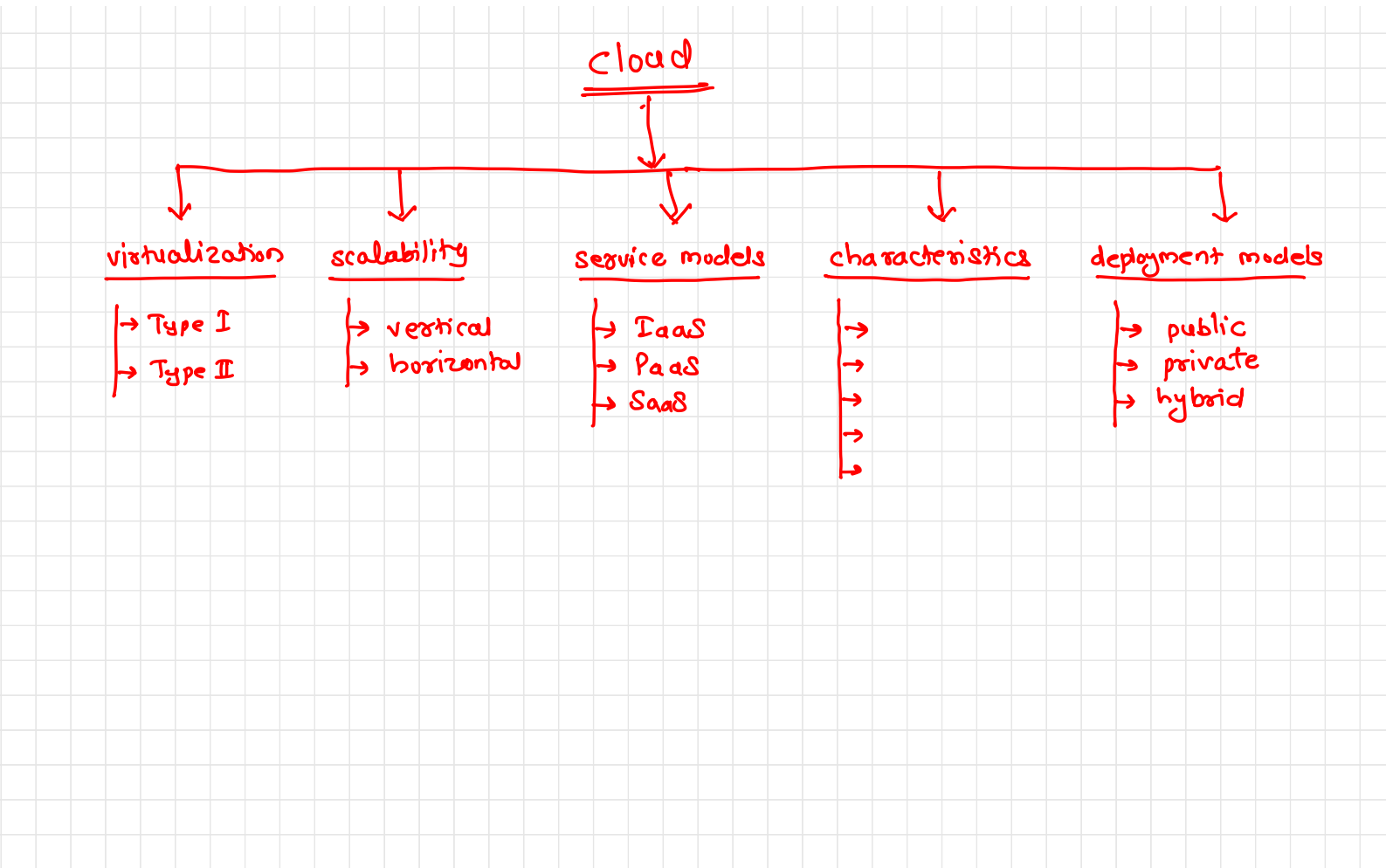
- Lower computer costs
- Improved performance
- Reduced software costs
- Instant software updates
- Improved document format compatibility
- Unlimited storage capacity
- Increased data reliability
- Universal document access
- Latest version availability

Disadvantages

- Requires a constant Internet connection
- Does not work well with low-speed connections
- Features might be limited
- Stored data might not be secure
- Stored data can be lost
- Each cloud systems uses different protocols and different APIs

Cloud Providers

- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- Rackspace
- DigitalOcean
- Alibaba Cloud
- Oracle Cloud
- IBM Cloud

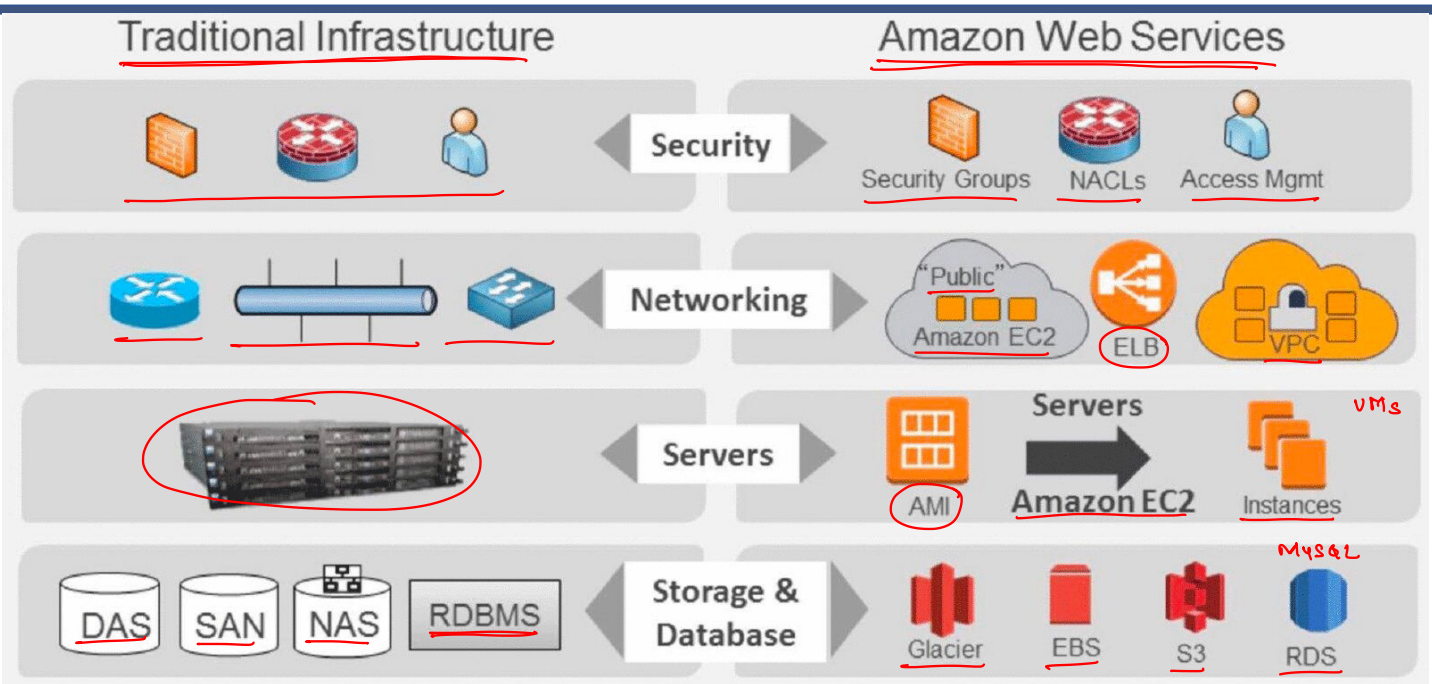


AWS

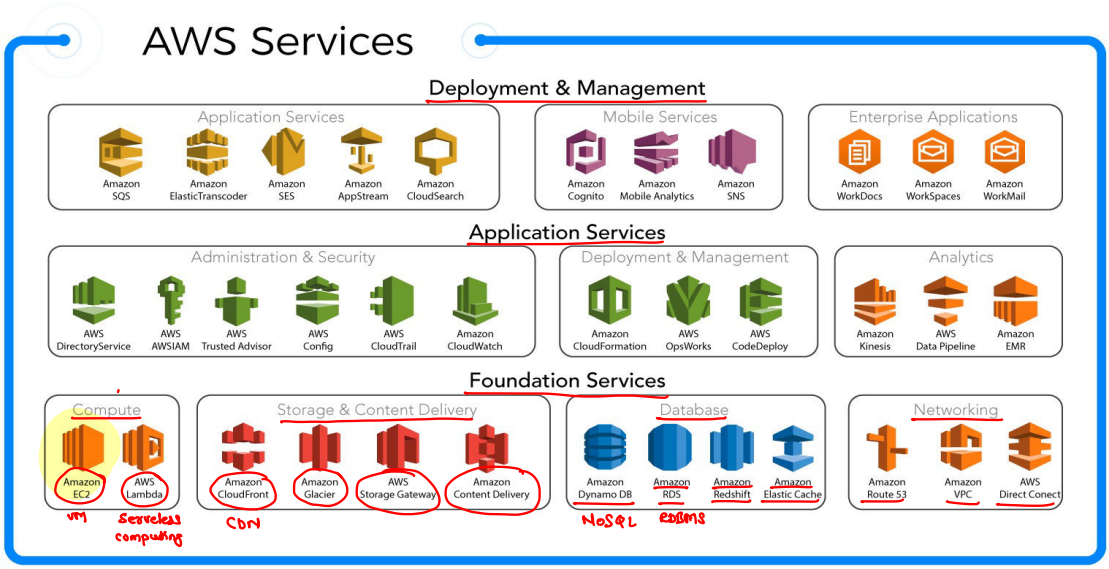
What is AWS ?

- AWS stands for Amazon Web Services
- Platform that offers flexible, reliable, scalable, easy-to-use and cost-effective cloud computing solutions : database, VMs, platforms etc
- Amazon's cloud implementation
- It's a combination of IaaS, PaaS and SaaS offerings

Traditional vs AWS



AWS Services



Global Infrastructure: Region

- Geographic area having availability zone(s)
- Collection of availability zones that are geographically located close to one other
- Every Region will act independently of the others, and each will contain at least two Availability Zones
- E.g.
  - US East: N. Virginia, Ohio
  - US West: N. California, Oregon
  - Asia Pacific: Mumbai, Seoul, Singapore, Sydney, Tokyo

Regions
↳ AZ
↳ DC

Global Infrastructure: Availability Zone [at least 2 DCs]

- Essentially the physical data centers of AWS
- This is where the actual compute, storage, network, and database resources are hosted that we as consumers provision within our Virtual Private Clouds (VPCs)
- Availability Zones are always referenced by their Code Name, which is defined by the AZs Region Code Name that the AZ belongs to, followed by a letter
- E.g.
  - the AZs within the eu-west-1 region (EU Ireland), are
    - eu-west-1a
    - eu-west-1b
    - eu-west-1c

Global Infrastructure: Edge Locations

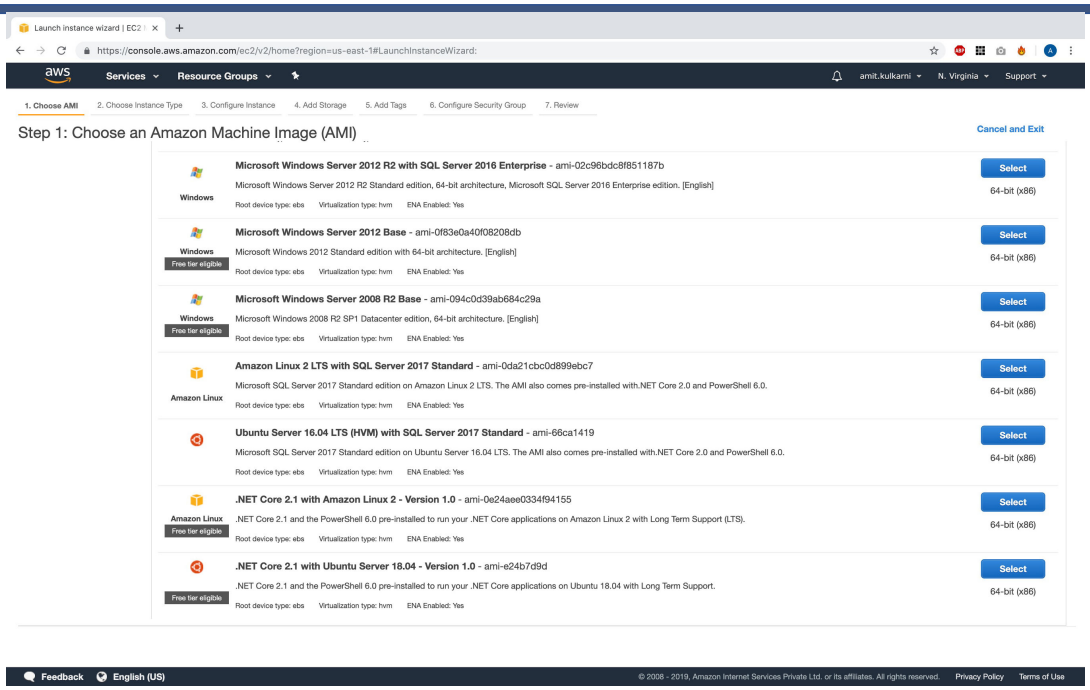
- Edge Locations are AWS sites deployed in major cities and highly populated areas across the globe
- Generally used to cache data and reduce latency for end-user access by using the Edge Locations as a global Content Delivery Network (CDN)
- Edge Locations are primarily used by end users who are accessing and using your services
- E.g.
  - Route 53: DNS Lookup
  - CloudFront
    - Content Delivery Network (CDN)
    - Cached contents, streaming distribution, acceleration

EC2

EC2

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud
- It is a virtual machine you will be building in the cloud
- EC2 instances are designed to mimic traditional on-premise servers, but with the ability to be commissioned and decommissioned on-demand for easy scalability and elasticity
- EC2 supports variety of operating systems:
  - Linux: Amazon Linux, Ubuntu, Red Hat Enterprise, SUSE Linux Enterprise Server, Fedora, Debian, CentOS, Gentoo Linux, Oracle Linux, FreeBSD
  - Windows: Windows Server, Windows
- Every instance comprised of
  - Amazon Machine Image (AMI)
  - Instance type
  - Network Interface
  - Storage

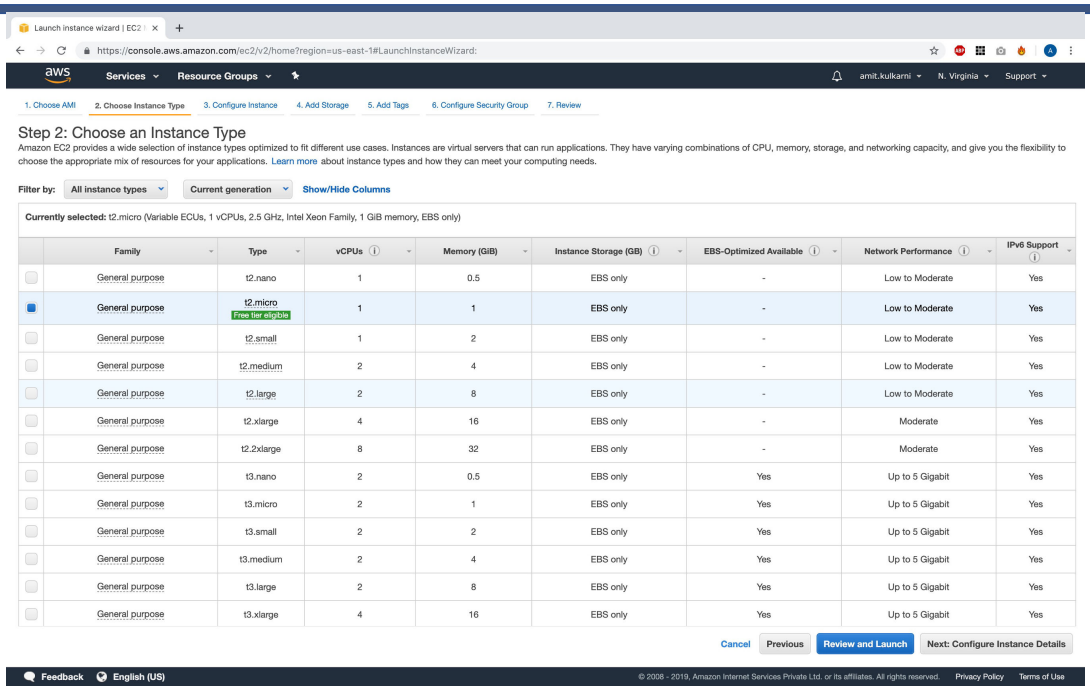
Step 1 - Choose AMI



Amazon Machine Image (AMI)

