



Software Development

Methodologies



Contents

principles + methodologies

Software Engineering

typeI / typeII + services + providers
virtualization + characteristics + types
aws

Cloud Computing

CI + delivery + deployment + jenkins

CI/CD Pipeline

Scrum

AGILE

models = waterfall + iterative

SDLC

docker, kubernetes, jenkins, maven,
git, ...
benefits + life cycle + tool chains

DevOps Introduction

horizontal scaling (volumes, ports)
benefits + docker (architecture, commands)

Containerization

types + methods + levels

Application Testing

lives + cucs + ducs
vcs + types + git (basics + architecture)
+ branching

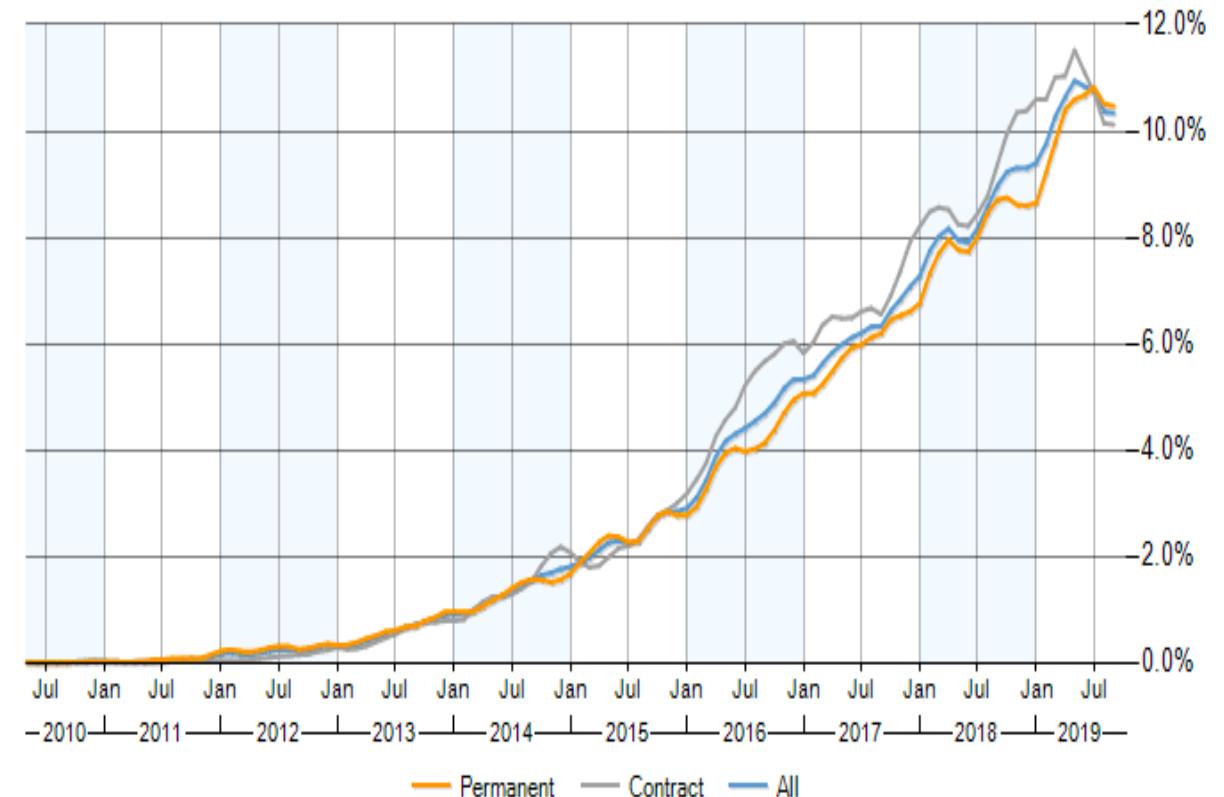
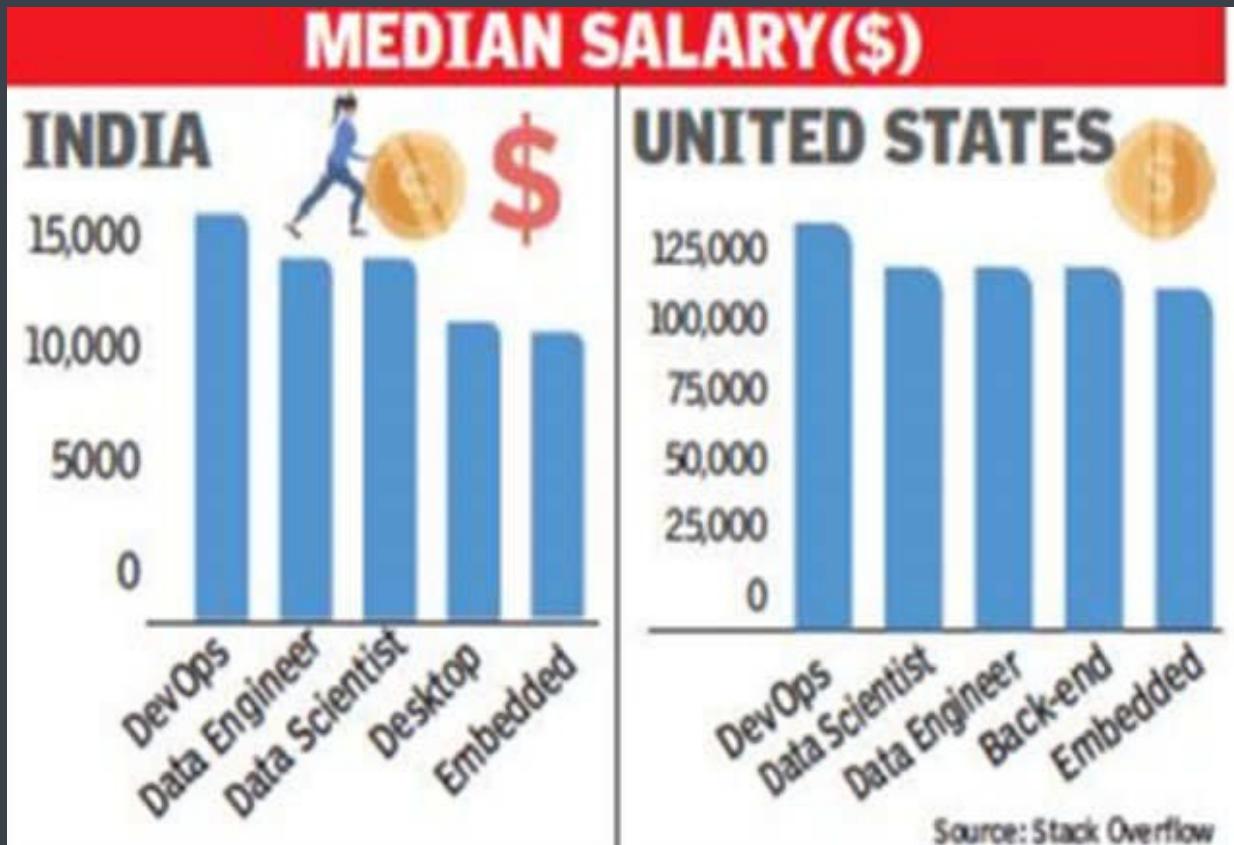
SCM

docker swarm + kubernetes

Container Orchestration



Why should you bother about DevOps?





Pre-requisites

- Basic Linux knowledge
- Any Development Platforms
- Any language (preferred JS/Java)
- Willingness of learning something new



About Instructor

- 13+ years of experience
- Associate Technical Director at Sunbeam
- Freelance Developer working in various domains using different technologies
- Developed 180+ mobile applications on iOS and Android platforms → native , react native
- Developed various websites using PHP, MEAN and MERN stacks
- Languages I love and use in every programming: C, C++, Python, JavaScript, TypeScript, PHP , Go





Software Engineering





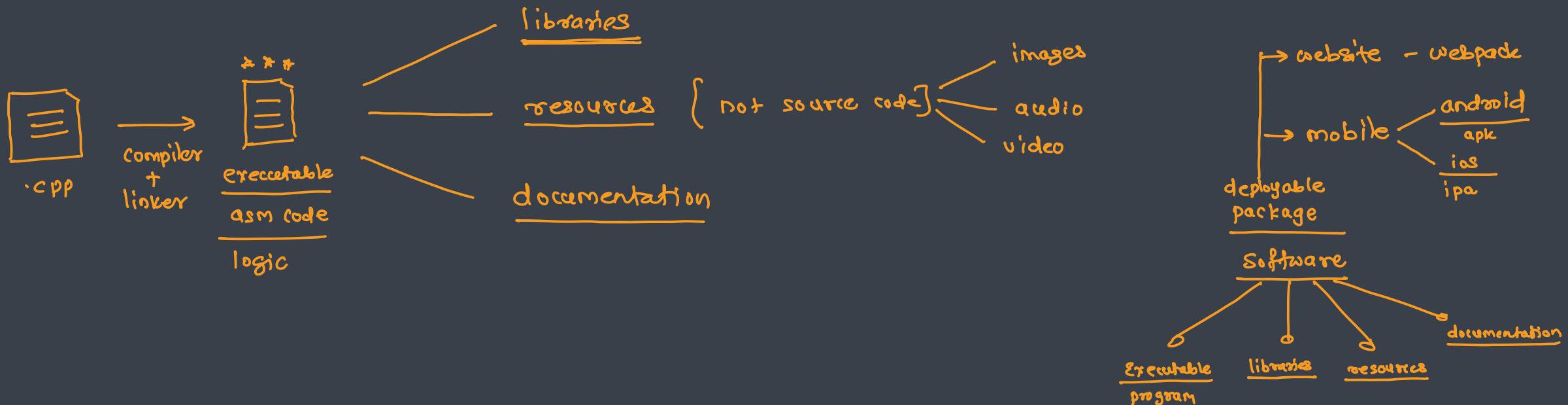
Introduction

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 : operating system, device drivers
- Application Software : office, winzip, audio players
- Engineering/Scientific Software : catia, Auto CAD ...
- Embedded Software : raspberry, IoT
- AI Software : tensorflow, pytorch etc
- Legacy Software : POS based
- Web/Mobile Software

→ websites, web services
 GUI REST - GET/POST/PUT/DELETE
 GraphQL - ?.

→ mobile applications



Why SE is important

- Helps to build complex systems in timely manner – within the said duration
deadline
- Ensures high quality of software
- Imposes discipline to work
- Minimizes software cost → within budget
- Decreases time



Software Evolution

→ Categories

① Calculate simple interest → formula

- **S-type (static-type)** → not changing frequently , simplest , small

- 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

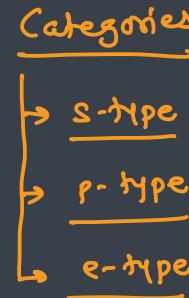
- **P-type (practical-type)** → Games

- Software with a collection of different procedures (ways)
- 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)** * * * SE

- 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

→ requirements are changing frequently



... : ...
£

e-comm → laws / taxes



E-Type software evolution laws

[challenges]

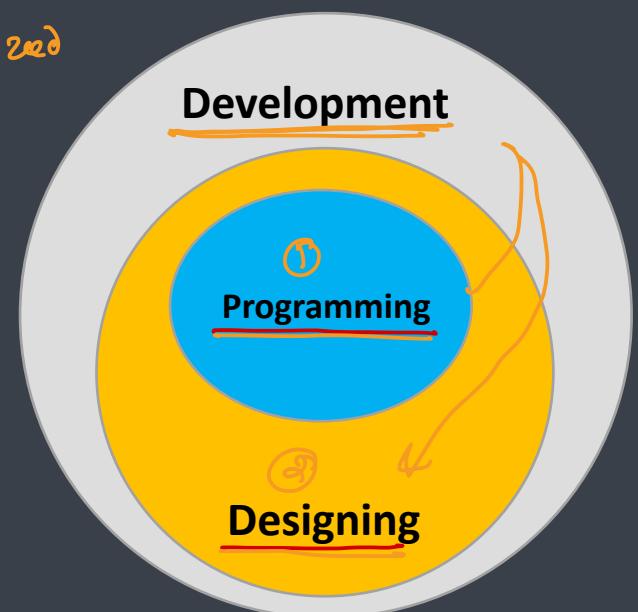
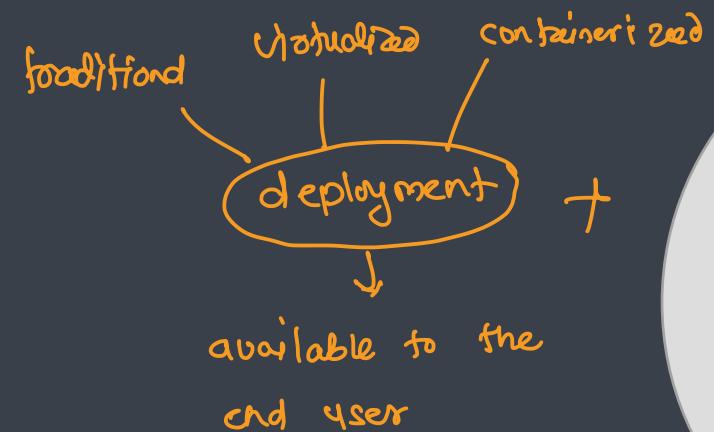
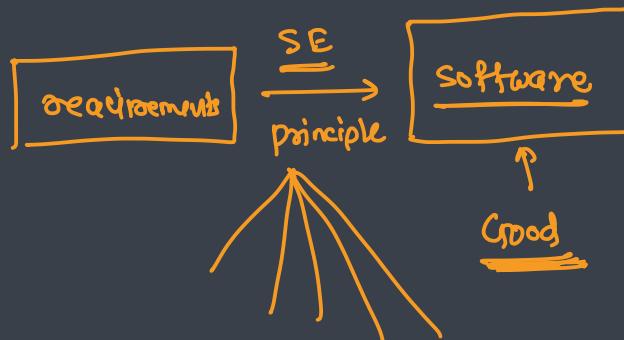
- Continuing change → evolve → features
 - An E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful
- Increasing complexity → more features
 - As software evolves, its complexity tends to increase unless work is done to maintain or reduce it
- 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 → more complexity
 - An E-type software system declines in quality unless rigorously maintained and adapted to a changing operational environment
- Feedback systems → fix feature / code
 - 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

development





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

→ functionality

- This tells us how well software works in operations

- Can be measured on:

- Budget: Software should be developed within the budget

→ Usability: Software should be usable by the end user

- Efficiency: Software should efficiently use the storage and space

- Correctness: Software should provide all the functionalities correctly without having any bug/issue

- Functionality: Software should meet all the requirements of the user

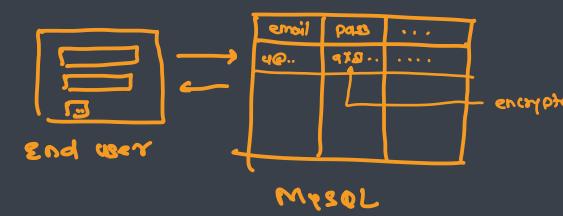
- Dependability: Software should contain all the dependencies in its package

- Security: Software should keep the data safe from any external threat

- Safety: Software should not be hazardous or harmful to the environment

Security

- ① Confidentiality → Encryption

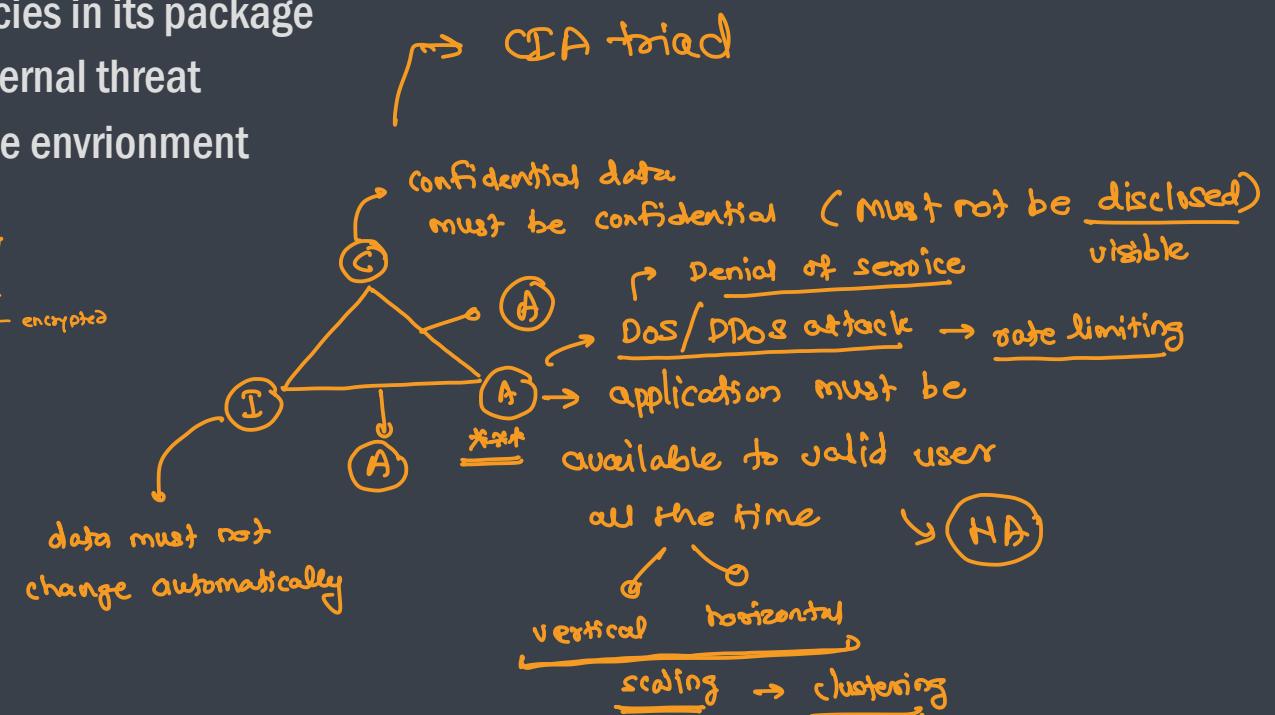


- ② Integrity → Crypto-JS SHA512

- ③ Availability → Cloud, docker, Kubernetes

- ④ Authentication → email/pass → if user is valid

- ⑤ Authorization → if valid user has got enough permission





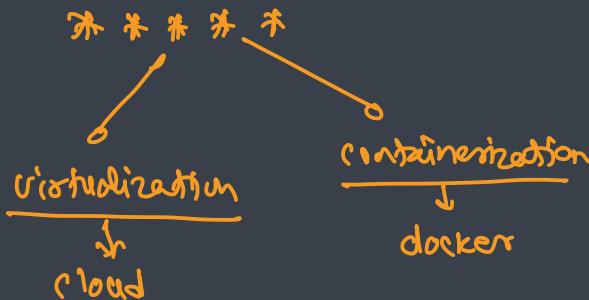
Transitional

- This aspect is important when the software is moved from one platform to another
- Can be measured on
 - Portability: If the software can perform the same operations on different environments and platforms, that shows its Portability
 - Interoperability: It is the ability of the software to use the information transparently
 - Reusability: If on doing slight modifications to the code of the software, we can use it for a different purpose, then it is reusable
 - Adaptability: It is an ability to adapt the new changes in the environment



Maintenance

- Briefs about how well a software has the capabilities to maintain itself in the ever-changing environment
- Can be measured on
 - Maintainability: The software should be easy to maintain by any user
 - Flexibility: The software should be flexible to any changes made to it
 - Extensibility: There should not be any problem with the software on increasing the number of functions performed by it
 - Testability: It should be easy to test the software
 - Modularity: A software is of high modularity if it can be divided into separate independent parts and can be modified and tested separately
 - Scalability: The software should be easy to upgrade

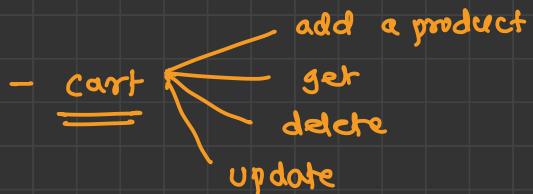
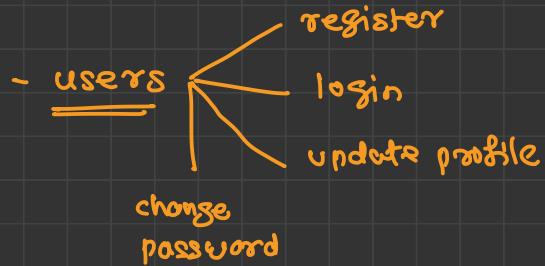
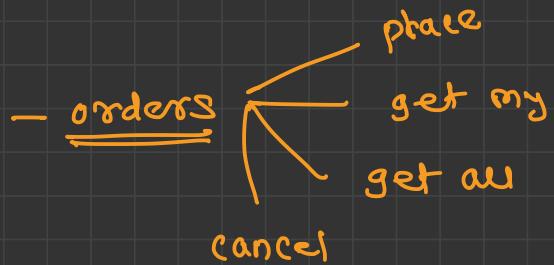
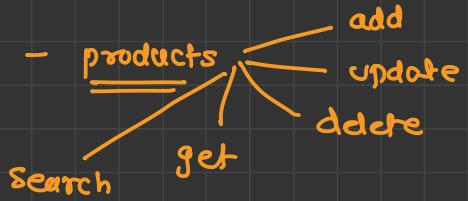




Software Development

Life Cycle

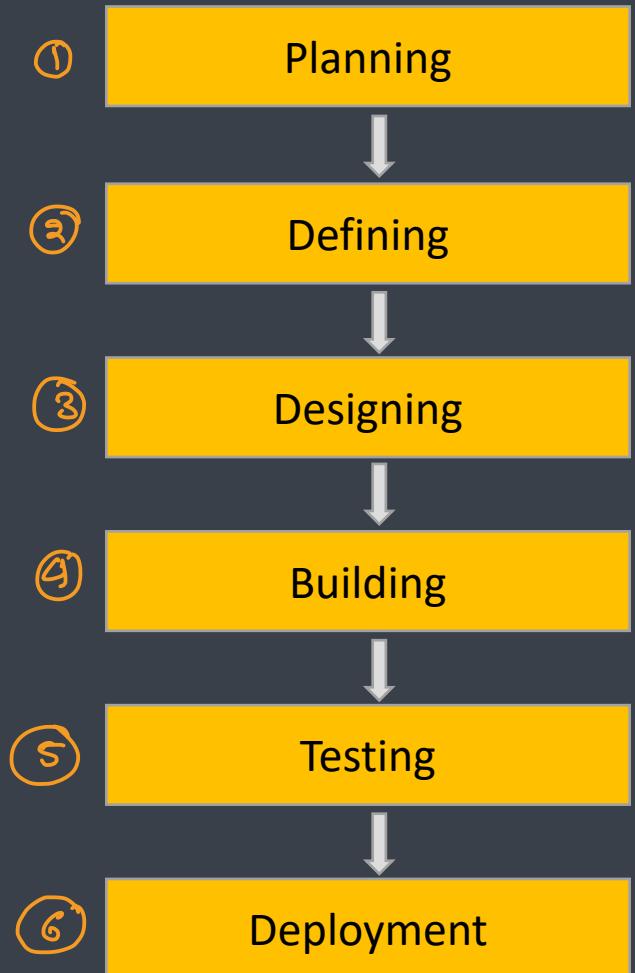
e-commerce





Overview

- Also called as Software Development Process (SDP)
- Is a well-defined, structured sequence of stages in software engineering to develop the intended software product within the duration & budget
- 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

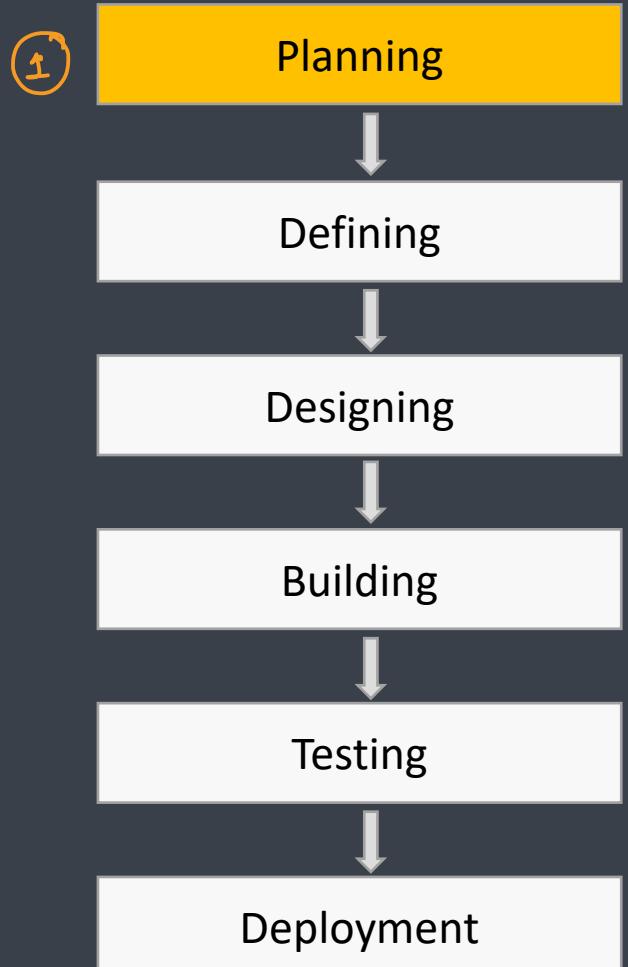


Planning and Requirement Analysis

→ finely grained requirements +
technical approaches

- 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

Requirement gathering + analyzing



stack [environment]

technical stacks

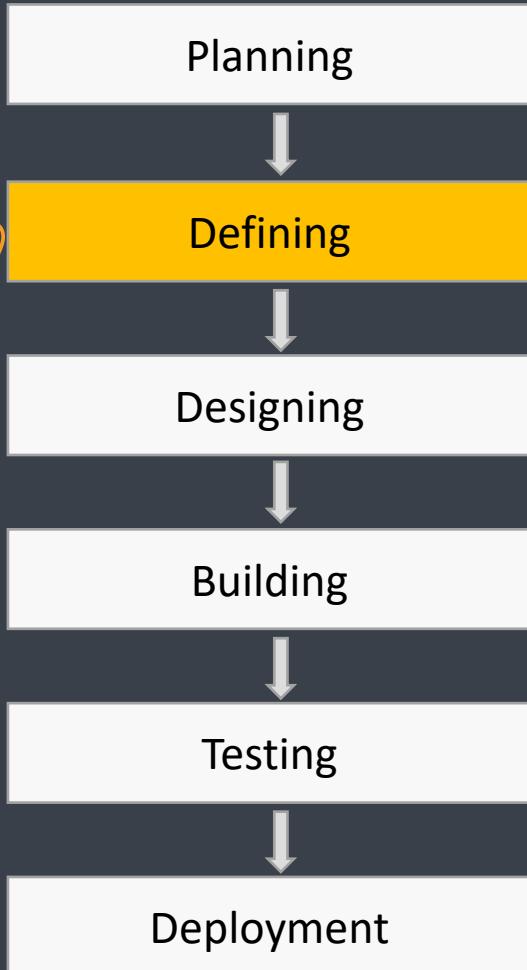
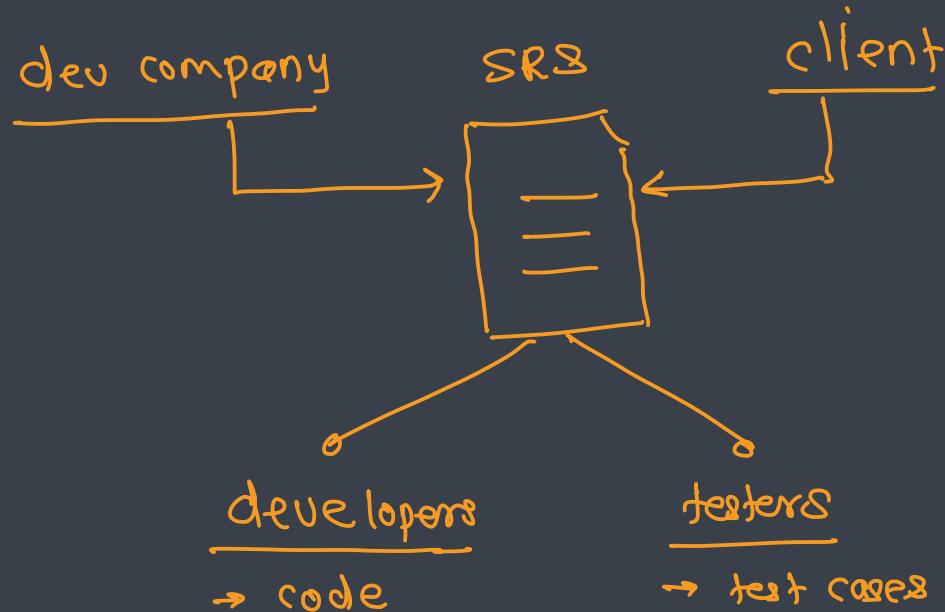
	<u>MERN</u>	<u>MEAN</u>	<u>LAMP</u>	<u>WISA</u>
① <u>Database</u>	Mongo / MySQL	mongo / mysql	MySQL	SQL Server
② <u>Web Server</u>	Express	express	Apache	IIS
③ <u>GUI framework</u>	React	Angular	PHP	Aspx
④ <u>platform</u>	NodeJS	NodeJS	Linux	Windows
language	JavaScript, TypeScript	JavaScript, TypeScript	PHP	C#



Defining Requirements

⇒ SRS document

- 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

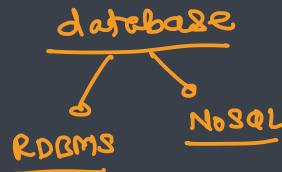
⇒ DDS = one or more applicable design



- 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

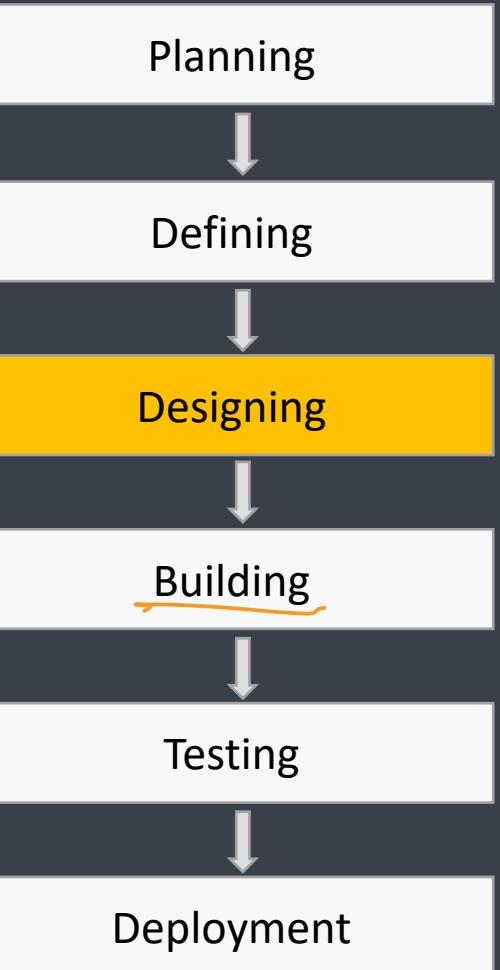
architectures

- client server
- peer to peer
- monolithic
- microservices



design

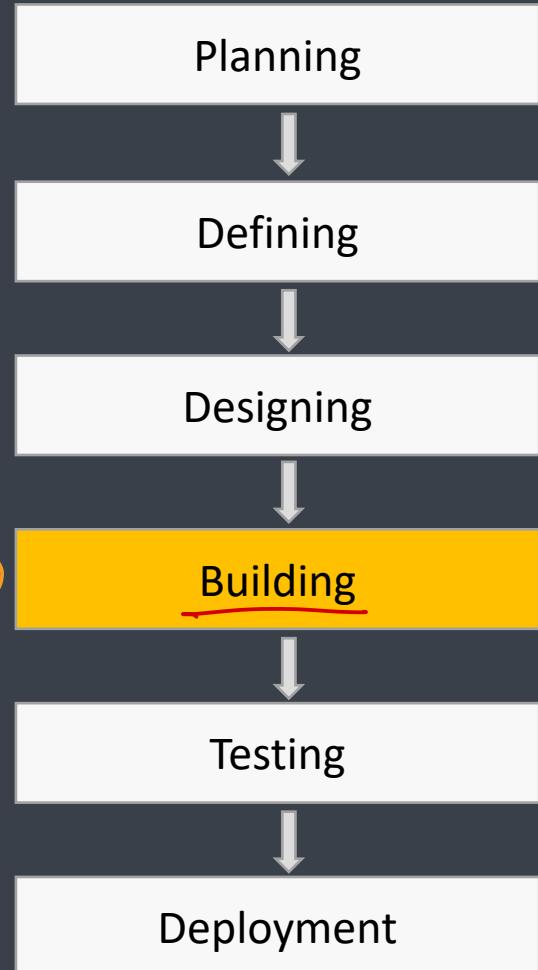
- architecture
- database schema
- screen designs
- workflow - application flow





Building or Developing the Product ⇒ application package [not tested]

- 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

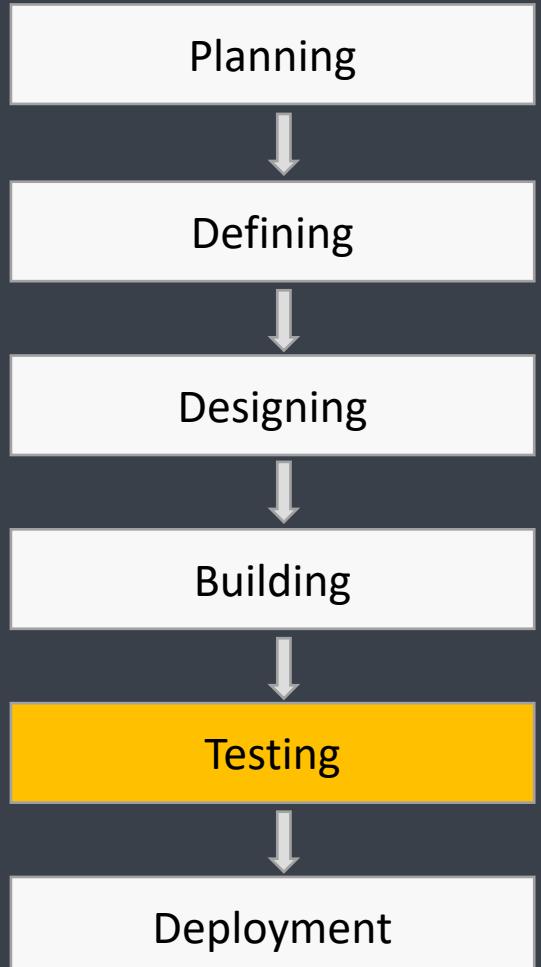




Testing the Product

⇒ Well tested application package (software)

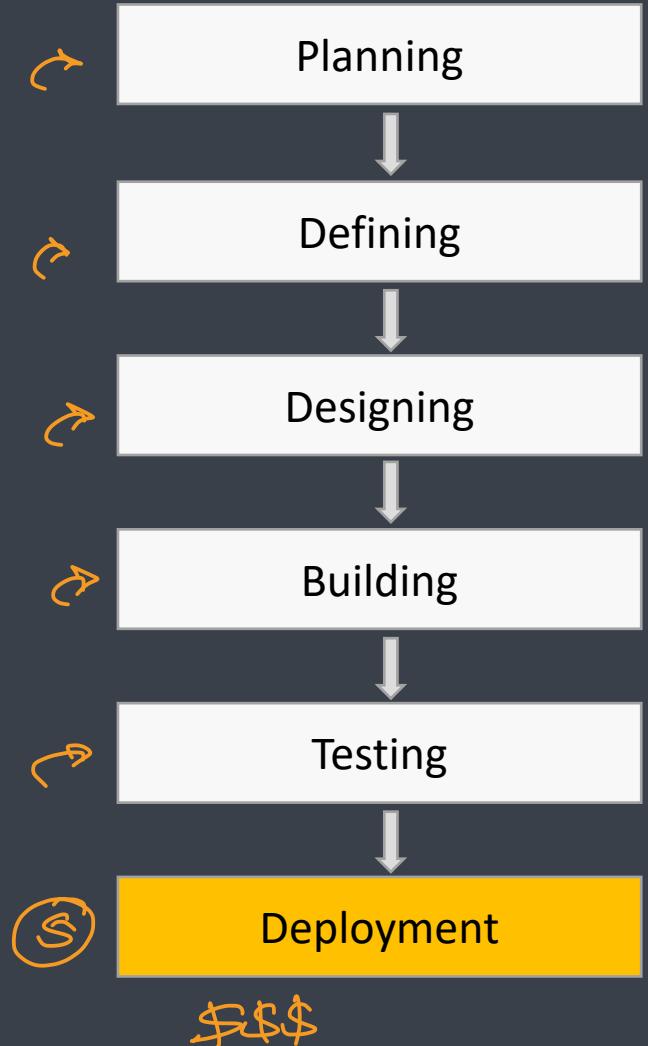
- This stage is usually a subset of all the stages as in the modern SDLC models → STLC
- The testing activities are mostly involved in all the stages of SDLC
- However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the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



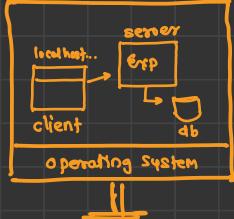


Deployment in the Market and Maintenance

- Once the product is tested and ready to be deployed it is released formally in the appropriate market → playstore , Appstore etc
- 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



developer env.



