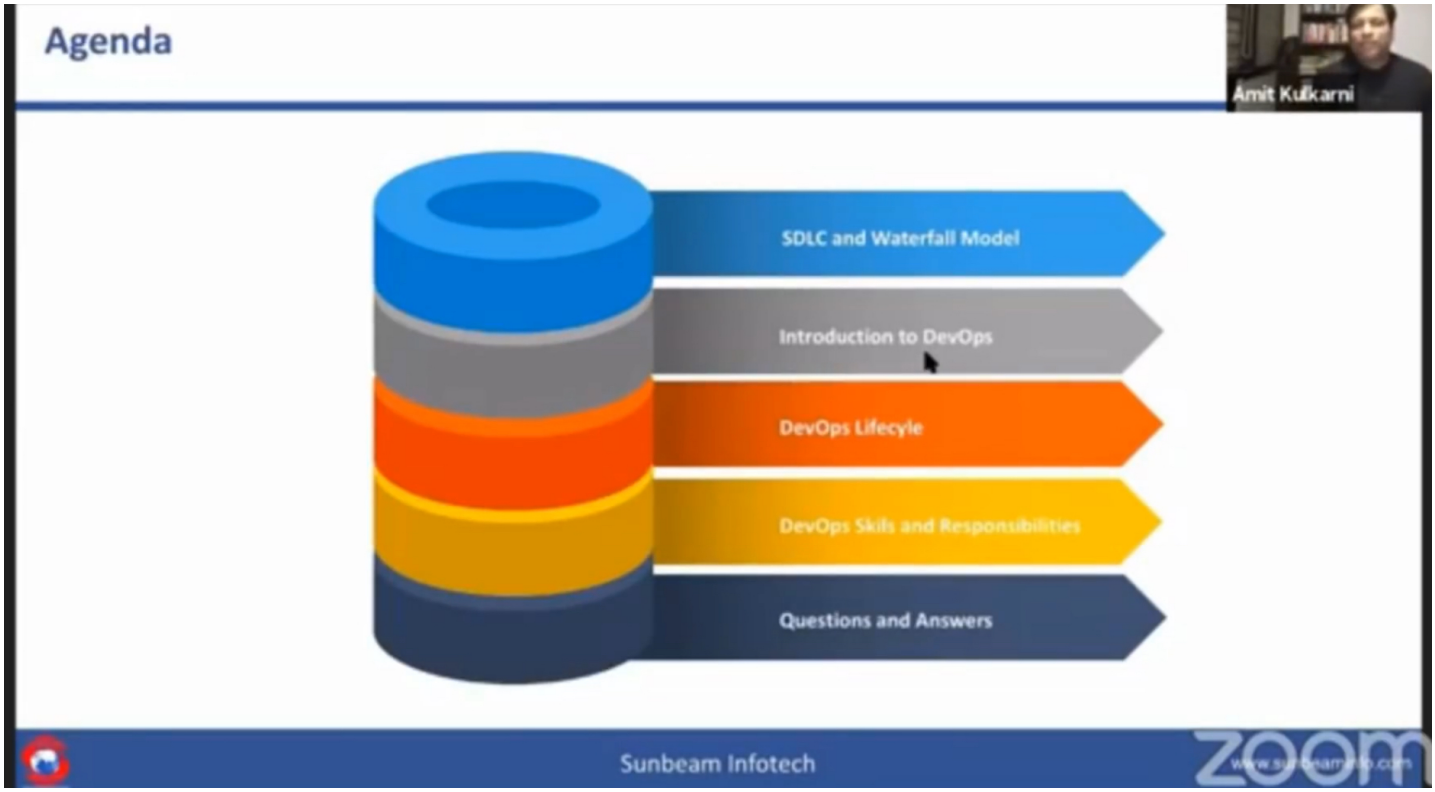
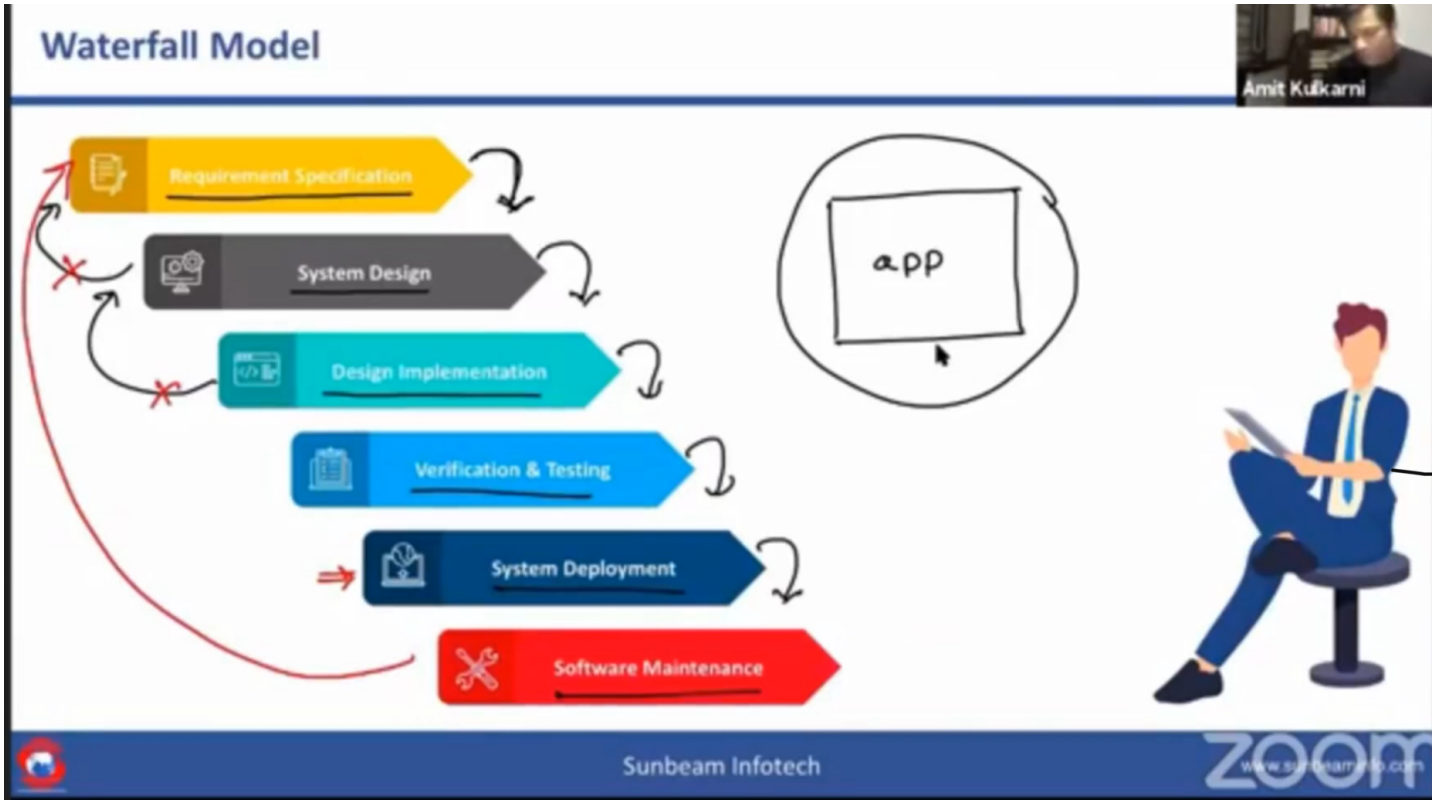
Title

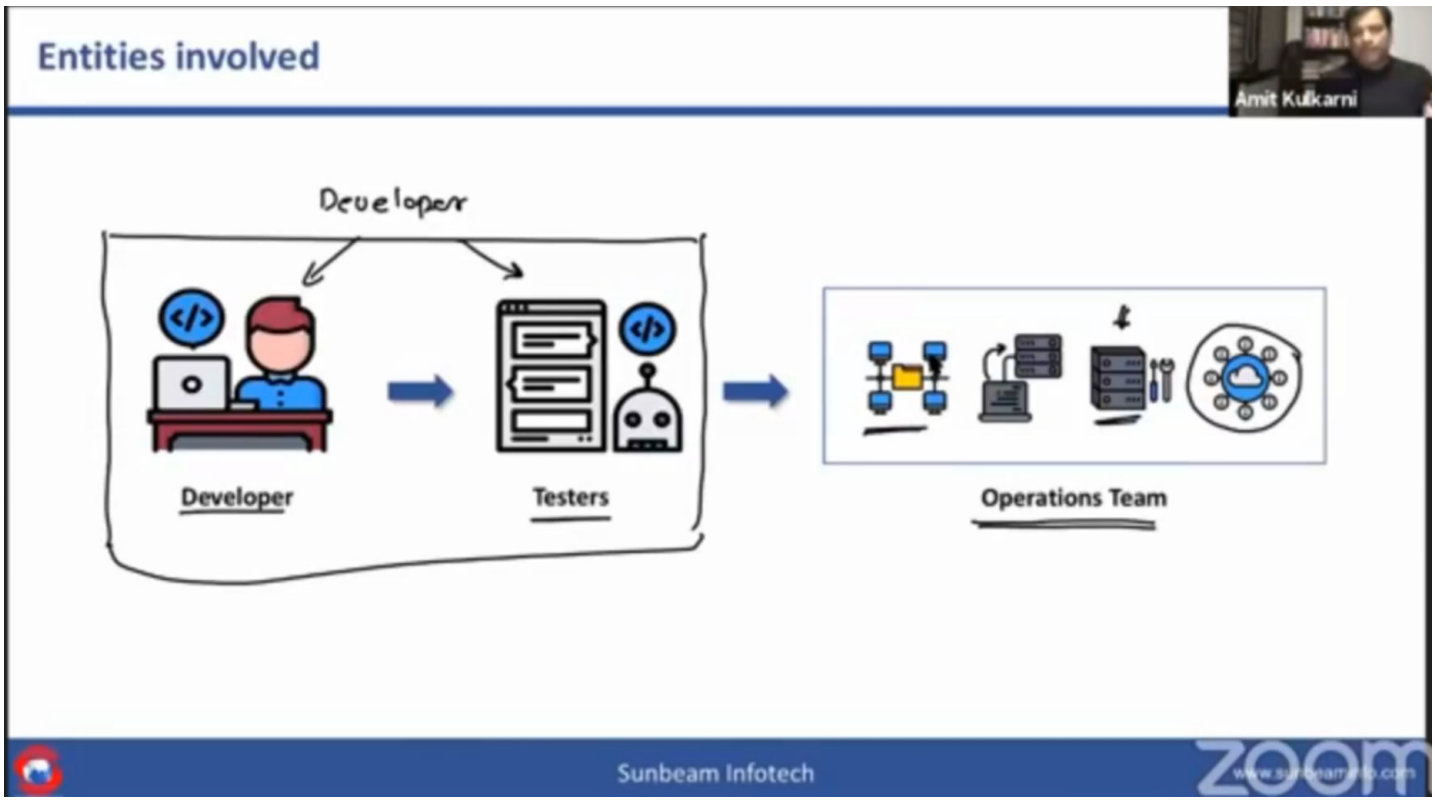

Agenda
Amit Kulkarni
SDLC and Waterfall Model
Introduction to DevOps
DevOps Lifecyle
DevOps Skils and Responsibilities
Questions and Answers
Sunbeam Infotech
zoom


Software Development Lifecycle
Amit Kulkarni
PLANNING
Talk to customer and understand the requirements
DEFINING
Define the requirements and stick to them
DESIGNING
Design the solution with right approach
BUILDING
TESTING
Make sure that your code is working
DEPLOYMENT
Make your app available for rest of the world
Sunbeam Infotech
zoom

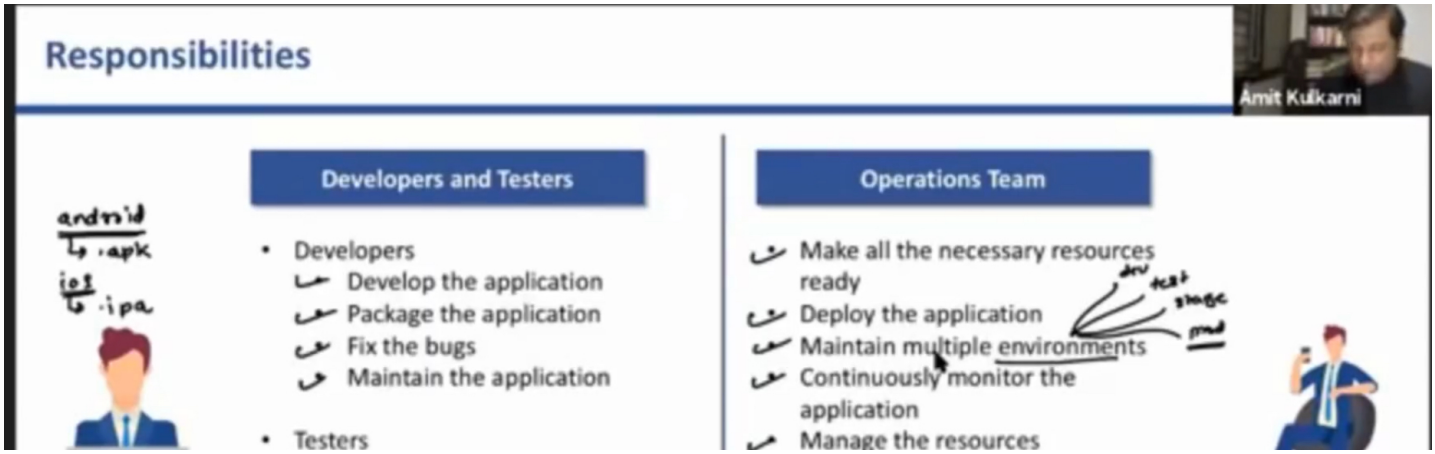
Waterfall Model



Entities involved

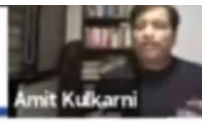


Responsibilities



- Thoroughly test the application manually or using test automation
- Report the bugs to the developer