- Operating System used to create virtual machine (EC2 instance)
- AMI are built for a specific region
- You can copy an AMI from one region to another
- You can also create a custom AMI with required applications/configuration
- AMI contains
  - Template for root volume
  - Launch permissions that control which account can use the AMI
  - EBS mapping that specifies the volume(s) to attach the instance when its launched
- AMI comes into two types
  - Instance store backed AMI
  - EBS backed AMI

Step 2 - Instance Type



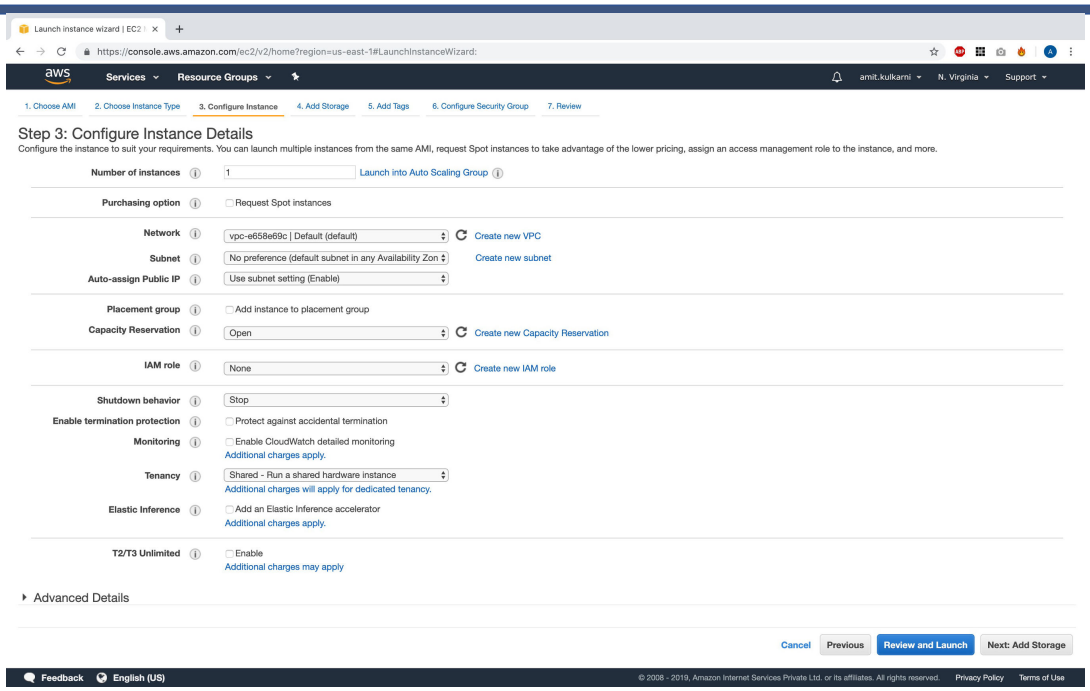
Instance Type

- Used to decide the EC2 instance configuration
- AWS provides various instance types [https://aws.amazon.com/ec2/instance-types/]
  - General purpose: A (ARM), T (Cheapest), M (Main)
  - Compute optimized: C (Compute)
  - Memory optimized: R (RAM), X (Extreme RAM), Z (High compute and memory)
  - Accelerated computing: P (Picture-GPU), G (Graphics), F (Fast)
  - Storage optimized: I (IOPS), D (Data), H (High Disk Throughput)

Instance Types

Type	Category	Description	Use Cases
M5	General Purpose	Balance of compute, memory and network resources	Mid-sized databases
C5	Compute Optimized	Advanced CPUs	Modelling, Analytics
H1	Storage Optimized	Local HDD Storage	Map Reduce
R4	Memory Optimized	More RAM for $	In-memory caching
X1	Memory Optimized	Terabytes of RAM and SSD	In-memory database
I3	IO Optimized	Local SSD storage, high IOPS	NoSQL databases
G3	GPU Graphics	GPUs with video encoders	3d rendering
P3	GPU Compute	GPUs with tensor cores	Machine Learning
F1	Accelerated Computing	FPGA, custom hardware accelerations	Genomics
T2	Burstable	Shared CPUs, lowest cost	Web servers

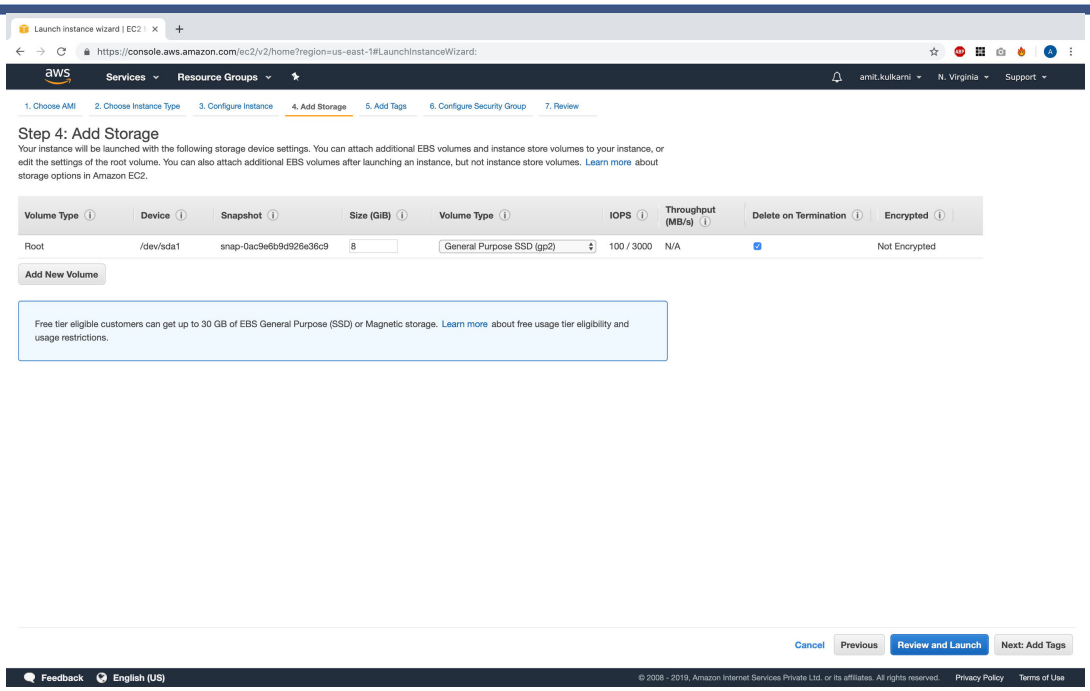
Step 3 – Configure Instance Details



On Demand

- Low cost and flexibility with no upfront cost
- Pay for hours used with no commitment
- Good option for developers or testers
- Ideal for
  - auto scaling groups
  - Unpredictable workloads

Step 4 – Add Storage



Storage options

- Block storage
  - Instance store
  - EBS volume
- Object storage
  - S3
- File sharing
  - EFS

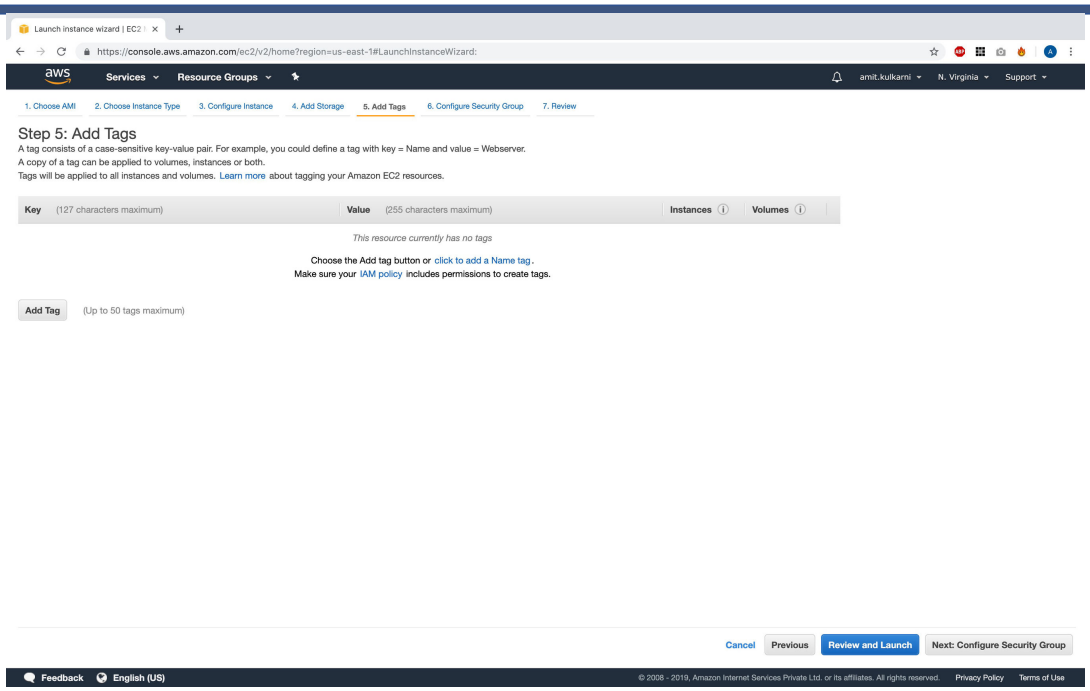
EBS – Solid State Drives (SSD)

- General Purpose (gp2)
  - Used for dev/test environments
  - Baseline performance: 100 IOPS – 3000 IOPS
  - Bursts up to 32000 IOPS (credit based)
  - Volume size from 1GiB to 16 TiB
- Provisioned IOPS (io1)
  - Used for mission critical applications that require sustained IOPS performance
  - Large database workloads
  - Volume size: 4GiB to 16TiB
  - Performs at provisioned level and can provision up to 64K IOPS per volume
  - NOTE: EC2 instances have a maximum of 80K IOPS total

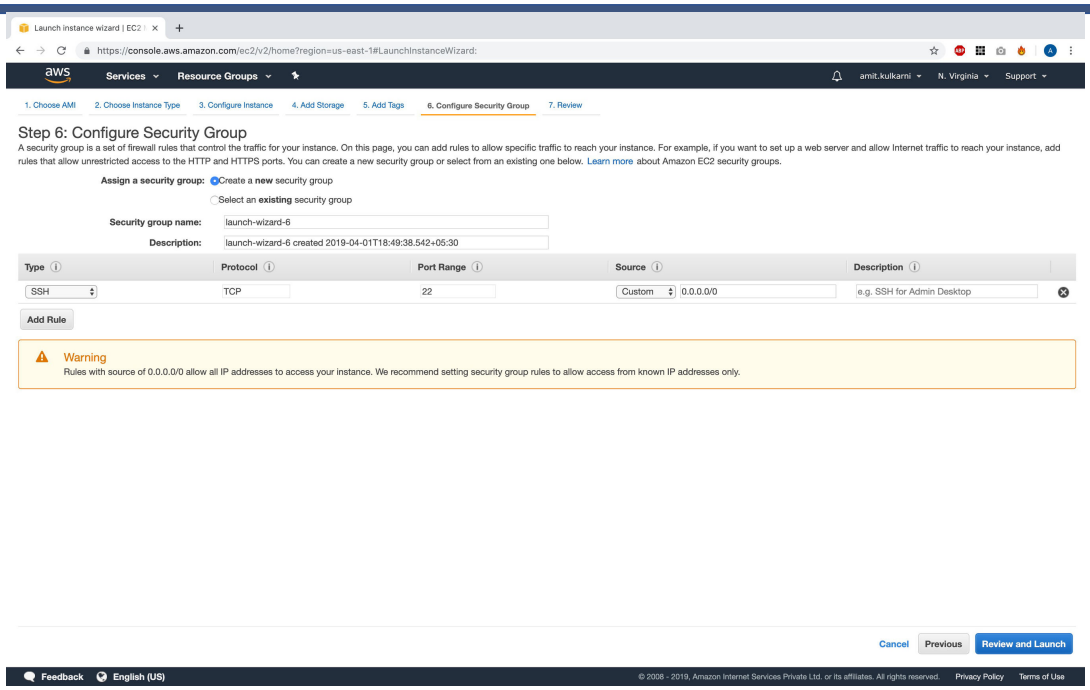
EBS – Hard Disk Drive (HDD)

- Can not be boot volume
- Size: 500GiB – 16TiB
- Types
  - Throughput Optimized (st1)
    - Low storage cost
    - Used for frequently accessed, throughout intensive workloads
    - E.g. Streaming, Big Data
    - Max throughput of 500 MiB/s
    - NOTE: EC2 instances have a maximum throughput of 1750MiB/s
  - Cold HDD (sc1)
    - Lowest cost
    - Infrequently accessed data
    - Max throughput of 250MiB/s

Step 5 – Add Tags



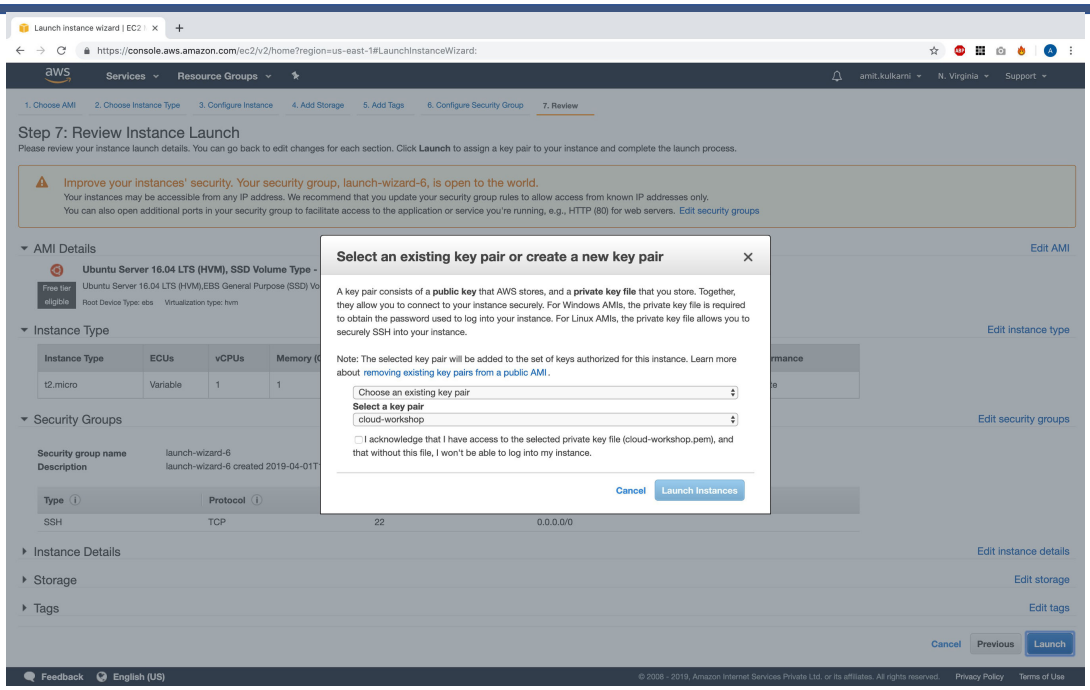
Step 6 - Add Security Group



Security Group

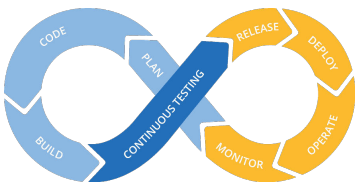
- Acts as a virtual firewall for your instance to control inbound and outbound traffic
- Controls the ports and protocols that can reach the front-end listener
- Every EC2 instance must have at least one security group attached
- Up to 5 security groups can be attached to an EC2 instance
- Security groups act at the instance level, not the subnet level
- Security group contains rules
  - You can specify allow rules, but not deny rules
  - You can specify separate rules for inbound and outbound traffic
  - When you create a security group, it has no inbound rules
  - By default, a security group includes an outbound rule that allows all outbound traffic
  - Security groups are stateful
  - Instances associated with a security group can't talk to each other unless you add rules allowing it
  - Security groups are associated with network interfaces