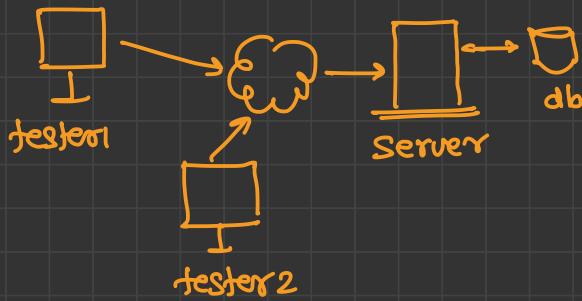
Pre-Production Env



final deploy
for making
app accessible
to the
end users

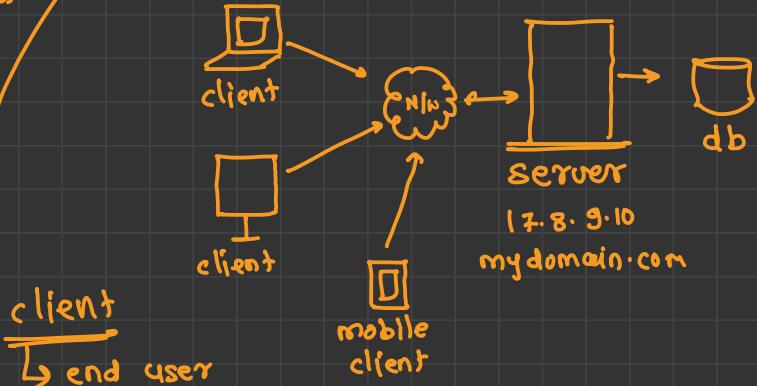
dev changes
deployed to
staging env
for testing

Staging Env



tested app
deployed to the
pre-prod env to
make sure that
it works with
similar conf
as prod

Production Environment



client
→ end user



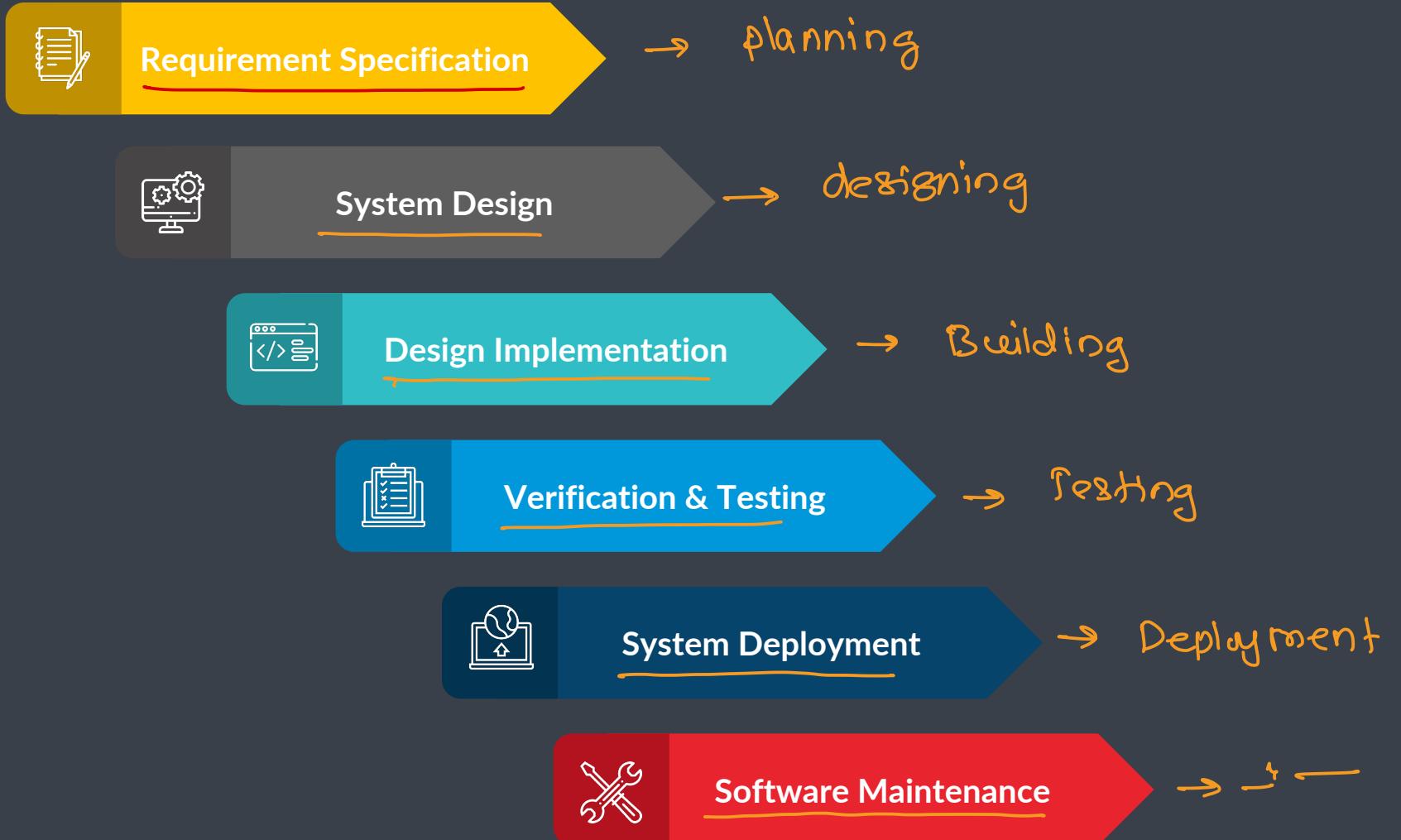
SDLC Models

- 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

RE-COMM → 2 years





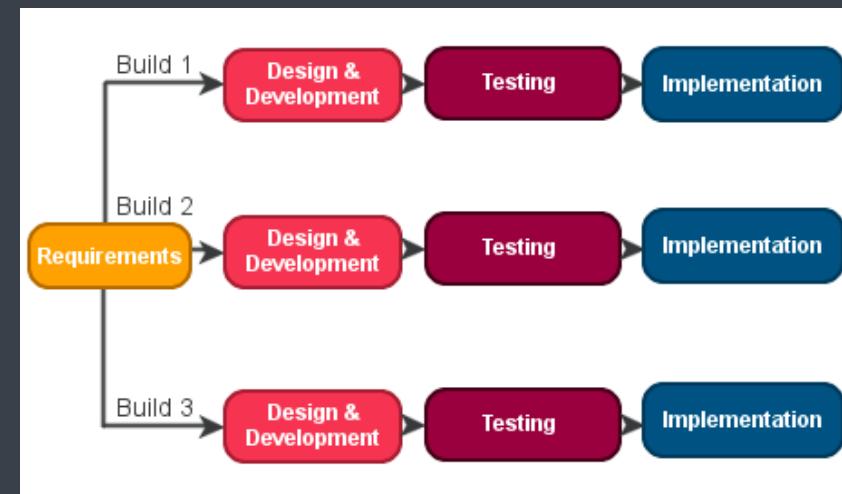
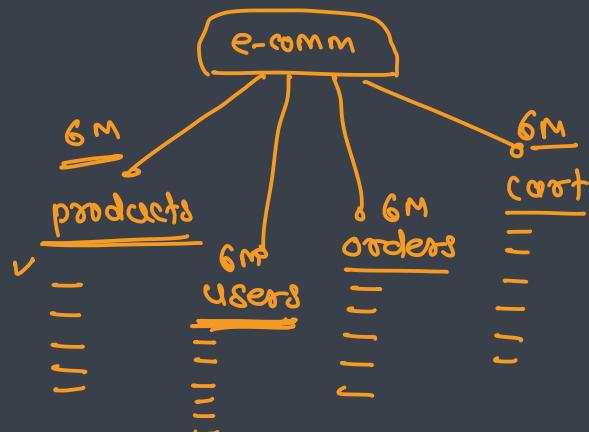
Waterfall Model – Procs and Cons

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model
- Each phase has specific deliverables and a review process
- Phases are processed and completed one at a time
- Works well for smaller projects where requirements are very well understood
- Clearly defined stages
- Well understood milestones
- Easy to arrange tasks
- Process and results are well documented
- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty
- Not a good model for complex and object-oriented projects
- Poor model for long and ongoing projects
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model
- It is difficult to measure progress within stages
- Cannot accommodate changing requirements
- Adjusting scope during the life cycle can end a project
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early



Iterative Model

- In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed
- An iterative life cycle model does not attempt to start with a full specification of requirements
- Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements
- This process is then repeated, producing a new version of the software at the end of each iteration of the model





Iterative Model Pros and Cons

- Some working functionality can be developed quickly and early in the life cycle
- Results are obtained early and periodically
- Parallel development can be planned
- Progress can be measured
- Less costly to change the scope/requirements
- Testing and debugging during smaller iteration is easy
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone
- Easier to manage risk - High risk part is done first
- With every increment, operational product is delivered
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment
- Risk analysis is better
- It supports changing requirements
- Initial Operating time is less
- More resources may be required
- Although cost of change is lesser, but it is not very suitable for changing requirements
- More management attention is required
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle
- Defining increments may require definition of the complete system
- Not suitable for smaller projects
- Management complexity is more
- End of project may not be known which is a risk
- Highly skilled resources are required for risk analysis
- Projects progress is highly dependent upon the risk analysis phase



* * * *

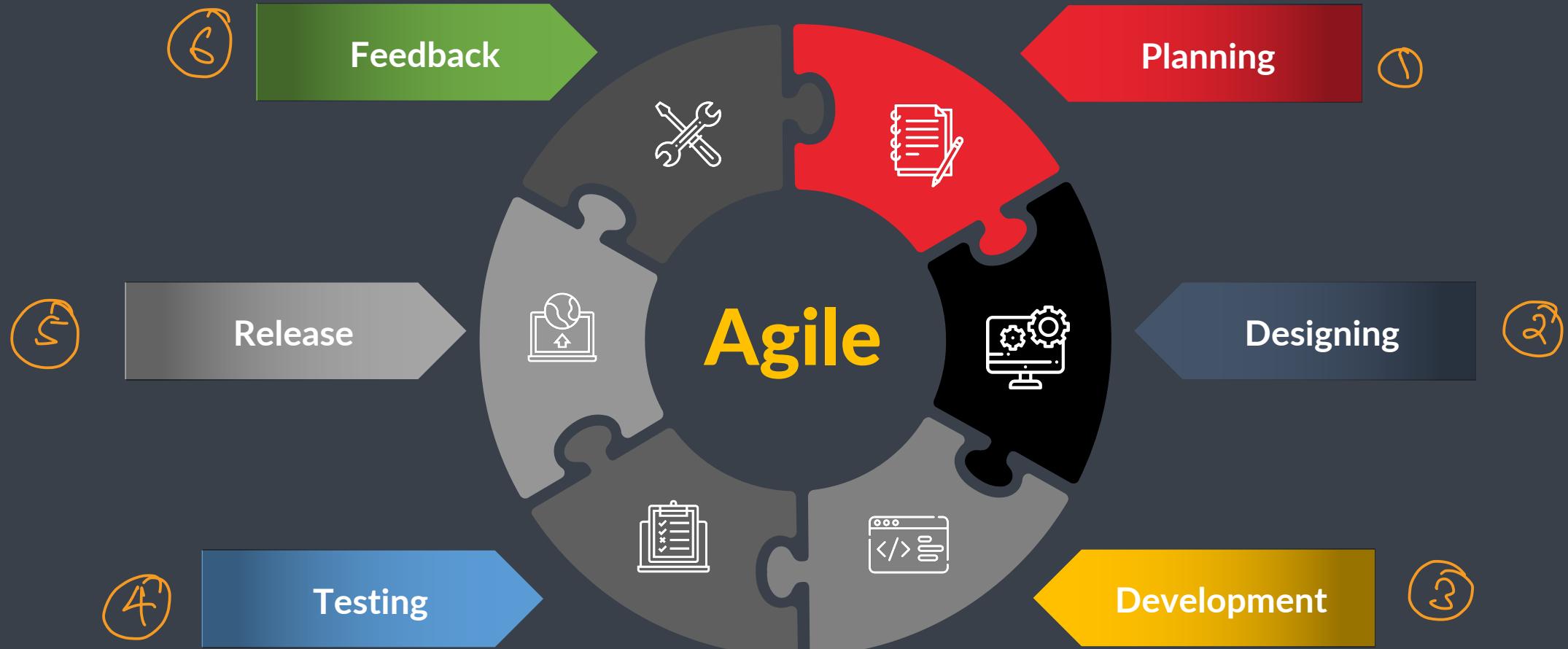
Agile Methodologies



extension to iterative model



Agile Development





Agile Manifesto

Years → Weeks

- In February 2001, at the Snowbird resort in Utah, 17 software developers met to discuss lightweight development methods
- The outcome of their meeting was the following Agile Manifesto for software development
 - We are uncovering better ways of developing software by doing it and helping others do it
 - Through this work, we have come to value
 - Individuals and interactions over Processes and tools
 - Working software over Comprehensive documentation
 - Customer collaboration over Contract negotiation
 - Responding to change over Following a plan
 - That is, while there is value in the items on the right, we value the items on the left more

individuals → client, dev, tester, PM
communication

- ① collaboration ★ ★
② Early feedback ✘ ✘



Principles of Agile Manifesto

■ Customer Satisfaction

- Highest priority is given to satisfy the requirements of customers through early and continuous delivery of valuable software

■ Welcome Change

- Changes are inevitable during software development
- Ever-changing requirements should be welcome, even late in the development phase.
- Agile processes should work to increase customers' competitive advantage

■ Deliver a Working Software → *incremental approach*

- Deliver a working software frequently, ranging from a few weeks to a few months, considering shorter time-scale

■ Collaboration *

- Business people and developers must work together during the entire life of a project

■ Motivation

- Projects should be built around motivated individuals
- Provide an environment to support individual team members and trust them so as to make them feel responsible to get the job done

■ Face-to-face Conversation

- Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team



Principles of Agile Manifesto

■ Measure the Progress as per the Working Software

- Working software is the key and it should be the primary measure of progress

■ Maintain Constant Pace

- Agile processes aim towards sustainable development
- The business, the developers, and the users should be able to maintain a constant pace with the project

■ Monitoring

- Pay regular attention to technical excellence and good design to enhance agility

■ Simplicity

- Keep things simple and use simple terms to measure the work that is not completed

■ Self-organized Teams

- An agile team should be self-organized and should not depend heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams

■ Review the Work Regularly * * *

- Review the work done at regular intervals so that the team can reflect on how to become more effective and adjust its behavior accordingly



Agile Methodologies

- The most popular Agile methods include

- Rational Unified Process (RUP)
 - ① ■ Scrum ★★★
 - Crystal Clear
 - Extreme Programming
 - Adaptive Software Development
 - Feature Driven Development
 - Dynamic Systems Development Method (DSDM)
- ② * 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
- Incremental builds are delivered to the customer in every two to three weeks time *sprint*
- Ideally used in the project where the requirement is rapidly changing
- The framework is made up of a Scrum team, individual roles, events, artefacts, and rules



Scrum Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
 ↳ Stakeholders
- 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
- Working software is the primary measure of progress
- Agile processes promote sustainable development
- Continuous attention to technical excellence and good design enhances agility
- Simplicity, the art of maximizing the amount of work not done, is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly



Scrum Pillars



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

Transparency

show the progress
to the client



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.

Inspection

inspect the requirement

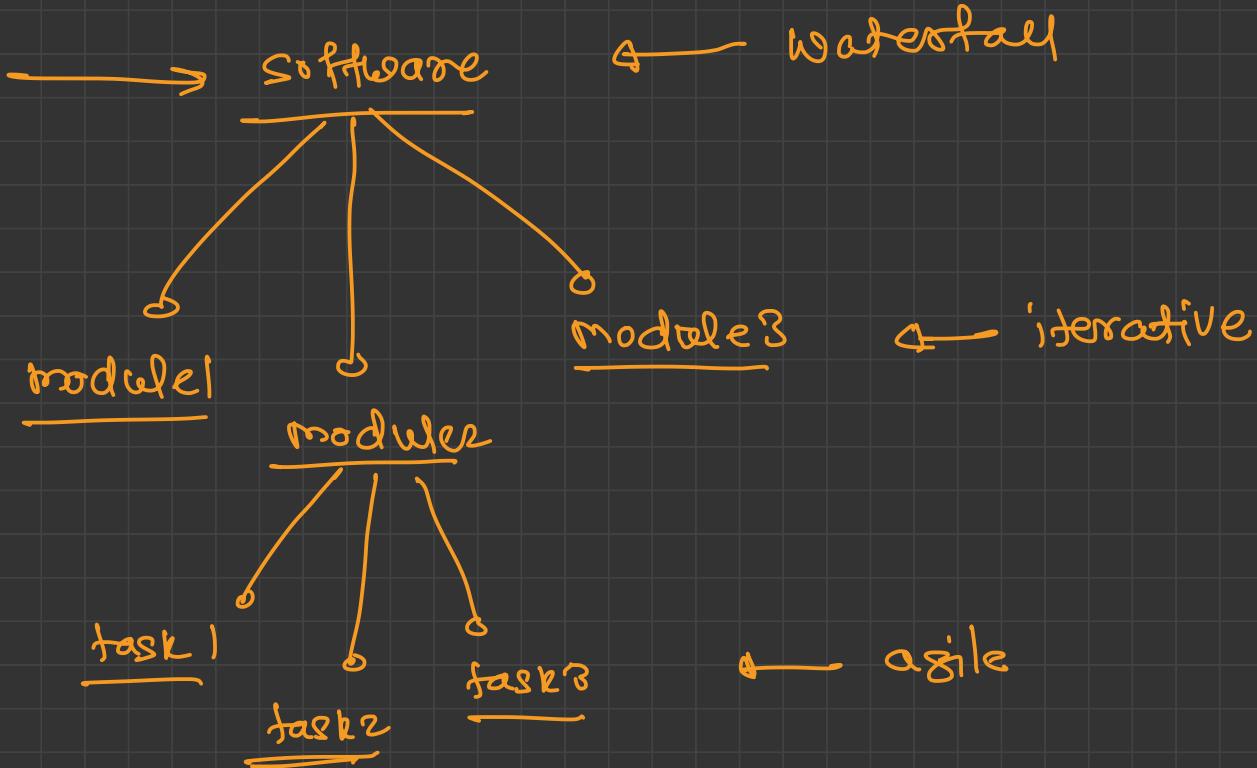


If a member of the Scrum team or a stakeholder notices that things aren't going according to plan, the team will need to change up what they're doing to fix this as quickly as possible

Adaptation

respect the change

S.R&D

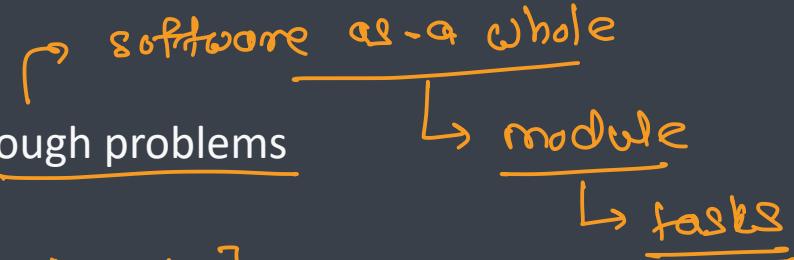




Scrum Values

Courage

Courage to do the right thing and work on tough problems



Focus

Focus on the sprint and its goal

time bound event → min → 1 week
max → 4 weeks

collection of tasks

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

→ events

Collaboration

- Essential principle which focuses collaborative work

→ developers + testers + client
+ team leaders +
project manager

→ time-bound / time-boxed event → Sprint

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

→ with respect to sprint

e-comm

- administrative tasks

- setup the dev environment

- products

collection of all tasks → product backlog

sprint
backlog

- create table to store product ~ 4 hrs
- create a web-service to add product → POST /product → 8 hrs
- create a frontend (GUI) to add product → 24 hrs
- create a ws to update product → PUT /product/: id
- create frontend to update product
- create a ws to delete a product → DELETE /product/: id
- create frontend to delete product

↗ task

product
backlog



Scrum Roles

client side

①

Product Owner

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

client + investors + team

②

Scrum Master

Tool

- 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

③

Dev Team

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



Product Backlog

collection of ALL tasks
to be completed

Sprint Backlog

collection of tasks which
need to finished within
a sprint

Increment

adding features / functionality
sprint by sprint



Scrum Events

↳ meetings

sprint → 1 week

① Sprint Planning

every 2h has → daily status meeting / stand up meeting

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

① add the tasks to sprint backlog
② decide sprint goal
③ generally conducted on thursday / friday before sprint starts

② Daily Scrum

- This is a very short time-boxed event
- Usually only lasting no more than 15 minutes

③ Sprint Review

- Collaborative events to demo what has been achieved and to help keep everyone who's involved working together

④ Sprint Retrospective

- About continuous process and improvement and we need to take what we've learned into the next sprint planning session

⑤ Sprint

- It is really the beating heart of scrum and all the scrum events take place in it



Agile vs traditional models

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



Agile - Advantages

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



Agile - Disadvantages

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



Scrum tools

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



Agile Methodologies – Scrum Terminologies

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



Agile Methodologies – Scrum Terminologies

- **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
- **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



Application Testing

① types

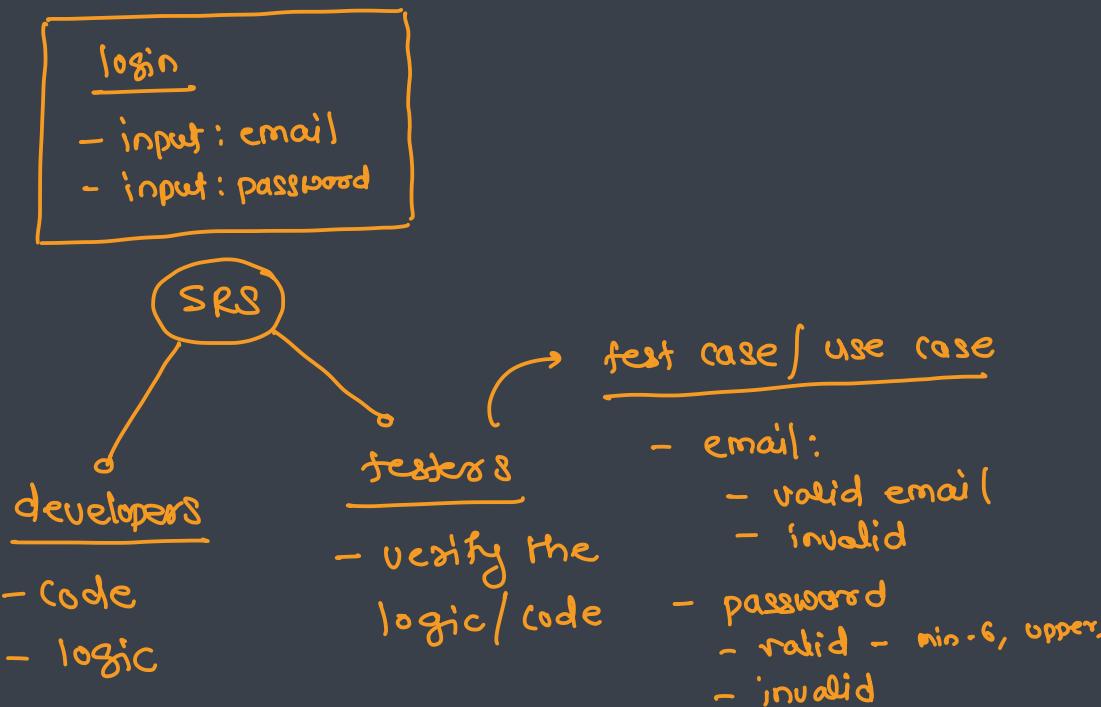
② methods

③ Levels



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 / defect
- executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements
- A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item
- Testing will be done by
 - Software Developer - unit
 - Software Tester - application testing
 - Project Lead/Manager
 - End User - every day 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 ↗ code coverage → unit testing → sonarqube
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified → acceptance level ⇒ 90%.
bug rate ⇒ 5%
=
- Management decision



Verification vs Validation

developer tester

- 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

Login
- Email
- Password

- 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

Login
password → valid ←
Upper + Punctuation ←
Lower + num ←



Quality Assurance vs Quality Control

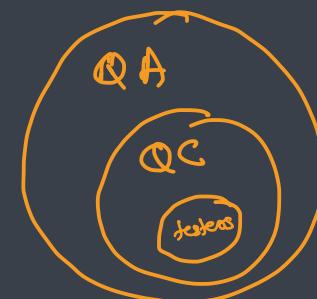
Set the rules / policies

Follow / implement rules / policies

- 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 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

testers





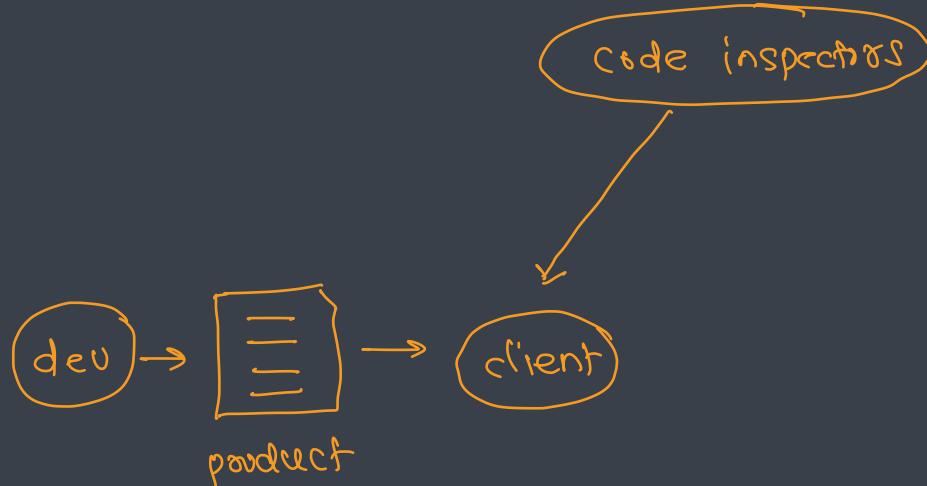
Checking the testing process

Audit → internal

- It is a systematic process to determine how the actual testing process is conducted within an organization or a team
- It is an independent examination of processes involved during the testing of a software
- It is a review of documented processes that organizations implement and follow
- Types: Legal Compliance Audit, Internal Audit, and System Audit

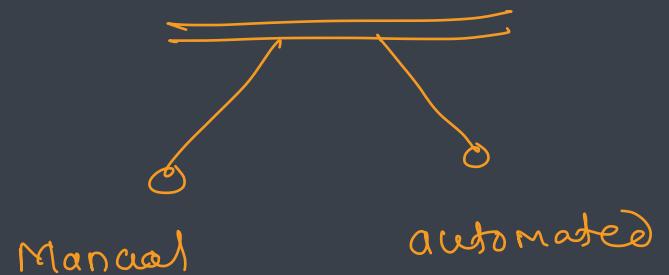
Inspection — external

- Inspection is a formal evaluation technique in which software requirements, designs, or codes are examined in detail
- Conducted by a person or a group other than the author to detect faults, violations of development standards, and other problems





Types





Manual Testing

- cheaper

- 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

} function testing

login
- password is valid → test case1
- email is valid → test case2

✓ sprint 1 → ✓ sprint 2
✓ login → product
...
↓

incremental



Automation Testing

→ Costlier

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

1) web sites → selenium, cypress

2) mobile apps → mobile tester, plcloudy

3) performance → JMeter, stress tester, load tester
available

} non-functional



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 → web site's GUI
 - any area where large number of users can access the software simultaneously
 - all GUI items
 - connections with databases
 - field validations

load testing
JMeter





When to Automate?

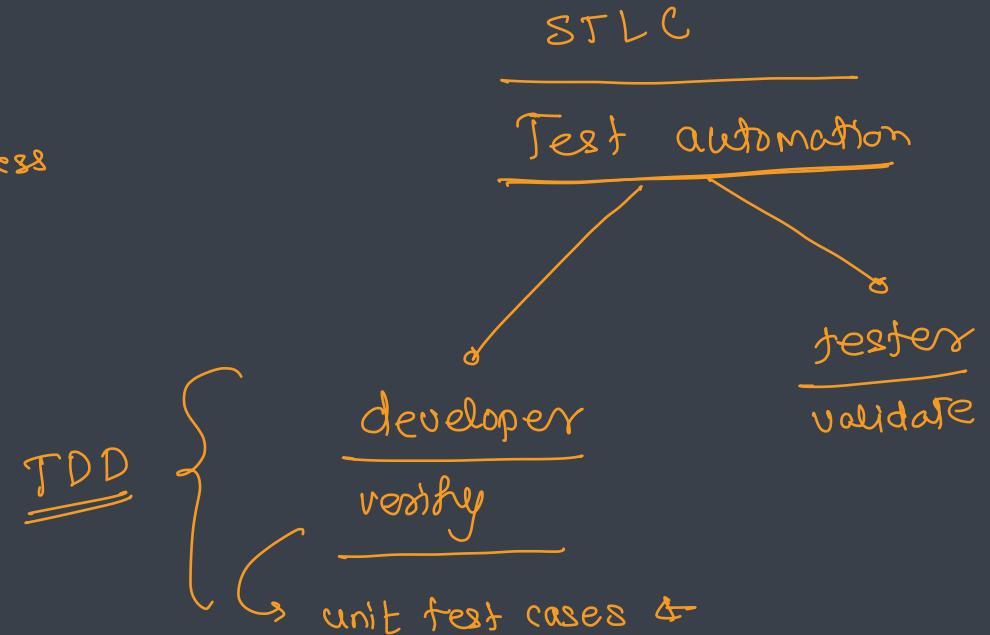
- Large and critical projects
- Projects that require testing the same areas frequently
- Requirements not changing frequently
- Accessing the application for load and performance with many virtual users
- Stable software with respect to manual testing
- Availability of time

→ agile



How to Automate?

- Identifying areas within a software for automation
- Selection of appropriate tool for test automation : selenium, cypress
- Writing test scripts : language → c, c++, python
- Development of test suits : collection of test cases
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues



```
function add (p1,p2){  
    return p1 * p2  
}  
  
const result = add (10,20)
```



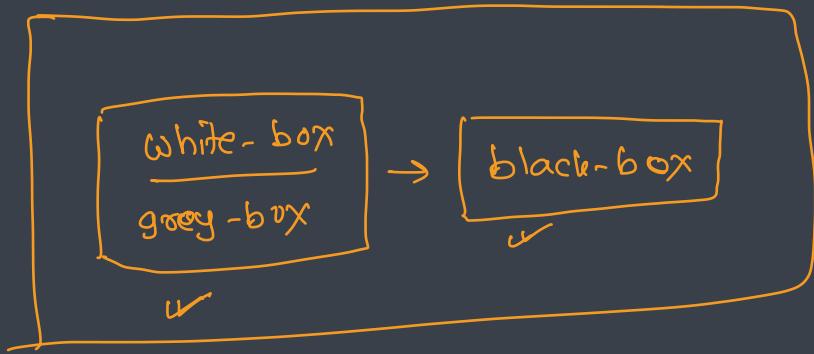
Software Testing Tools

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

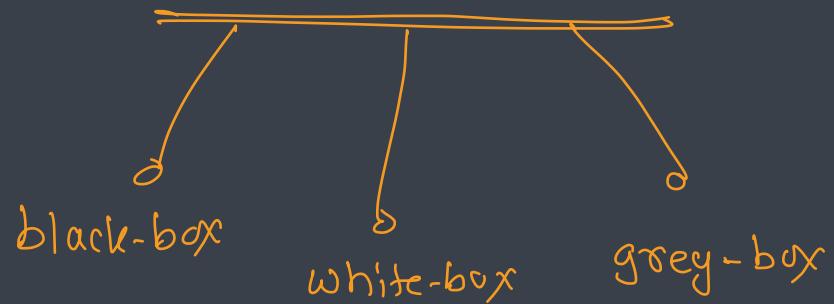
} load tester



festing



Methods





Black-Box Testing

- cheapest

- 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

- Costliest

- 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 ↗ UI ↗ SRS / PPS
 - 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





Functional Testing

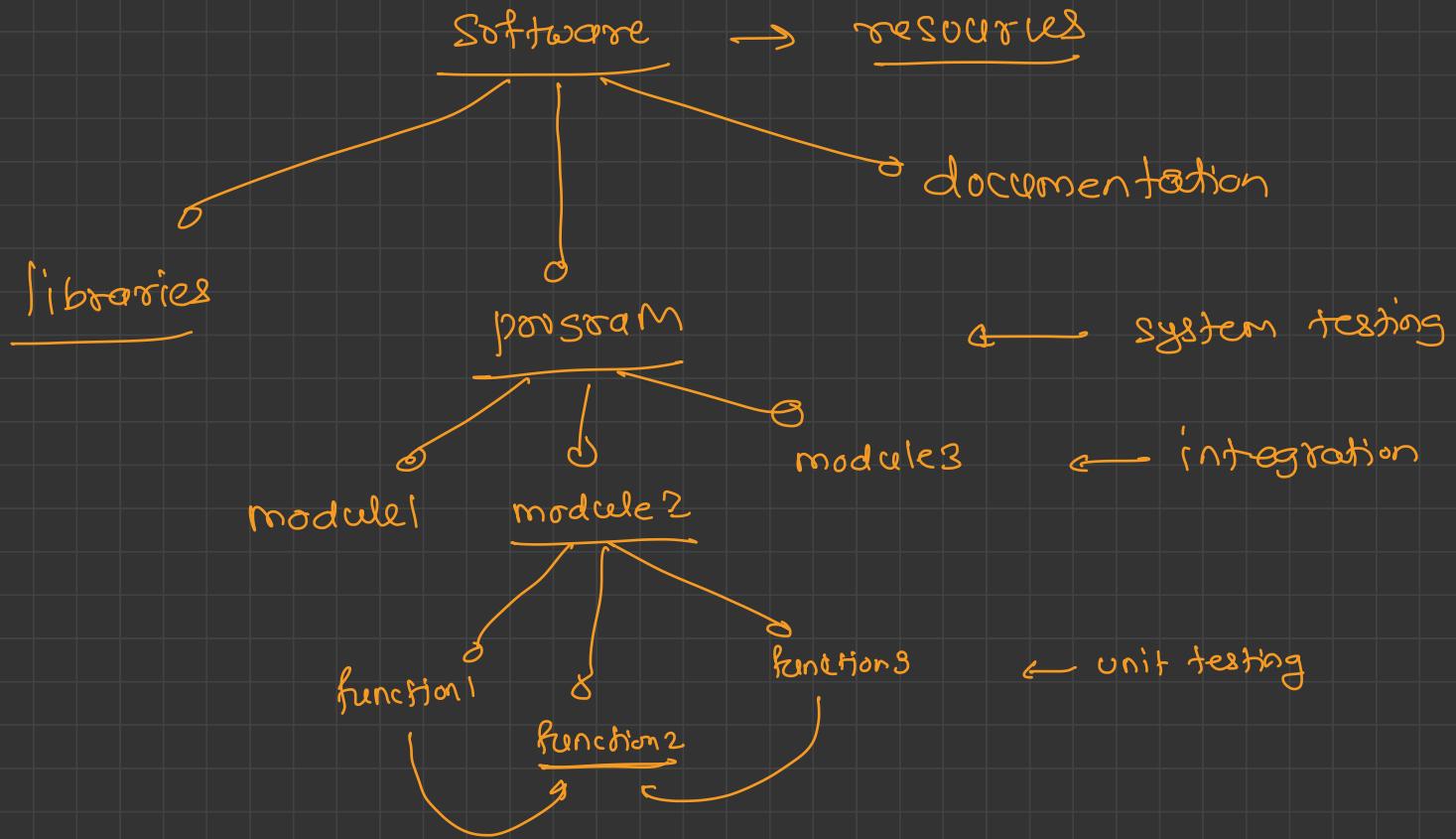
- This is a type of black-box testing that is based on the specifications of the software that is to be tested
- The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for
- Functional testing of a software is conducted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

```
const result = add ( p1, p2 ) :
  ↑  
actual  
[ 10, 20 ] = [ 20 ]  
expected
```



Unit Testing

Unit = function

code coverage → code covered by unit testing

1



- 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
 - functions
- 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

Integration Testing

→ end-to-end

2



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



System Testing

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

Regression Testing

4

- 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

[User Acceptance Testing] → [UAT environment]

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



Alpha Testing

→ internally

- 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

→ production / Live environment

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



Non-Functional Testing

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

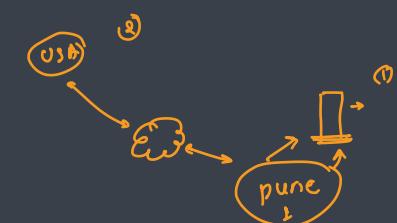
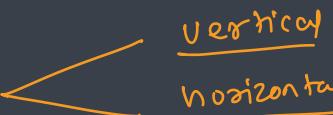
load testing
stress testing





Performance Testing

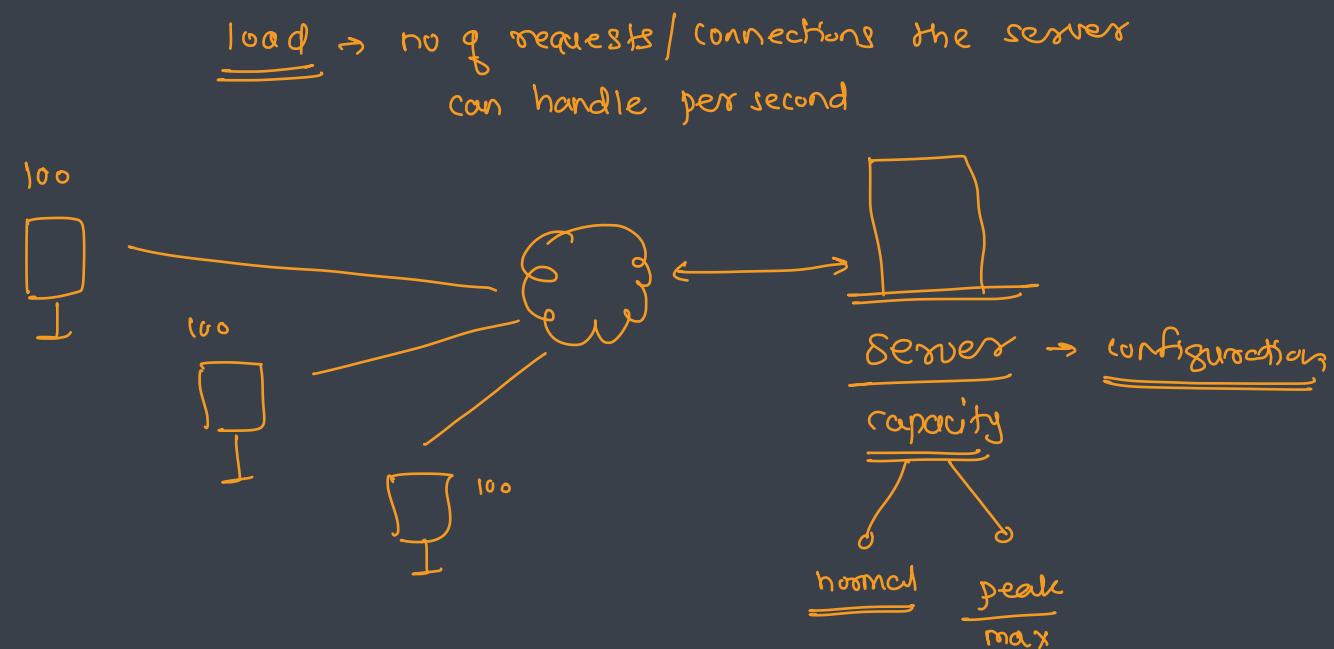
- It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software
- There are different causes that contribute in lowering the performance of a software
 - Network delay → latency → time it takes to send request & get response
 - Client-side processing
 - Database 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





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 . JMeter

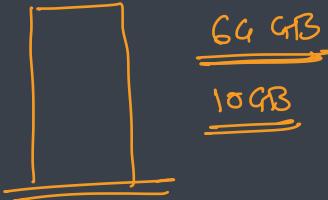




Stress Testing

- abnormal conditions 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.