Challenges



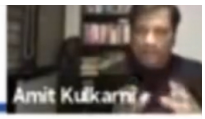
Developers and Testers

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

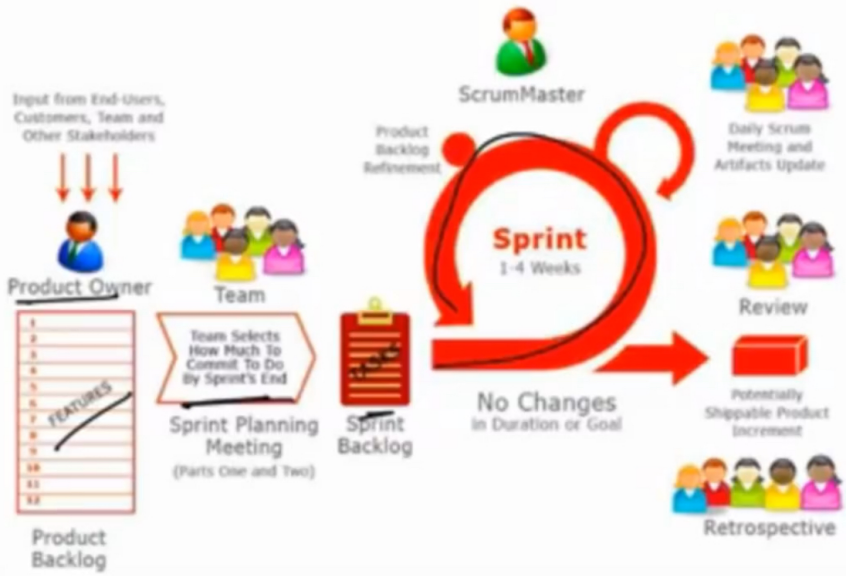
Operations Team

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

Agile Development



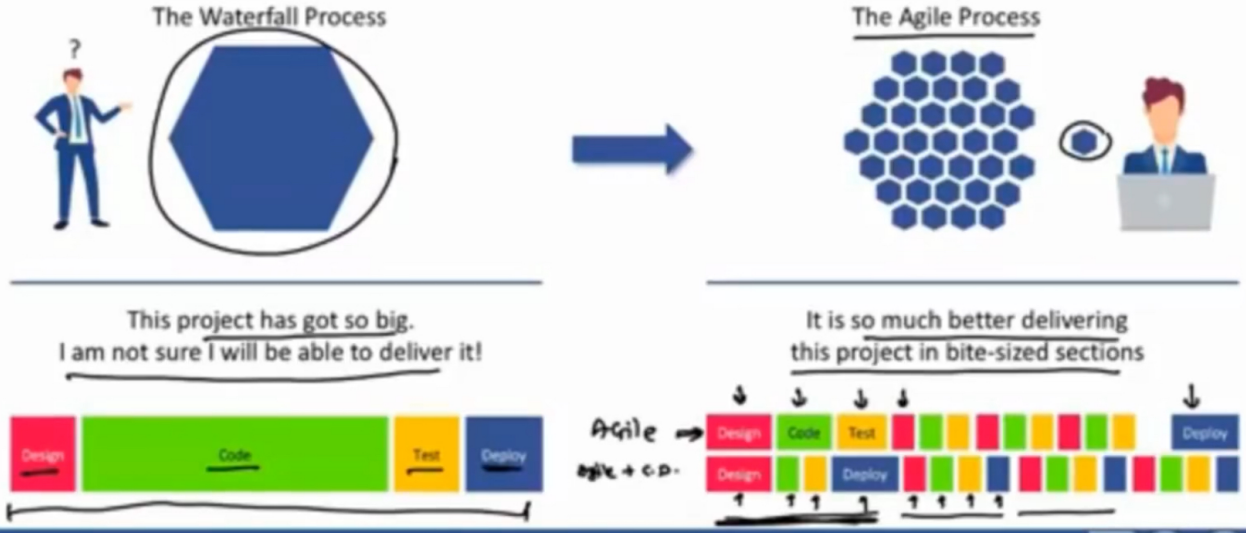
Scrum Process



Amit Kulkarni

Sunbeam Infotech

Waterfall Vs Agile



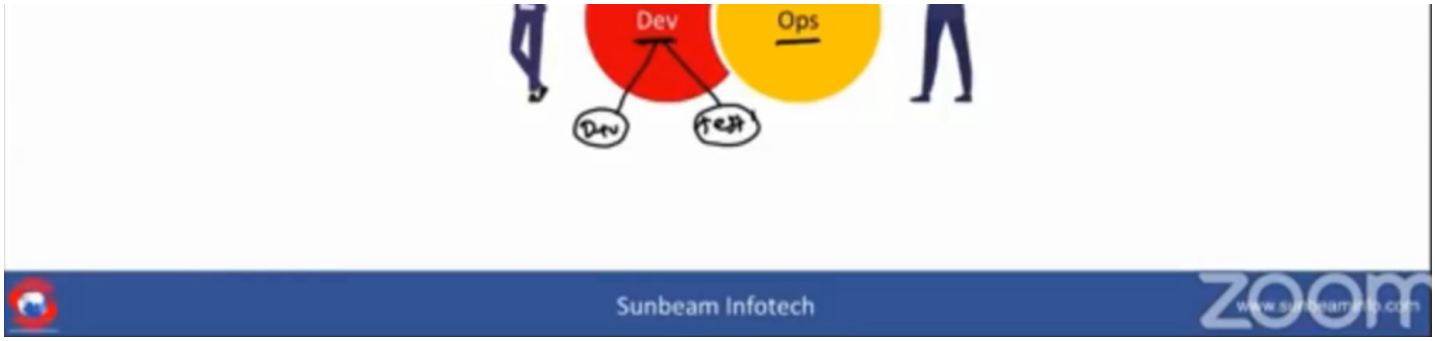
Amit Kulkarni

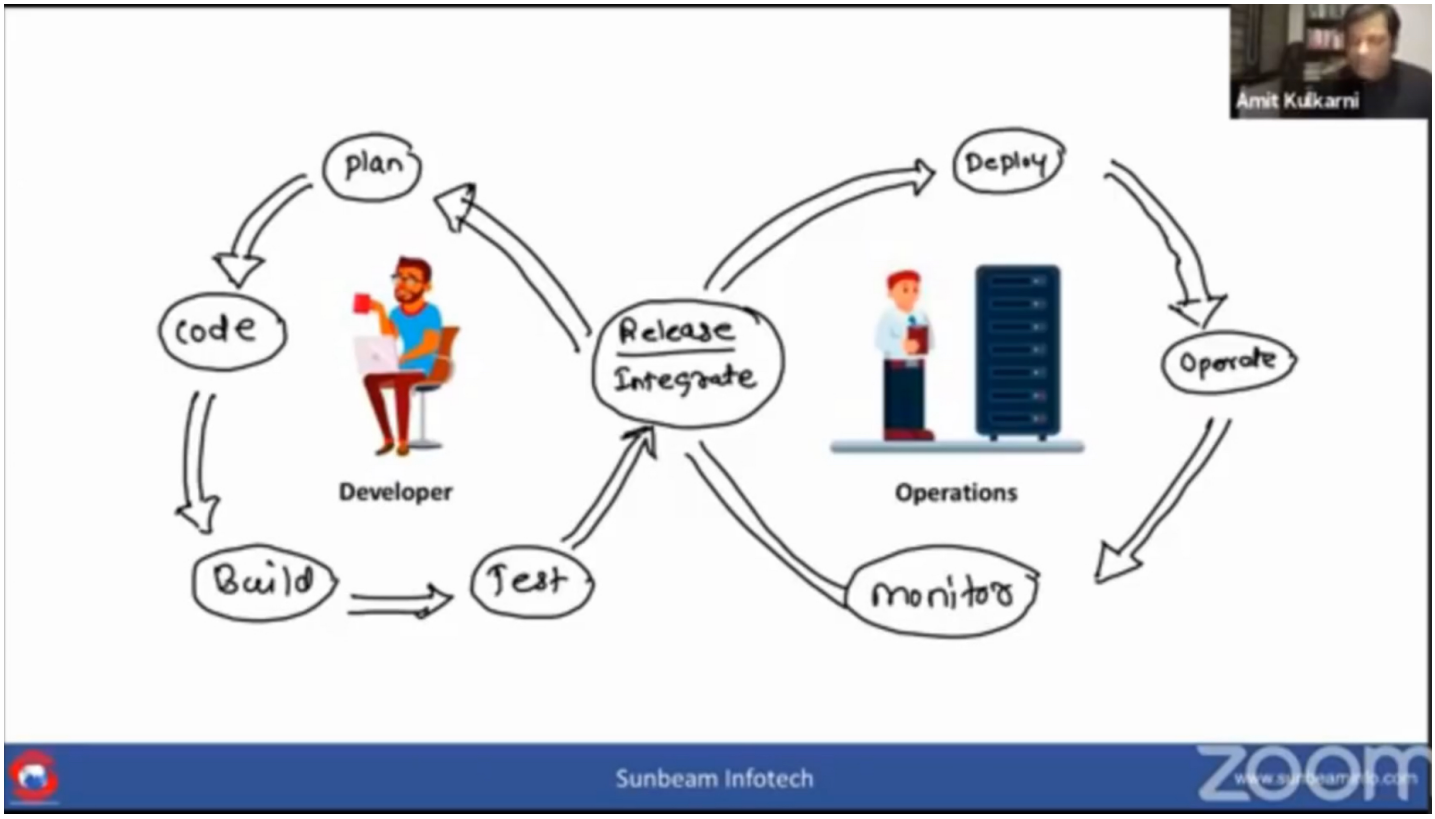
Sunbeam Infotech

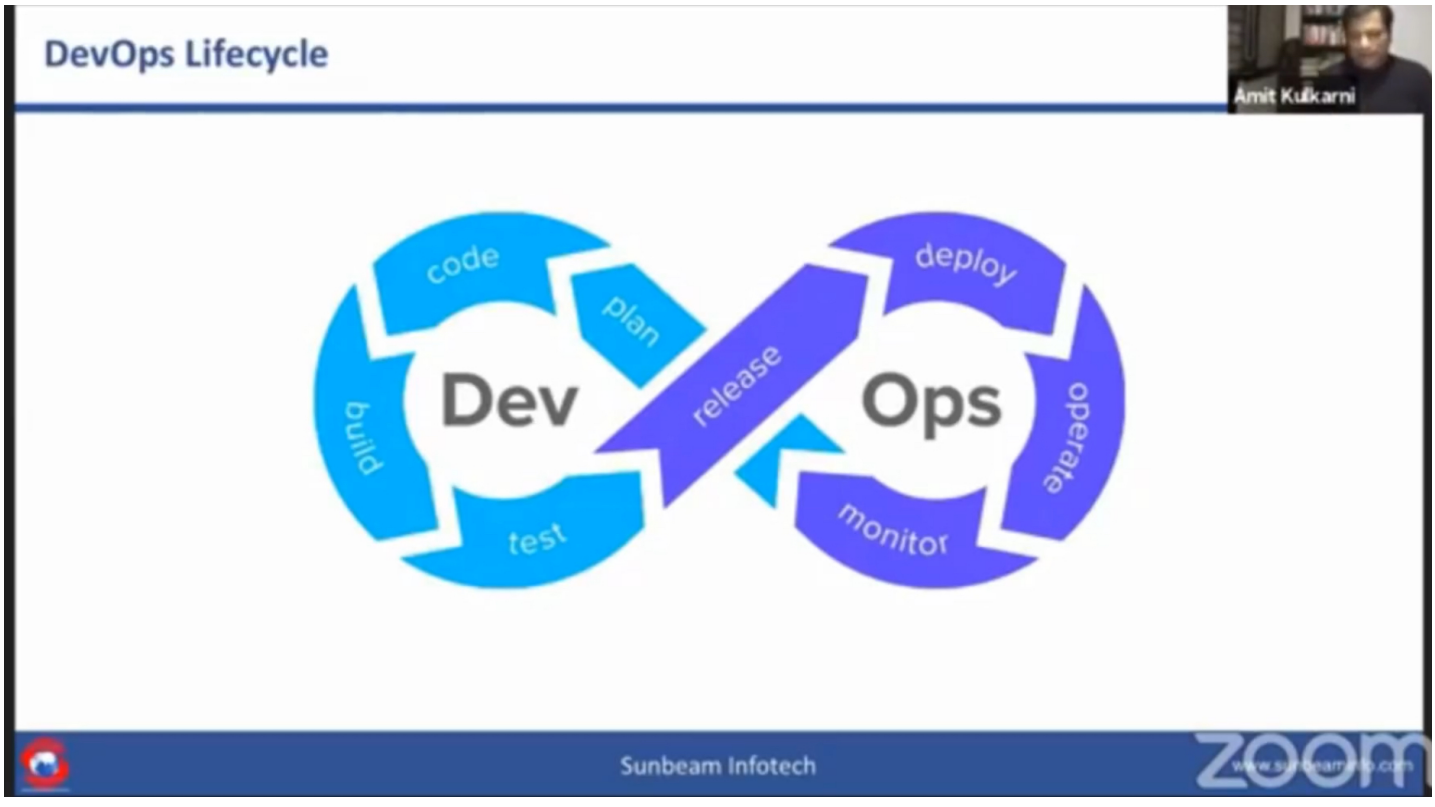
DevOps

DevOps is a software development strategy which brings the gap between the Dev and the Ops side of the company

Amit Kulkarni


Dev
Ops
Dev
Test
Sunbeam Infotech

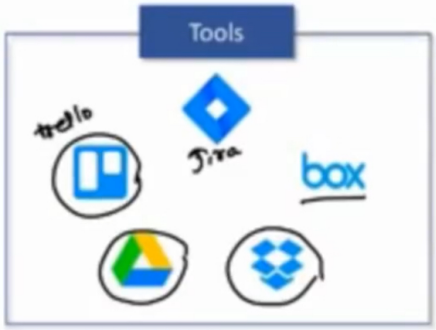

Amit Kulkarni
plan
code
Build
Test
Release
Integrate
Deploy
Operate
monitor
Developer
Operations
Sunbeam Infotech


DevOps Lifecycle
Amit Kulkarni
code
plan
build
test
release
deploy
operate
monitor
Dev
Ops
Sunbeam Infotech


DevOps Lifecycle - Plan

Amit Kulkarni

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

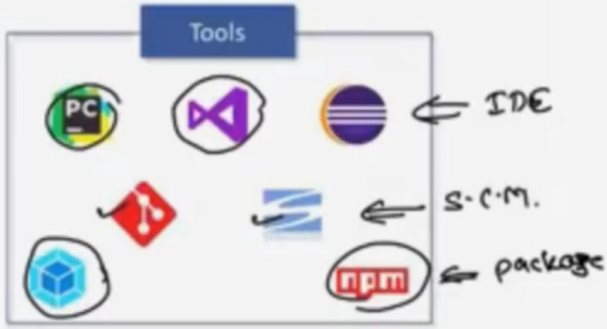


Sunbeam Infotech

DevOps Lifecycle - Code

Amit Kulkarni

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



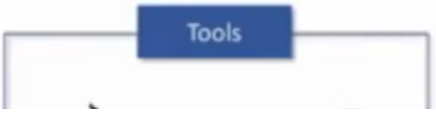
Sunbeam Infotech

DevOps Lifecycle -Build

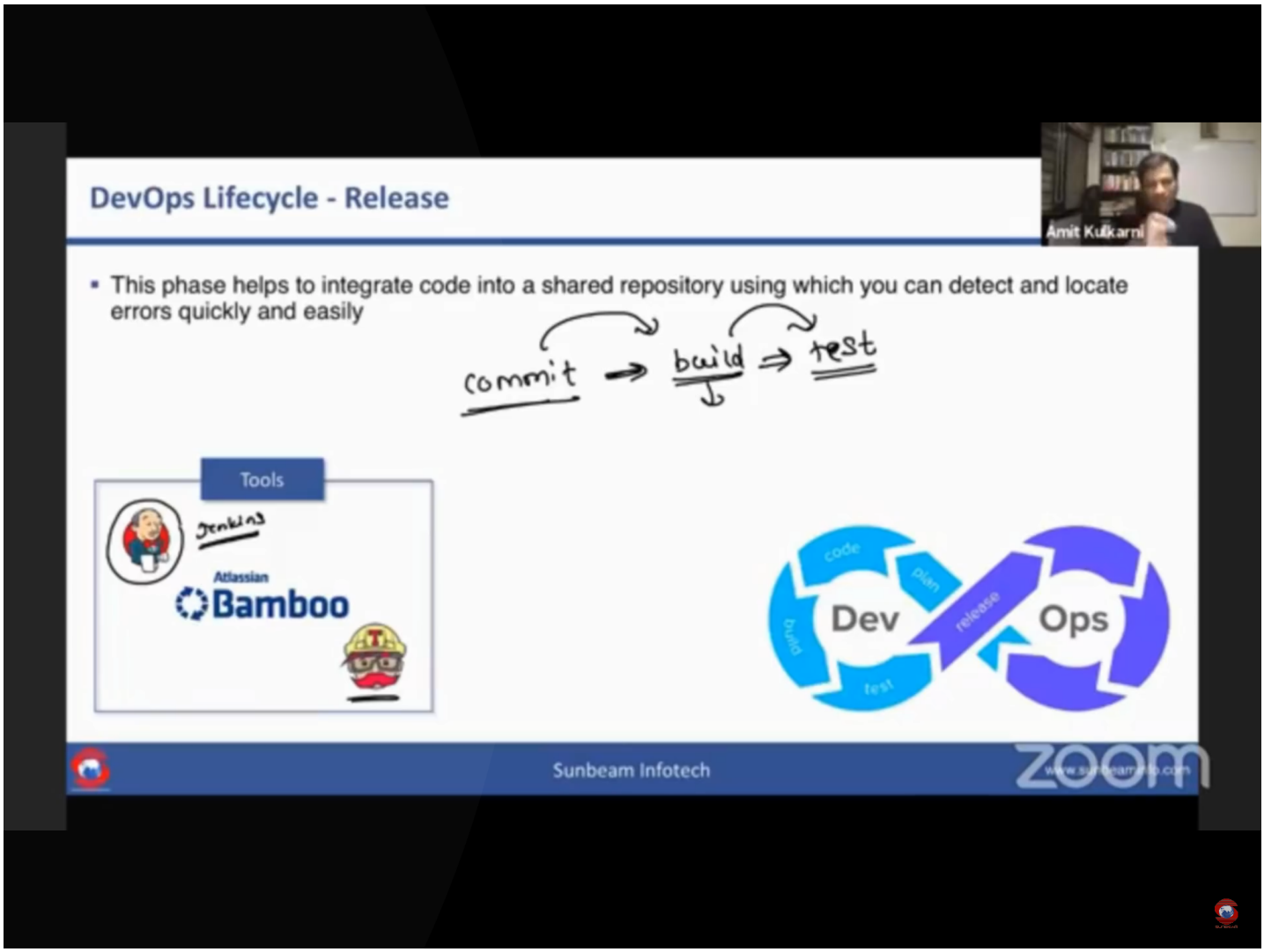
Amit Kulkarni

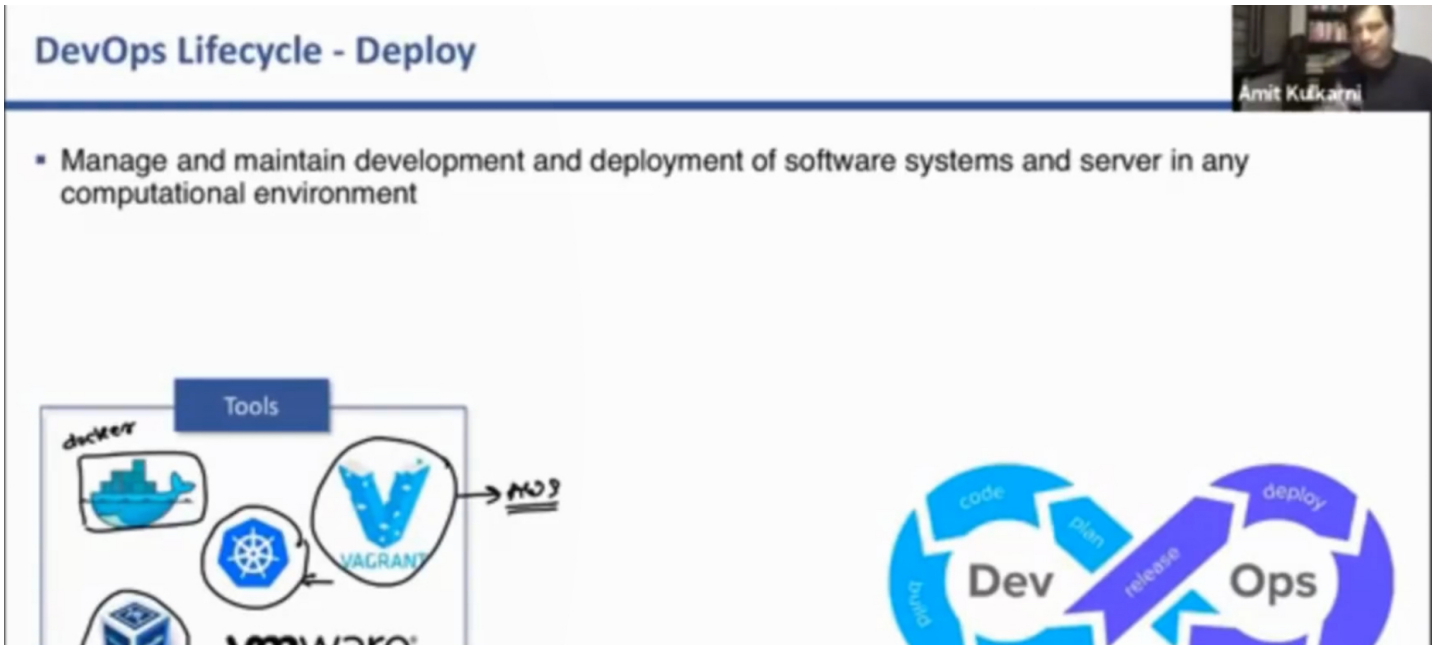
- Integrating the required libraries
- Compiling the source code
- Create deployable packages

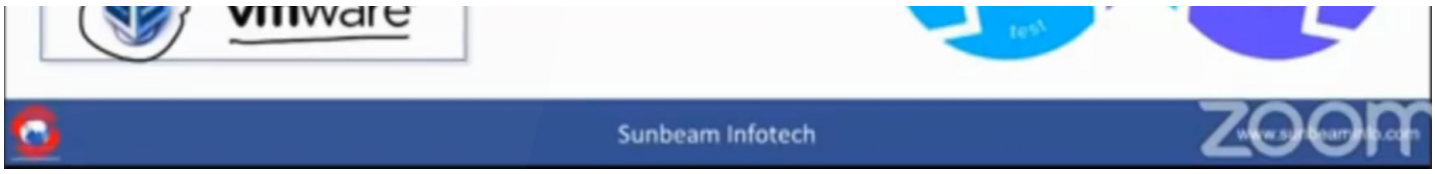
android = .apk
ios = .ipa
web = html
angular



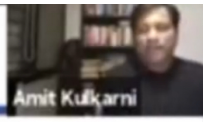

Ant
Gradle
maven
code
plan
build
Dev
Ops
Sunbeam Infotech
zoom


DevOps Lifecycle - Release
Amit Kulkarni
This phase helps to integrate code into a shared repository using which you can detect and locate errors quickly and easily
commit
build
test
Tools
Jenkins
Atlassian
Bamboo
code
plan
build
test
release
Dev
Ops
Sunbeam Infotech
zoom

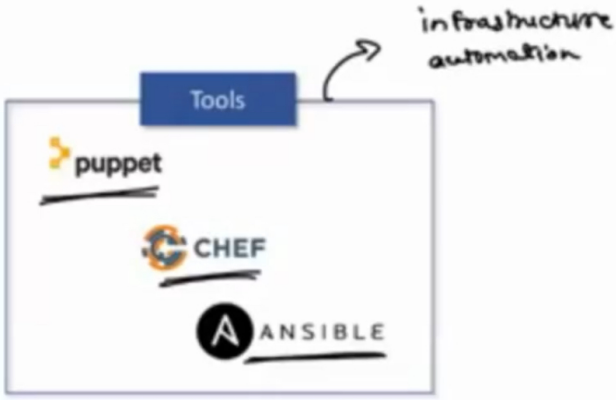

DevOps Lifecycle - Deploy
Amit Kulkarni
Manage and maintain development and deployment of software systems and server in any computational environment
Tools
docker
VAGRANT
AWS
code
plan
build
release
deploy
Dev
Ops



DevOps Lifecycle - Operate

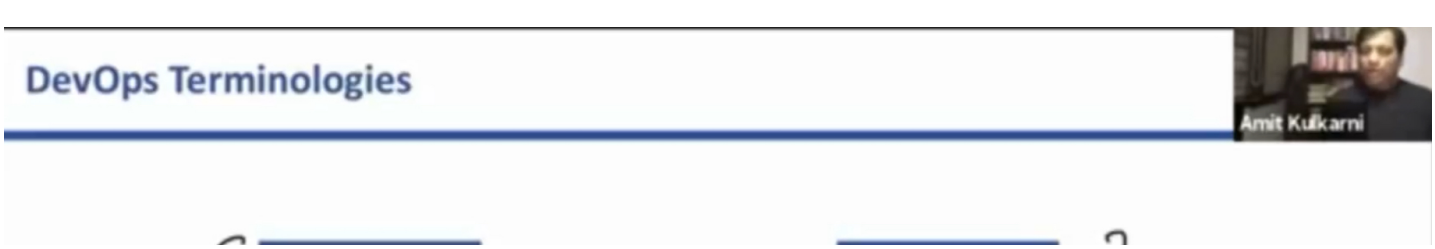


- This stage where the updated system gets operated

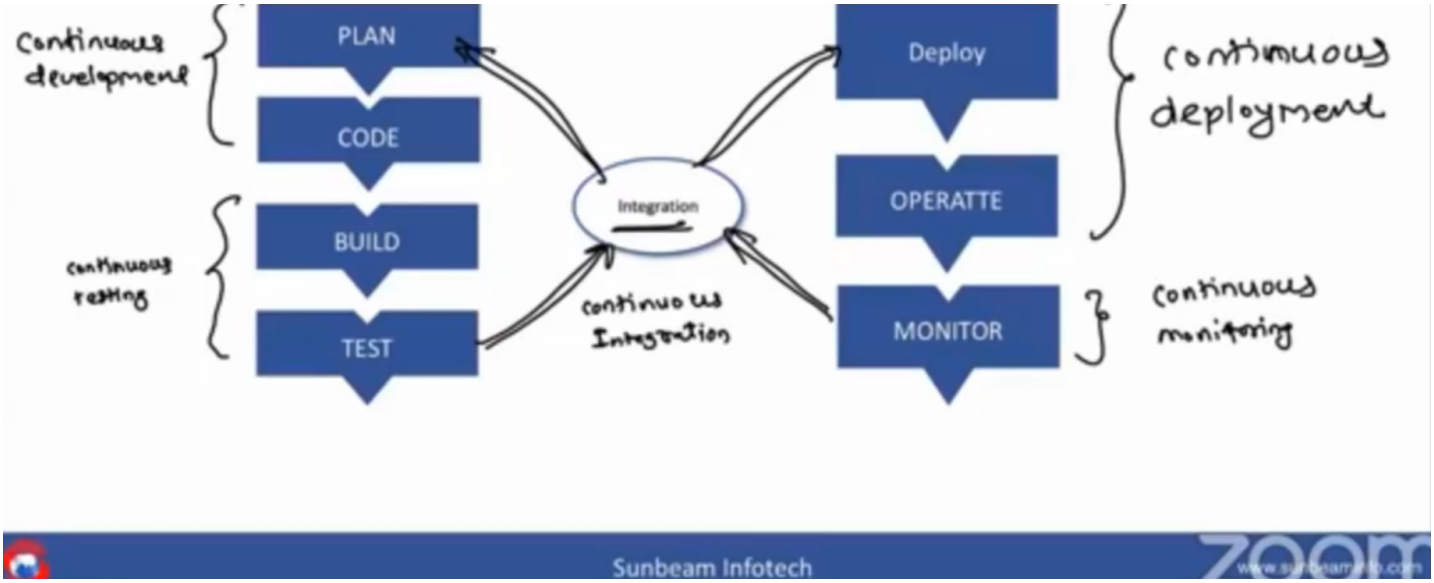


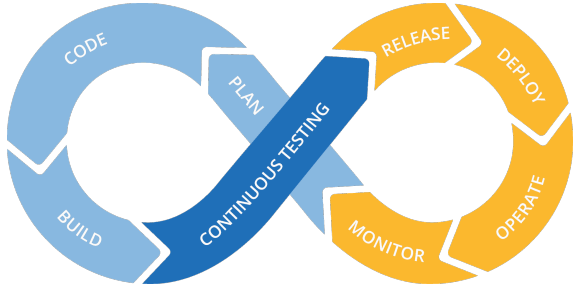
DevOps Lifecycle - Monitor

- 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

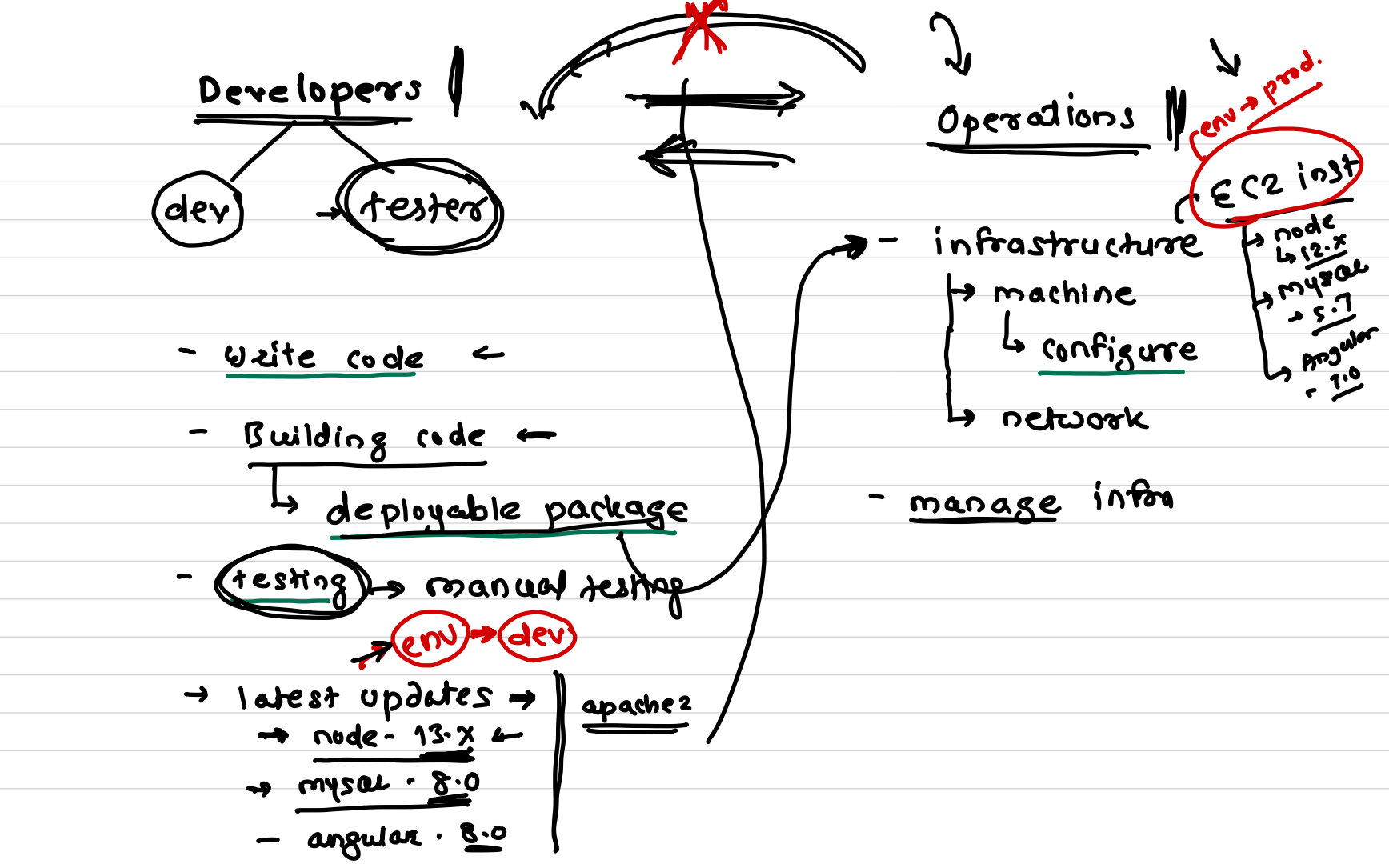


DevOps Terminologies


Continuous development
PLAN
CODE
BUILD
TEST
continuous testing
Integration
continuous Integration
Deploy
OPERATTE
MONITOR
continuous deployment
continuous monitoring
Sunbeam Infotech


CODE
PLAN
BUILD
CONTINUOUS TESTING
RELEASE
DEPLOY
MONITOR
OPERATE

DevOps


Developers
dev
tester
Operations
env → prod.
EC2 inst
node 12.x
mysql 5.7
Angular 7.0
infrastructure
machine
configure
network
manage infra
write code
Building code
deployable package
testing
manual testing
env
dev
latest updates
node - 13.x
mysql - 8.0
angular - 8.0
apache2

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

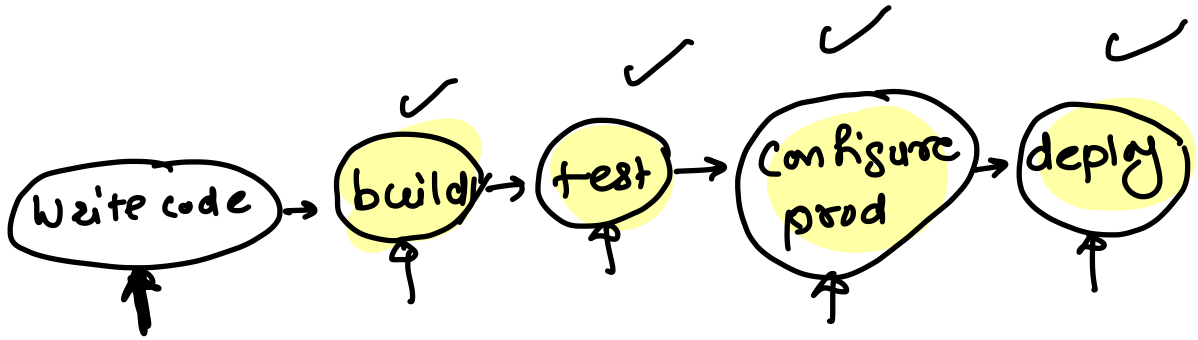
Solutions to the problem

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

Overview

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

cloud [AWS]



Why DevOps is Needed?

