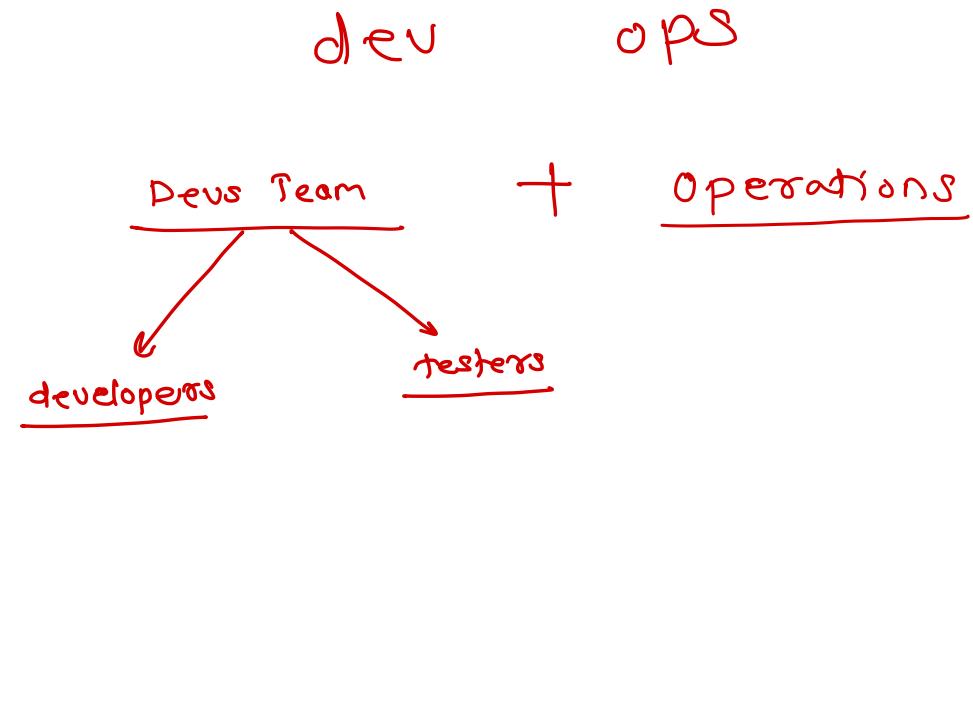
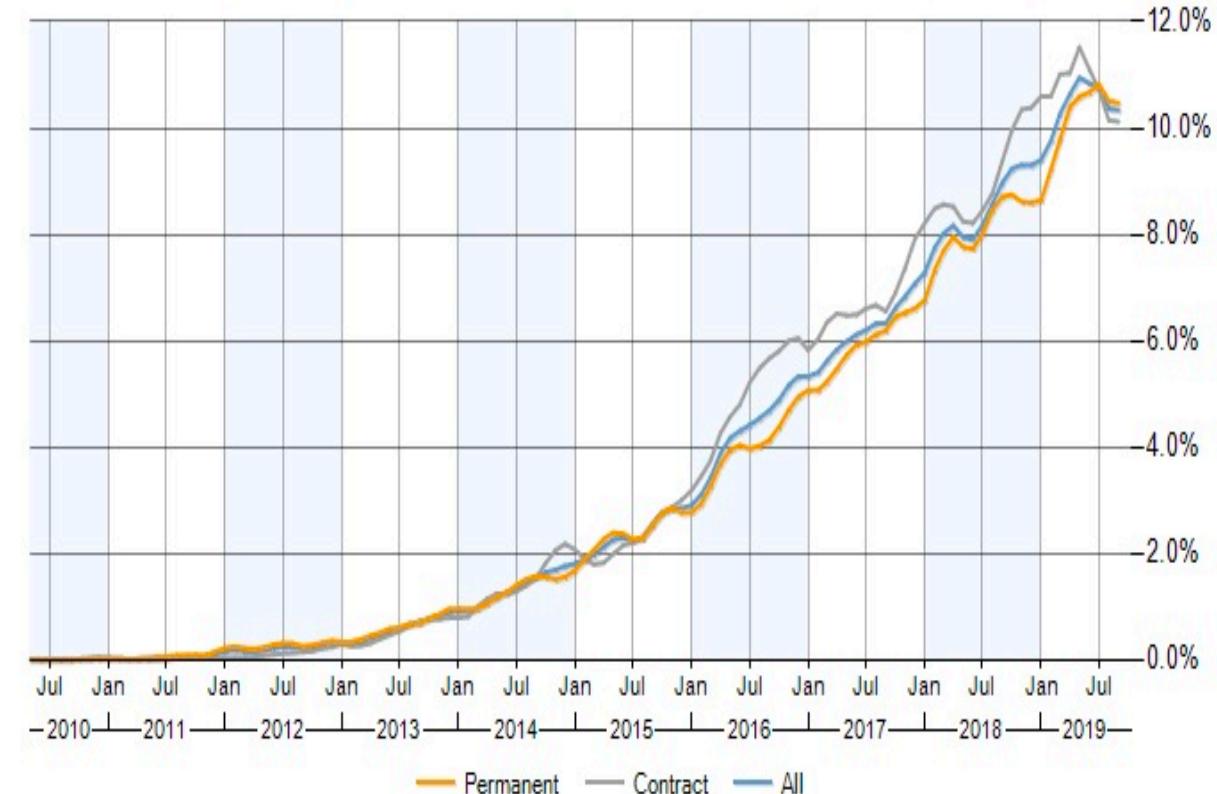
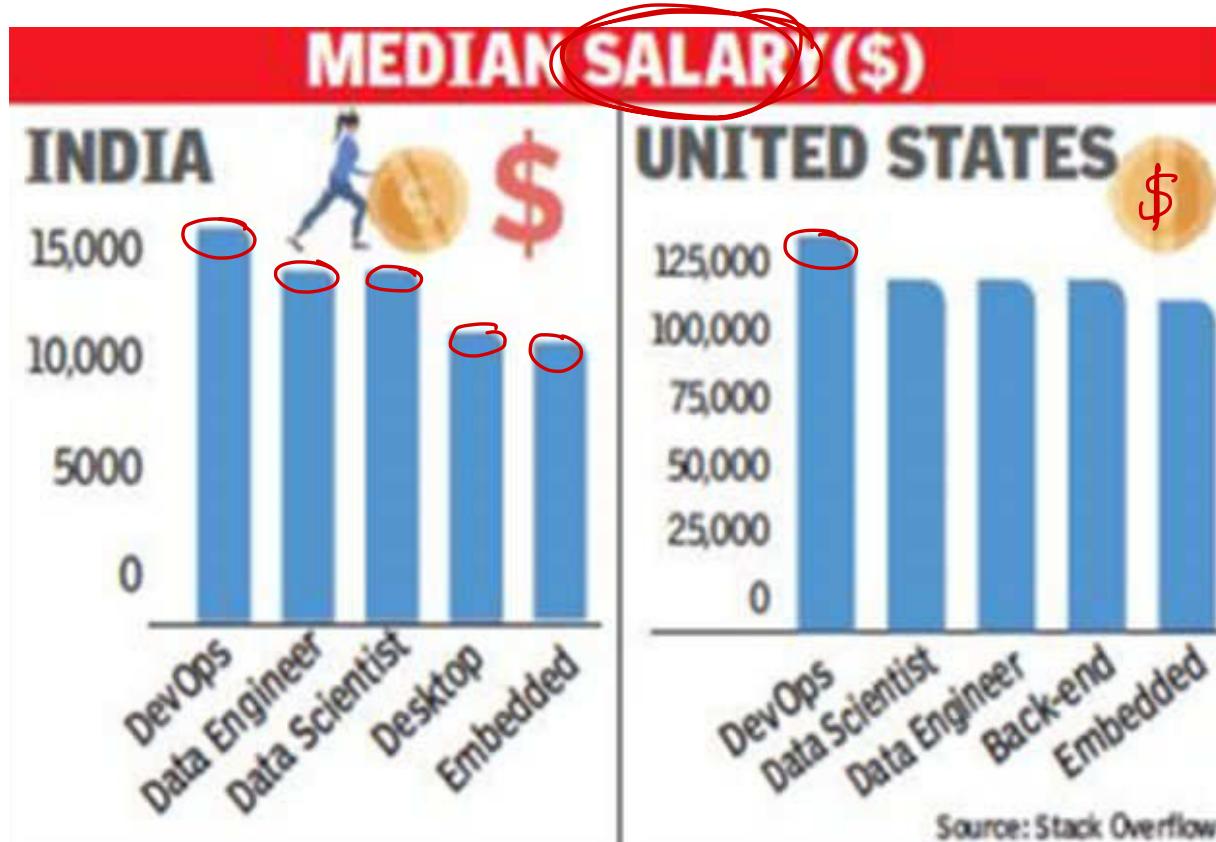


# Contents

- Software Engineering : SDLC + Agile + DevOps
- SDLC VS Agile
- Application Testing : types, methods
- Cloud Computing : deployment
- DevOps
- Source Control Management : git
- Continuous Integration : jenkins
- Containerization : Docker
- Container Orchestration : Docker swarm  
Kubernetes



# Why should you bother about DevOps?





Software Engineering

# 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 : OS, device drivers (closer to the system)
- Application Software : Calendar, calculator, notepad (closer to the user)
- Engineering/Scientific Software : Catia, matlab, autocad
- Embedded Software :
- AI Software
- Legacy Software : old software
- Web/Mobile Software :

# Why SE is important

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

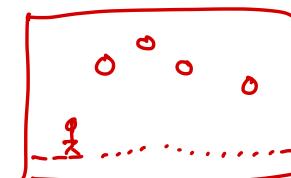
# Software Evolution

## ■ **S-type (static-type)**

- 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)**

- 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)**

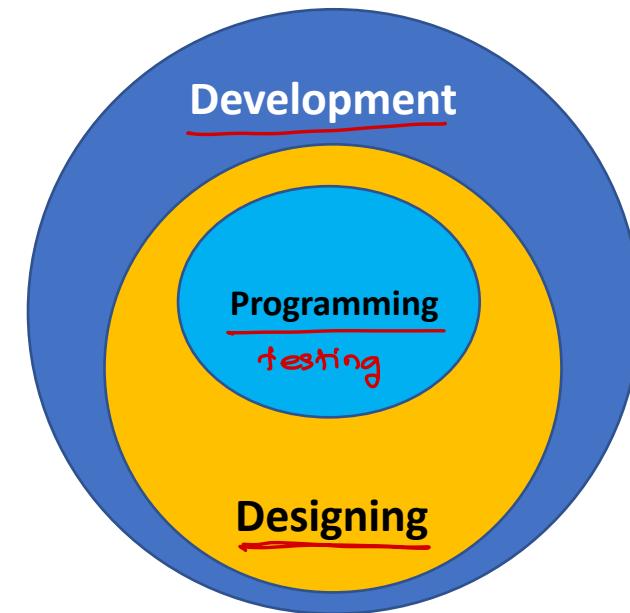
- Works closely as the requirement of real-world environment
- Has a high degree of evolution as there are various changes in laws, taxes etc. in the real world
- E.g. online trading software

# 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**
  - 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 (time & money)
- Usability
- Efficiently
- Correctness
- Functionality
- Dependability
- Security \*
- Safety

## Transitional

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

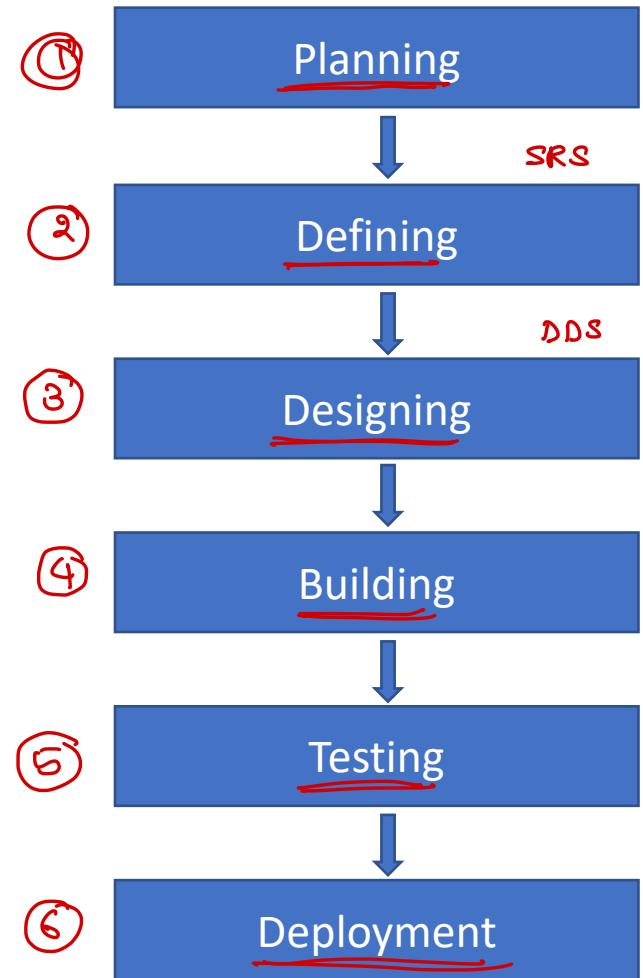
# Maintenance

- 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 : cloud / containerization
    - vertical
    - horizontal

**SDLC**

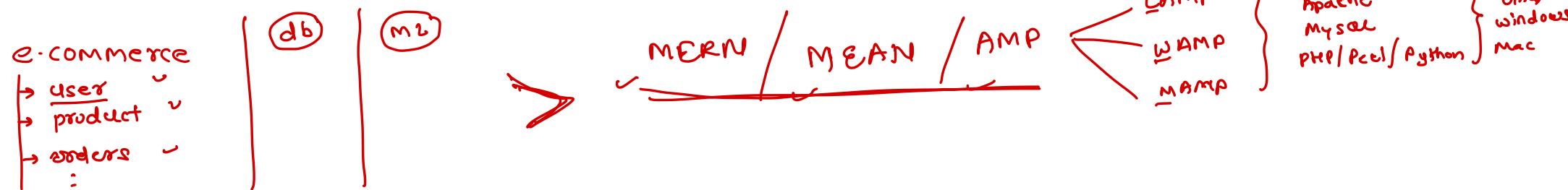
# Overview

- 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

## 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

## 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 [ android: play store , ios: app store , website: server(cloud) ]
- Sometimes product deployment happens in stages as per the business strategy of that organization ↗(internal / UAT)
- 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

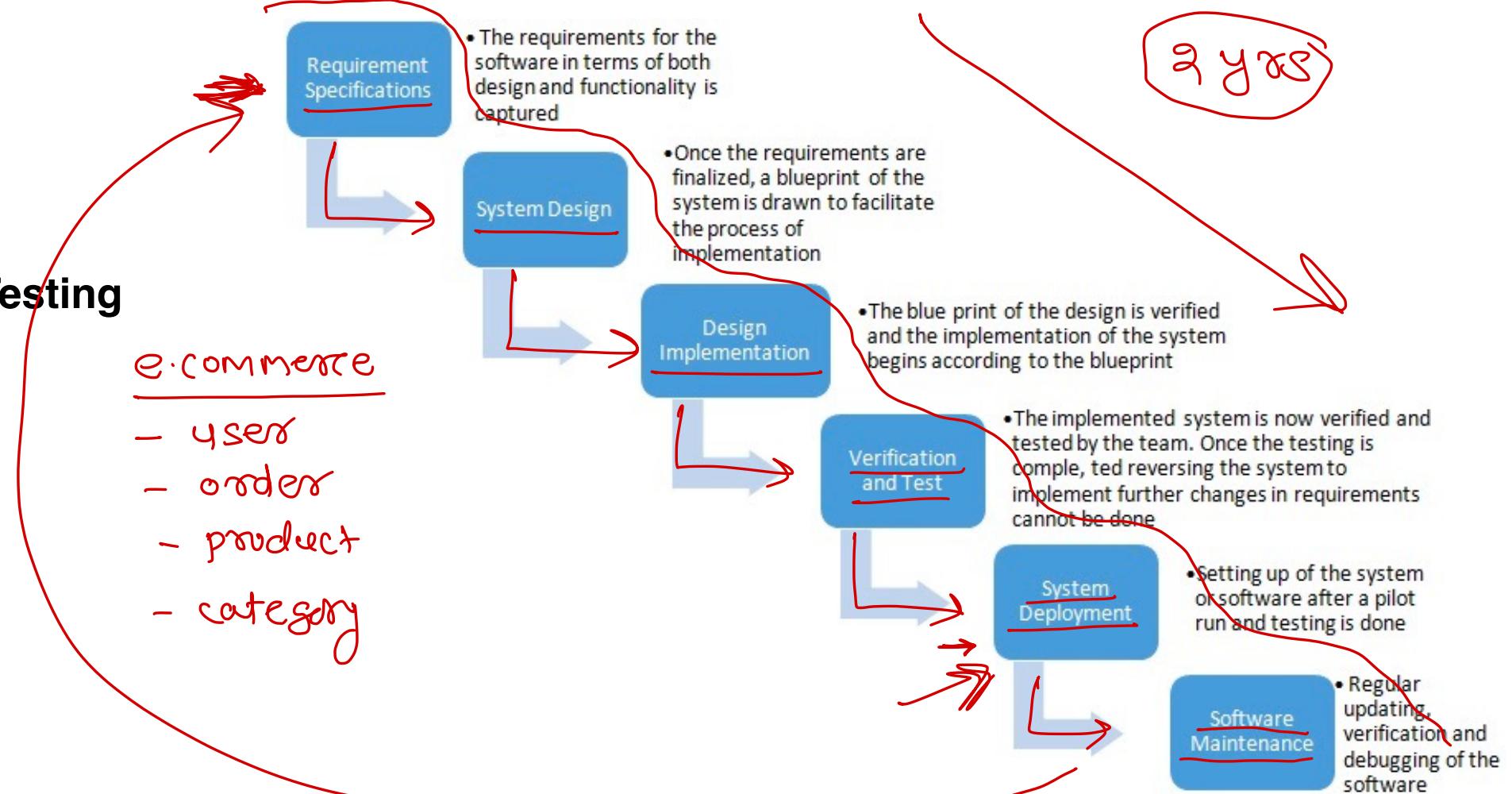
# SDLC Models

---

- 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

# Waterfall Model

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



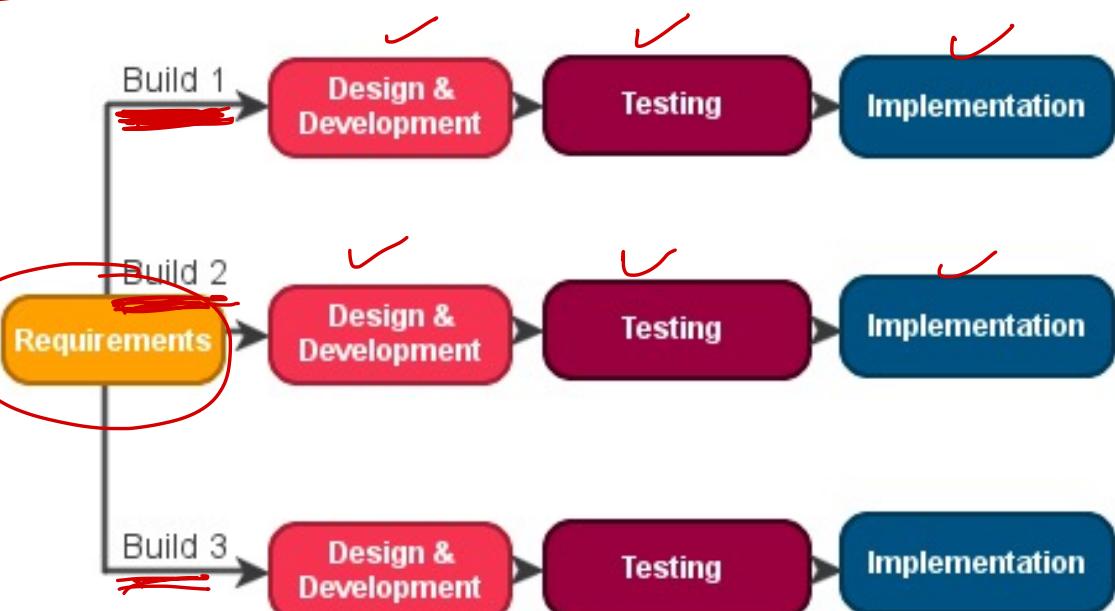
# Iterative Model

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

# e-commerce

- ✓ - user → UI - backend - DB
  - ✓ - order → ↗ ↗ ↗
  - ✓ - product → ↗ ↗ ↗
  - category → ↗ ↗ ↗

2 months



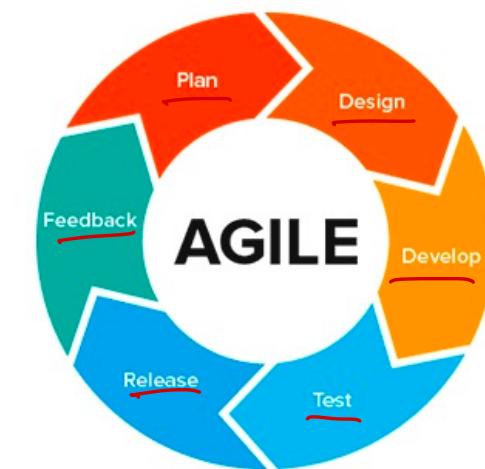
# Agile Methodologies



# Agile Methodologies

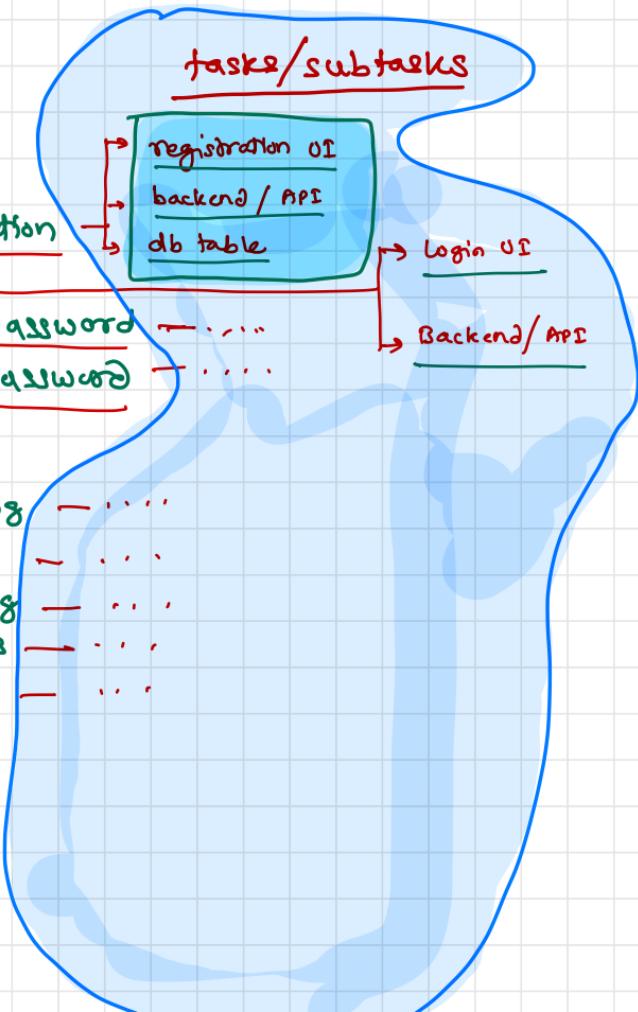
- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements
- The tasks are divided to time boxes (small time frames) to deliver specific features for a release
- Iterative approach is taken and working software 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

Sprint



## Blogging

- user
  - registration
  - login
  - forget password
  - reset password
- blogging
  - create blog
  - edit blog
  - delete blog
  - comments
  - rating
  - ...  
.  
.  
.



time box event = 2 weeks → Sprint

1] Sprint 1 : 2 weeks

→ registration UI

→ backend API

- table

} → build 1 ✓

+

2] Sprint 2 : 2 weeks

- login API

- login UI

} → build 2

3]

:  
:  
:  
:

# 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)
  - Kanban

# What is Scrum ?

- 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 *sprint*
- 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 *(e-type)*
- The framework is made up of a Scrum team, individual roles, events, artefacts, and rules

Jira

# Scrum Principles

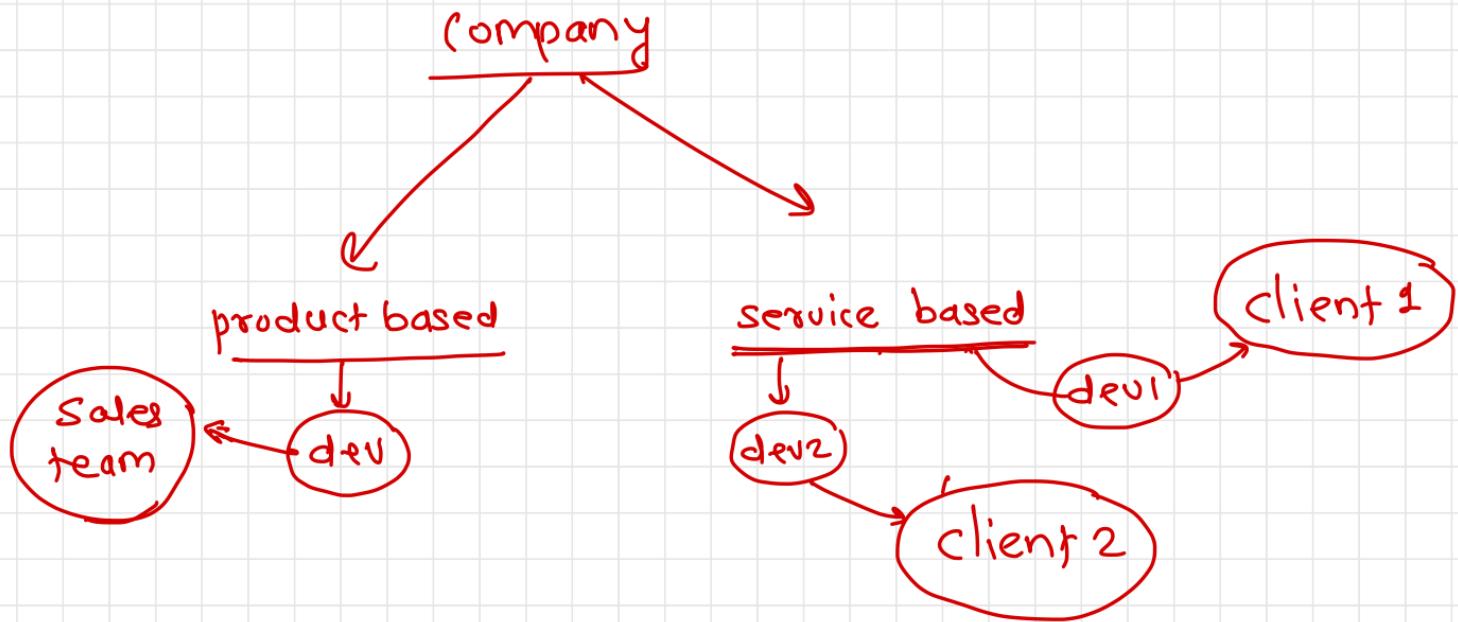
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business 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

daily stand up

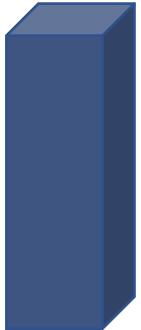
# Scrum Principles

- 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

→ customer



# Scrum Pillars



Transparency

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

dev + tester + ops + client



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

## 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



- 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



- 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



- 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

:   $\rightarrow$  o/p of process

### Product Backlog

collection of all tasks/  
subtasks / stories

### Sprint Backlog

collection of tasks  
selected for a sprint

### Increment

# Scrum Events

## Sprint Planning

- Happens at the start of every sprint
- it should probably be about between four and eight hours

Stand up

## 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

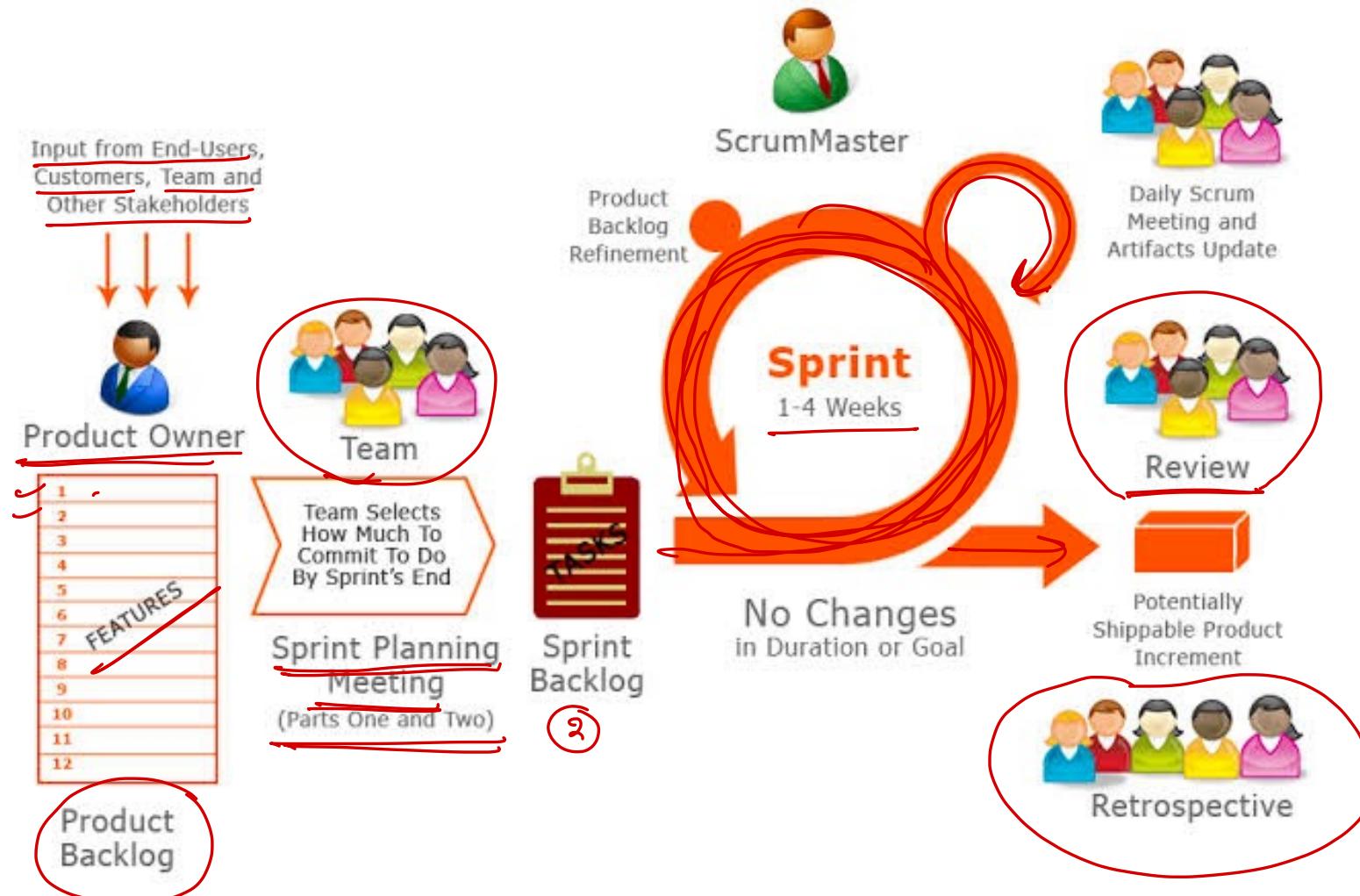
## Sprint Retrospective

- 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	<u>Incremental value and risk management</u>	<u>Phased approach with an attempt to know everything at the start</u>
2	<u>Embracing change</u>	<u>Change prevention</u>
3	<u>Deliver early, fail early</u>	<u>Deliver at the end, fail at the end</u>
4	<u>Transparency</u>	<u>Detailed planning, stagnant control</u>
5	<u>Inspect and adapt</u>	<u>Meta solutions, tightly controlled procedures and final answers</u>
6	<u>Self managed</u>	<u>Command and control</u>
7	<u>Continual learning</u>	<u>Learning is secondary to the pressure of delivery</u>