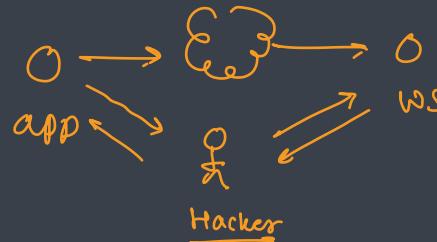
Usability Testing

- 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
 - ① ✓ ■ Confidentiality
 - ② ✓ ■ Integrity
 - ③ ✓ ■ Authentication
 - ④ ✓ ■ Availability
 - ⑤ ✓ ■ Authorization
 - ✓ ■ Non-repudiation → man-in-the-middle attack
 - ✓ ■ Input checking and validation
 - ✓ ■ SQL insertion attacks





Portability Testing

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



Cypress

Selenium

System Testing



Overview

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



Advantages

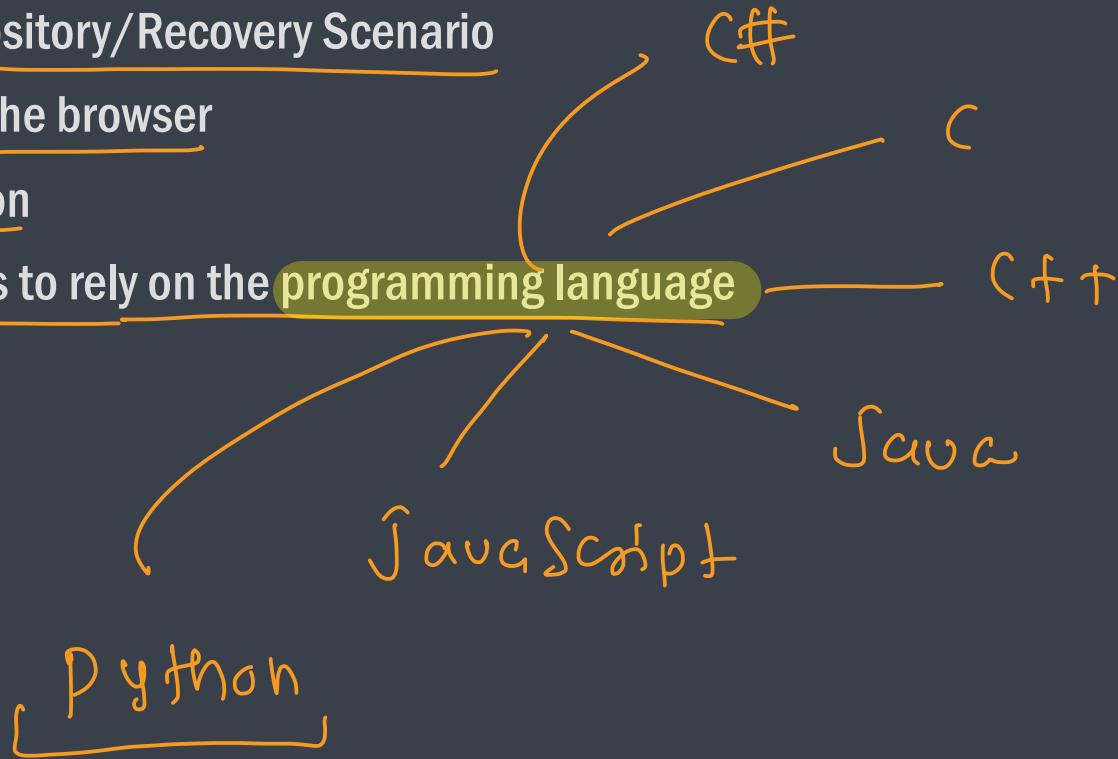
- Selenium is an open-source tool
- Can be extended for various technologies that expose DOM
 - React, c#, C++, Java
- Has capabilities to execute scripts across different browsers
- Can execute scripts on various operating systems
 - macos
 - Linux
 - Windows
- Supports mobile devices
- Executes tests 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



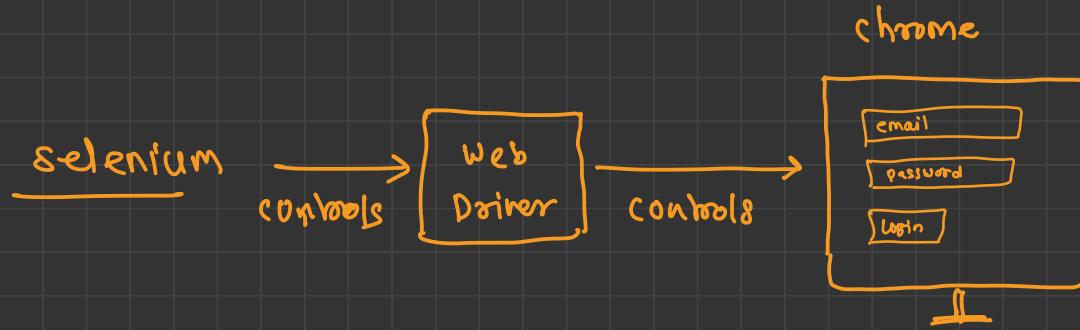


Disadvantages

- 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**



- 1] open browser
- 2] visit the URL
- 3] find email input
- 4] enter valid email
- 5] find password input
- 6] enter valid password
- 7] click the login button
- 8] check the output
- 9] close the browser





WebDriver

→ browser specific

- 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



Locators

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

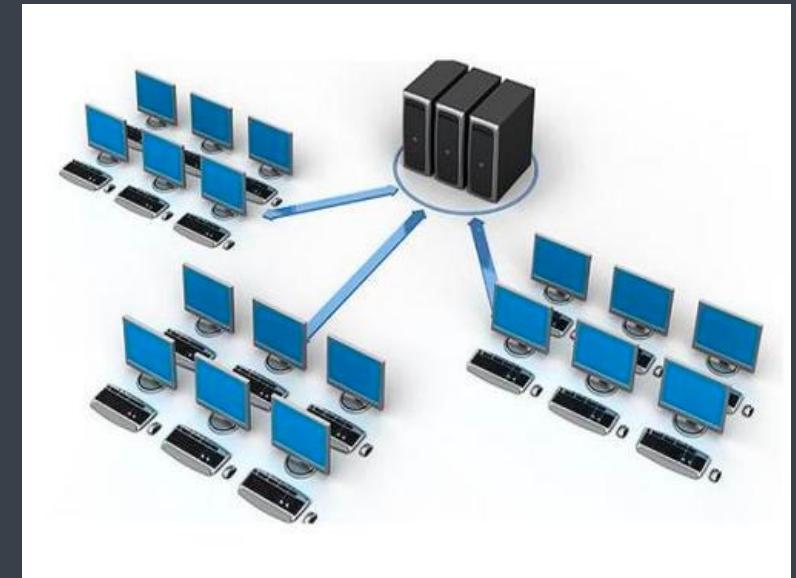


Networking



What is network ?

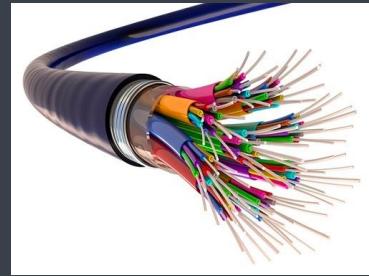
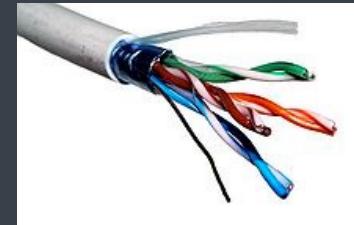
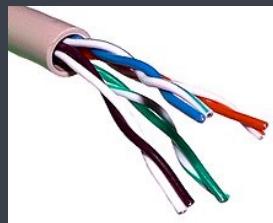
- It is the interconnection of multiple devices, generally termed as Hosts connected using multiple paths
- The purpose of network is
 - sending/receiving data or media
- It involves various devices like hubs, switches, routers etc.





Wired network

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





Wireless network

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





Network Types

■ Personal Area Network

- Smallest network which is very personal to the user
- E.g. BlueTooth

■ Local Area Network

- Spans across building(s) and operated under single administrative system
- E.g. company, school network
- Technologies: TokenRing or Ethernet

■ Metropolitan Area Network

- Spans across cities
- E.g. cable network
- Technologies: high speed fiber optics

■ Wide Area Network

- Spans across countries
- Technologies: ATM, Frame Relay



What is a network topology ?

- Physical arrangement of computers is known as topology
- Famous topologies
 - Bus
 - Ring
 - Token Ring
 - Star
 - Mesh



ISO OSI model

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



Application Layer

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



Presentation Layer

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



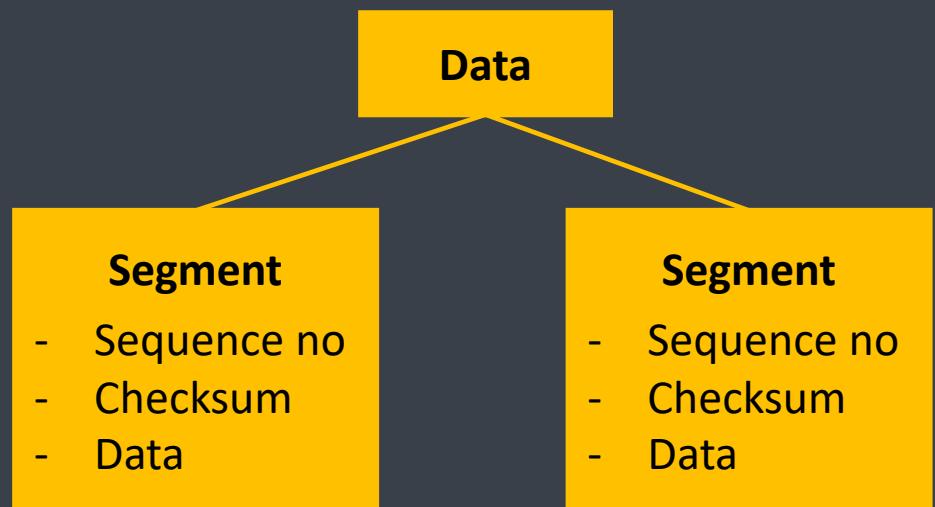
Session Layer

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



Transport Layer

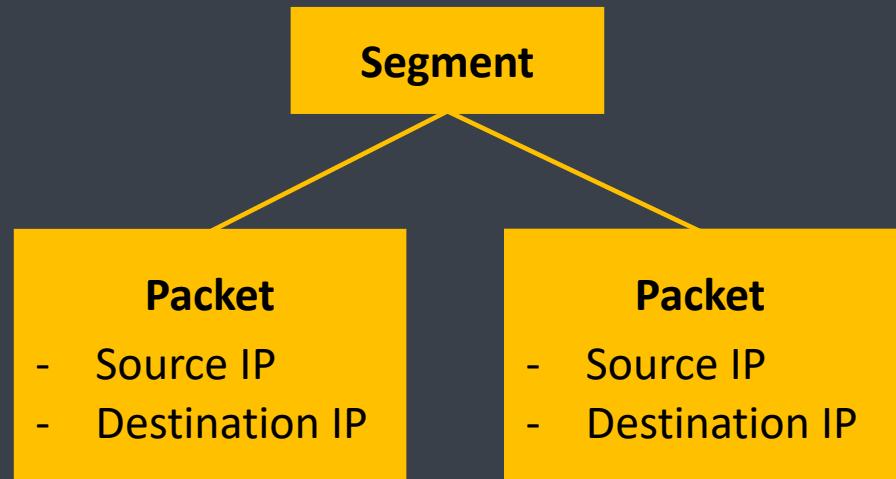
- Provide host-to-host communication services for applications
- Creates Segment (data unit) containing
 - Sequence number
 - Checksum
 - Port number
- Protocols
 - **TCP**
 - Connection oriented protocol
 - Provides: Flow Control, Error checking
 - Guarantees data delivery
 - Slower than UDP
 - E.g.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





Network Layer

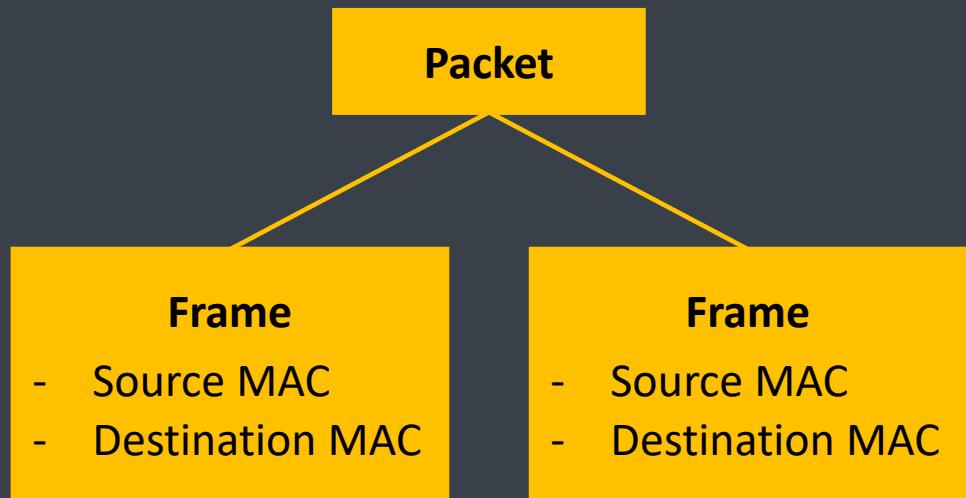
- Responsible for packet forwarding including routing through intermediate routers
- Responsible for splitting segment into packets containing
 - Source IP address
 - Destination IP address
- Protocols
 - **IP:** Internet Protocol
 - **IPX:** Internetwork Packet Exchange
 - **IPSec:** Internet Protocol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





Data Link Layer

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





Data Link Layer: Logical Link Layer

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



Data Link Layer: Media Access Control Layer

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



Physical Layer

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



Addressing Modes: MAC Address

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



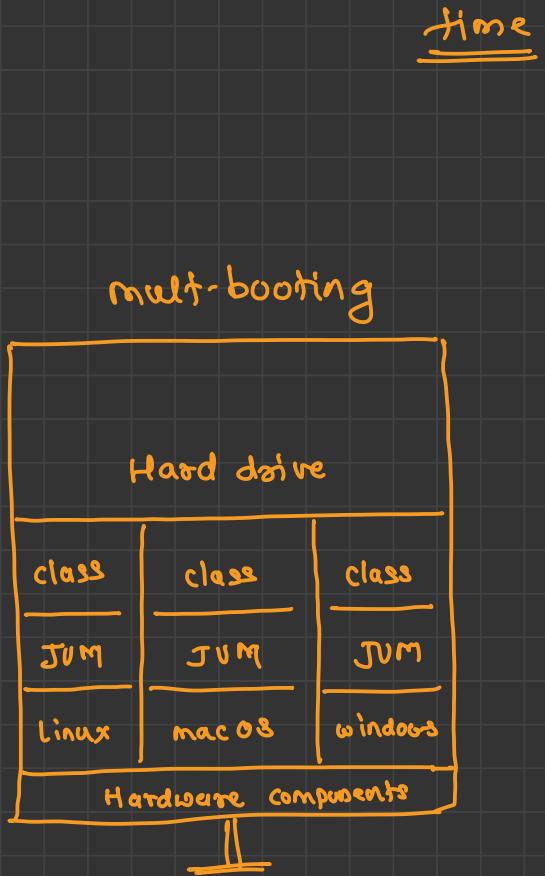
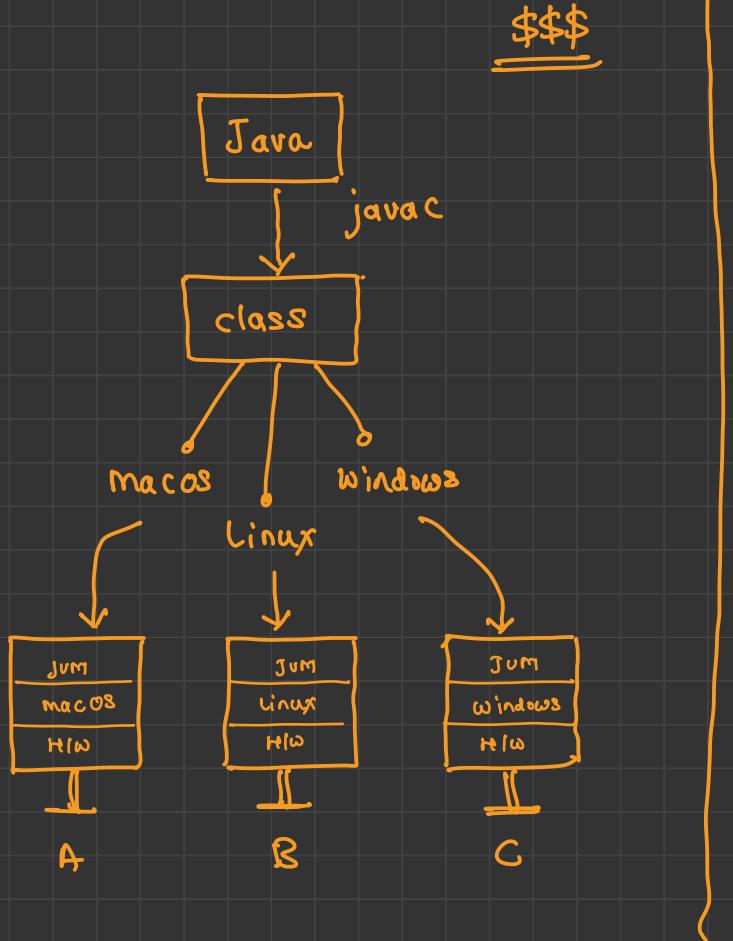
Addressing Modes: IP Address

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



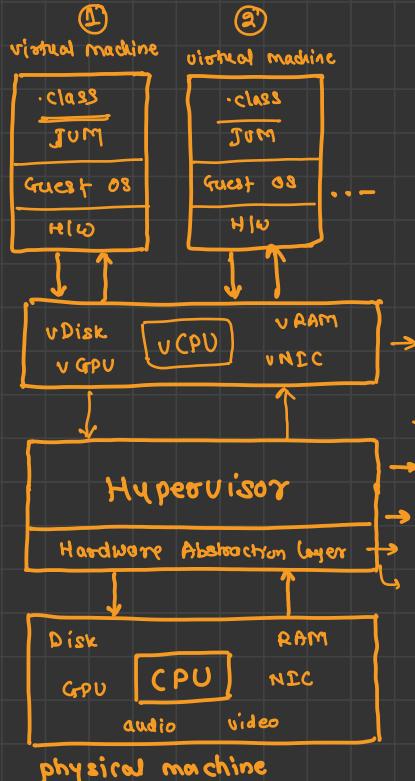
Virtualization





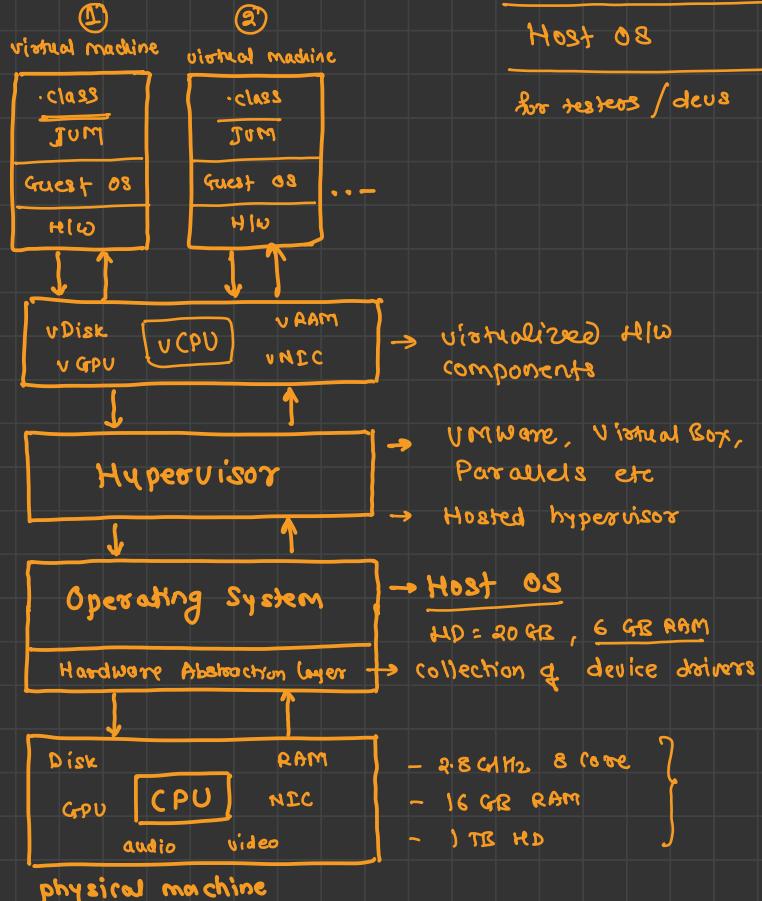
Type I

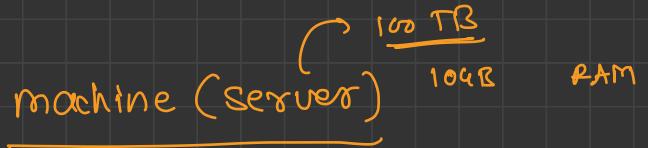
Bare-metal hypervisor



Type II

Hosted Hypervisor





- server rack

- sow

- soom

- floor

- building

→ a.z

- region

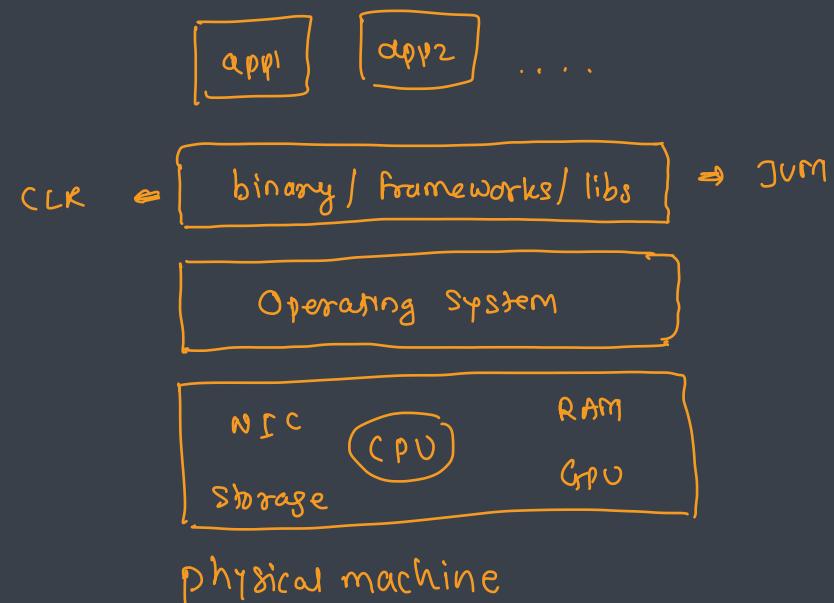
- providers

Closed



Traditional Deployment

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





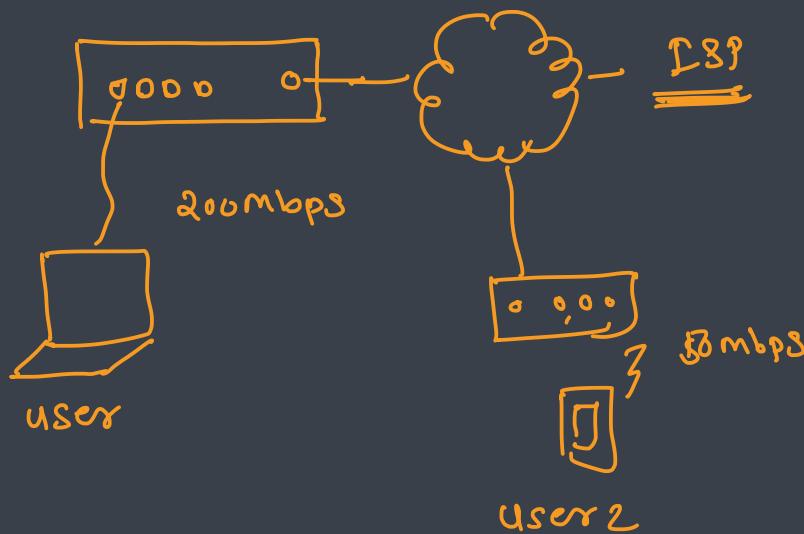
What is virtualization

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



Network Virtualization

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





Storage Virtualization

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



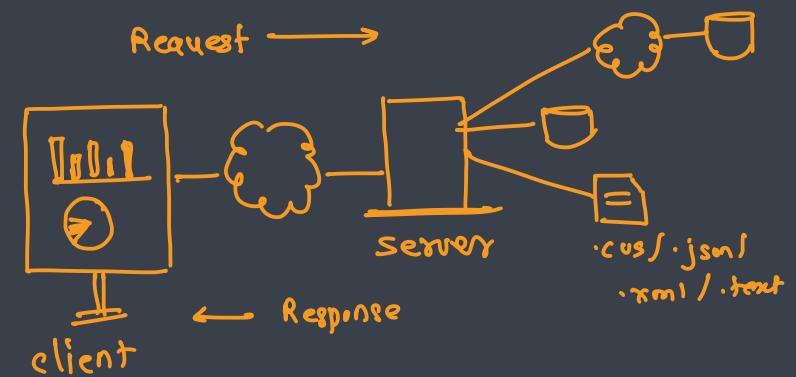
Redundant Array of
Inexpensive Disks



Data virtualization

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

REST → GraphQL



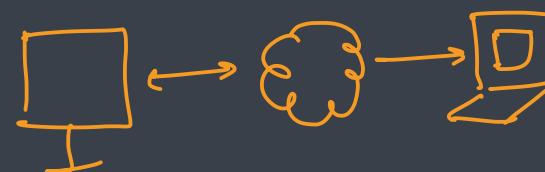


Desktop virtualization

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

VNC

Remote Desktop Protocol





Application Virtualization

: websites

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



Hardware Virtualization

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



Virtual Machine

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

umdk 5



Types of hardware virtualization

Type I — No Host OS → hypervisor → customized OS

- A Type 1 hypervisor runs directly on the host machine's physical hardware, and it's referred to as a **bare-metal hypervisor**
- It doesn't have to load an underlying OS first
- With direct access to the underlying hardware and no other software, it is **more efficient** and provides **better performance**
- It is best suited for enterprise computing or data centers → **Cloud providers**
- E.g. **VMware ESXi, Microsoft Hyper-V server and open source KVM**

Type II → Uses Host OS

→ host OS

- A Type 2 hypervisor is typically installed on top of an **existing OS**, and it's called a **hosted hypervisor**
- It relies on the host machine's pre-existing OS to manage calls to CPU, memory, storage and network resources
- E.g. **VMware Fusion, Oracle VM VirtualBox, Oracle VM Server for x86, Oracle Solaris Zones, Parallels and VMware Workstation**

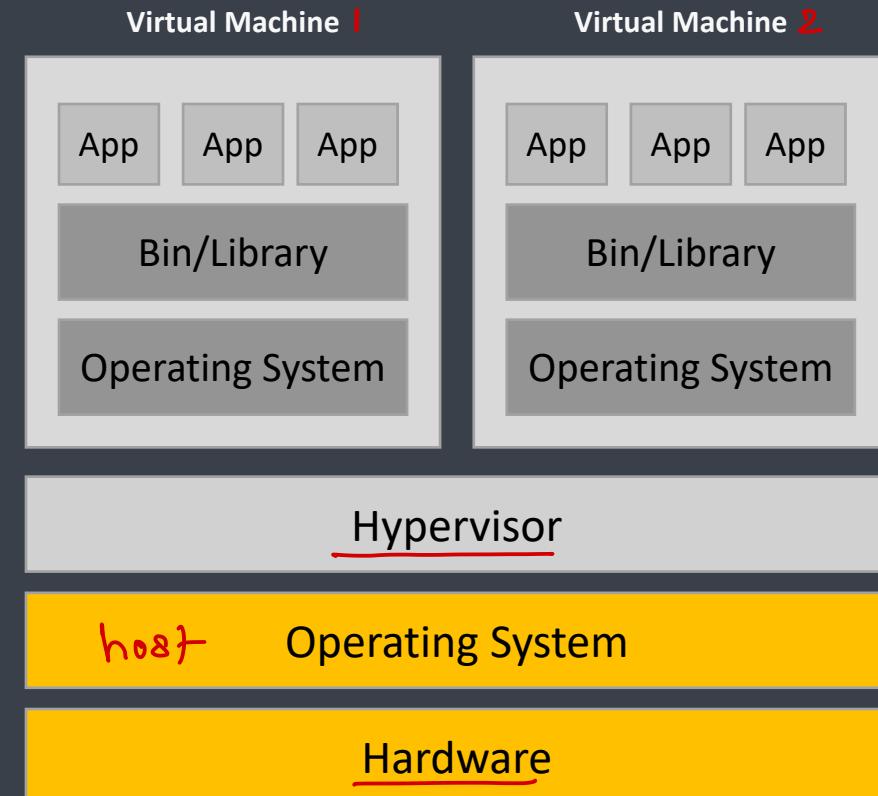
→ **developers / tester**

→ **HAL**



Virtualized Deployment

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





Advantages of virtualization

- **Lower costs**

- Virtualization reduces the amount of hardware servers necessary within a company and data center
- This lowers the overall cost of buying and maintaining large amounts of hardware

- **Easier disaster recovery**

- Disaster recovery is very simple in a virtualized environment
- Regular snapshots provide up-to-date data, allowing virtual machines to be feasibly backed up and recovered
- Even in an emergency, a virtual machine can be migrated to a new location within minutes

- **Easier testing**

- Testing is less complicated in a virtual environment
- Even if a large mistake is made, the test does not need to stop and go back to the beginning
- It can simply return to the previous snapshot and proceed with the test

- **Quicker backups**

- Backups can be taken of both the virtual server and the virtual machine
- Automatic snapshots are taken throughout the day to guarantee that all data is up-to-date
- Furthermore, the virtual machines can be easily migrated between each other and efficiently redeployed

- **Improved productivity**

- Fewer physical resources results in less time spent managing and maintaining the servers
- Tasks that can take days or weeks in a physical environment can be done in minutes
- This allows staff members to spend majority of their time on more productive tasks, like raising revenue and fostering business initiatives



Cloud

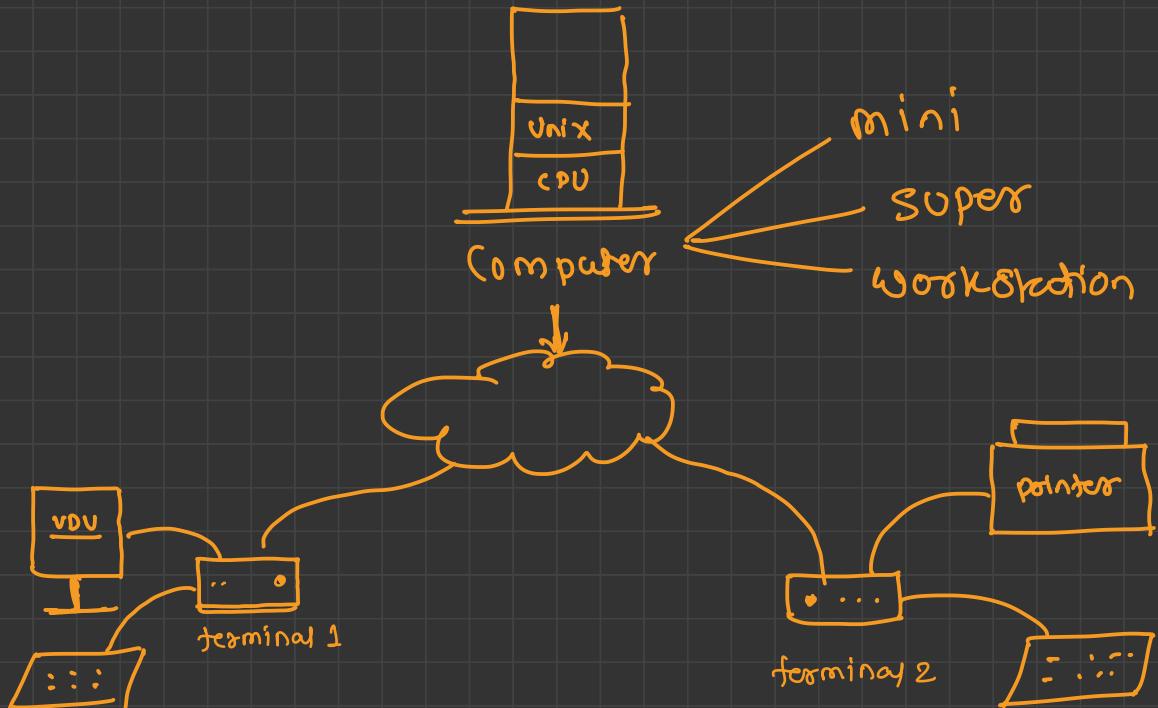


Computing Model

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

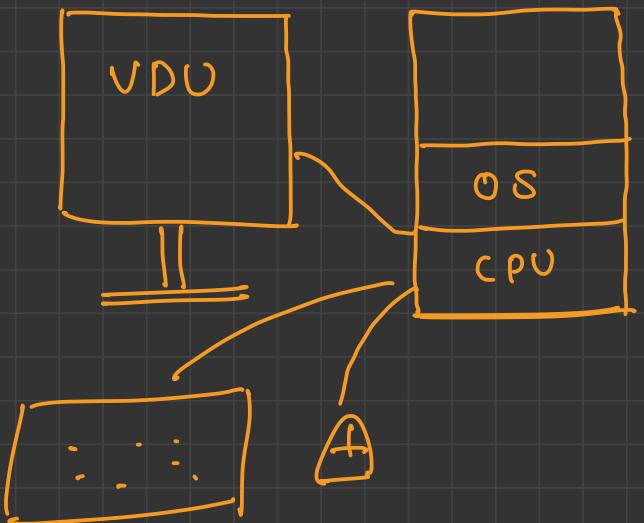
pre-desktop computing

vdu - visual display unit



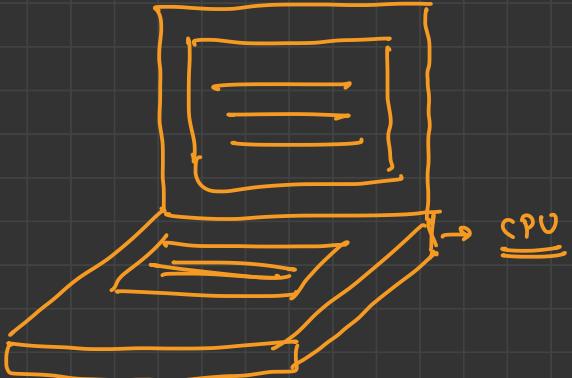
Desktop computer

desktop

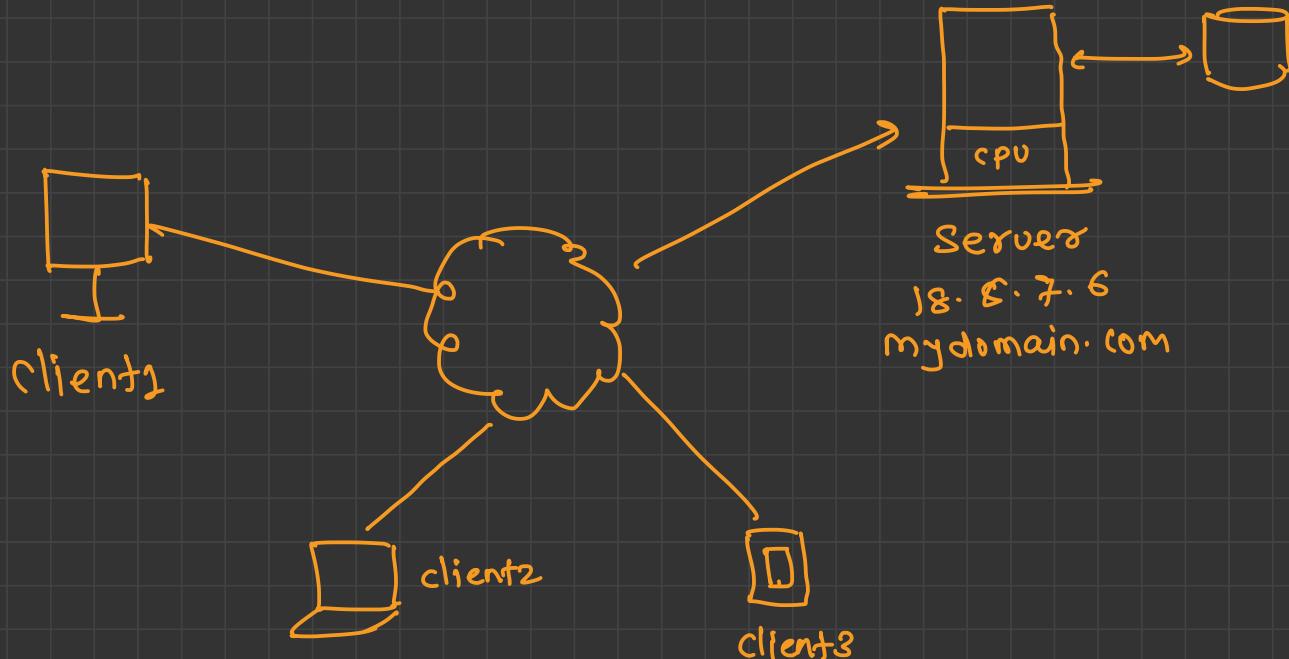


laptop

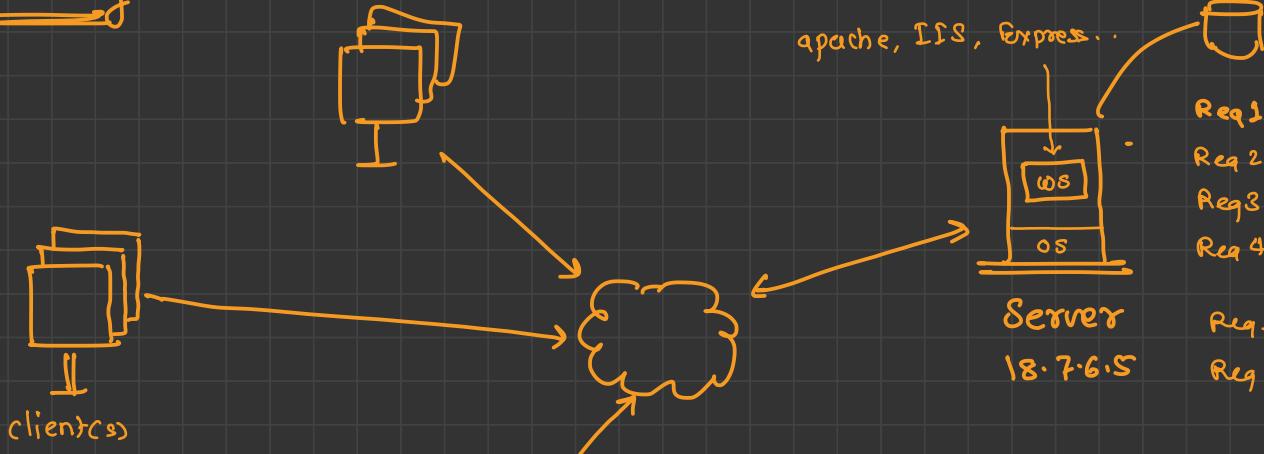
Dynabook



client-server computing



Scaling



cons

- limit on configuration
- Single Point of failure [SPOF]
- static in nature
↳ downtime



clients)

→ $2 \times 2.8\text{GHz}$ Xeon $\frac{1\text{CPU} = 2\text{ threads}}{2 \times \text{CPUs} = 4\text{ threads}}$
 128 GB RAM
 10 TB HD

↓ Vertical Scaling

→ $8 \times 2.8\text{GHz}$ CPUs
 256 GB RAM
 10 TB

- upgrading configuration

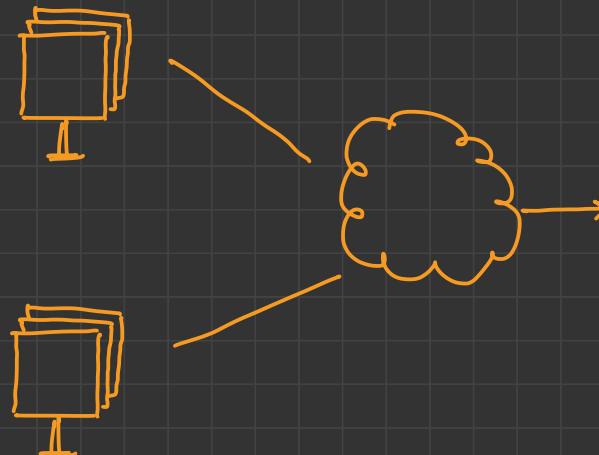
DDoS: Distributed Denial of Service

Horizontal scaling

↳ highly available

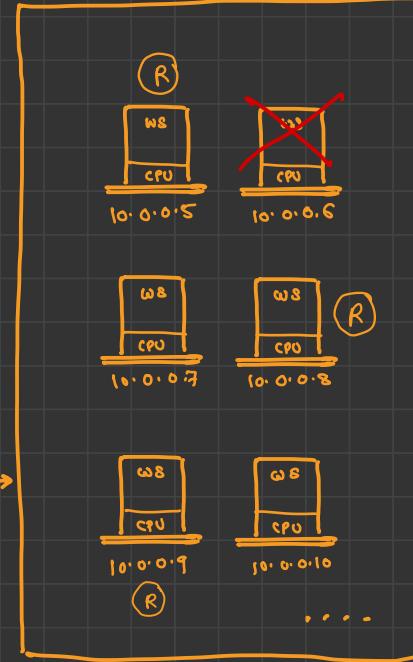
* fault isolation

* stateless



Network

Cluster of servers



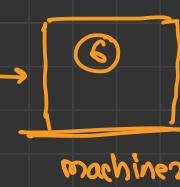
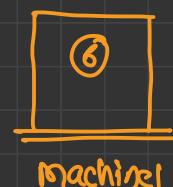
database

→ RDBMS

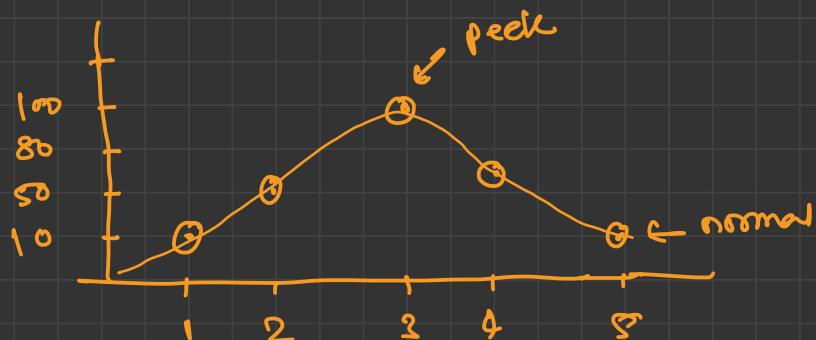
→ Primary

→ Secondary
backup

→ NoSQL



Year	load	Machines	demand q servers #	Investment	
1	10K	2	200	$5 \times 2 = 10$	$\text{maintenance} = 15^k$
2	50K	3	300	$5 \times 3 = 15$	$\text{administrator} = 50^k$
3	100K	6	600	$5 \times 6 = 30$	$\hookrightarrow \text{Operational Expenditure}$
4	60K	6	400	$5 \times 6 = 30$	$= \underline{\text{opex}}$ ✓
5	10K	6	200	$5 \times 6 = 30$	



30 L

Capital expenditure

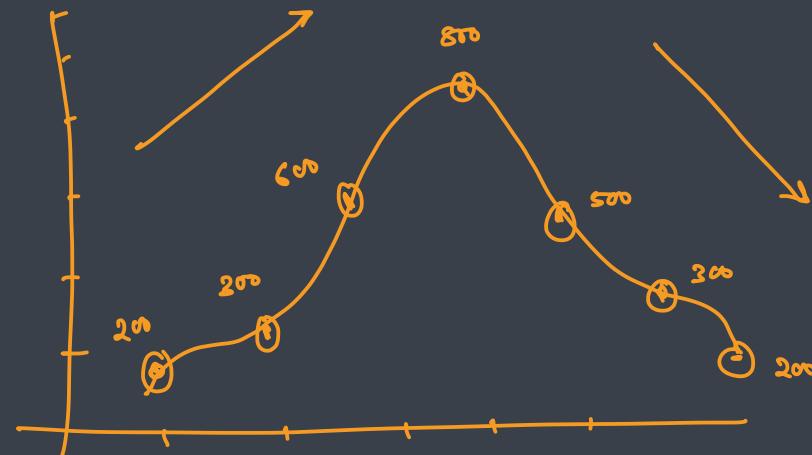
↓

CapEx



What is cloud computing ?