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

What is DevOps ?

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

Common misunderstanding

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

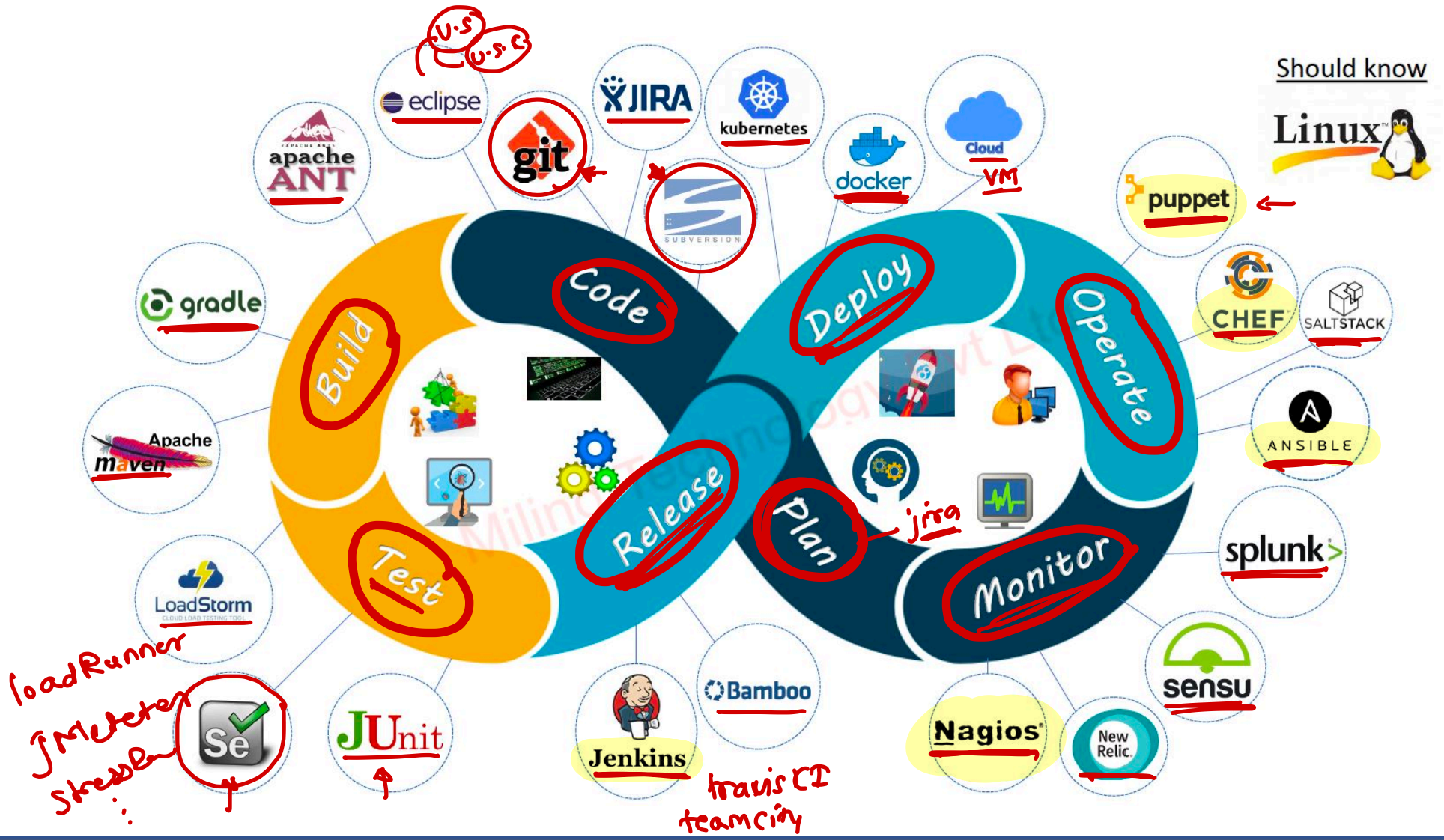
Reasons to use DevOps

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

Reasons to use DevOps

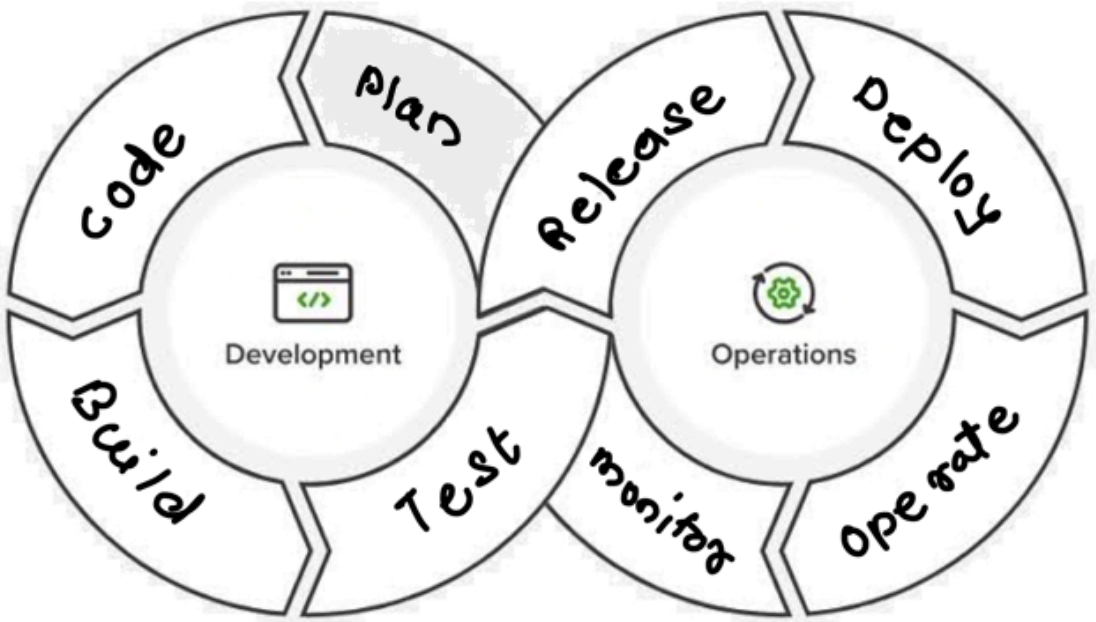
- **Cost Efficiency:** DevOps offers cost efficiency in the software development process which is always an aspiration of IT companies' management
- **Breaks larger code base into small pieces:** DevOps is based on the agile programming method. Therefore, it allows breaking larger code bases into smaller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

DevOps Lifecycle



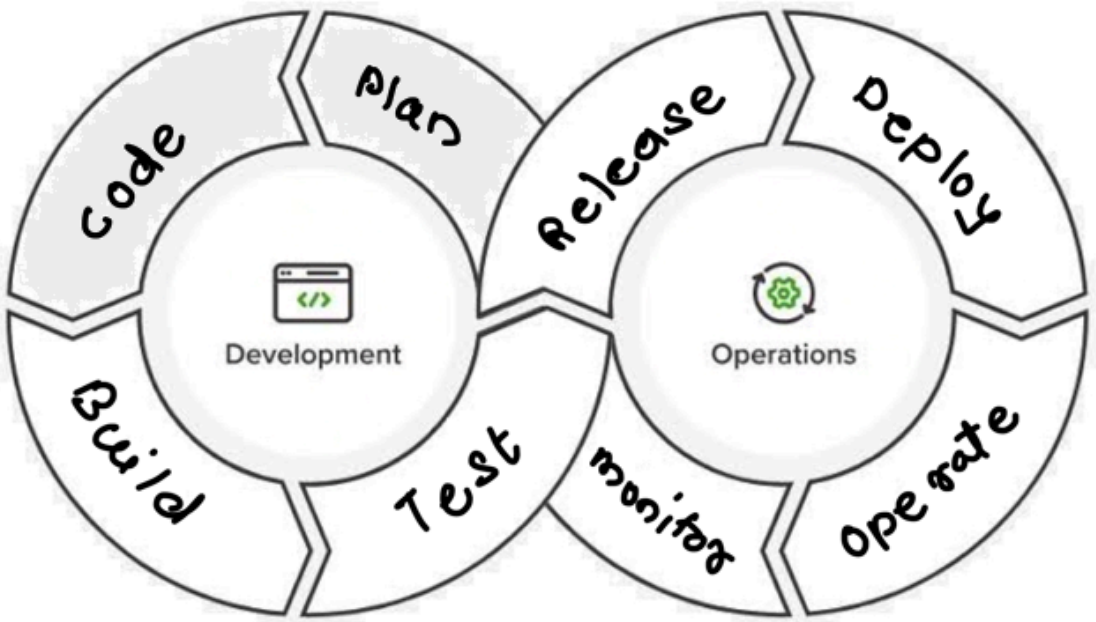
DevOps Lifecycle - Plan

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



DevOps Lifecycle - Code

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



DevOps Lifecycle -Build

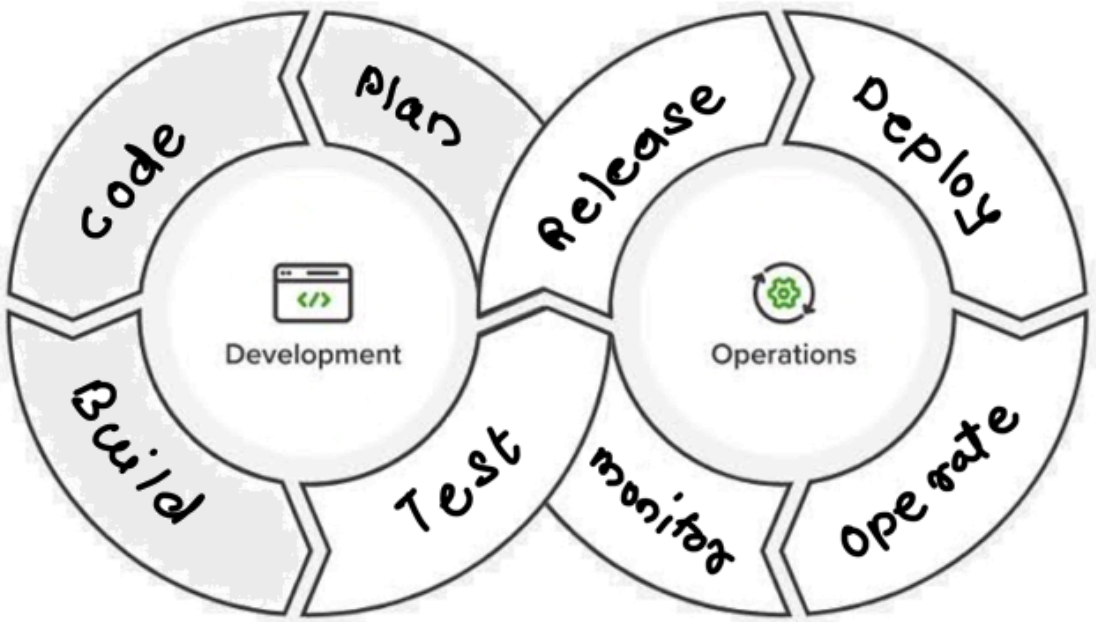
- Integrating the required libraries
- Compiling the source code
- Create deployable packages
- Build tools
  - Maven
  - Gradle
  - Ant

① Java
↳ maven > .jar, .war

② Angular
↳ ng build > .html, .js

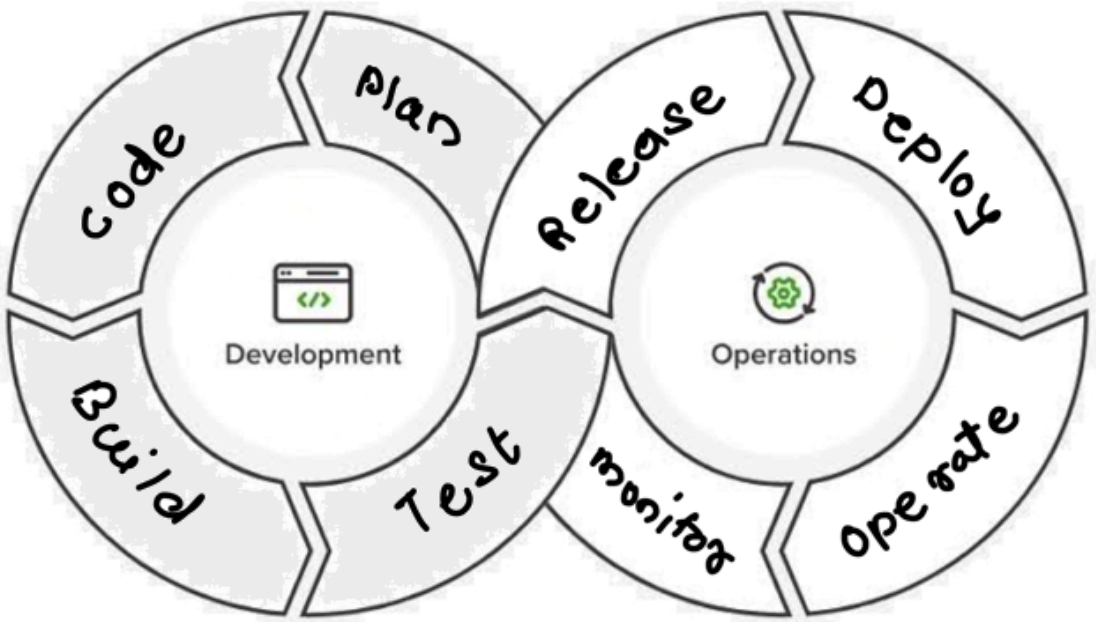
③ Android
↳ java → gradle > .apk

④ iOS
↳ swift → xcodebuild > .ipa



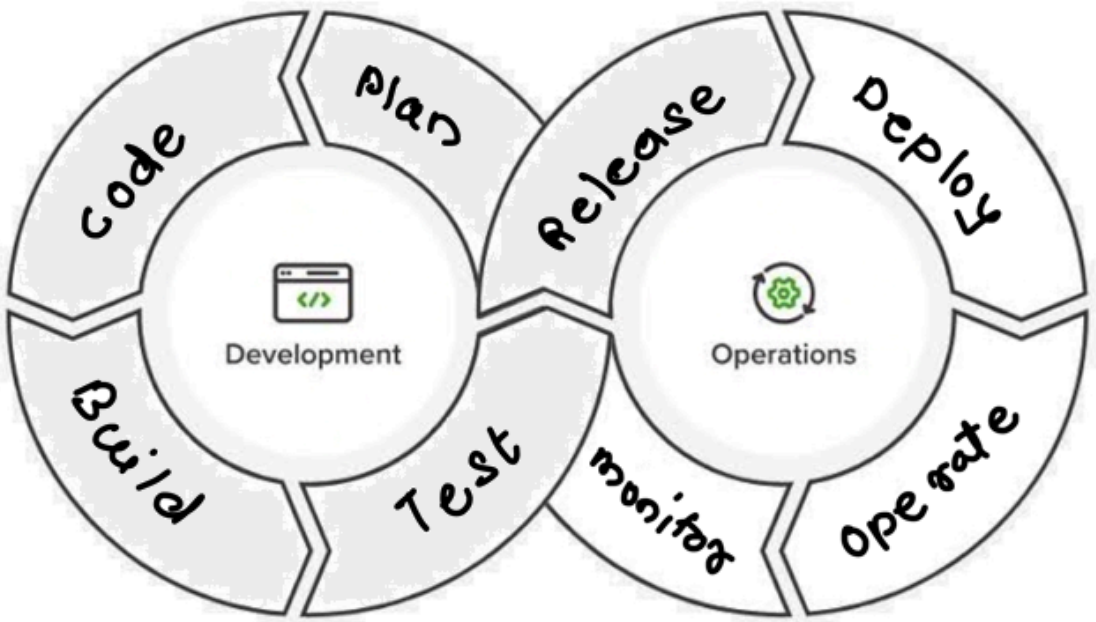
DevOps Lifecycle - Test

- Process of executing automated tests
- The goal here is to get the feedback about the changes as quickly as possible
- Testing tools
  - JMeter → load
  - Selenium → java, c++, js, python
  - JUnit → java
  - QUnit → ...
  - NUnit → .net
  - Appium → android



DevOps Lifecycle - Release

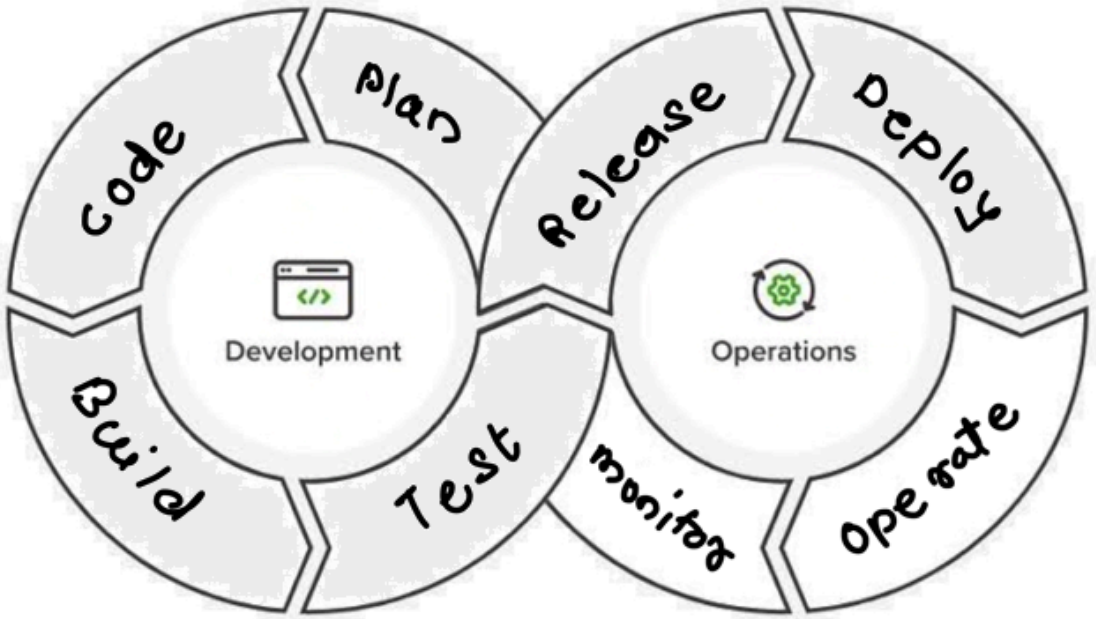
- This phase helps to integrate code into a shared repository using which you can detect and locate errors quickly and easily
- Release tools
  - Jenkins
  - Travis CI
  - Bamboo
  - GitLab CI