## 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/>
- Quicksrum – <https://www.quicksrum.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

---

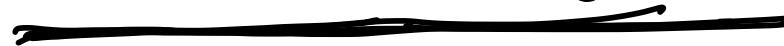
- **Sprint:** time-boxed event of 30 days, or less, that serves as a container for the other Scrum events and activities.
- **Sprint Backlog:** an overview of the development work to realize a Sprint's goal, typically a forecast of functionality and the work needed to deliver that functionality.
- **Sprint 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

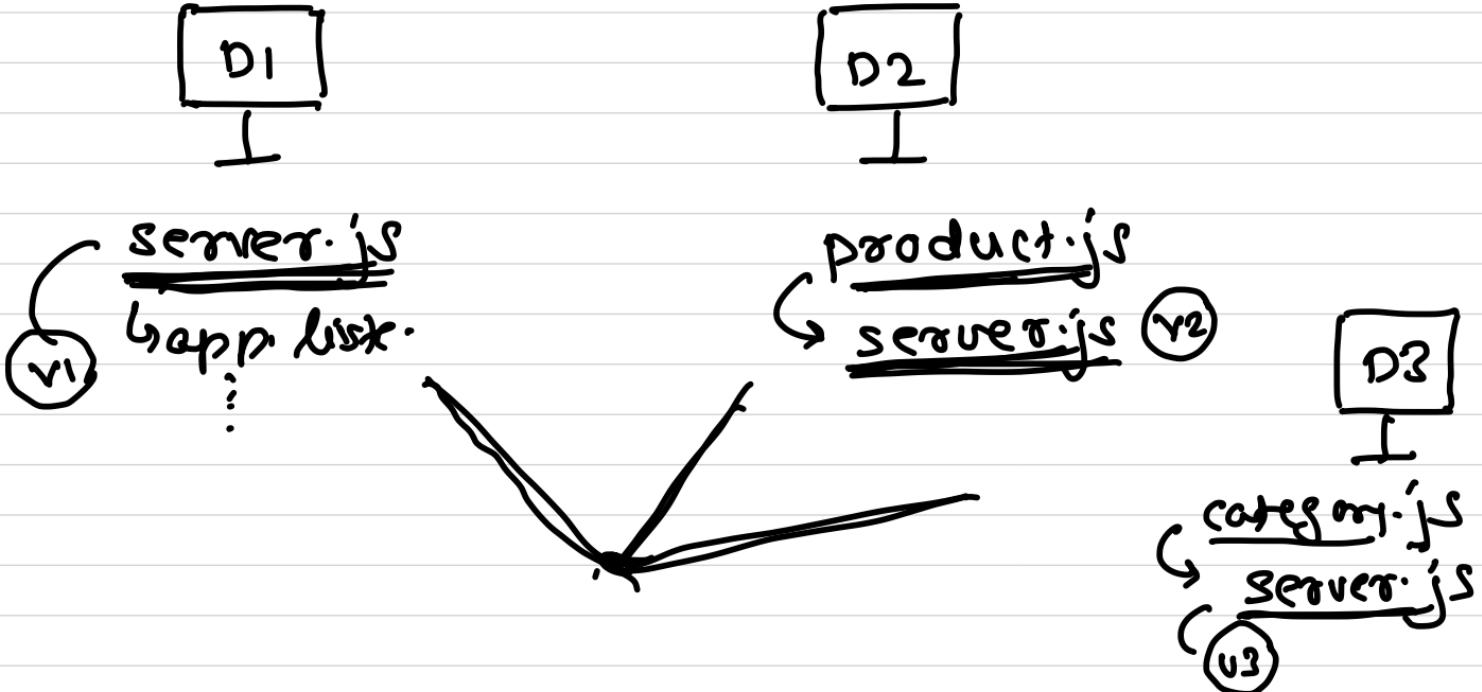
# Agile Methodologies – Scrum Terminologies

---

- **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

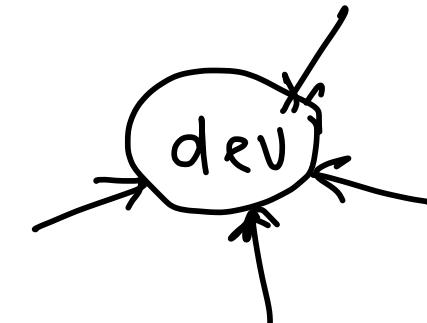
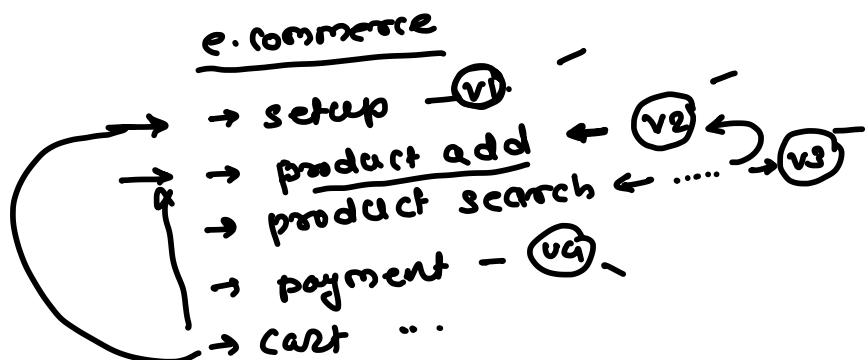
Source Code Management





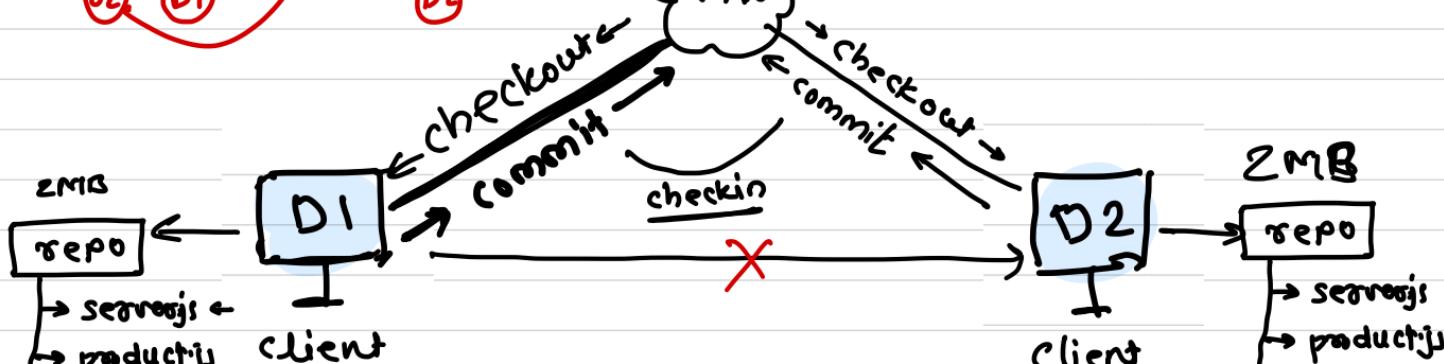
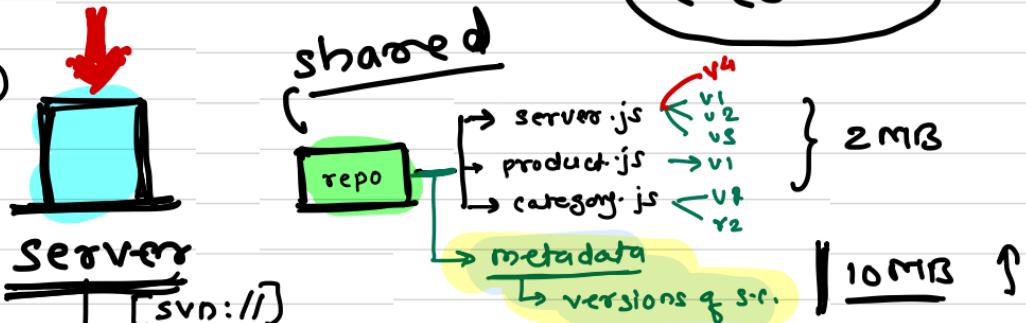
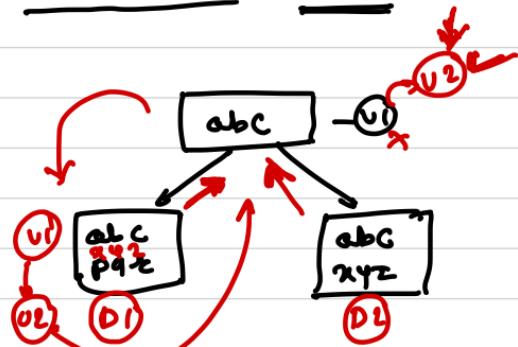
# Version Control System

- Also known as revision control or source code control system 
- Is the management of documents (source code) → team of developers
- Logical way to organize and control the revisions of source code
- Tracks and provides control over the changes made in the code
- E.g.
  - CVS ←
  - SVN ←
  - **Git** ← (highlighted)
  - Bazaar ←



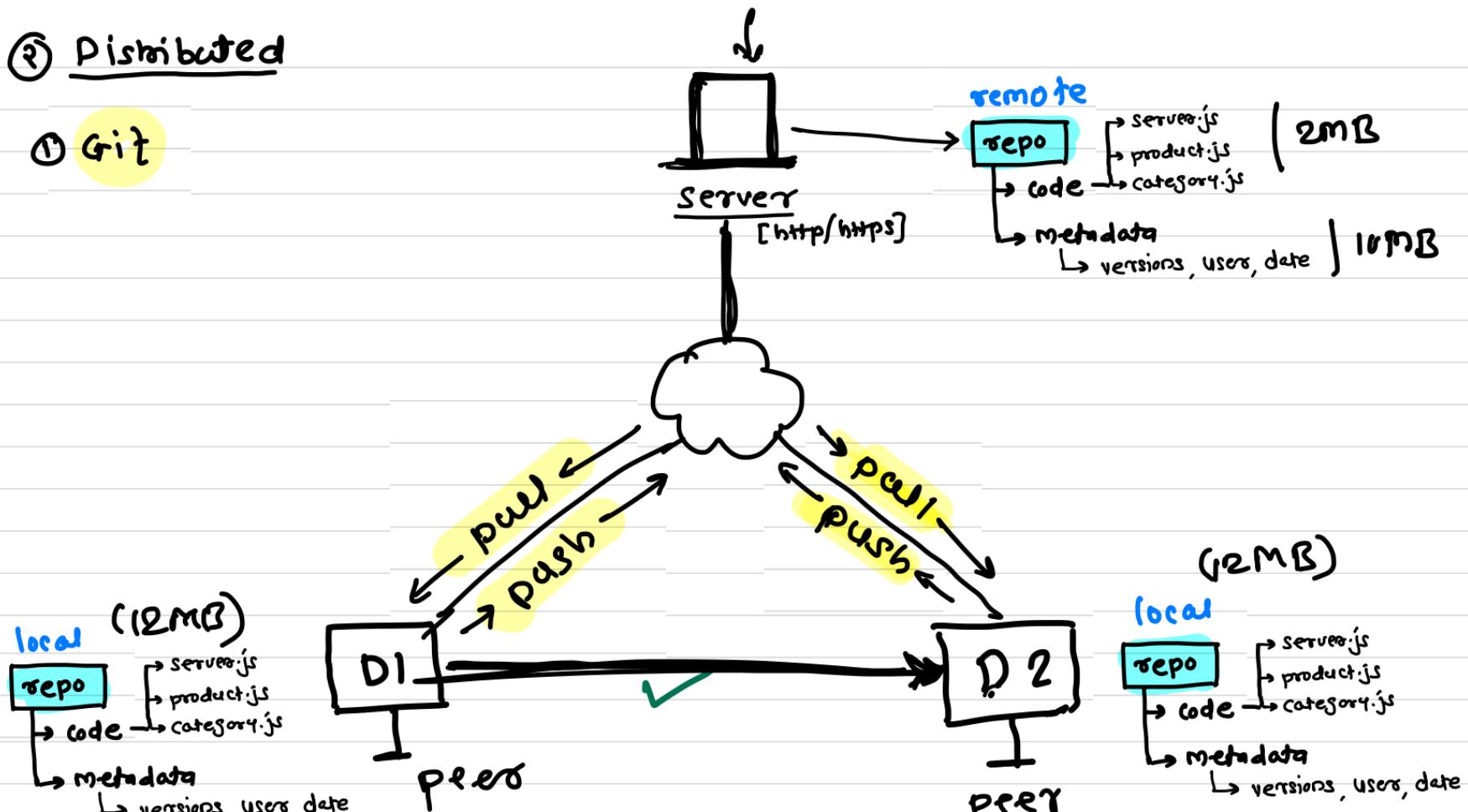
# ① Client - Server → SCM

- ① code version system (CVS)
- ③ Subversion → SVN



## ② Distributed

### ① Git



# VCS Types – Client-Server

---

- Only one server maintains shared repository
- Every developer sends the changes to the same repository
- Disadvantages
  - not scalable ←
  - dependency on the server ←
- E.g.
  - Open source
    - ✓ CVS (Concurrent Version System)
    - ✓ SVN (Subversion)
  - Proprietary
    - ✓ AccuRev
    - ✓ Razor
    - ✓ TeamCity
    - ✓ Vault —
    - ✓ Visual SourceSafe

## VCS Types – Distributed

- Takes peer-to-peer approach to version control
  - Synchronizes repositories by exchanging patches from peer to peer
  - There is no single server which maintains the code, rather user has a working copy and full change history
  - DisAdvantages
    - Allows users to work productively even when not connected to internet
    - Common operations like commit, version history etc. are faster because there is no need to communicate with server
    - Communication with server is necessary only when developer wants to share the changes with others
    - Allows private work, users don't need to publish the changes for early drafts
    - Working copies function effectively as backups → rep
    - Permits centralized control of the release version of code
- master branch
- branches

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

↳ GPL 2

## History

- The development of Git began on 3 April 2005 ↪
- Torvalds announced the project on 6 April
- It became self-hosting as of 7 April
- The first merge of multiple branches took place on 18 April ↪
- Torvalds achieved his performance goals on 29 April
- On 16 June Git managed the kernel 2.6.12 release
- Torvalds turned over maintenance on 26 July 2005 to Junio Hamano, a major contributor to the project

# Characteristics

- Strong support for non-linear development → branch
- Distributed development
- Compatibility with existent systems and protocols → http | https
- Efficient handling of large projects
- Cryptographic authentication of history
- Toolkit-based design → tools
- Pluggable merge strategies  
↳ branches

# Advantages

---

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

# Terminologies

- **Repository**
    - Directory containing .git folder
  - **Object**
    - Collection of key-value pairs
  - **Blobs (Binary Large Object)**
    - 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 → once**
    - 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
- version  
→ date & time  
→ author  
→ what changes ?

# Terminologies

- Pull      **remote → local**

- 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      **local → remote**

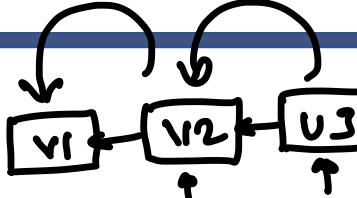
- Push operation copies changes from a local repository instance to a remote
- This is used to store the changes permanently into the Git repository

- 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

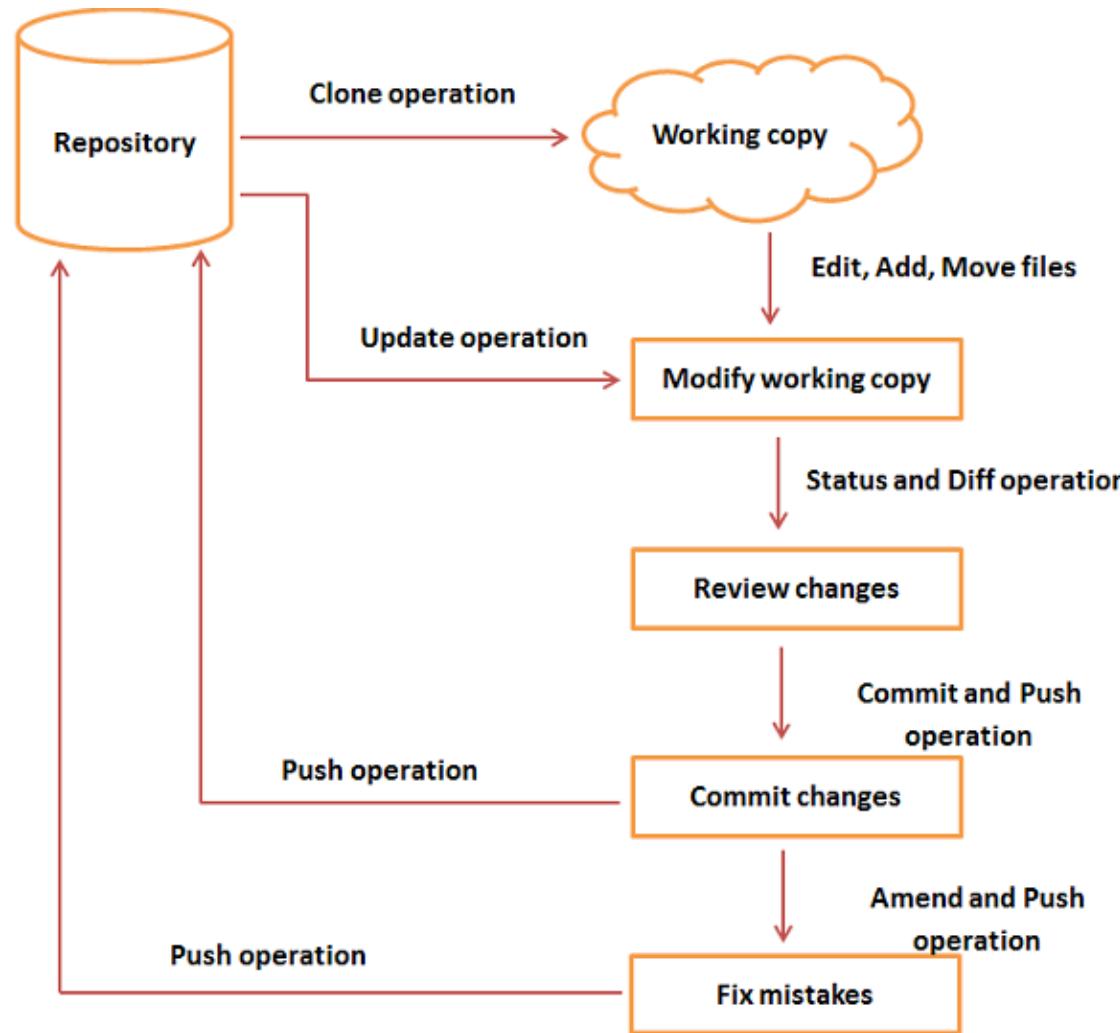
## Commits

- 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

---

- 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

## Why it is required ?

---

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

# Branch management commands

---

- **Create a branch**

- > git branch <branch name>

- **Checkout a branch**

- > git checkout <branch name>

- **Merge a branch**

- > git merge <branch name>

- **Delete a branch**

- > git branch -d <branch name>

# GitHub

# Overview

---

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

# Workflow

---

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

# 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

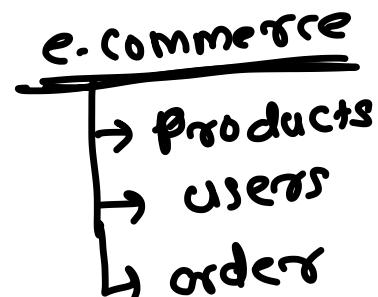


## Software Engineering

---

# Introduction

- Software → to solve a problem = exe + libraries + documentation  
▪ 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 ; requirements , design, test plan  
▪ 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 Evolution

## S-type (static-type)

- 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

↓

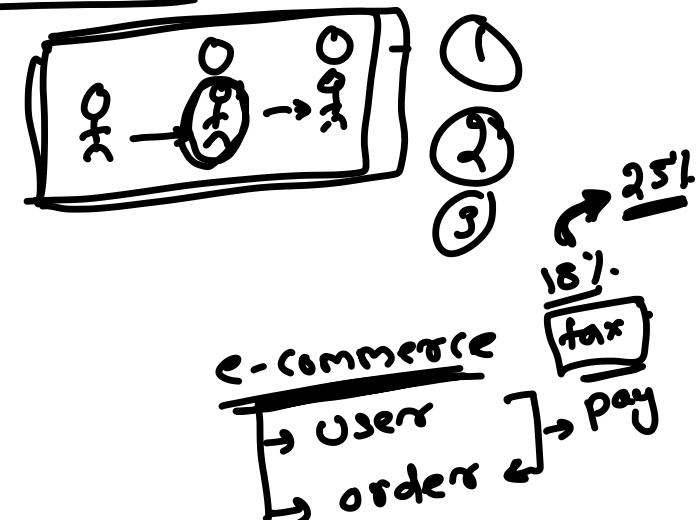
$$10 + 20 = 30$$

## P-type (practical-type)

- 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) → changes

- 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 — 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 ← Agile

- 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.

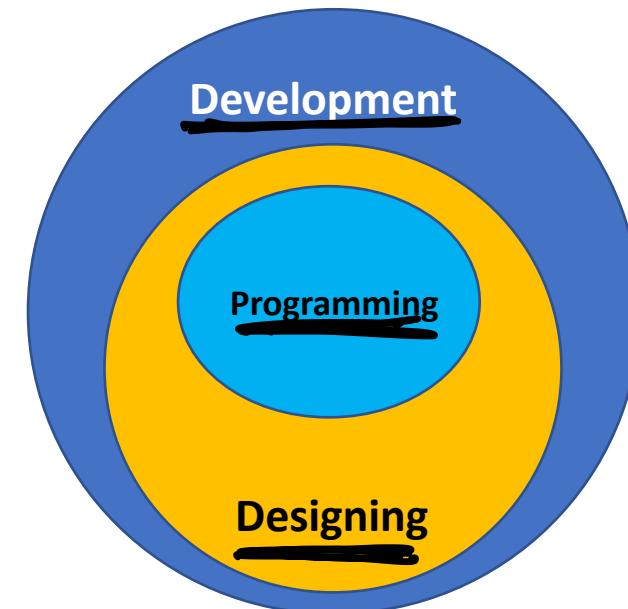
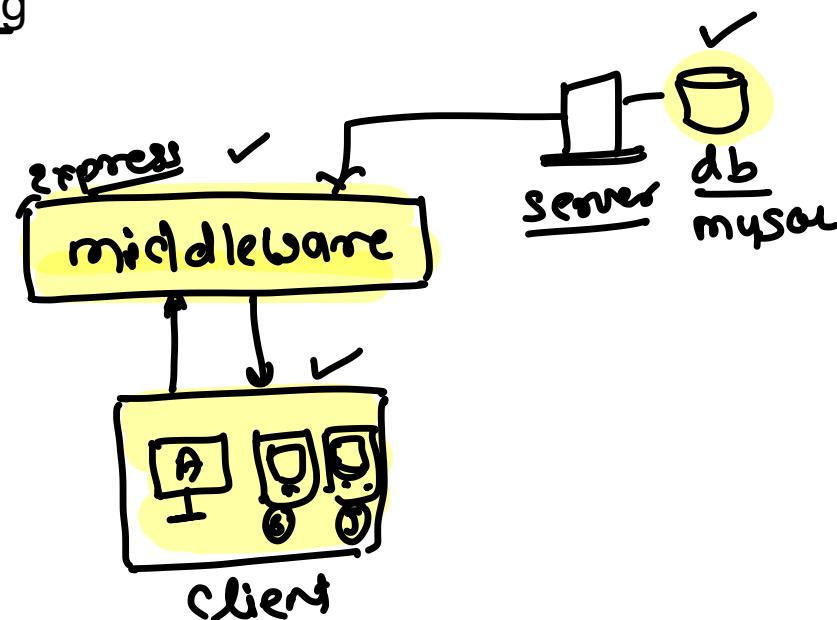
# Software Paradigms

- 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

- coding
    - testing



## 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
  - ✓ Operational
  - ✓ Transactional
  - ✓ Maintenance

# Operational

- This tells us how well software works in operations

- Can be measured on:

- ✓ Budget
- ✓ Usability ←
- ✓ Efficiently
- ✓ Correctness ←
- ✓ Functionality ←
- ✓ Dependability ←
- ✓ Security
- ✓ Safety



## Transitional

---

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

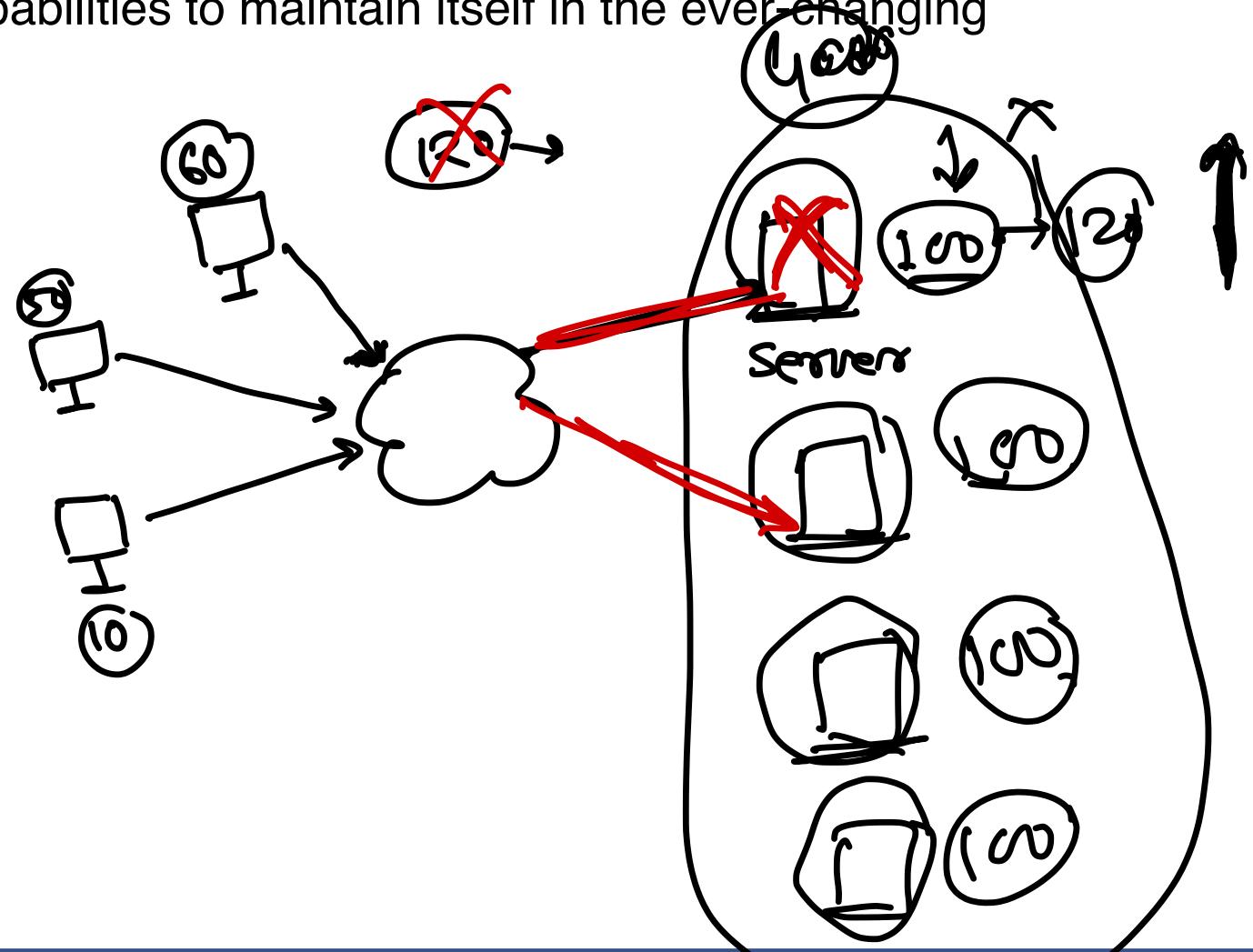
# Maintenance

- 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

→ Vertical  
→ upgrade the server

→ Horizontal  
→ cloning servers

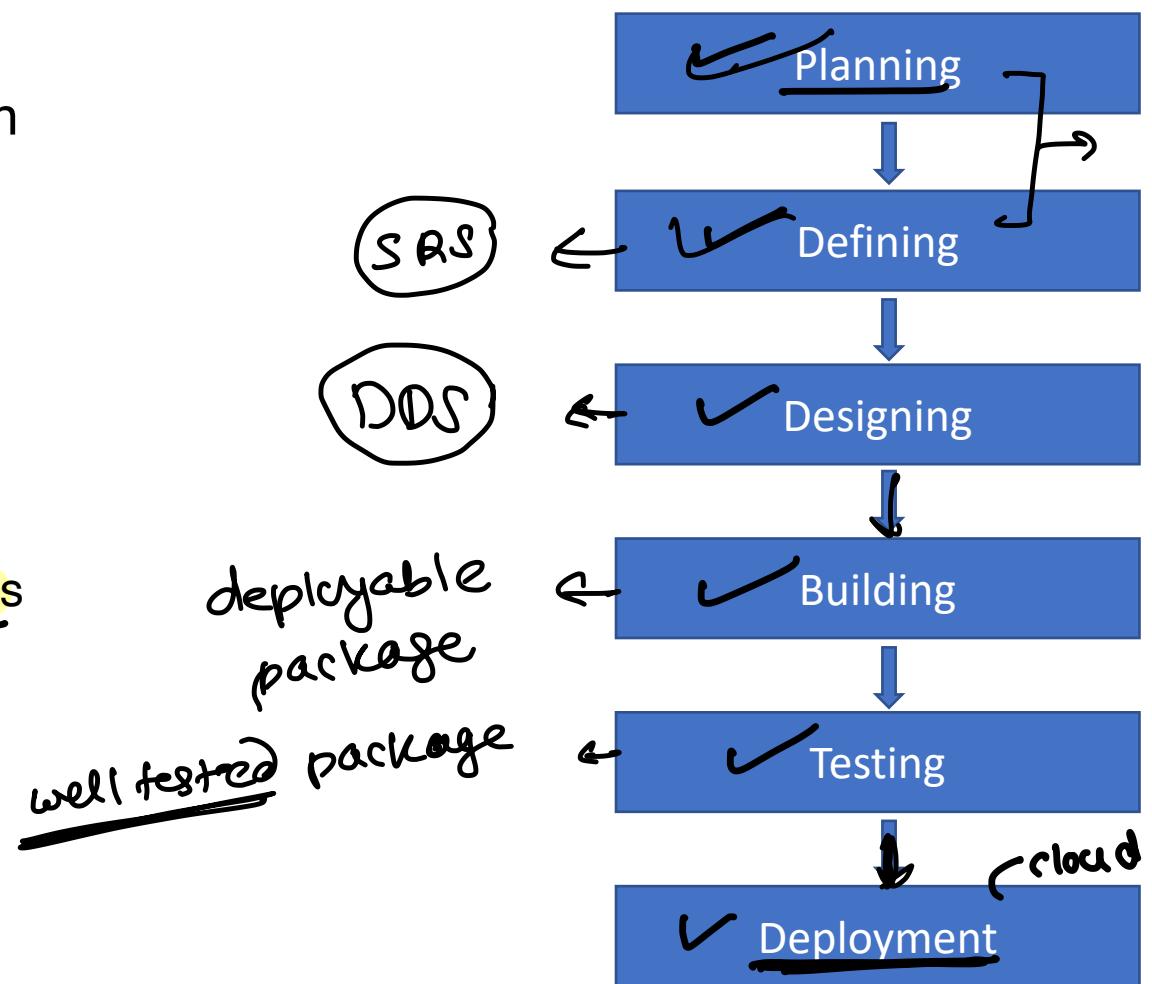


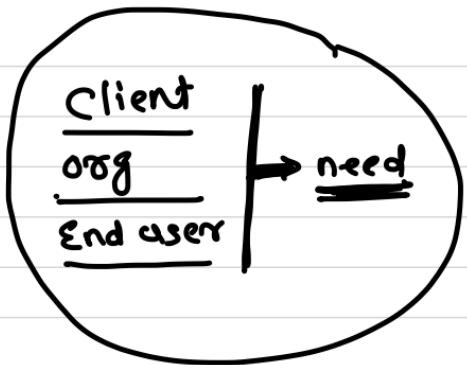
A yellow hand icon with fingers slightly spread, positioned behind the text.