Step 7 – Review and create PEM file



EC2 Key Pairs

- Uses PEM format (Privacy Enhanced Mail)
- Used to authenticate a client when logging into EC2 instance
- Each key pair consists of a public key and a private key
- AWS stores the public key on the instance and your are responsible for storing the private key
- To log into the instance you must create and authenticate with key pair
  - Linux instances have no password and you use a key pair to log in
  - With windows you use a key pair to obtain the administrator password and then log into the instance with RDP
- During the creation process of an EC2 instance you are required to either create a new key pair or use existing pair
- The private key is available for download and stored on your local drive
- NOTE: it will be available only once in the form of .pem file



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 →

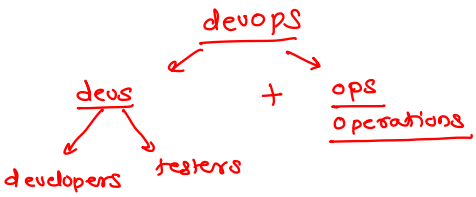
containers = Docker

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

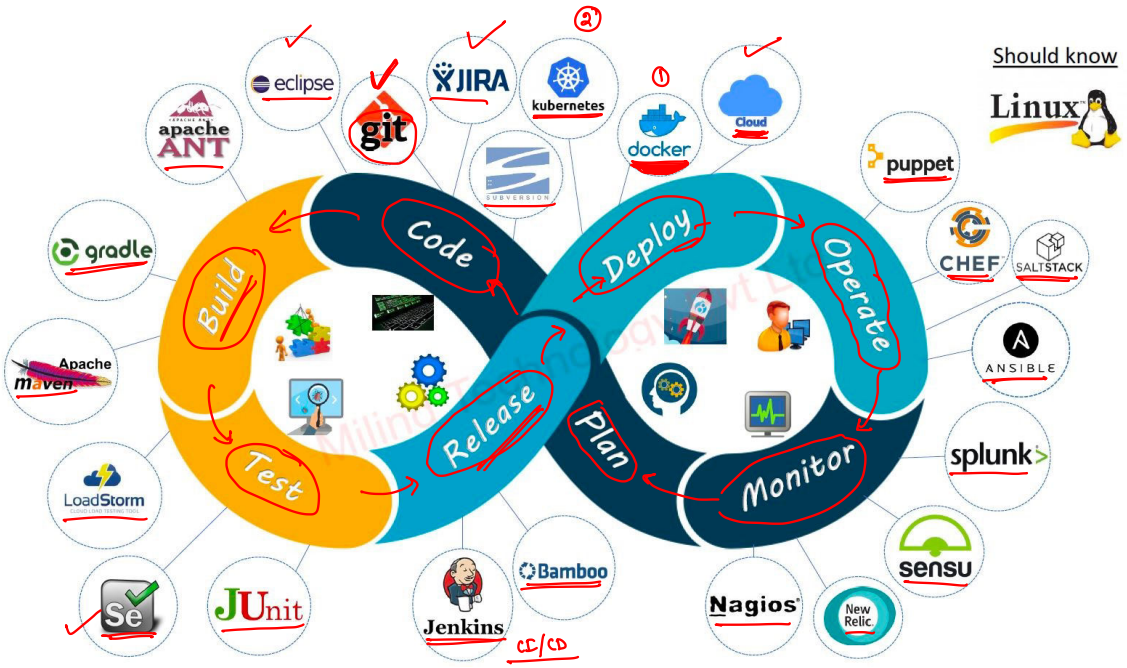
Reasons to use DevOps

- **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

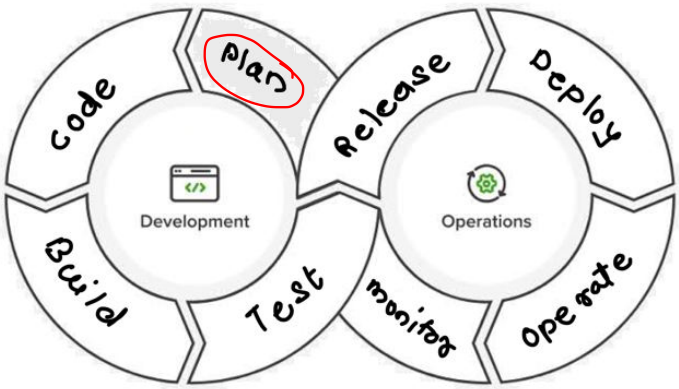
- **Cost Efficiency:** DevOps offers cost efficiency in the software development process which is always an aspiration of IT companies' management
- **Breaks larger code base into small pieces:** DevOps is based on the agile programming method. Therefore, it allows breaking larger code bases into smaller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

DevOps Lifecycle



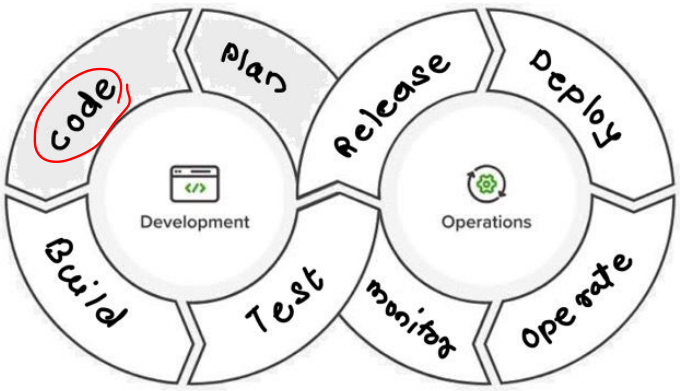
DevOps Lifecycle - Plan

- 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



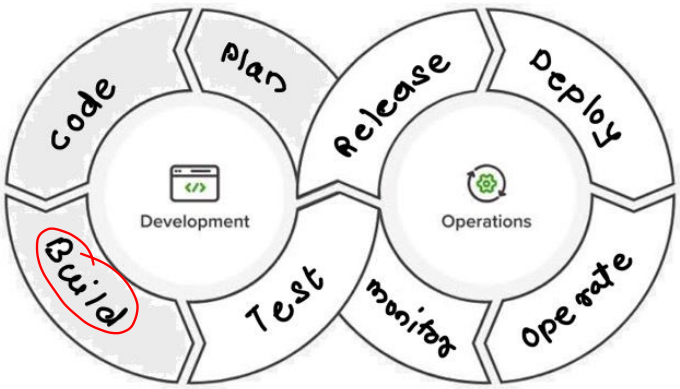
DevOps Lifecycle - Code

- 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.



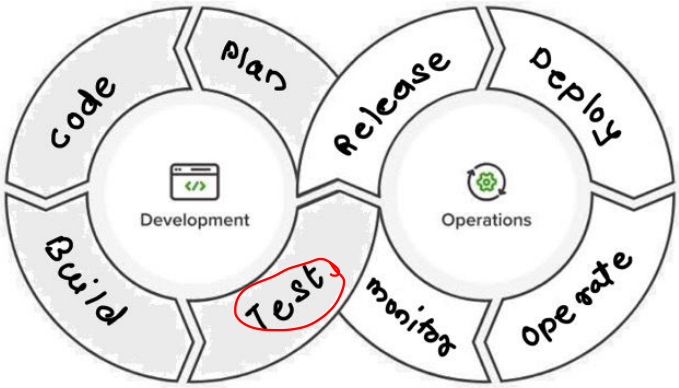
DevOps Lifecycle -Build

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



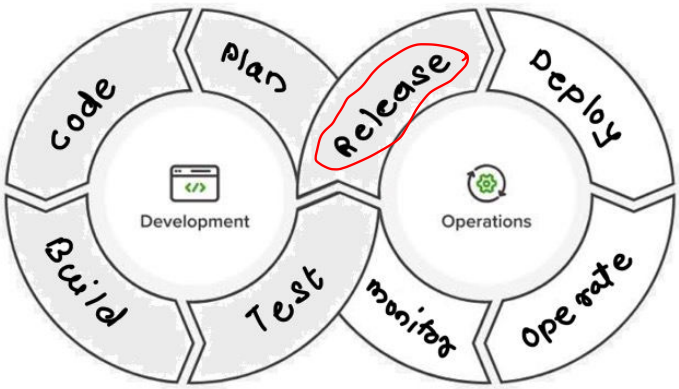
DevOps Lifecycle - Test

- 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



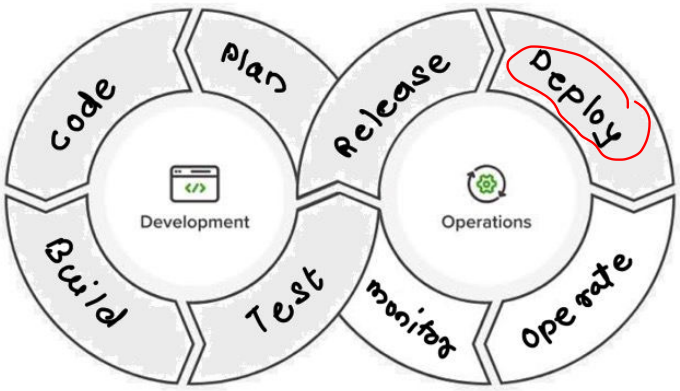
DevOps Lifecycle - Release

- 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

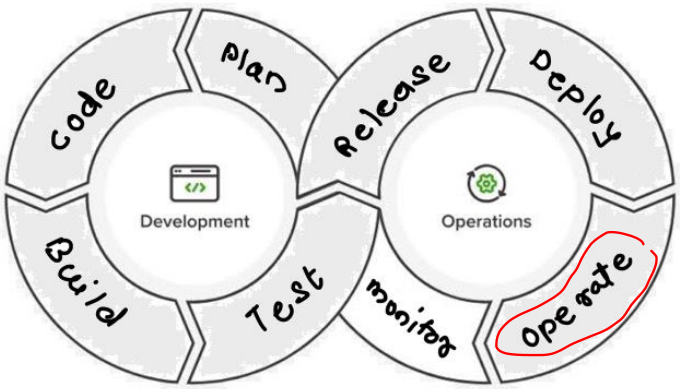
- Manage and maintain development and deployment of software systems and server in any computational environment
- Deployment tools
  - Docker
  - Kubernetes
  - Virtual Machines
- Configuration management tools
  - Puppet
  - Chef
  - Ansible



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

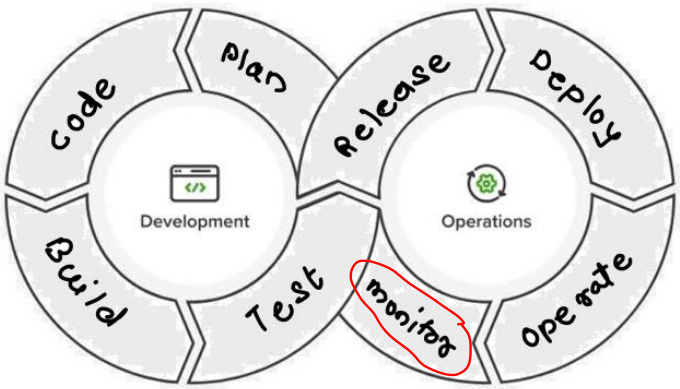
- Linux : RHCSA ($150) / RHCE ($200) , Linux+ ($250)
- cloud : AWS : 45 certificates :- Associate ($150) + professional ($300) → architect, developer, sysops
  GCP : 20 certificates
  Azure : 19 certificates

- Docker : DCA ($200)
- Kubernetes : CKA ($300), CKAD ($300), CKS ($300)
- 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

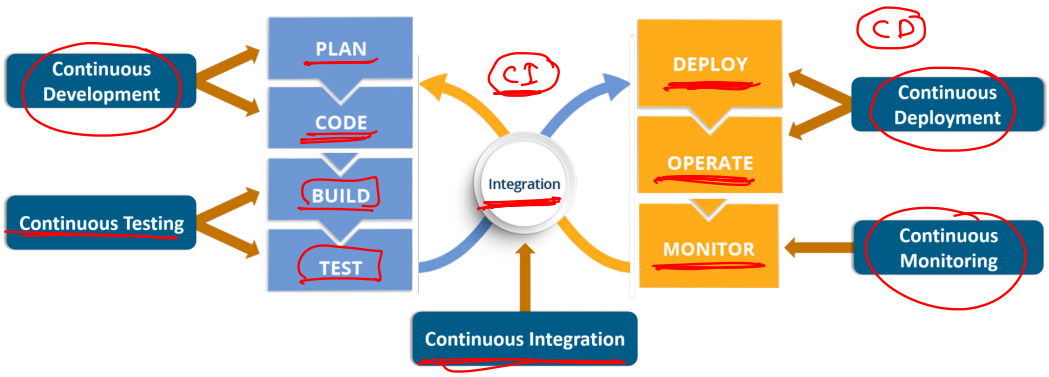
new relic



DevOps Terminologies

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

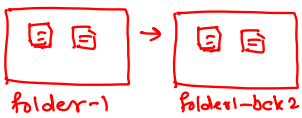
Continous Learning :)



Source Code Management

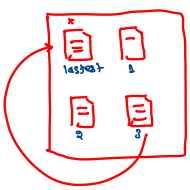
Why do we need Version Control System?

- 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 → SCM



What is Version Control System?