DevOps Lifecycle - Deploy

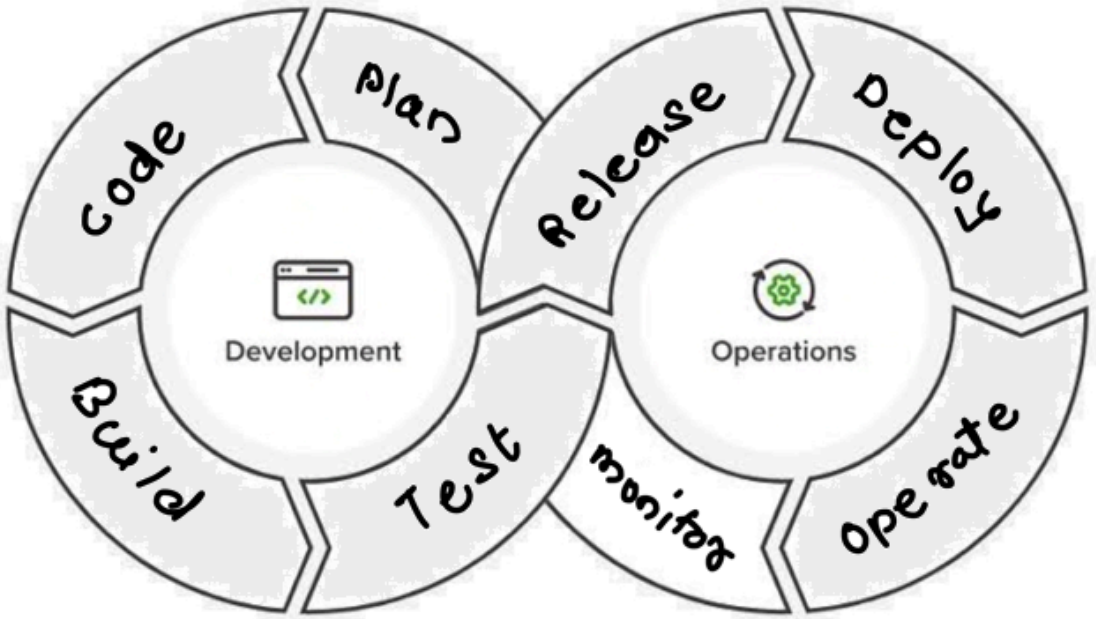
- Manage and maintain development and deployment of software systems and server in any computational environment
- Deployment tools
  - Docker / VM – EC2 instance
  - Kubernetes / docker swarm
  - Virtual Machines
- Configuration management tools
  - Puppet
  - Chef
  - Ansible

many machines



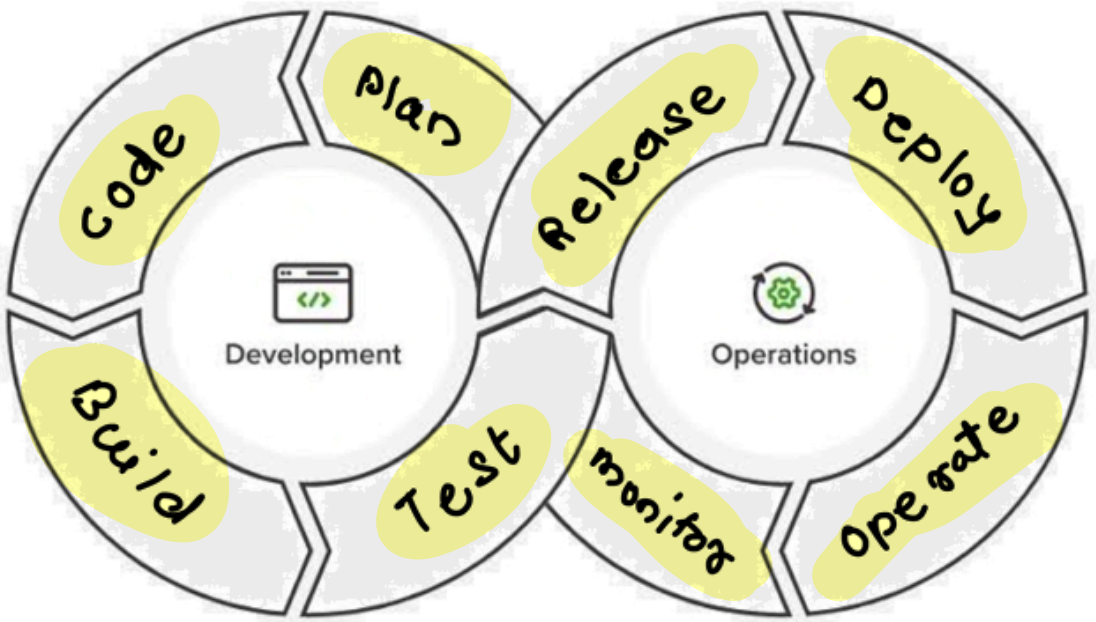
DevOps Lifecycle - Operate

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



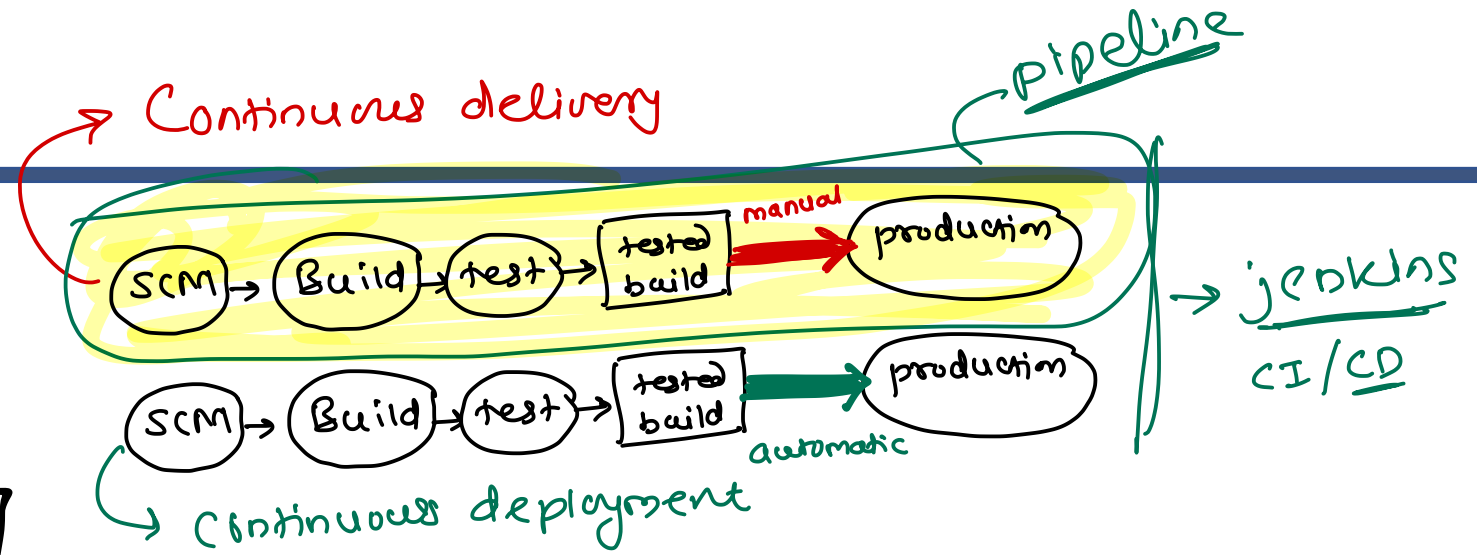
DevOps Lifecycle - Monitor

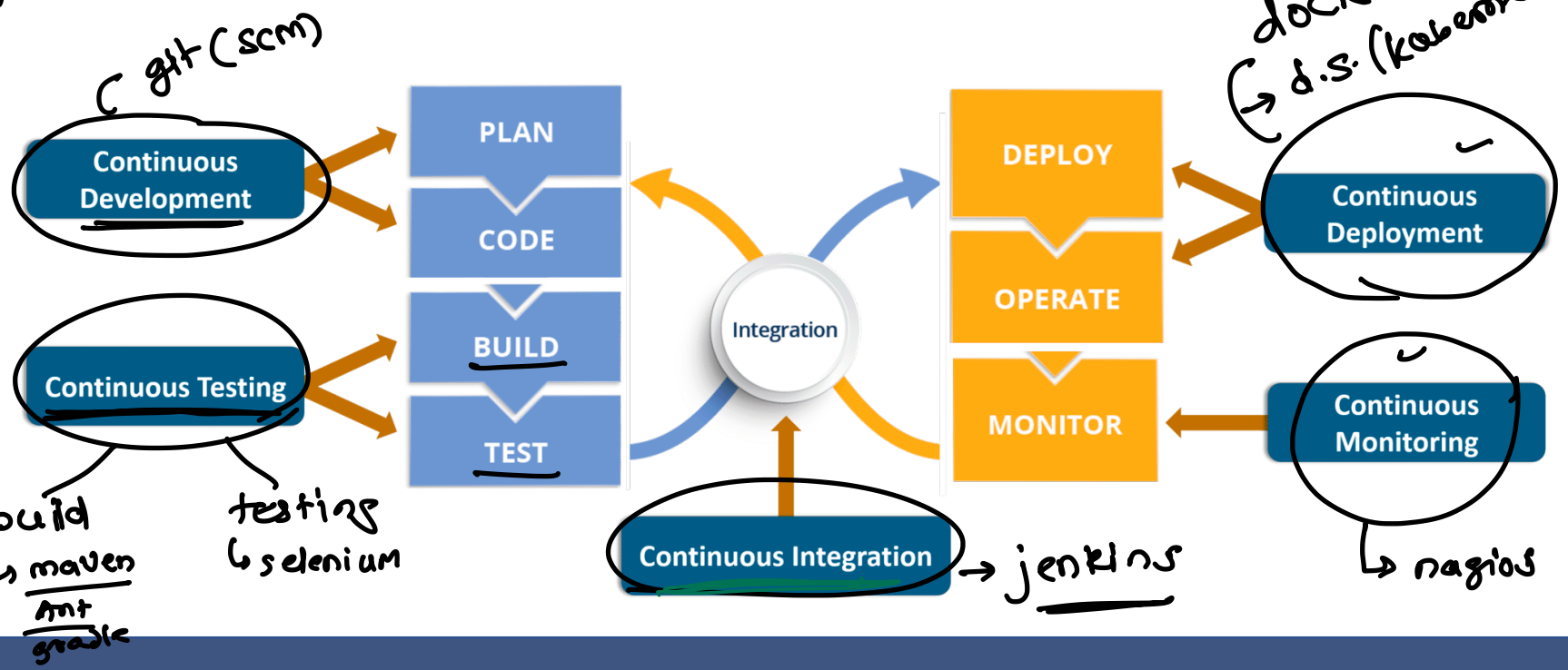
- It ensures that the application is performing as expected and the environment is stable
- It quickly determines when a service is unavailable and understand the underlying causes
- Monitoring tools
  - Nagios
  - Sensu
  - Splunk
  - DataDog



DevOps Terminologies

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





Source Code Management

scm / vcs

Why do we need Version Control System?

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

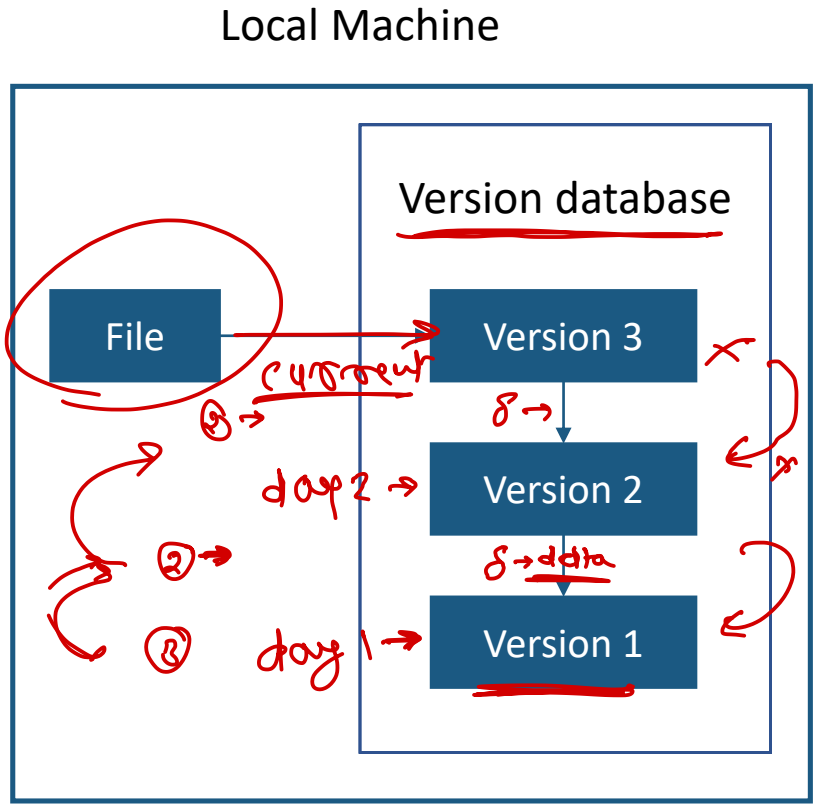
What is Version Control System?

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

Local Version Control System

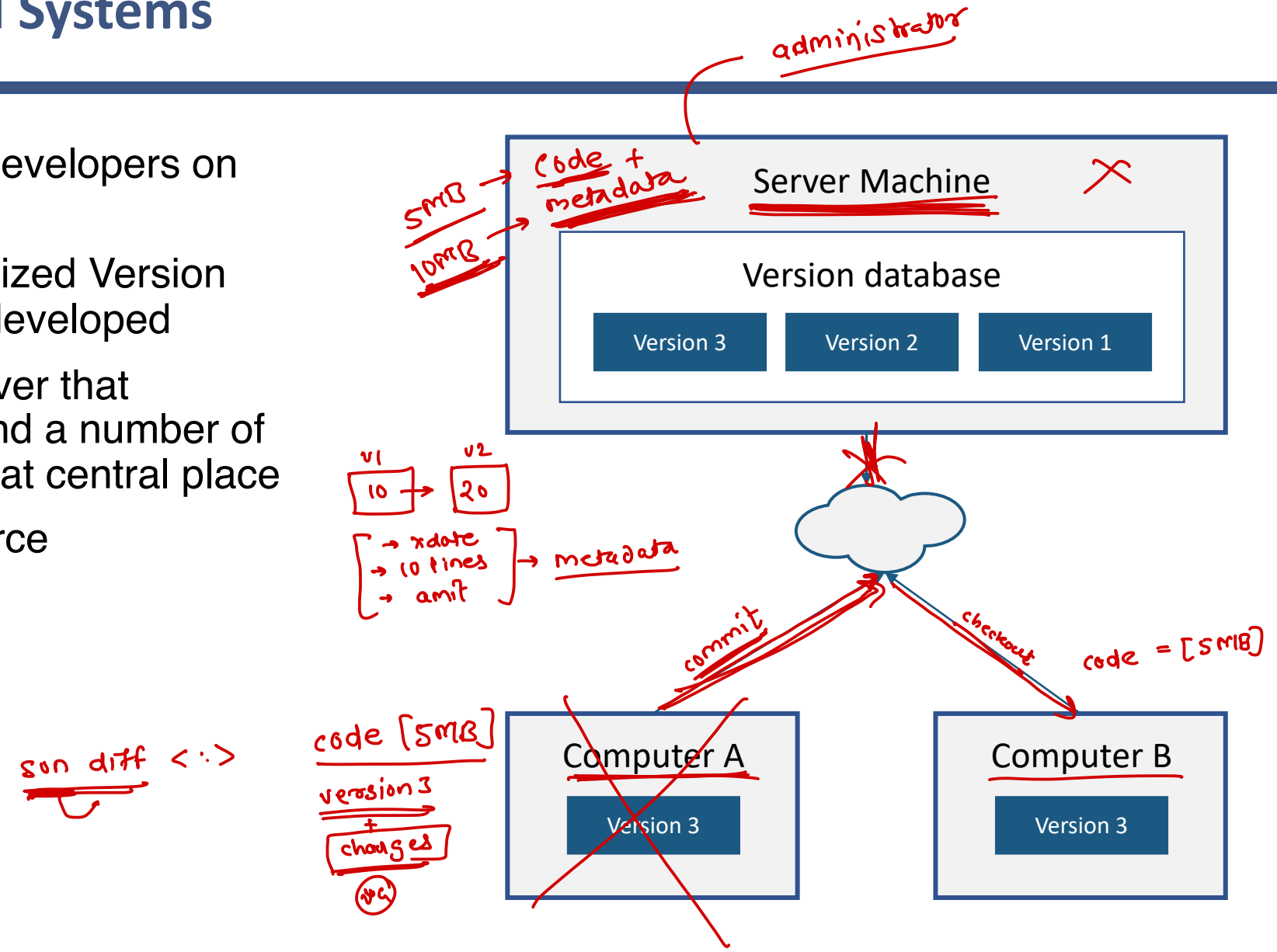
- Contains simple database that kept all the changes to files under revision control
- One of the more popular VCS tools was a system called RCS
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk
- It can then re-create what any file looked like at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

update
- add
- modify
- delete



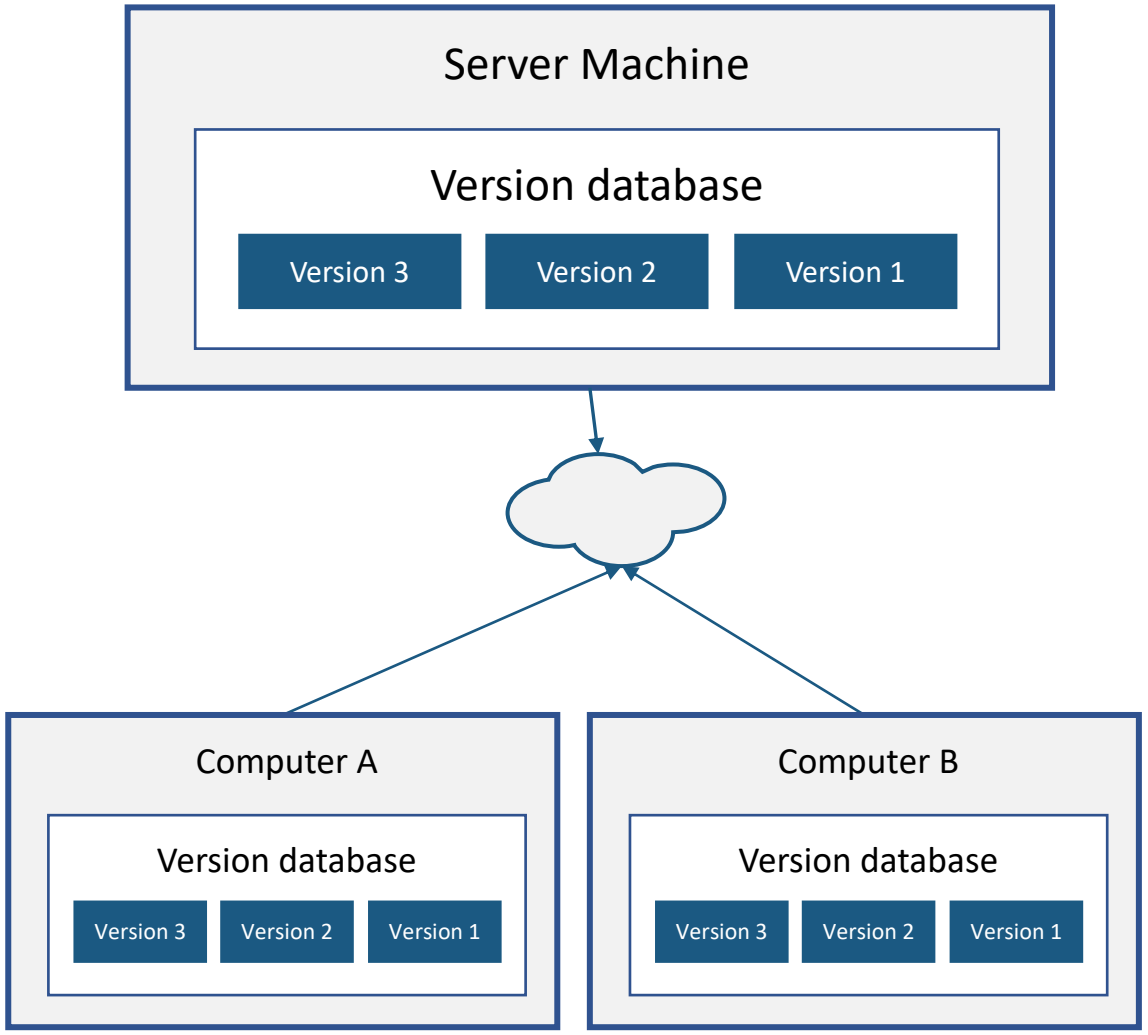
Centralized Version Control Systems

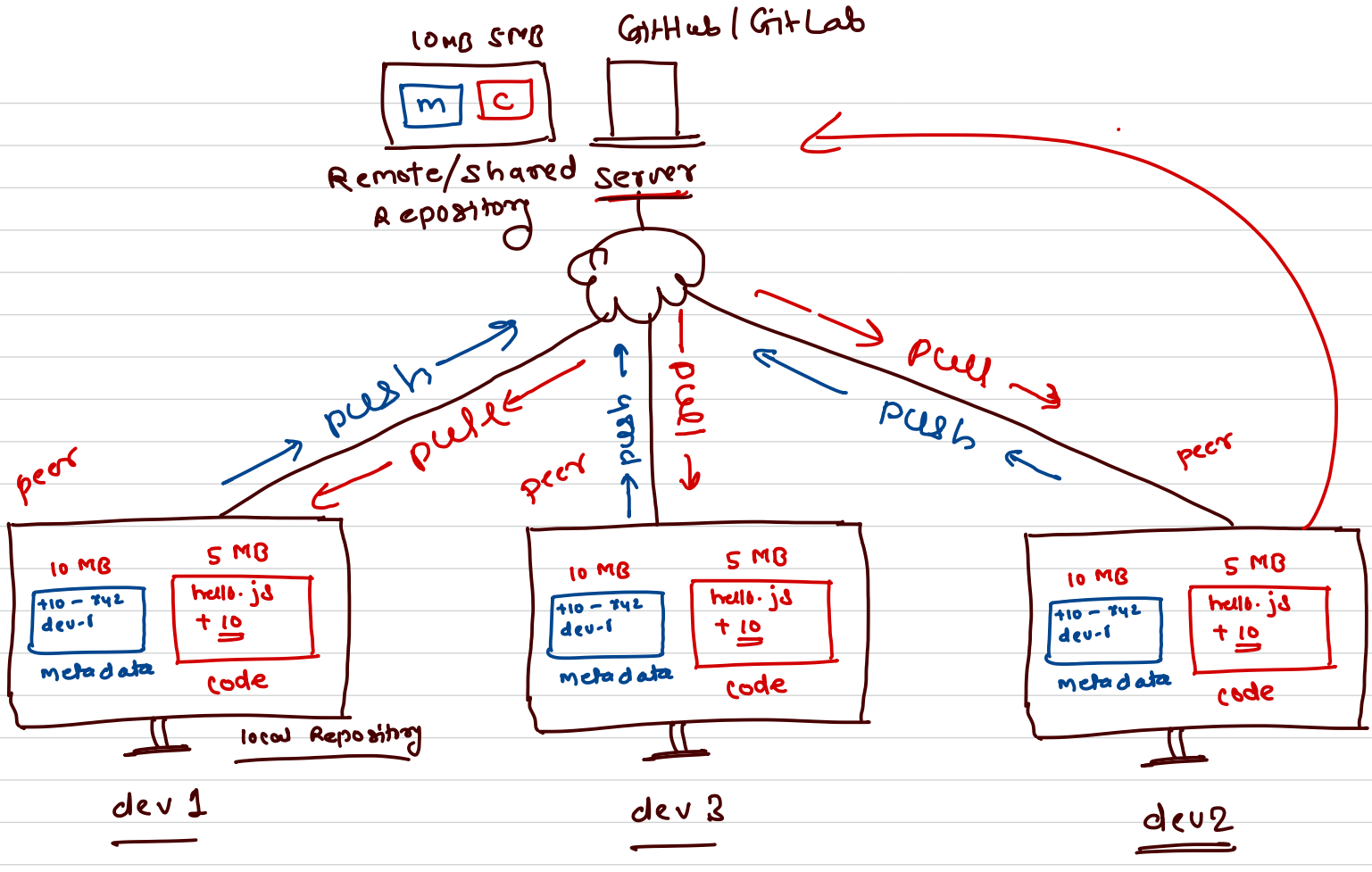
- People need to collaborate with developers on other systems
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place
- E.g.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