**SDLC**

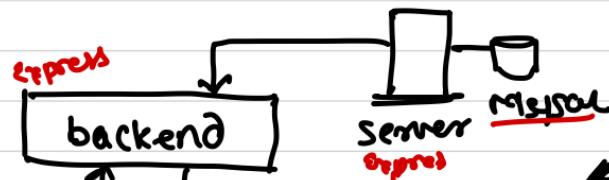
# Overview

- Also called as Software Development Process
- Is a well-defined, structured sequence of stages in software engineering to develop the intended software product **good**
- 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





e-commerce



develop

organization

① planning ←

② requirement gathering

③ requirement analysis →

④ Designing → software architect

↳ architecture

↳ platform - java script

↳ database → RDBMS - NOSQL  
↳ MySQL → Mongo

↳ architecture

→ backend → express -  
→ frontend → angular -

SRS

DDS

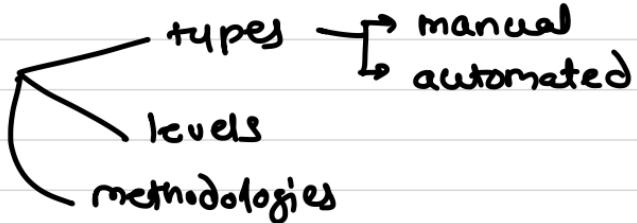
✓ ⑤ Building (development)

↳ programming language

→ IDE, compilers...—

→ create a deployable package

✓ ⑥ Testing



✓ ⑦ Deployment

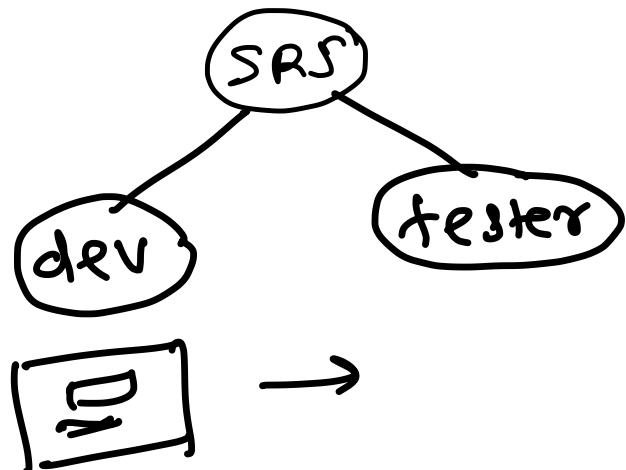
→ package → production environment [cloud]

## 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

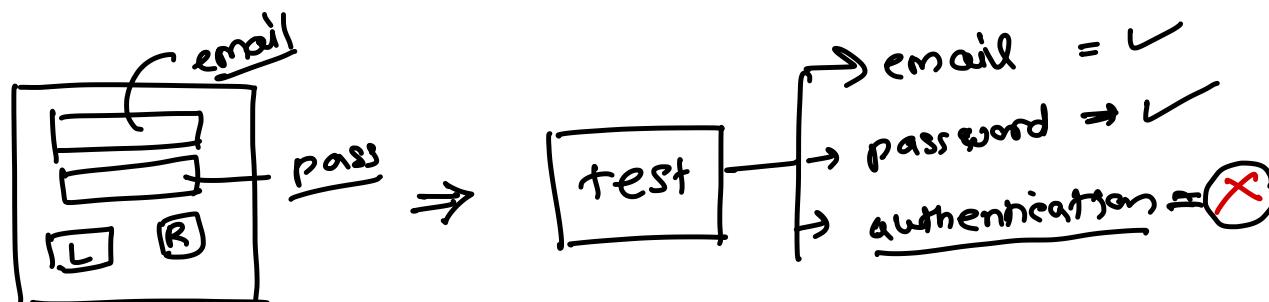
## Building or Developing the Product

- 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

## Testing the Product

SDLC

- 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



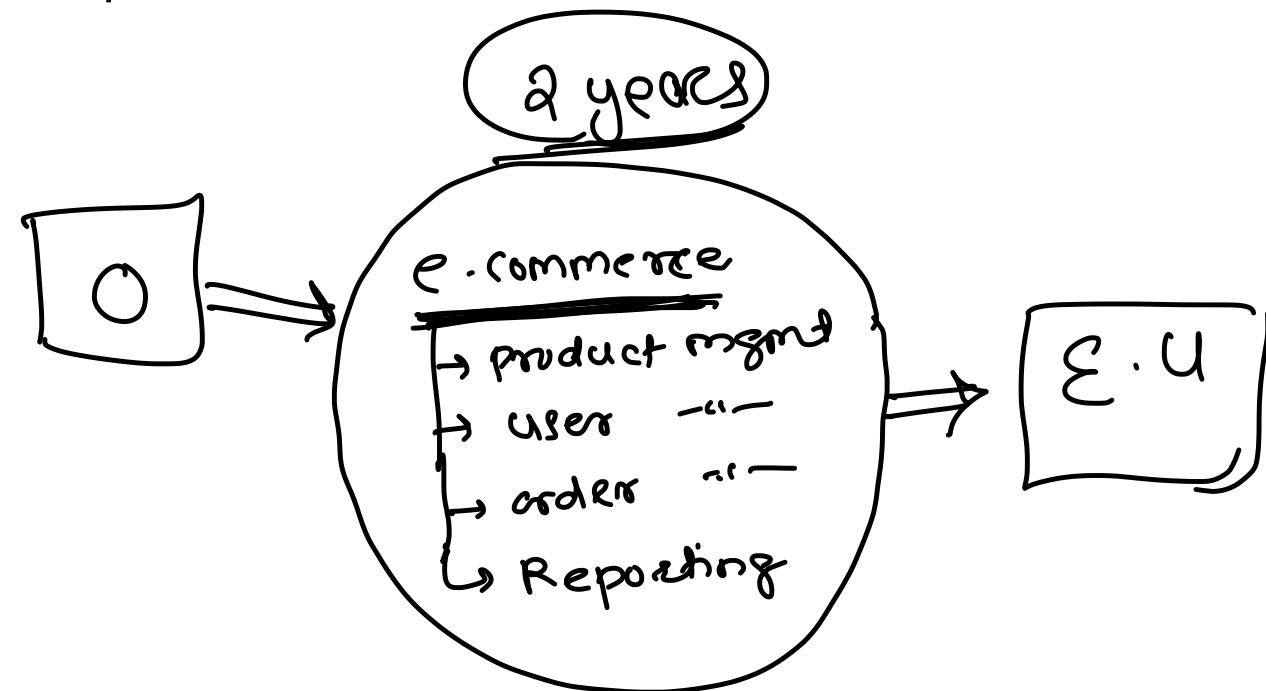
## Deployment in the Market and Maintenance

- 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

1

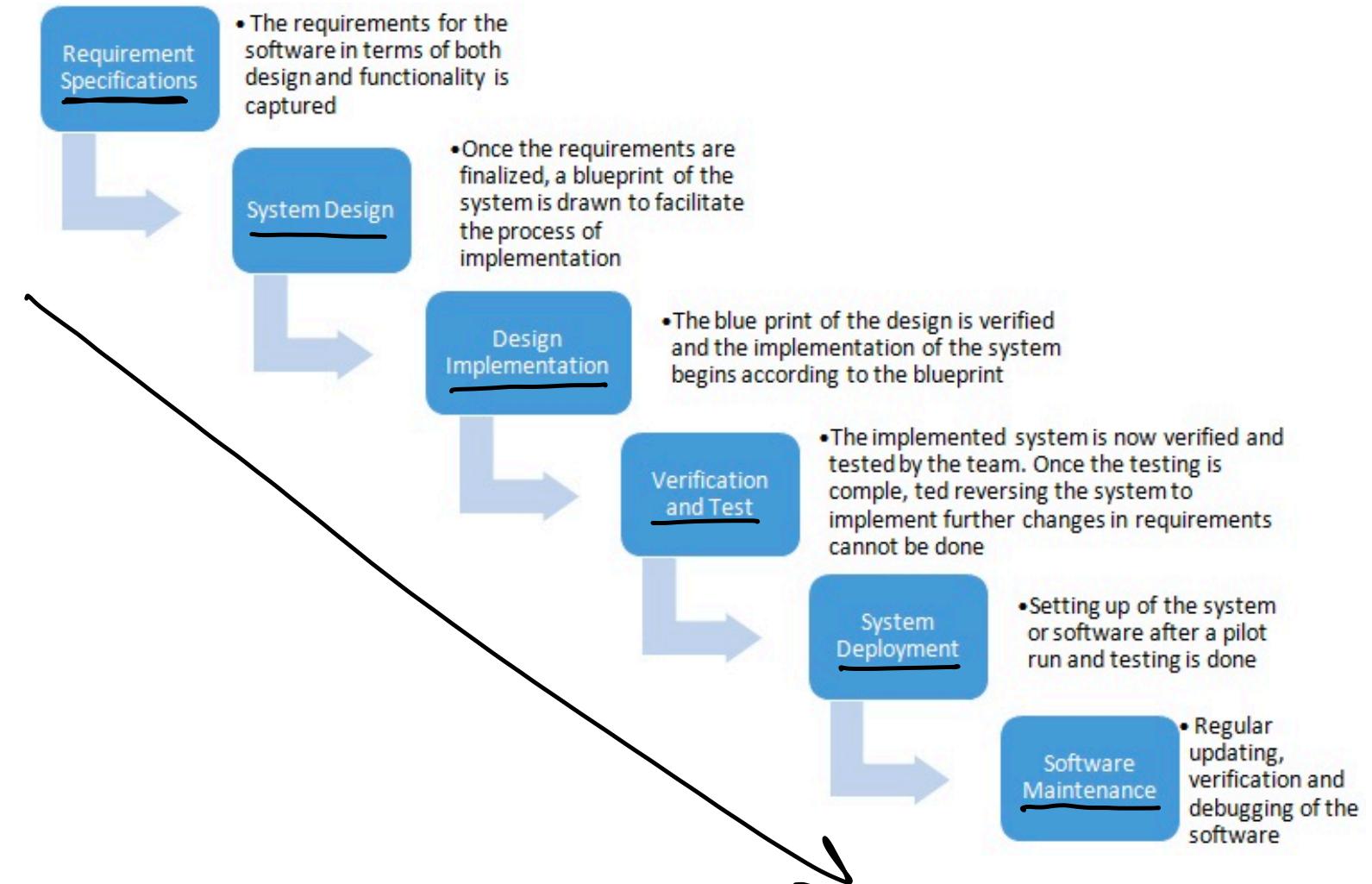
# SDLC Models

- 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



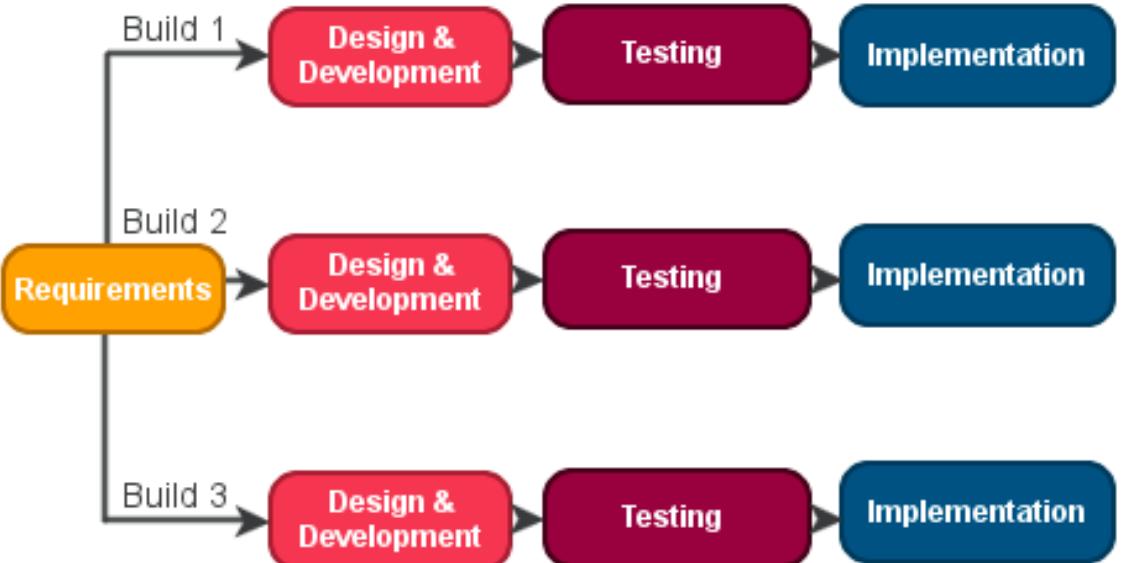
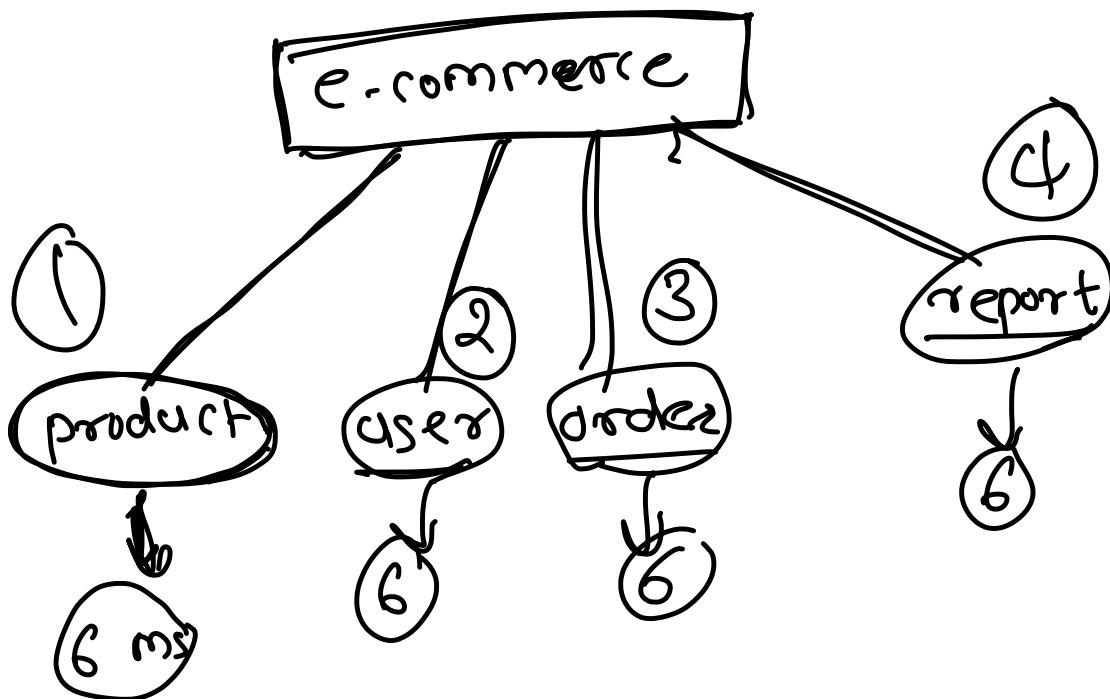
# Waterfall Model

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



# Iterative Model

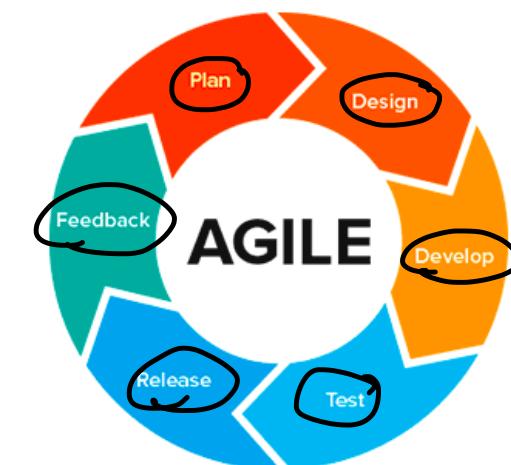
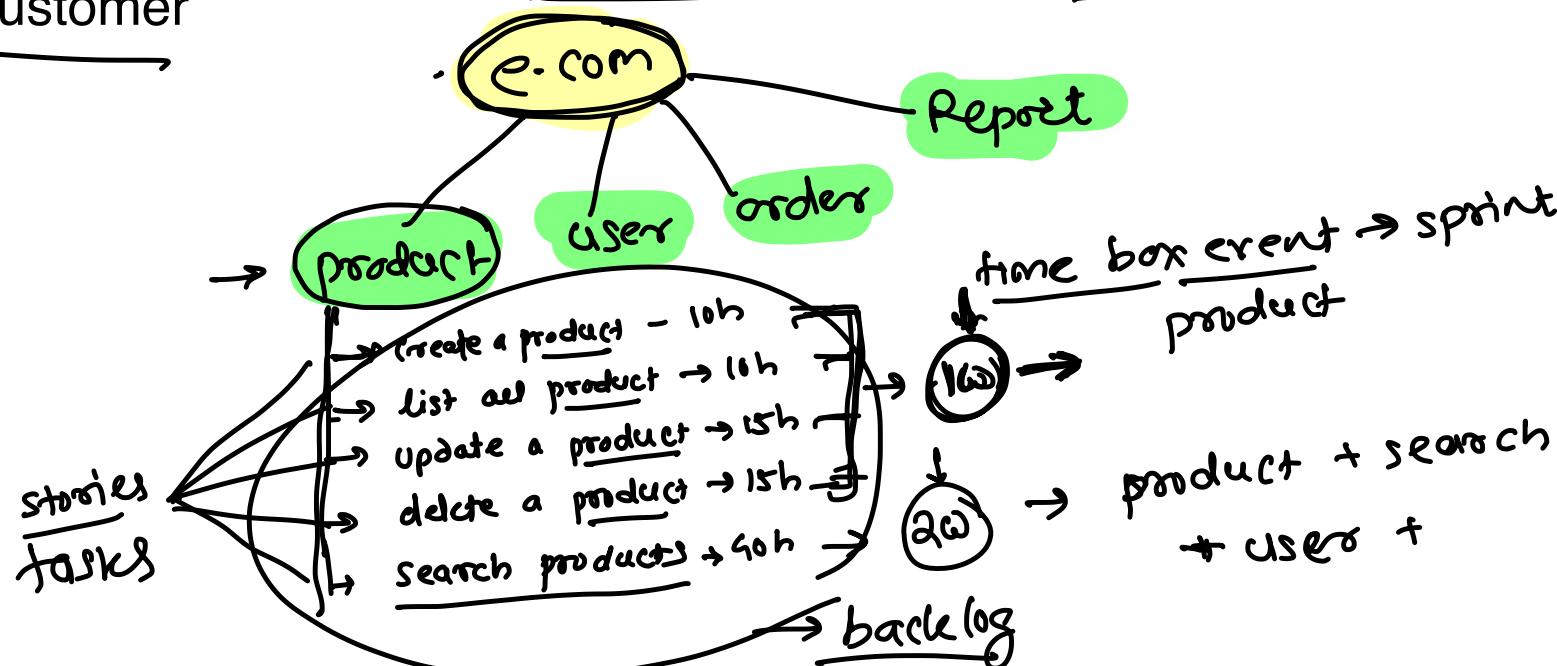
- implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented



# 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**

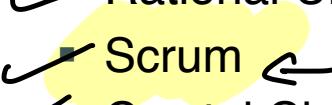
- continuous customer interaction is very important to get proper product requirements

- **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)

# Agile Methodologies - Scrum

- 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
- S-type

# 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.
- **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

↳ daily standup

# Agile Methodologies – Scrum Principles

- **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

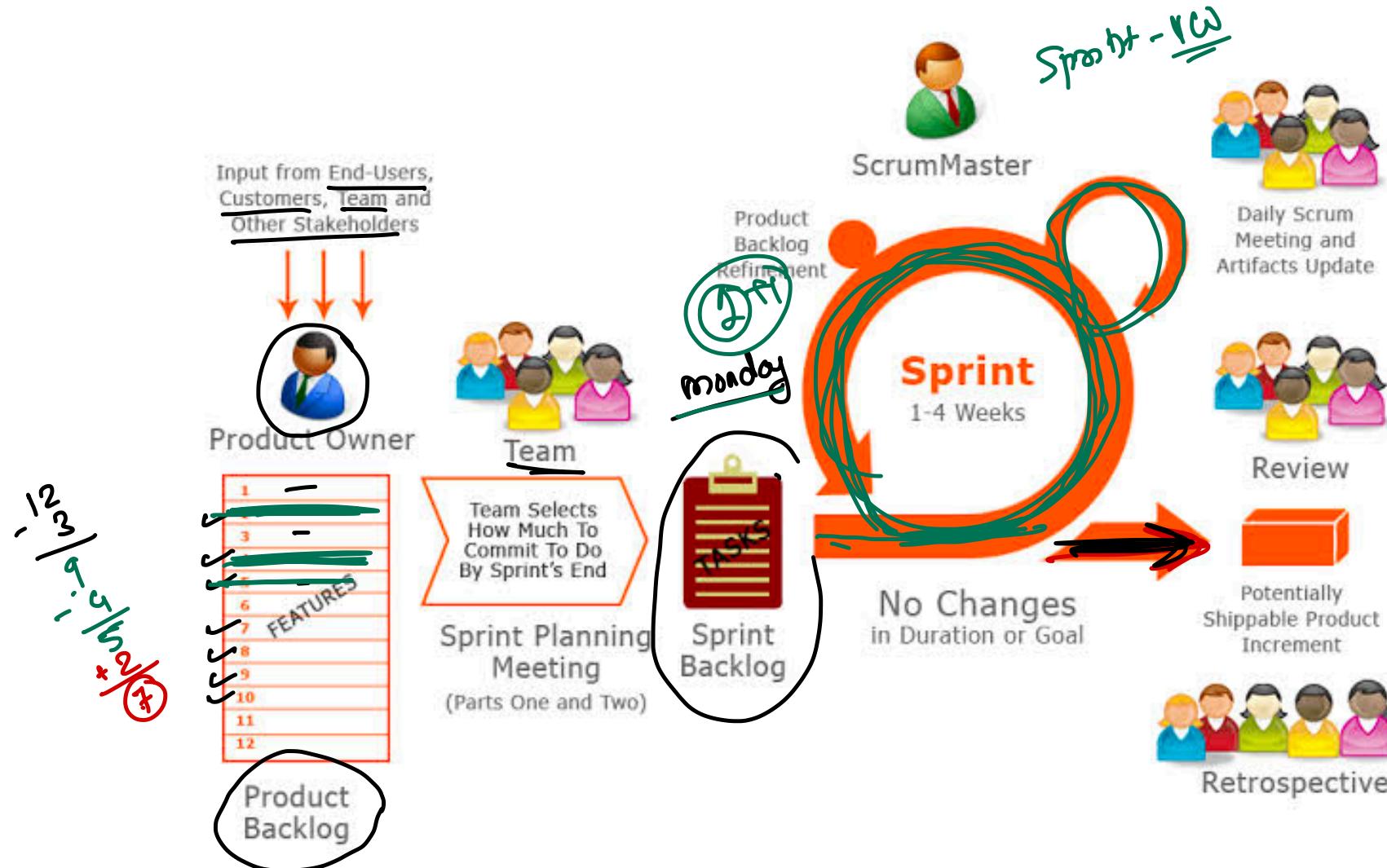
- **Time-boxing**

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

- **Iterative Development** 

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

# Scrum Process



## 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/>
- Quicksrum – <https://www.quicksrum.com/>
- Manuscript – <https://www.manuscript.com/>
- Scrumwise – <https://www.scrumwise.com/>
- Axosoft – <https://www.axosoft.com/>

# 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.



**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  
*bugs/ issues*
- executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements [SRS]
- 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
  - ✓ Software Tester
  - ✓ Project Lead/Manager
  - ✓ End User

## 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 (85%)
- 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

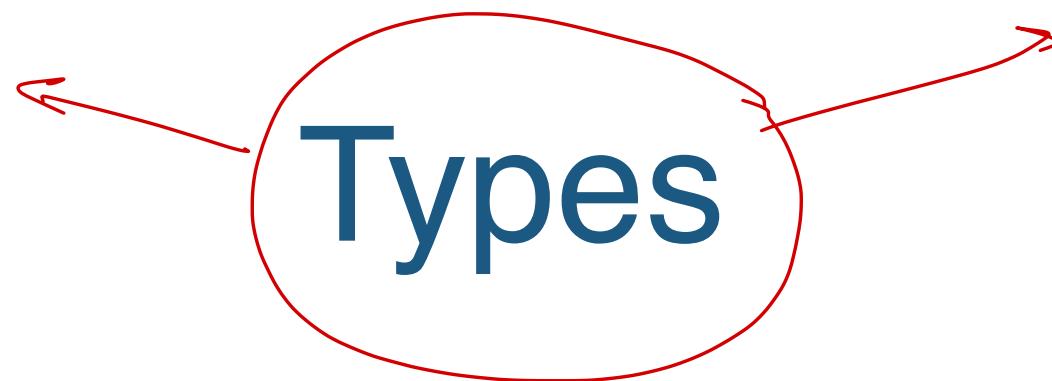
## ■ Audit

- 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

- 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

manual



- automated
- Selenium (python/js/java..)
  - TestNG (Java)
  - Cypress (---)
  - Jest (Javascript)
  - ⋮
  - ⋮
  - ⋮

# 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 [Selenium]
- 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?

---

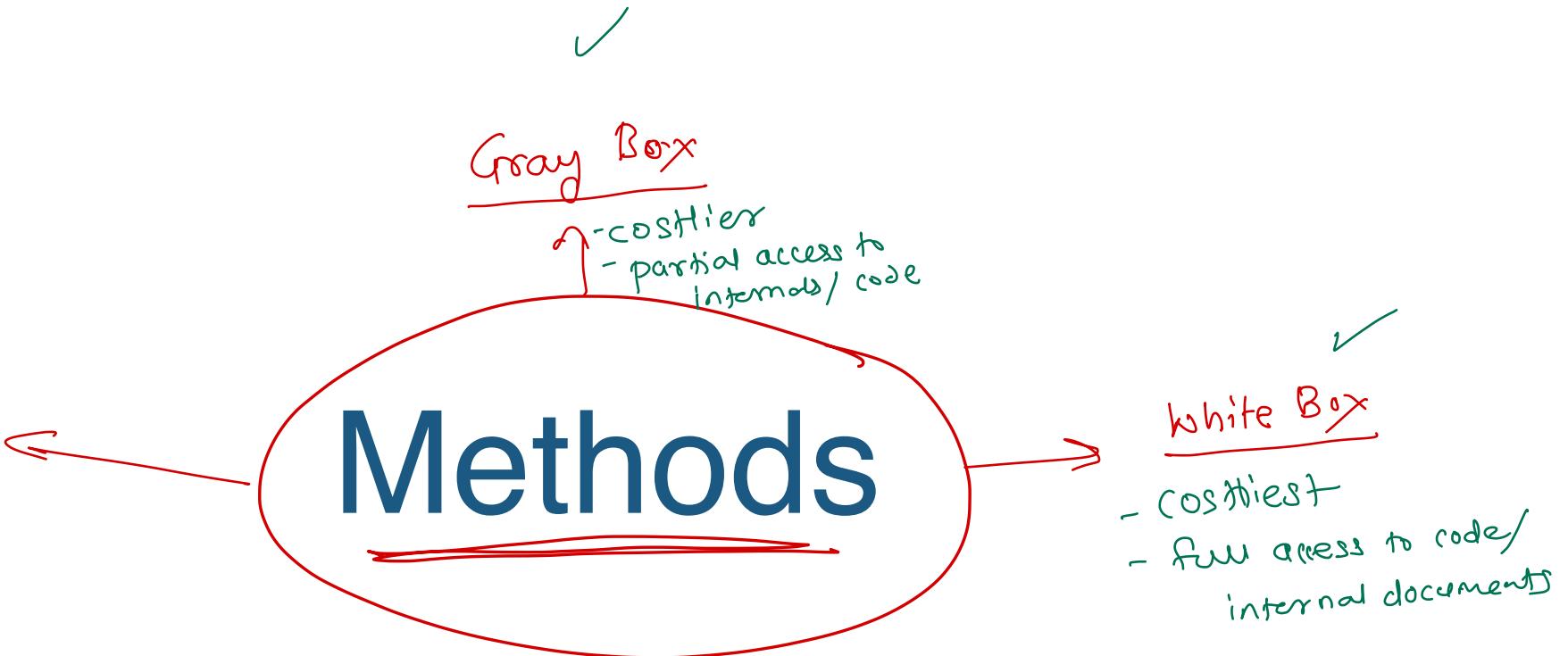
- Identifying areas within a software for automation
- Selection of appropriate tool for test automation
- Writing test scripts
- Development of test suits
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues



# Software Testing Tools

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

✓  
Black-Box  
- cheapest  
- no access to  
internals/ code



# Black-Box Testing

code / technology / design

- 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

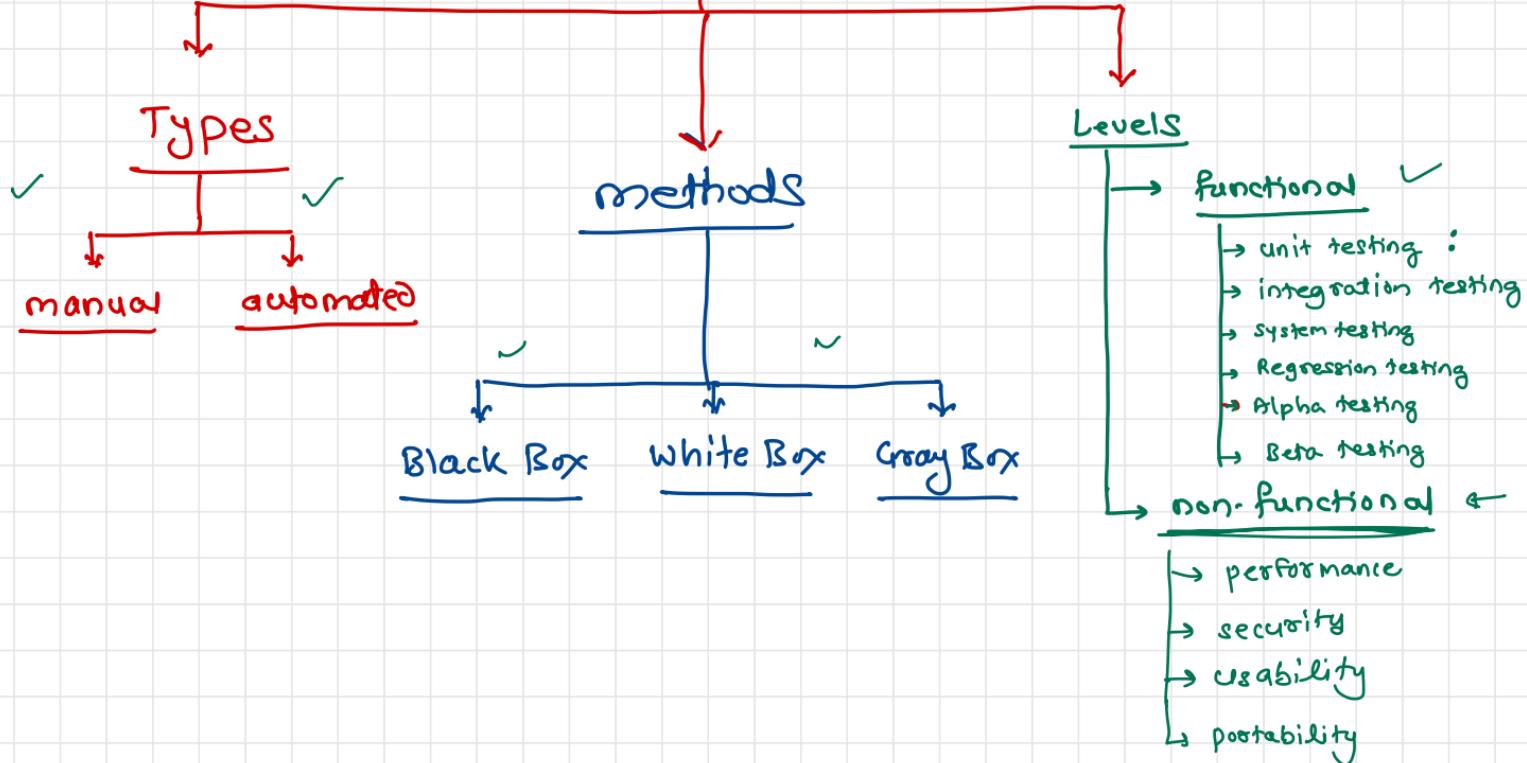
- A technique to test the application with having a limited knowledge of the internal workings of an application
- Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database
- Advantages
  - Offers 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