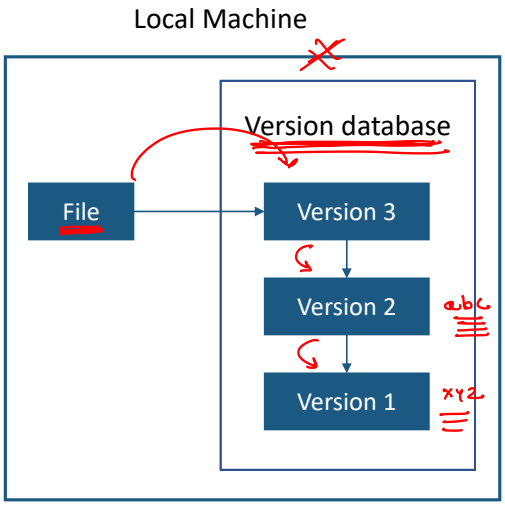
- System that records changes to a file(s) over time so that you can recall specific versions later
- It allows you
  - to revert files to a previous state
  - to revert the entire project to a previous state
  - to compare changes over time
  - to see who last modified something that might be causing 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

deprecated

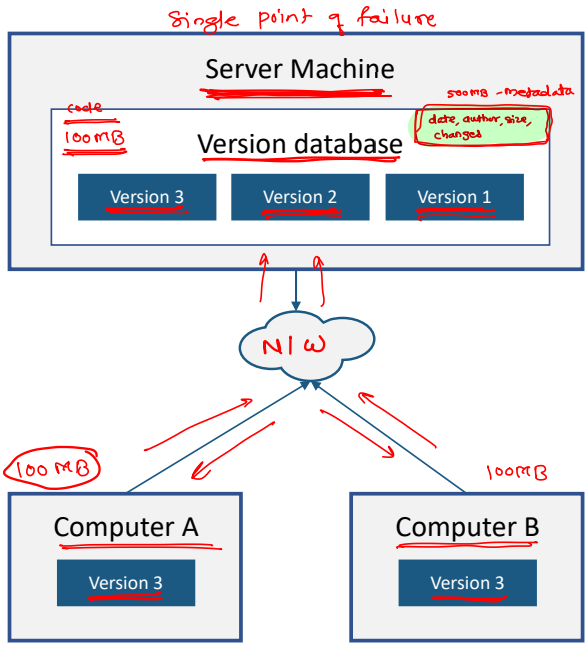
- Contains simple database that kept all the changes to files under revision control
- One of the more popular VCS tools was a system called RCS
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk
- It can then re-create what any file looked like at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



Centralized Version Control Systems

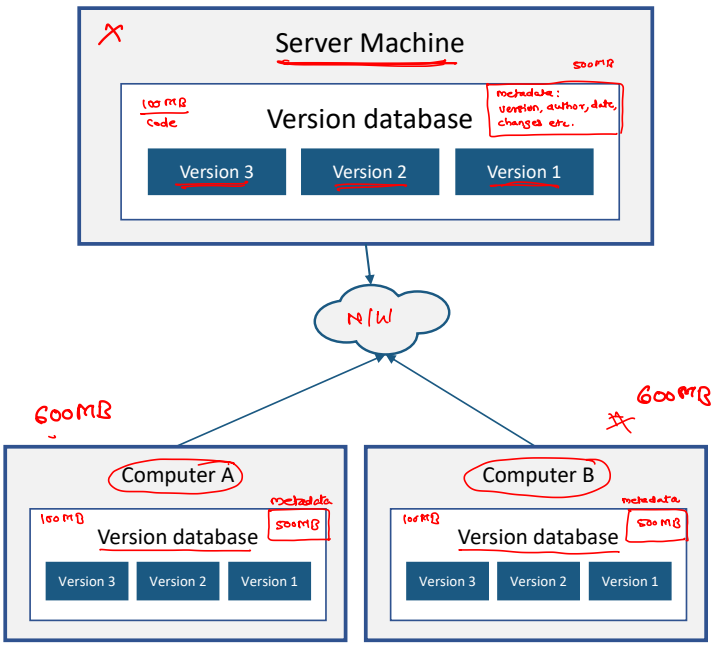
- People need to collaborate with developers on other systems
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place
- E.g.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

disadvantages
- single point of failure
- not scalable



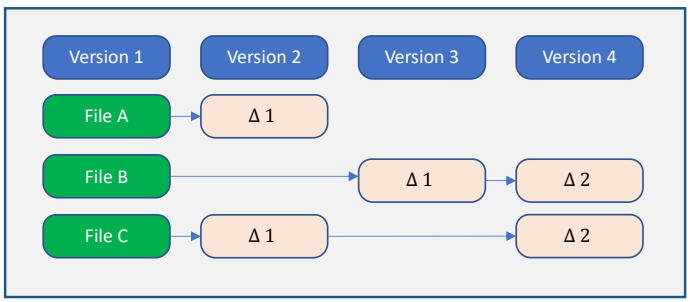
Distributed Version Control Systems

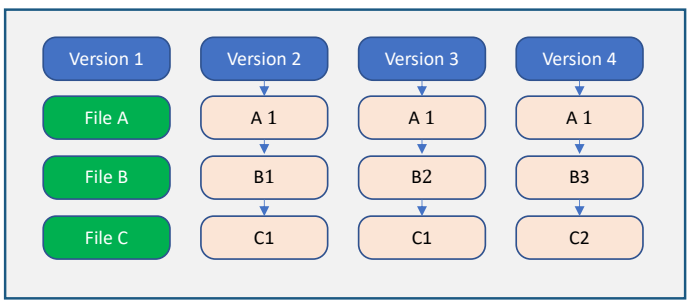
- Clients don't just check out the latest snapshot of the files, rather they fully mirror the repository
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- Every checkout is really a full backup of 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

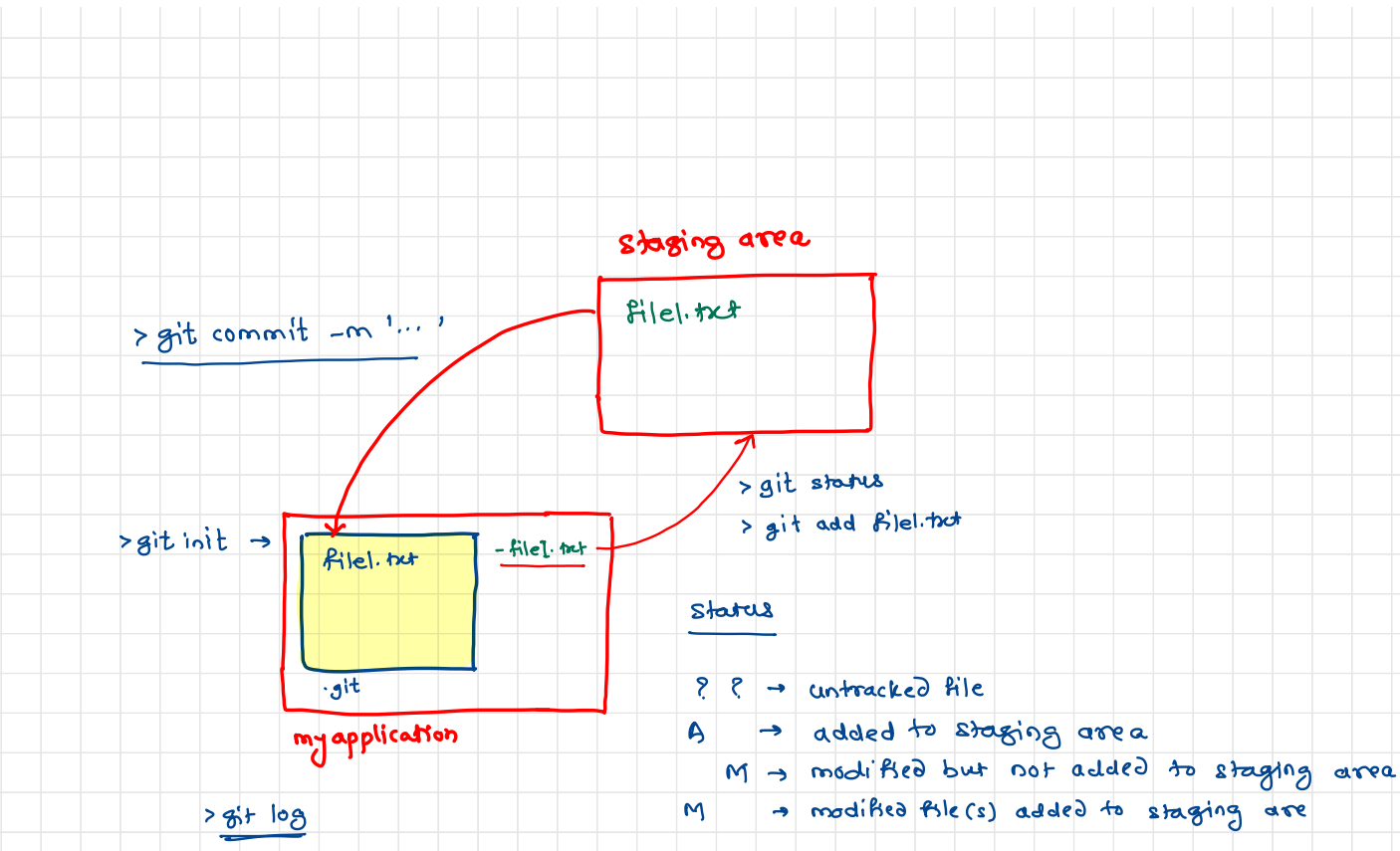
- The Linux kernel is an open source software project of very large scope
- From 1991–2002, changes to the software were passed around as patches and archived files
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper
- In 2005, the relationship with BitKeeper broken down 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

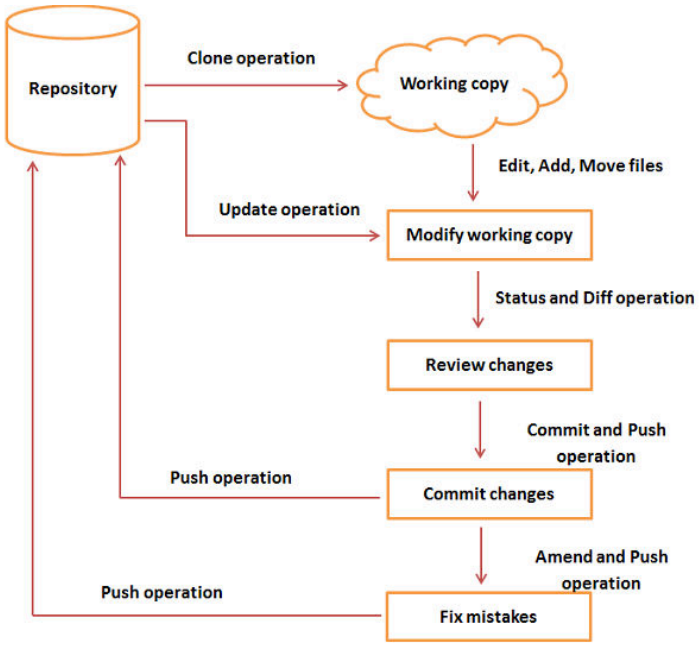
Terminologies

- 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>

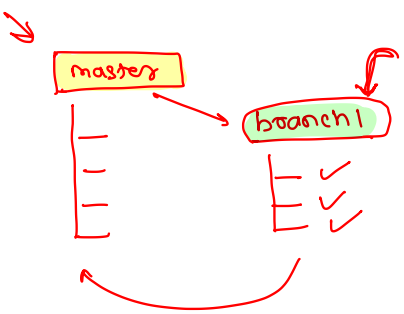
- **Remove unwanted changes**

  > 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**

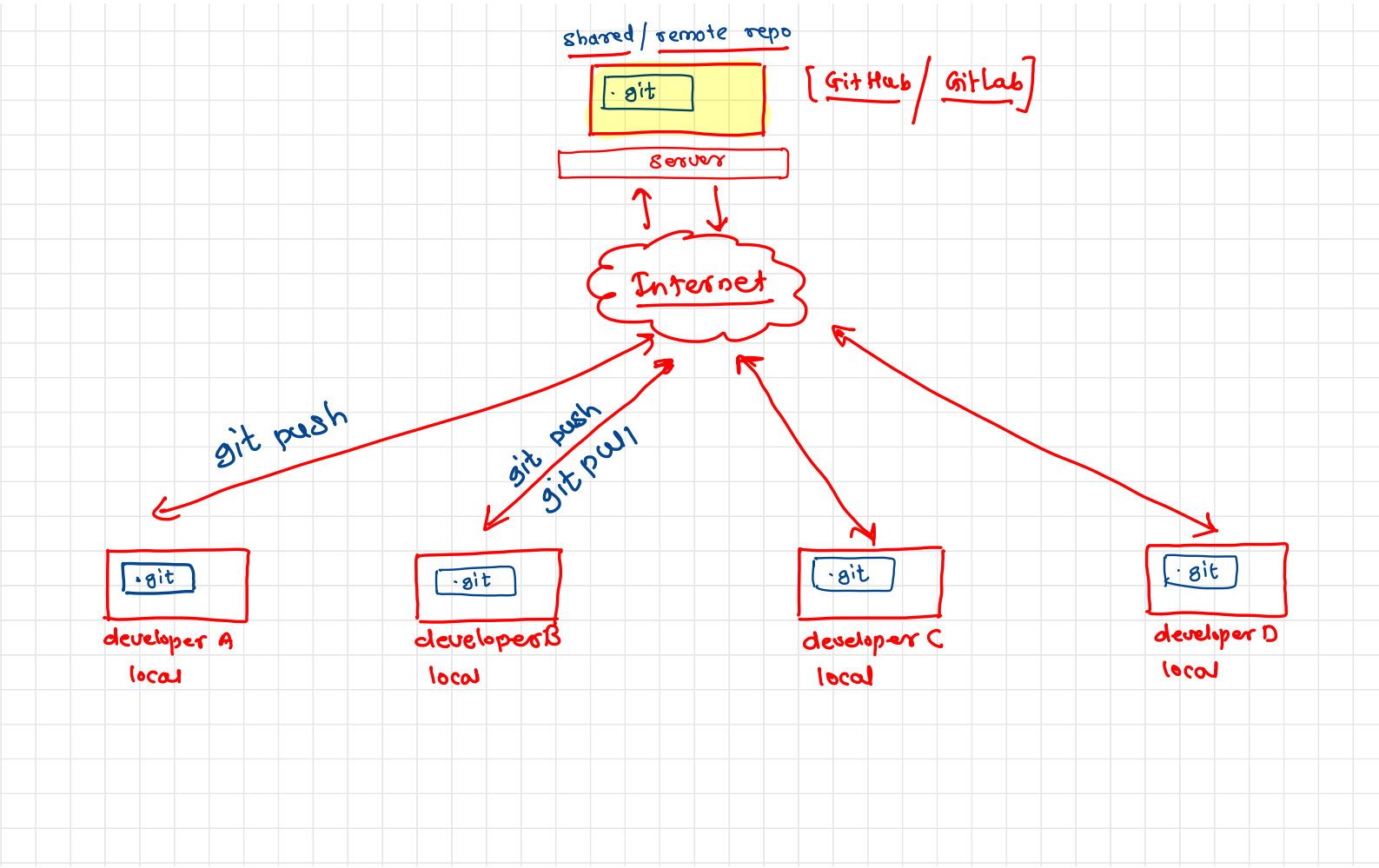
  > git checkout <branch name>

- **Merge a branch**

  > git merge <branch name>

- **Delete a branch**

  > git branch –d <branch name>


shared / remote repo
.git
[GitHub / GitLab]
server
Internet
git push
git push
git pull
.git
developer A
local
.git
developerB
local
.git
developer C
local
.git
developer D
local

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>

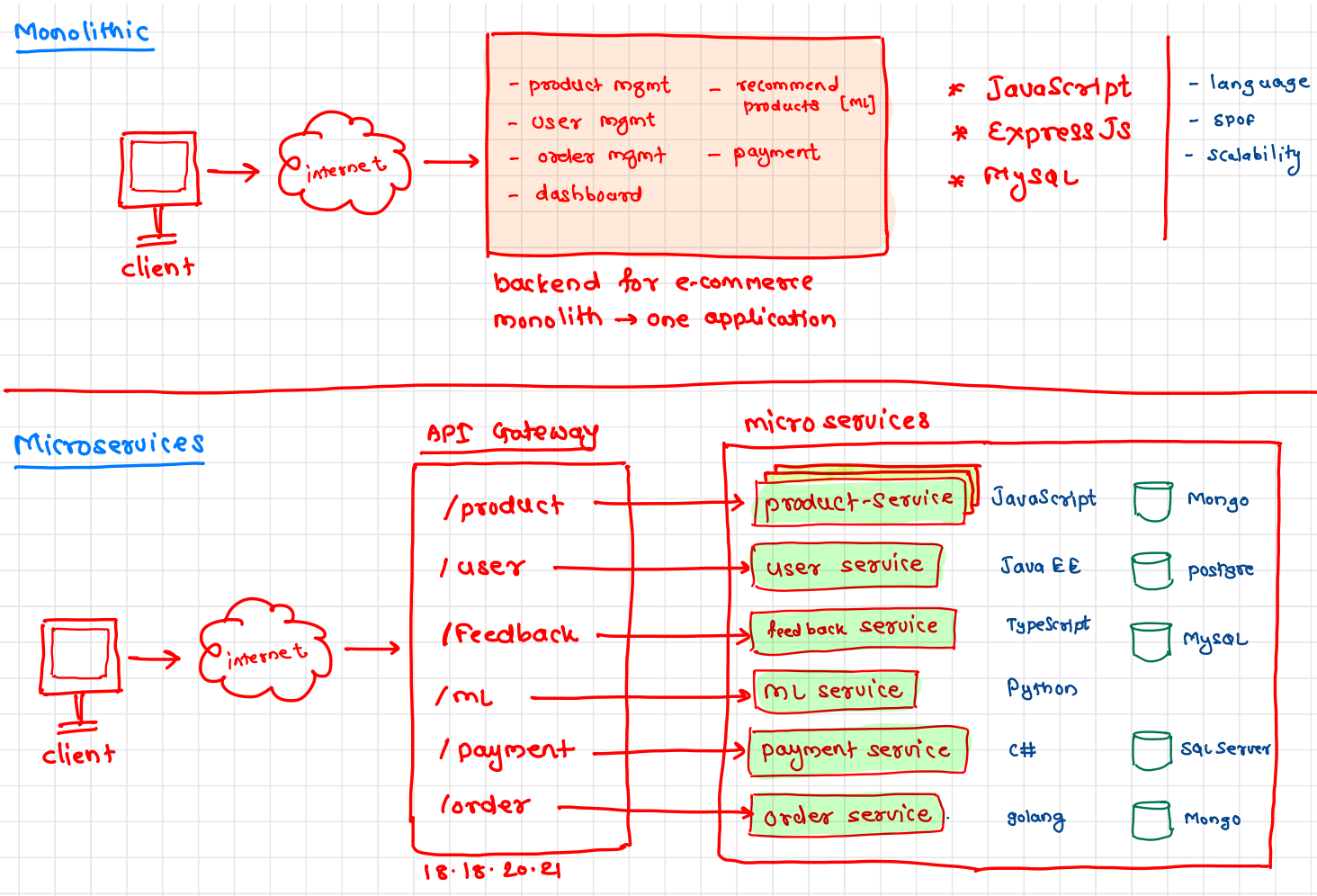
- **Push the changes**

  > git push <name> <branch>

- **Pull the changes**

  > git pull

Microservices

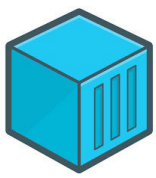


Overview

- 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

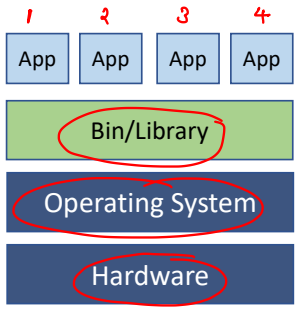
- Easier to Build and Maintain Apps
- Improved Productivity and Speed
- Code for different services can be written in different languages
- Services can be deployed and then redeployed independently without compromising the integrity of an application
- Better fault isolation; if one microservice fails, 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