Distributed Version Control Systems

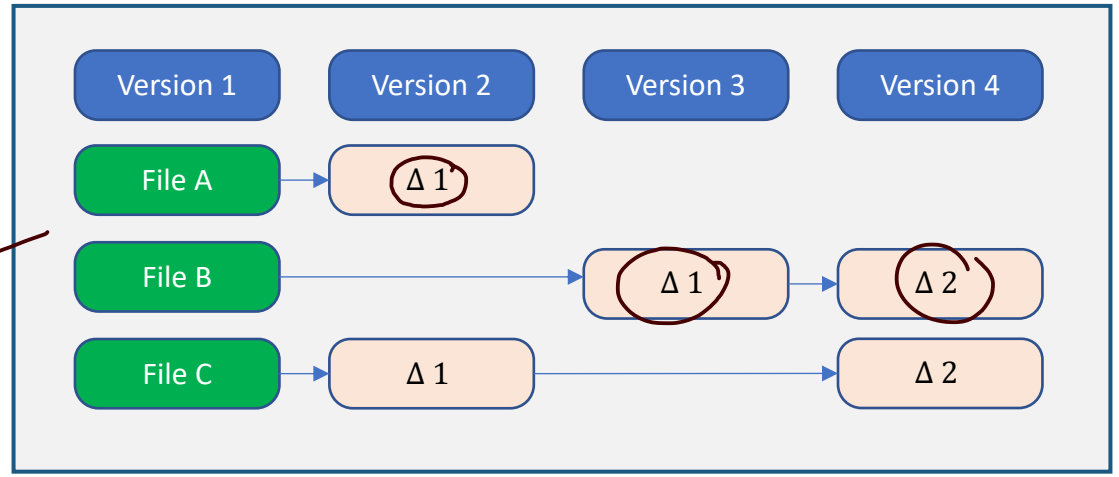
- Clients don't just check out the latest snapshot of the files, rather they fully mirror the repository
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- Every checkout is really a full backup of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

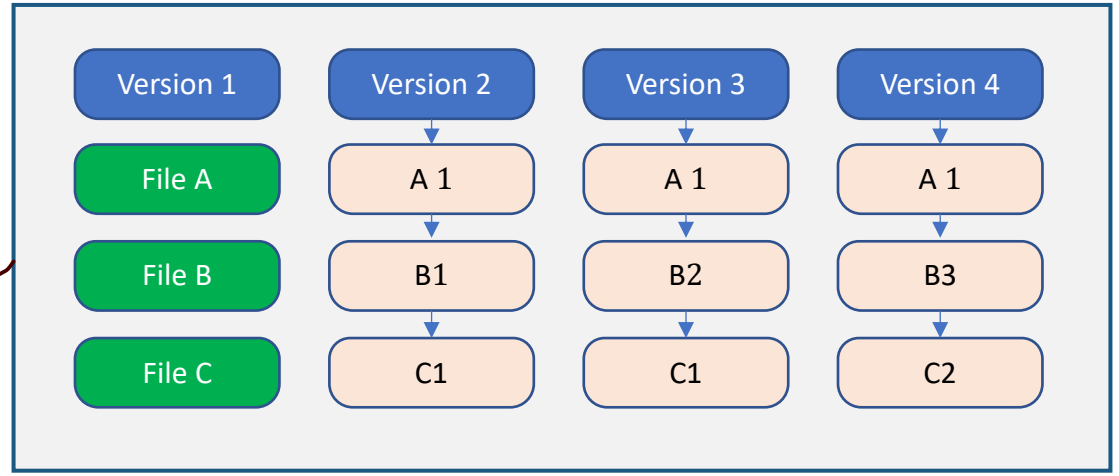



10MB SMB
m
c
GitHub / GitLab
Remote/shared Repository
Server
push
pull
peer
peer
push
pull
pull
push
peer
10 MB
5 MB
+10 – 742
dev-1
hello.js
+10
metadata
code
local Repository
dev 1
dev 3
dev2

What is Git?

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





Git

Overview

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

A little bit history about Git

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

Characteristics

- Strong support for non-linear development : branch
- Distributed development
- Compatibility with existent systems and protocols (http)
- Efficient handling of large projects
- Cryptographic authentication of history (SHA)
- Toolkit-based design
- Pluggable merge strategies

Advantages

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

Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

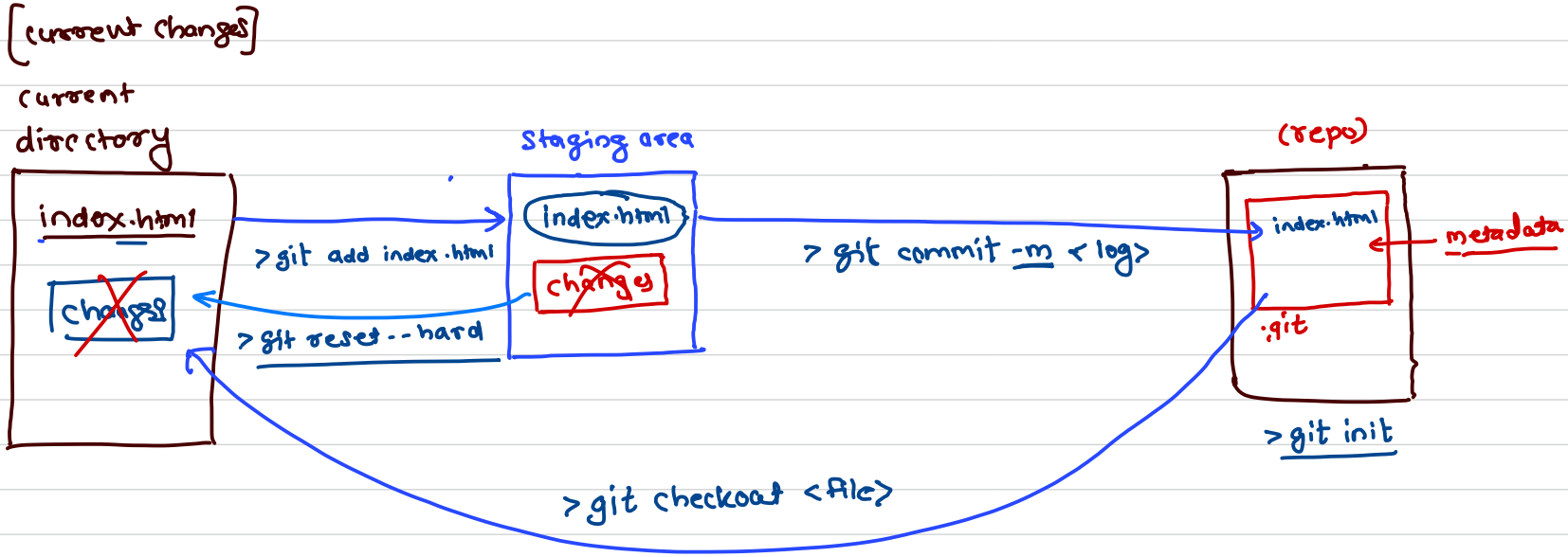
- **Setup global properties**

> git config --global user.name <user name>

> git config --global user.email <user email>

> git config --global core.editor <editor>

> git config –global merge.tool vimdiff


[current changes]
current directory
index.html
changes
Staging area
index.html
changes
(repo)
index.html
metadata
.git
> git add index.html
> git reset --hard
> git commit -m <log>
> git init
> git checkout <file>

Basic Commands

- **Initialize a repository**

  > git init

- **Checking status**

  > git status

- **Adding files to commit**

  > git add .

- **Committing the changes**

  > git commit –m ’<log message>’

Basic Commands

- **Checking logs**

  > git log

- **Checking difference**

  > git diff

- **Moving item**

  > git mv <source> <destination>

Terminologies

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

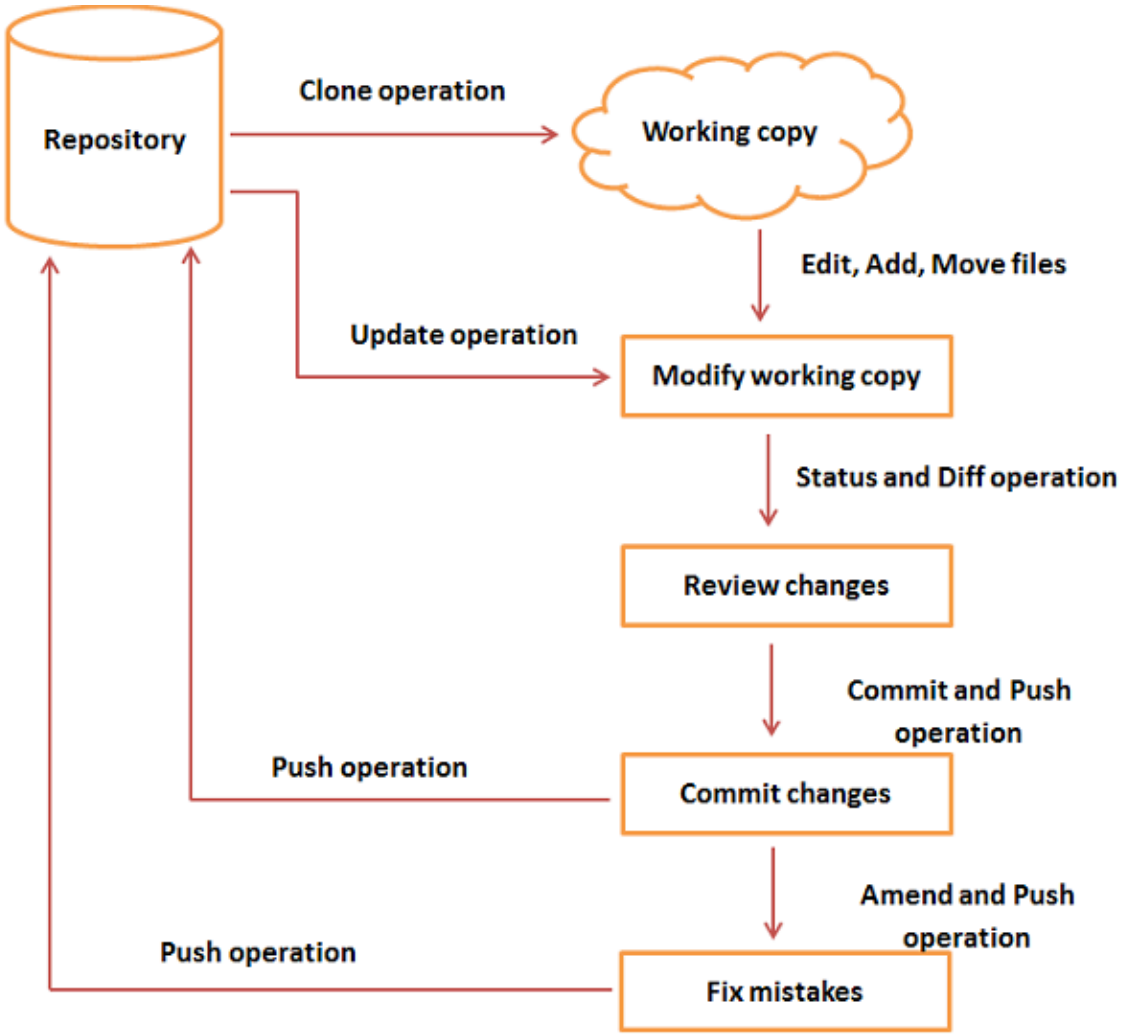
Terminologies

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

Terminologies

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

Life Cycle


Repository
Clone operation
Working copy
Edit, Add, Move files
Update operation
Modify working copy
Status and Diff operation
Review changes
Commit and Push operation
Push operation
Commit changes
Amend and Push operation
Push operation
Fix mistakes

Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

- **Setup global properties**

> git config --global user.name <user name>

> git config --global user.email <user email>

> git config --global core.editor <editor>

> git config –global merge.tool vimdiff

Basic Commands

- **Initialize a repository**

  > git init

- **Checking status**

  > git status

- **Adding files to commit**

  > git add .

- **Committing the changes**

  > git commit –m ’<log message>’

Basic Commands

- **Checking logs**

  > git log

- **Checking difference**

  > git diff

- **Moving item**

  > git mv <source> <destination>

Basic Commands

- **Rename item**

  > git mv <old> <new>

- **Delete Item**

  > git rm <item>

- **Remove unwanted changes**

  > git checkout file

Branch

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

Why is it required ?

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

Branch management commands

- **Create a branch**

  > git branch <branch name>

- **Checkout a branch**

  > git checkout <branch name>

- **Merge a branch**

  > git merge <branch name>

- **Delete a branch**

  > git branch –d <branch name>

GitHub

Overview

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

Workflow

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

Workflow commands

- **Add remote repository**

  > git remote add <name> <url>

- **Clone remote repository**

  > git clone <url>

- **Push the changes**

  > git push <name> <branch>

- **Pull the changes**

  > git pull

Source Code Management

scm / vcs

Why do we need Version Control System?

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

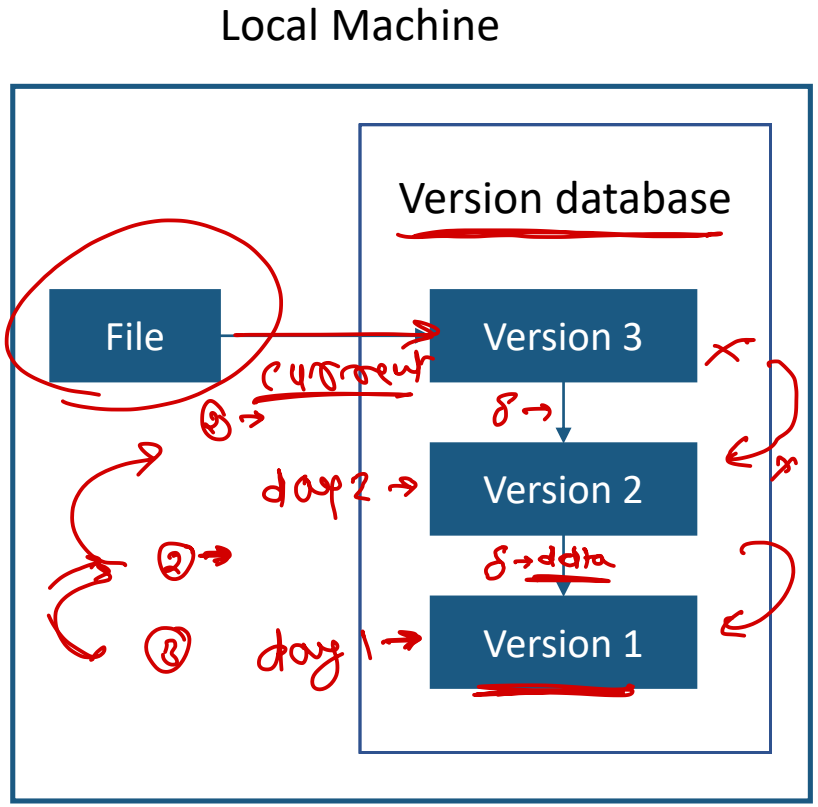
What is Version Control System?

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

Local Version Control System

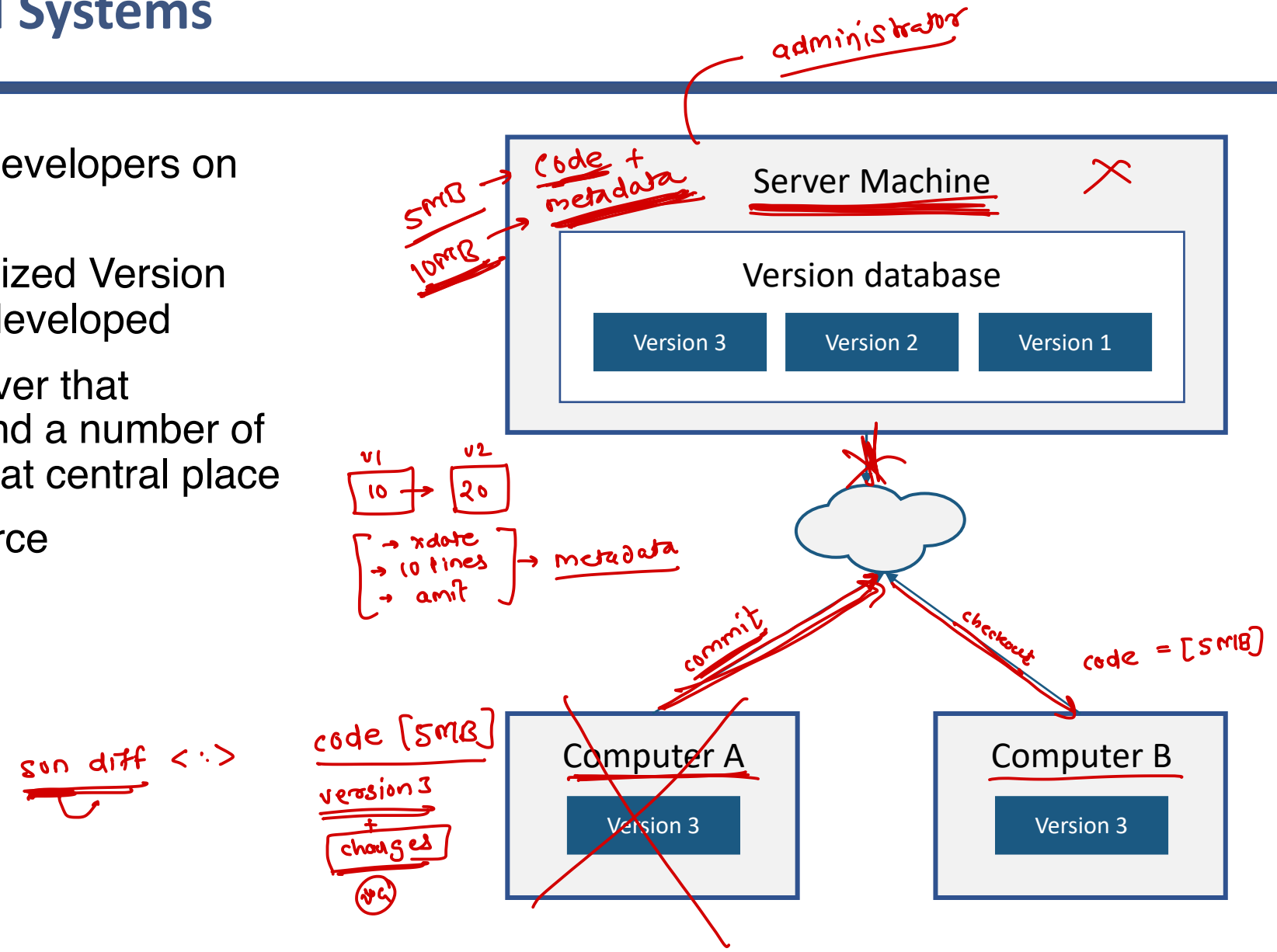
- Contains simple database that kept all the changes to files under revision control
- One of the more popular VCS tools was a system called RCS
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk
- It can then re-create what any file looked like at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