# Testing



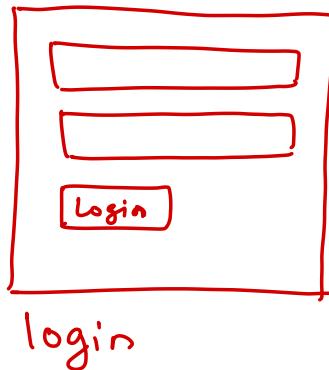


# Functional Testing

- 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 [ SRS ]
- Includes
  - ✓ □ Unit Testing
  - ✓ □ Integration Testing
  - ✓ □ System Testing
  - ✓ □ Regression Testing
  - ✓ □ Acceptance Testing
  - ✓ □ Alpha Testing
  - ✓ □ Beta Testing

# 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



functionality: user authentication

test data:

{ email: "test@test.com", password: "12345" }, → valid

{ email: "+@", password: "" } → invalid

function add ( p1, p2 ) {

    return p1 + p2

} expected result

test data:

{ p1: 20, p2: 20, answer: 40 } → +ve

{ p1: 20, p2: 20, answer: 50 } → -ve

const ans1 = add ( 20, 20 )  
→ actual result

# ① Unit Testing

= individual function testing

- 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
- 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

### sonarqube

\* Javascript : Jasmine, Jest

\* Java : JUnit

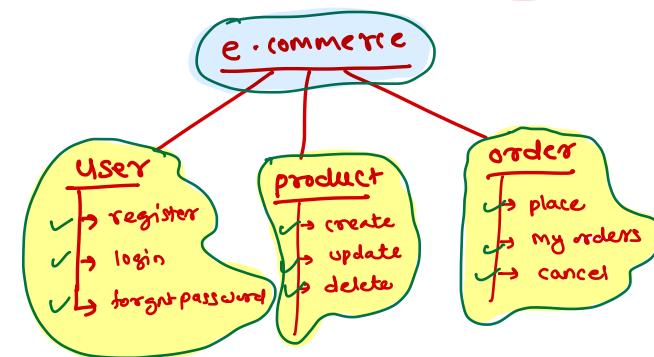
\* .net : Nunit

\* python : Pyunit

# Integration Testing

integration of functions / modules

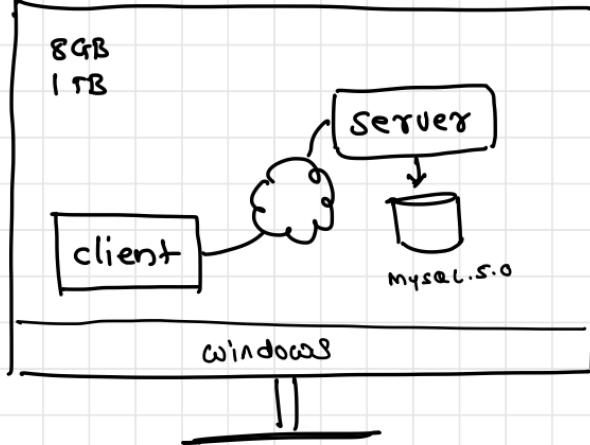
- Integration testing is defined as the testing of combined parts of an application to determine if they function correctly
- Integration testing can be done in two ways
  - Bottom-up integration testing
    - This testing begins with unit testing, followed by tests of progressively higher-level combinations 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

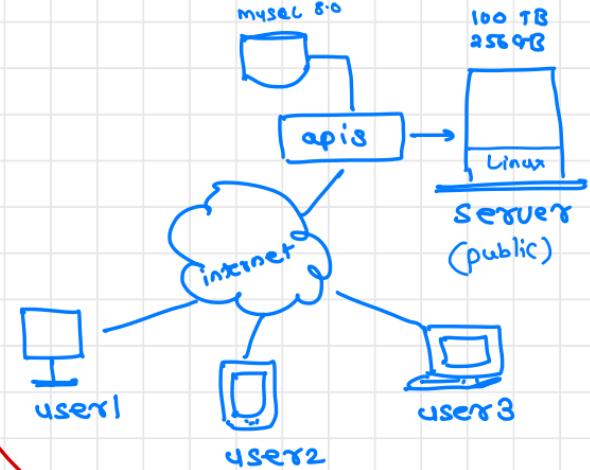
combine all modules / parts

- System testing tests the system as a whole [application 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

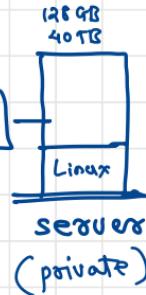
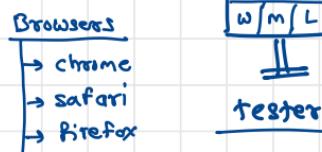


Dev environment

QA environment



production



# 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 deduce 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

: Release Candidate (RC)

- 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 (comment / issues / bugs...)
- 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

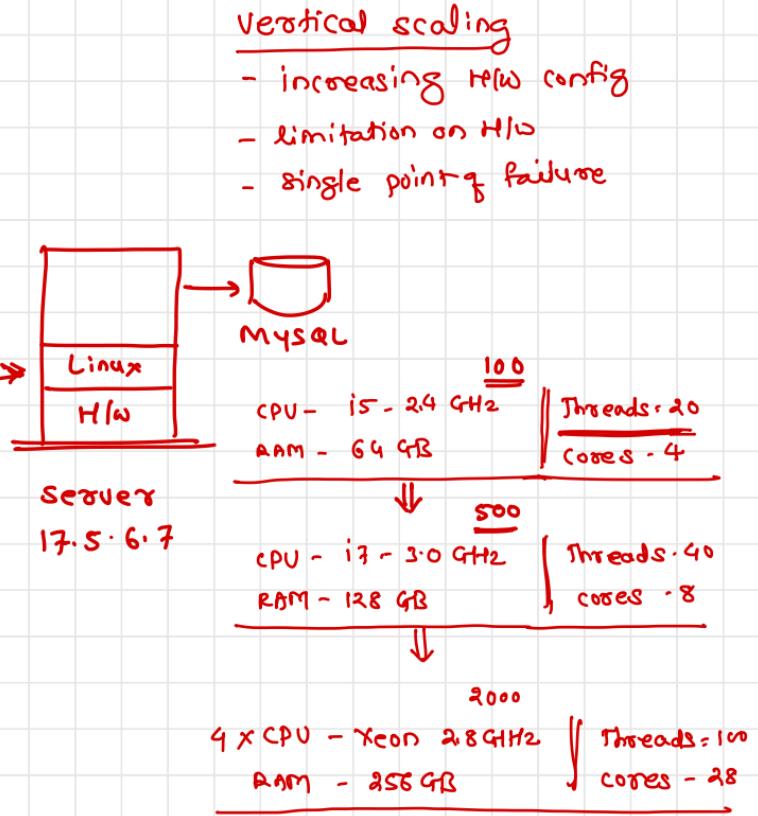
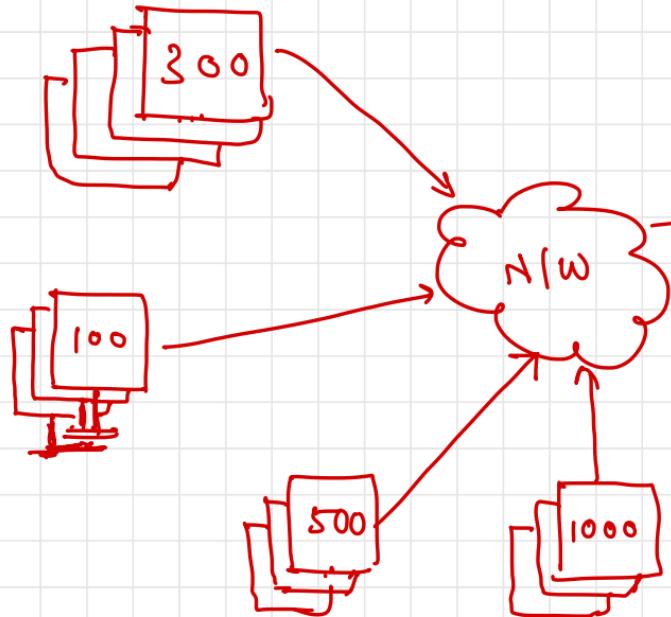
# Non-Functional Testing

---

- 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

# Performance Testing

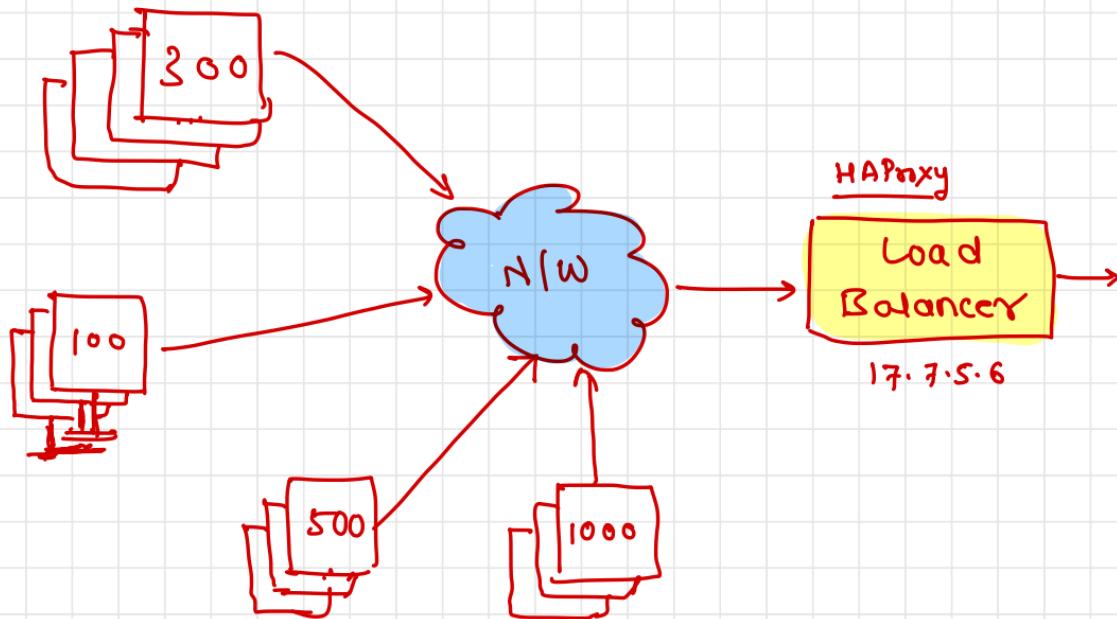
- It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software
- There are different causes that contribute in lowering the performance of a software
  - Network delay
  - Client-side processing
  - Database transaction processing
  - Load balancing between servers
  - Data rendering
- 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
  - Stability
  - Scalability
- Performance testing can be either qualitative or quantitative and can be divided into
  - Load testing
  - Stress testing



Scalable  
→ Vertical

Horizontal scaling

Highly Available



Cluster

## 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

---

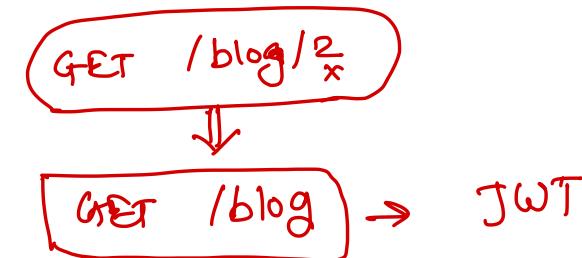
- Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation
- Can be defined as
  - efficiency of use
  - learn-ability
  - memory-ability
  - errors/safety
  - Satisfaction
- UI testing can 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
- ✓ ■ Integrity
- ✓ ■ Authentication
- ✓ ■ Availability
- ✓ ■ Authorization : JWT
- ✓ ■ Non-repudiation
- ✓ ■ Input checking and validation ; SQL injection
- ✓ ■ SQL insertion attacks ; Helmet



# 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

used for UI testing

- 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

## 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

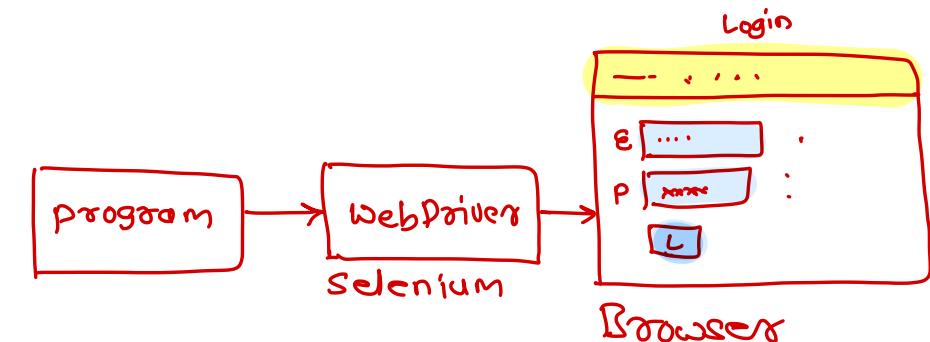
## Locators

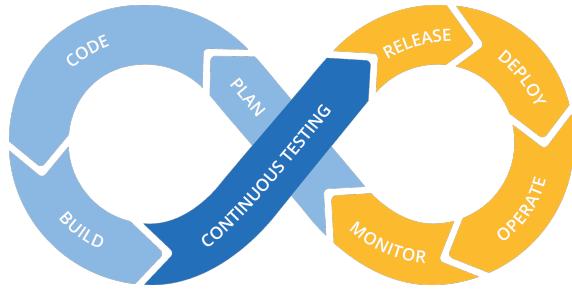
( document.getElementById( ... )

- 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





DevOps  
~~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, cvs, sun, bazar*)
- 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**

Containerization  
→ Kubernetes

## Overview

Developer Tester

- 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

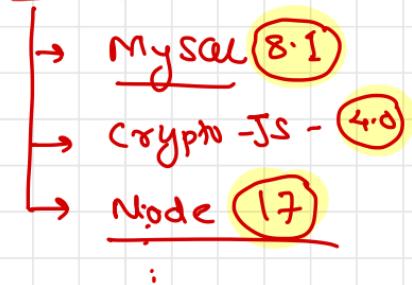
## Why DevOps is Needed?

- 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 sync causing further delays

## Development

- ① Develop / code
- ② testing
- ③ Creating build

### Express



## Operations

- ① managing resources
  - machines
  - HW
  - network

- ② deployment of app

- MySQL → 8.0
- Node - 16
- Crypto-JS - 3.5

# 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 → dev - deployment
  - Continuously measuring application performance



\* mindset

\* culture

## 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

---

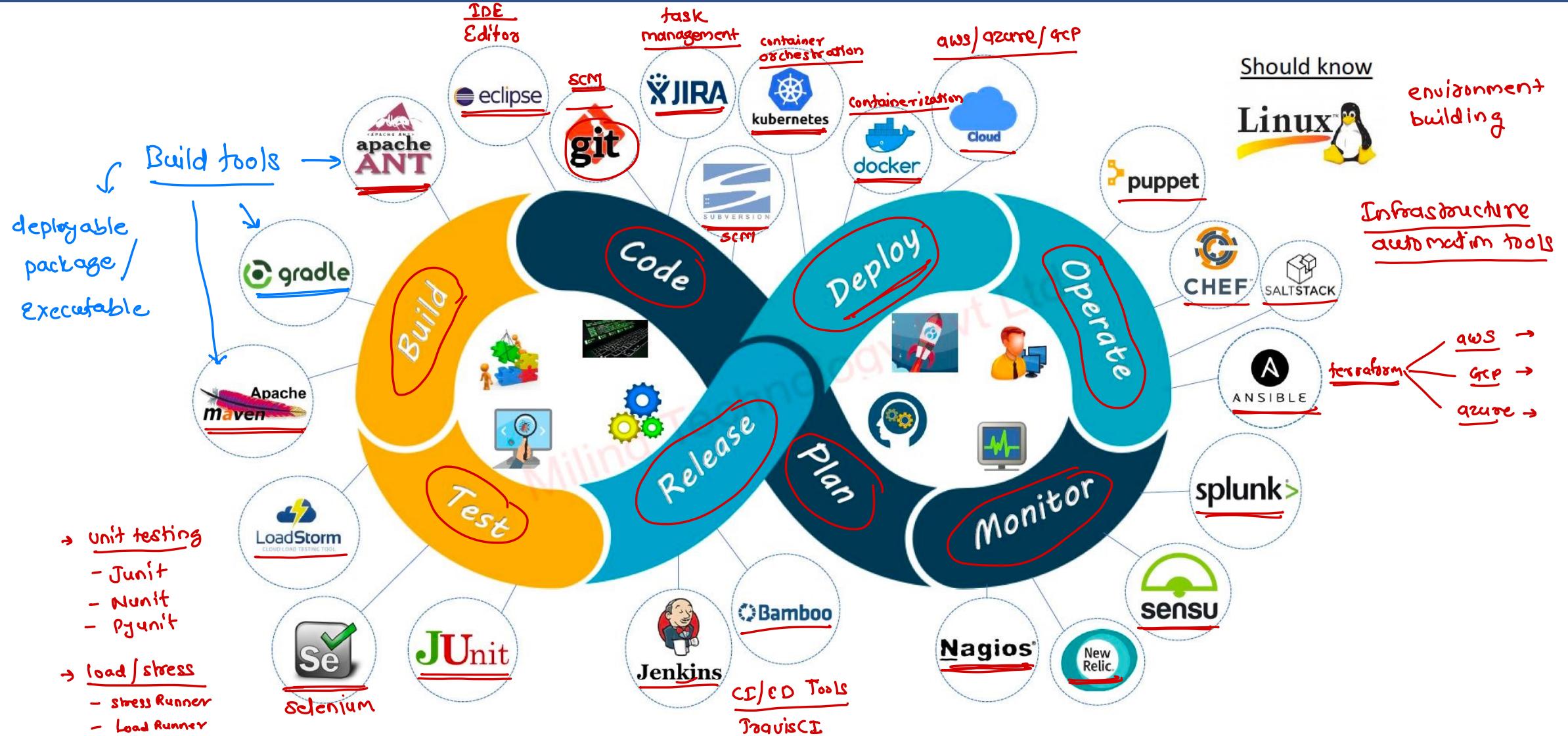
- **Predictability:** DevOps offers significantly lower failure rate of new releases
- **Reproducibility:** Version everything so that earlier version can be restored anytime
- **Maintainability:** Effortless process of recovery in the event of a new release crashing or disabling the current system
- **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

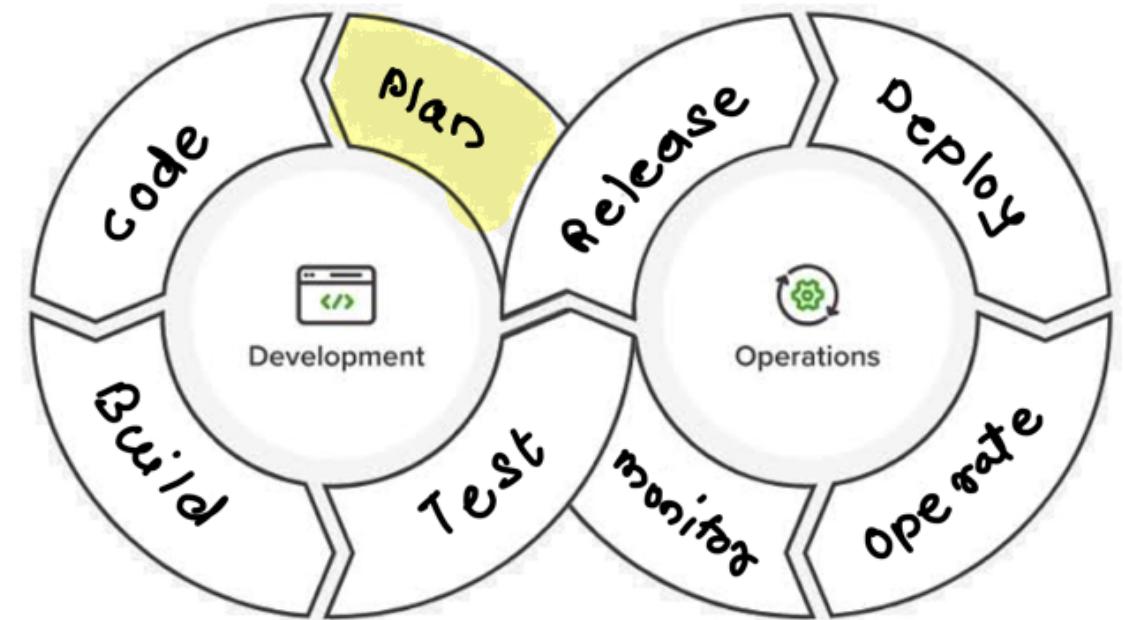
# DevOps Lifecycle



# DevOps Lifecycle - Plan

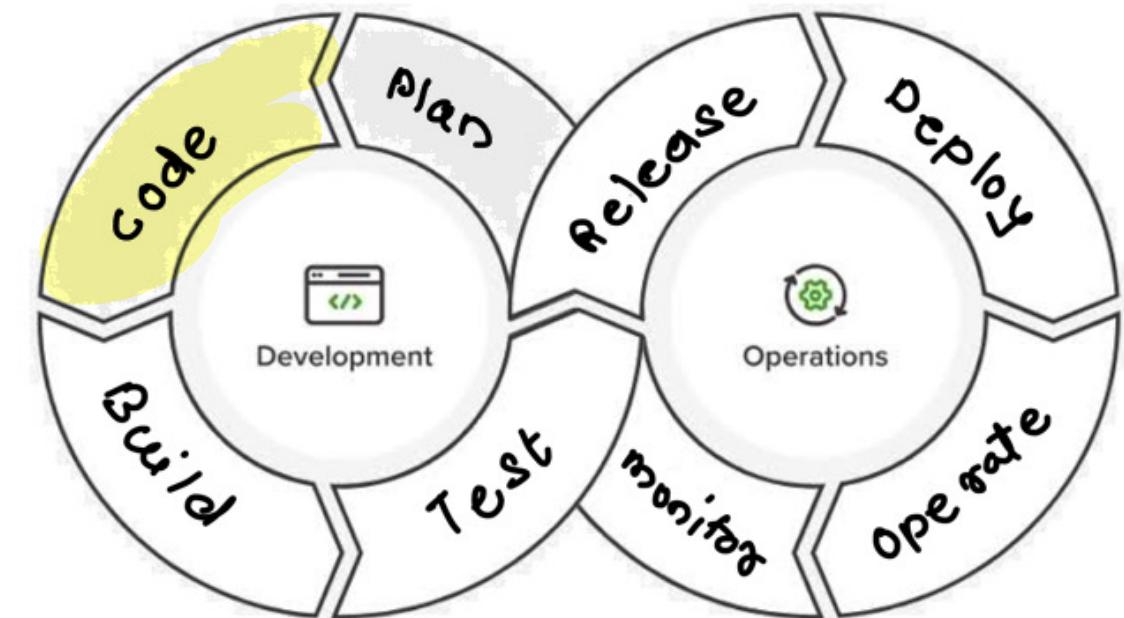
---

- 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

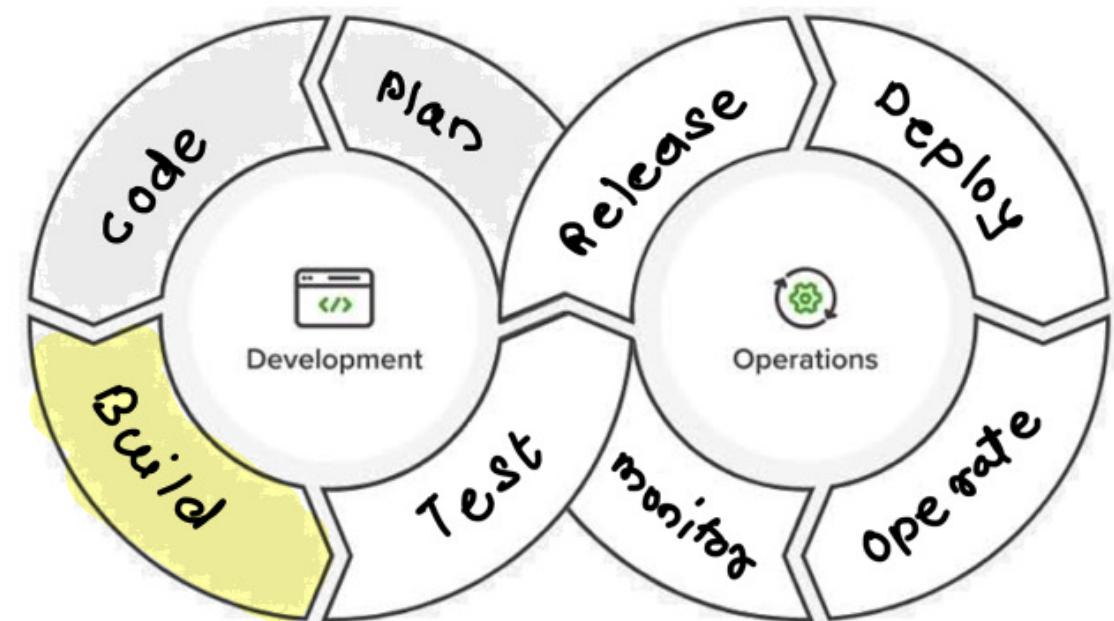
- Second stage where developer writes the code using favorite programming language
- Coding Tools
  - IDEs: Eclipse, Visual Studio etc.
  - SCM: Git, Subversion, CVS etc.
  - Package management: npm etc.



# DevOps Lifecycle -Build

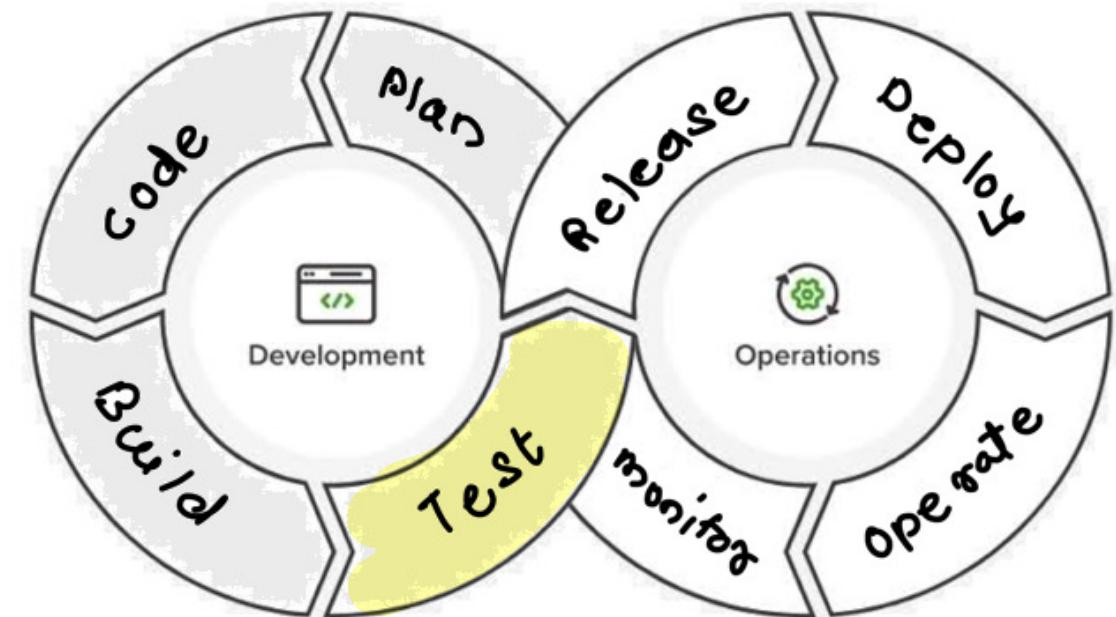
- Integrating the required libraries
- Compiling the source code
- Create deployable packages
- Build tools
  - Maven
  - Gradle
  - Ant