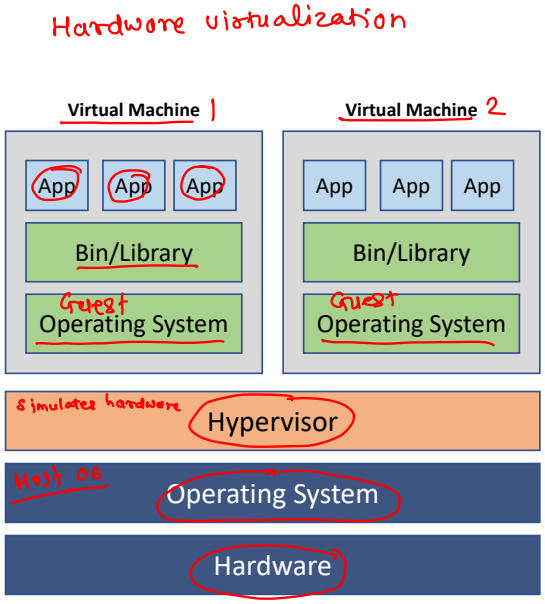
Traditional Deployment

- 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

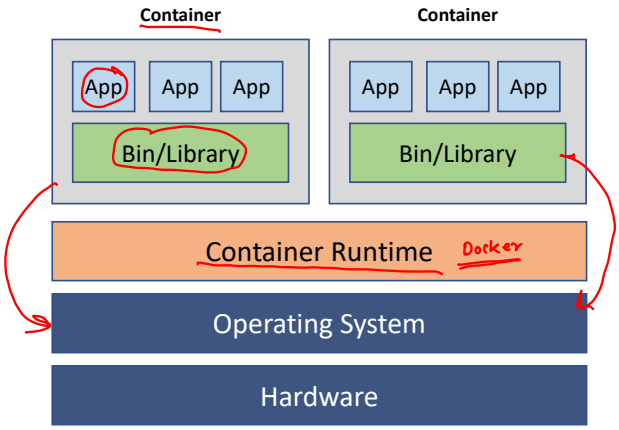


Container deployment

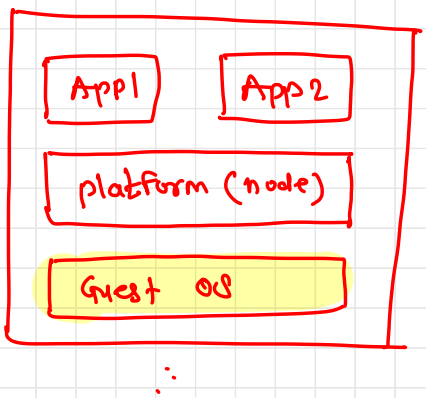
container - lightweight VM

- 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

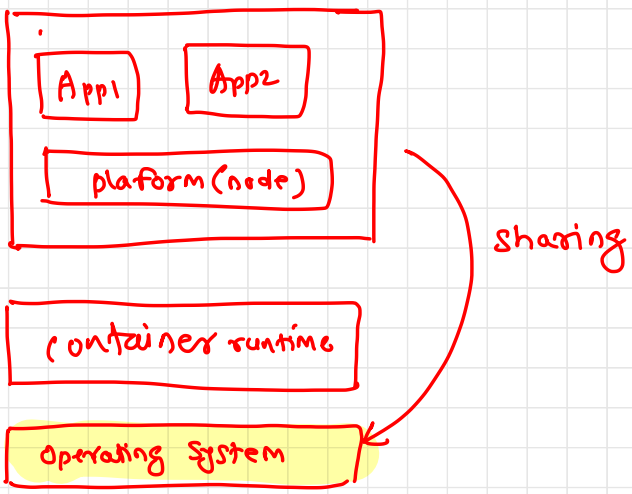
AWS, GCP, Azure
Linux, Mac OS, Windows



virtualization



containerization



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

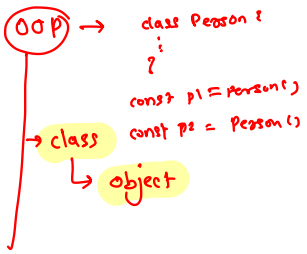
Docker

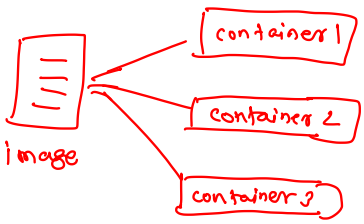
Overview

- 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**

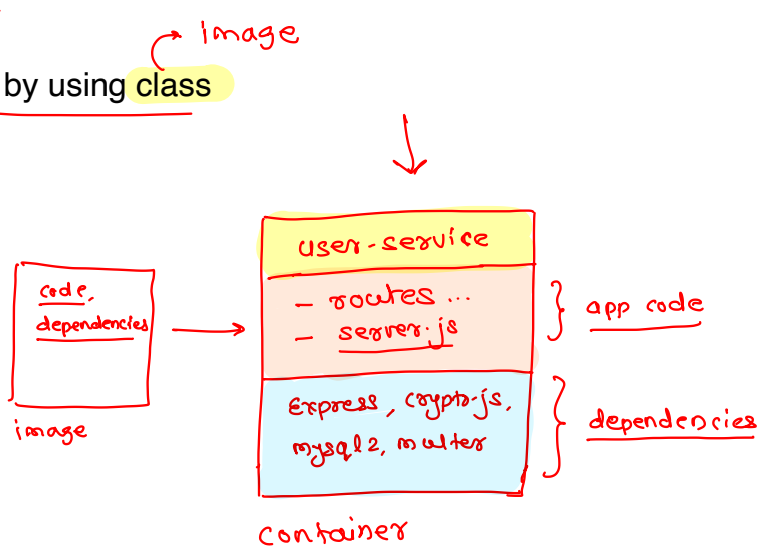
  > docker image tag <image name> <tag>

- **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>

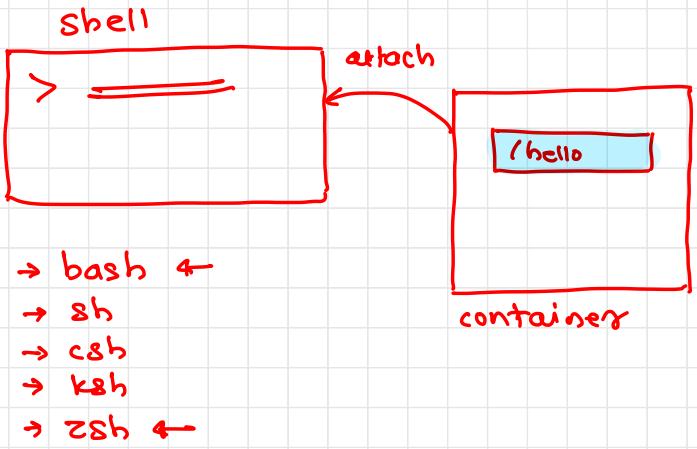
Commands related to containers

- **Create an image from current state of a container**

  > docker container commit <container>

- **Get the processes running in the container**

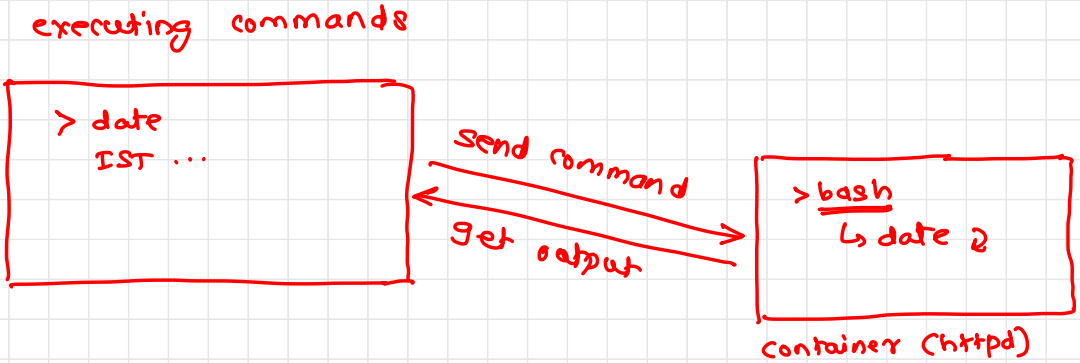
  > docker container top <container>

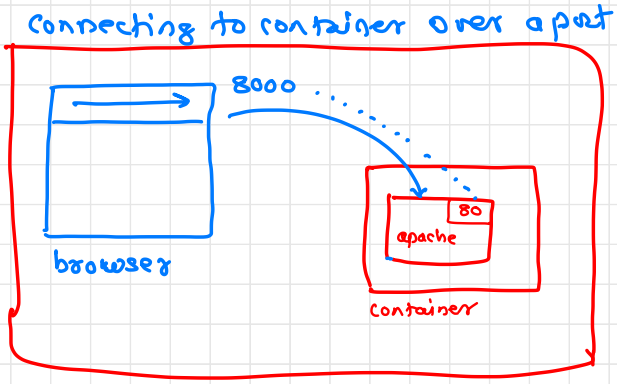


container starts

→ attached

→ detached





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

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

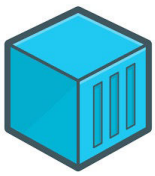
- Represents key-value pair
- The value can be identified by using unique key
- Key and value are separated by using colon (:)
- E.g.
  - name: person1
  - address: “India”
  - phone: +9145434345
  - age: 40
  - hobbies:
  - - reading
  - - playing

Sequence

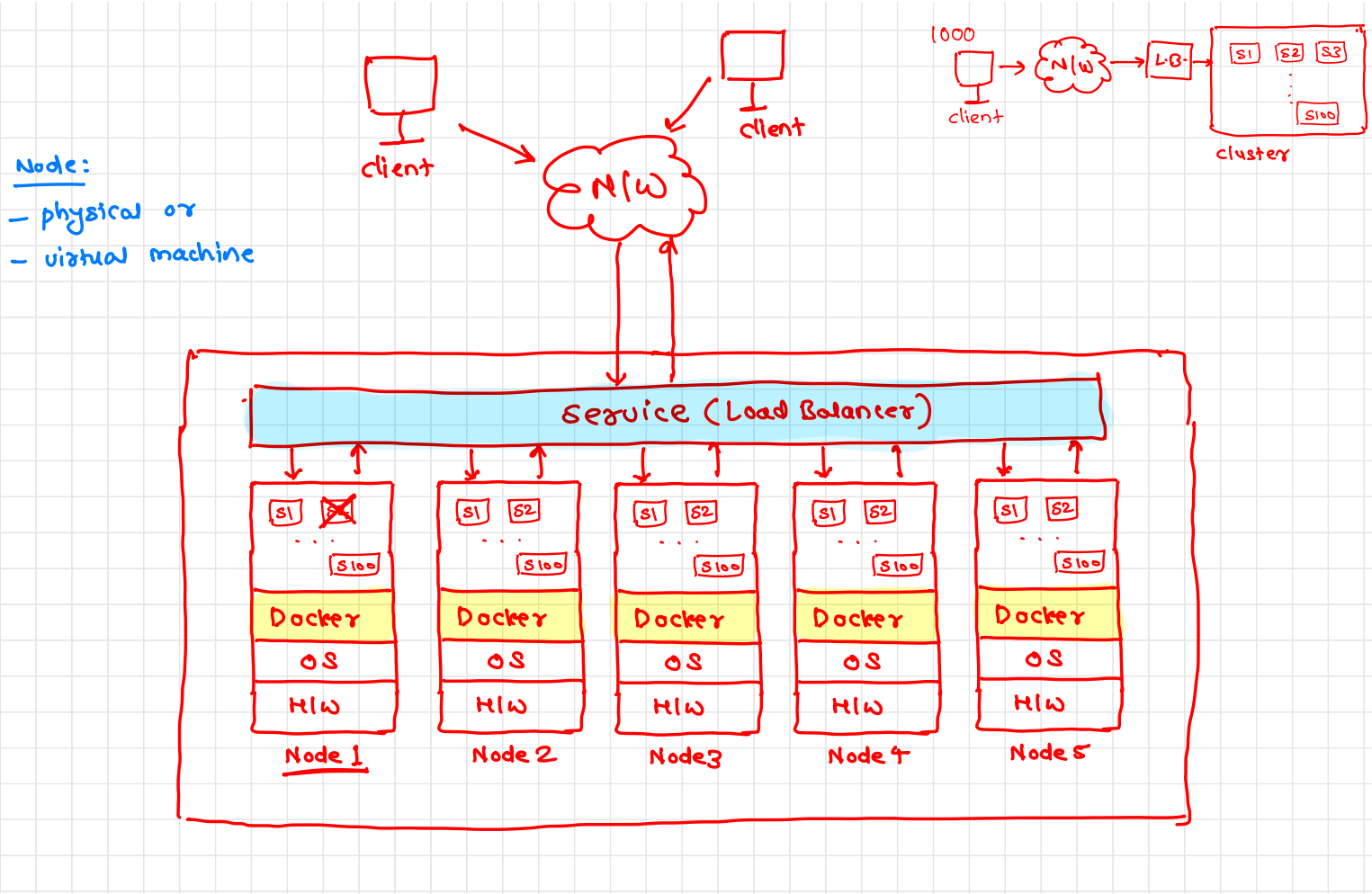
- Represents list of values
- Must be written on separate lines using dash and space
- Please note that space after dash is mandatory
- E.g.
  - # pet animals
  - - cat
  - - dog
  - # programming languages
  - - C
  - - C++
  - - Java