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





What is Data Center ?

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

- Servers *tower server*
 - Security
 - WAN
 - Storage *physical SAN NAS*
 - File Sharing
- ↑ databases . . .





Terminologies

Scalability

- refers to the idea of a system in which every application or piece of infrastructure can be expanded to handle increased load

Vertical

Horizontal

Elasticity

- the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible

adding

removing

Autoscaling

Availability

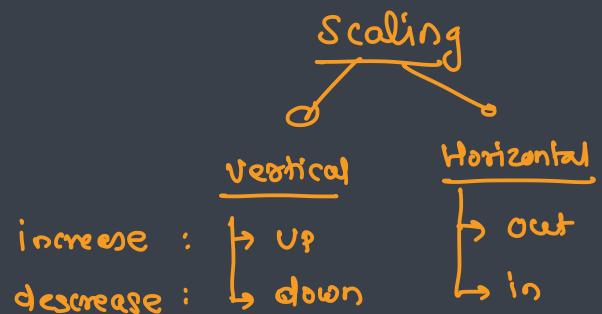
- refers to the ability of a user to access information or resources in a specified location and in the correct format

Information Assurance

- availability, integrity, authentication, confidentiality and nonrepudiation

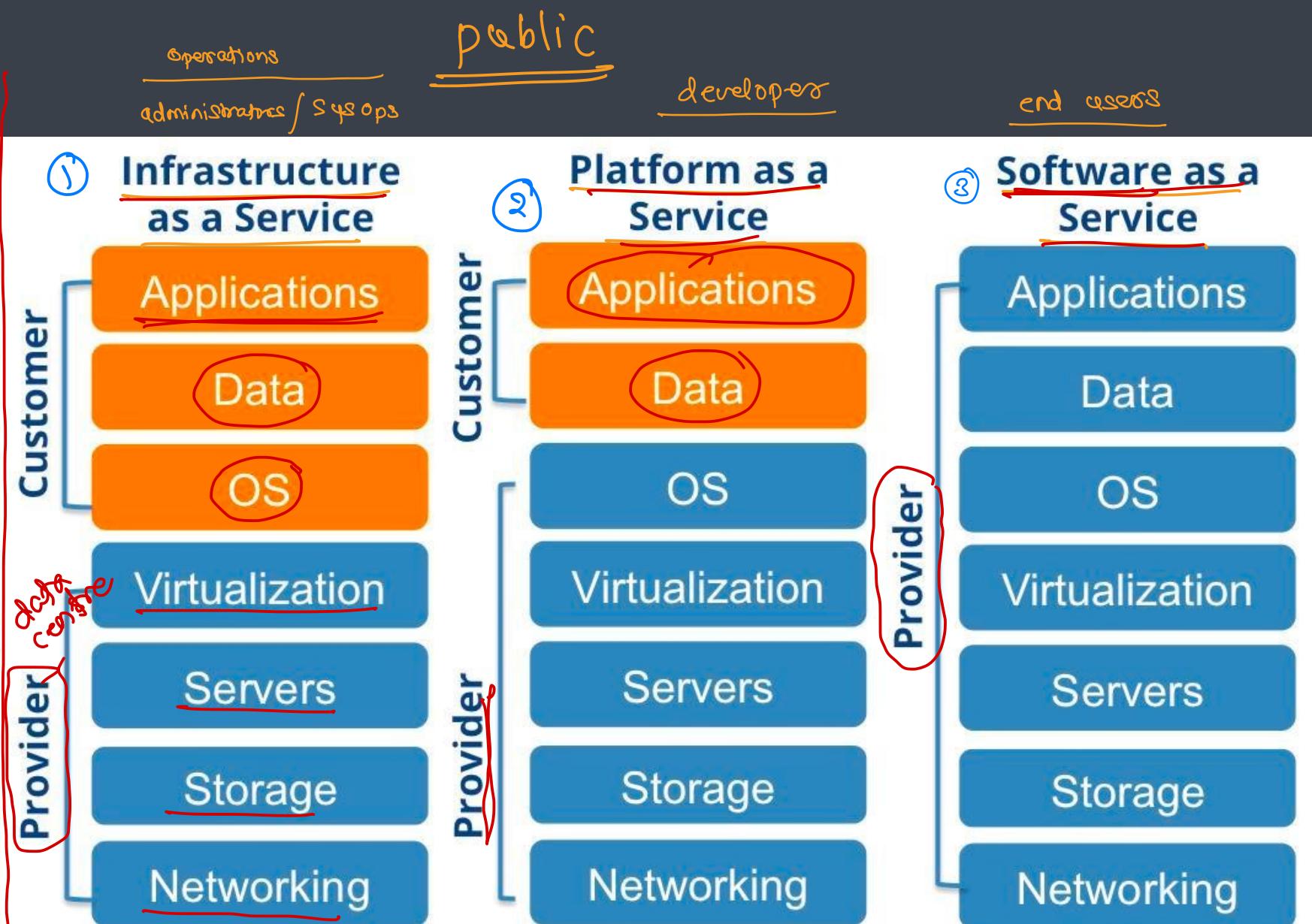
On-demand service

- A model by which a customer can purchase cloud services as needed





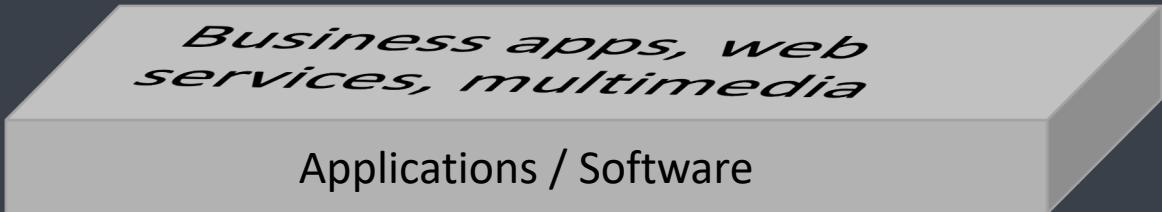
Service Models





Service Models

end user
Software
as a Service (SaaS)



Google Apps,
Facebook, YouTube,
Dropbox, Google Photos

developers
Platform
as a Service (PaaS)

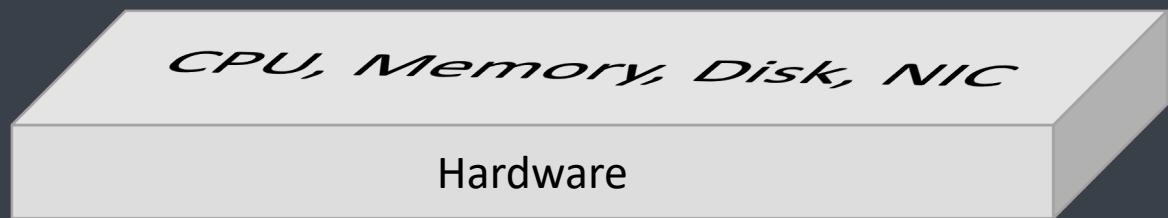


Google App Engine,
Amazon Simple DB, S3,
Microsoft Azure

operations
Infrastructure
as a Service (IaaS)



Amazon EC2,
Google Compute VM,
Azure VM



Data Center
Provider

service models

① IaaS : Infrastructure as a Service

② PaaS : Platform as a Service

③ SaaS : Software as a Service

④ faaS : function as a Service ; → Lambda Service

⑤ DaaS : Database as a Service ; → RDBMS & NoSQL

Maria
MySQL
Oracle
PostgreSQL
SQL Server

Redis
Redshift
:



Service Models: IaaS

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



Service Models: PaaS

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



Service Models: SaaS

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



Cloud Computing Characteristics

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





Cloud Deployment Models: Public

→ QoS / GCP / Azure → internet

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

→ Open



Cloud Deployment Models: Private

→ Capex + opex

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

①

②



Cloud Deployment Models: Hybrid

CapeX + OpEx

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



Cloud Services

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



Advantages

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



Disadvantages

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



Cloud Providers

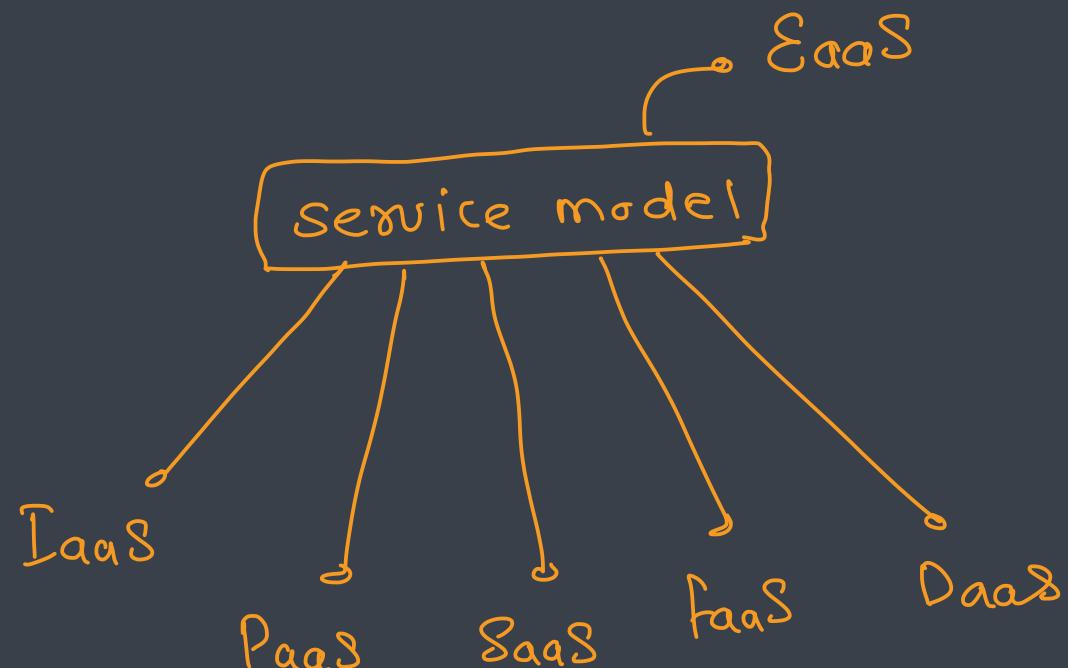
- Amazon Web Services ⭐⭐⭐
- Google Cloud Platform
- Microsoft Azure
- Rackspace
- DigitalOcean
- Alibaba Cloud
- Oracle Cloud
- IBM Cloud





What is AWS ?

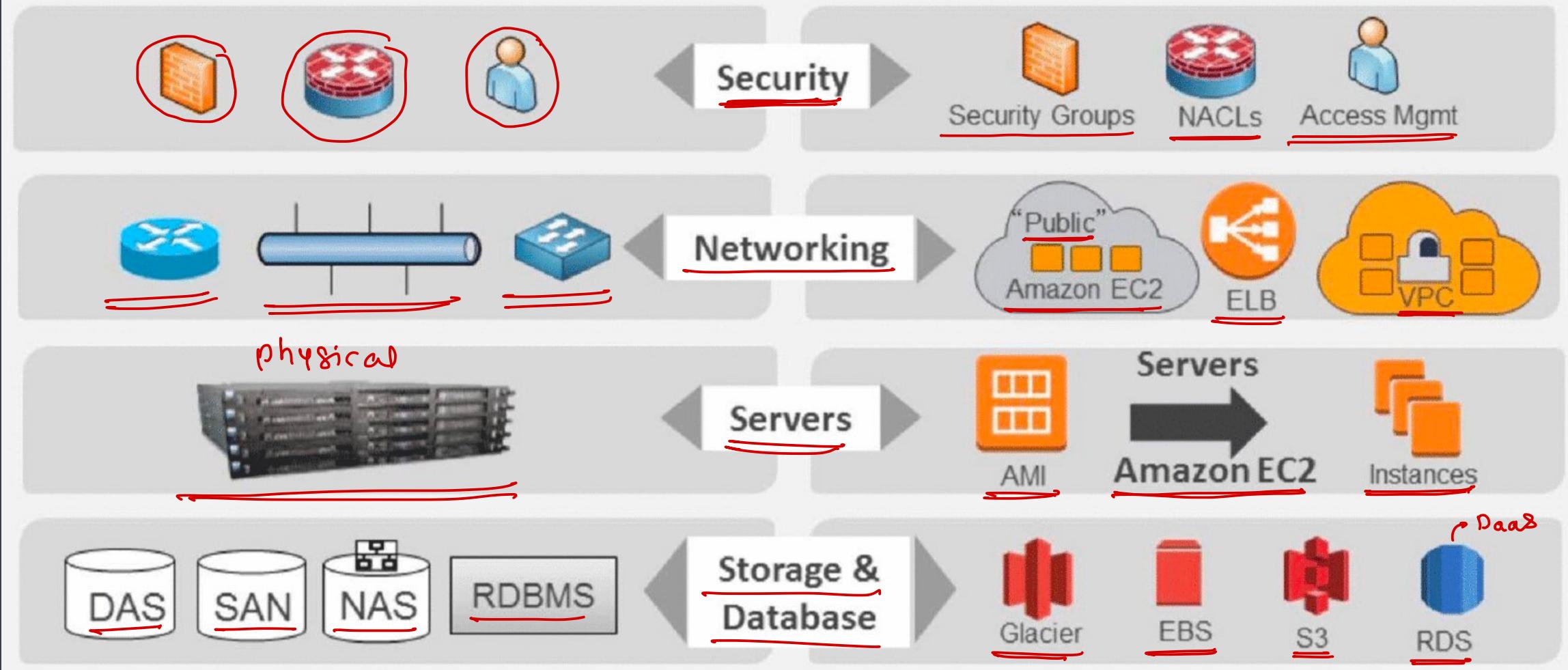
- AWS stands for Amazon Web Services
- Platform that offers flexible, reliable, scalable, easy-to-use and cost-effective cloud computing solutions
- Amazon's cloud implementation
- It's a combination of IaaS, PaaS and SaaS offerings



Traditional vs AWS

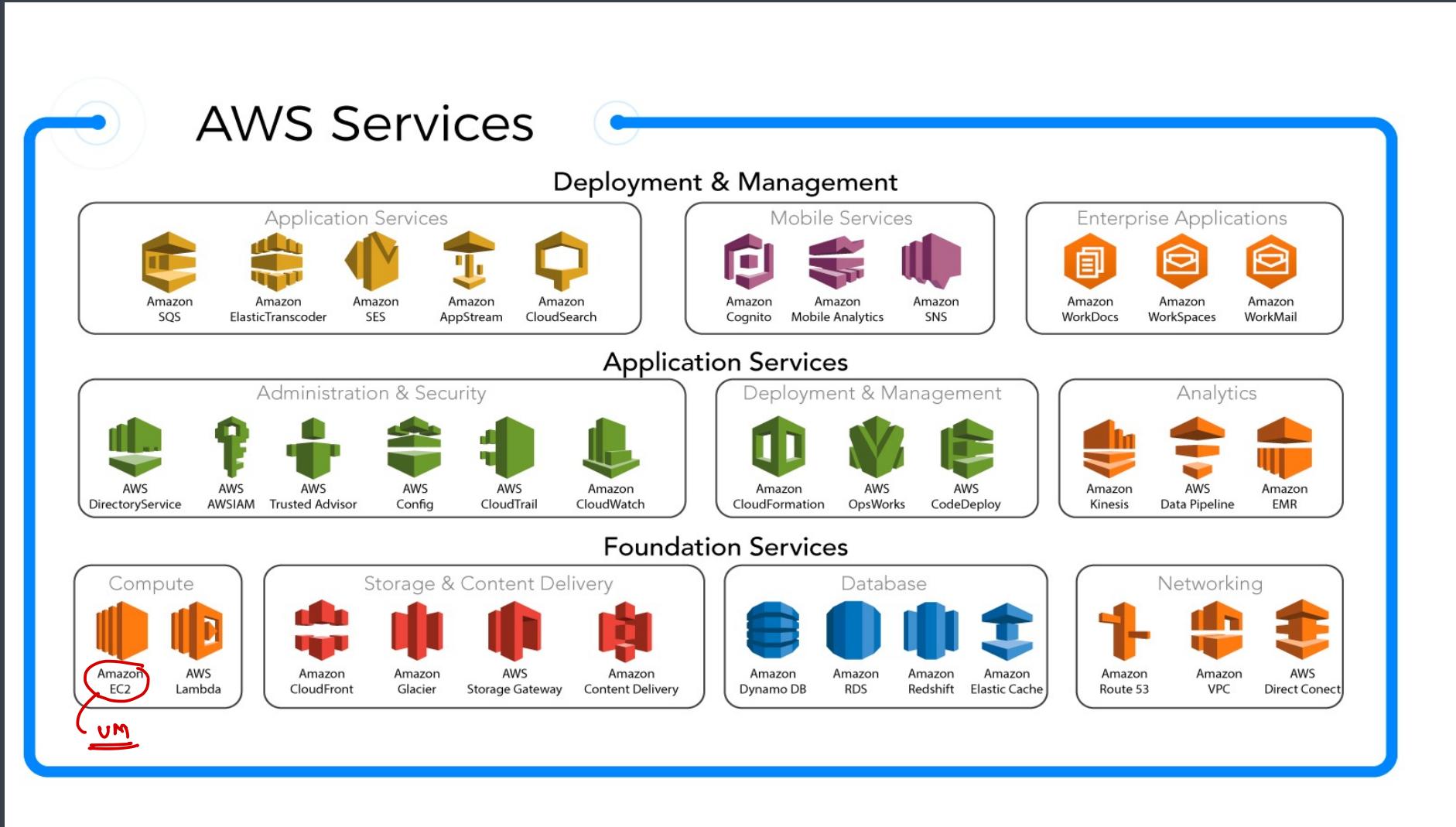


Traditional Infrastructure





AWS Services





Global Infrastructure: Region → Mumbai

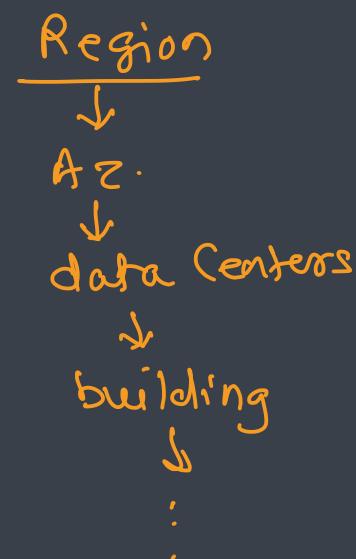
- Geographic area having availability zone(s)
- Collection of availability zones that are geographically located close to one other
- Every Region will act independently of the others, and each will contain at least two Availability Zones
- E.g.
 - US East: N. Virginia, Ohio
 - US West: N. California,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

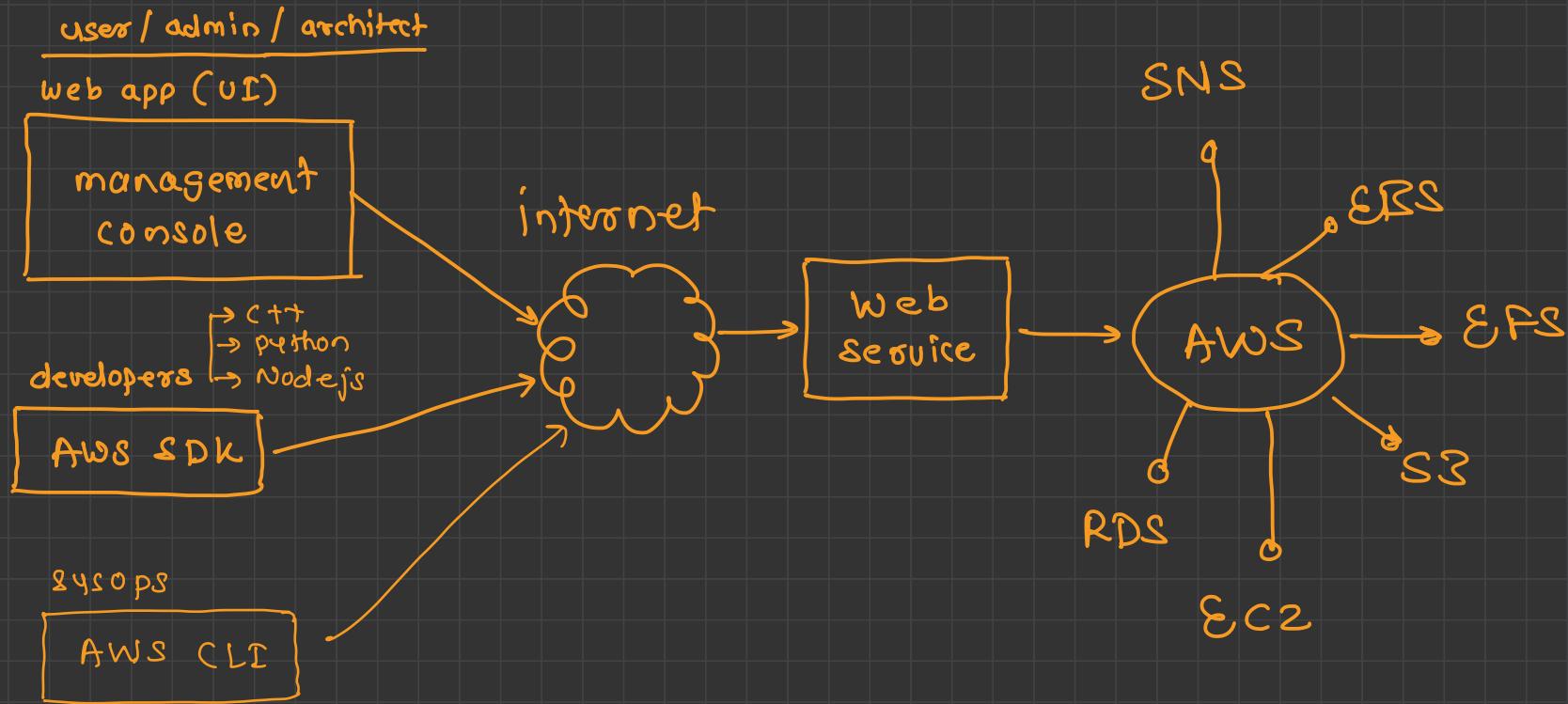


Global Infrastructure: Availability Zone

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

A.Z. = at least 2 data centers
↑







Global Infrastructure: Edge Locations

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



EC2



EC2

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



Amazon Machine Image (AMI)

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



Instance Type

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



Instance Types

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



Security Group

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



EC2 Key Pairs

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



DevOps



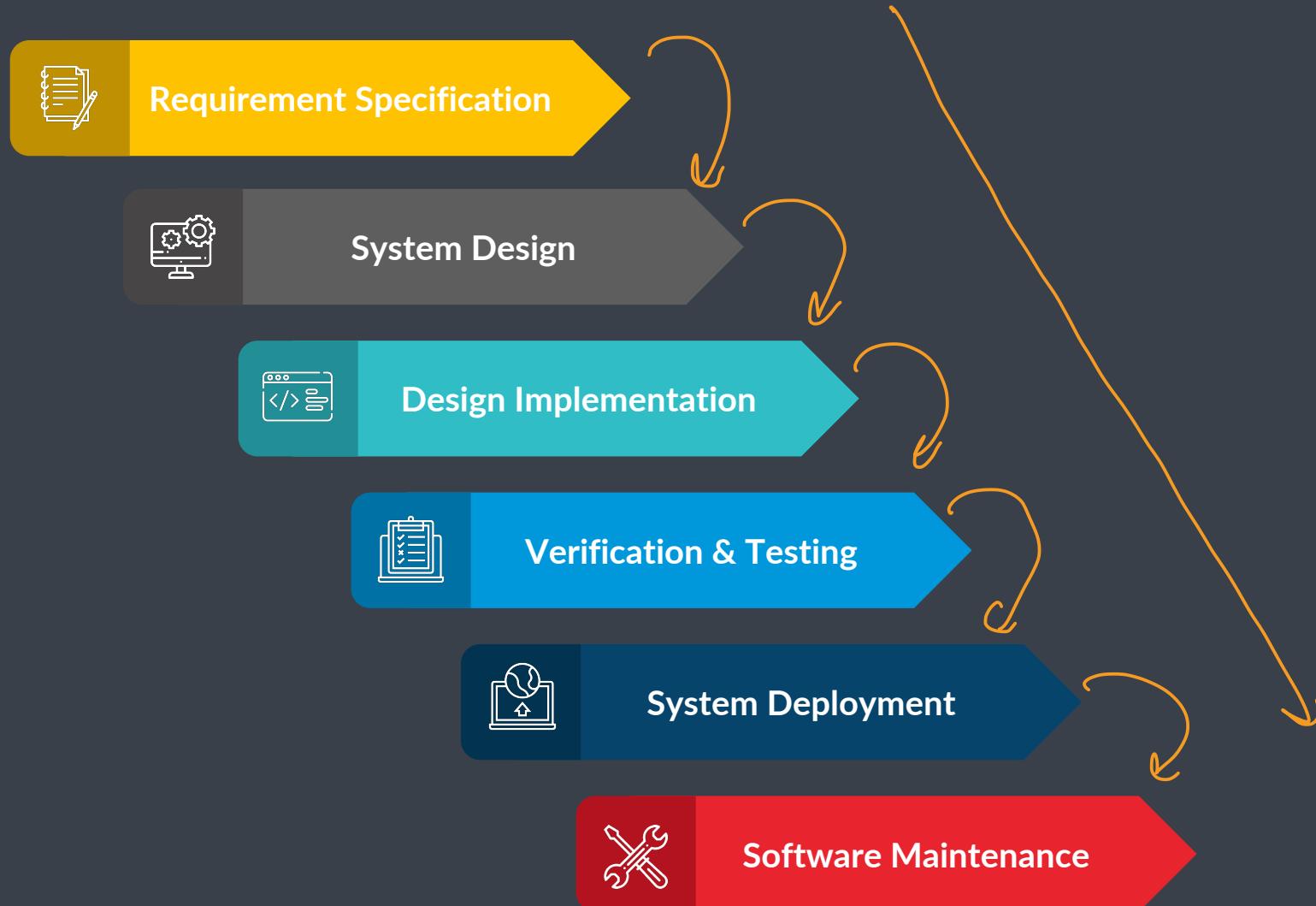


Software Development Lifecycle





Waterfall Model





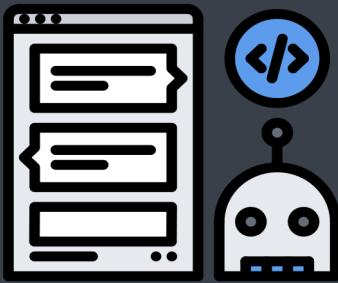
Entities involved



Developer

building

①



Testers

testing

②



Operations Team

deployment

③



Responsibilities

Dev team
↑

Developers and Testers

- Developers ↗ C, C++, C#, JS ...
 - Develop the application
 - Package the application ↘ webpack, apk, ipa
 - Fix the bugs
 - Maintain the application
- Testers
 - Thoroughly test the application manually or using test automation
 - Report the bugs to the developer



Operations Team

- Make all the necessary resources ready
- Deploy the application ↙ traditional, virtualized, containerized
- Maintain multiple environments
- Continuously monitor the application
- Manage the resources





Challenges

Developers and Testers

- The process is slow
- The pressure to work on the newer features and fix the older code
- Not flexible



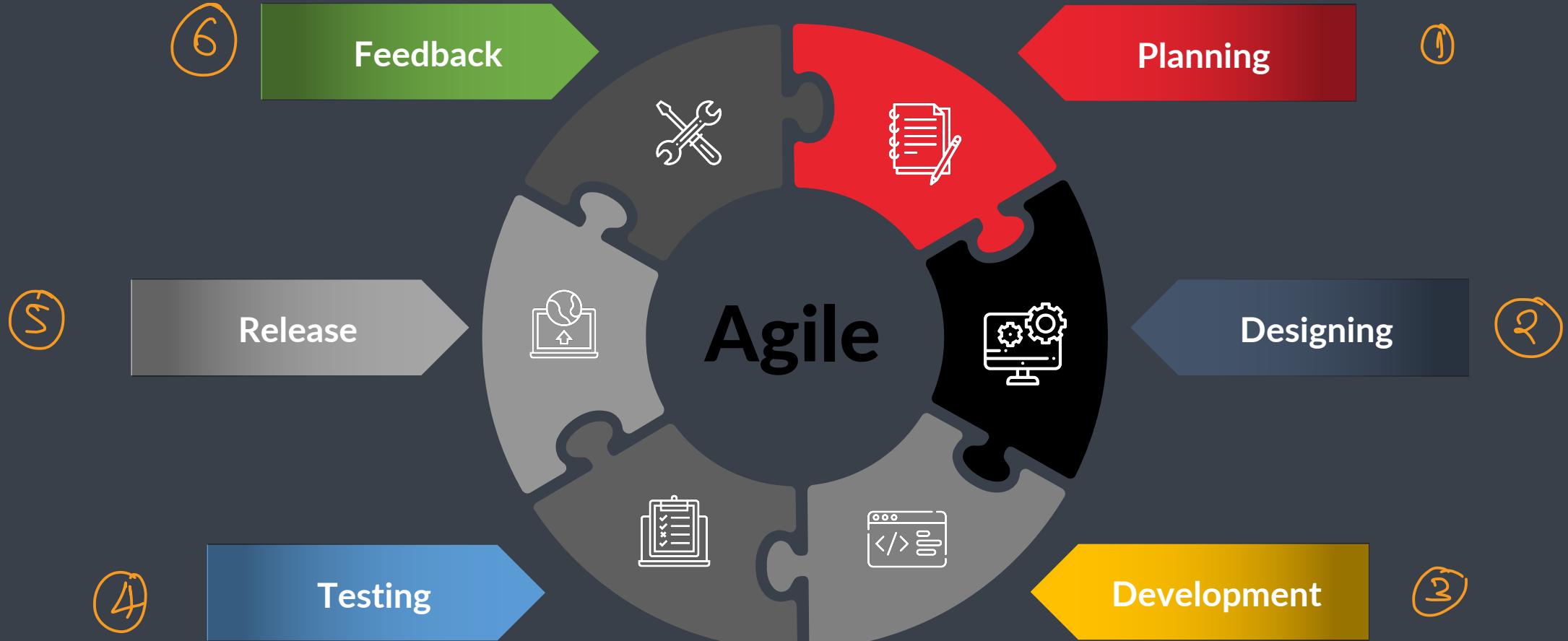
Operations Team

- Uptime 🚀
- Configure the huge infrastructure
- Diagnose and fix the issue





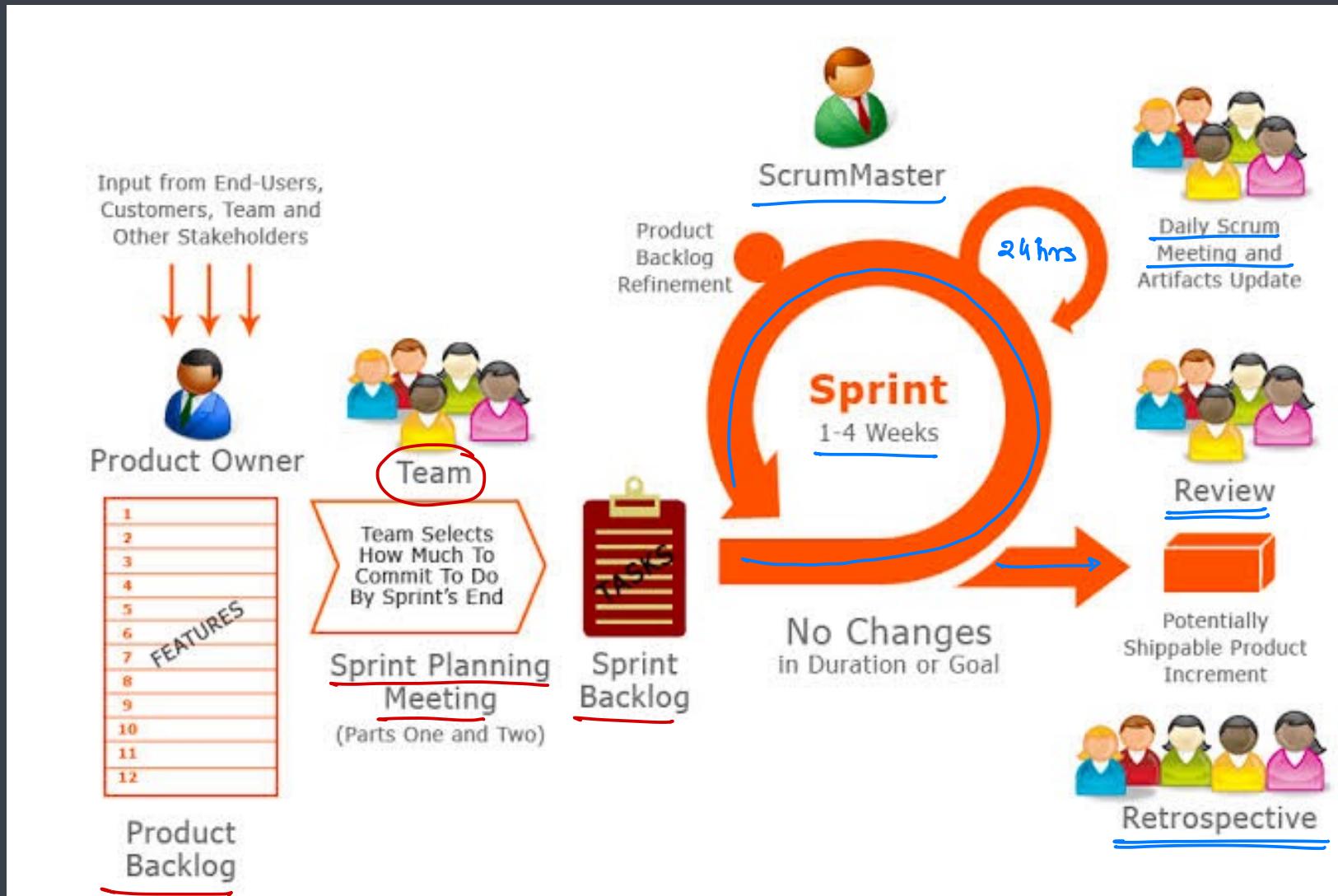
Agile Development





Scrum Process

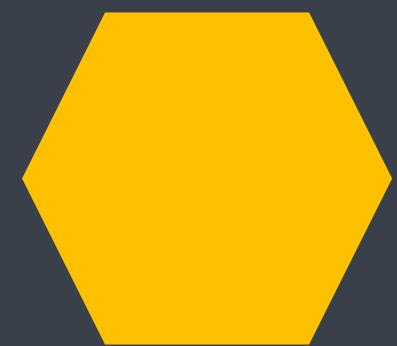
|| Continuous process ||





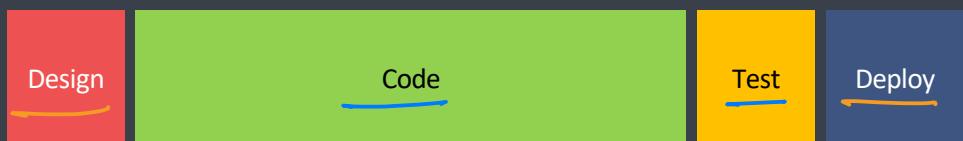
Waterfall Vs Agile

The Waterfall Process



whole

This project has got so big.
I am not sure I will be able to deliver it!