update
- add
- modify
- delete



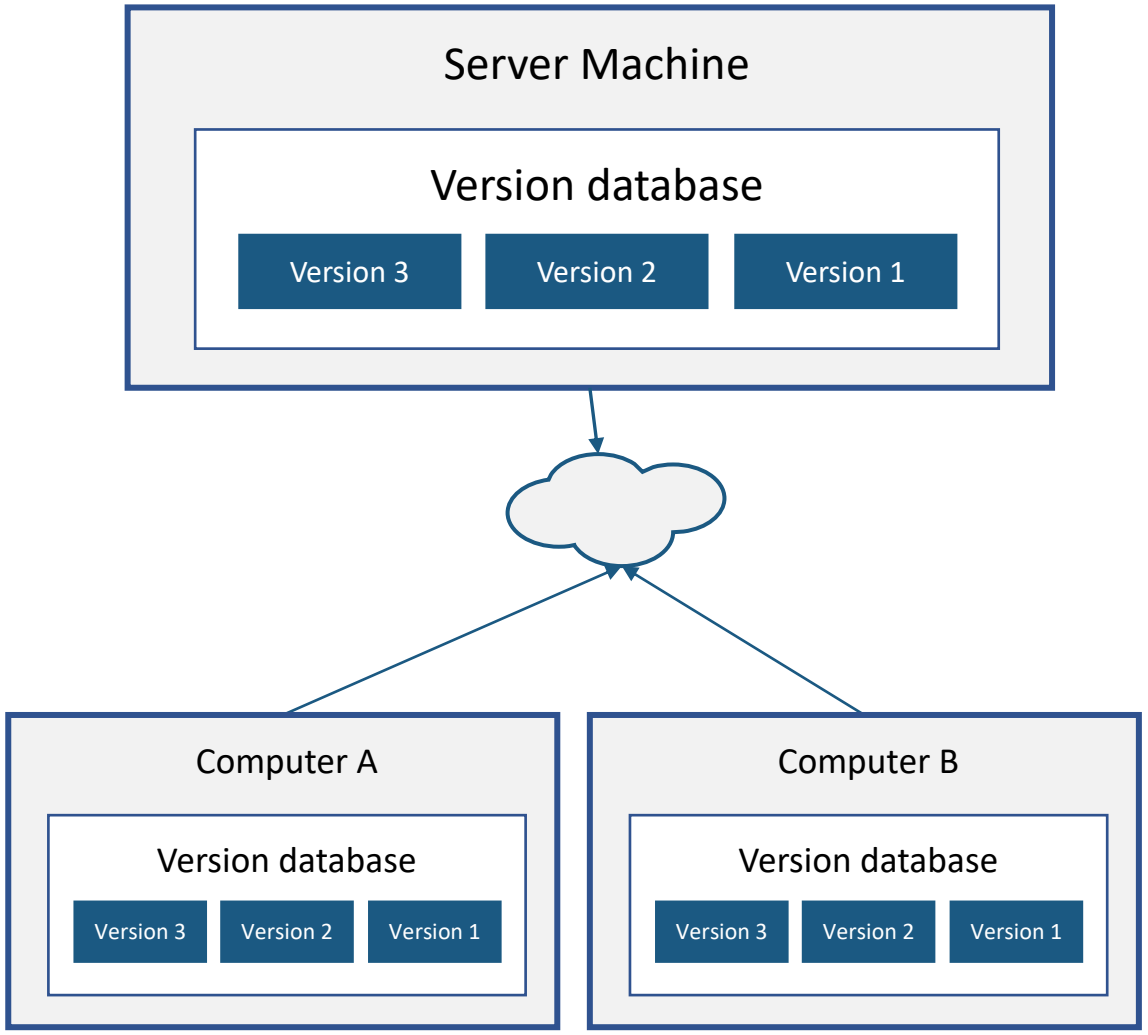
Centralized Version Control Systems

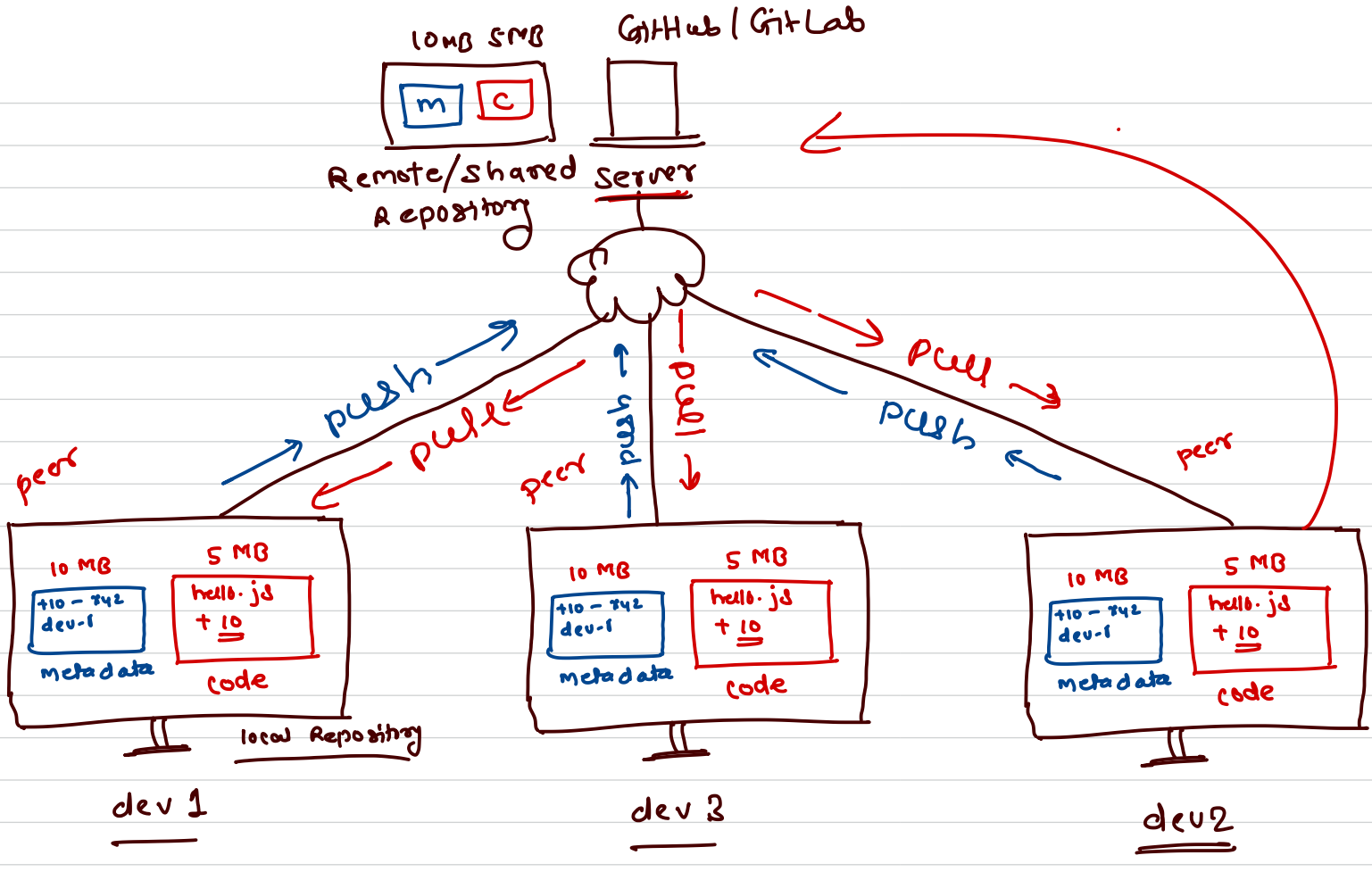
- People need to collaborate with developers on other systems
- To deal with this problem, Centralized Version Control Systems (CVCSs) were developed
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place
- E.g.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



Distributed Version Control Systems

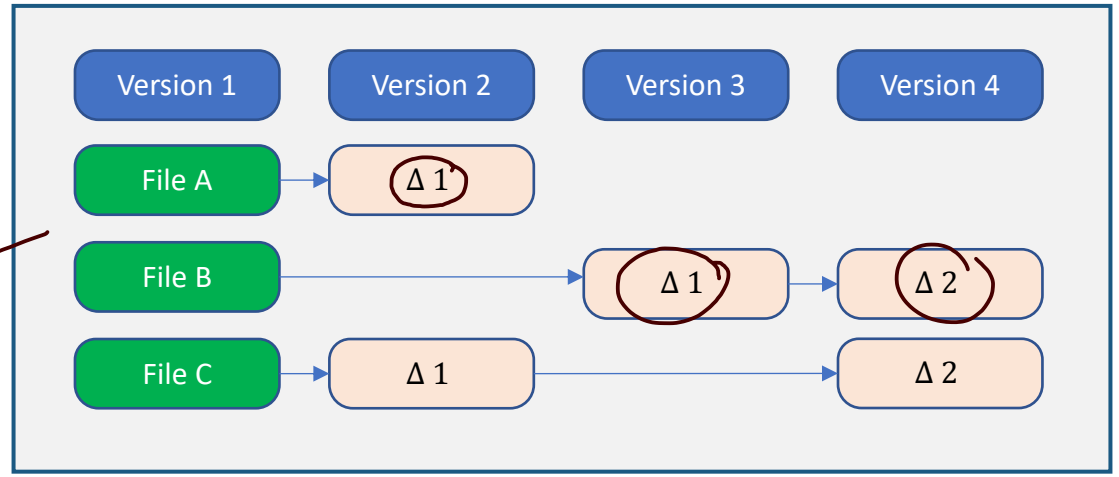
- Clients don't just check out the latest snapshot of the files, rather they fully mirror the repository
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it
- Every checkout is really a full backup of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

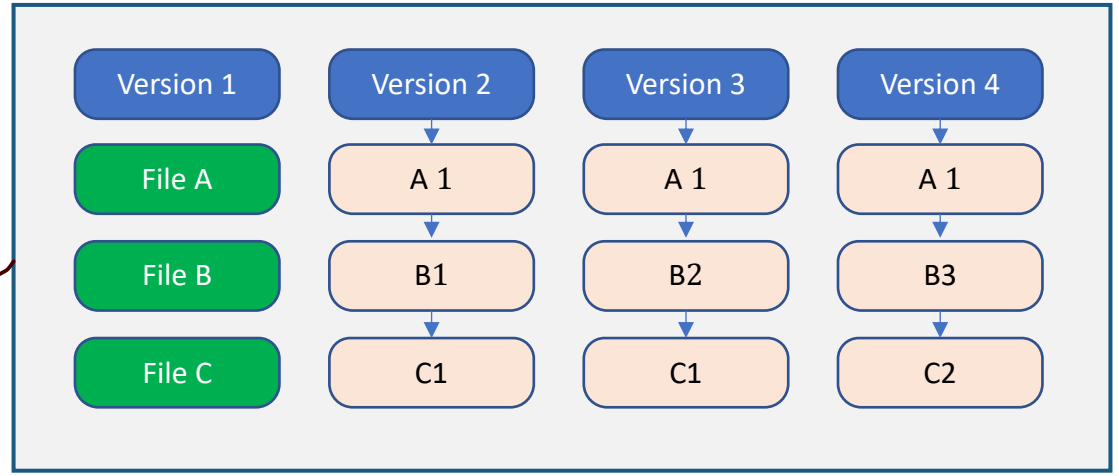



10MB SMB
m
c
GitHub / GitLab
Remote/shared Repository
Server
push
pull
peer
peer
push
pull
pull
push
peer
10 MB
5 MB
+10 – 742
dev-1
hello.js
+10
metadata
code
local Repository
dev 1
dev 3
dev2

What is Git?

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





Git

Overview

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

A little bit history about Git

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

Characteristics

- Strong support for non-linear development : branch
- Distributed development
- Compatibility with existent systems and protocols (http)
- Efficient handling of large projects
- Cryptographic authentication of history (SHA)
- Toolkit-based design
- Pluggable merge strategies

Advantages

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

Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

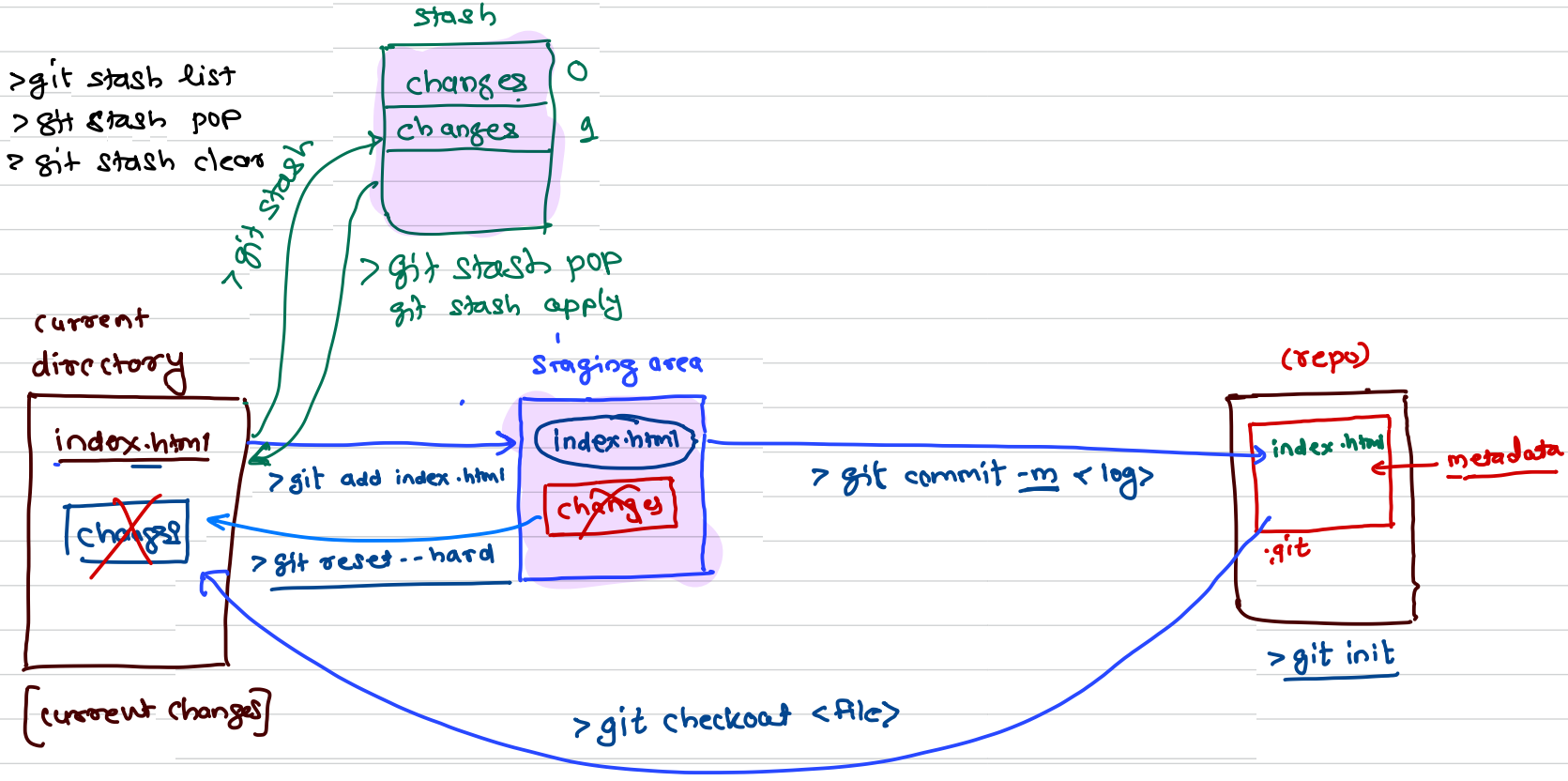
- **Setup global properties**

> git config --global user.name <user name>

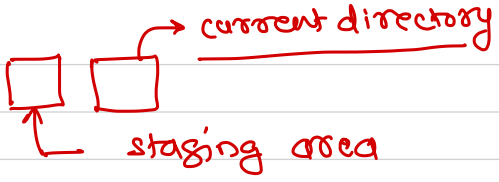
> git config --global user.email <user email>

> git config --global core.editor <editor>

> git config –global merge.tool vimdiff


stash
changes
changes
0
1
>git stash list
>git stash pop
>git stash clear
>git stash
>git stash pop
git stash apply
current
directory
index.html
changes
[current changes]
Staging area
index.html
changes
>git add index.html
>git reset --hard
>git commit -m <log>
(repo)
index.html
.git
metadata
>git init
>git checkout <file>

Status



?	?	→ untracked
A		→ added to the stage area
	M	→ modified in current directory
M		→ modified and added to stage area
U	U	→ file has conflicts

Basic Commands

- **Initialize a repository**

  > git init

- **Checking status**

  > git status

- **Adding files to commit**

  > git add .

- **Committing the changes**

  > git commit –m ’<log message>’

Basic Commands

- **Checking logs**

  > git log

- **Checking difference**

  > git diff

- **Moving item**

  > git mv <source> <destination>

Terminologies

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

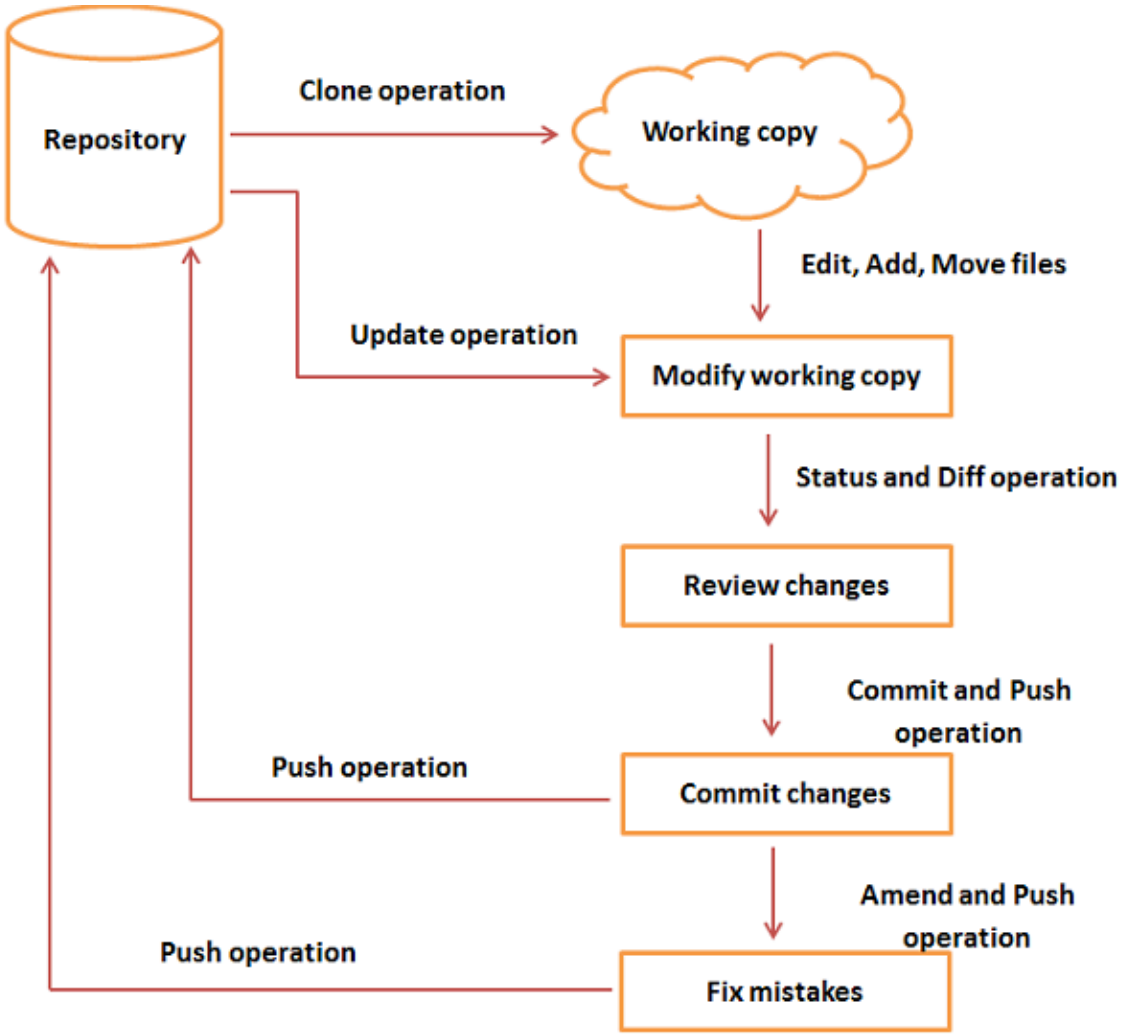
Terminologies

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

Terminologies

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

Life Cycle


Repository
Clone operation
Working copy
Edit, Add, Move files
Update operation
Modify working copy
Status and Diff operation
Review changes
Commit and Push operation
Push operation
Commit changes
Amend and Push operation
Push operation
Fix mistakes

Installation and first time setup

- **Install git on ubuntu**

> sudo apt-get install git

- **List the global settings**

> git config --global --list

- **Setup global properties**

> git config --global user.name <user name>

> git config --global user.email <user email>

> git config --global core.editor <editor>

> git config –global merge.tool vimdiff

Basic Commands

- **Initialize a repository**

  > git init

- **Checking status**

  > git status

- **Adding files to commit**

  > git add .

- **Committing the changes**

  > git commit –m ’<log message>’

Basic Commands

- **Checking logs**

  > git log

- **Checking difference**

  > git diff

- **Moving item**

  > git mv <source> <destination>

Basic Commands

- **Rename item**

  > git mv <old> <new>

- **Delete Item**

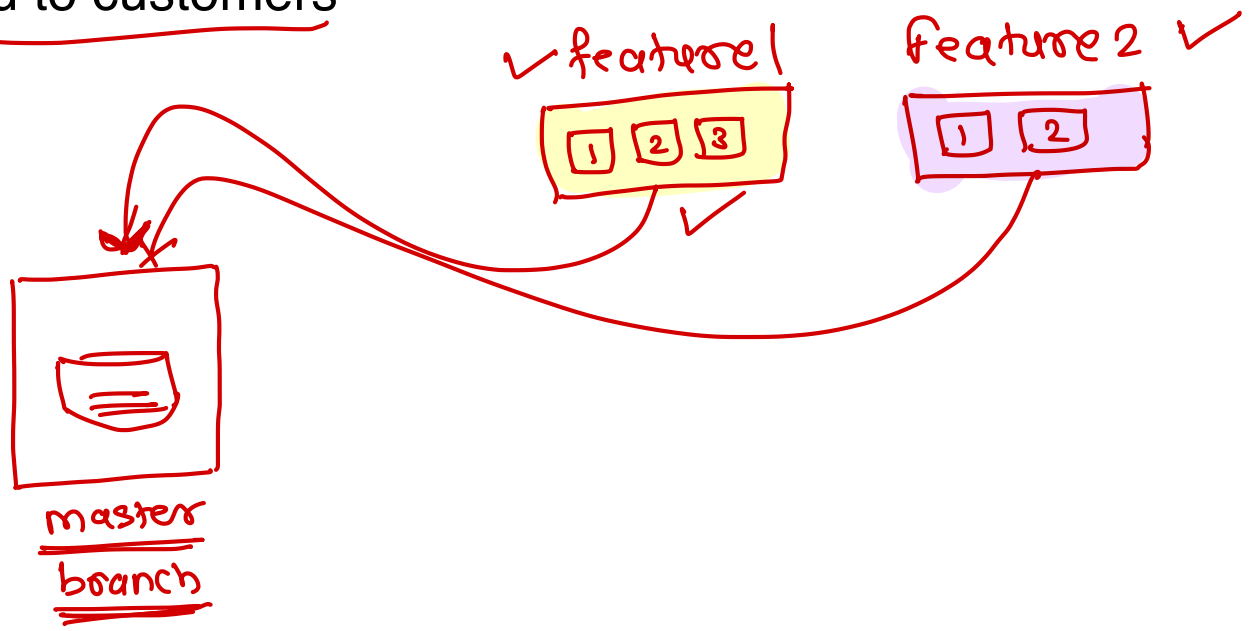
  > git rm <item>

- **Remove unwanted changes**

  > git checkout file

Branch → master

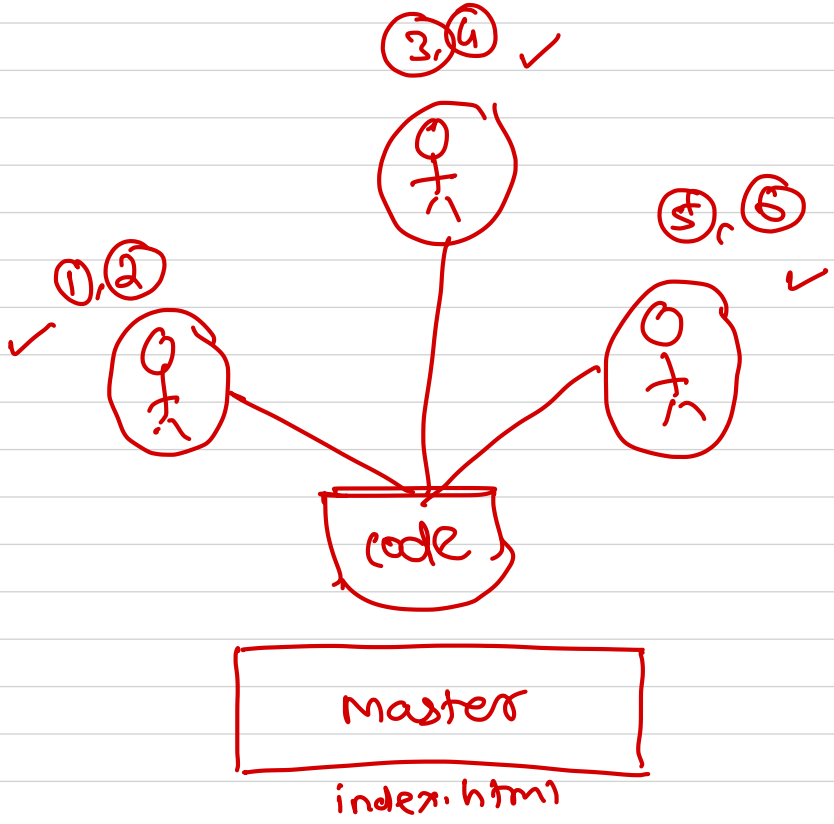
- Allows another line of development
- A way to write code without affecting the rest of your team
- Generally used for feature development
- Once confirmed the feature is working you can merge the branch in the master branch and release the build to customers



master
① latest changes
② stable
③ non-buggy ?
④ non-crashing

Sunbeam website - Features

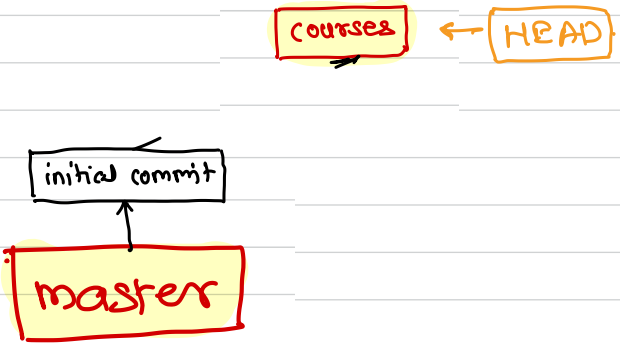
1. courses
2. students
3. Faculties
4. Feedbacks
5. about-us
6. contact-us