. libraries +  
compiled code → package



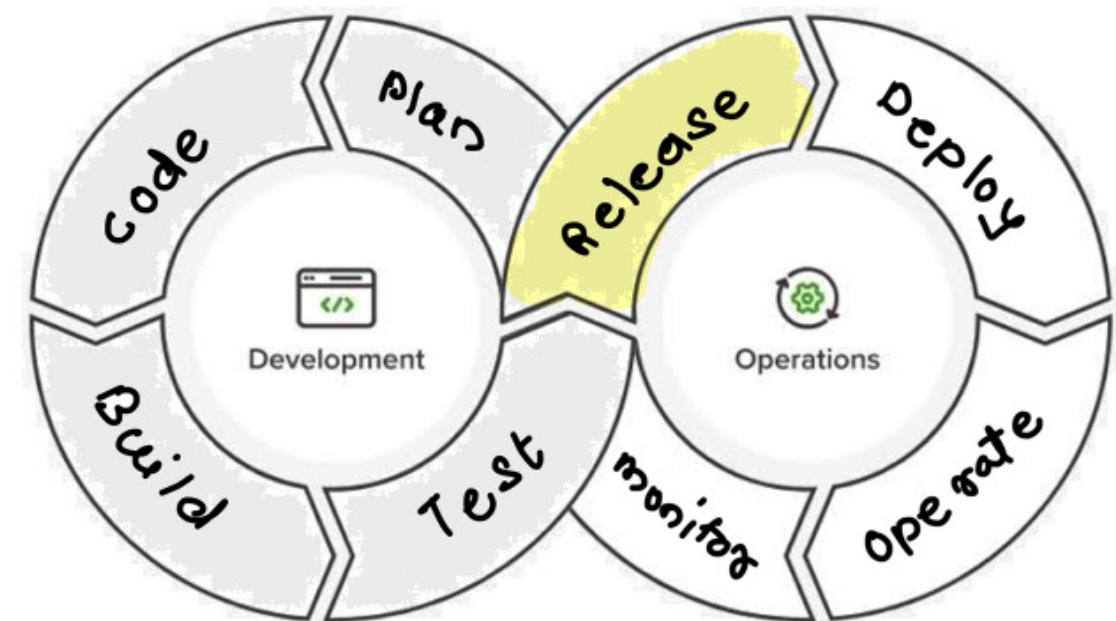
# DevOps Lifecycle - Test

- 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



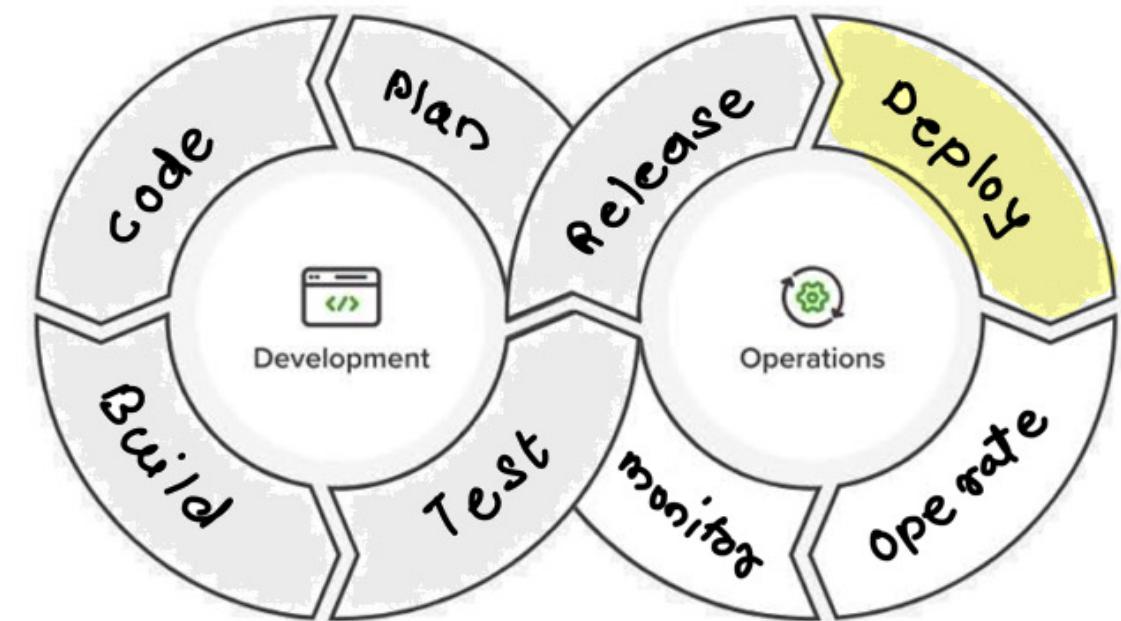
# DevOps Lifecycle - Release

- 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



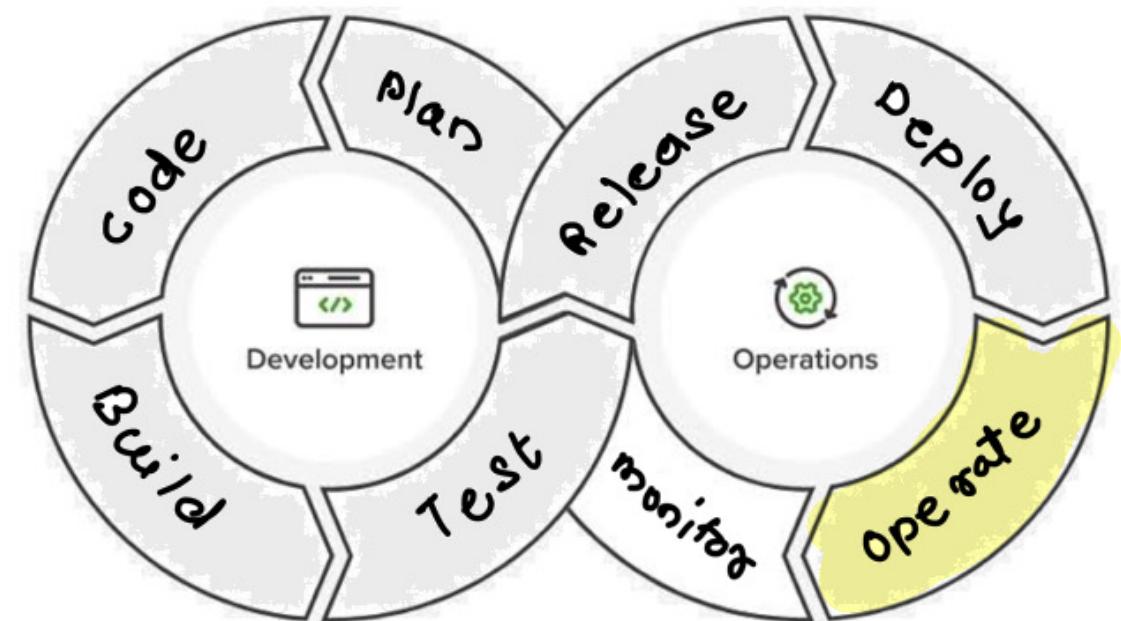
# DevOps Lifecycle - Operate

- This stage where the updated system gets operated

- Operating Tools

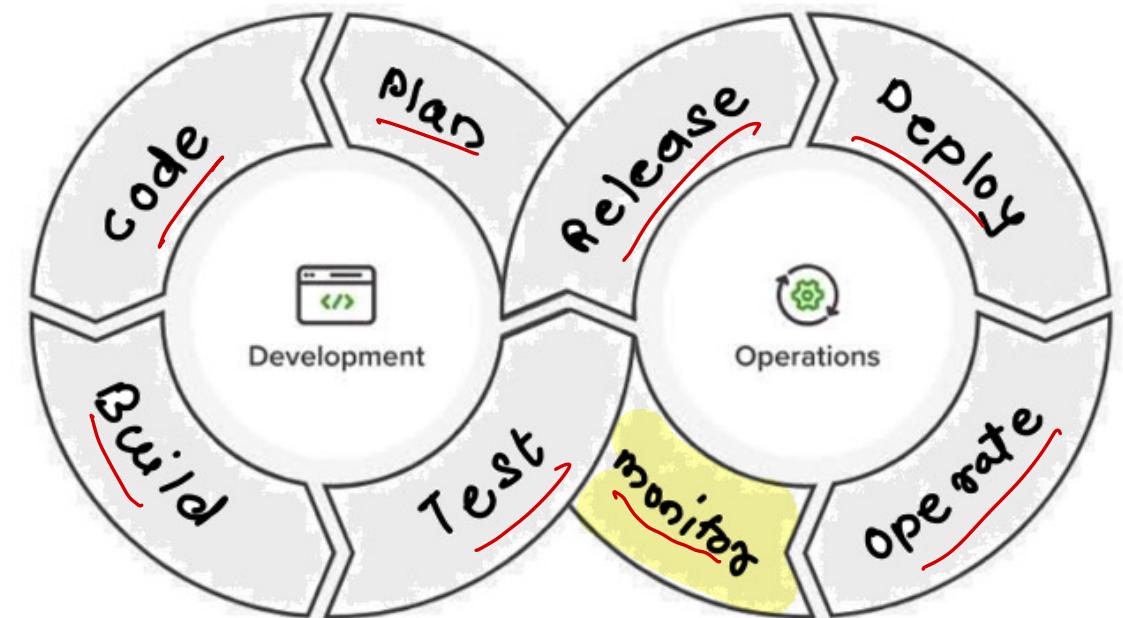
- Puppet
- Chef
- Ansible

Configuration management



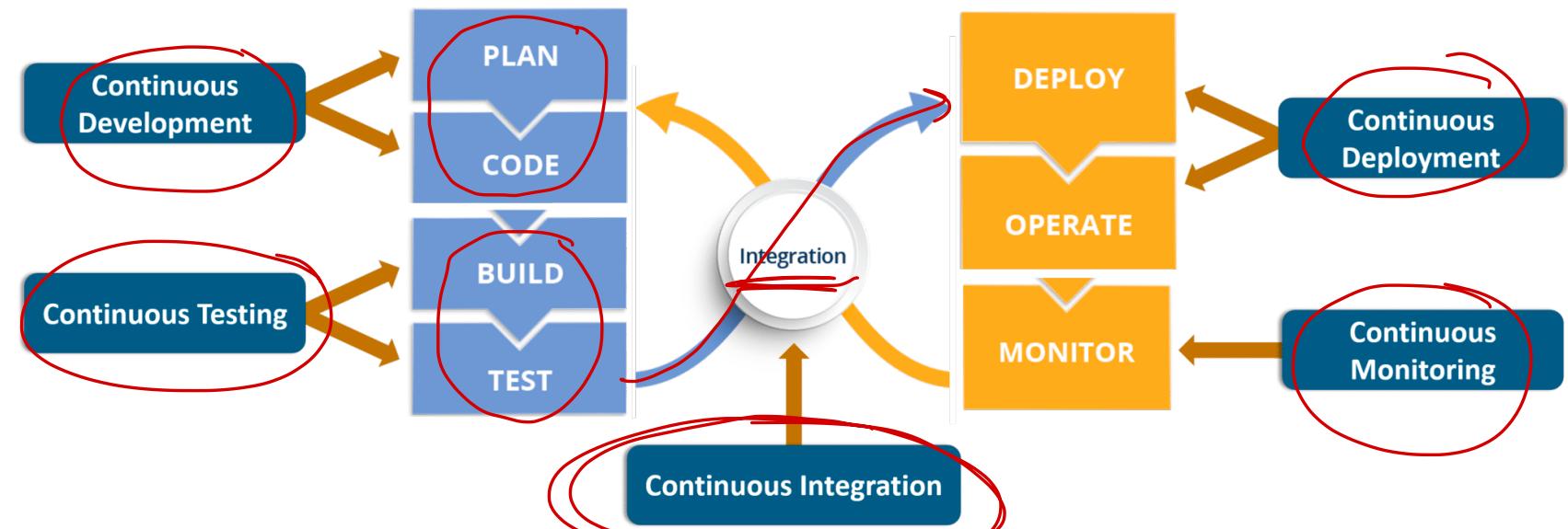
# DevOps Lifecycle - Monitor

- 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
- Monitoring tools
  - Nagios
  - Sensu
  - Splunk
  - DataDog



# DevOps Terminologies

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



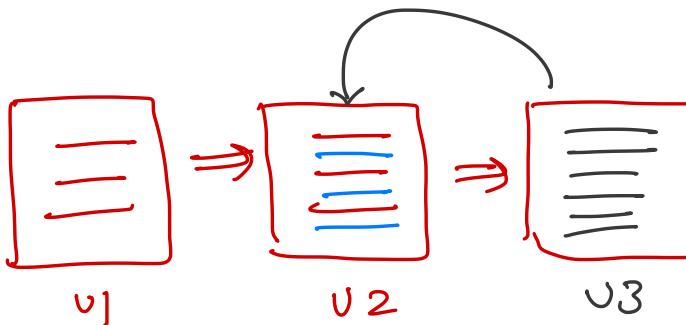
# Source Code Management

## Why do we need Version Control System? (SCM)

- 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)

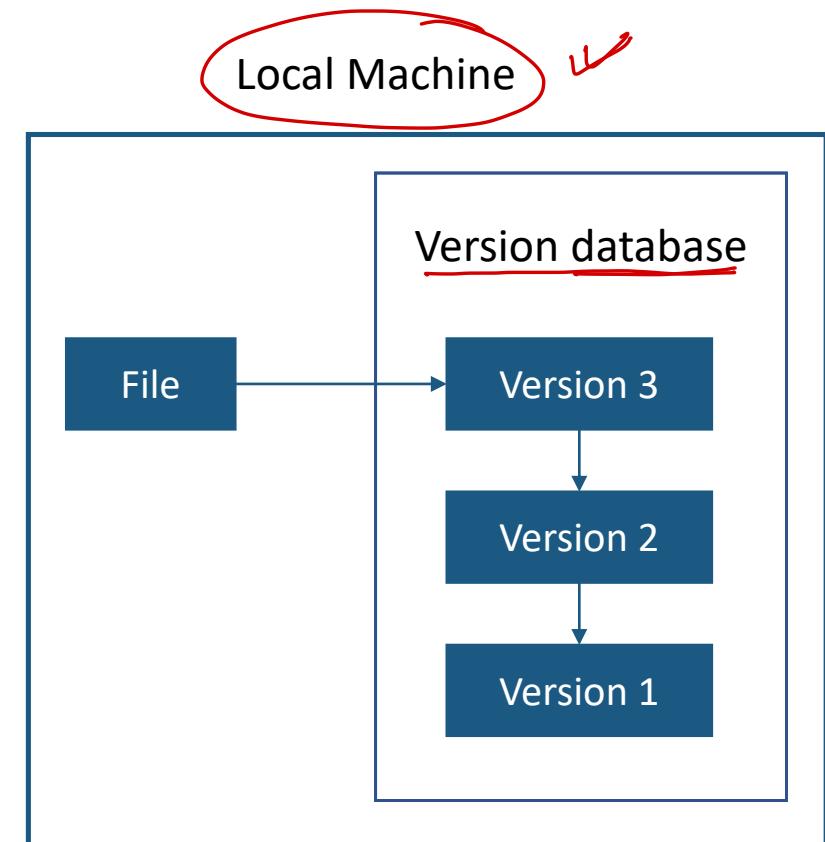
# What is Version Control System?

- 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

- 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



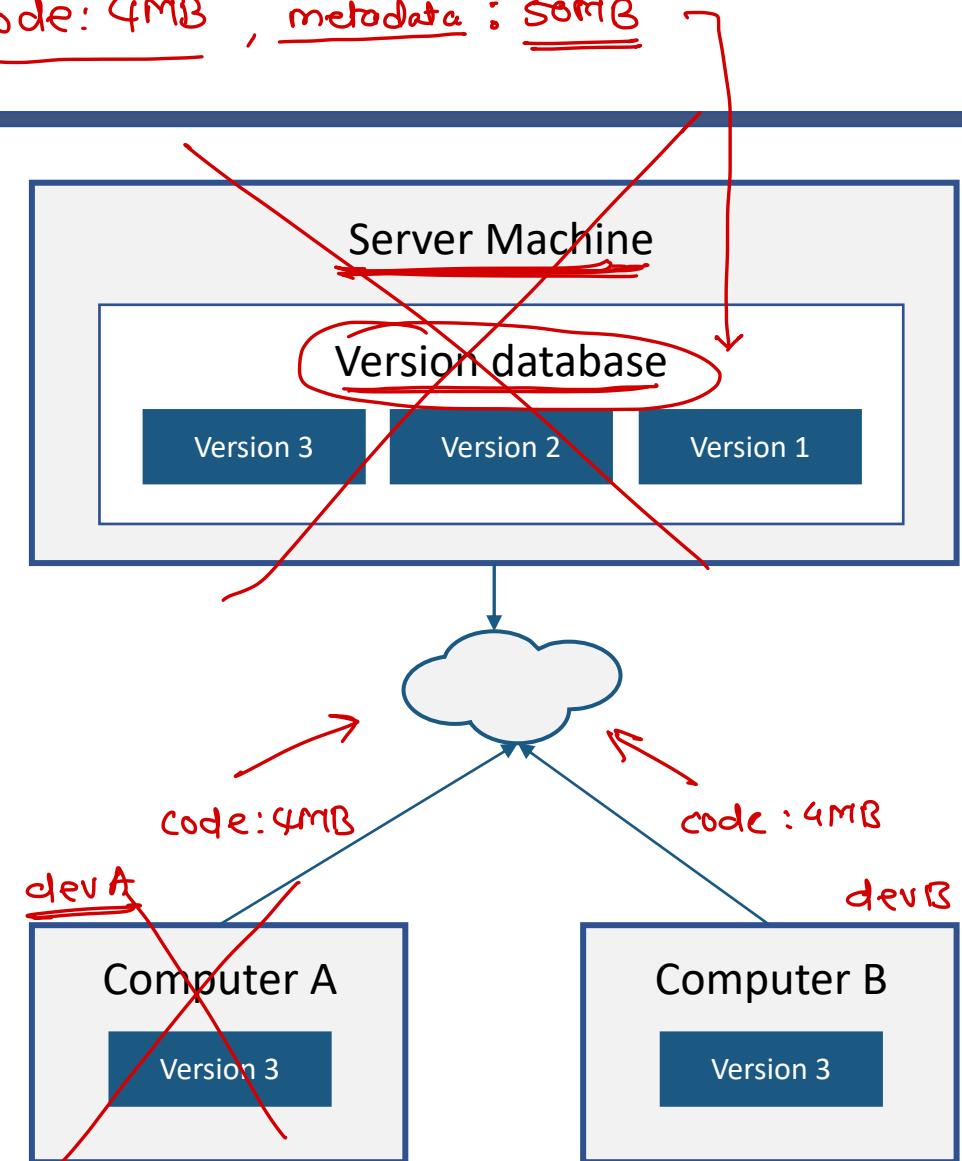
# Centralized Version Control Systems

- 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

## Limitations

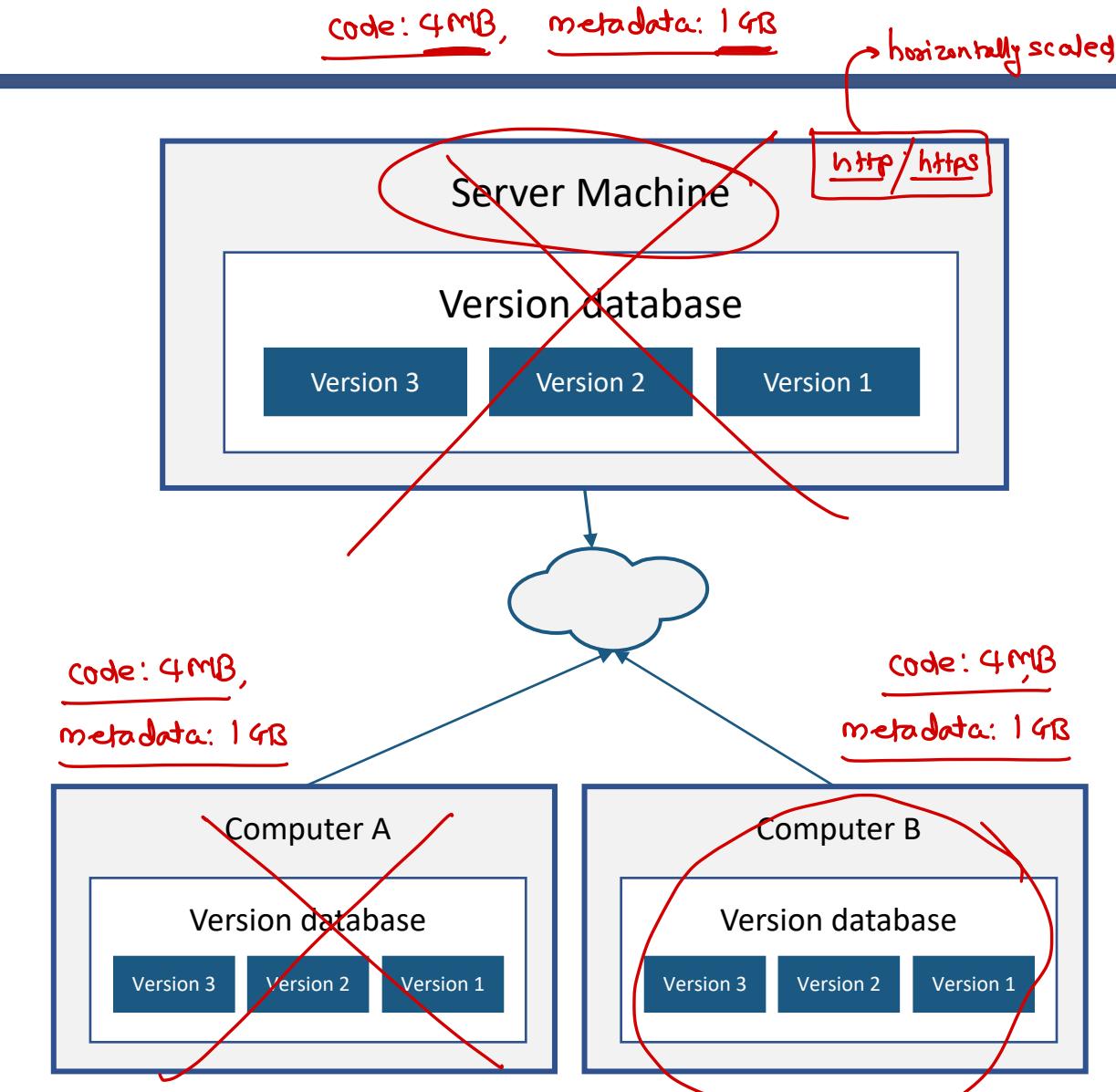
- ① no of users are limited
- ② single point of failure

Code: 4MB , metadata : 50MB



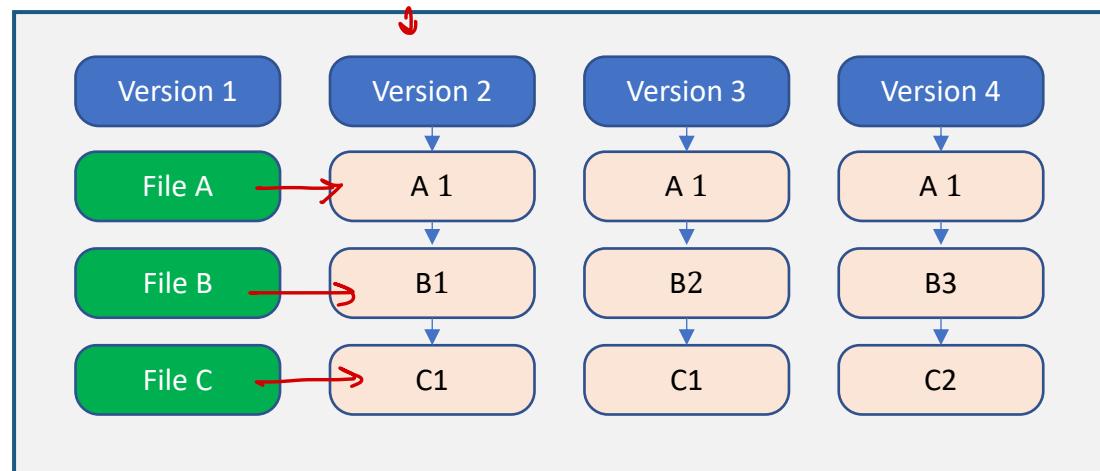
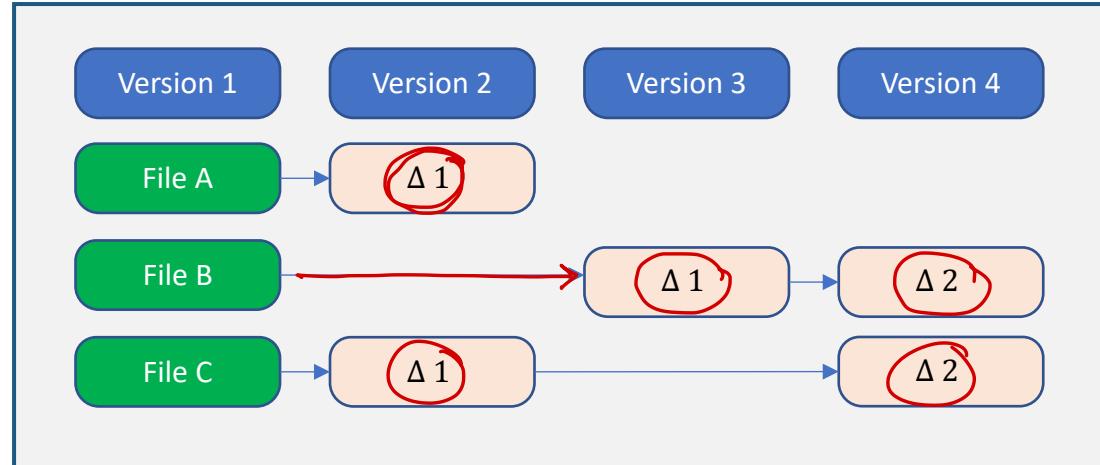
# Distributed Version Control Systems

- 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 (patch files)
- Git thinks of its data more like a set of snapshots of a miniature filesystem (backup)
- 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 (branches)
- Distributed development
- Compatibility with existent systems and protocols (http, 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 (**Binary Large Object**)
  - 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

---

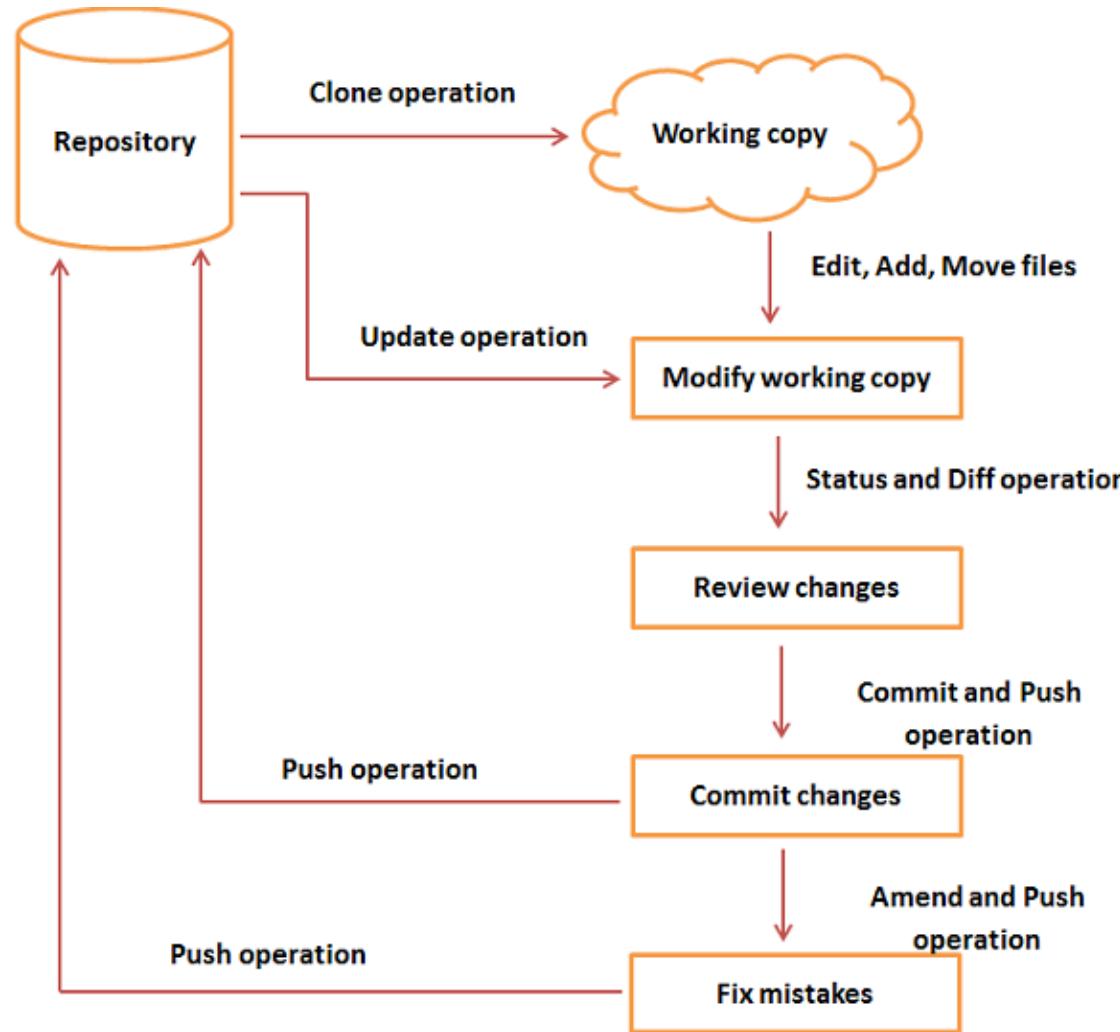
- Pull
  - Pull operation copies the changes from a remote repository instance to a local
  - The pull operation is used for synchronization between two repository instances
- Push
  - Push operation copies changes from a local repository instance to a remote
  - This is used to store the changes permanently 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

---

- 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

## Why is it required ?

---

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

# Branch management commands

---

- **Create a branch**

- > git branch <branch name>

- **Checkout a branch**

- > git checkout <branch name>

- **Merge a branch**

- > git merge <branch name>

- **Delete a branch**

- > git branch -d <branch name>

# GitHub

# Overview

---

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

# Workflow

---

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

# 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

① introduction

② virtualization

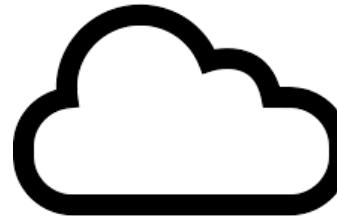
↳ type I  
↳ type II

③ service models

↳ IaaS  
↳ PaaS  
↳ SaaS

④ deployment models

↳ public  
↳ private  
↳ hybrid



Cloud Computing

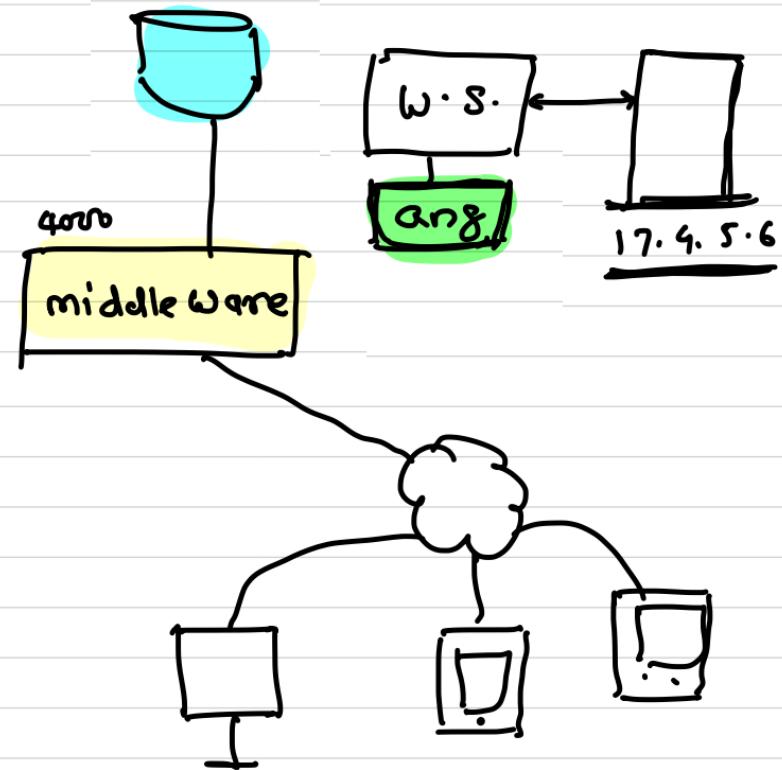
① MySQL db

② backend (express)

③ Frontend (angular)

http://localhost:4200 X

public

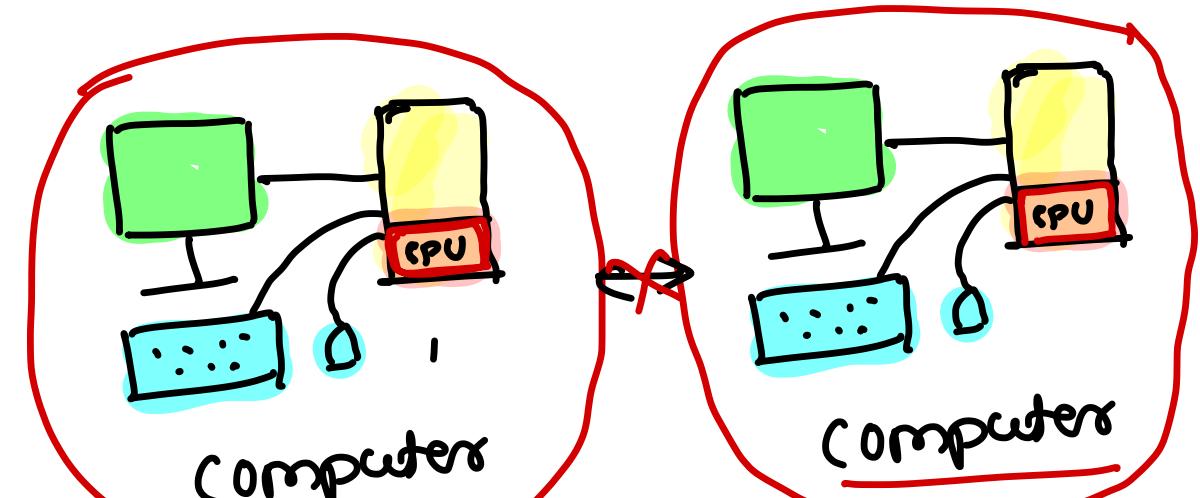
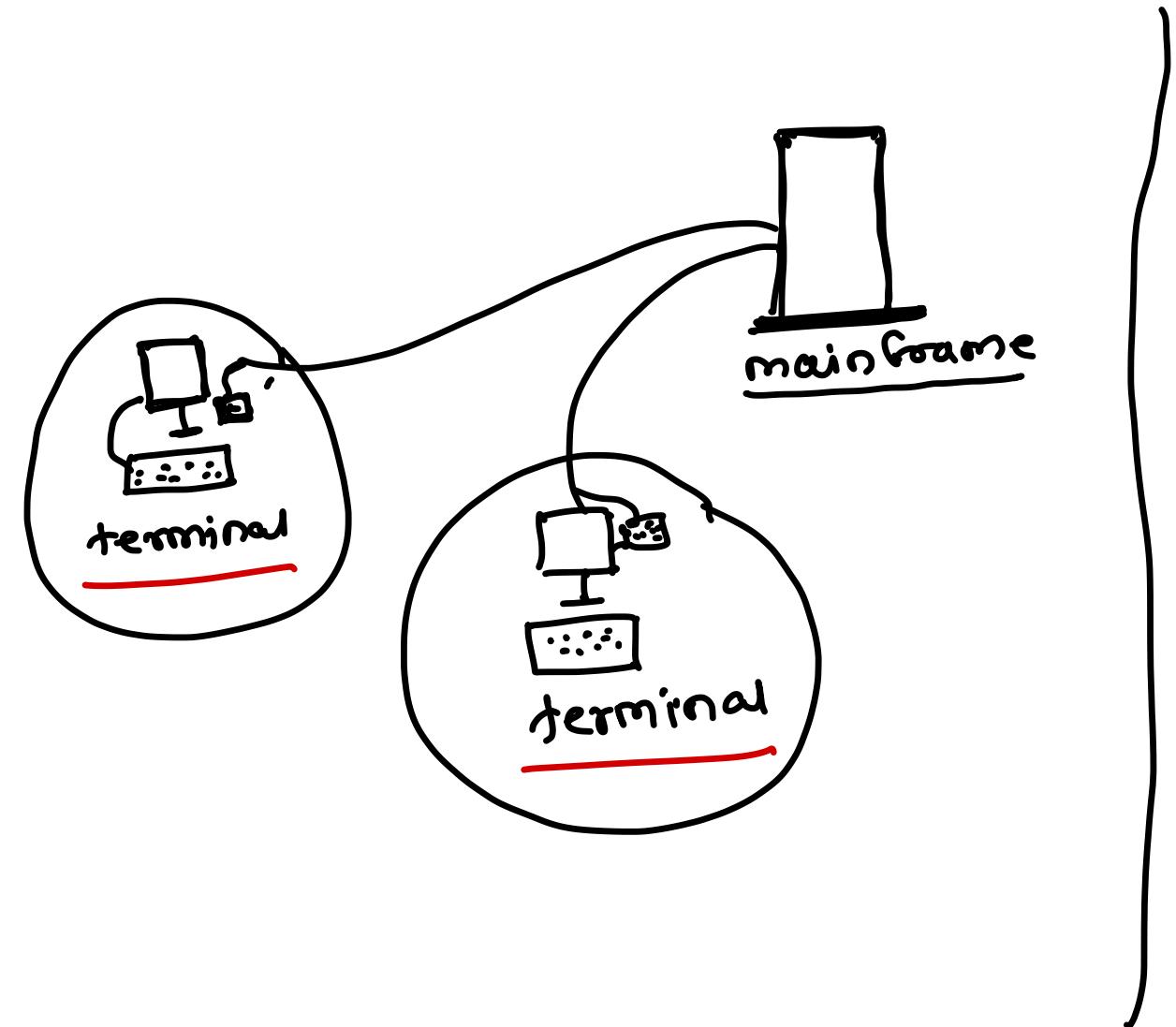