The Agile Process



task / story



It is so much better delivering
this project in bite-sized sections





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



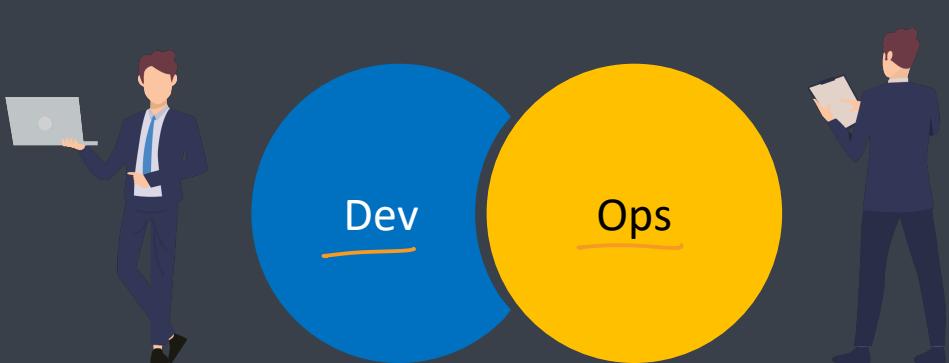
Solutions to the problem

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



What is DevOps ?

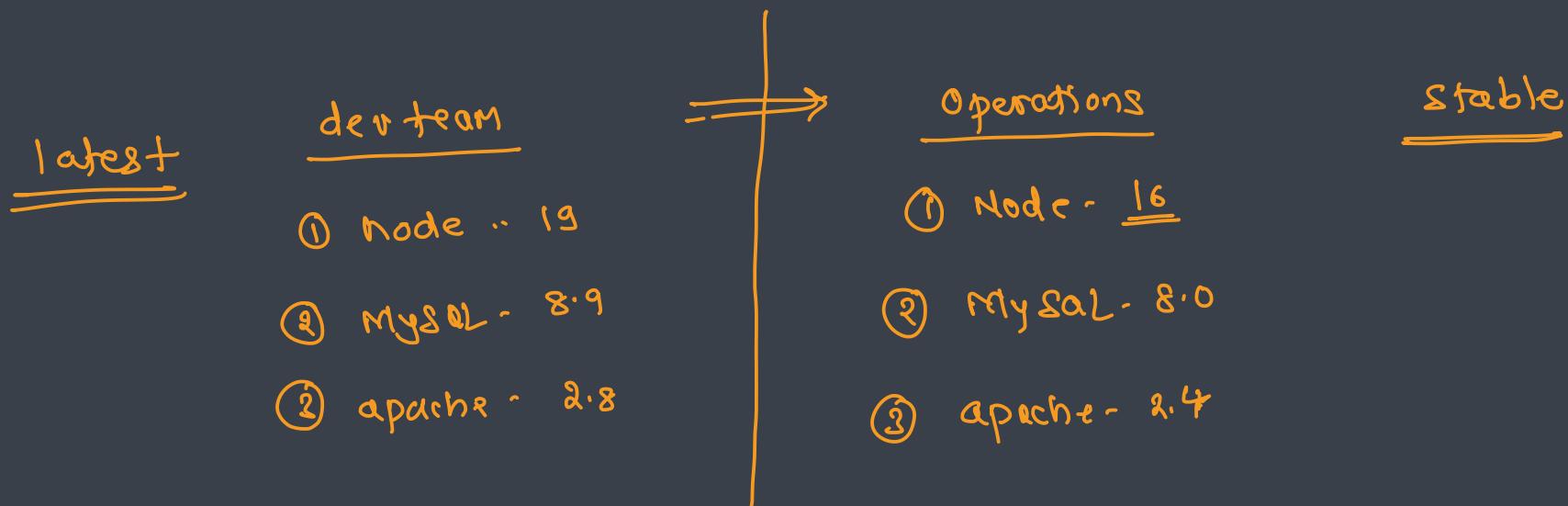
- DevOps is a combination of two words development and operations
- Promotes collaboration between Development and Operations Team to deploy code to production faster in an automated & repeatable way
- DevOps helps to increases an organization's speed to deliver applications and services
- It allows 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
- 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 → building + testing + deployment
 - Continuously measuring application performance – monitoring





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





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

DevOps – mindset / continuous
processes

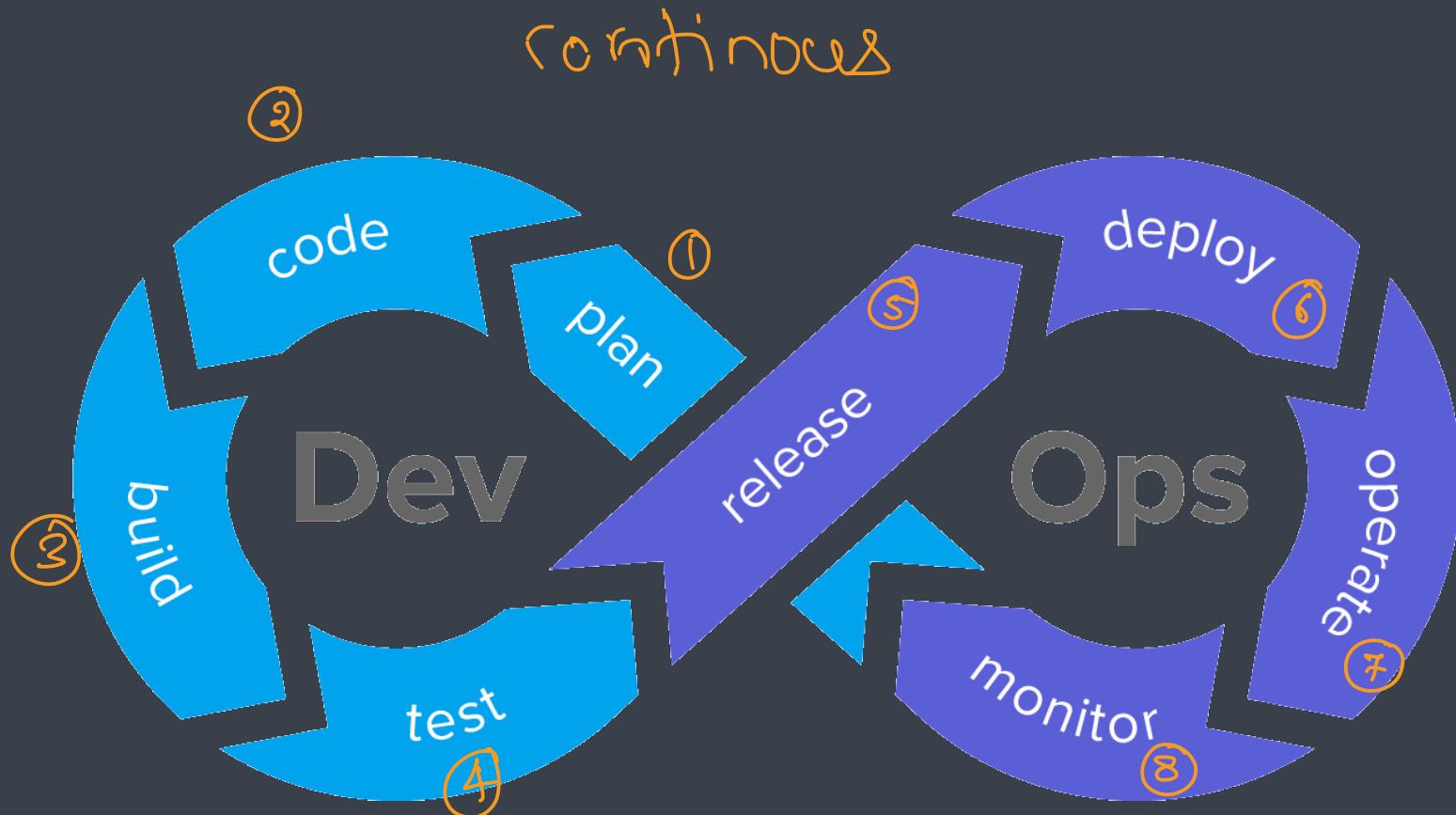


Reasons to use DevOps

- **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



DevOps Lifecycle



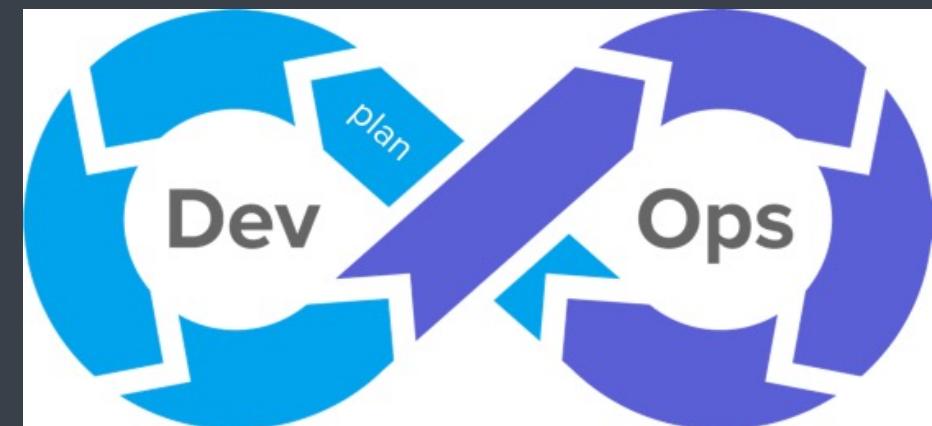
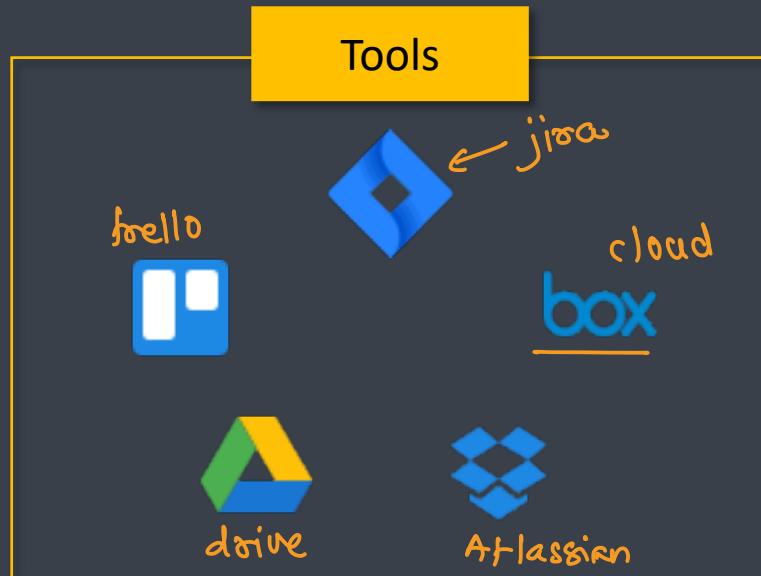


DevOps Lifecycle - Plan

- First stage of DevOps lifecycle where you plan, track, visualize and summarize your project before you start working on it

① resources

② tools → Excel, txt file, doc file

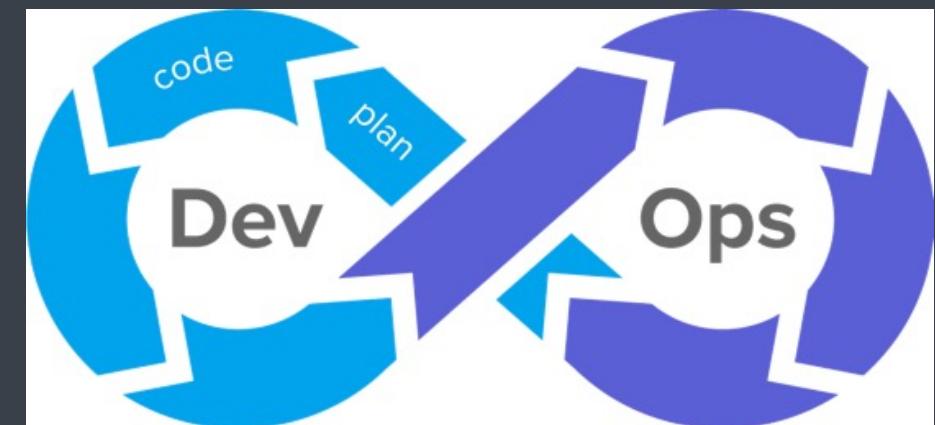
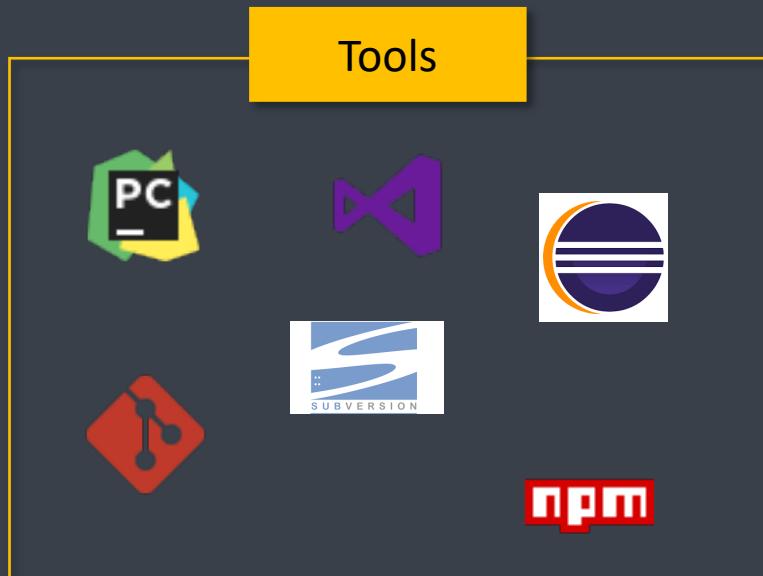




DevOps Lifecycle - Code

- Second stage where developer writes the code using favorite programming language

- ① IDEs - VS, Eclipse, PyCharm etc...
- ② Editors - VS code, vim
- ③ languages - C, C++, Java, C#, JS etc..
- ④ SCM - SVN, CVS, perforce, bazaar, git



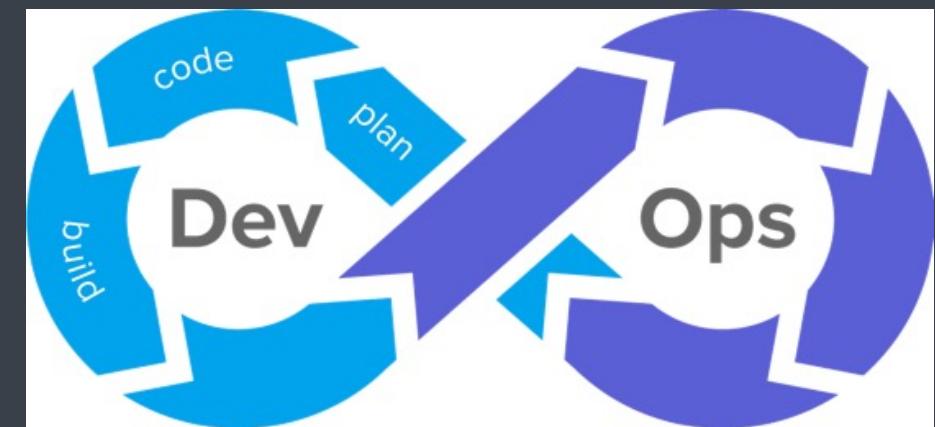


DevOps Lifecycle -Build

- Integrating the required libraries
- Compiling the source code
- Create deployable packages

- Compiling + adding resources
+ dependency resolution = deployable package

- * package managers → npm, nuget, chocolate, pip
- * build manager → ant, maven, gradle

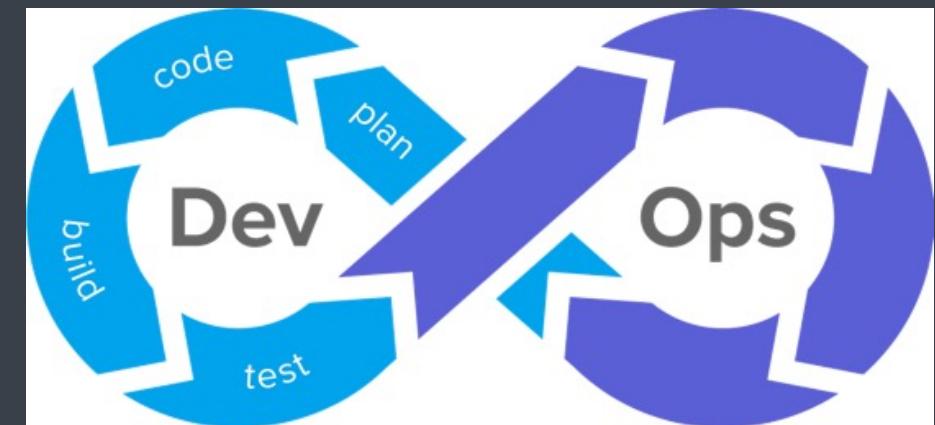
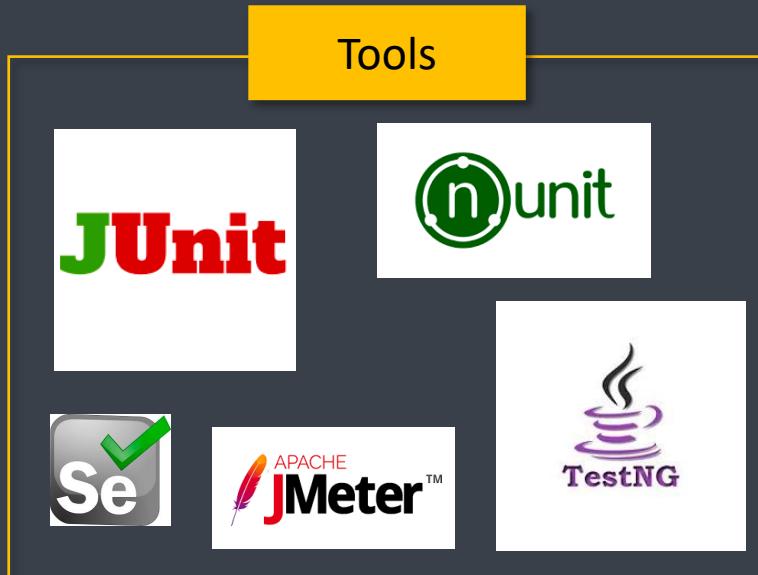




DevOps Lifecycle - Test

- Process of executing automated tests
- The goal here is to get the feedback about the changes as quickly as possible

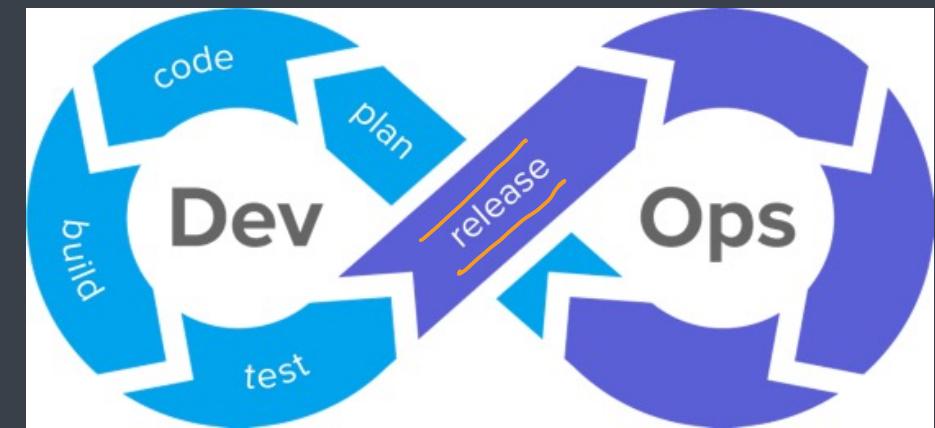
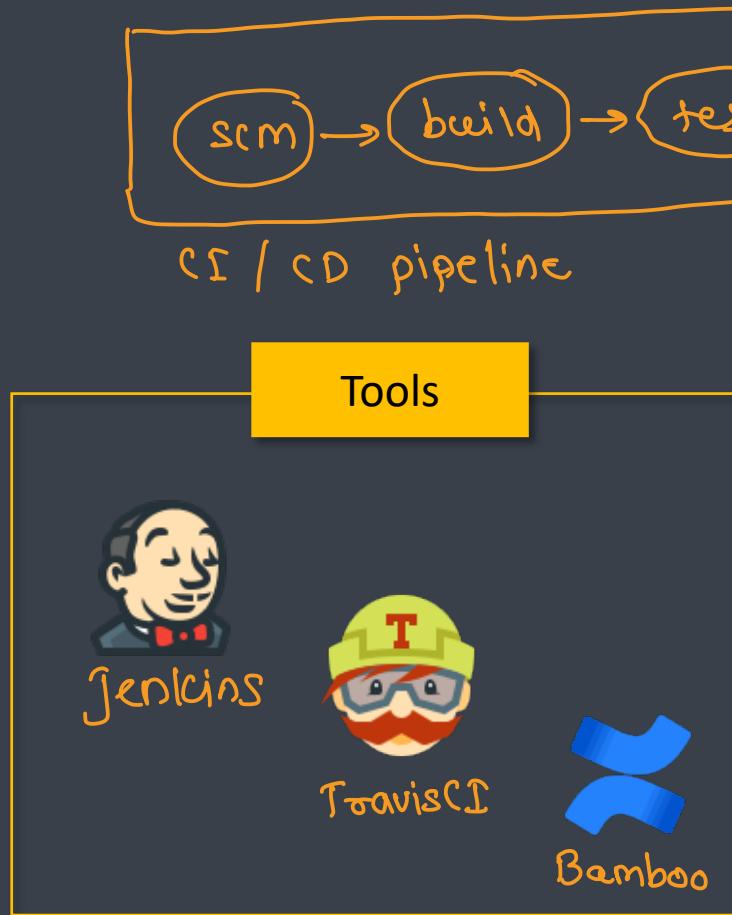
- ① unit testing → JUnit, NUnit, Jasmine,
- ② UI testing → selenium, cypress
- ③ load stressing → JMeter, loadrunner etc





DevOps Lifecycle - Release

- This phase helps to integrate code into a shared repository using which you can detect and locate errors quickly and easily





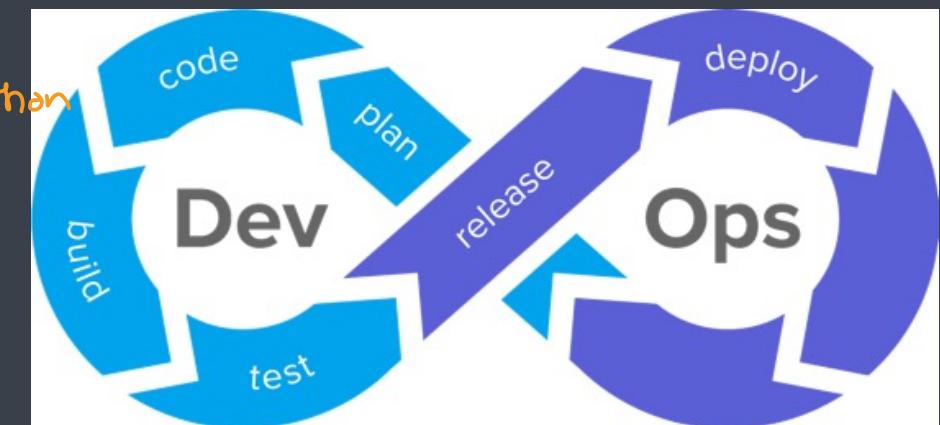
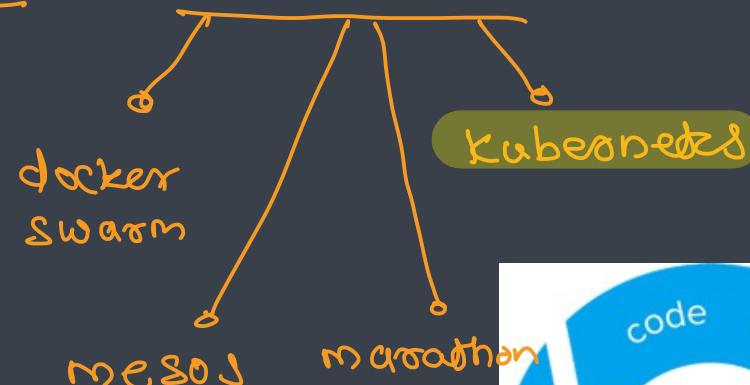
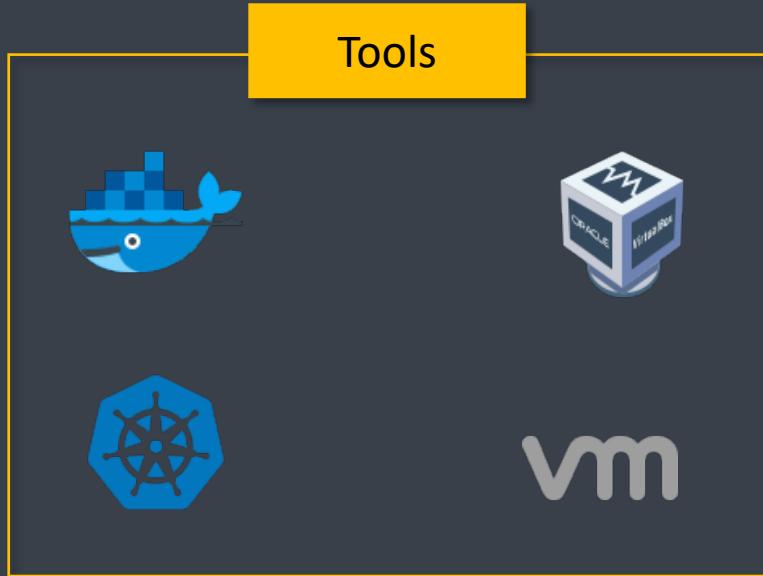
DevOps Lifecycle - Deploy

- Manage and maintain development and deployment of software systems and server in any computational environment

① traditional → physical machines

② virtualized → using virtual machines → autoScaling

③ containerization → using containers / Docker → orchestration

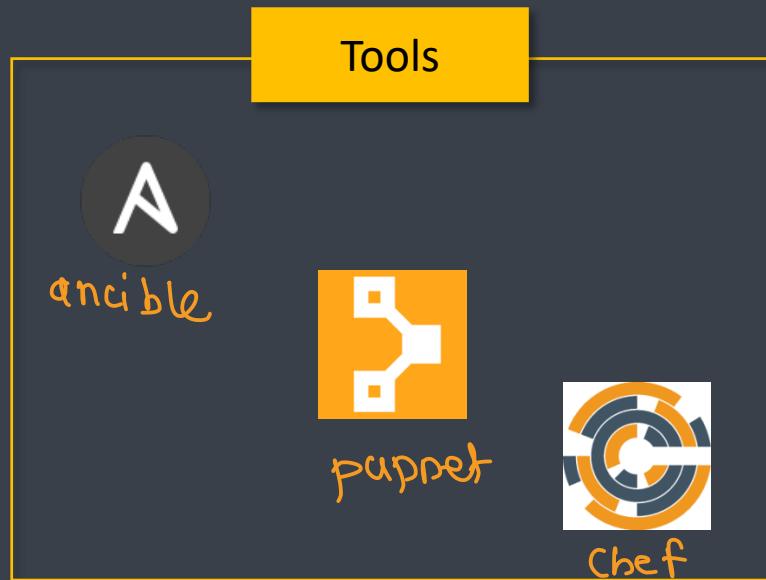




DevOps Lifecycle - Operate

- This stage where the updated system gets operated

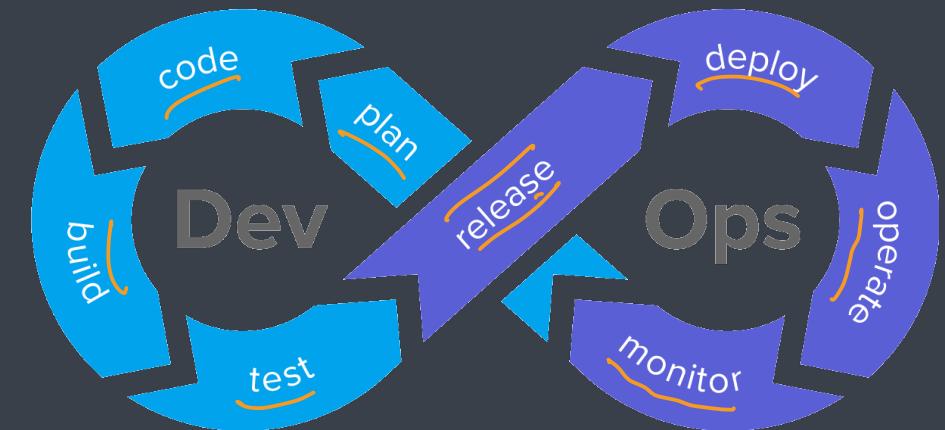
Configuration tools - puppet / chef/ ansible





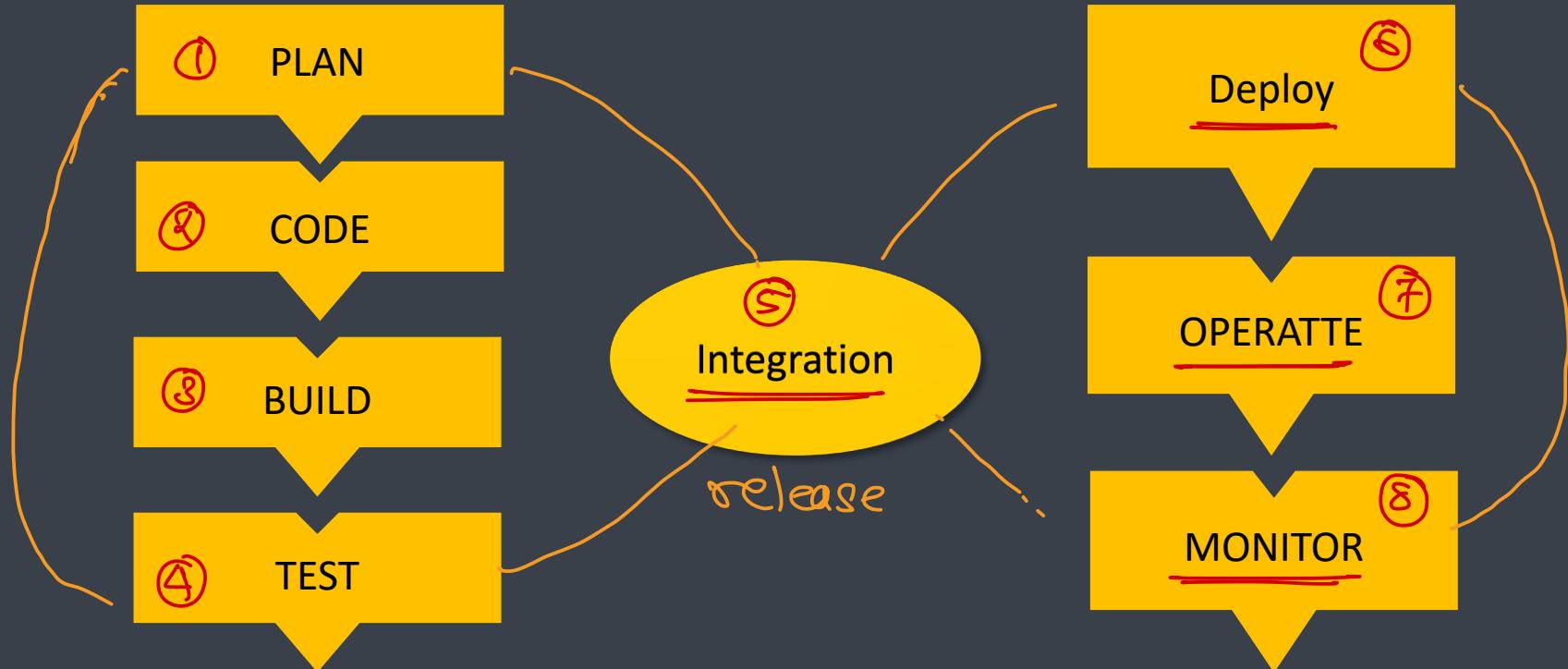
DevOps Lifecycle - Monitor

- 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





DevOps Terminologies

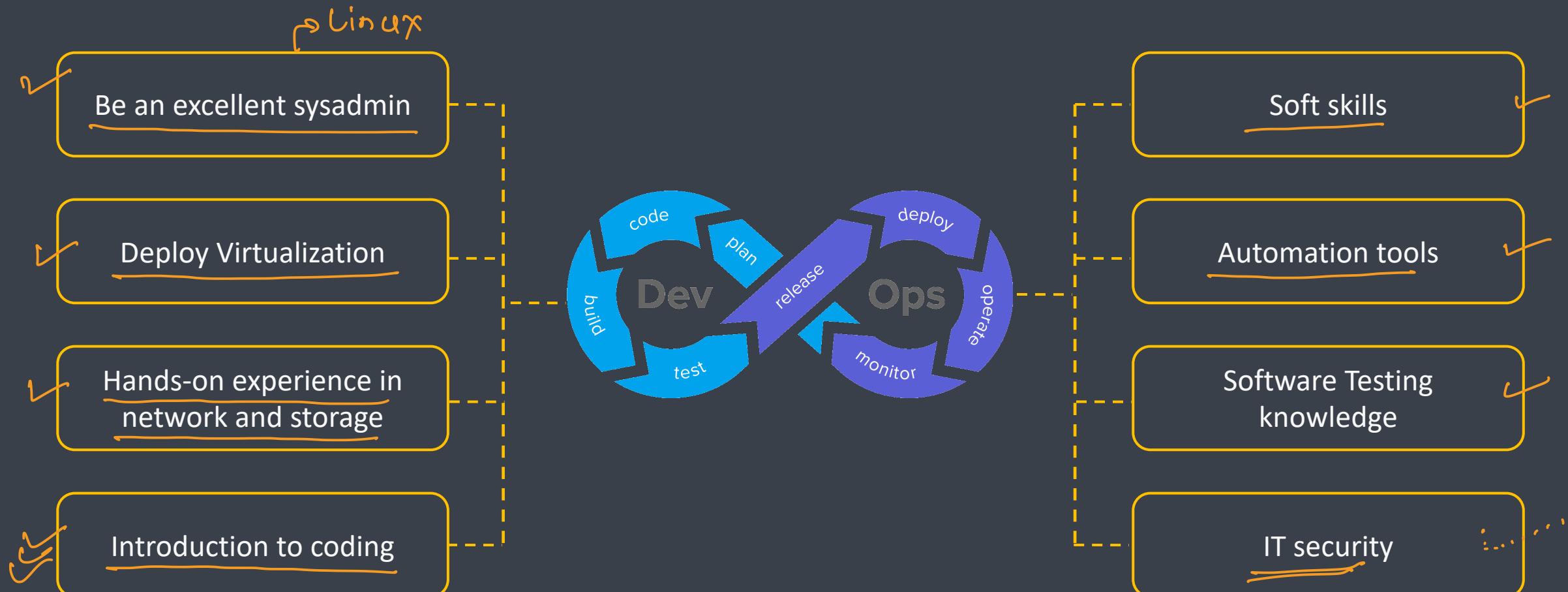


* continuous processes

- ① continuous development \Rightarrow plan + code
- ② continuous testing \Rightarrow build + test
- ③ continuous integration \Rightarrow integration / release
- ④ continuous deployment \Rightarrow deploy
- ⑤ continuous configuration \Rightarrow operation
- ⑥ continuous monitoring \Rightarrow monitoring



Responsibilities of DevOps Engineer



Skills of a DevOps Engineer



Skills	Description
Tools	<ul style="list-style-type: none">• Version Control – Git/SVN• Continuous Integration – Jenkins• Virtualization / Containerization – Docker/Kubernetes• Configuration Management – Puppet/Chef/Ansible• Monitoring – Nagios/Splunk
Network Skills	<ul style="list-style-type: none">• General Networking Skills• Maintaining connections/Port Forwarding
Other Skills	<ul style="list-style-type: none">• Cloud: AWS/Azure/GCP• Soft Skills• People management skill

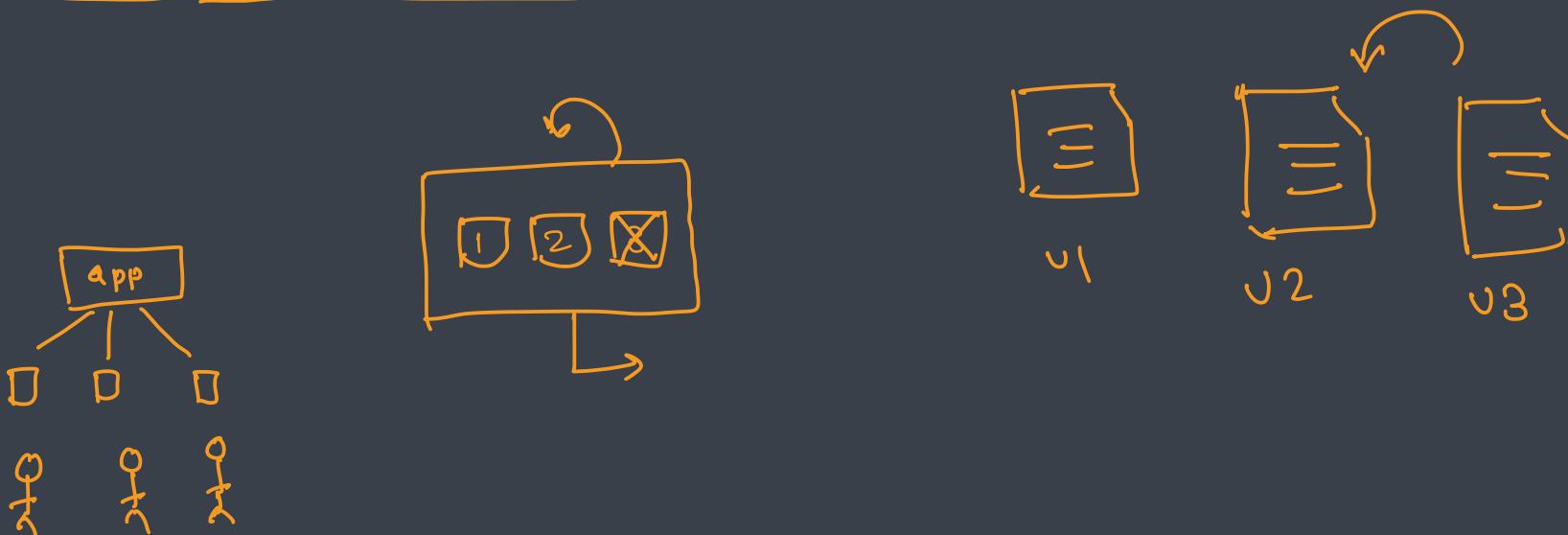


Source Code Management / Version Control System



Version Control System

- Version control is a system that allows the software team to manage changes to the source code over time
- This software tool makes it easier for developers to collaborate on different projects separating their tasks through branches
- It also gives the possibility to turn back to earlier versions for comparing and fixing the mistakes if needed
- Version Control Systems (VCS) also known as SCM (Source Code Management) or RCS (Revision Control System) are software tools for keeping track of changes to the source code over time





Benefits

git log

Long-term change history

- The changes made by developers, including the creating, modification, and deletion of files over the years, can be seen in history
- It will allow going back to the previous version for analyzing bugs and fixing problems

Branching and merging *

- Branching helps work in an independent manner and not interfere with each other's work
- Merging brings the works together and allows seeing if there are conflicts between those works