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



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

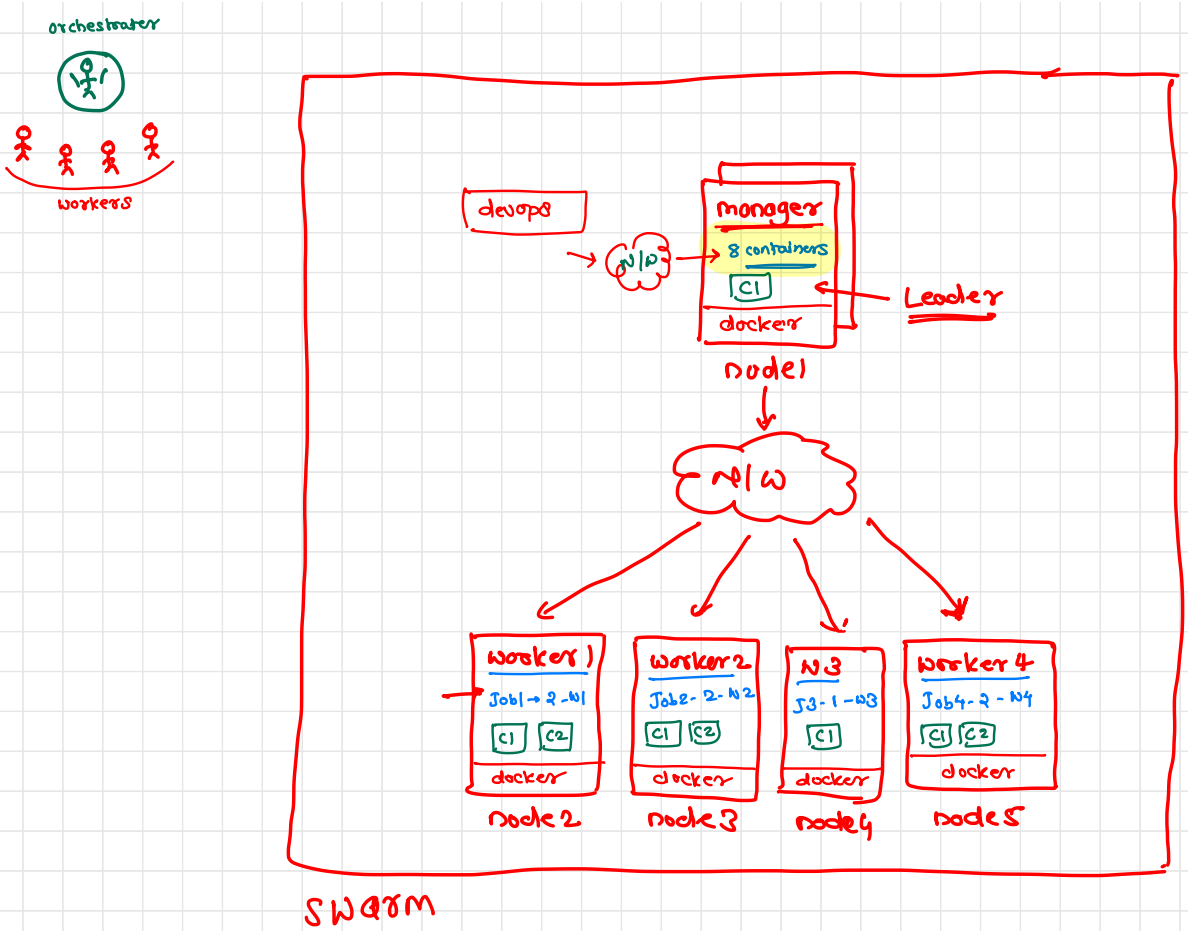
Docker Swarm

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

cluster / docker swarm

↳ frontend | backend | database



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 [No external installation is needed]
- Decentralized design
- Declarative service model
- Scaling
- Desired state reconciliation
- Multi-host networking
- Service discovery
- Load balancing
- Secure by default
- Rolling updates

Service - 100 containers
20% → (80)
(20)

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>

What is Kubernetes ?

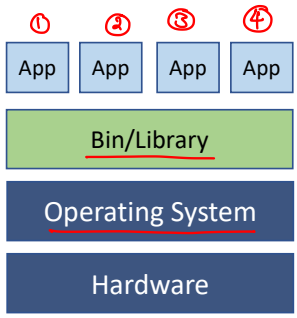
- container orchestration

- 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

declarative configuration
→ xml - x
→ yaml → ✓
→ json - ✓

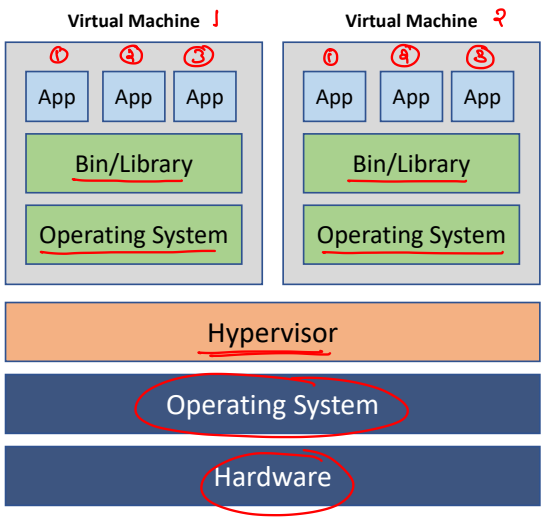
Traditional Deployment

- 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



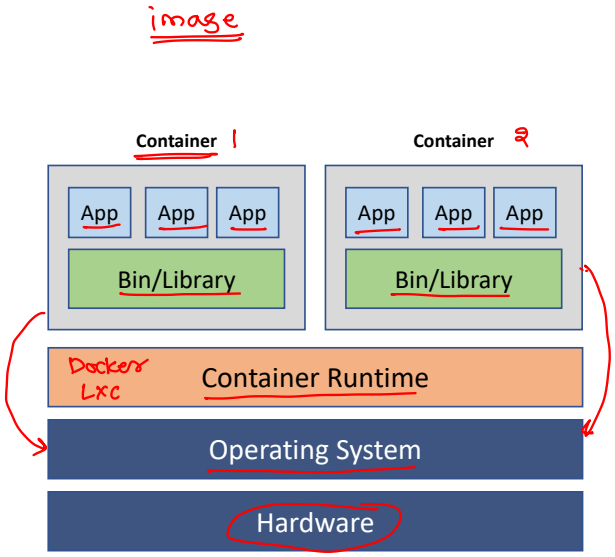
Virtualized Deployment

- 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

- 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?

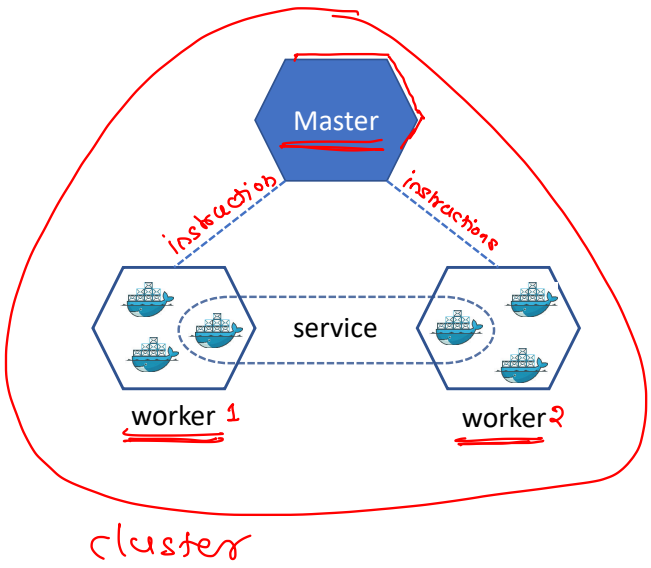
- **Self-healing**
  - Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve
- **Secret and configuration management**
  - Kubernetes lets you store and 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

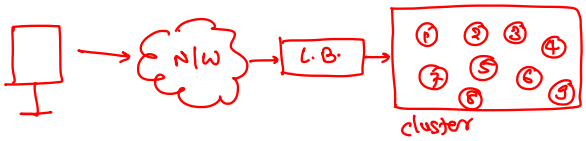
- Does not limit the types of applications supported
- Does not deploy source code and does not build your application
- Does not provide application-level services as built-in services
- Does not dictate logging, monitoring, or alerting solutions
- Does not provide nor mandate a configuration language/system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

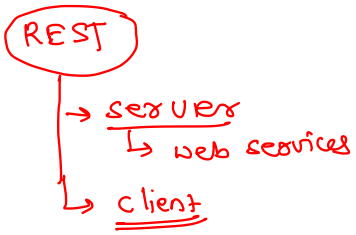
Kubernetes Cluster

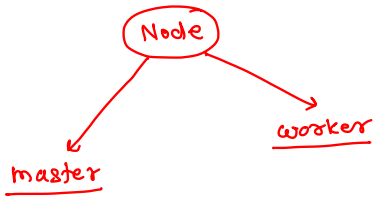
- When you deploy Kubernetes, you get a cluster.
- A cluster is a set of machines (nodes), that run containerized applications managed by Kubernetes
- A cluster has at least one worker node and at least one master node
- The worker node(s) host the pods that are the components of the application
- The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

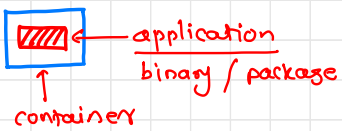


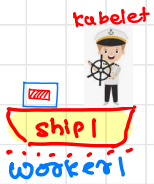
Kubernetes Architecture

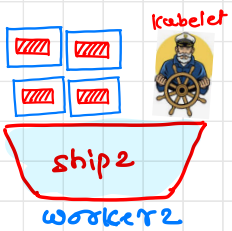


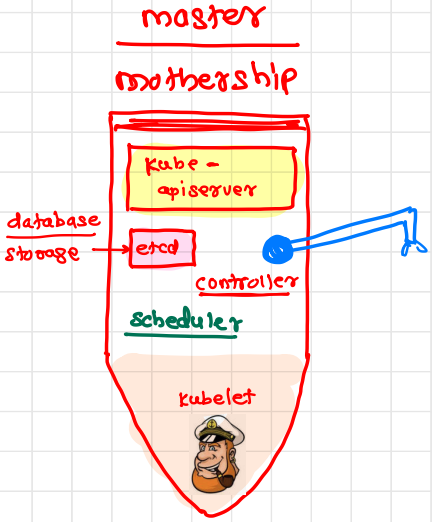


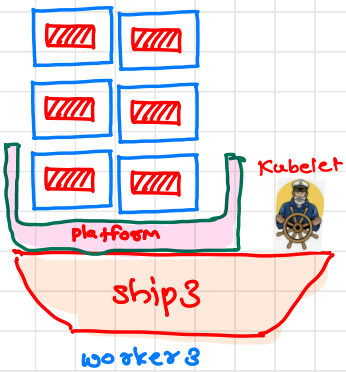




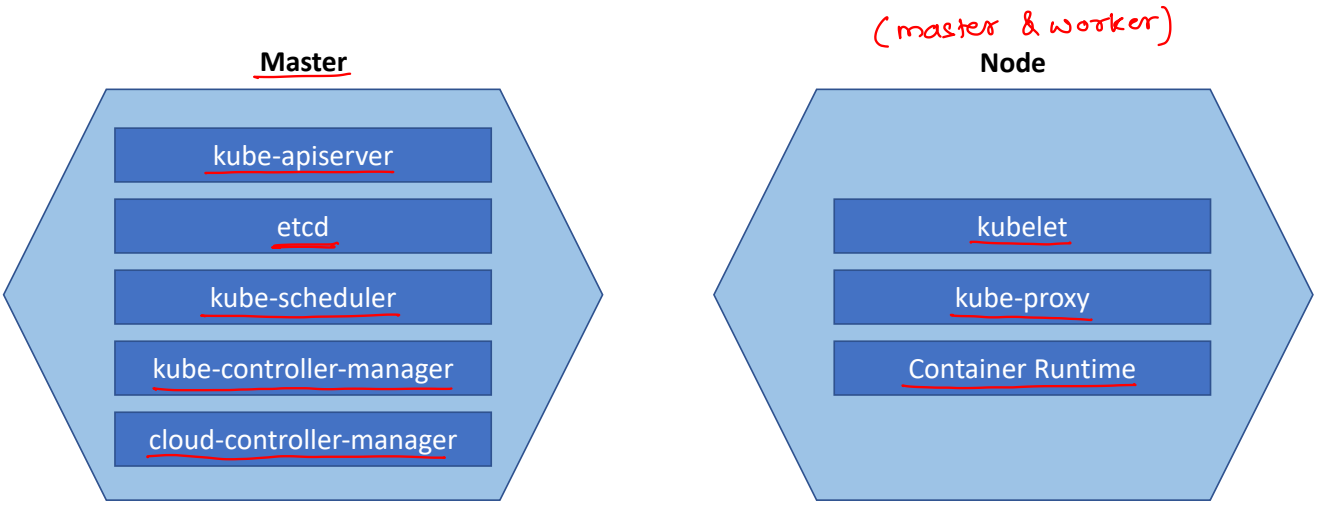








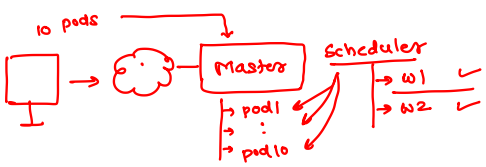
Kubernetes Components



Master Components

- Master components make global decisions about the cluster 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** (database)
   - 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

- **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
- **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
  - including master & worker
- **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** [docker]
  - 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

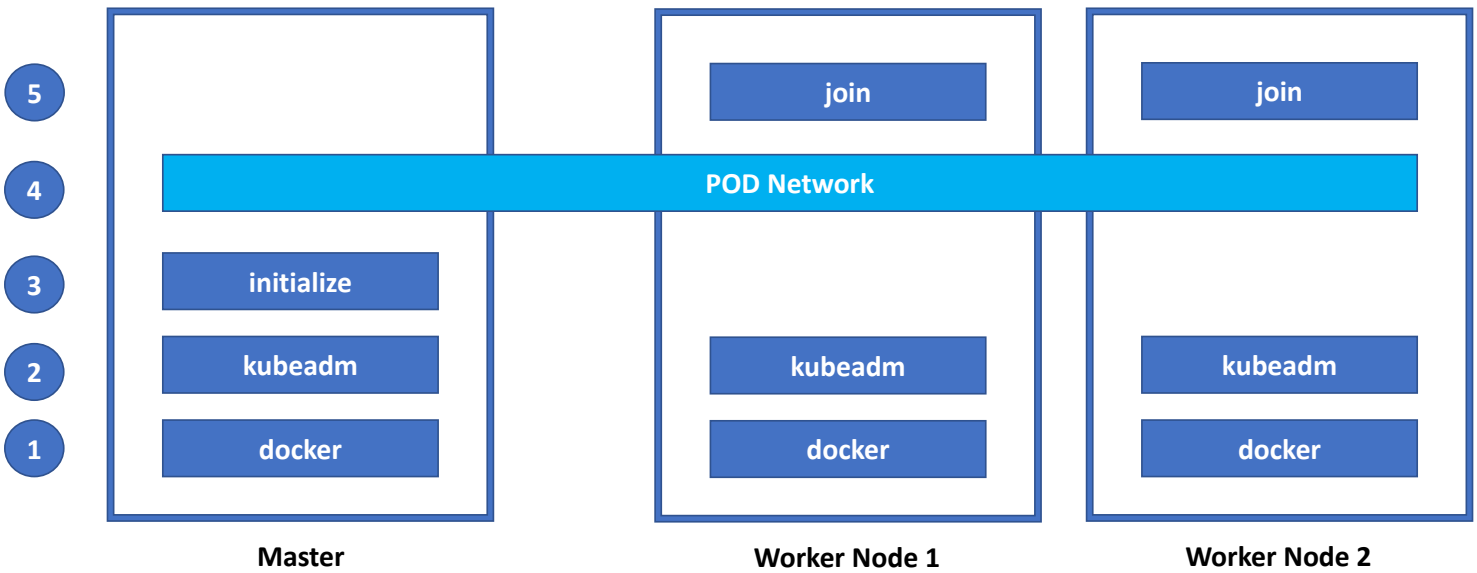
> kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml

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

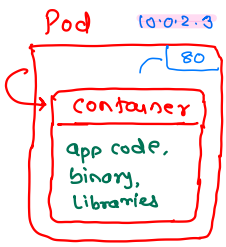


Kubernetes Objects

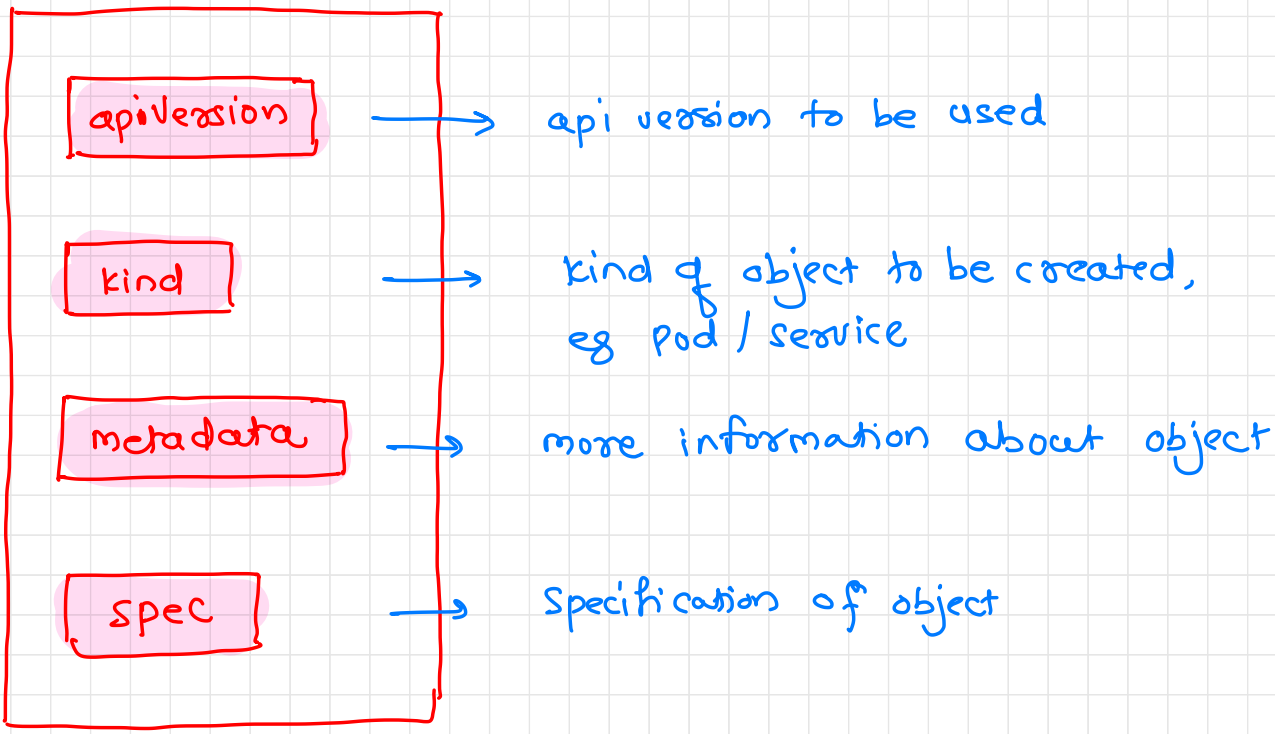
- The basic Kubernetes objects include
  - Pod
  - Service
  - Volume
  - Namespace
- Kubernetes also contains higher-level abstractions build upon the basic objects
  - Deployment
  - DaemonSet
  - StatefulSet
  - ReplicaSet
  - Job

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

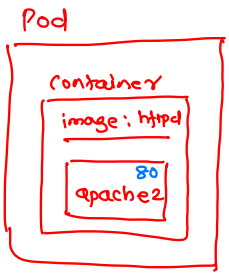


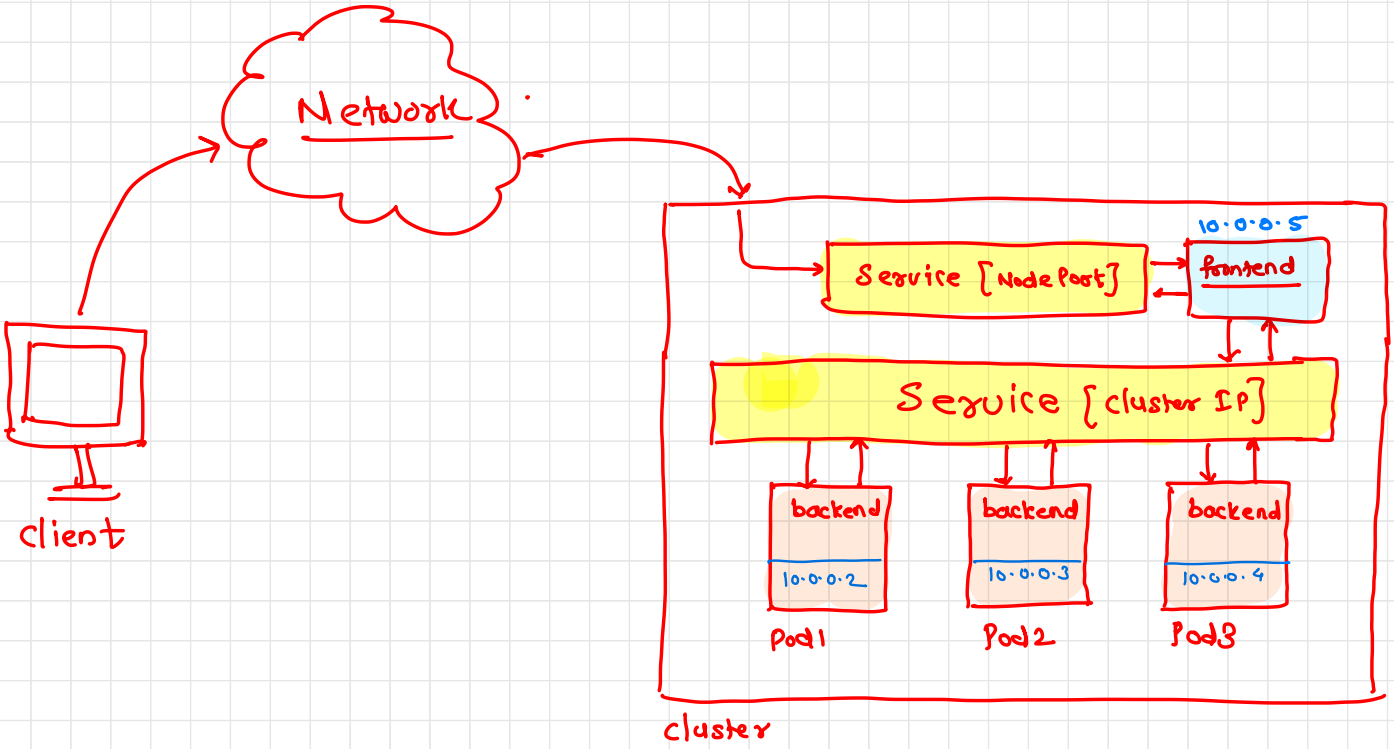
a pod may run one or more containers within it.



YAML to create Pod

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

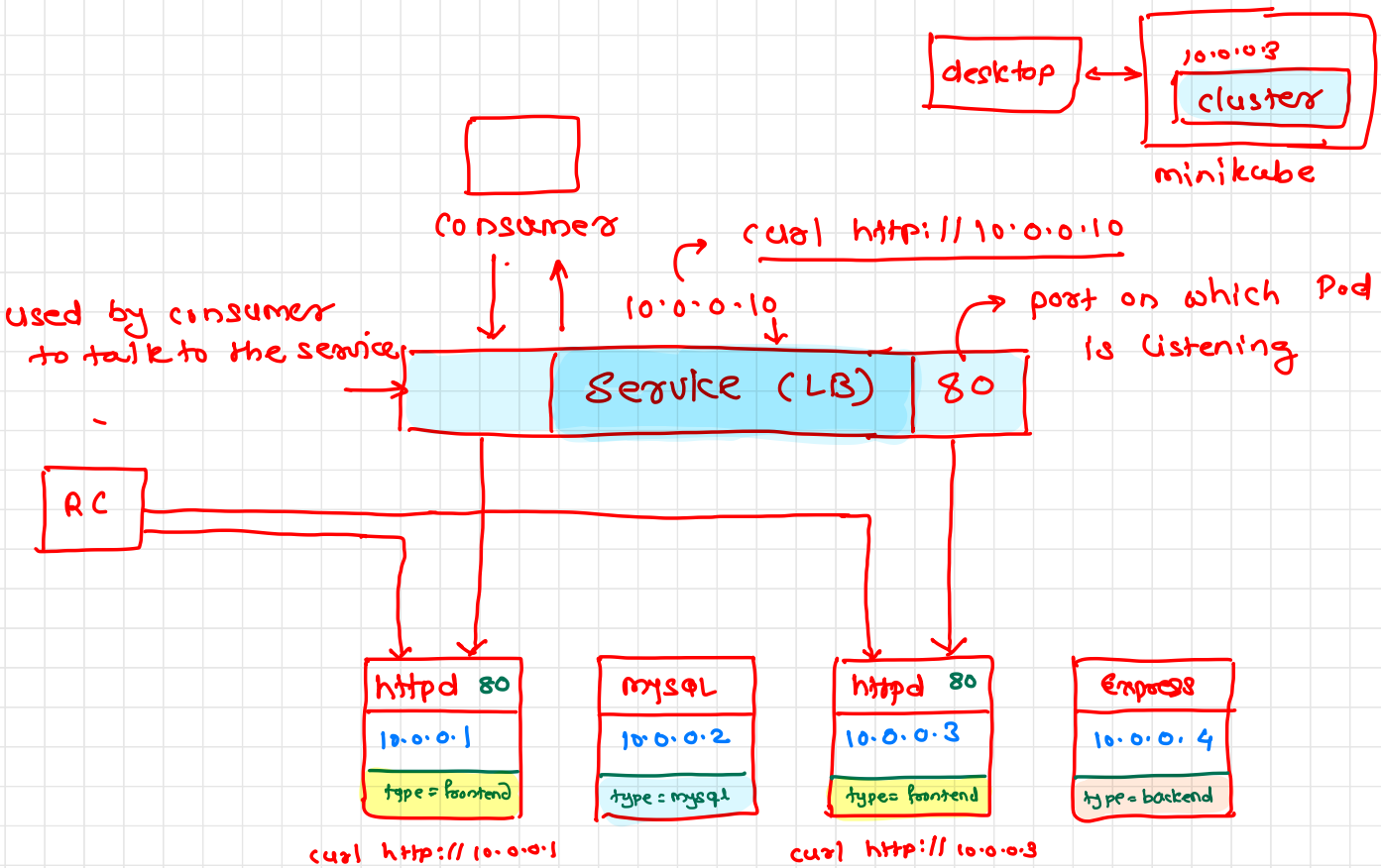


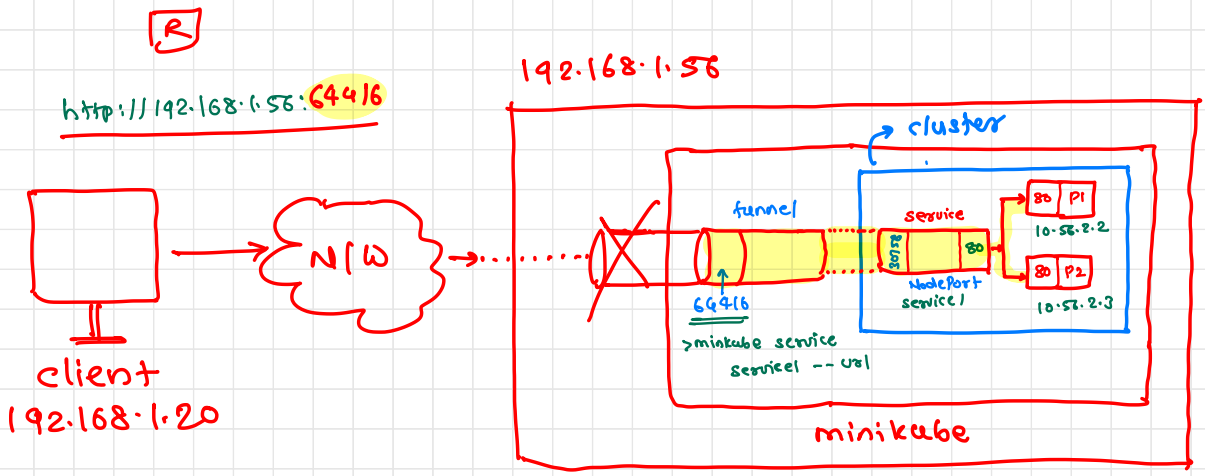


Service

- 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

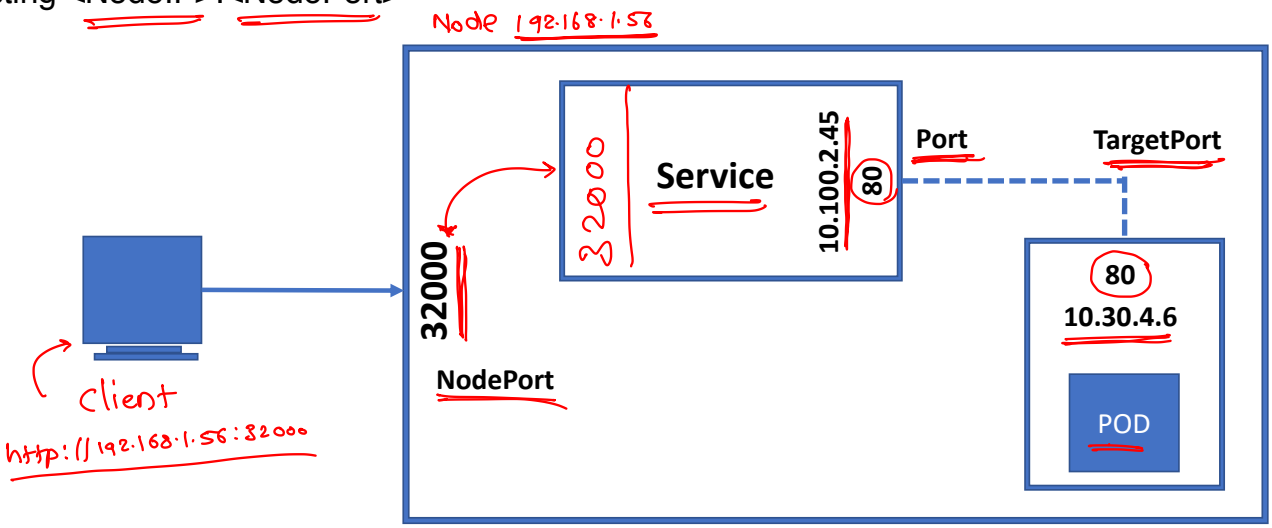
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```


desktop
10.0.0.3
cluster
minikube
consumer
curl http://10.0.0.10
used by consumer to talk to the service
10.0.0.10
port on which Pod is listening
Service (LB)
80
RC
httpd 80
10.0.0.1
type = frontend
mysql
10.0.0.2
type = mysql
httpd 80
10.0.0.3
type= frontend
Express
10.0.0.4
type = backend
curl http://10.0.0.1
curl http://10.0.0.3


R
http://192.168.1.56:64416
192.168.1.56
cluster
NIW
funnel
service
30178
80
NodePort service1
80 P1
10.52.2.2
80 P2
10.52.2.3
64416
>minikube service service1 --url
client
192.168.1.20
minikube