# Computing Models

---

- ✓ Desktop Computing ←
- ✓ Client-Server Computing ←
- ✓ Cluster Computing ←
- ✓ Grid Computing ←
- ✓ Cloud Computing ←

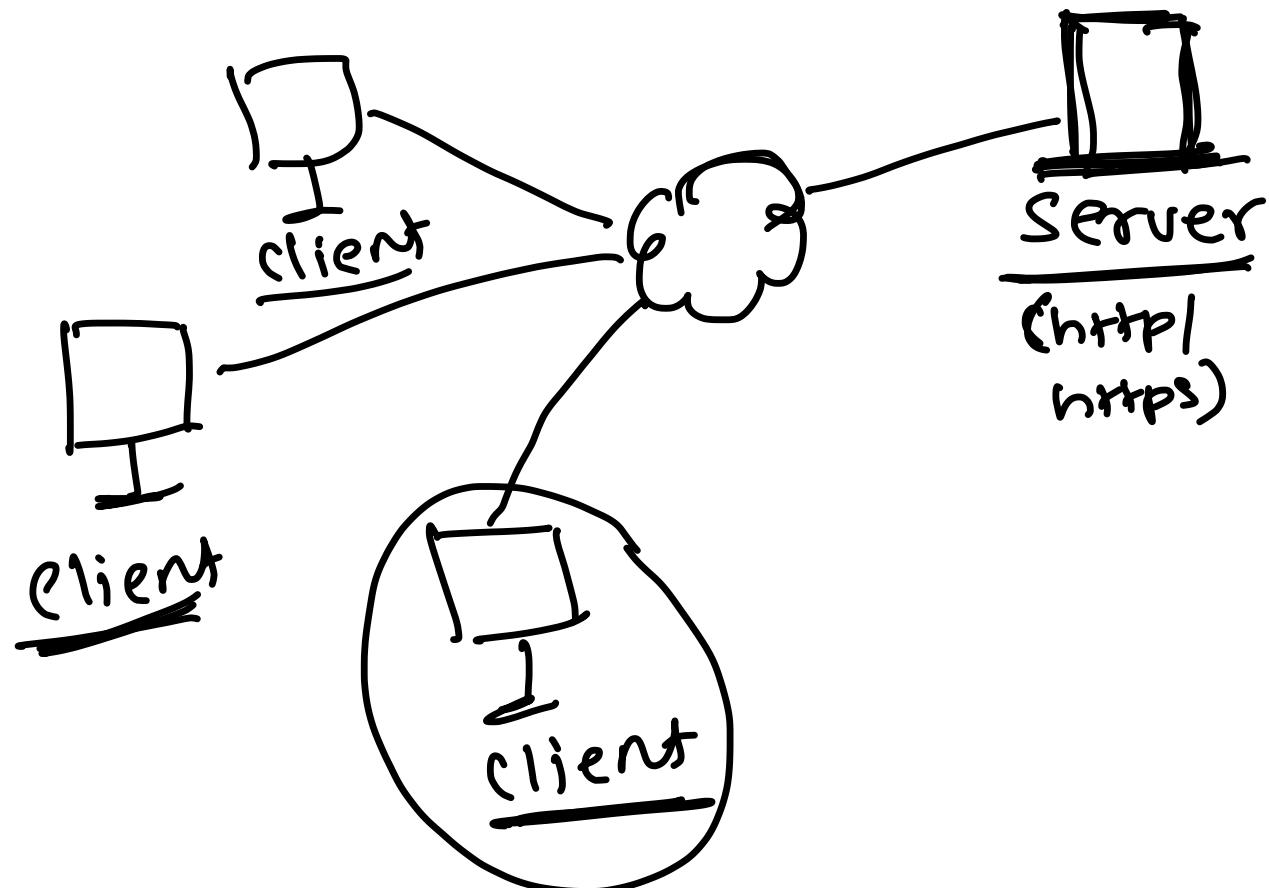
## desktop computing



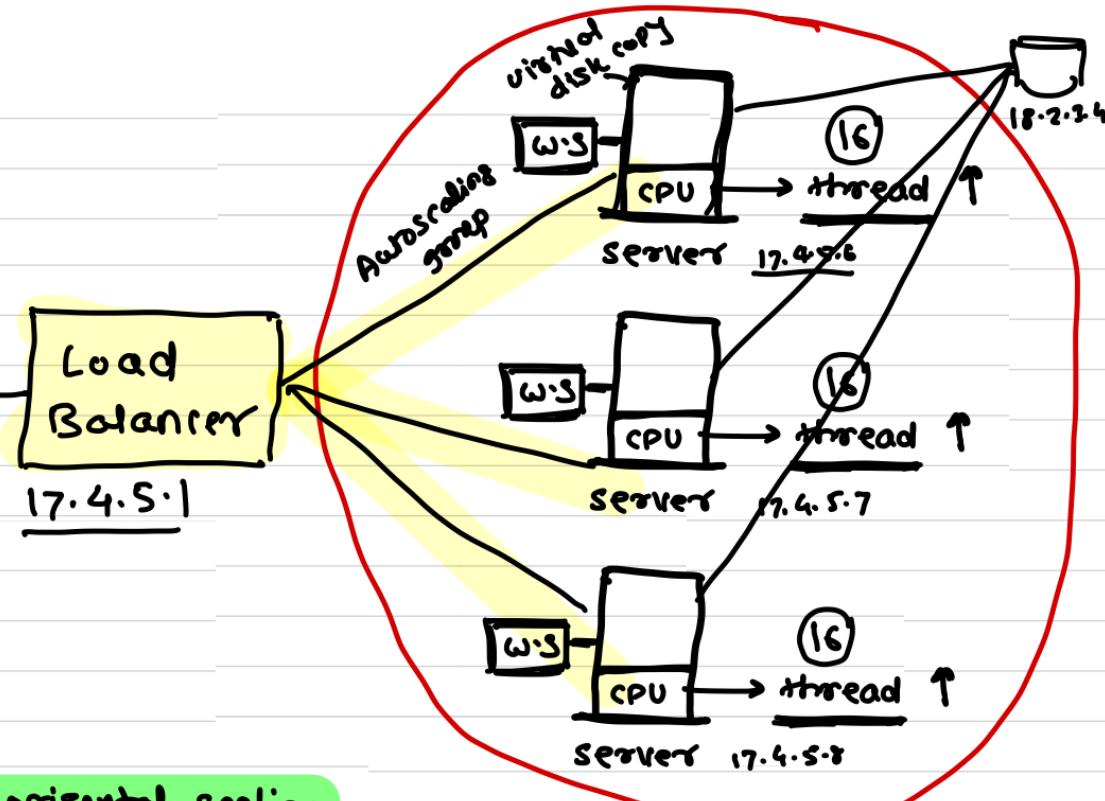
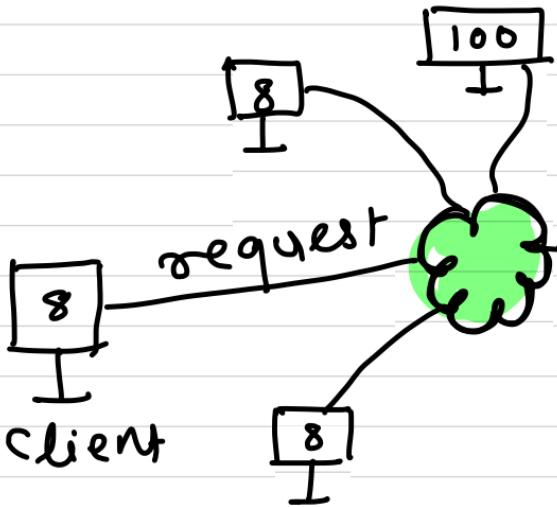
A hand-drawn diagram illustrating the compilation process. On the left, the text "hello.cpp" is underlined and followed by the word "source". An arrow points from this text to a box containing the letters "gcc". Another arrow points from the "gcc" box to the right. To the right of the "gcc" box, the text "hello.out" is underlined, followed by the words "executable" and "[ASM code]". Below that, the word "[CPU]" is written.

hello.cpp → **gcc** → hello.out  
source  
executable  
[ASM code]  
[CPU]

# client-server computing



# clusters



scaling problem

→ **Vertical scaling**

- scaling up
- upgrade configuration
- scaling down
- degrade configuration

→ **horizontal scaling**

- scale out
- create clones
- scale in
- terminate clones

## What is Cloud ?

- 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
- In addition, the platform provides on demand services, that are always on, anywhere, anytime and any place.
- Pay for use and as needed, elastic
  - scale up and down in capacity and functionalities
- The hardware and software services are available to
  - general public, enterprises, corporations and businesses markets etc

Internet

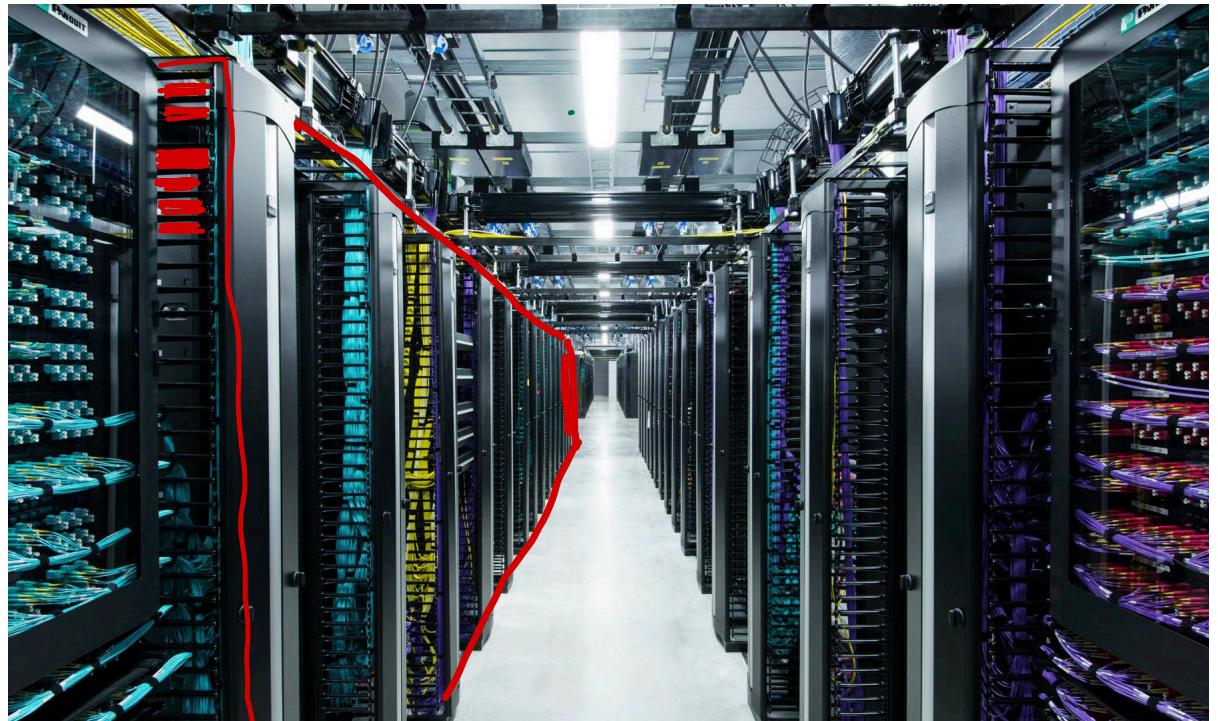
# Data Center

- 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 (compute)
- ✓ Security
- ✓ WAN
- ✓ Storage
- ✓ File Sharing

provider  
region  
availability zone  
data center  
building  
Floor  
row  
→ server rack  
↳ server

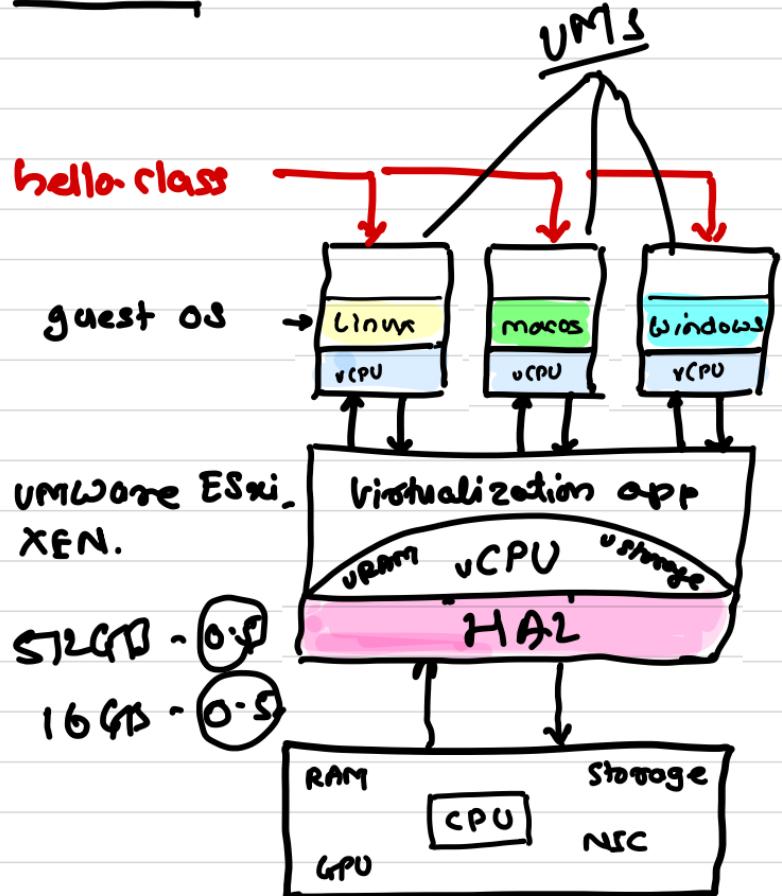
- 4 T.B. RAM
- 128 T.B. HD
- 128 cores & 4 GPU
- 9.6 cores & CPU



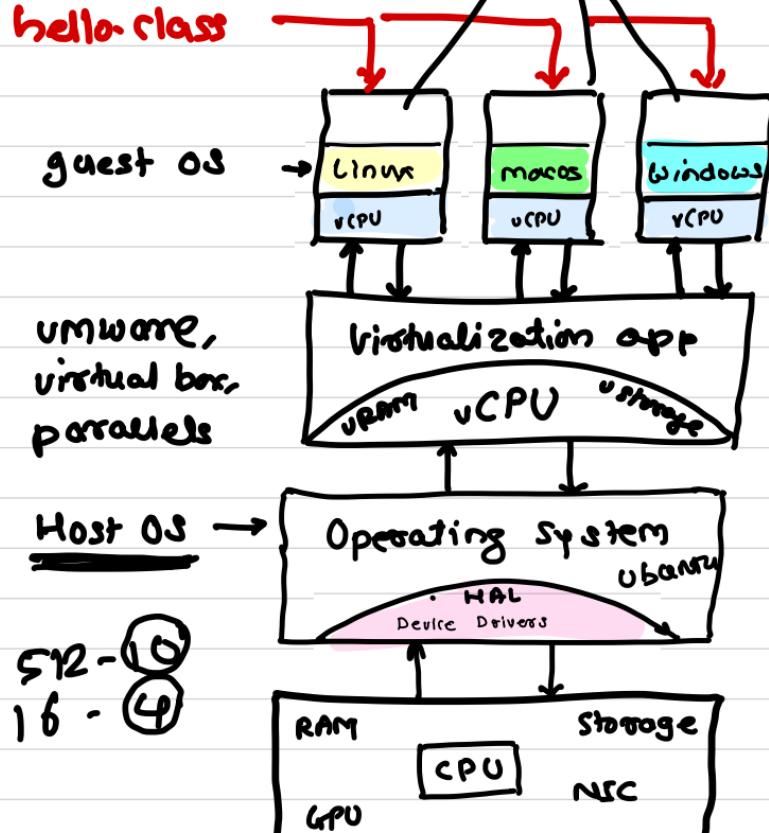
# Virtualization

- 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
  - Virtualization using VM
    - ✓ Type – I
    - ✓ Type – II
  - Containerization : Docker

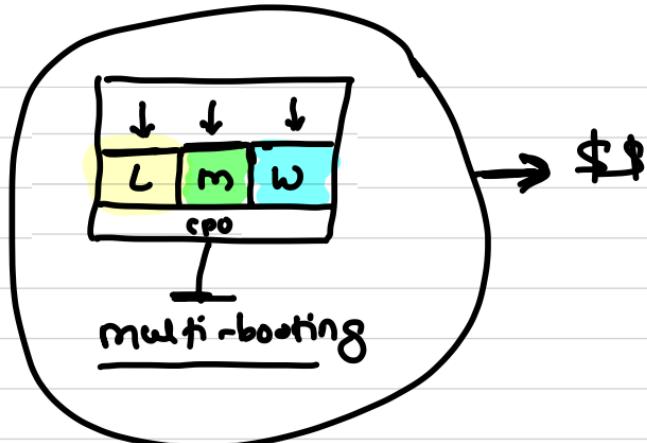
## Type I - cloud providers [AWS, GCP, Azure]



## Type II

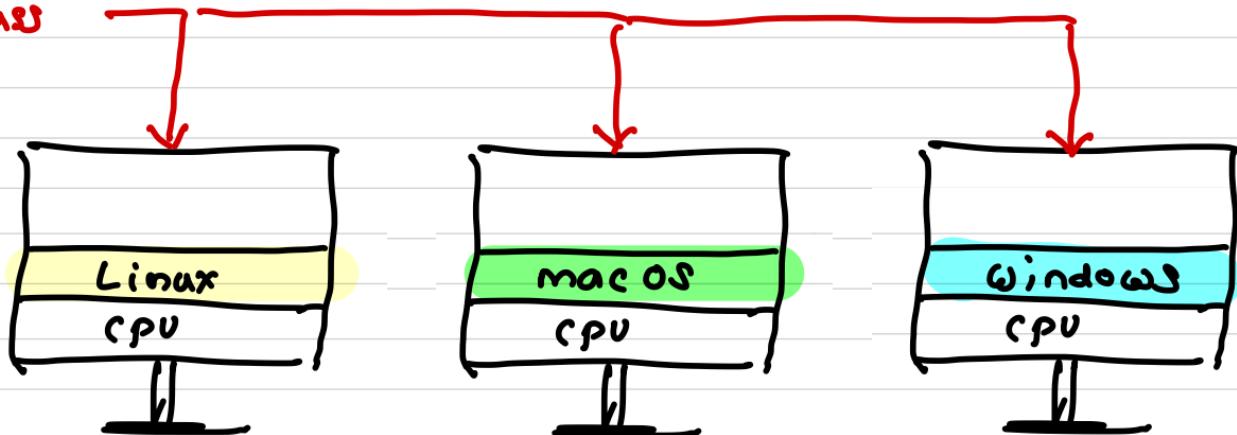


hello.java  
- javac  
↓  
hello.class



\$ \$

\$ \$ \$ \$



# Virtualization

---

# Cloud Computing Characteristics

## **On-demand self-service**



- A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider

## **Broad network access**

- Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms

## **Resource pooling** → Storage, compute, network, etc.. .

- The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand

## **Rapid elasticity**

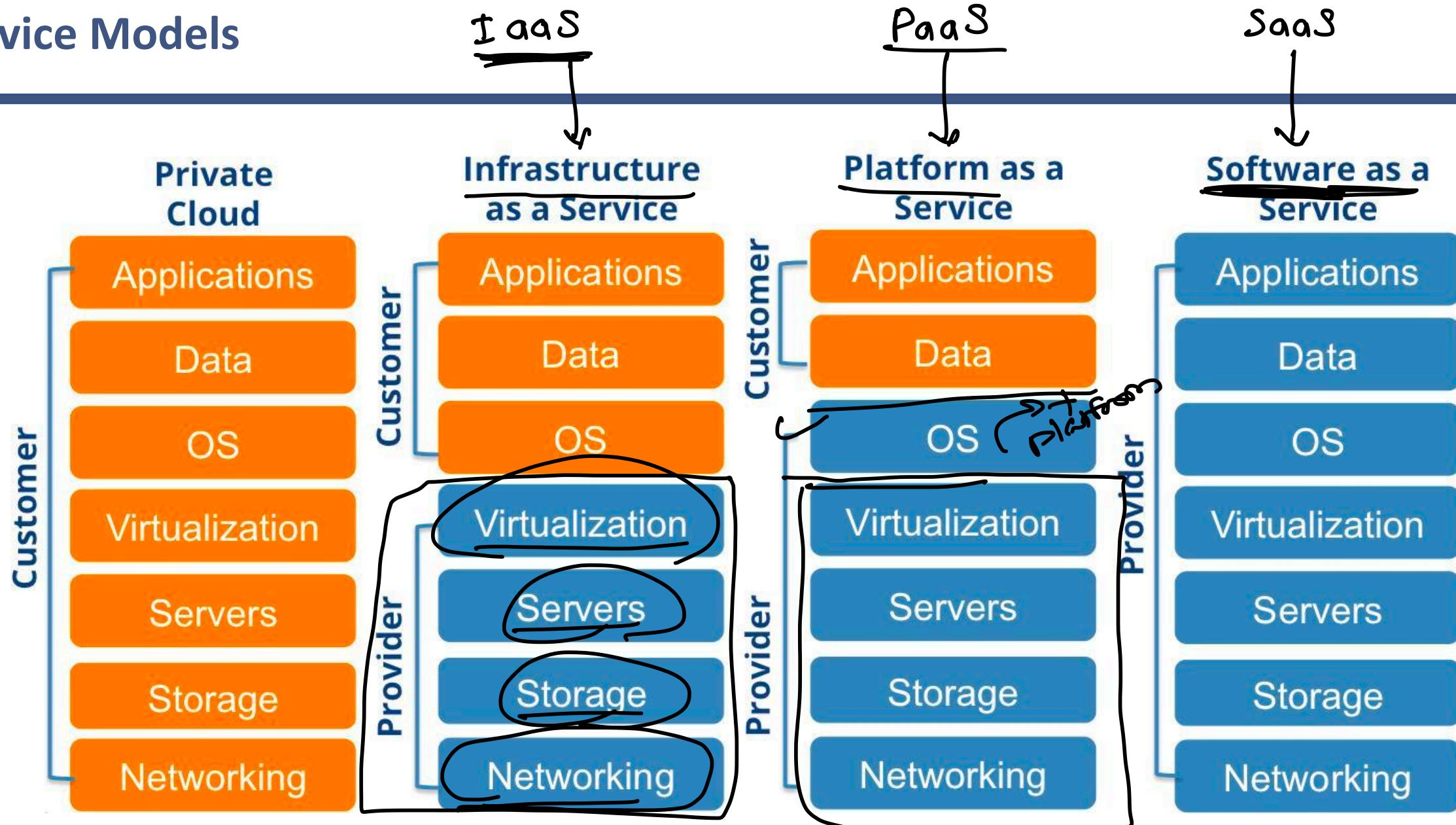
→ growing  
↳ shrinking

- Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand

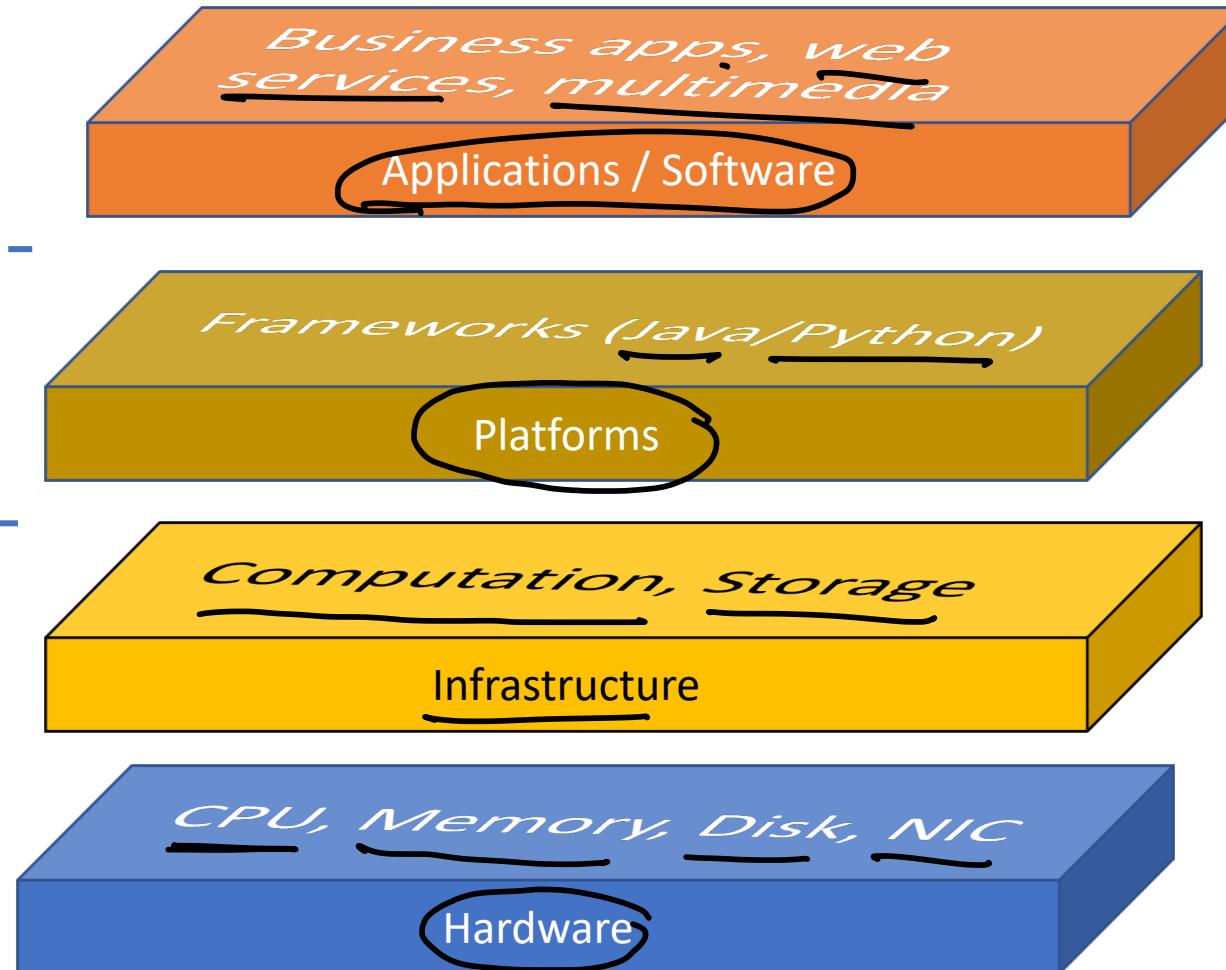
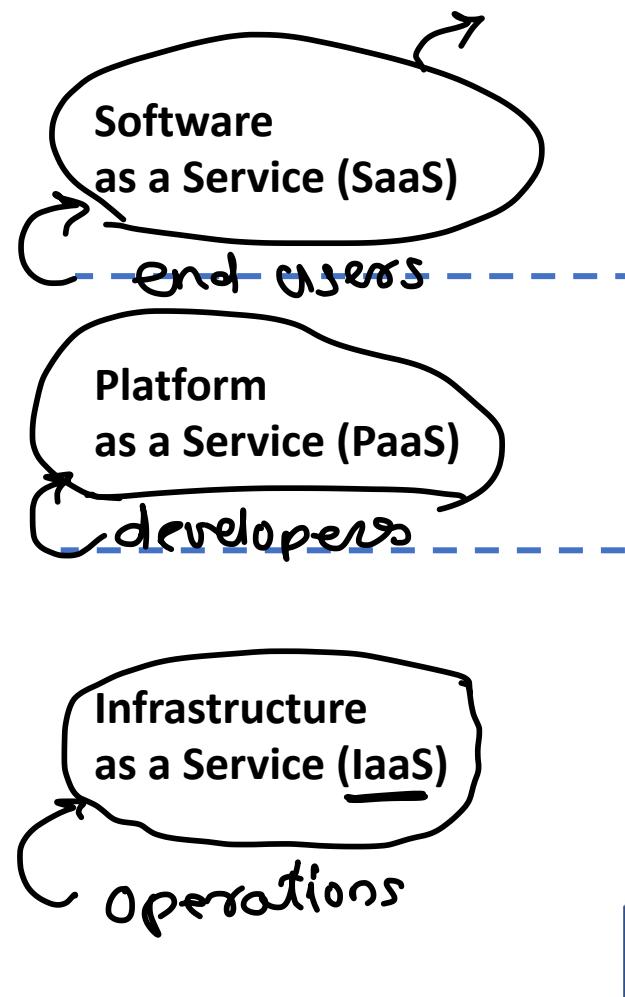
## **Measured service** - VM- 10 hrs , DB- 2 days

- Resource usage can be monitored, controlled and reported, providing transparency for the provider and consumer