Traceability

- Ability to trace each change and connect it to project management and bug tracking software, as well as to annotate each change with a message describing the purpose of the change

Synchronization

- The up-to-date codes can be fetched from the repository

Backup and Restore

- Files are saved at any time and restored from the last saved one

Undoing

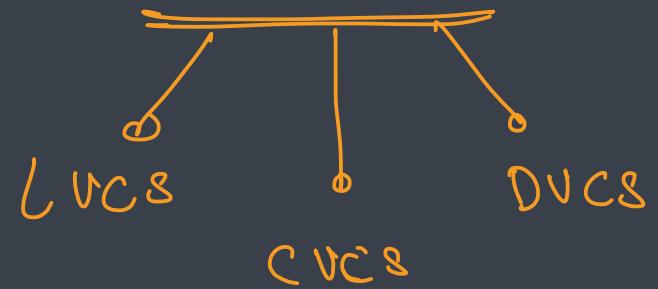
- You can undo both the last known version and the last one created a long time ago

Branching and Merging *

- Changes are made on a branch and after being approved, they can be merged with the master branch



Types



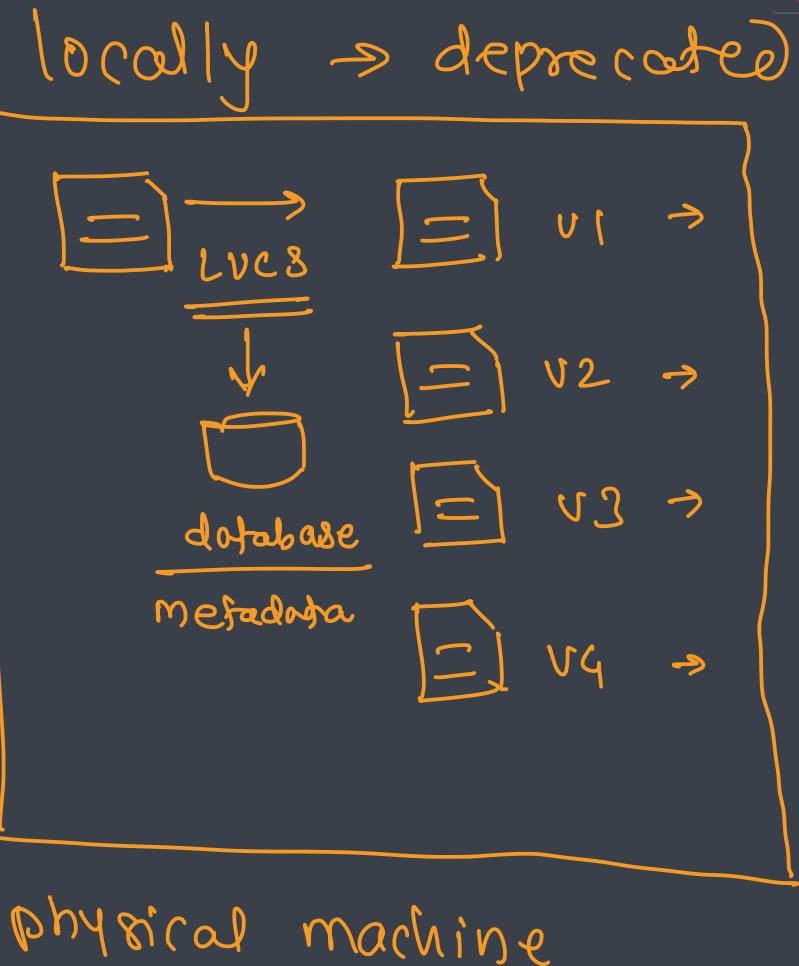


Local Version Control System

→ SPOF

- Local VCSs were created to prevent issues like confusing the directories and accidentally writing or copying to the wrong file
- It is a simple database that keeps all the changes to files under revision control
- One of the most popular VCS tools was a system called Revision Control System (RCS), which is still distributed today, although being an earlier version control system
- It allows users to make their revisions of a document, commit changes, and merge them. RCS was originally developed for programs but is also useful for text documents or configuration files that are frequently revised

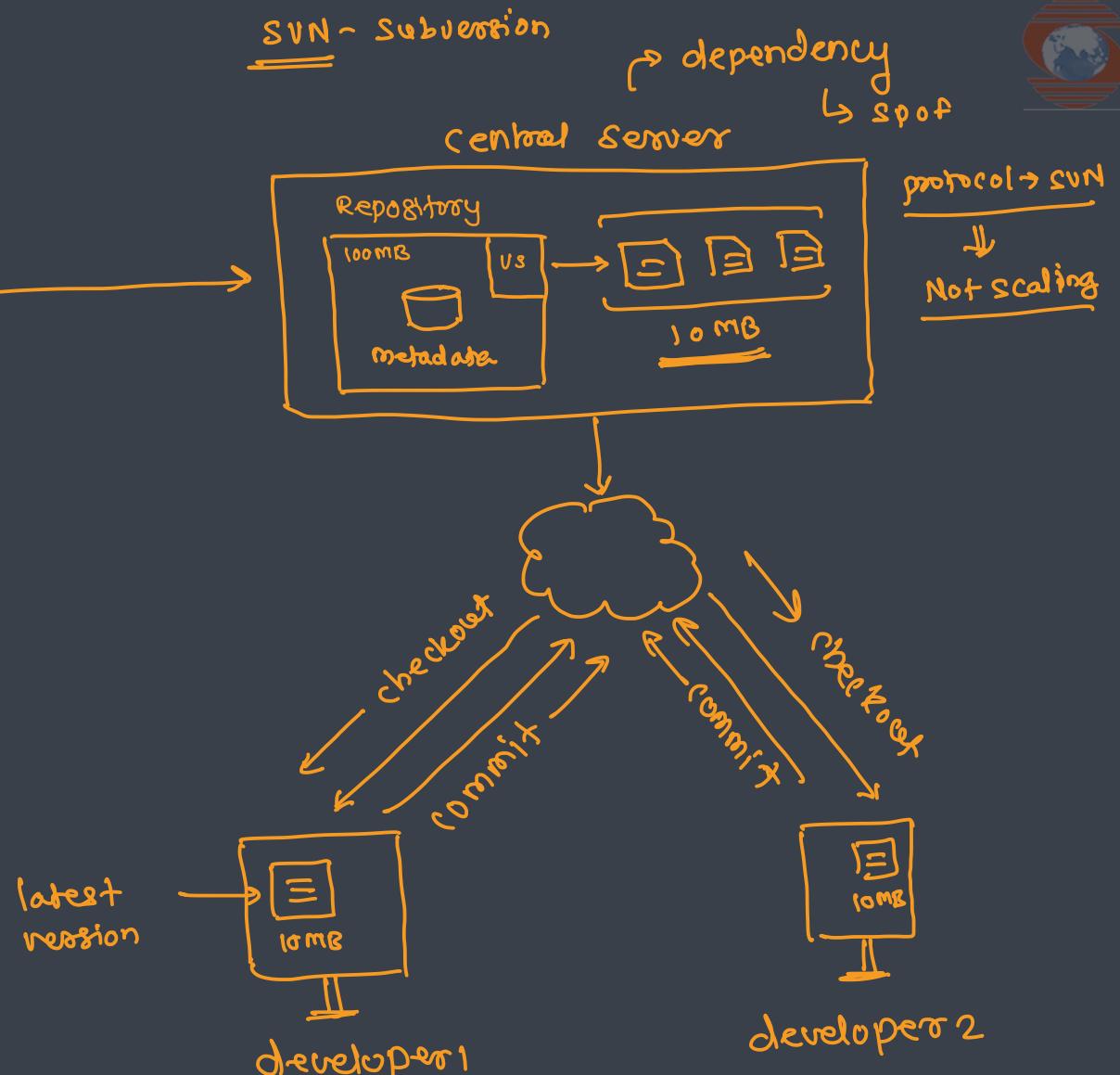
e.g. RCS, SourceSafe (ms)



Centralized Version Control System

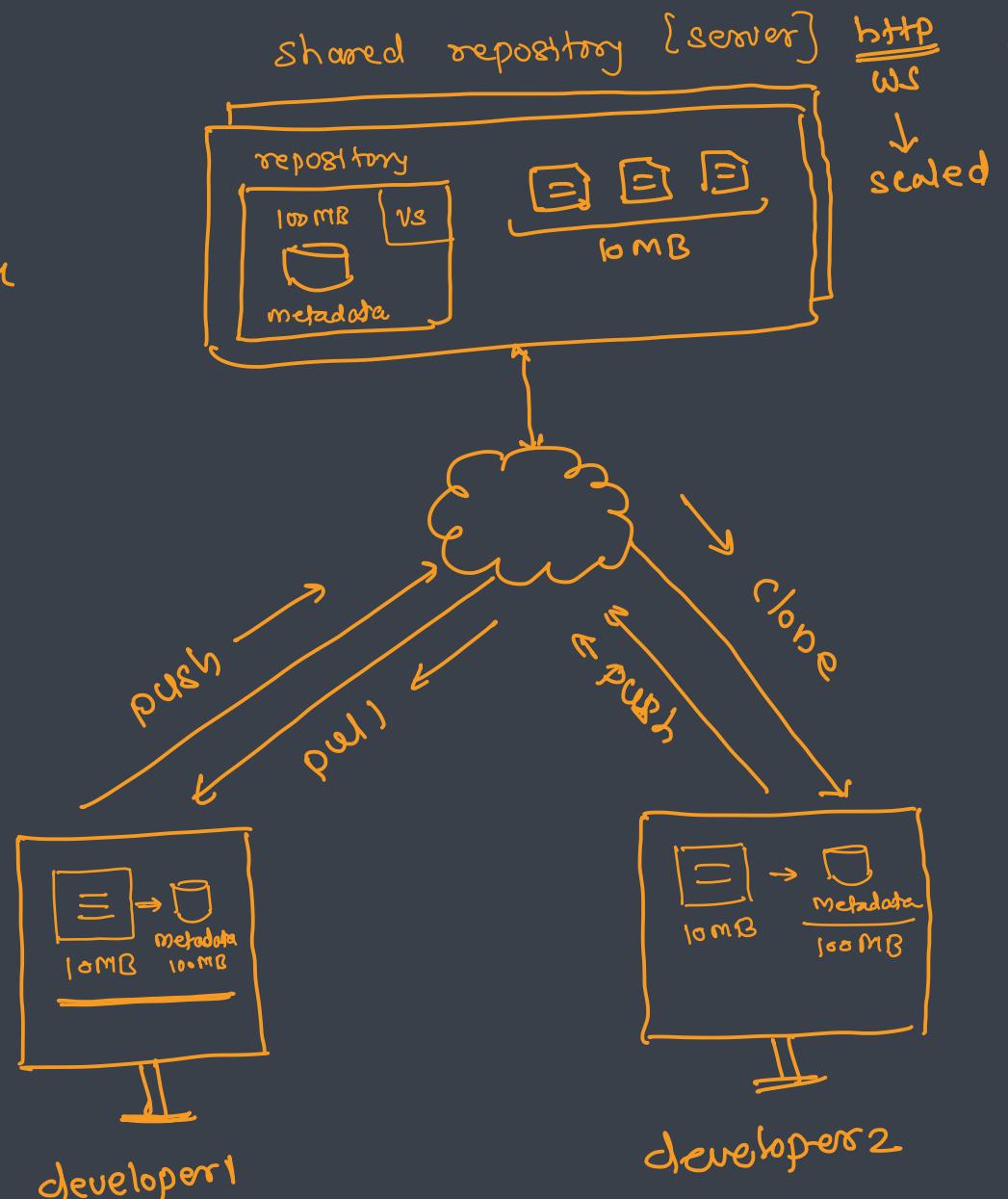
- In centralized systems, all the versioned files, as well as a number of clients that check out files from that central place, are included in a single server
- For many years, this has been the standard for version control
- Centralized Version Control Systems are CVS, Subversion, and Perforce

- Collaboration
- SPOF → Server



Distributed Version Control System

- In Distributed Version Control Systems (DVCS), clients fully mirror the repository, including its full history — metadata
- If the server that these systems were collaborating with dies, the client repositories can be copied back up to the server to restore it
- Distributed Version Control Systems are **Git, Mercurial, Bazaar or Darcs**





Git



What is Git ?

- 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

GNU → GNU is not unix , XNU
↳ Gu is not unix
↳ GN is not unix
↳ NU
recursive acronym



History

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



Characteristics

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

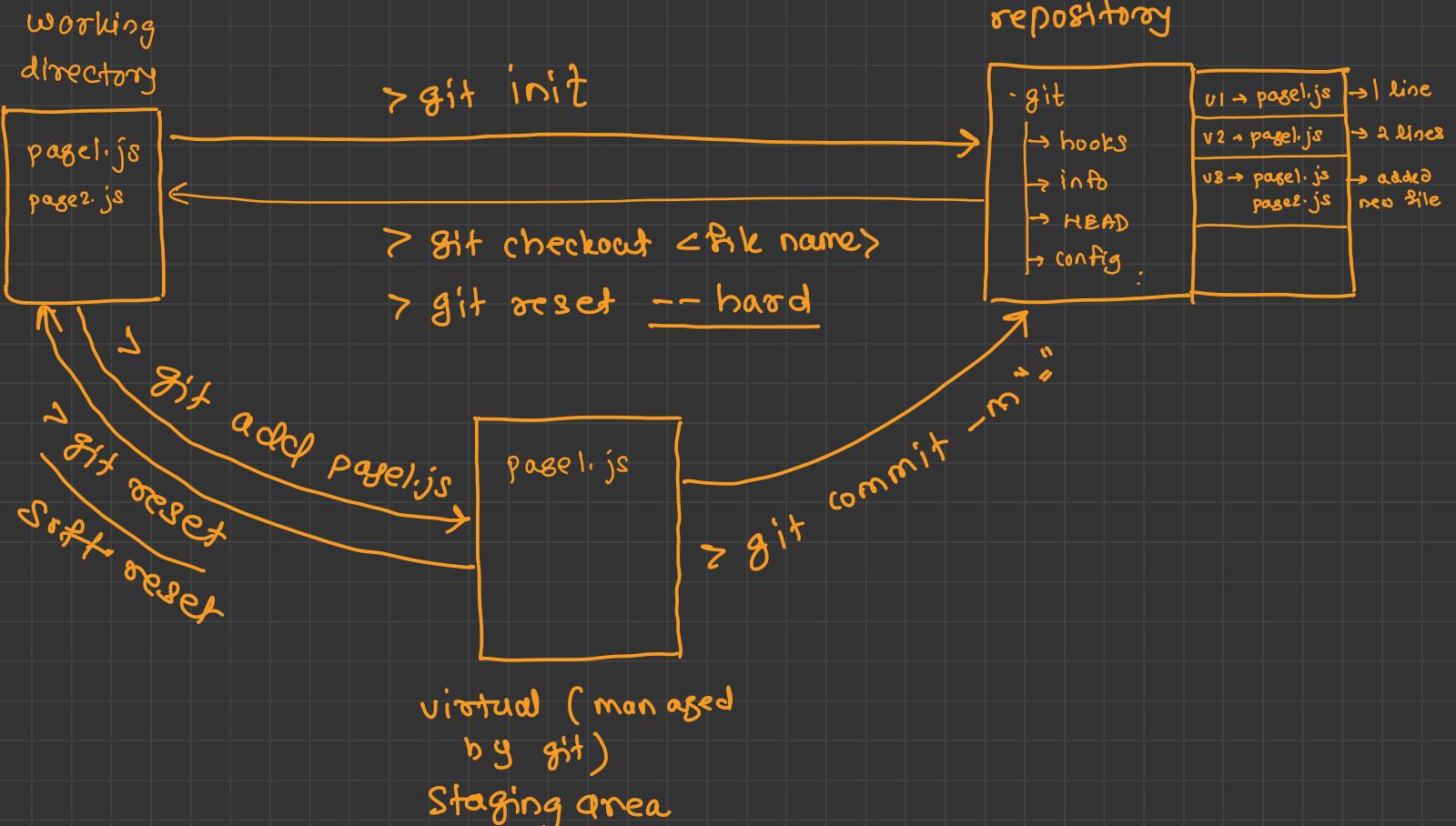


Advantages

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



Basic Workflow



? git status -s

[?] [?]

→ untracked file → git does not find any information of this file in its repository

[A] []

→ the changes are staged & the file(s) will be added to the repo

[] [m]

→ the file is modified but changes have not yet moved to the staging area [unstaged changes]

[m] []

→ the file modified. the changes are staged [the changes are moved to staging area]



git init

- The `git init` command is used to generate a new, empty Git repository or to reinitialize an existing one
- With the help of this command, a `.git` subdirectory is created, which includes the metadata, like subdirectories for objects and template files, needed for generating a new Git repository



git config

- The git config command is a function that sets configuration variables
- It controls git look and operation
- Levels
 - Local (--local) → repository specific
 - When no configuration option is passed git config writes to a local level, by default
 - The repository of the .git directory has a file that stores local configuration values
 - Global (--global) → per user
 - The application of the global level configuration includes the operating system user
 - Global configuration values can be found in a file placed in a user's home directory
 - System (--system) → per machine
 - The System-level configuration includes all users on an operating system and all repositories
 - System-level configuration file is located in a git config file of the system root path



git add

- The git add is a command, which adds changes in the working directory to the staging area
- With the help of this command, you tell Git that you want to add updates to a certain file in the next commit
- But in order to record changes, you need to run git commit too
- In combination with the commands mentioned above, git status command is also needed to see which state the working directory and the staging area are in



git commit

- The git commit command saves all currently staged changes of the project
- Commits are created to capture the current state of a project
- Committed snapshots are considered safe versions of a project because Git asks before changing them
- Before running git commit command, git add command is used to promote changes to the project that will be then stored in a commit
- **Working of commit**
 - Git snapshots are committed to the local repository
 - Git creates an opportunity to gather the commits in the local repository, rather than making a change and commit it immediately to the central repository
 - This has many advantages splitting up a feature into commits, grouping the related commits, and cleaning up local history before committing it to the central repository
 - This also gives the developers an opportunity to work in an isolated manner



git log

- The git log command shows committed snapshots
- It is used for listing and filtering the project history, and searching for particular changes
- The git log only works on the committed history in comparison with git status controlling the working directory and the staging area



Branching



Branching

- Branching allows developers to branch out from the original code base and work separately
- 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



git branch

- The git branch command creates, lists and deletes branches
- It doesn't allow switching between branches or putting a forked history back together again
- Git branches are a pointer to a snapshot of the changes you have made
- A new branch is created to encapsulate the changes when you want to fix bugs or add new features
- This helps you to clean up the future's history before merging it
- Git branches are an essential part of everyday workflow
- Git does not copy files from one directory to another, it stores the branch as a reference to a commit



Monolithic Architecture

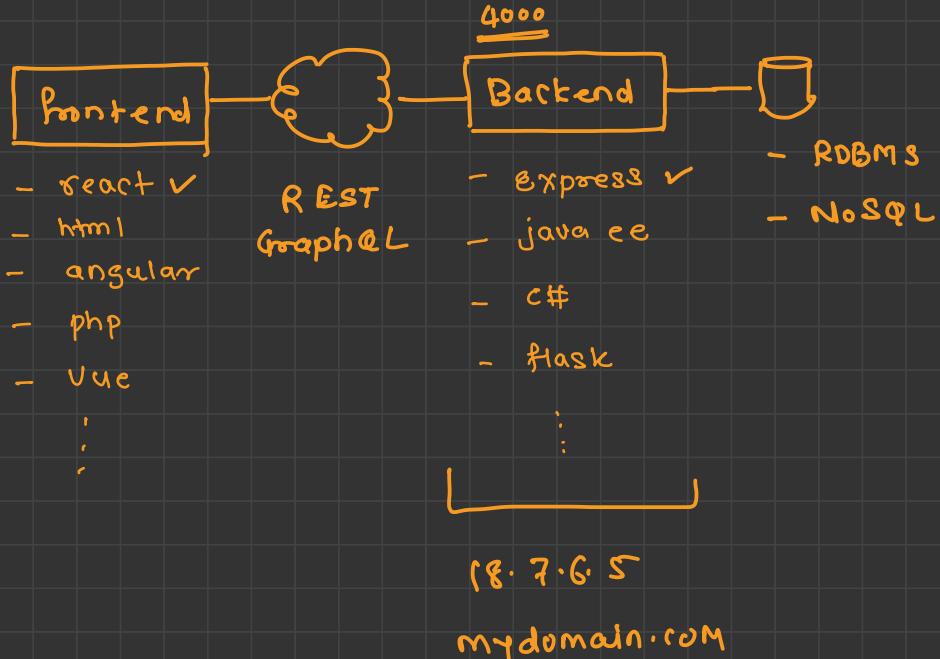
one



one package

E-commerce

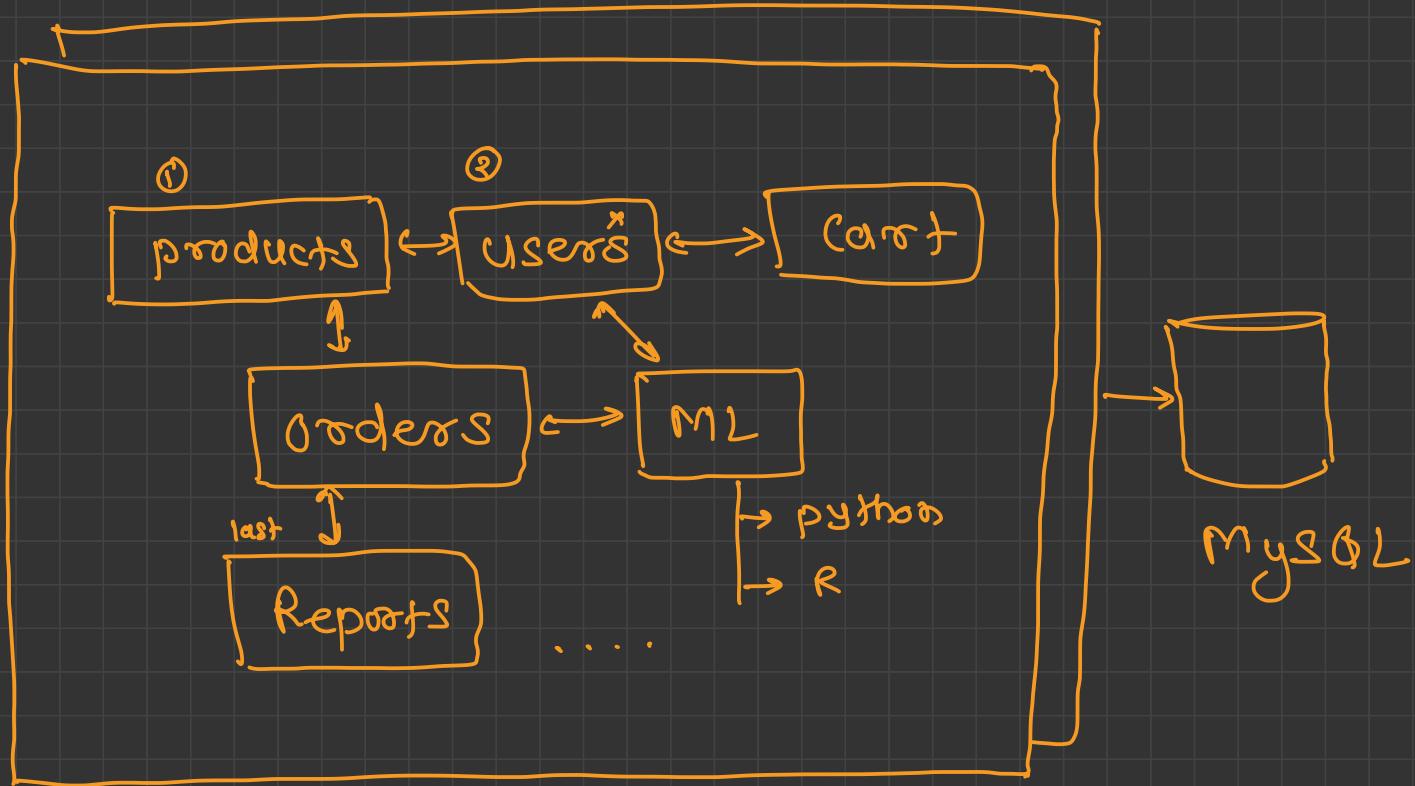
- products
- users
- cart
- orders



language: JS , database : MySQL

backend -server

- server.js
- routes
 - products.js → GET /product , post /product , put /product/:id
DELETE /product/:id
 - user.js → POST /user/signup , POST /user/signin
 - cart.js .
.
 - order.js :
.



SS

— monolithic app
fastest



What is Monolithic Architecture ?

one

package

- A monolithic architecture is a **traditional model** of a software program, which is built as a **unified unit** that is **self-contained** and **independent** from other applications
- The word “monolith” is often attributed to something large and glacial, which isn’t far from the truth of a monolith architecture for software design
→ product
- A monolithic architecture is a singular, large computing network with one **code base** that couples all of the business concerns together
- To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface
- Monoliths can be convenient early on in a project's life for **ease of code management**, cognitive overhead, and deployment
- This allows everything in the monolith to be released at once.
↓
package



Pros

- Easy deployment
 - One executable file or directory makes deployment easier
- Development
 - When an application is built with one code base, it is easier to develop
- Performance
 - In a centralized code base and repository, one API can often perform the same function that numerous APIs perform with microservices
- Simplified testing
 - Since a monolithic application is a single, centralized unit, end-to-end testing can be performed faster than with a distributed application
- Easy debugging
 - With all code located in one place, it's easier to follow a request and find an issue.



Cons

- Slower development speed

- A large, monolithic application makes development more complex and slower

- Scalability

- You can't scale individual components

- Reliability

- If there's an error in any module, it could affect the entire application's availability

- Barrier to technology adoption

- Any changes in the framework or language affects the entire application, making changes often expensive and time-consuming

- Lack of flexibility

- A monolith is constrained by the technologies already used in the monolith

- Deployment

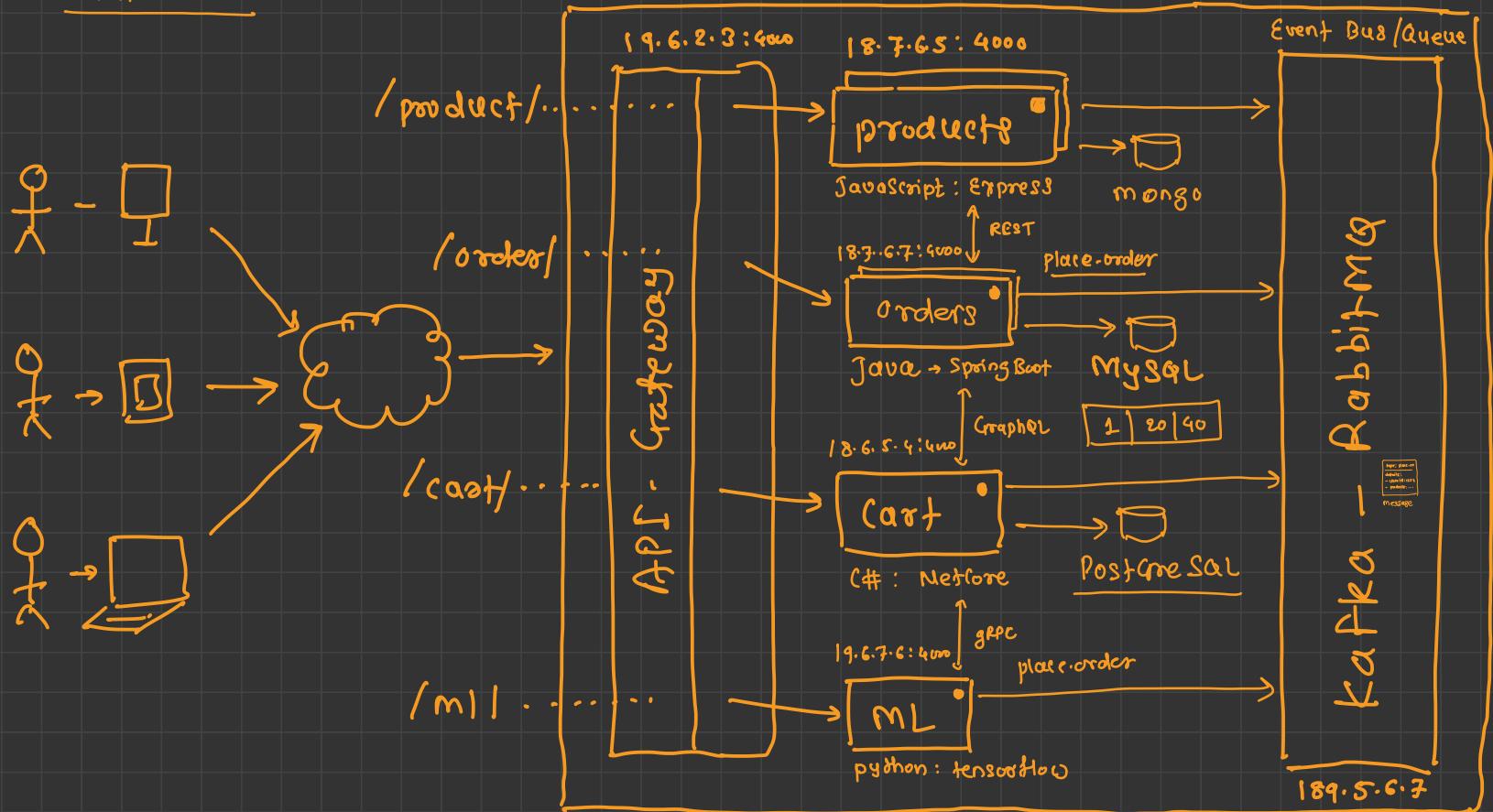
- A small change to a monolithic application requires the redeployment of the entire monolith



Microservices



fault-isolation



back-end - microservices architecture



What is Microservices Architecture ?

- A microservices architecture, also simply known as microservices, is an architectural method that relies on a series of independently deployable services
- These services have their own business logic and database with a specific goal
- Updating, testing, deployment, and scaling occur within each service
- Microservices decouple major business, domain-specific concerns into separate, independent code bases
- Microservices don't reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole
- Adopting microservices often goes hand in hand with DevOps, since they are the basis for continuous delivery practices that allow teams to adapt quickly to user requirements

↗ projects → packages



Pros

- Agility
 - Promote agile ways of working with small teams that deploy frequently
- Flexible scaling
 - If a microservice reaches its load capacity, new instances of that service can rapidly be deployed to the accompanying cluster to help relieve pressure
 - We are now multi-tenant and stateless with customers spread across multiple instances. Now we can support much larger instance sizes
- Continuous deployment
 - We now have frequent and faster release cycles. Before we would push out updates once a week and now we can do so about two to three times a day.
- Highly maintainable and testable
 - Teams can experiment with new features and roll back if something doesn't work
 - This makes it easier to update code and accelerates time-to-market for new features. Plus, it is easy to isolate and fix faults and bugs in individual services
- Independently deployable
 - Since microservices are individual units they allow for fast and easy independent deployment of individual features.
- Technology flexibility
 - Microservice architectures allow teams the freedom to select the tools they desire
- High reliability
 - You can deploy changes for a specific service, without the threat of bringing down the entire application
- Happier teams
 - The Atlassian teams who work with microservices are a lot happier, since they are more autonomous and can build and deploy themselves without waiting weeks for a pull request to be approved



Cons

Development sprawl

- These add more complexity compared to a monolith architecture, as there are more services in more places created by multiple teams
- If development sprawl isn't properly managed, it results in slower development speed and poor operational performance

Exponential infrastructure costs

- Each new microservice can have its own cost for test suite, deployment playbooks, hosting infrastructure, monitoring tools, and more

Added organizational overhead

- Teams need to add another level of communication and collaboration to coordinate updates and interfaces

Debugging challenges

- Each microservice has its own set of logs, which makes debugging more complicated
- Plus, a single business process can run across multiple machines, further complicating debugging

Lack of standardization

- Without a common platform, there can be a proliferation of languages, logging standards, and monitoring

Lack of clear ownership

- As more services are introduced, so are the number of teams running those services
- Over time it becomes difficult to know the available services a team can leverage and who to contact for support

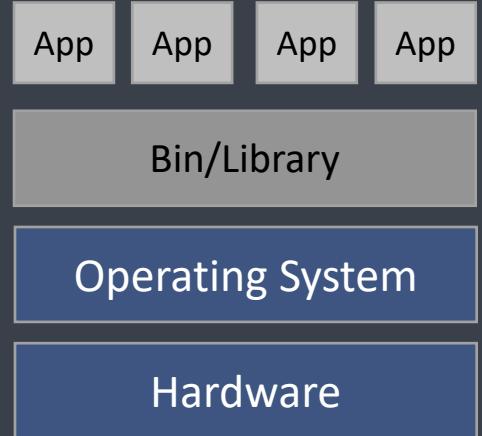


Containerization



Traditional Deployment

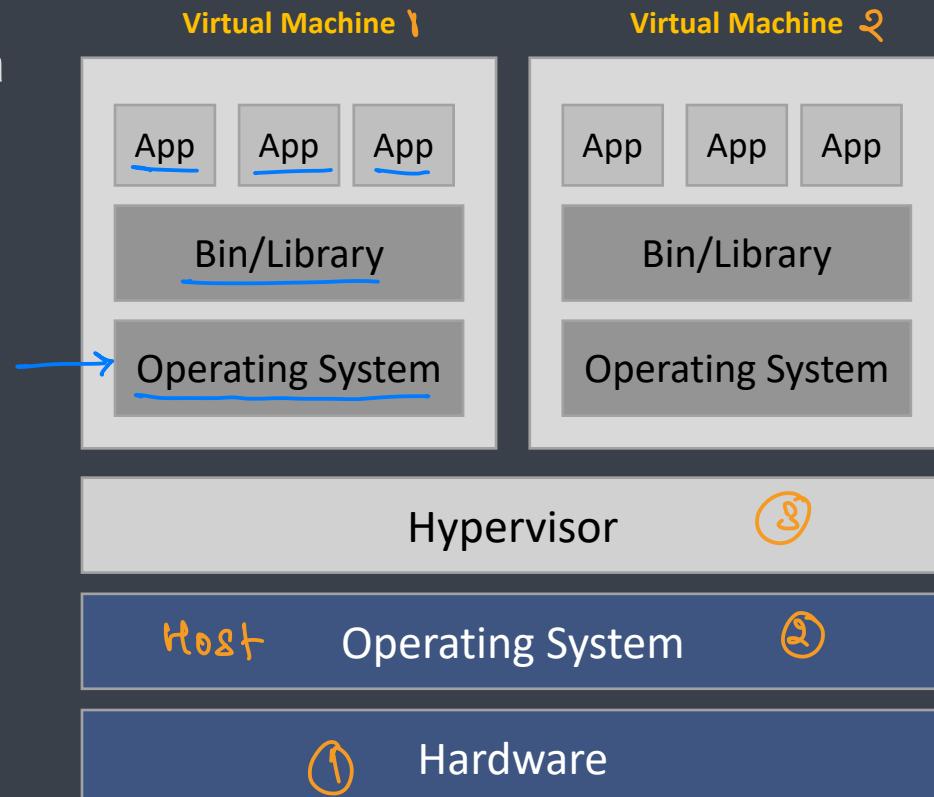
- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application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



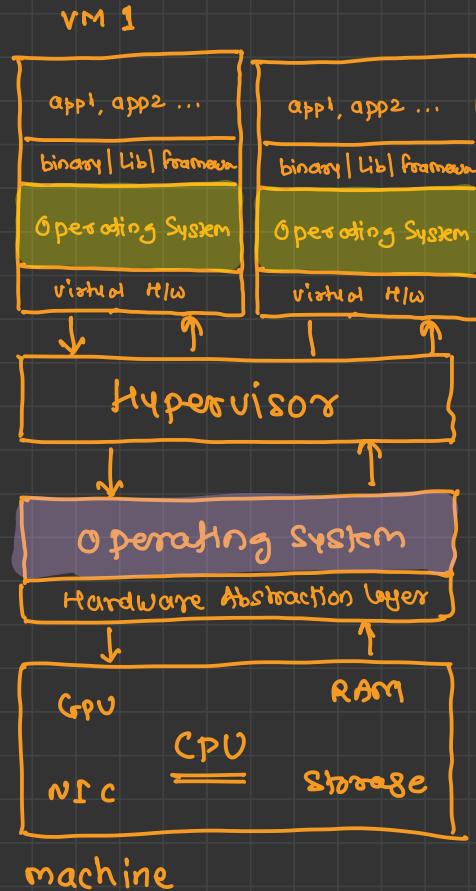


Virtualized Deployment

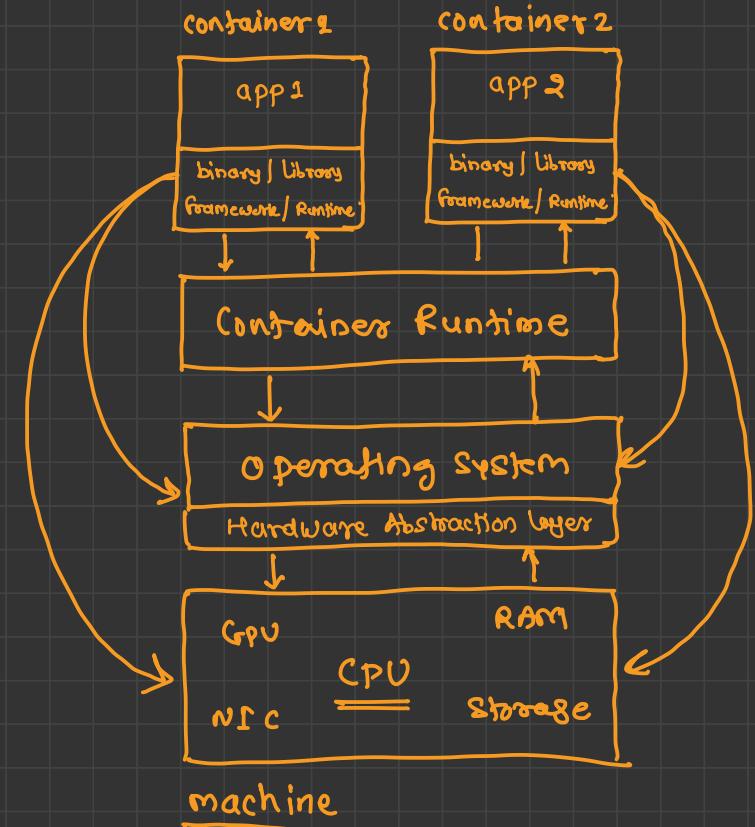
- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application
- Virtualization allows better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



Virtualization



Containerization



What is Containerization



→ minimilistic OS
shared with physical OS / HW

→ libraries / frameworks / runtime
→ resources

- Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure
- More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications
- Containerization allows developers to create and deploy applications faster and more securely
- With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors
- For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system
- Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run
- This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues



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 ~~heavyweight~~ *lite weight*
- 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 vs Virtualization



<u>Virtual Machine</u>	<u>mutable</u>	<u>Container</u>	<u>immutable</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 <i>- fast</i>	
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	



Benefits

▪ 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 both Linux and Windows operating systems
- The container ecosystem has shifted to engines managed by the Open Container Initiative (OCI)
- Software developers can continue using agile or DevOps tools and processes for rapid application development and enhancement

▪ 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
- Not only does this drive higher server efficiencies, it also reduces server and licensing costs while speeding up start-times as there is no operating system to boot

▪ 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
- Development teams can identify and correct any technical issues within one container without any downtime in other containers
- Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers



Benefits

■ 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, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies, reducing server and licensing costs

■ Ease of management

- 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. Kubernetes, perhaps the most popular container orchestration system available, is an open source technology (originally open-sourced by Google, based on their internal project called Borg) that automates Linux container functions originally
- Kubernetes works with many container engines, such as Docker, but it also works with any container system that conforms to the Open Container Initiative (OCI) standards for container image formats and runtimes

■ 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



Docker





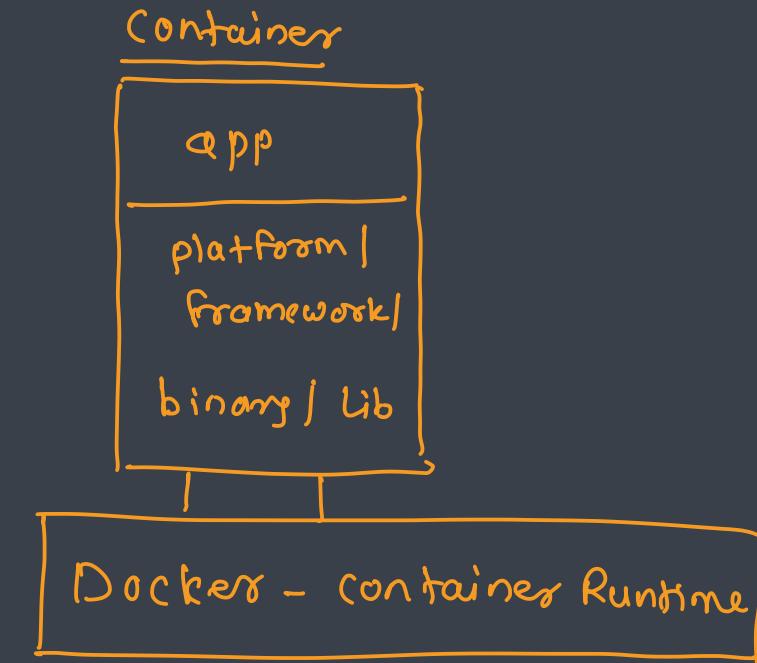
What is Docker ?