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
```

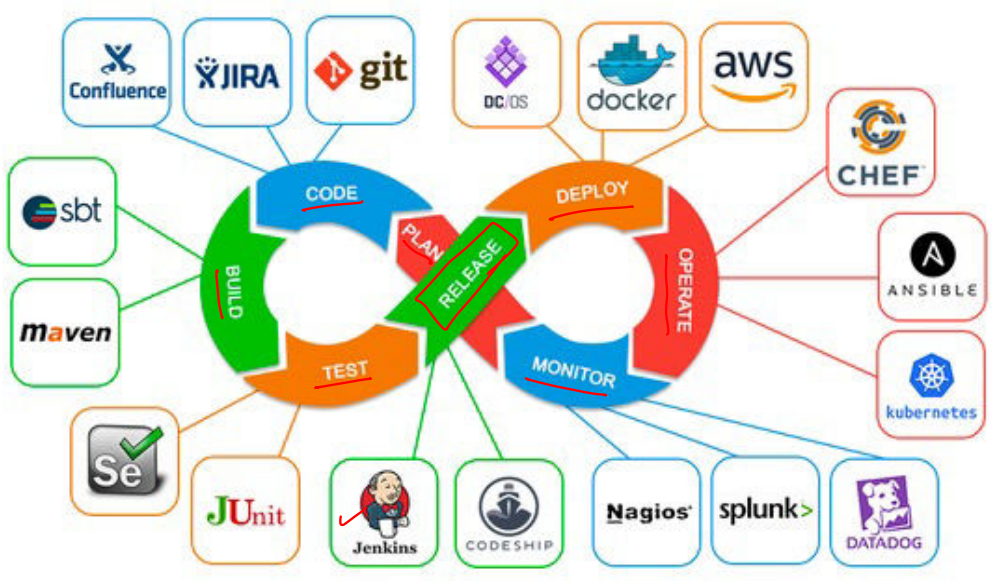

Volume

- 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

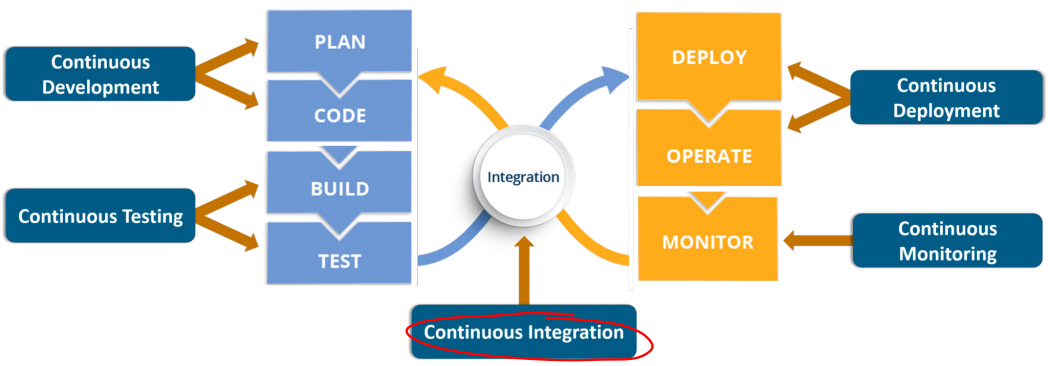
- Namespaces are intended for use in environments with many users spread across multiple teams, or projects
- Namespaces provide a scope for names
- Names of resources need to be unique within a namespace, but not across namespaces
- Namespaces can not be nested inside one another and each Kubernetes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

DevOps Lifecycle



DevOps Terminologies

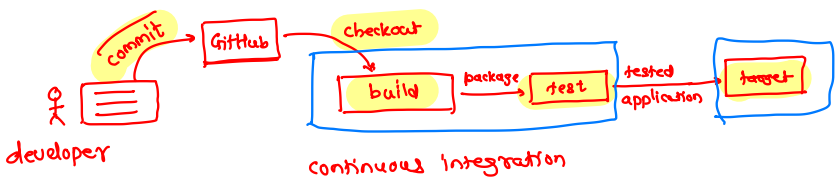
- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Continuous Monitoring



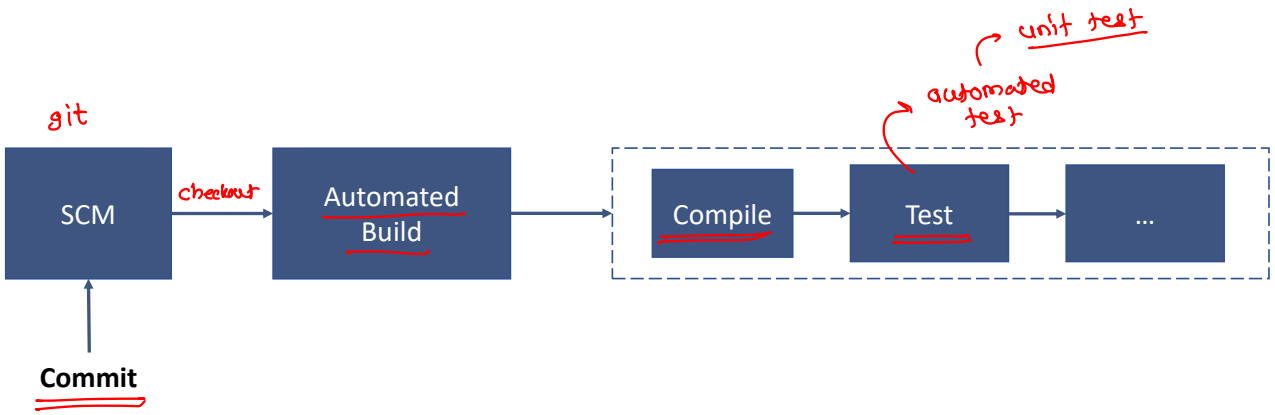
Continuous Integration

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

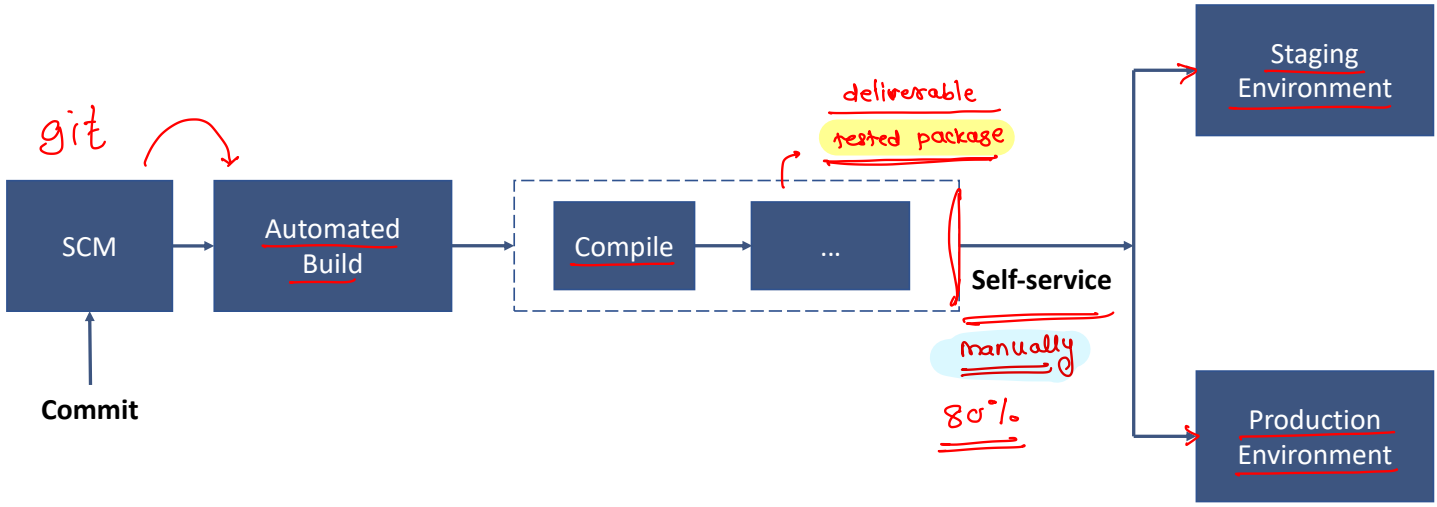


CI Best Practices

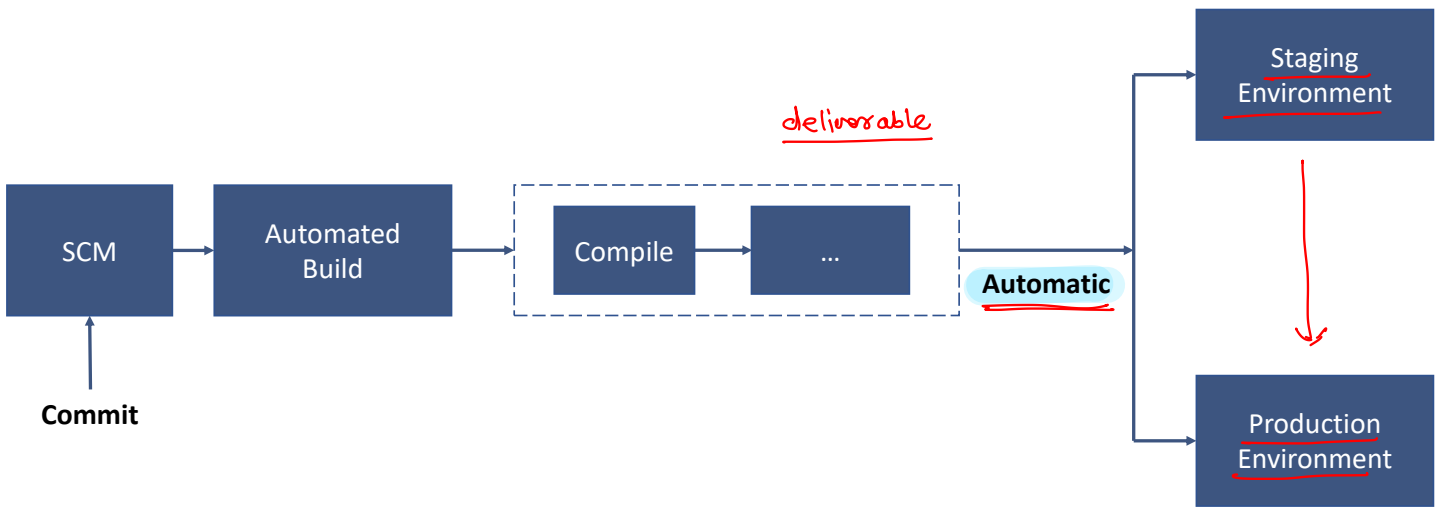
- Frequent commits
- No long running branches
- Automated test execution
- Fix broken builds

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

Continuous Delivery

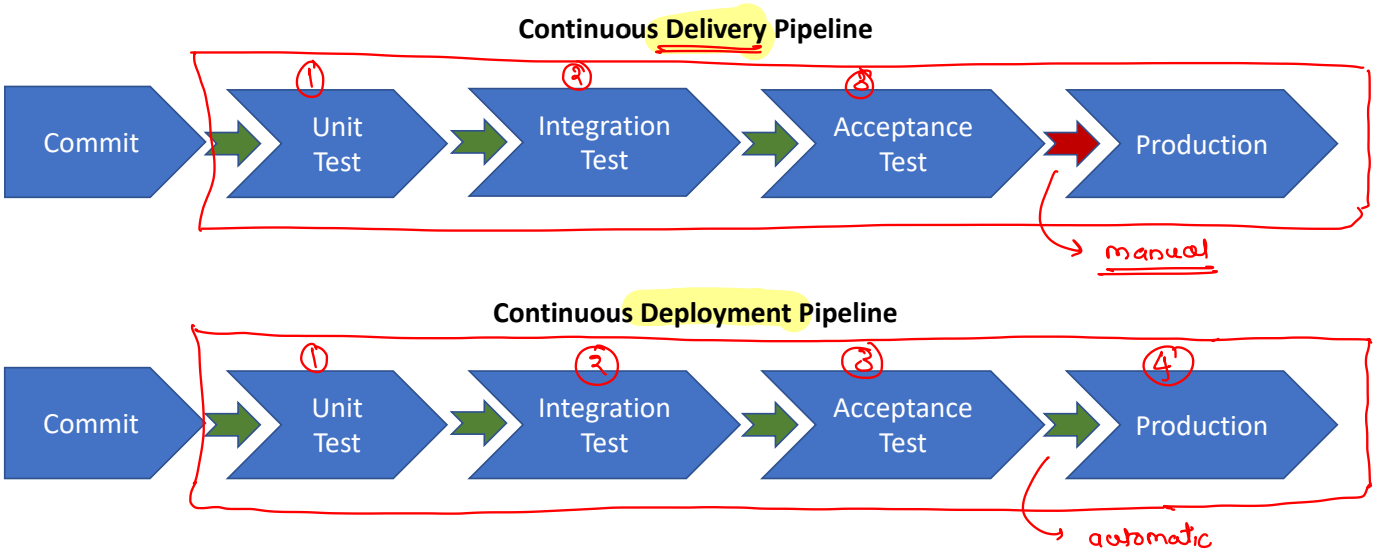


Continuous Deployment



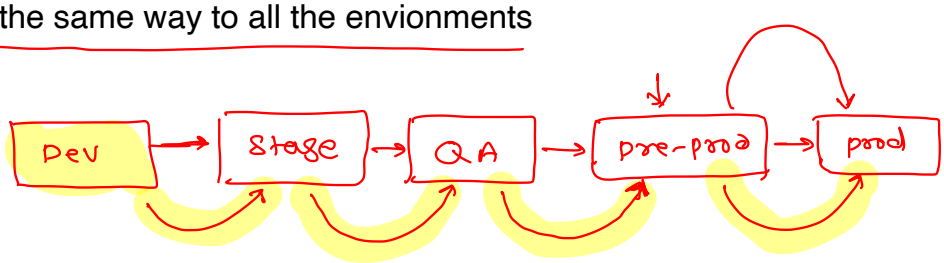
CI/CD Pipeline

pipeline = collection of stages



CD Best Practices

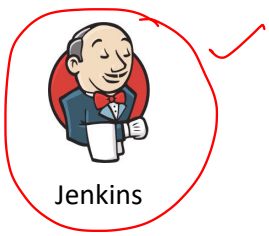
- Version control all configuration
- Stop the line immediately for a failure
- Build your binaries only once
- Deploy the same way to all the envionments



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



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

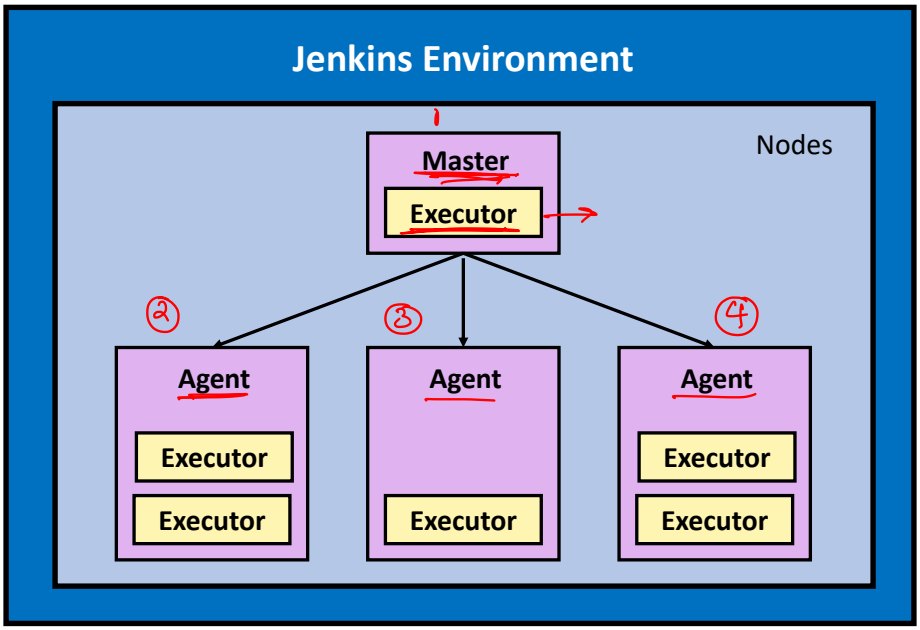
Where is it came from ?

- 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

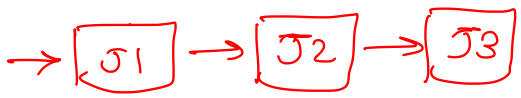


Terminologies

- **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