# Service Models



# Service Models



Google Apps,  
Facebook, YouTube,  
Dropbox, Google Photos

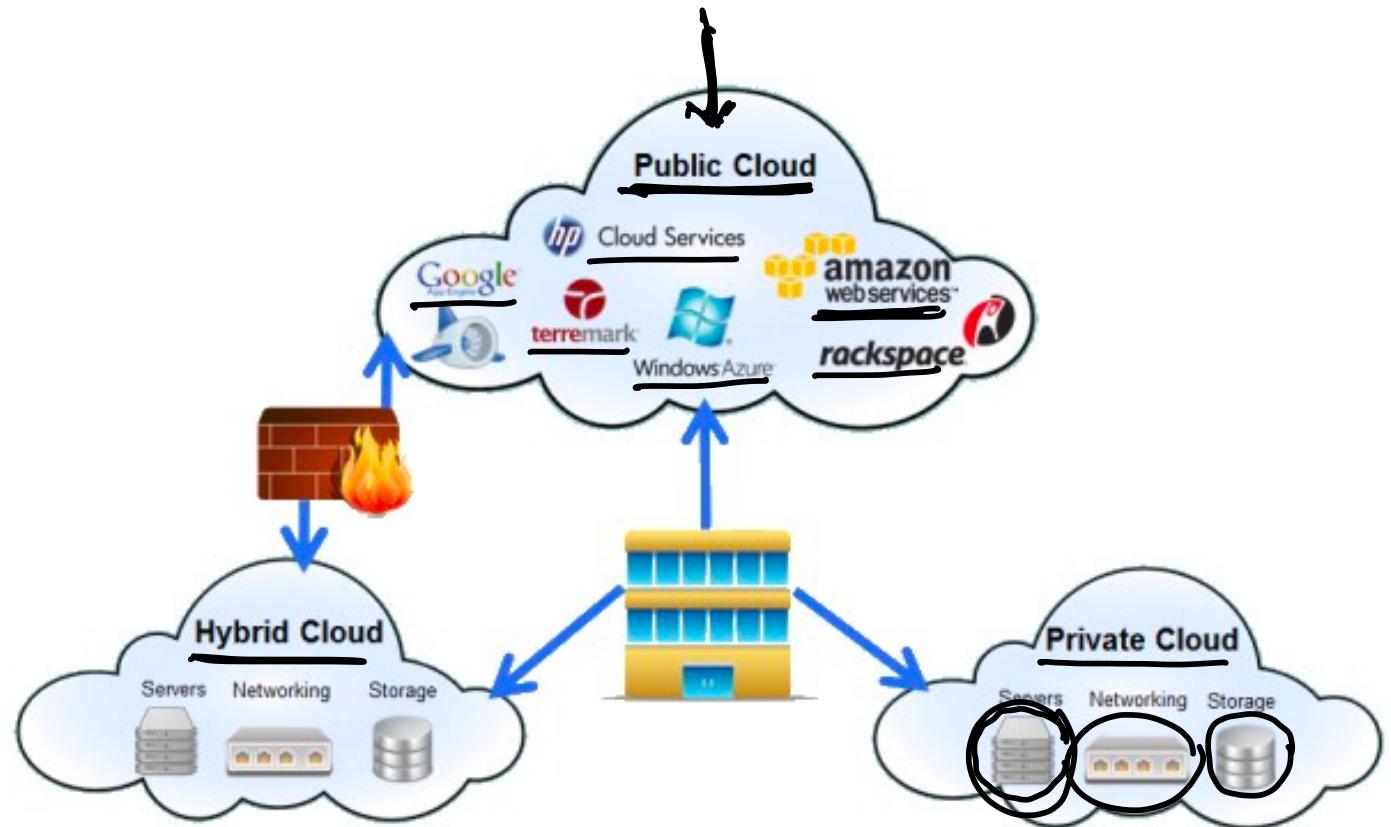
Google App Engine,  
Amazon Simple DB, S3,  
Microsoft Azure

Amazon EC2,  
Google Compute VM,  
Azure VM

Data Center

# Deployment models

- Private Cloud
- Public Cloud
- Hybrid Cloud



# Cloud Services

---

- Compute: used to create the Virtual Machine
- 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 → *internet*
- Improved document format compatibility
- Unlimited storage capacity
- Increased data reliability
- Universal document access
- Latest version availability

## Disadvantages

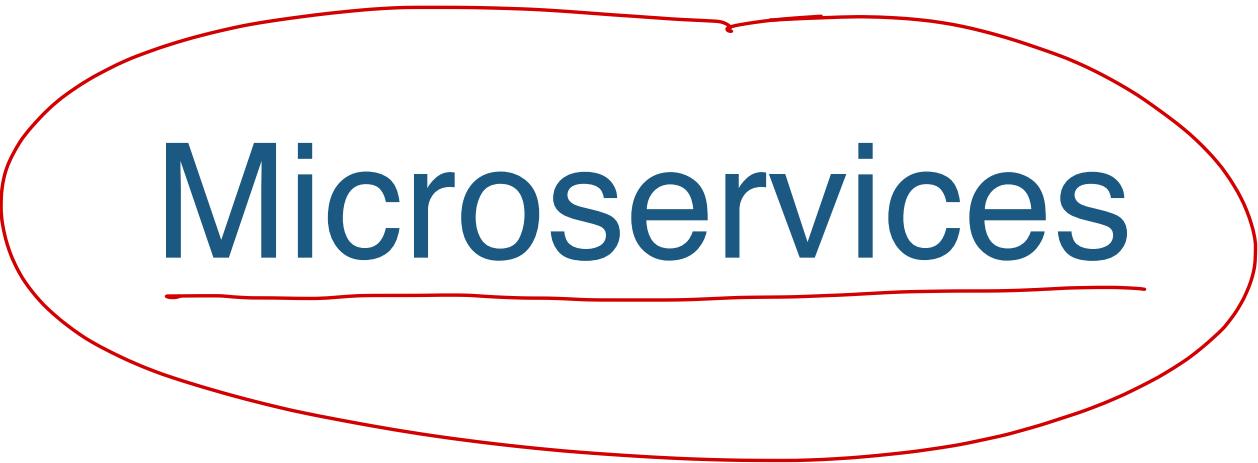
---

- 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 Vendors

---

- AWS →
- Google Compute →
- Azure →
- Others
  - Rackspace
  - digital ocean
  - linode
  - heroku



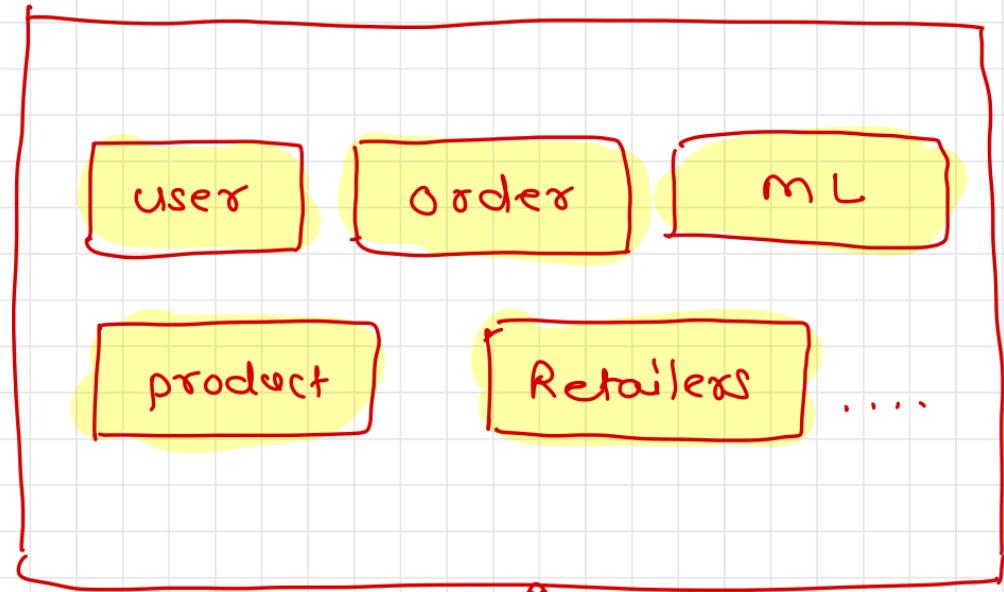
**Microservices**

## e-commerce

- user : signup, signing, profile ...
- order : listing, place, ...
- product : CRUD
- Machine learning : ..
- reports : ....
- retailers : ...
- administrators : ....
- : .

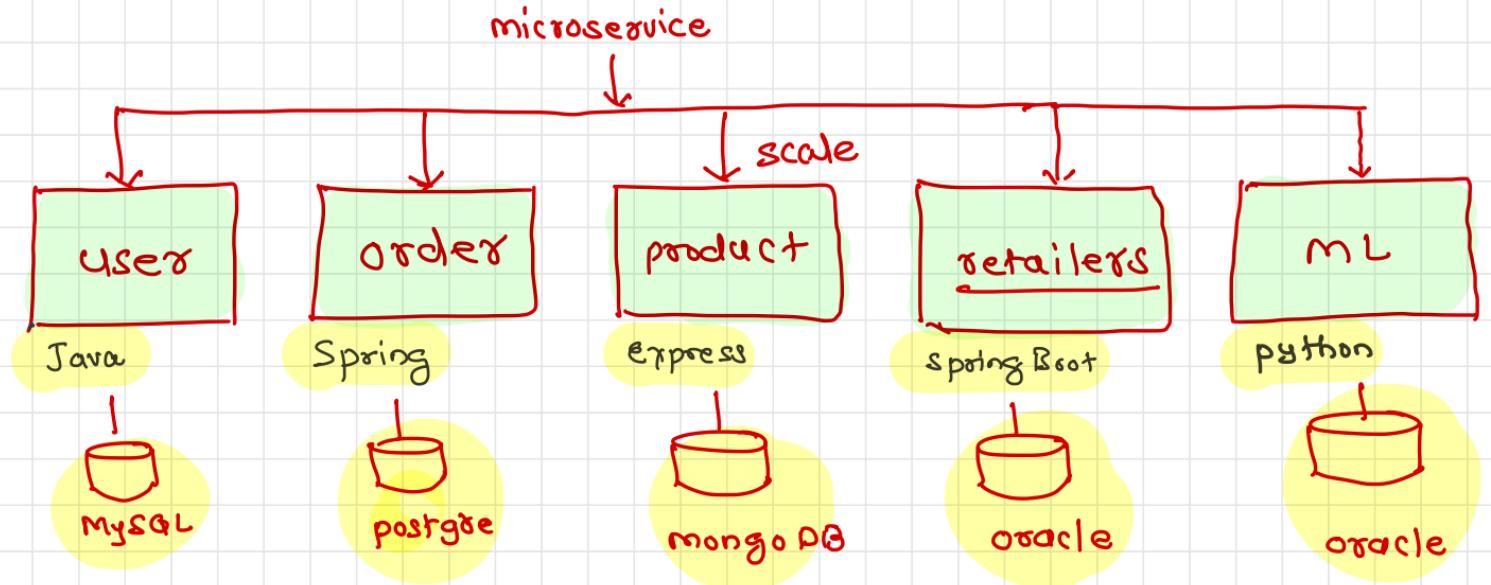


monolithic



## Technology stack

- Java
- Spring Boot
- REST
- MySQL



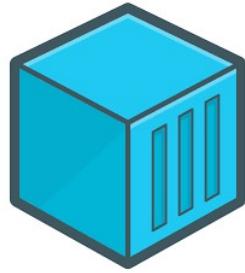
# Overview

---

- 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

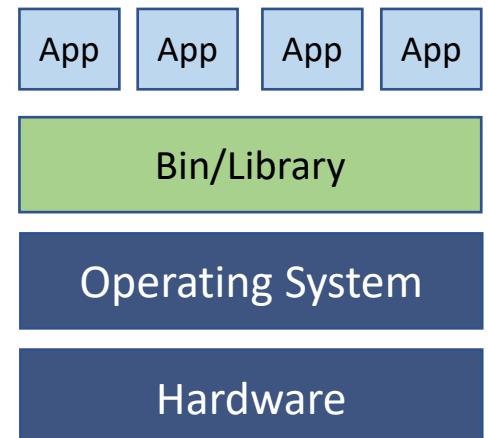


# Containerization



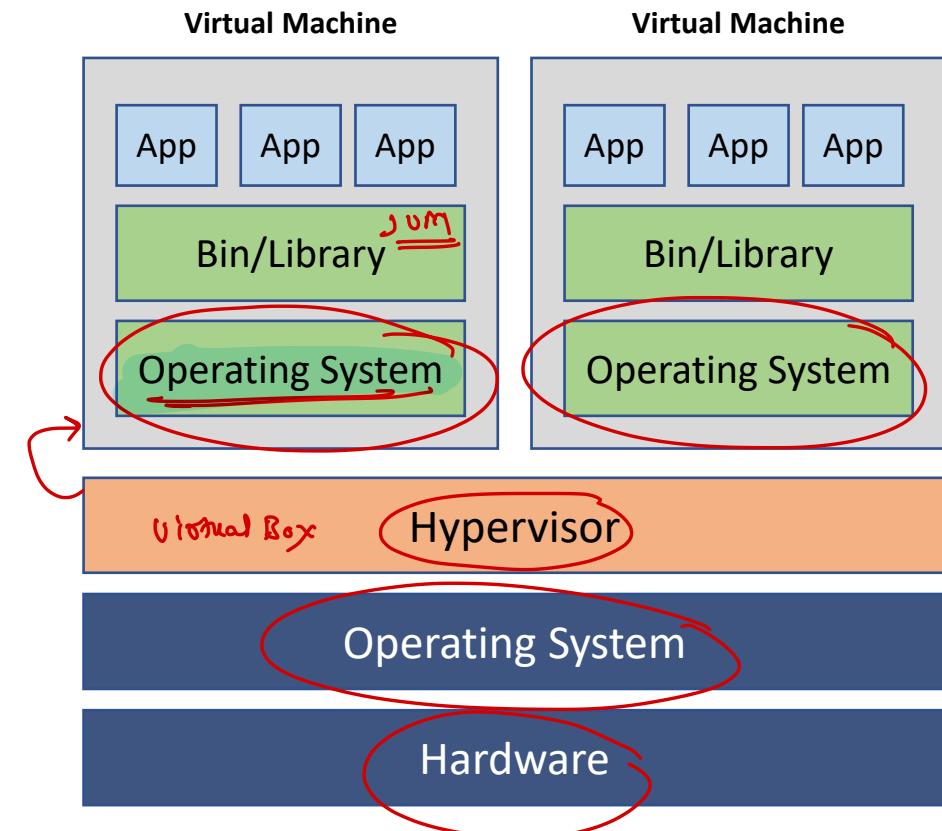
# Traditional Deployment

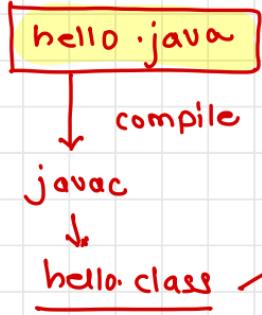
- Early on, organizations ran applications on physical servers.
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application 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

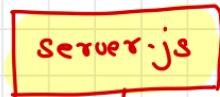
- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application
- Virtualization allows better 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





javac

hello.class



1

## Operating system

- HAL : → Hardware Abstraction Layer
  - ↳ collection of device drivers



utilities

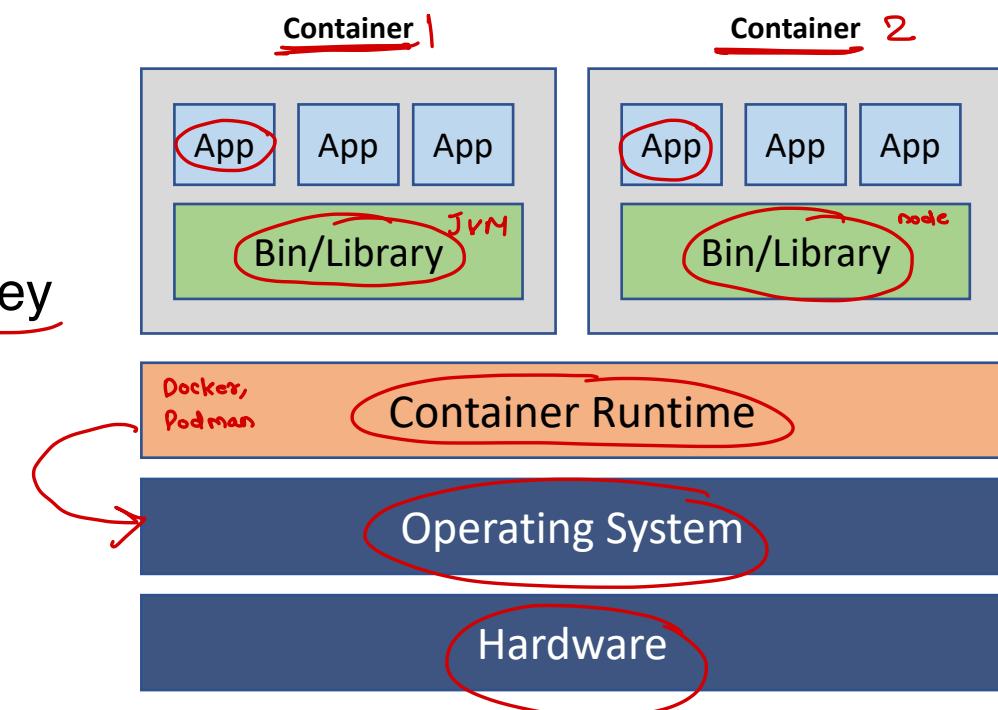
HAL

system files

....

# 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



# 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

<u>Virtual Machine</u>	<u>Container</u>
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

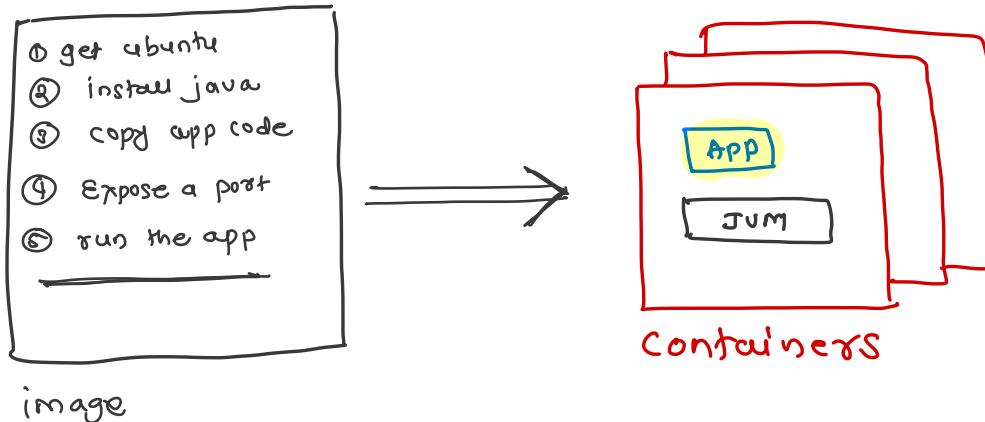
# Overview

---

- 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 Paas provider called as dotCloud which was using the Linux Containers behind the screen to run containers
- In 2013, the dotCloud became Docker Inc
- It comes in two editions
  - Enterprise Edition (EE) ~~\$ \$\$~~
  - Community Edition (CE) ~~free~~

# Images

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



# Commands related to images

---

- **List all the images**

> docker image ls

- **Download an image from docker hub**

> docker image pull <image name>

- **Get the details of selected image**

> docker image inspect <image name>

- **Delete an image**

> docker image rm <image name>

# Commands related to images

---

- **Push the image to docker hub**

> docker image push <image name>

- **Tag an image**

> docker image tag <image name> <tag>

- **Build an image with Dockerfile**

> docker image build <Dockerfile>

## Containers

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

# Commands related to containers

---

- **List the running containers**

> docker container ls

- **List all the containers (including stopped)**

> docker container ls -a

- **Create and start container**

> docker container run <image>

- **Start a stopped/created container**

> docker container start <container>

# Commands related to containers

---

- **Stop a container**

> docker container stop <container>

- **Remove a container**

> docker container rm <container>

- **Get the stats of selected container**

> docker container stats <container>

- **Execute a command in a container**

> docker container exec <container>

## Commands related to containers

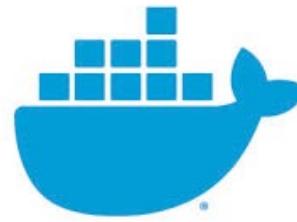
---

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

> docker container commit <container>

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

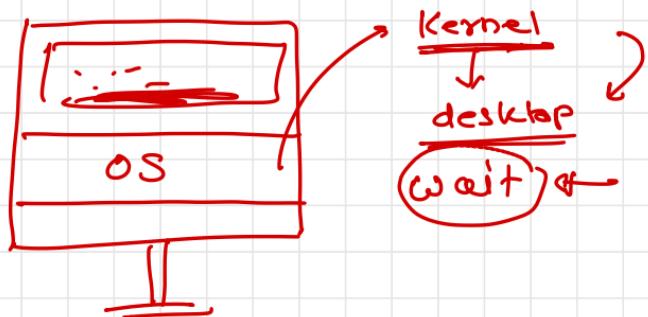
> docker container top <container>



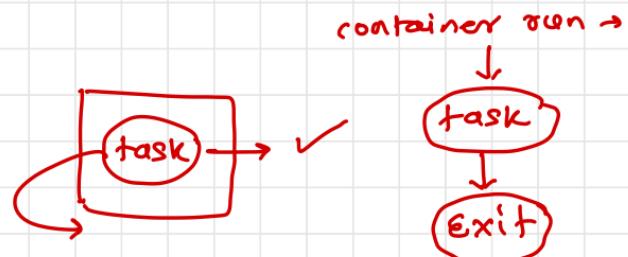
# Docker Compose

---

## multi-function



## single-function



docker run <image>

↳ creates a container [process]

↳ starts the job in attached mode

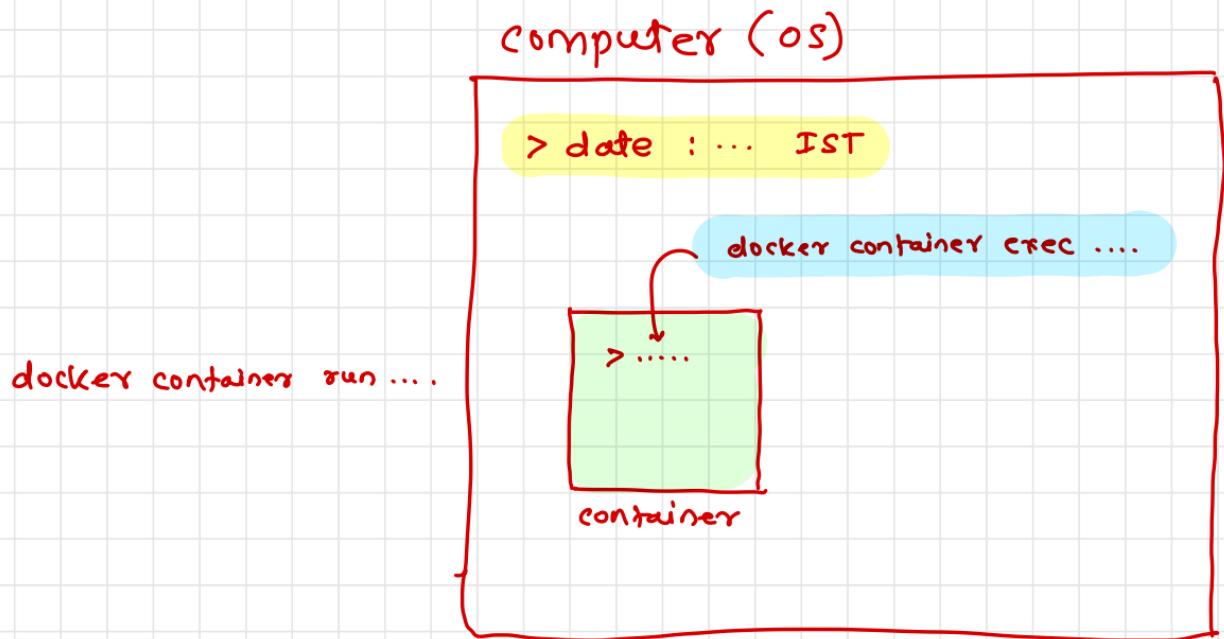
## modes

### attached

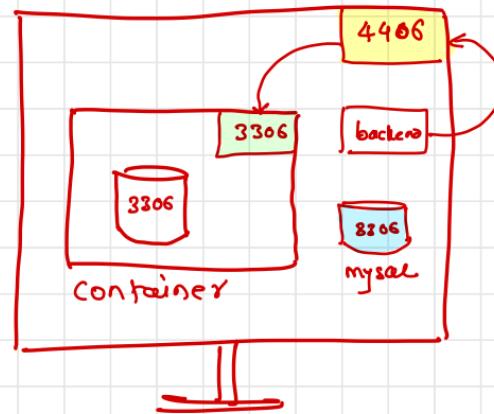
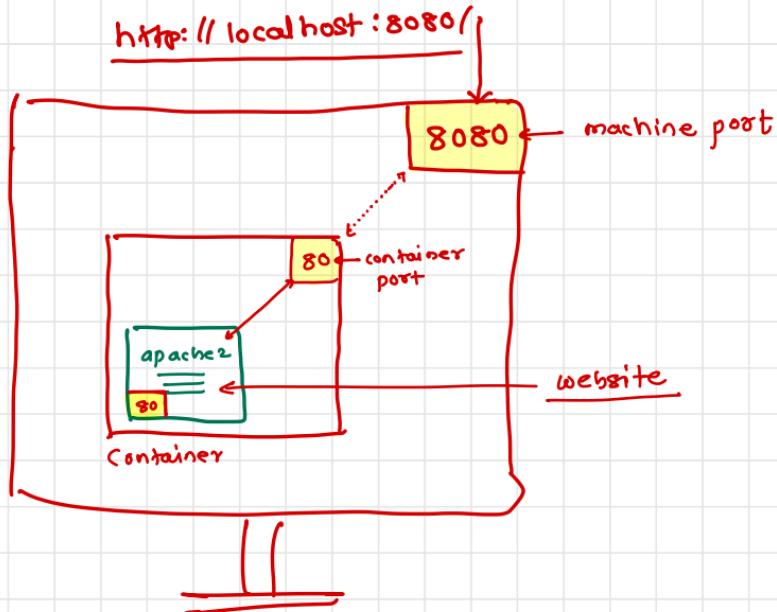
- task starts in foreground
- & captures the terminal
- by default

### detached

- container runs in background
- two ways
  - start already stopped container
  - sleep in