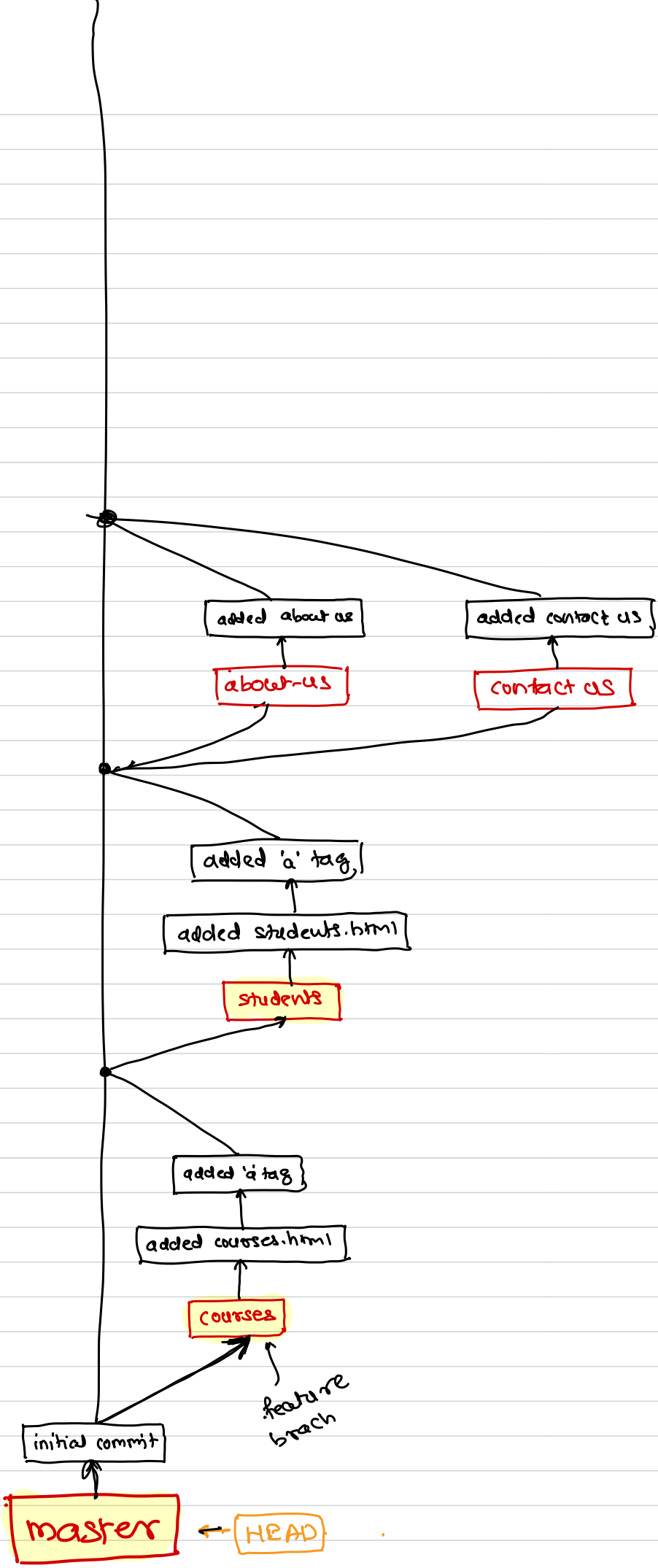


> git branch courses

> git brach

> git checkout courses




added about us
added contact us
about-us
contact us
added 'a' tag
added students.html
students
added 'a' tag
added courses.html
courses
feature branch
initial commit
master
HEAD

Why is it required ?

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

Branch management commands

- **Create a branch**

  > git branch <branch name>

- **Checkout a branch**

  > git checkout <branch name>

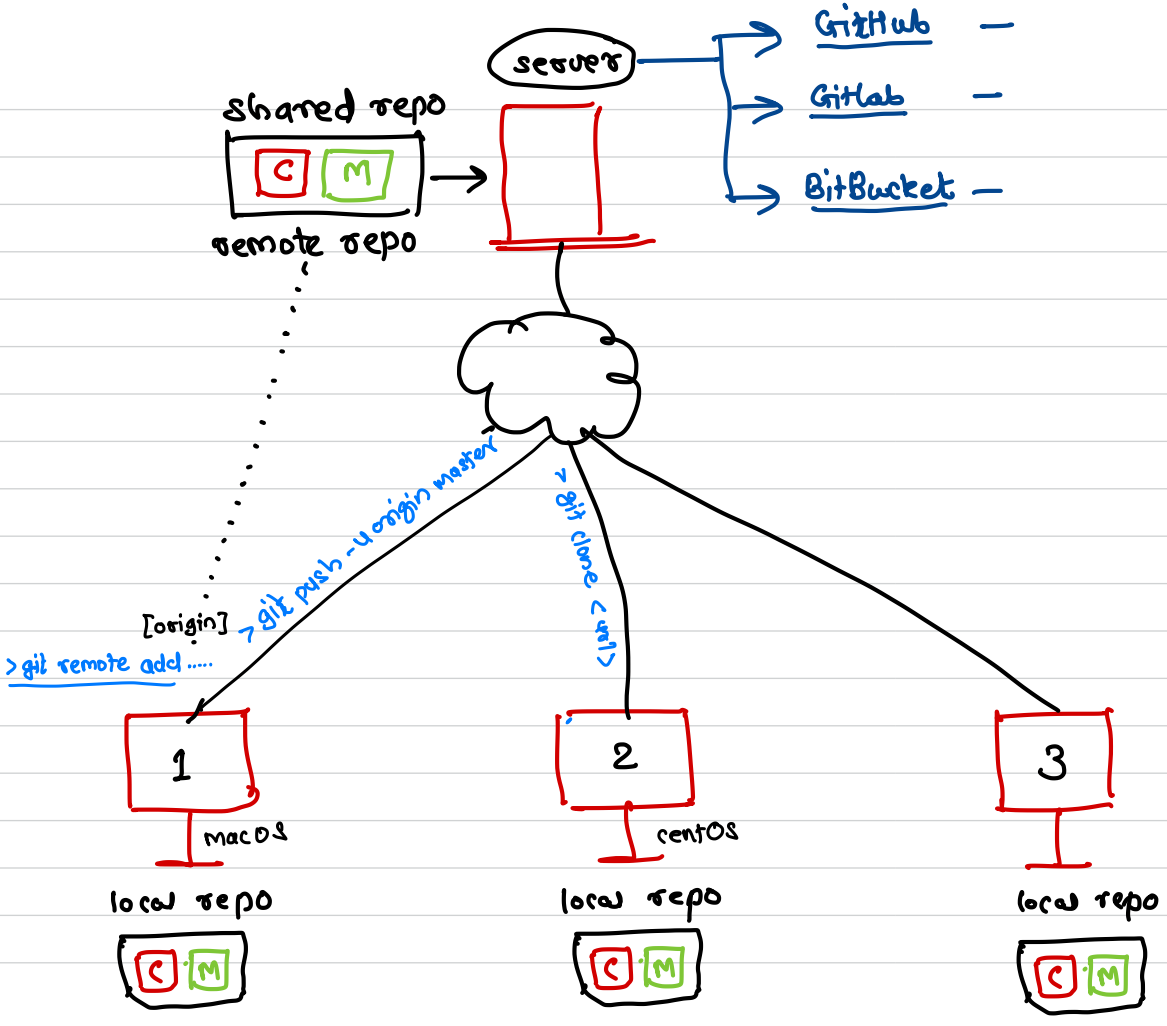
- **Merge a branch**

  > git merge <branch name>

- **Delete a branch**

  > git branch –d <branch name>

GitHub


server
GitHub
Gitlab
BitBucket
shared repo
C
M
remote repo
[origin]
> git remote add .....
> git push -u origin master
> git clone <url>
1
macOS
local repo
2
centOS
local repo
3
local repo

Overview

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

Workflow

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

Workflow commands

- **Add remote repository**

  > git remote add <name> <url>

- **Clone remote repository**

  > git clone <url>

- **Push the changes**

  > git push <name> <branch>

- **Pull the changes**

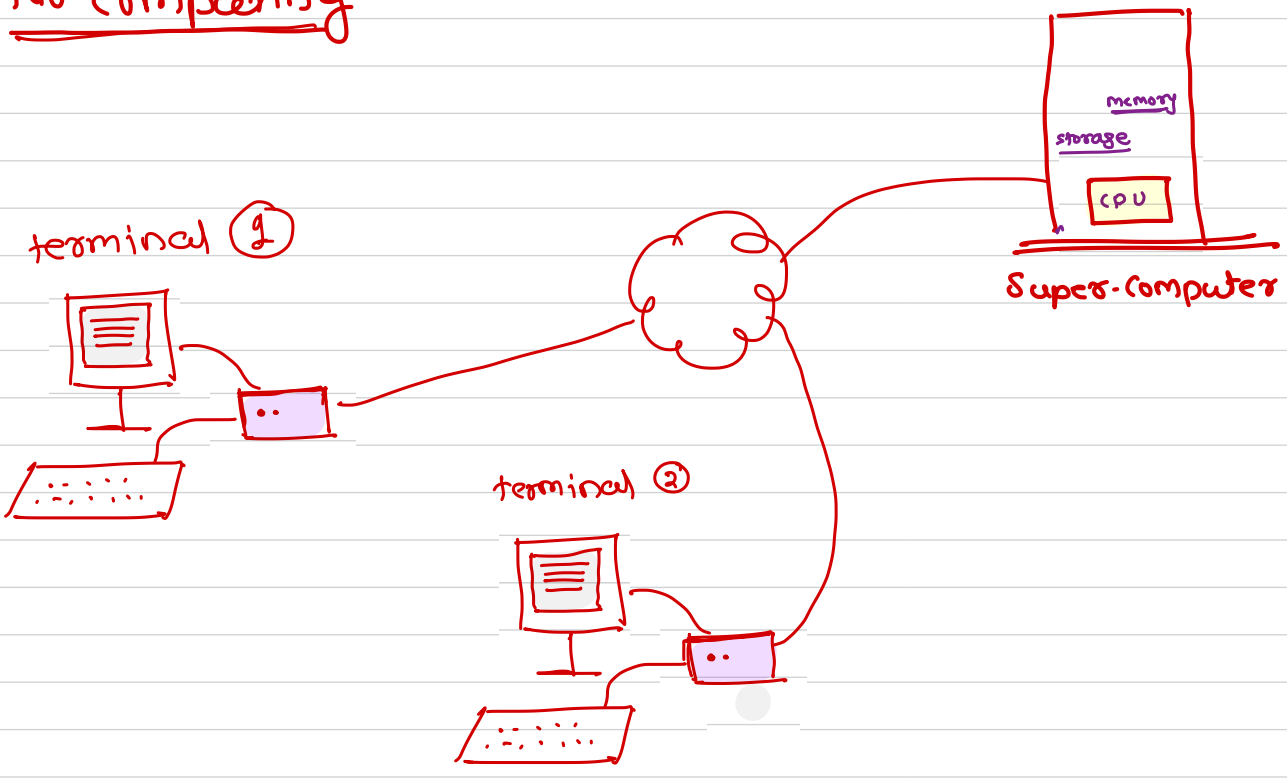
  > git pull

Cloud Computing

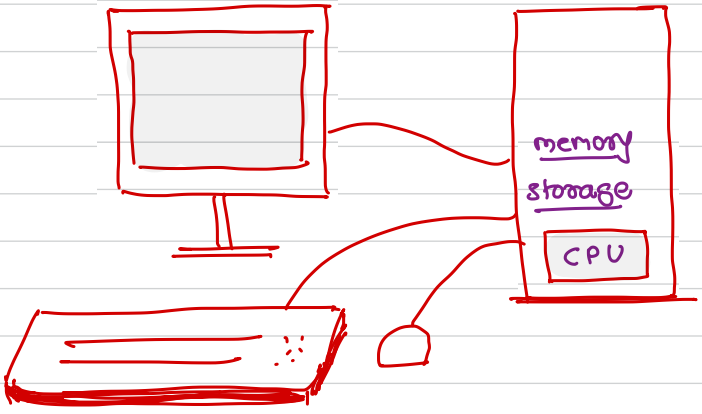
Computing Model

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

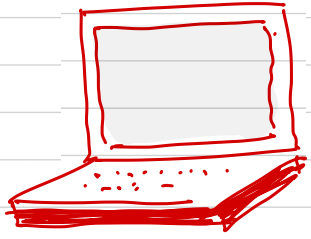
⑥ No competing



① Desktop computing

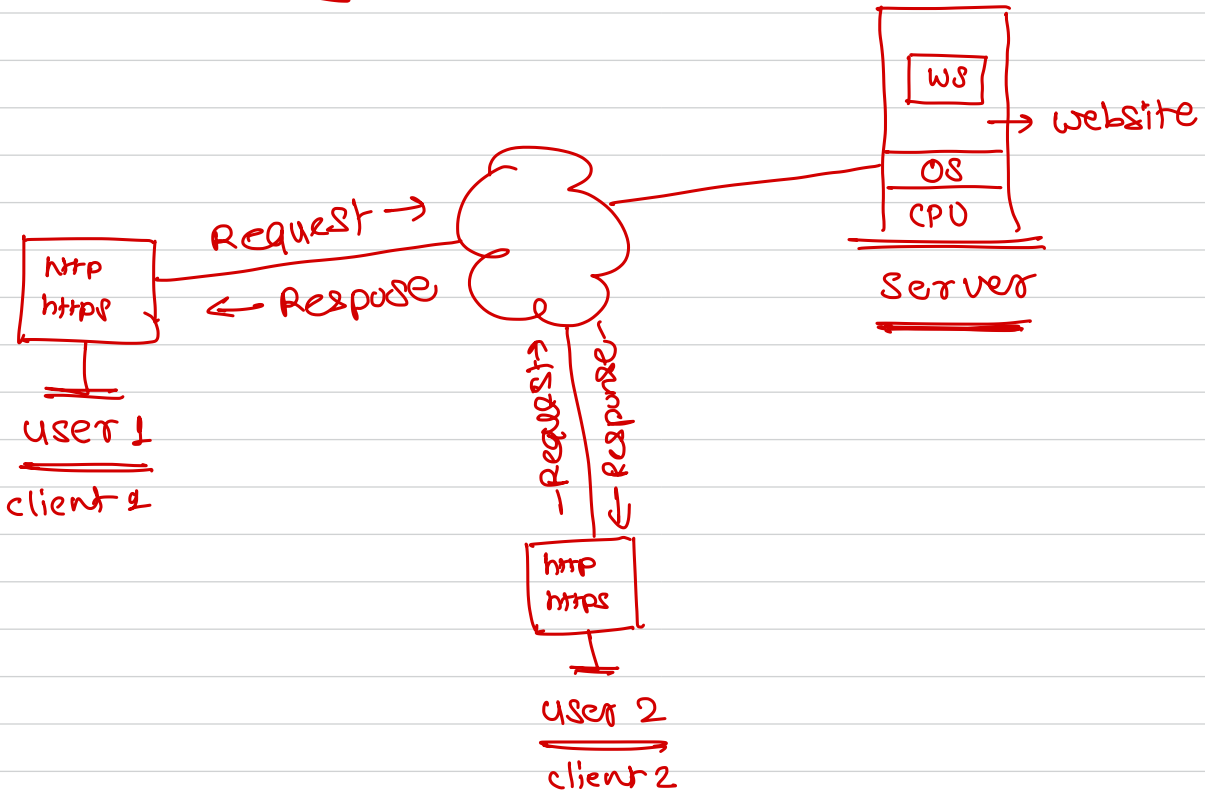


desktop

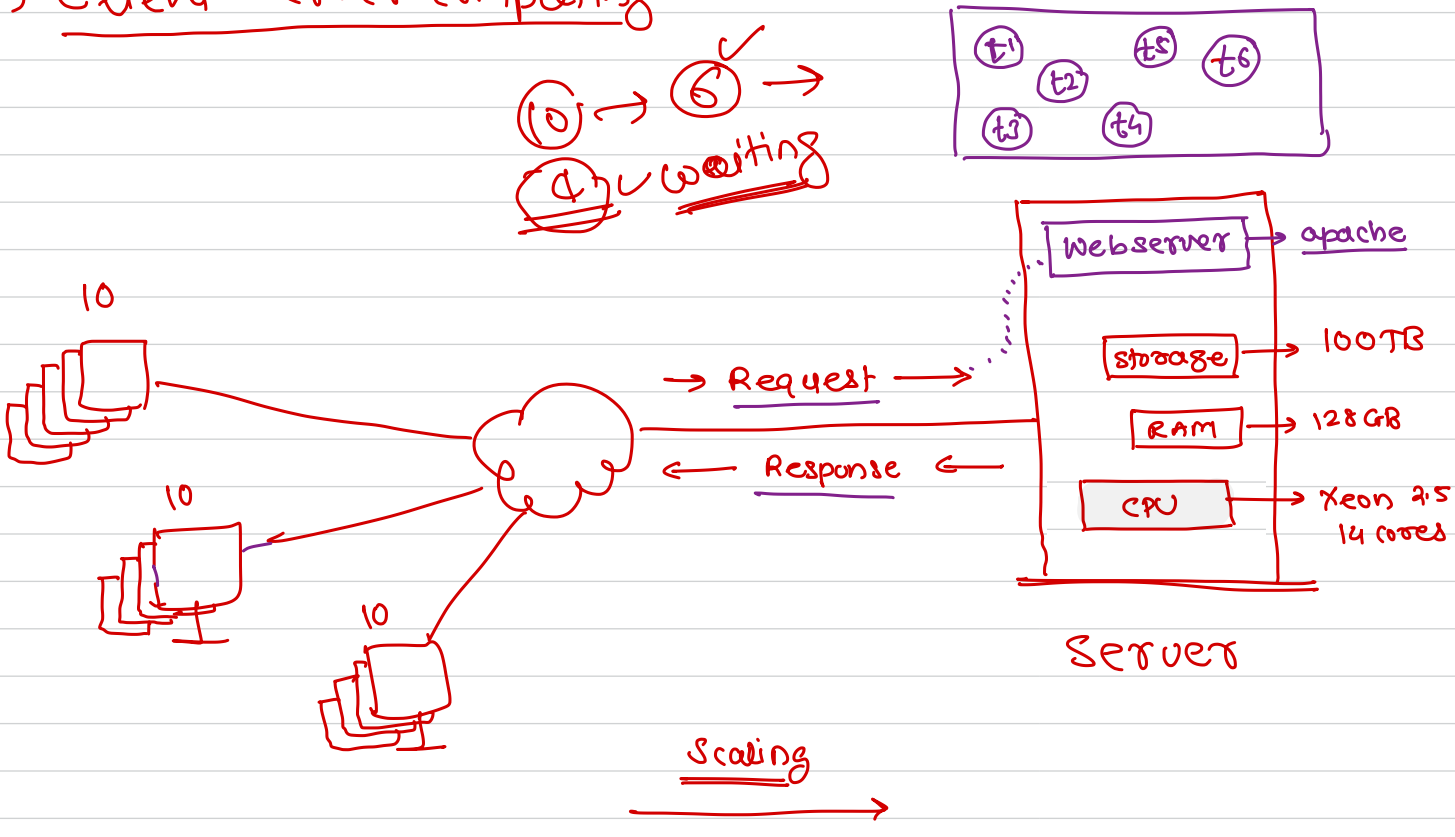


Laptop
(notebook (netbook)