- Docker is an open source platform that enables developers to build, deploy, run, update and manage containers—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment

- Why docker?

- It is an easy way to create application deployable packages
- Developer can create ready-to-run containerized applications
- It provides consistent computing environment
- It works equally well in on-prem as well as cloud environments
- It is light weight compared to VM

☞ containers





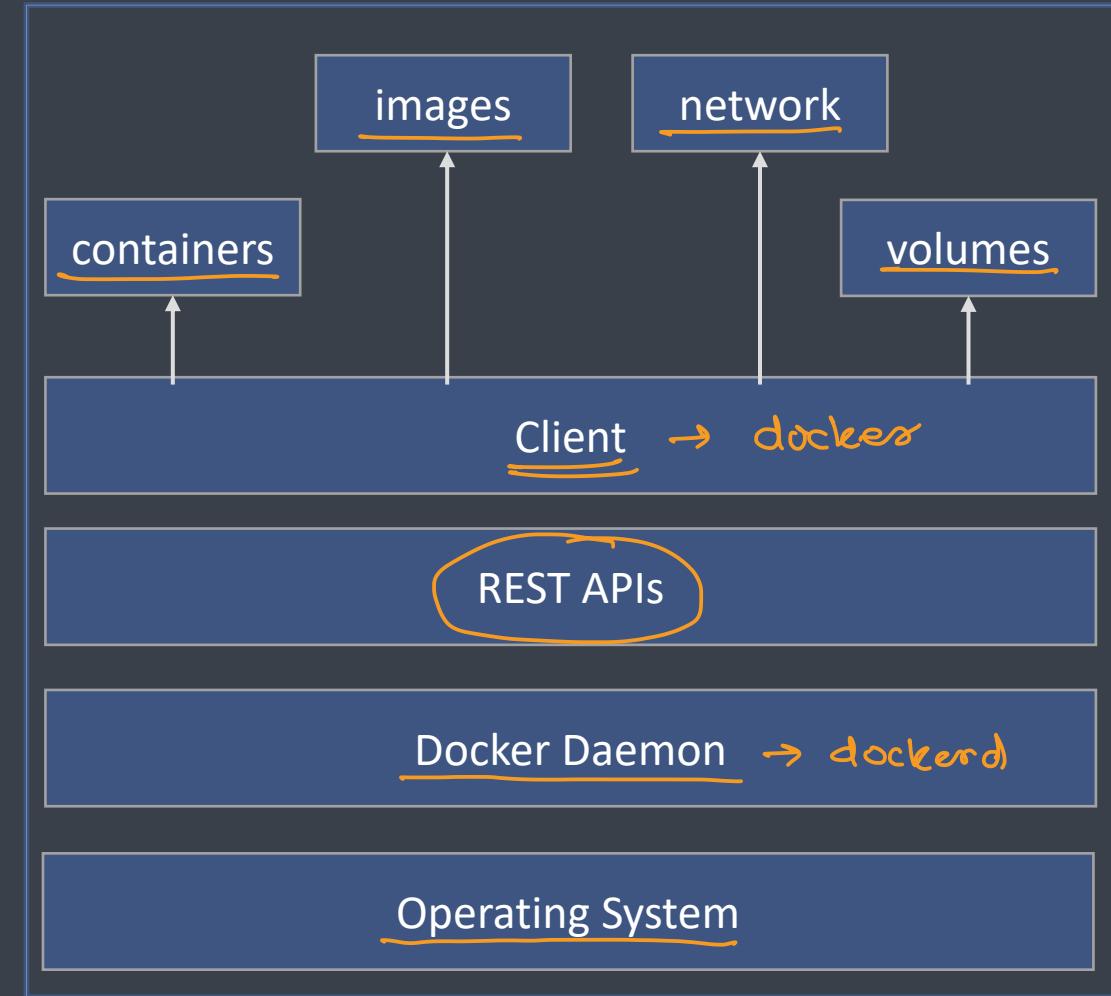
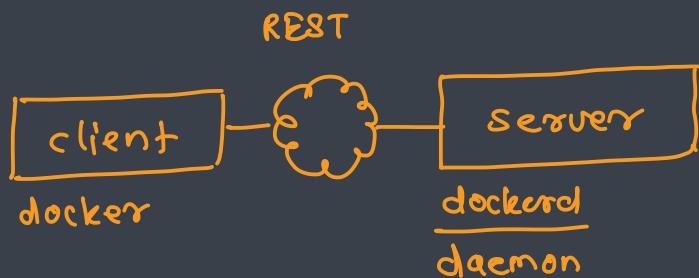
Little history about Docker

- Docker Inc, started by Solomon Hykes, is behind the docker tool
- Docker Inc started as Paas provider called as dotCloud
- In 2013, the dotCloud became Docker Inc
- Docker Inc was using LinuX Containers (LXC) before version 0.9
- After 0.9 (2014), Docker replaced LXC with its own library libcontainer which is developed in Go programming language
- Its not the only solution for containerization
 - “FreeBSD Jails”, launched in 2000
 - LXD is next generation system container manager build on top of LXC and REST APIs
 - Google has its own open source container technology Imctfy (Let Me Contain That For You)
 - Rkt is another option for running containers



Docker Architecture

- Docker daemon (`dockerd`)
 - Continuous running process
 - Manages the containers
- REST APIs
 - Used to communicate with docker daemon
- Client (`docker`)
 - Provides command line interface
 - Used to perform all the tasks





libcontainer

- Docker has replaced LXC by libcontainer, which is used to manage the containers
- Libcontainer uses
 - Namespaces
 - Creates isolated workspace which limits what container can see
 - Provides a layer of isolation to the container
 - Each container runs in a separate namespace
 - Processes running in a namespace can interact with other processes or use resources which are the part of the same namespace
 - E.g. process ID, network, IPC, Filesystem
 - Control Groups (cgroups)
 - Used to share the available resources to the containers
 - It optionally enforces limits and constraints on resource usage
 - It limits how much a container can use
 - E.g. CPU, Disk space, memory



libcontainer

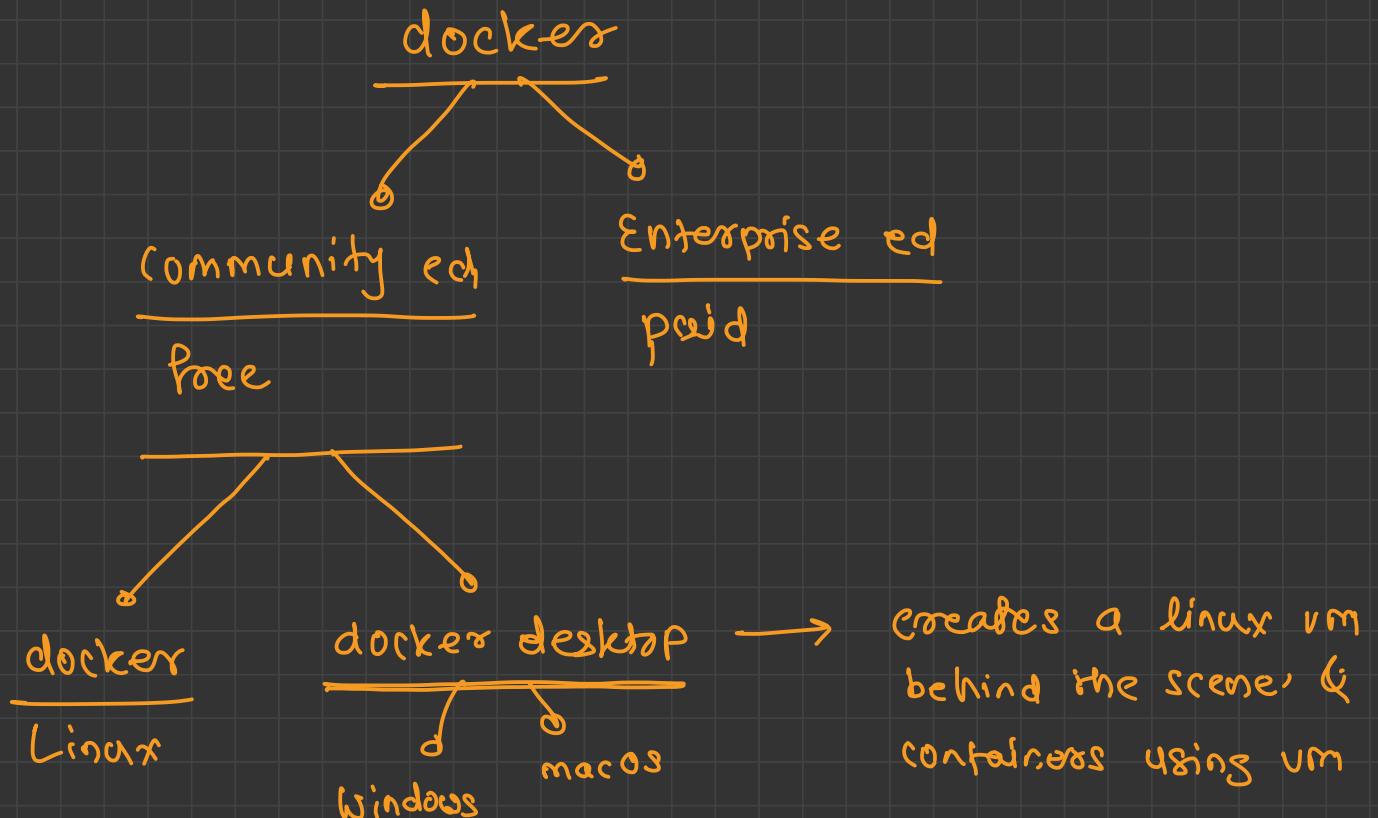
▪ Union File System (UnionFS)

- It uses layers
- It is a lightweight and very fast FS
- Docker uses different variants of UnionFS
 - Aufs (advanced multi-layered unification filesystem)
 - Btrfs (B-Tree FS)
 - VFS (Virtual FS)
 - Devicemapper



Docker Objects

- Images: read only template with instructions for creating docker containers
- Container: running instance of a docker image
- Network: network interface used to connect the containers to each other or external networks
- Volumes: used to persist the data generated by and used by the containers
- Registry: private or public collection of docker images
- Service: used to deploy application in a docker multi node cluster





What is Docker image ? → read only → immutable

- A Docker image is a file used to execute code in a Docker container
- Docker images act as a set of instructions to build a Docker container, like a template
 - ↳ disk image [like AMI]
- Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments
- A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container
- Docker images have multiple layers, each one originates from the previous layer but is different from it
- The layers speed up Docker builds while increasing reusability and decreasing disk use
- Image layers are also read-only files
- Once a container is created, a writable layer is added on top of the unchangeable images, allowing a user to make changes



Docker Container



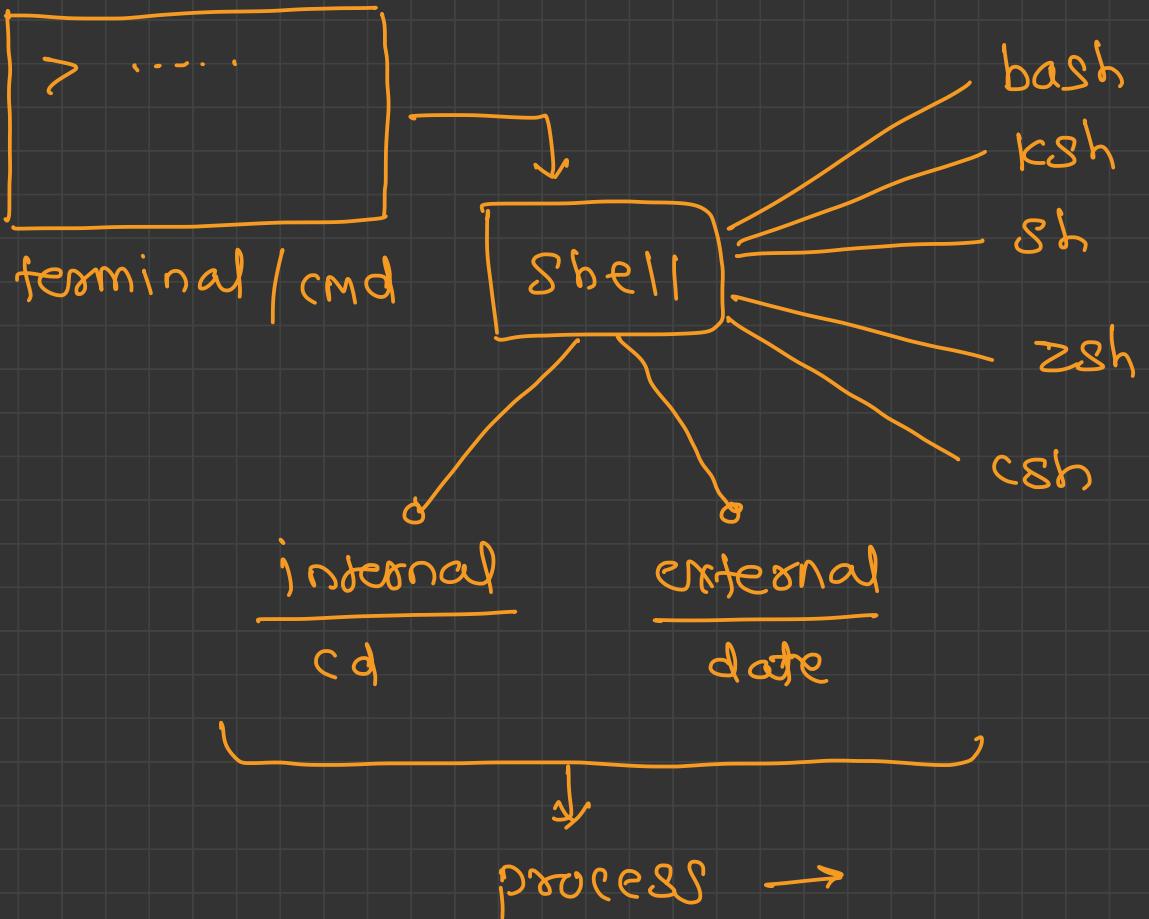
Docker Container

- It is a running aspect of docker image
- Contains one or more running processes [app]
- It is a self-contained environment → micro-service
- It wraps up an application into its own isolated box (application running inside a container has no knowledge of any other applications or processes that exist outside the container)
- A container can not modify the image from which it is created
- It consists of
 - Your application code
 - Dependencies
 - Networking
 - Volumes → docker home
- Containers are stored under /var/lib/docker
- This directory contains images, containers, network volumes etc



Basic Operations

- Creating container
- Starting container
- Running container
- Listing running containers
- Listing all containers
- Getting information of a container
- Stopping container
- Deleting container

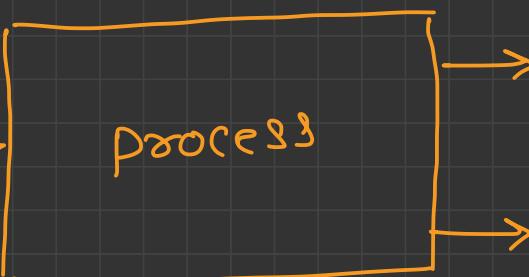


Attached

Keyboard



stdin
standard input

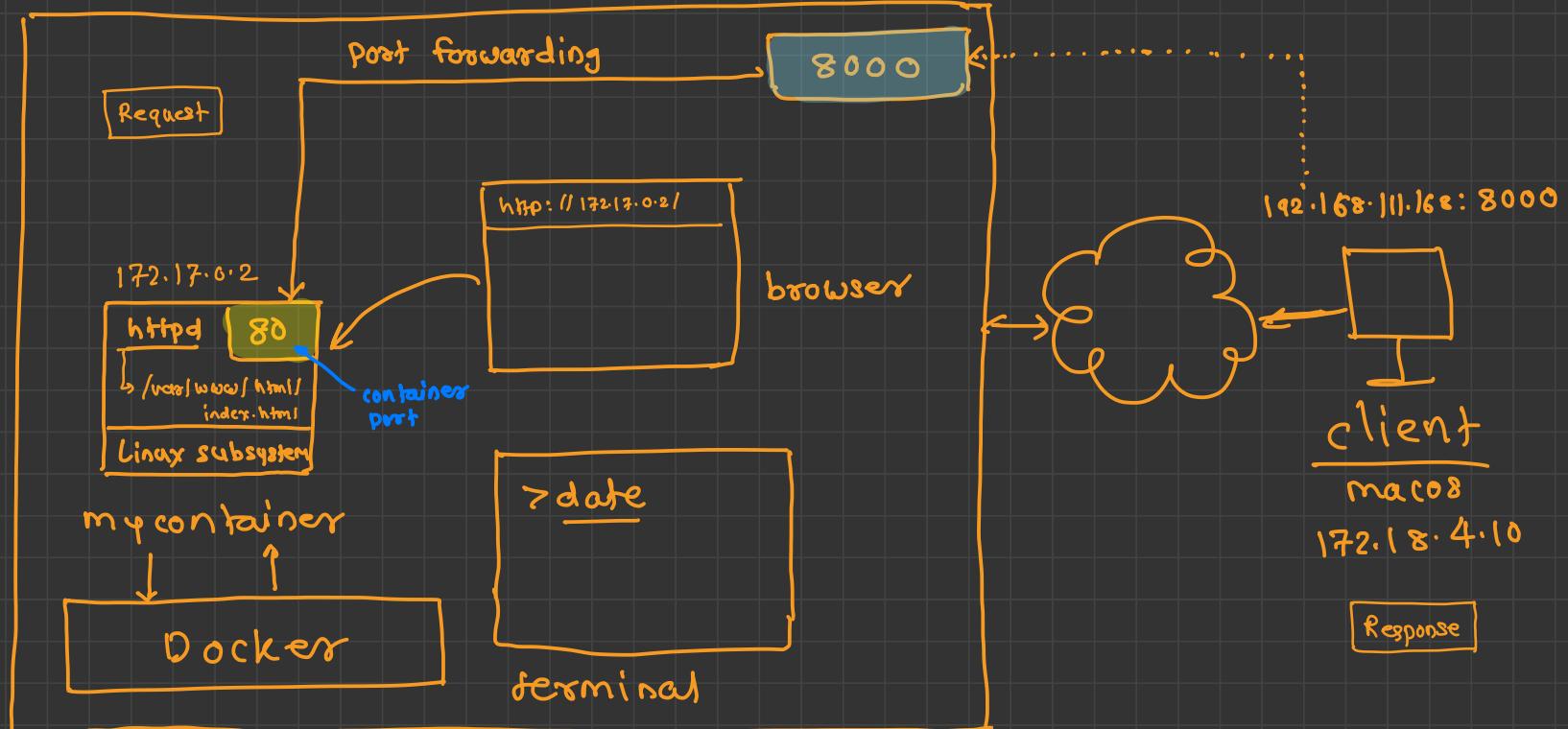


Standard output →
Stdout

Standard error →
Stderr



Ubuntu



machine

date & time
of local machine



Attaching a container

- There are two ways to attach to a container
- Attach
 - Used to attach the container
 - Uses only one input and output stream
 - Task
 - Attach to a running container
- Exec
 - Mainly it is used for running a command inside a container
 - Task
 - Execute a command inside container



Hostname and name of container

- To check the host name
 - Go inside the container
 - Check the hostname by using a command hostname
- Docker uses the first 12 characters of container id as hostname
- Docker automatically generates a name at random for each container



Publishing port on container

- Publishing a port is required to give an external access to your application
- Port can be published only at the time of creating a container
- You can not update the port configuration on running container
- Task
 - Run a httpd container with port 8080 published, to access apache externally

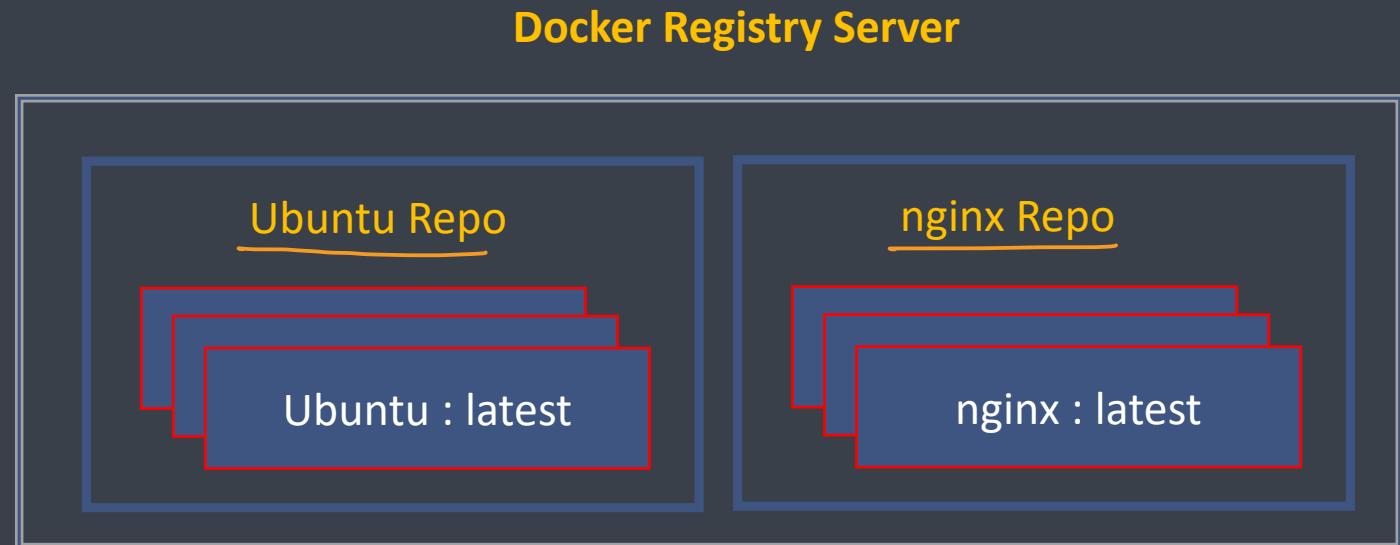


Docker Images (Advanced)



Docker Image

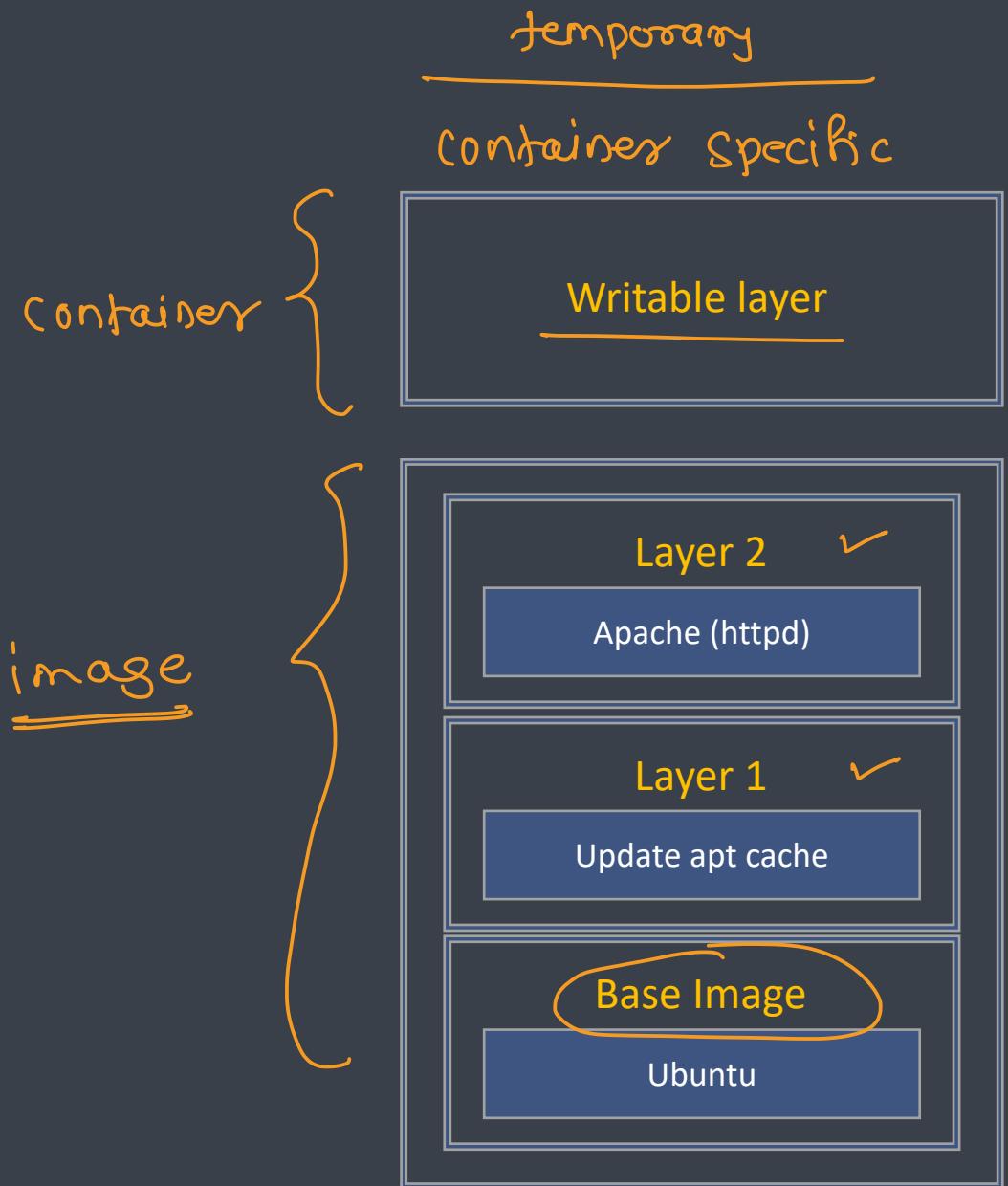
- Read-only instructions to run the containers
- It is made up of different layers
- Repositories hold images
- Docker registry stores repositories
- To create a custom image
 - Commit the running container ✗
 - Use a Dockerfile ←
- Task
 - Create a container
 - Create a directory and a file within it
 - Commit the container to create a new image





Layered File System

- Docker images are made of layered FS
- Docker uses UnionFS for implementing the layered docker images
- Any update on the image adds a new layer
- All changes made to the running container are written inside a writable layer





Dockerfile

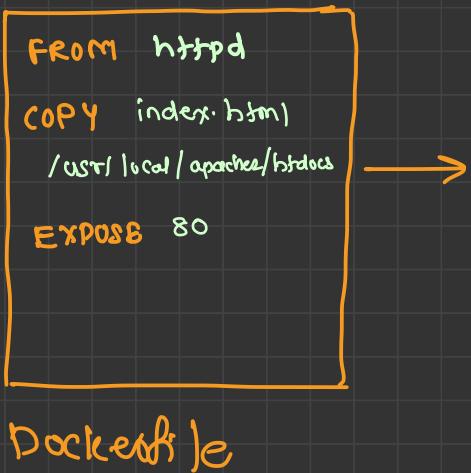
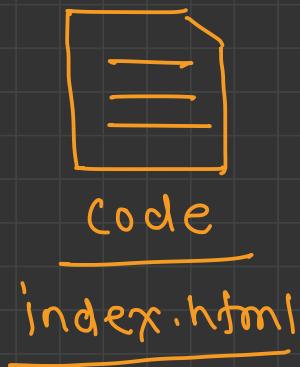
- The Dockerfile contains a series of **instructions paired with arguments**
- Each instruction should be in upper-case and be followed by an argument
- Instructions are processed from top to bottom
- Each instruction adds a new layer to the image and then commits the image
- Upon running, changes made by an instruction make it to the container



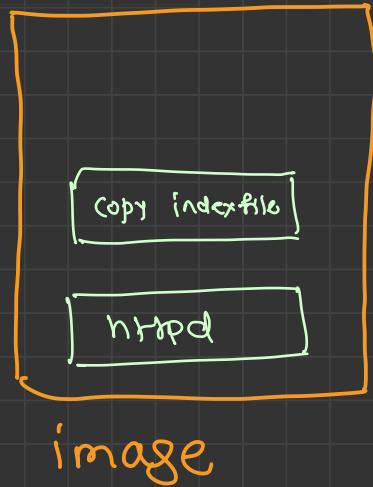
Dockerfile instructions

- FROM
- ENV
- RUN
- CMD
- EXPOSE
- WORKDIR
- ADD
- COPY
- LABEL
- MAINTAINER
- ENTRYPOINT

`docker image build -t <image name>`

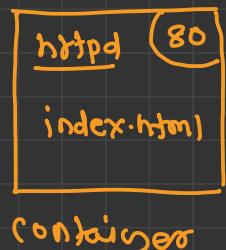


Dockerfile



image

→ docker container run
—name



container

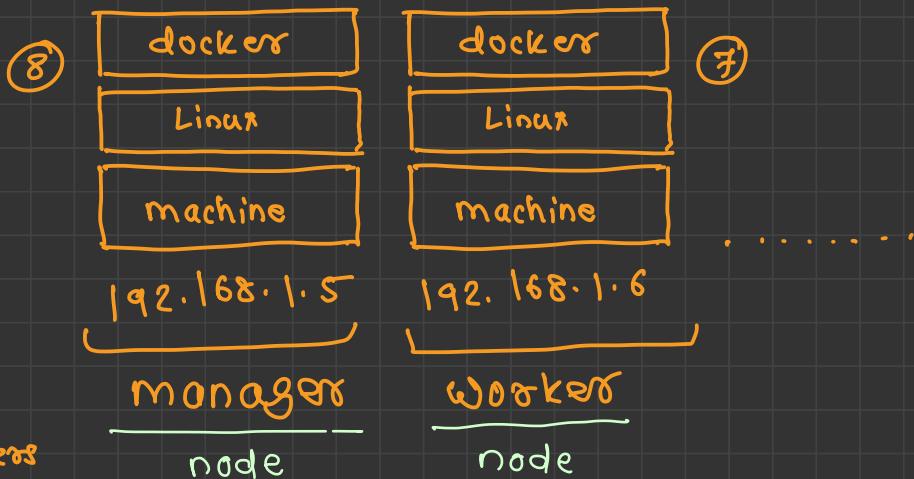
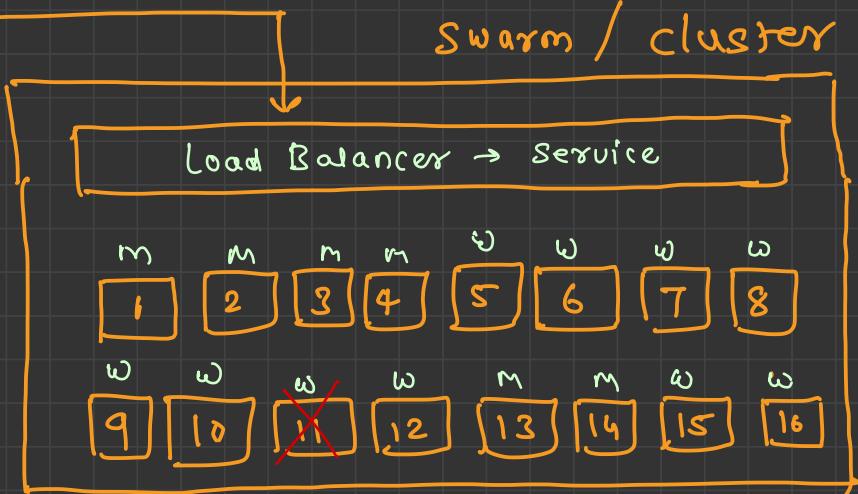
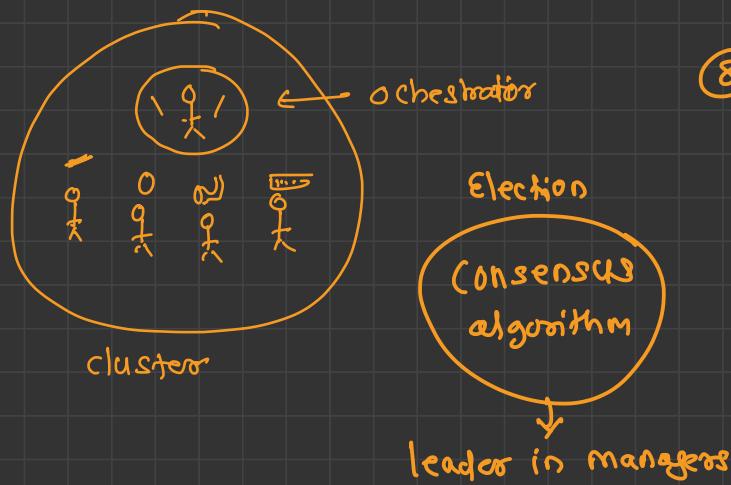
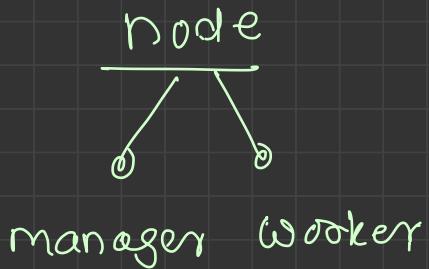
* every docker image requires a base image



Orchestration



normal: 15, peak: 20
[desired count]





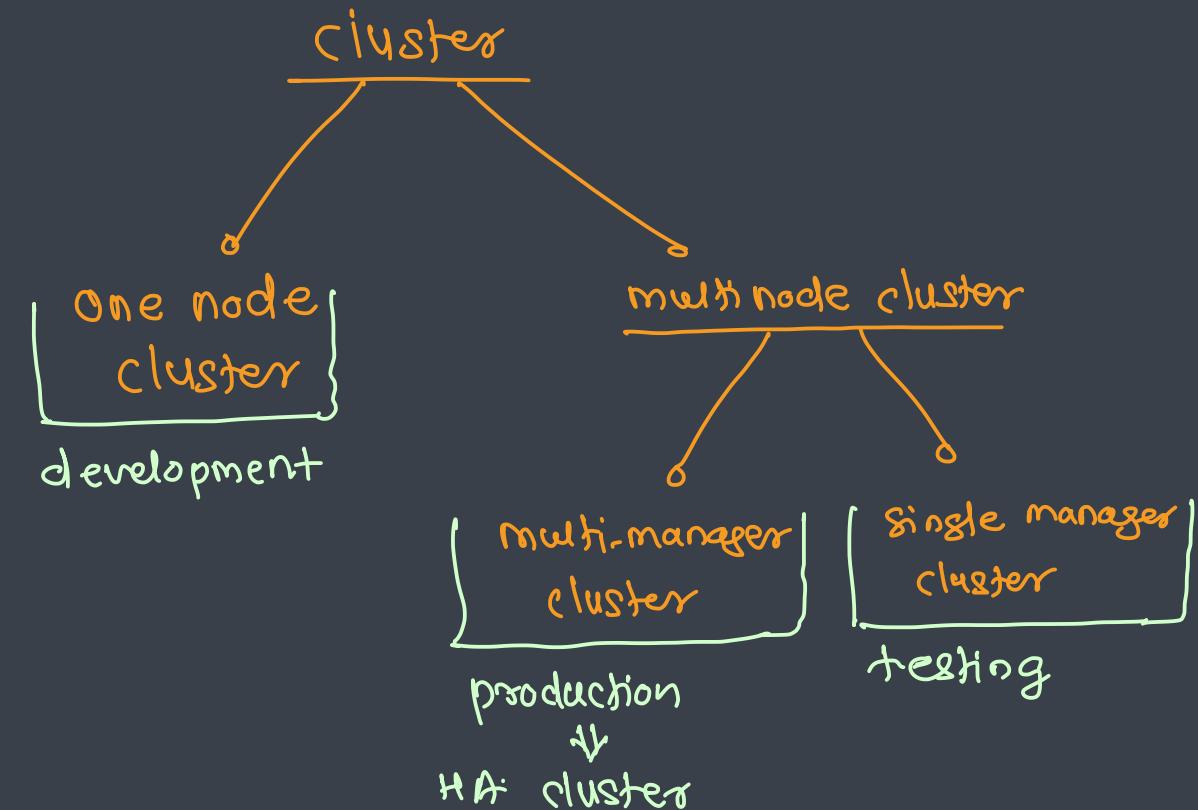
Container Orchestration

- 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 ✓ desired count < 10k
 - * * * ▪ Kubernetes - desired count > 10k
 - Mesos
 - Marathon



Docker Swarm

- 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
- It is secure by default
- It is built using **Swarmkit** (framework / library)

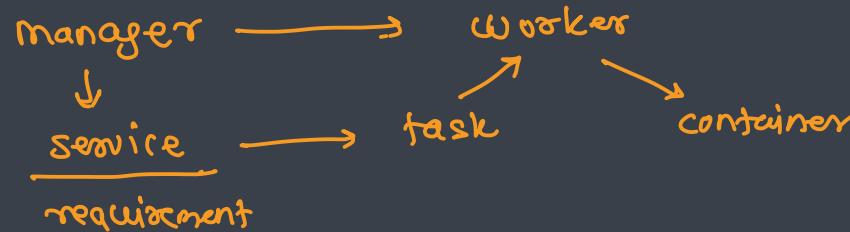
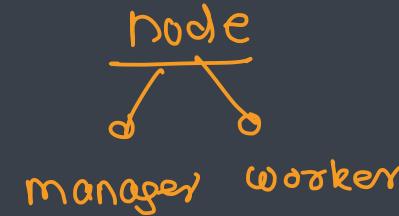




What is a swarm?

no de8

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





Features

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



Nodes

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



Services and tasks

■ Service

- A service is the definition of the tasks to execute on the manager or worker nodes
- It is the central structure of the swarm system and the primary root of user interaction with the swarm
- When you create a service, you specify which container image to use and which commands to execute inside running containers

■ Task

- A task carries a Docker container and the commands to run inside the container
- It is the atomic scheduling unit of swarm
- Manager nodes assign tasks to worker nodes according to the number of replicas set in the service scale
- Once a task is assigned to a node, it cannot move to another node
- It can only run on the assigned node or fail



Create a Swarm and add workers to swarm

- In our swarm we are going to use 3 nodes (one manager and two workers)
- Every node must have Docker Engine 1.12 or newer installed
- Following ports are used in node communication
 - TCP port 2377 is used for cluster management communication
 - TCP and UDP port 7946 is used for node communication
 - UDP port 4789 is used for overlay network traffic



```
docker swarm init --advertise-addr <ip>
```



```
docker swarm join --token <token>
```



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>`



Overlay Network

- It is a computer network built on top of another network
- Sits on top of the host-specific networks and allows container, connected to it, to communicate securely
- When you initialize a swarm or join a host to swarm, two networks are created
 - An overlay network called as ingress network
 - A bridge network called as docker_gwbridge
- Ingress network facilitates load balancing among services nodes
- Docker_gwbridge is a bridge network that connect overlay networks to individual docker daemon's physical network



Service

- Definition of tasks to execute on Manager or Worker nodes
- Declarative Model for Services
- Scaling
- Desired state reconciliation
- Service discovery
- Rolling updates
- Load balancing
- Internal DNS component



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>`

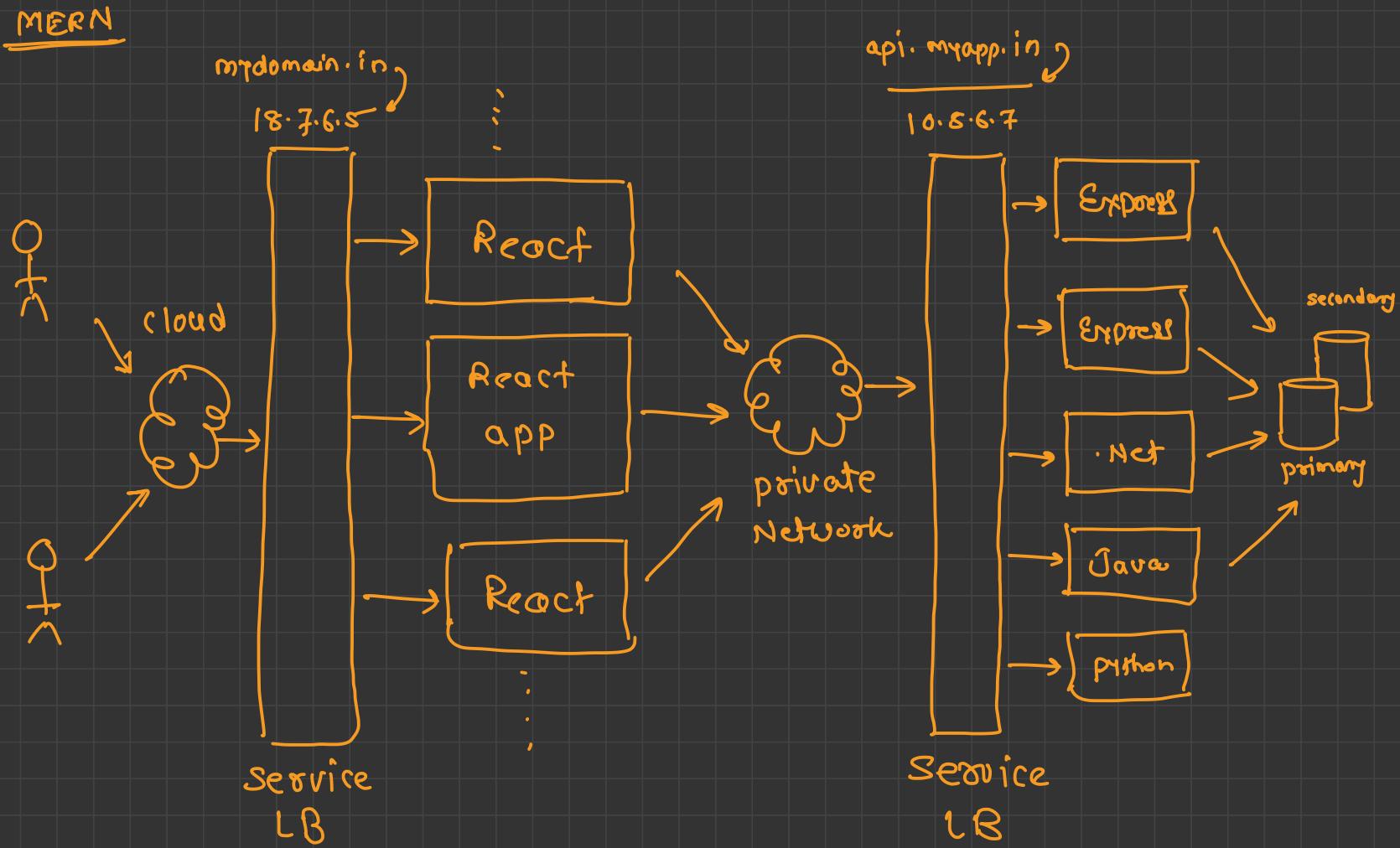
- **Update service**

> `docker service update --image <image> <service>`

- **Delete service**

> `docker service rm <service>`

MERN



On-prem

- customer owned

Cloud

\$\$\$

Self-managed

- managed by customer
- EC2

managed service

- managed by provider

ECS

Docker Swarm



Kubernetes





What is Kubernetes ?