## Port forwarding



# 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

*creating images*

*creating containers*

## 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"
```

docker-compose.yml



## Build and run the application

---

- To run the application use

> docker-compose up *-d*

- To stop the containers

> docker-compose stop

- To remove the containers

> docker-compose down

> *docker-compose build*



**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

(GNU) → GNU is not unix

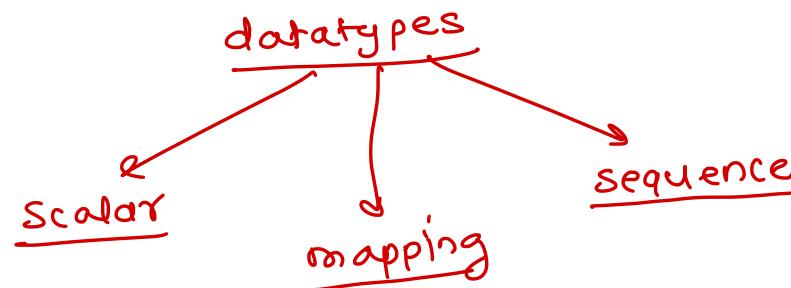
## Features

---

- 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

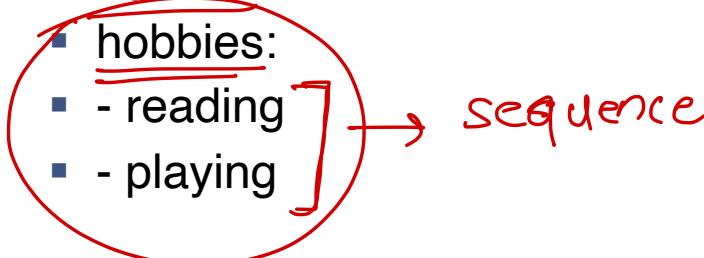
- 4.5

- 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

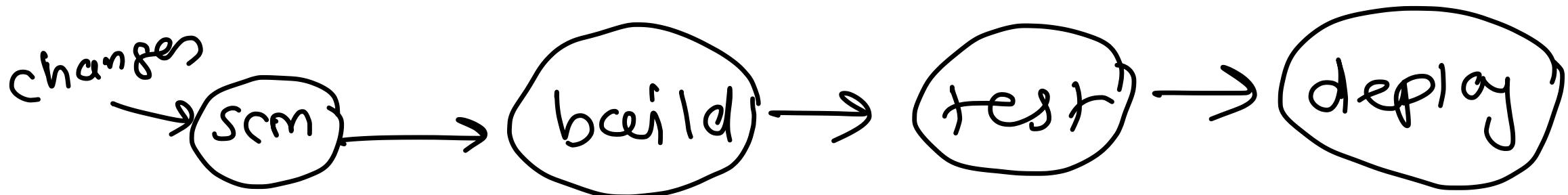
- E.g.

- products:
    - - title: product 1
    - price: 100
    - description: good product
    - - title: product 2
    - price: 300
    - description: useful product

# 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
- The main aim of CI is to prevent manual integration problem
- CI requires developers to have frequent builds
- The common practice is that whenever a code commit occurs, a build should be triggered



# 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

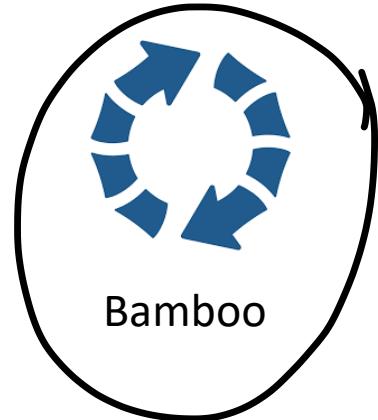
---



Jenkins



TeamCity



Bamboo



GitLab CI



Travis CI



Jenkins

## Overview

- Jenkins is a powerful application that allows continuous integration and continuous delivery of projects
- It is a free source that can handle any kind of build or continuous integration
- It was first started as project Hudson at Sun Microsystems in 2004 and was first released in Feb 2005
- In 2011, Oracle created fork of Hudson as Jenkins, since when these two projects exist as two independent projects



# Why Jenkins?

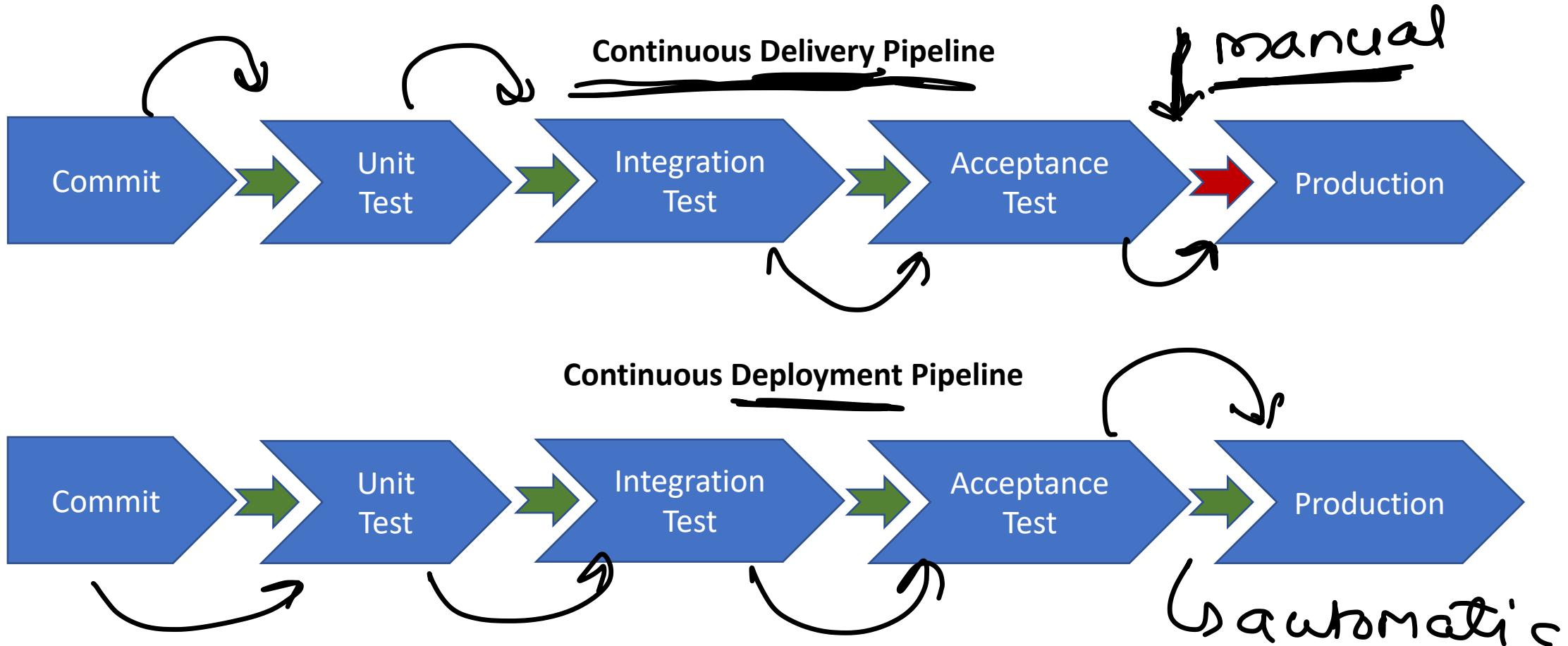
---

- It is open source and free
- It has got plugin support
- It has a huge community
- It is fast and reliable
- It has good OS support
- It supports scripted build

Linux  
macOS  
Windows



# CI/CD Pipeline



# Jenkins Projects

---

- Freestyle
  - Used to build the project with any SCM and any build system
- Pipeline
  - Suitable for building pipelines or organizing complex activities that do not easily fit in the free style
- Multi configuration
  - Suitable for projects that need large number of different configurations
- Folder
  - Used to create containers (folders) to organize the jobs
- GitHub Organization
  - Used for GitHub projects



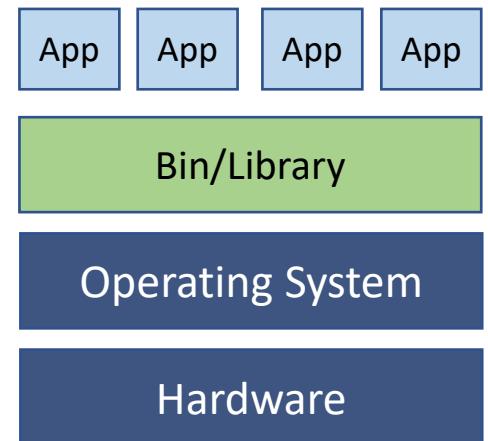
# What is Kubernetes ?

---

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

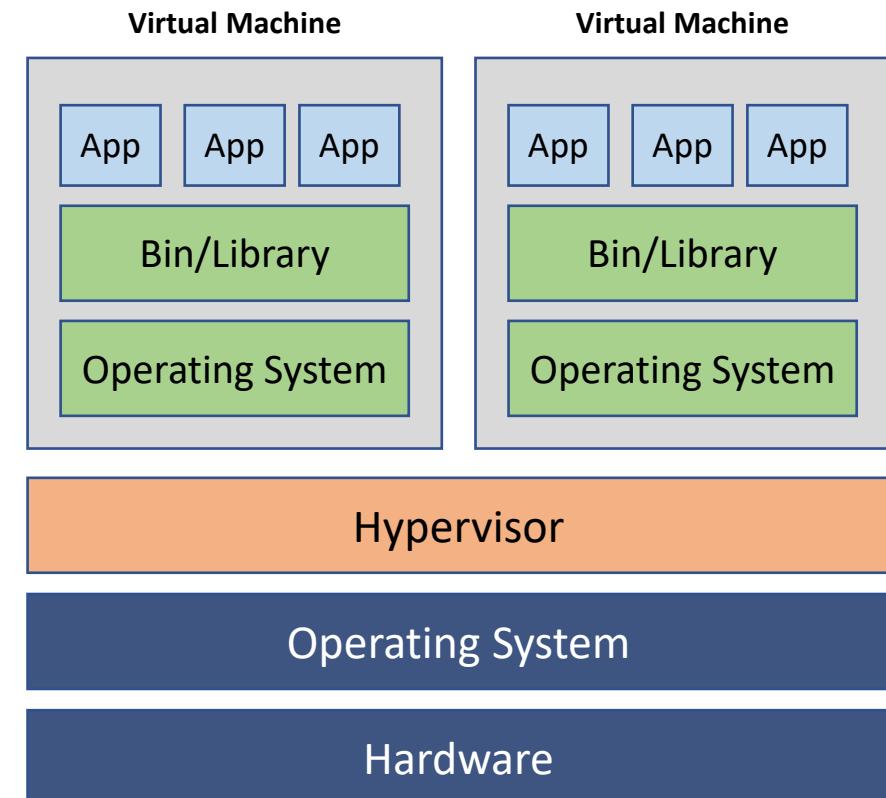
# Traditional Deployment

- 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



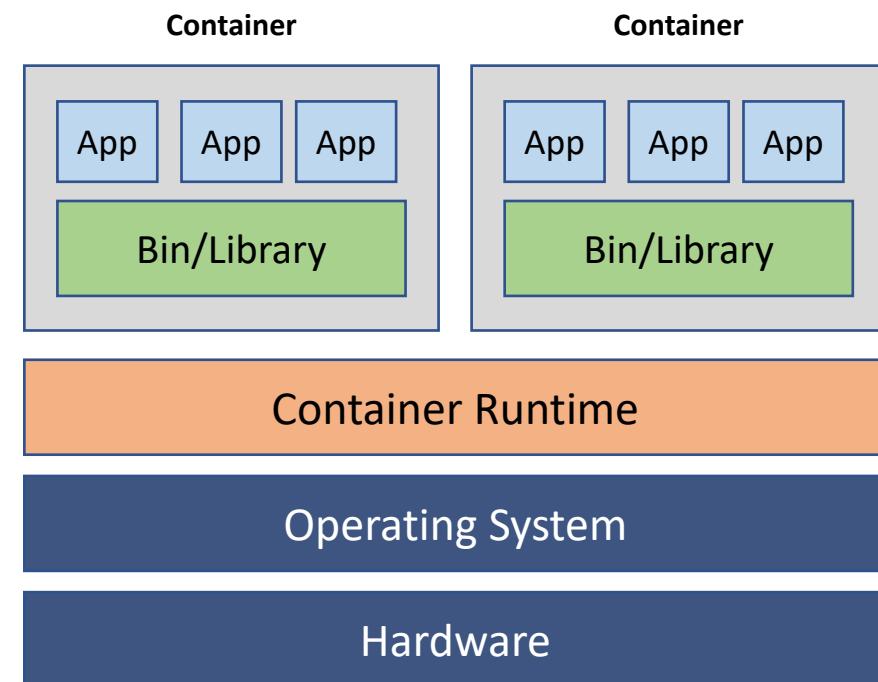
# Virtualized Deployment

- 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 *& definition in yaml*
- 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



✓ restarted, killed, recreated

## 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

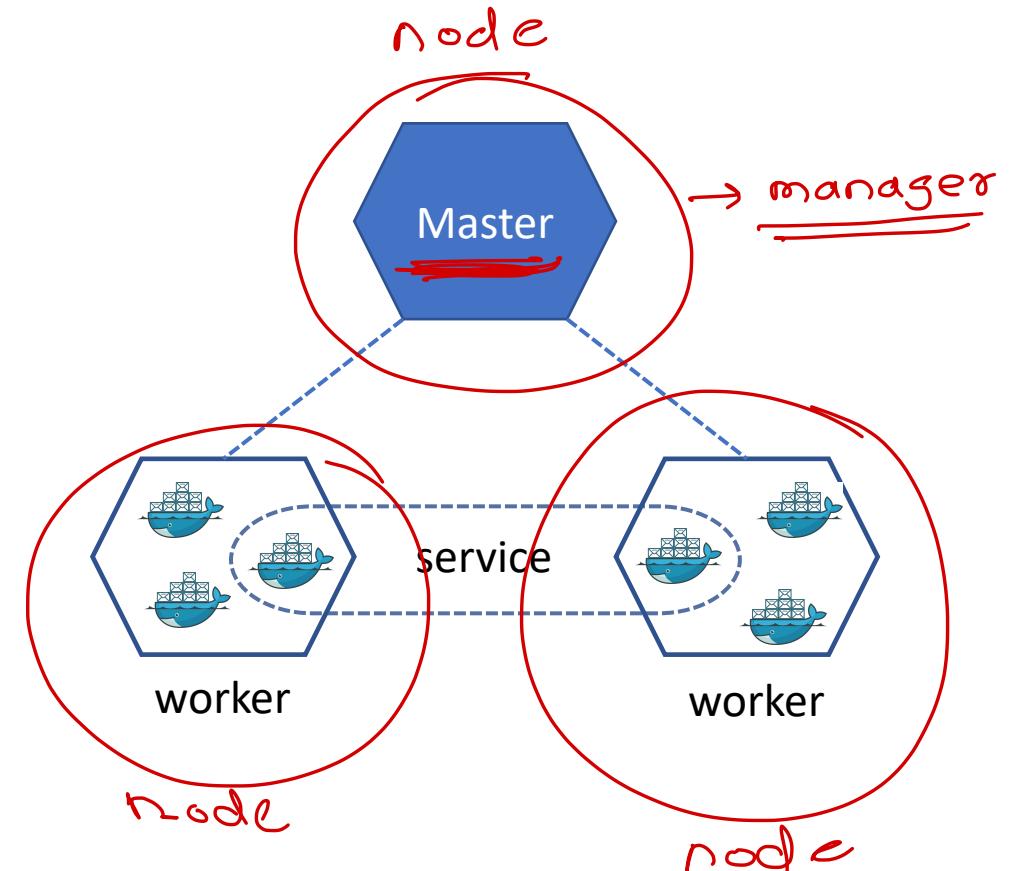
## What Kubernetes is not

- Does not limit the types of applications supported  
*jenkins*
- Does not deploy source code and does not build your application  
*gradle/maven*
- 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

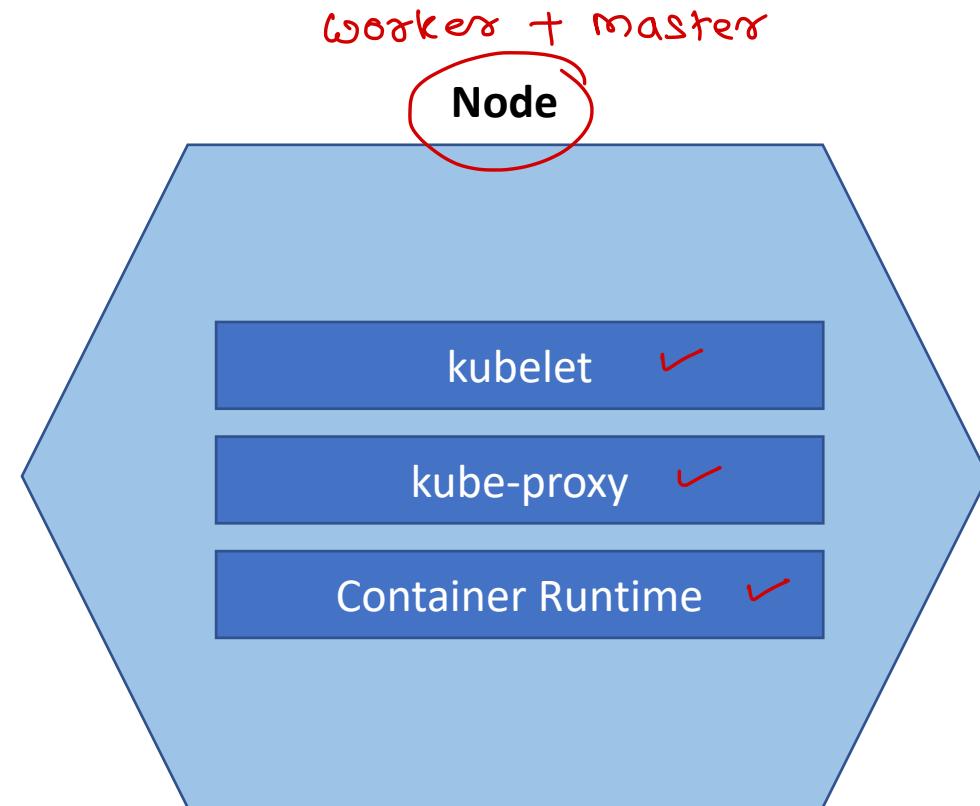
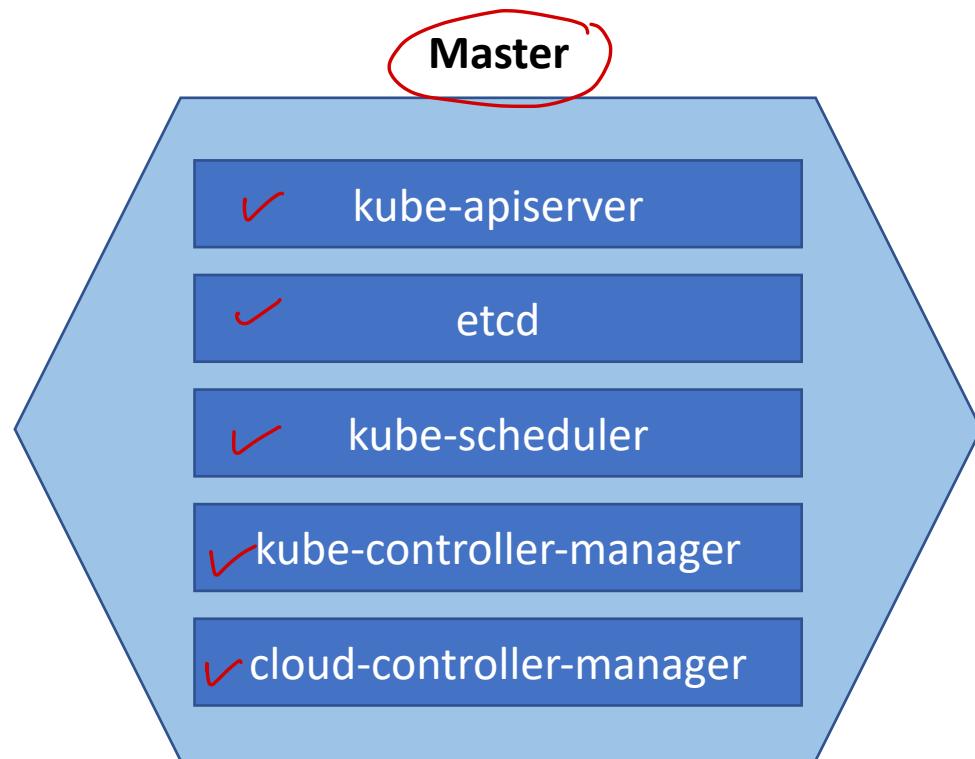
# Kubernetes Cluster

minikube

- 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 Components



# Master Components

- Master components make **global decisions** about the and they detect and respond to cluster events
- Master components can be run on any machine in the cluster
- **kube-apiserver**
  - The API server is a component that exposes the Kubernetes API
  - The API server is the **front end** for the Kubernetes
- **etcd**
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
- **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

worker + master

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

## Create Cluster

---

- Use following commands on both master and worker nodes

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

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

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

```
> sudo apt-get update
```

```
> sudo apt-get install -y kubelet kubeadm kubectl
```

```
> sudo apt-mark hold kubelet kubeadm kubectl
```

# Initialize Cluster Master Node

---

- Execute following commands on master node

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

```
> mkdir -p $HOME/.kube
```

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

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

- Install pod network add-on

```
> kubectl apply -f
```

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

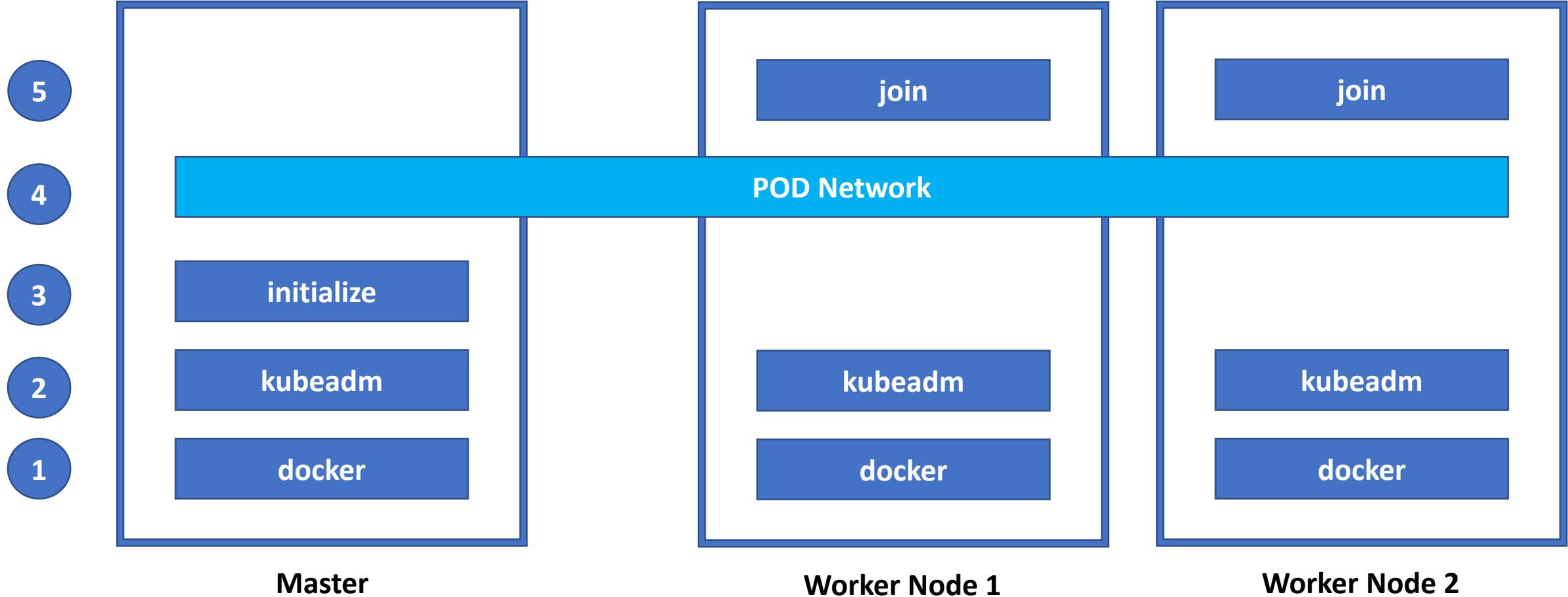
## Add worker nodes

---

- Execute following command on every worker node

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

# Steps to install Kubernetes



# Kubernetes Objects

---

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

# Pod

---

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

## YAML to create Pod

---

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

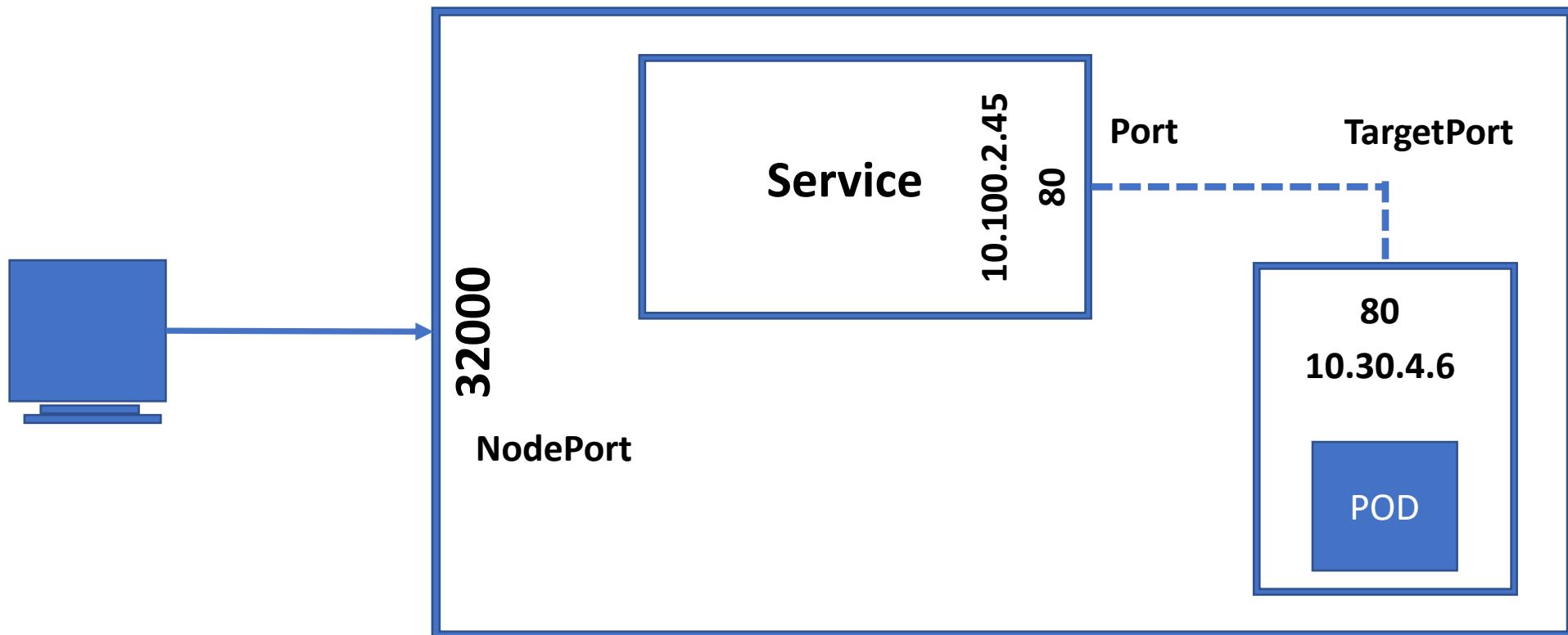
# Service

- 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
  - ClusterIP
    - Exposes the Service on a cluster-internal IP
    - Choosing this value makes the Service only reachable from within the cluster
  - LoadBalancer
    - Used for load balancing the containers
  - NodePort

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

## Service Type: NodePort

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



# Replication Controller

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

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

# Deployment

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

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

# Volume

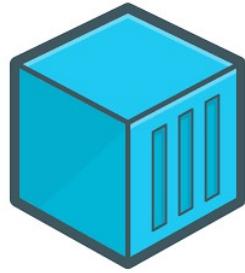
---

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

# Namespace

---

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



# Container Orchestration

→ docker swarm  
→ kubernetes

A red bracket on the left side of the slide groups the two items listed below it, pointing towards them from above. The bracket is drawn with a thick red line and has a small red arrowhead at each end pointing down towards the text.

# Overview

- Container orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments
- 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
- 

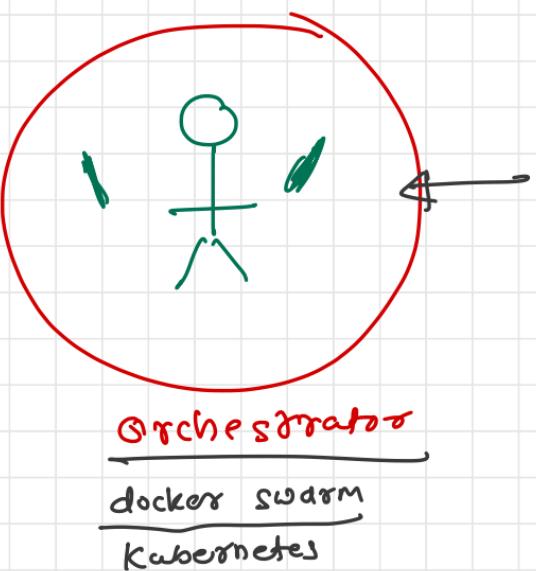
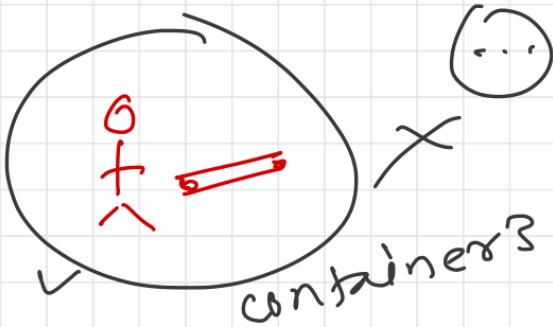
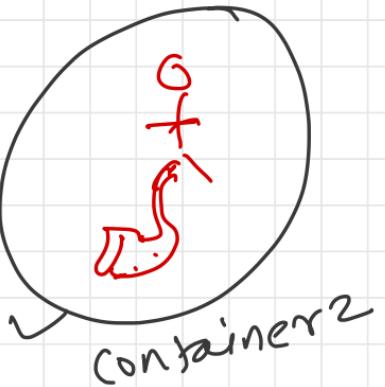
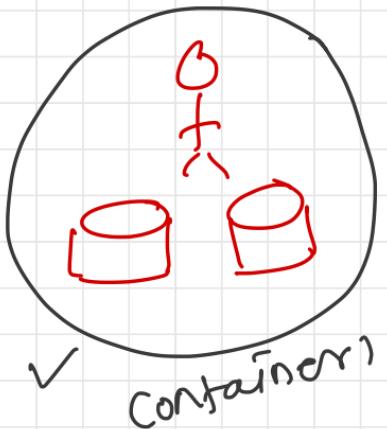


Docker Swarm

## 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



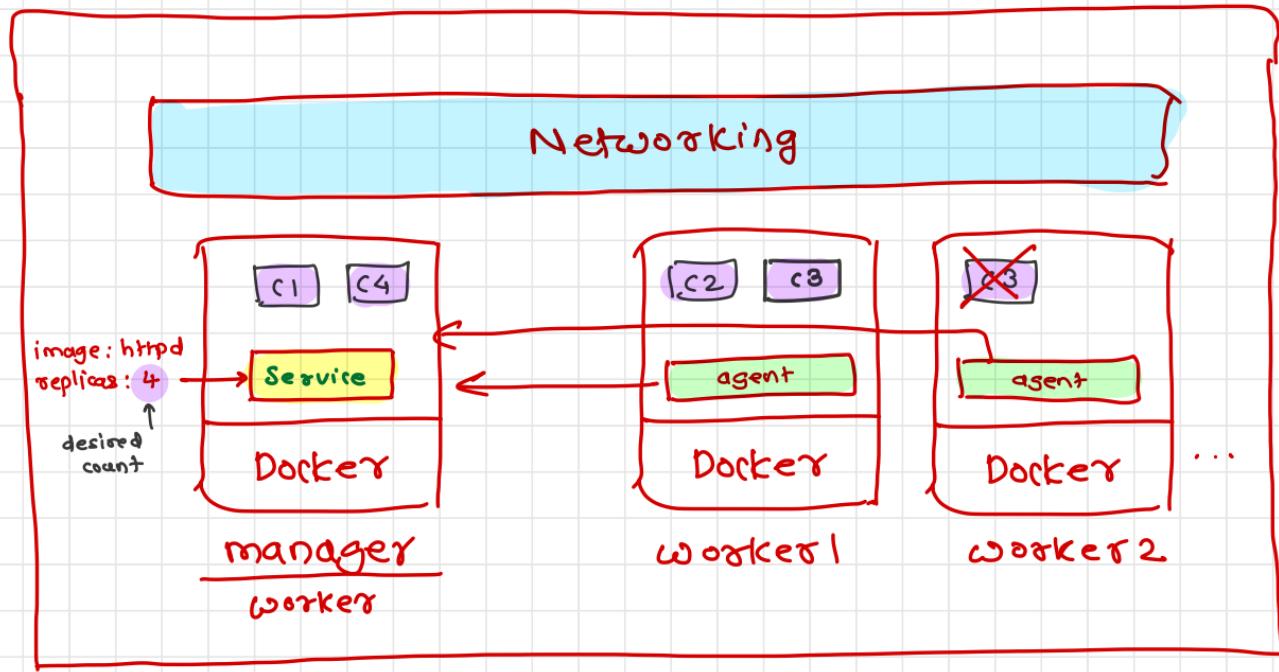
# What is a swarm?

- 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 *replicas = 5*
- Docker works to maintain that desired state
  - For instance, if a worker node becomes unavailable, Docker schedules that node's tasks on other nodes
- A **task** is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container
- When Docker is running in swarm mode, you can still run standalone containers on any of the Docker hosts participating in the swarm, as well as swarm services
- A key difference between standalone containers and swarm services is that only swarm managers can manage a swarm, while standalone containers can be started on any daemon

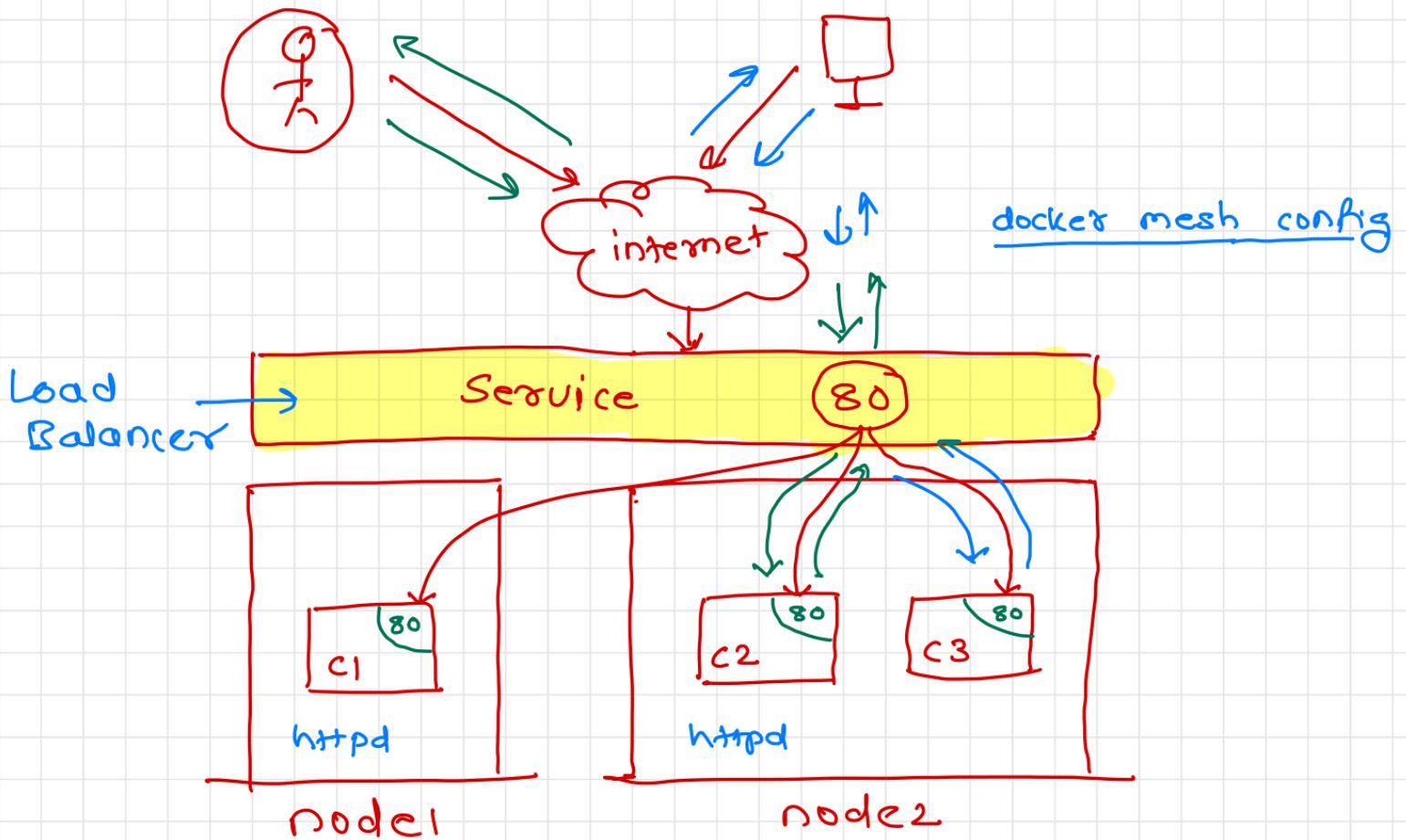
# Features

---

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



cluster (swarm)



# 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>