- Portable, extensible, open-source platform for managing containerized workloads and services
- Facilitates both declarative configuration and automation YAML
- 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

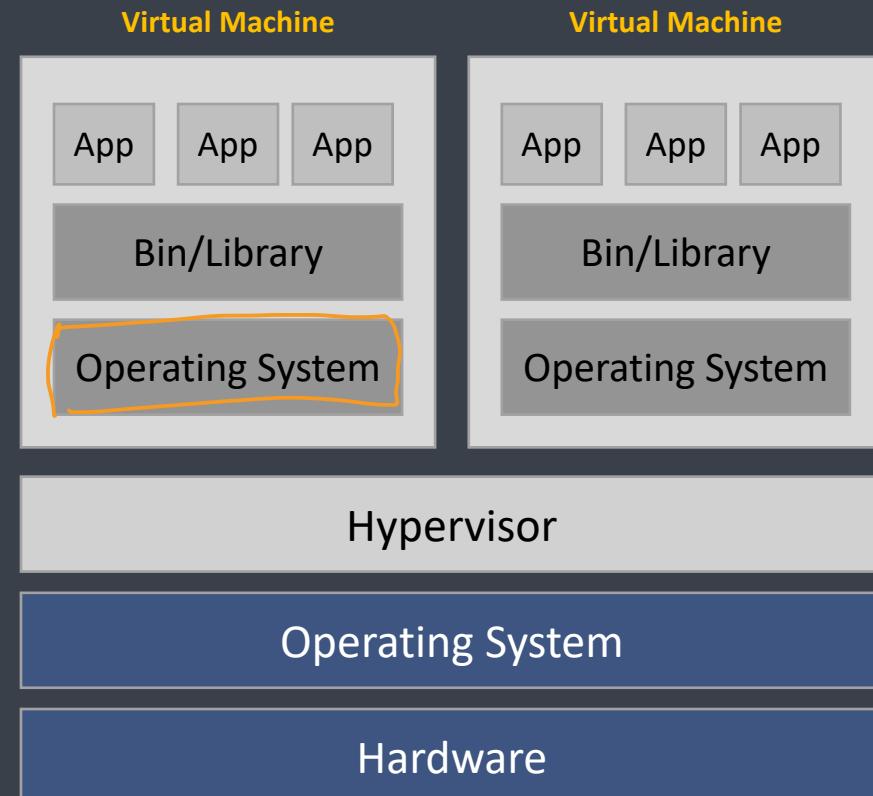
- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application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

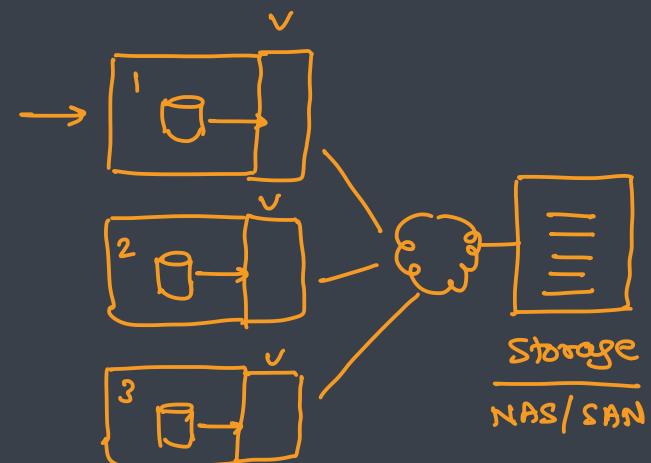
- Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more

Automated rollouts and rollbacks

- You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate

Automatic bin packing

- You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks
- You tell Kubernetes how much CPU and memory (RAM) each container needs
- Kubernetes can fit containers onto your nodes to make the best use of your resources





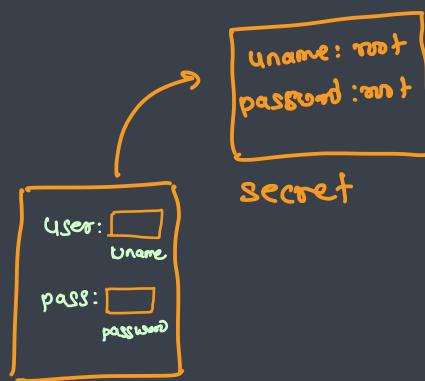
What Kubernetes provide?

▪ Self-healing

- Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve

▪ Secret and configuration management

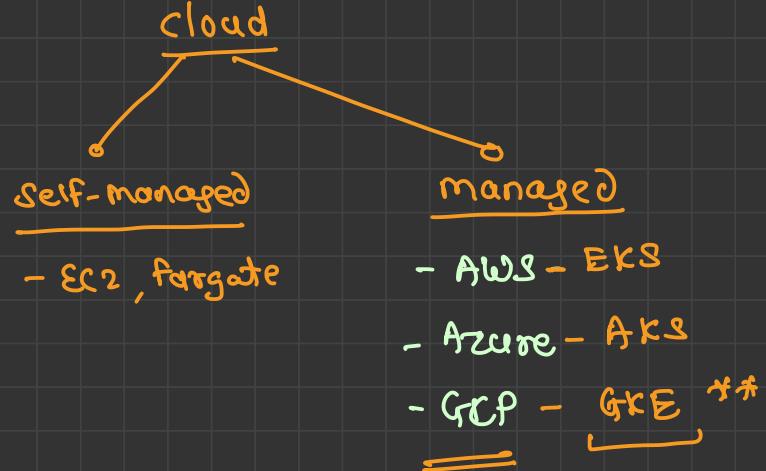
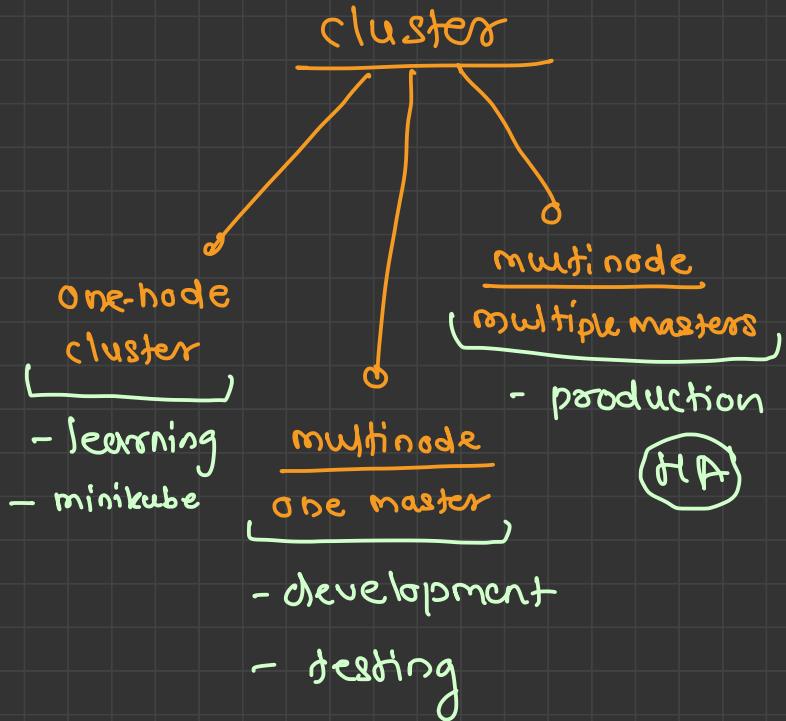
- Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys
- You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration





What Kubernetes is not

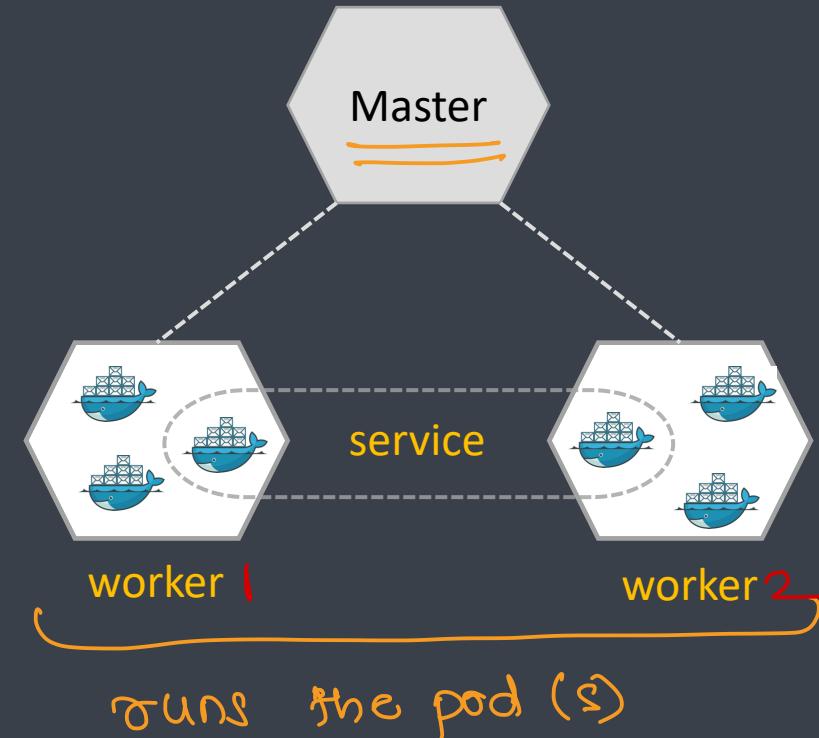
- Does not limit the types of applications supported → build tools → ant, maven, gradle
- Does not deploy source code and does not build your application
- Does not provide application-level services as built-in services
- Does not dictate logging, monitoring, or alerting solutions → logging: prometheus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

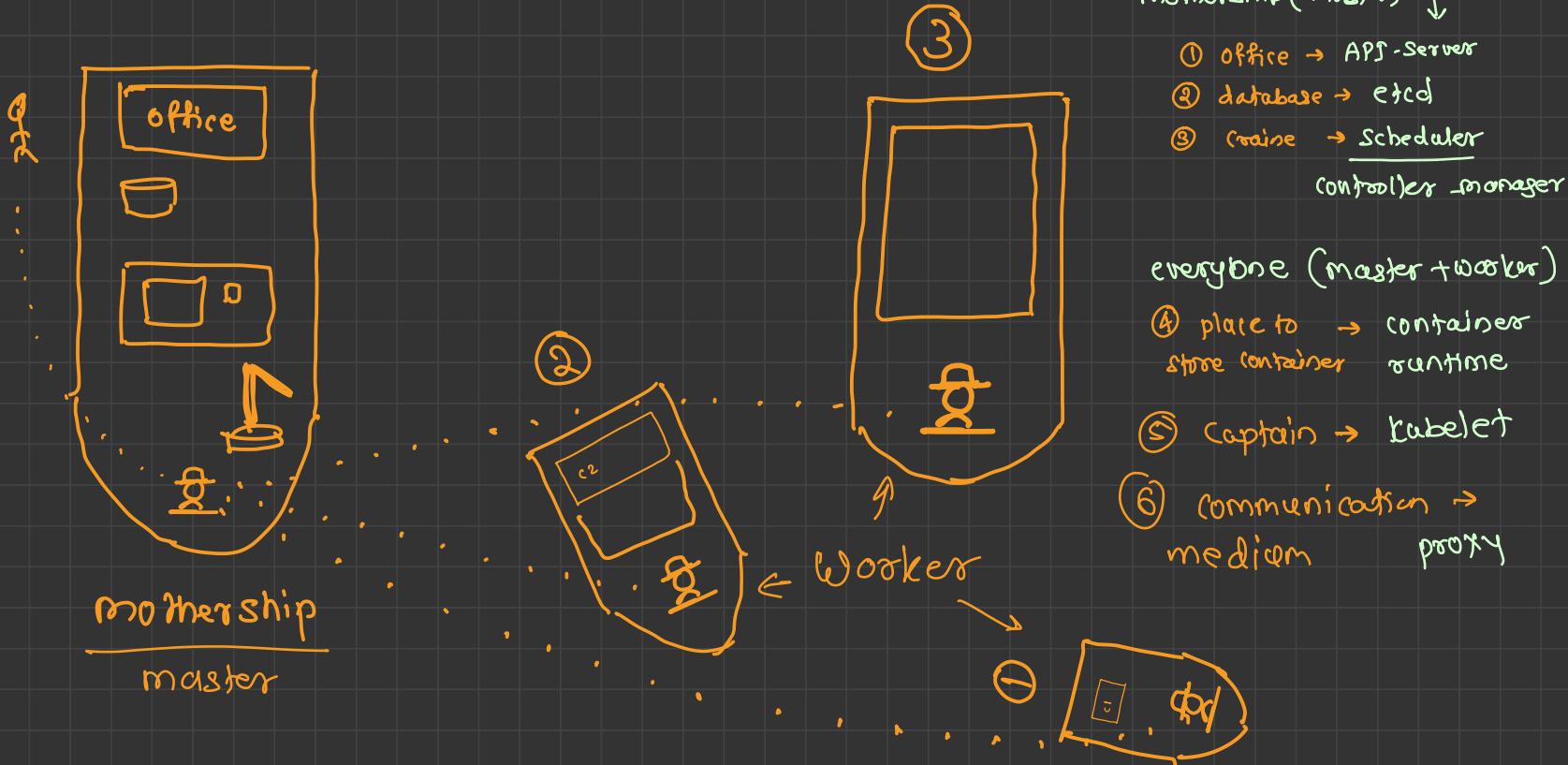




Kubernetes Cluster

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





mothership (master) ↓

① office → API-Server

② database → etcd

③ cronie → Scheduler

controller manager

everyone (master + worker)

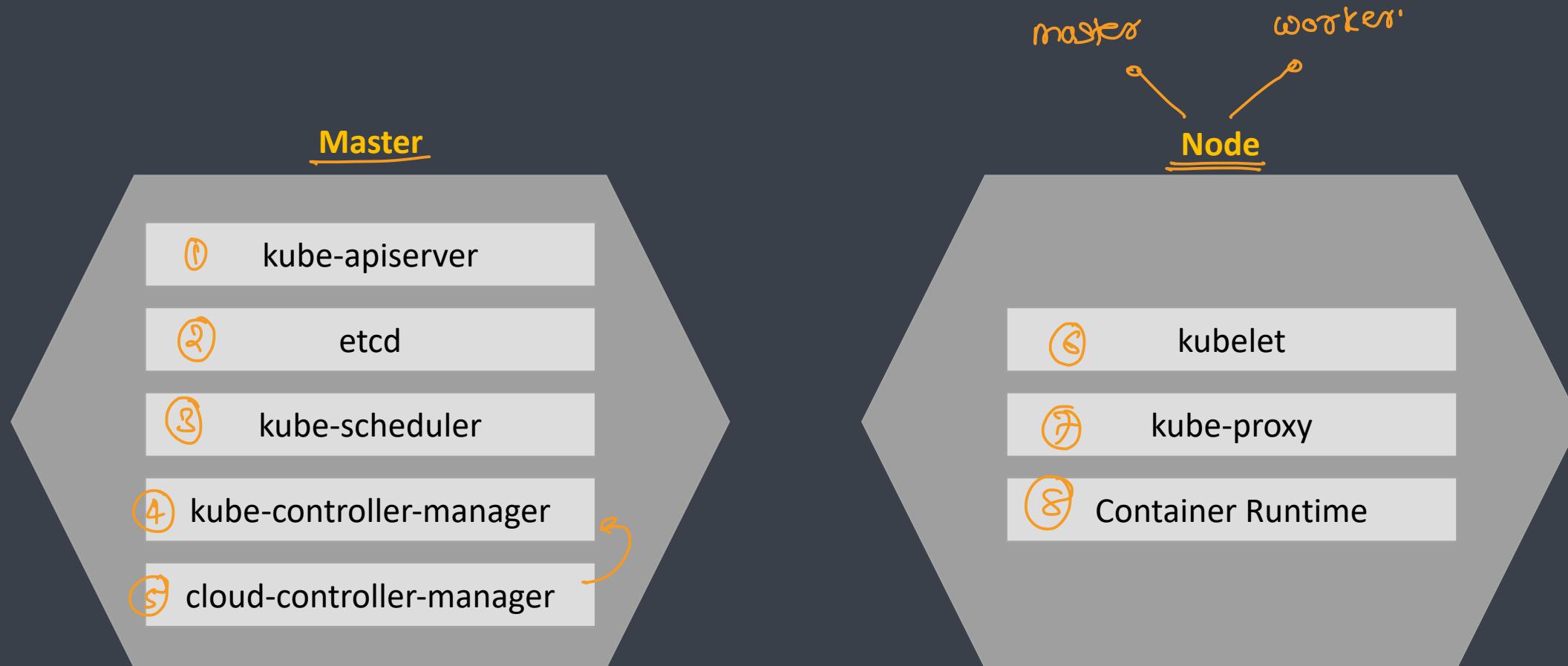
④ place to store container runtime

⑤ captain → kubelet

⑥ communication → medium proxy



Kubernetes Components





Master Components

- Master components make global decisions about the *cluster* and they detect and respond to **cluster events**
 - Master components can be run on any machine in the cluster
 - **kube-apiserver**
 - The API server is a component that exposes the Kubernetes API → **REST**
 - 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 → **database** → **cluster state**
 - **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

Pod stopped
Node going down
Node resources are exhausted



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

monolithic

▪ Node Controller: Responsible for noticing and responding when nodes go down.

- ReplicaSet → ▪ Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system
- ↳ desired count
- 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

security

■ 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

⇒ master + worker

- 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

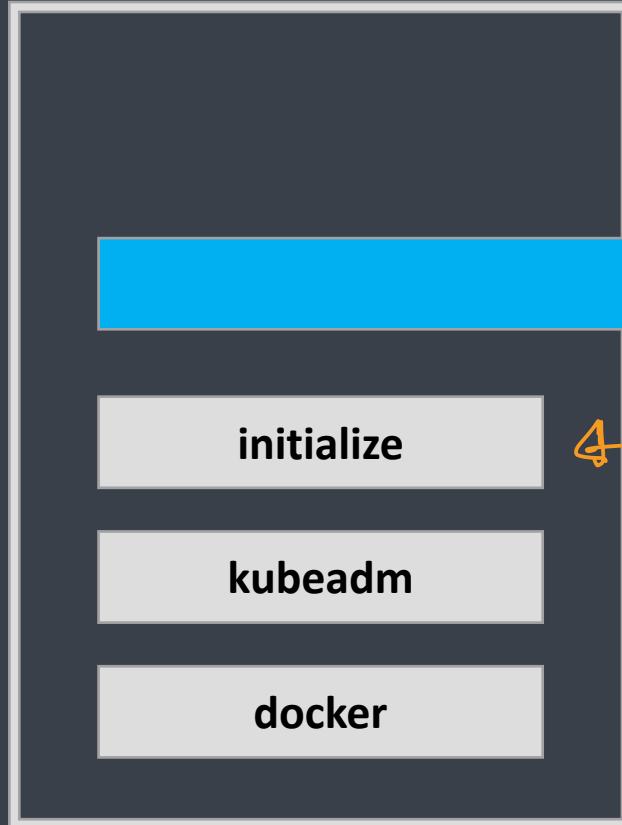
5

4

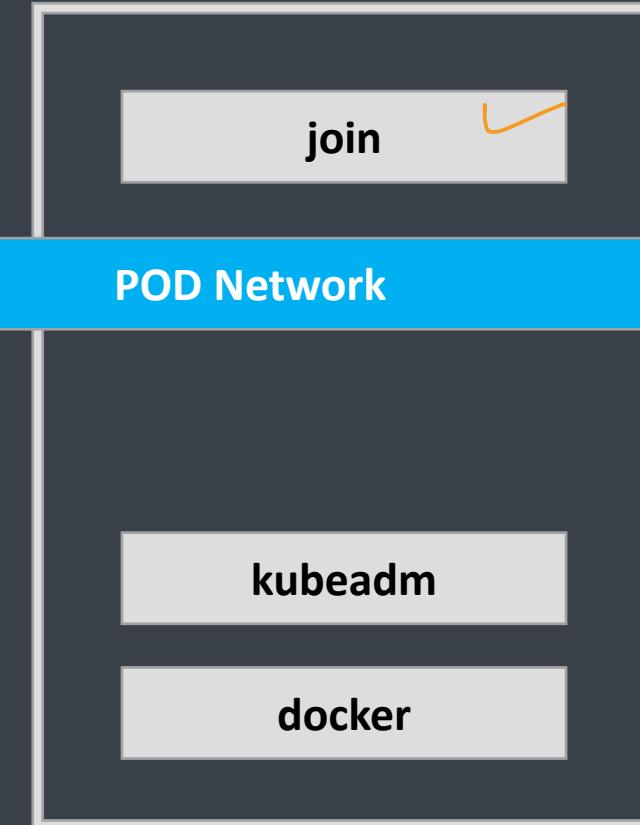
3

2

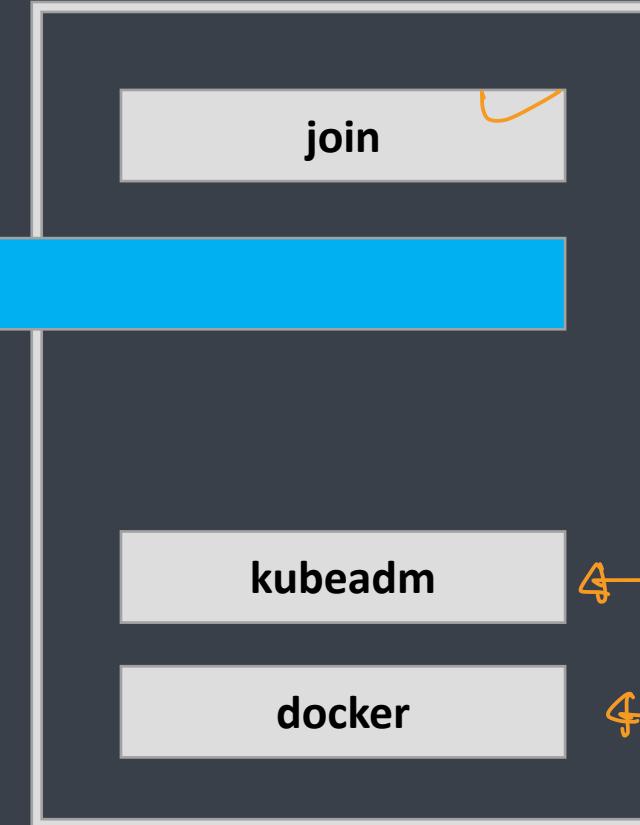
1



Master



Worker Node 1



Worker Node 2



Kubernetes Objects

- The basic Kubernetes objects include

- *** ✓■ Pod → to create/run containers
- ** ✓■ Service → L.B.
- ✓■ Volume → storage
- * ✓■ Namespace → group of resources

- Kubernetes also contains higher-level abstractions build upon the basic objects

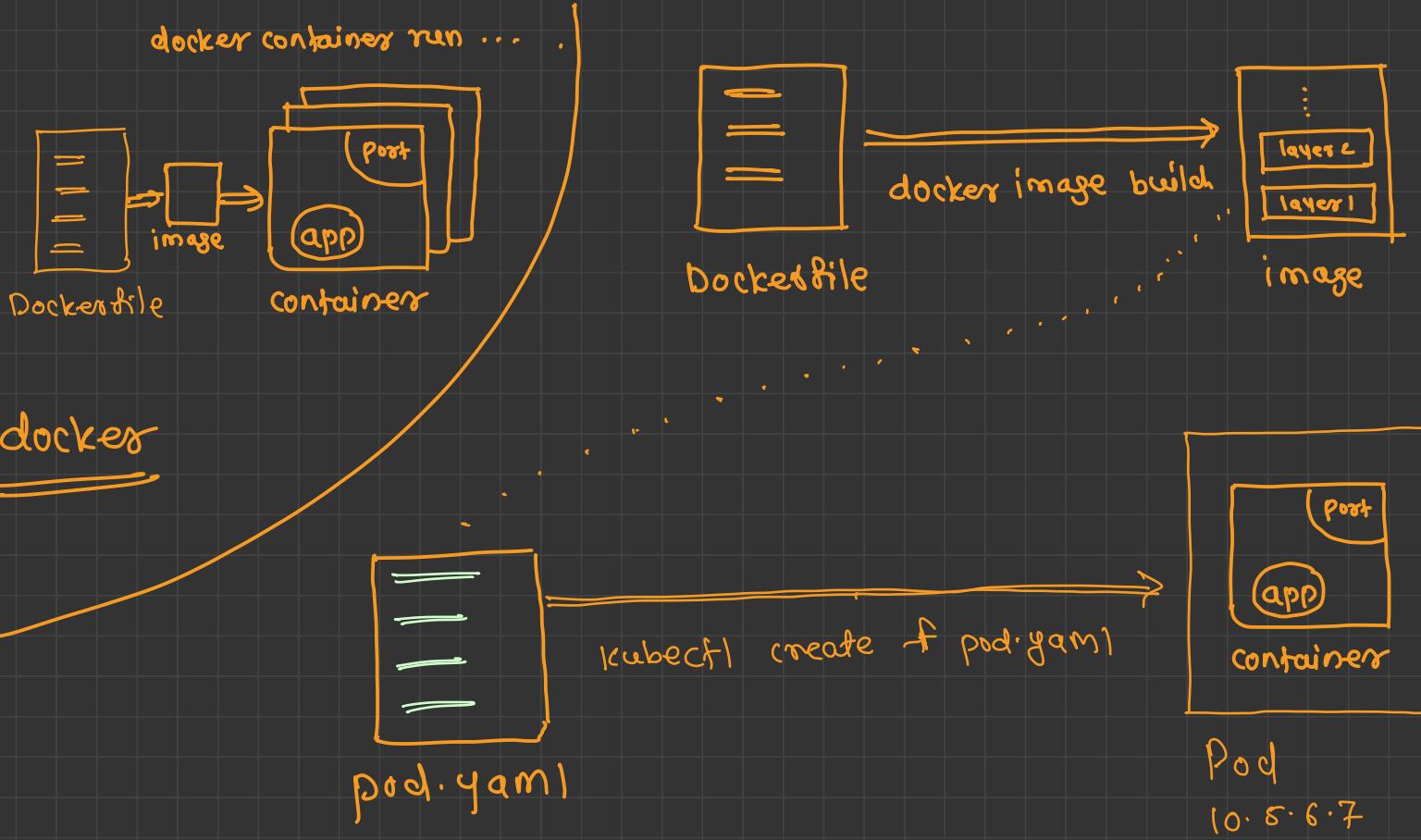
- ✓■ Deployment
 - ✓■ DaemonSet
 - ✓■ StatefulSet
 - ✓■ ReplicaSet
 - ✓■ Job
- 
- Pod



Namespace → group of objects [pods / services ..]

- 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

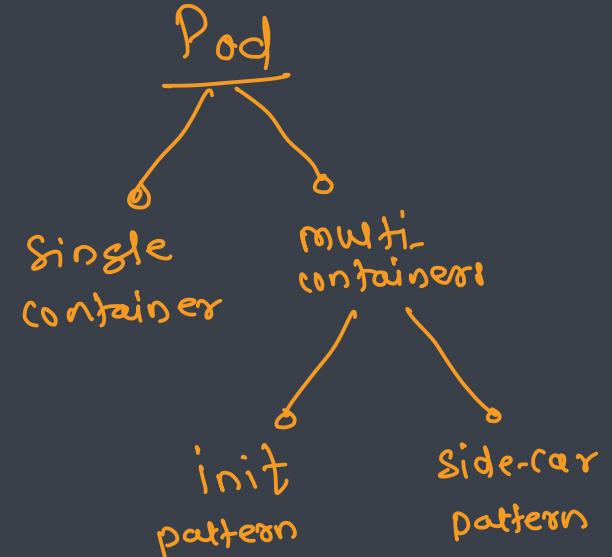






Pod

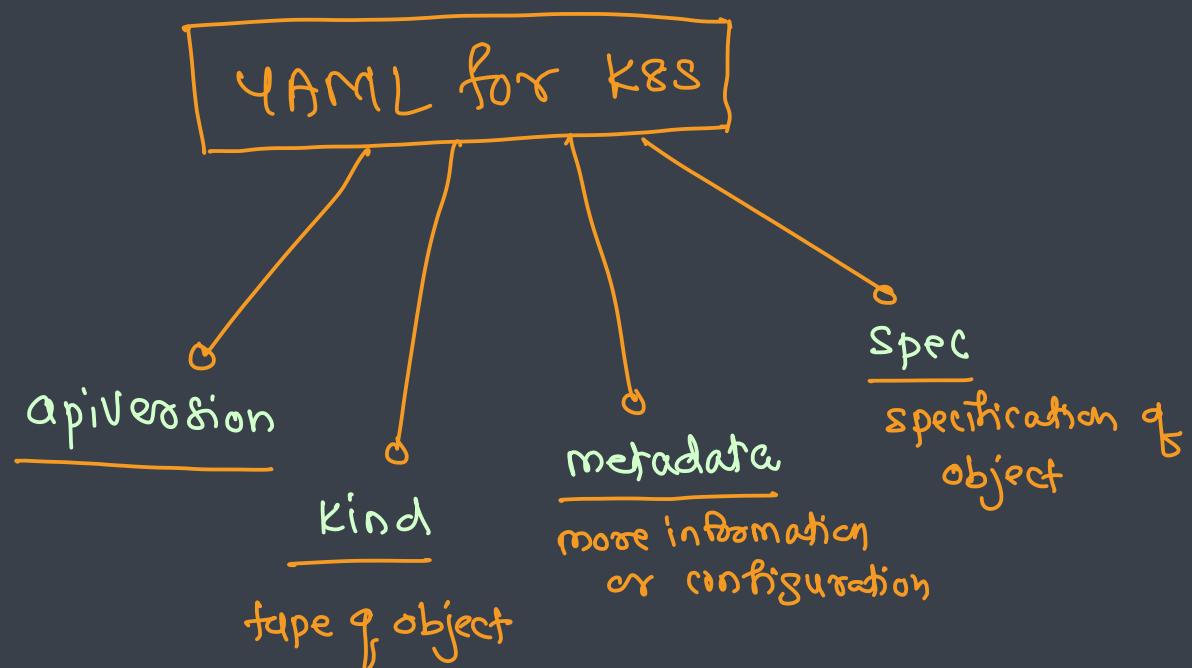
- A Pod is the basic execution unit of a Kubernetes application
- The smallest and simplest unit in the Kubernetes object model that you create or deploy
- A Pod represents processes running on your Cluster
- Pod represents a unit of deployment
- A Pod encapsulates
 - application's container (or, in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





YAML to create Pod

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





Service

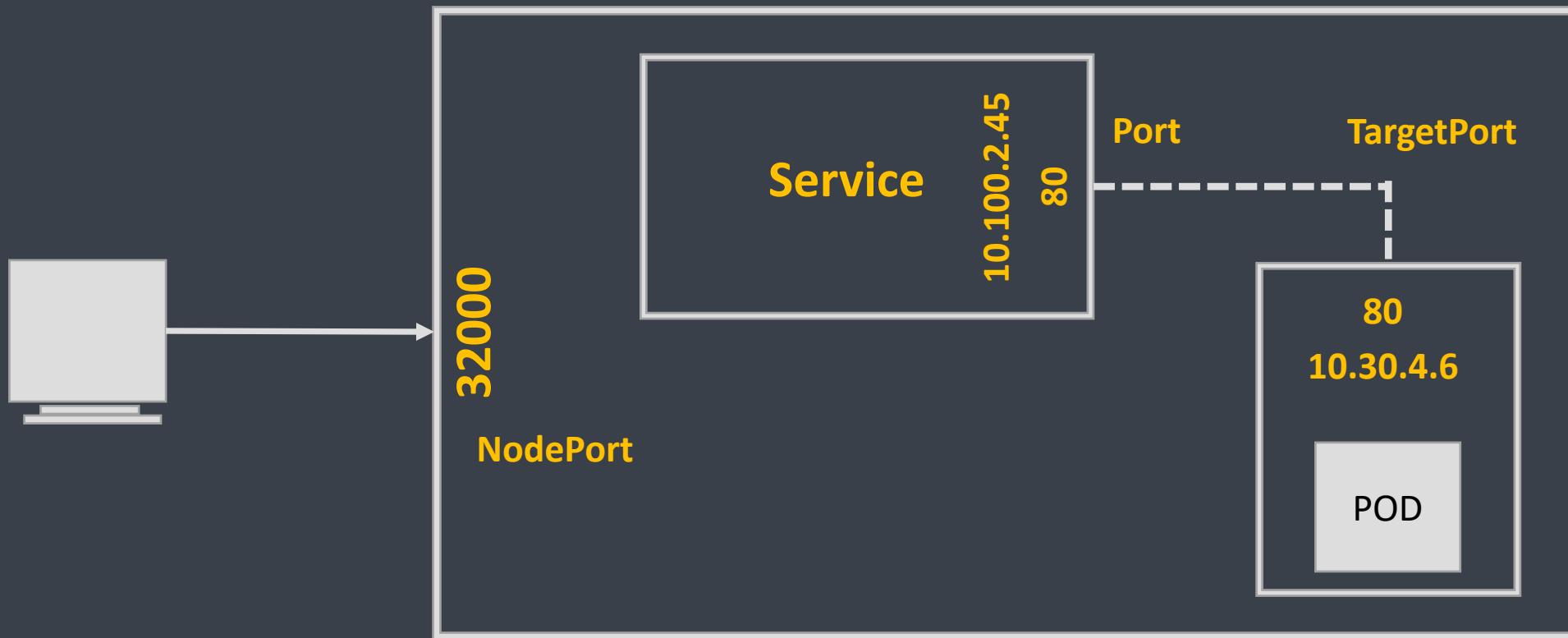
- An abstract way to expose an application running on a set of Pods as a network service
- Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service)
- Service Types
 - ClusterIP
 - Exposes the Service on a cluster-internal 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>





Replica Set → Similar to Service in Docker Swarm

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



```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx ← vs name
spec:
  replicas: 3 → desired count
  selector:
    matchLabels:
      app: nginx ← label
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

A hand-drawn style diagram showing a curly brace grouping the 'template' and 'spec' sections of the YAML code, with an arrow pointing from it to the text 'create pod'.



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