2.1.) Client-Server computing



2.2) Client-Server computing

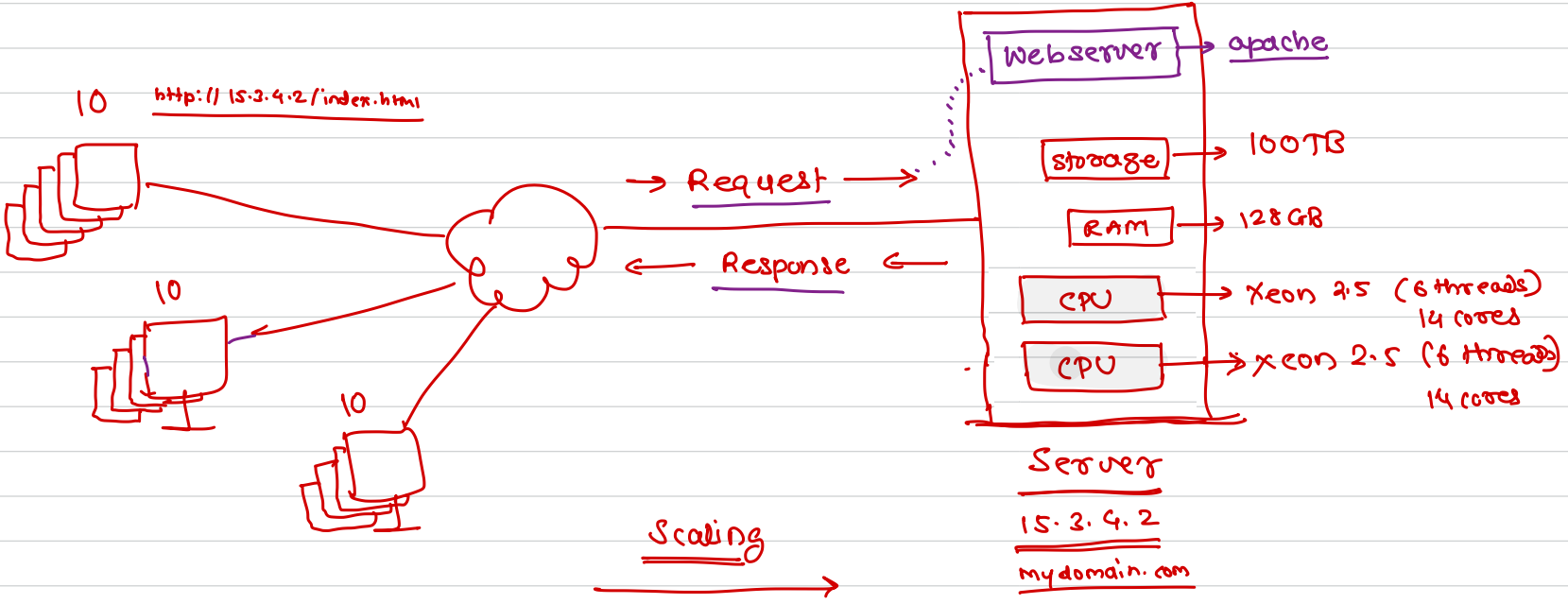


Scaling
- Vertical
- Horizontal

Vertical scaling

→ up → increase
→ down → decrease

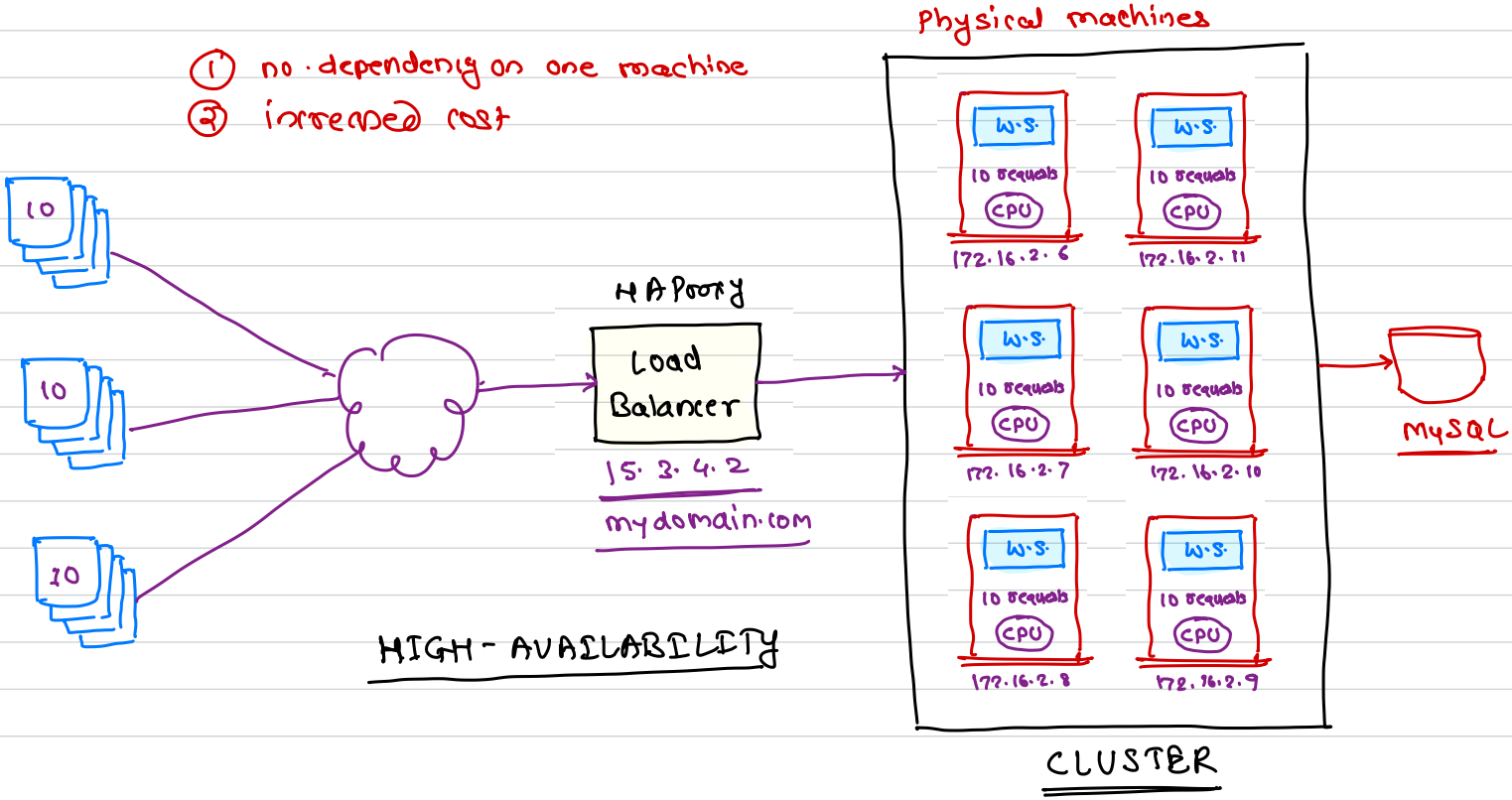
① dependency on one machine
② limit on the configuration



③ cluster computing
distributed computing

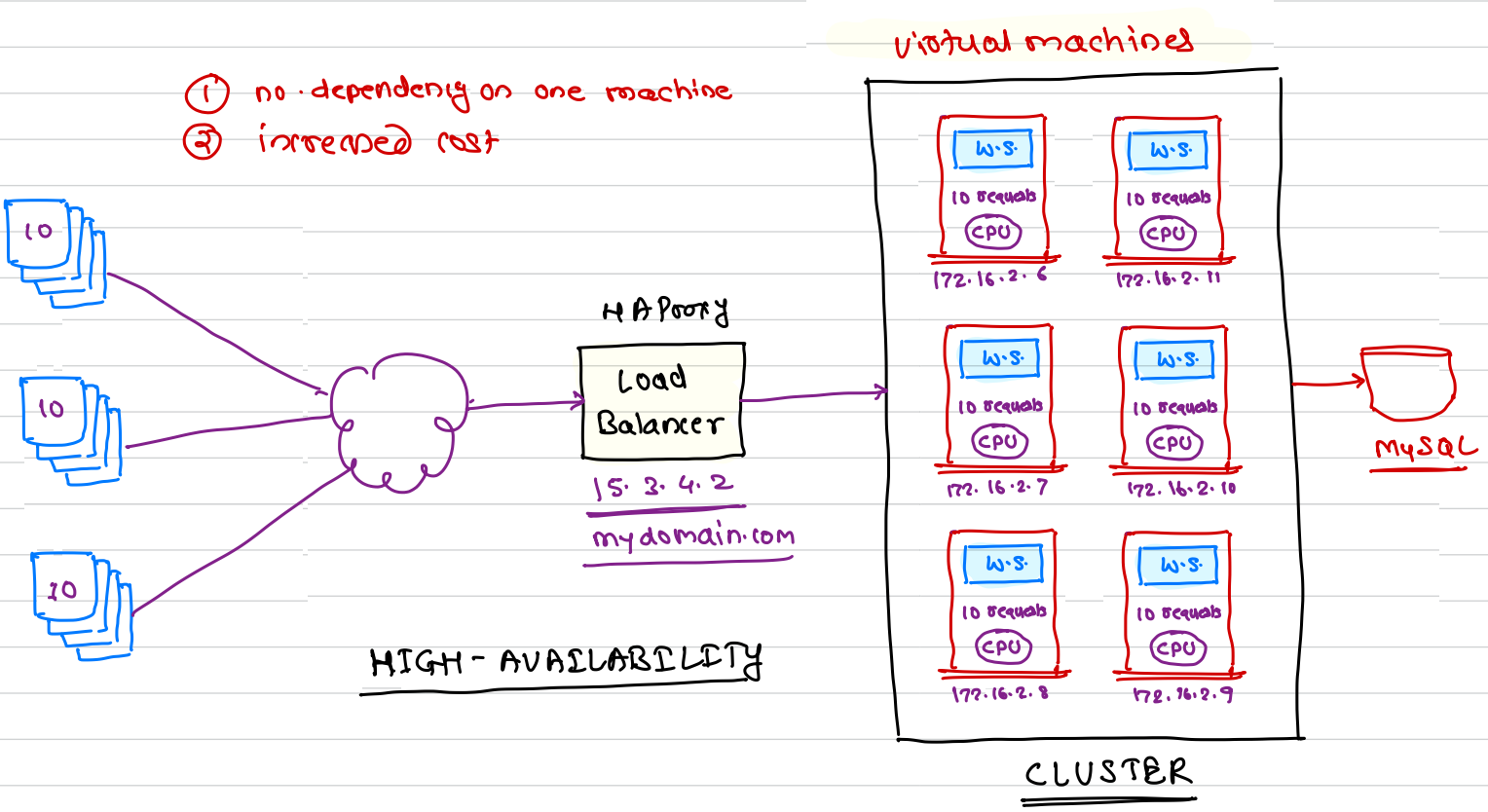
Horizontal scaling
→ out → adding more machines
→ in → remove machines

① no dependency on one machine
② increased cost



④ cloud computing

① no dependency on one machine
② increased cost



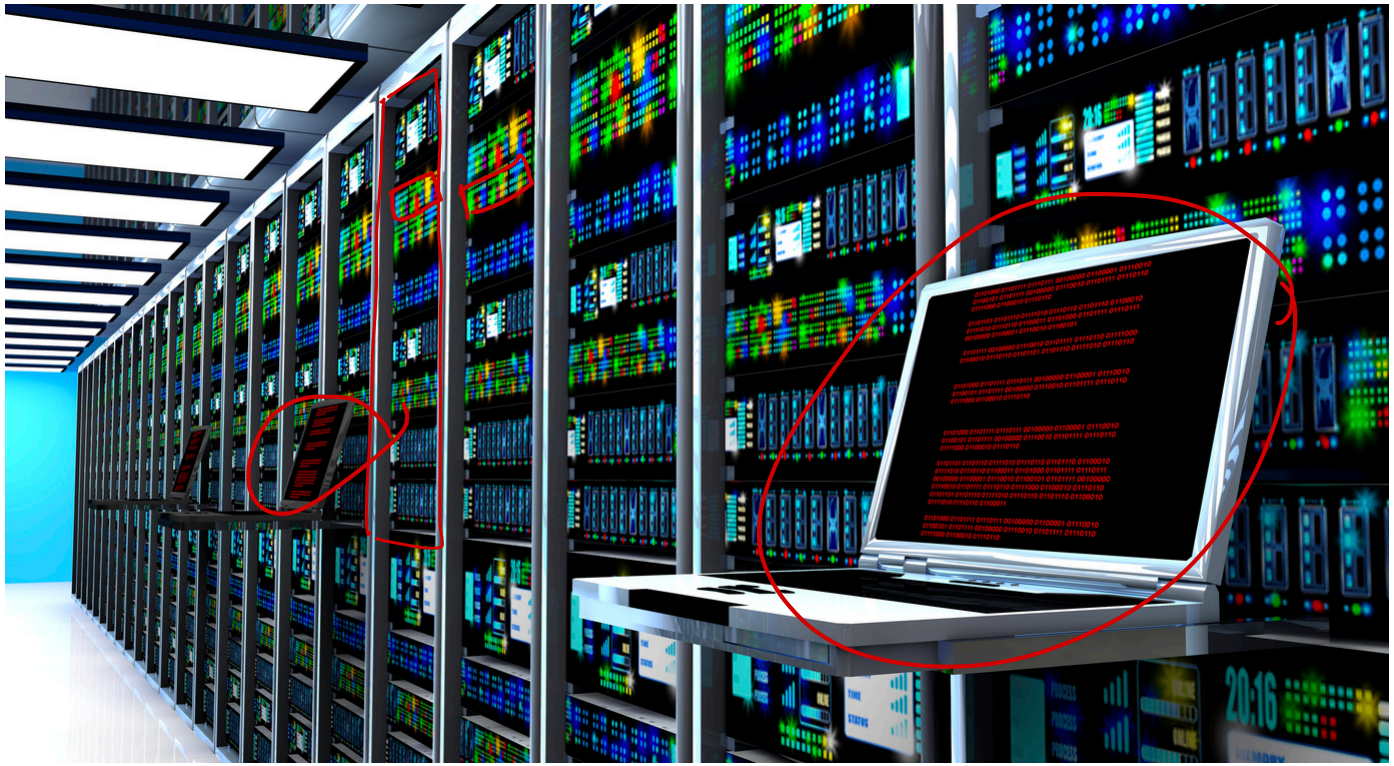
HIGH-AVAILABILITY

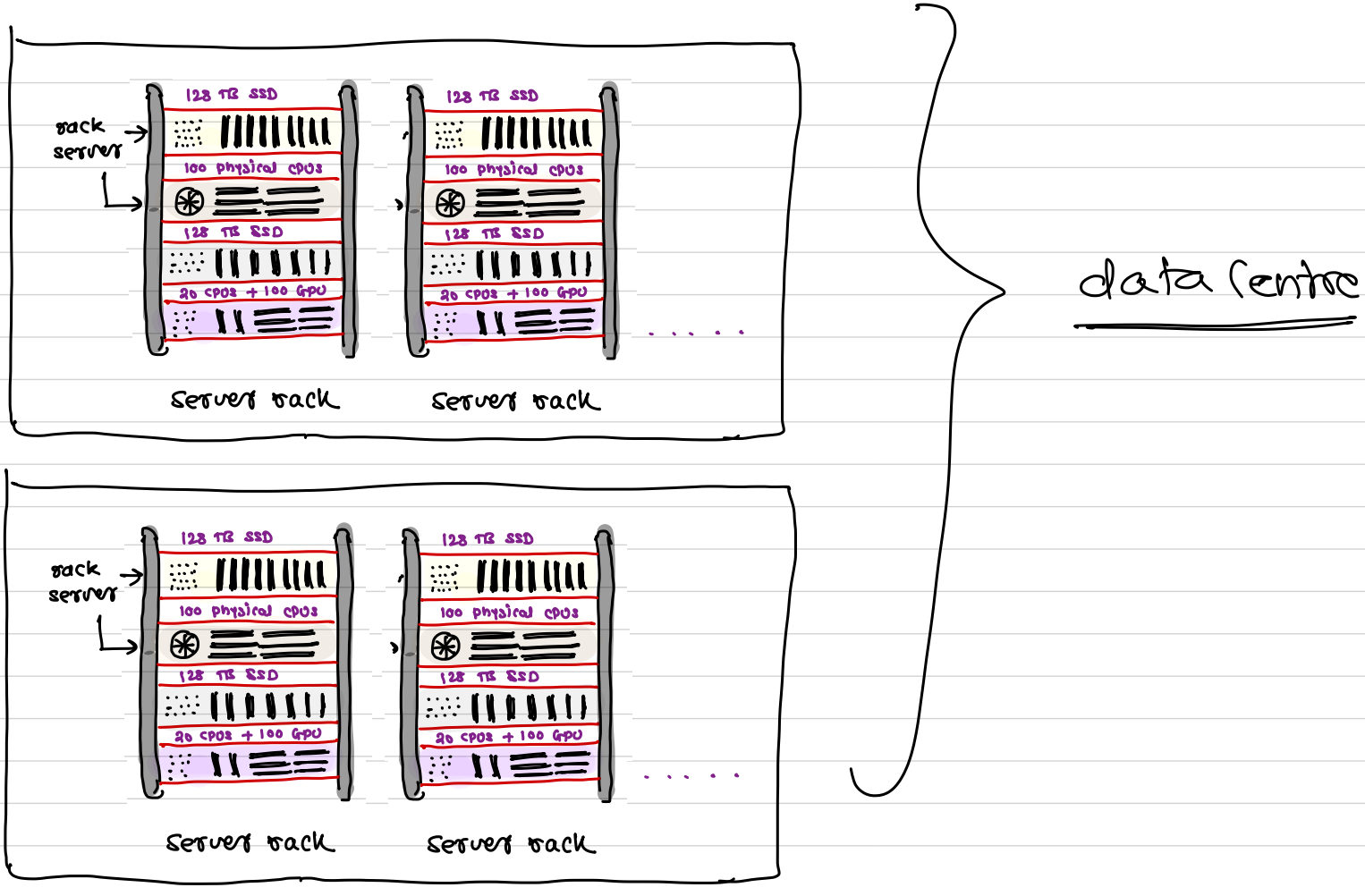
What is cloud computing ?

- 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

What is Data Center ?

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

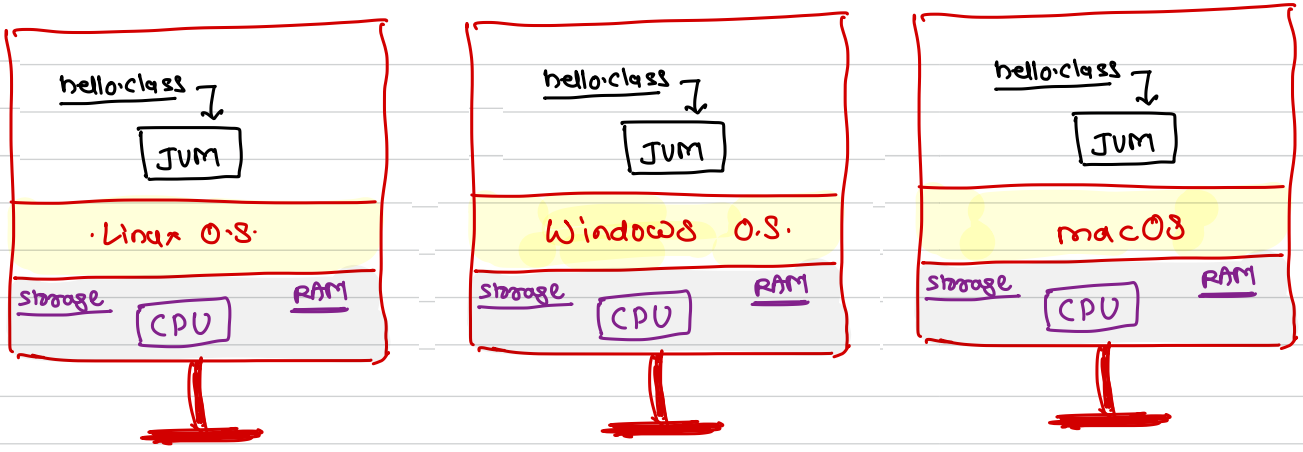



rack server
128 TB SSD
100 physical CPUs
128 TB SSD
20 CPUs + 100 GPU
Server rack
Server rack
rack server
128 TB SSD
100 physical CPUs
128 TB SSD
20 CPUs + 100 GPU
Server rack
Server rack
data centre

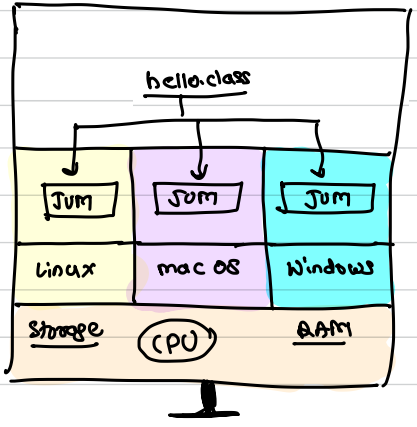
What is Virtualization ?

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

①



②



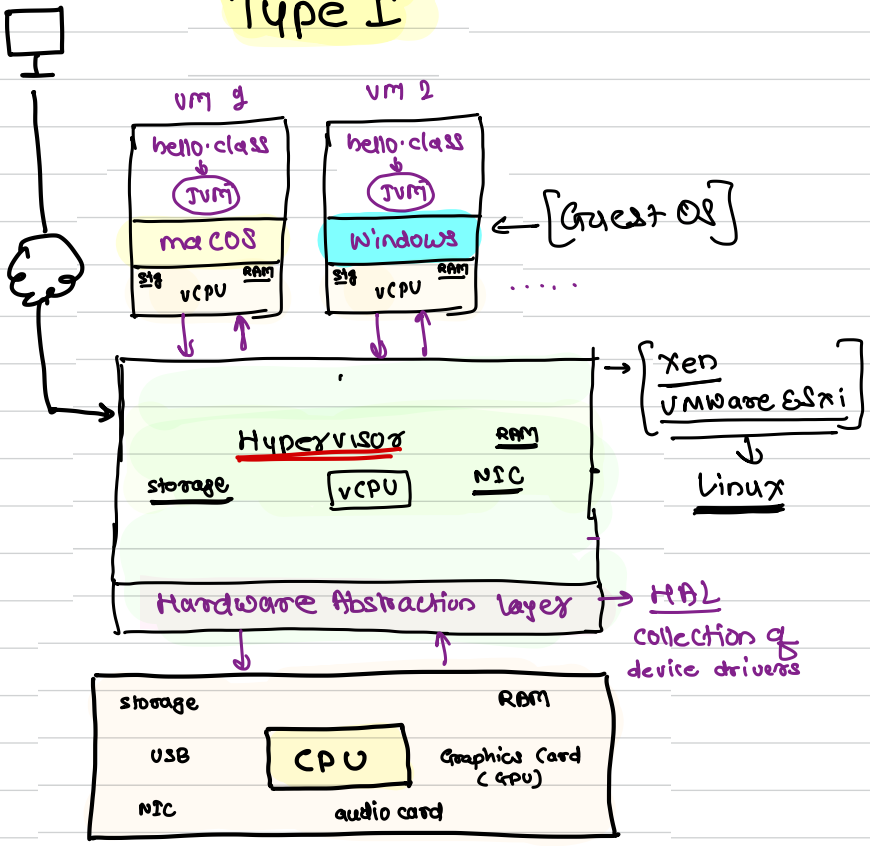
multibooting

cloud providers

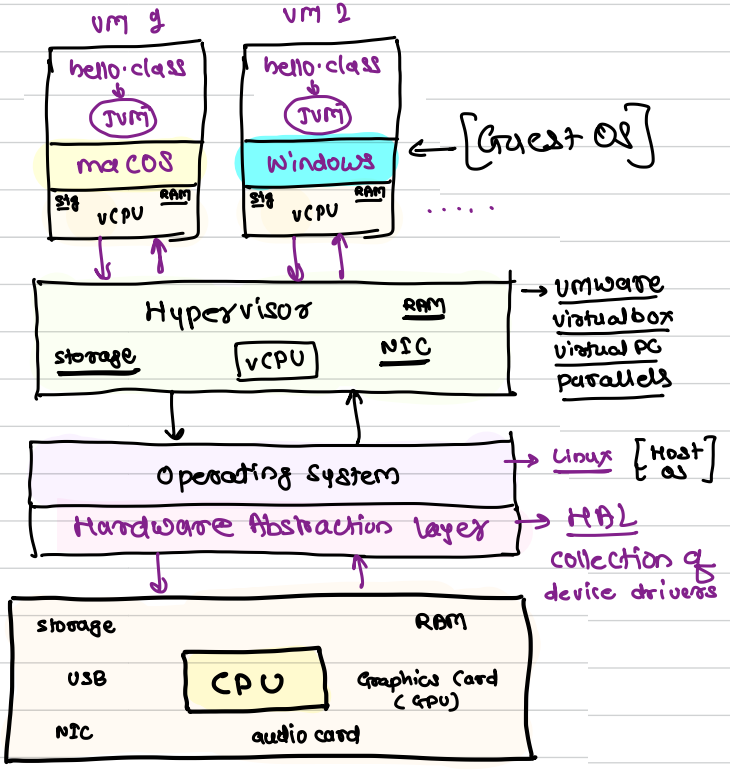
Virtualization

developers & testers, End Us

Type I

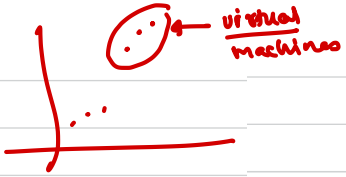


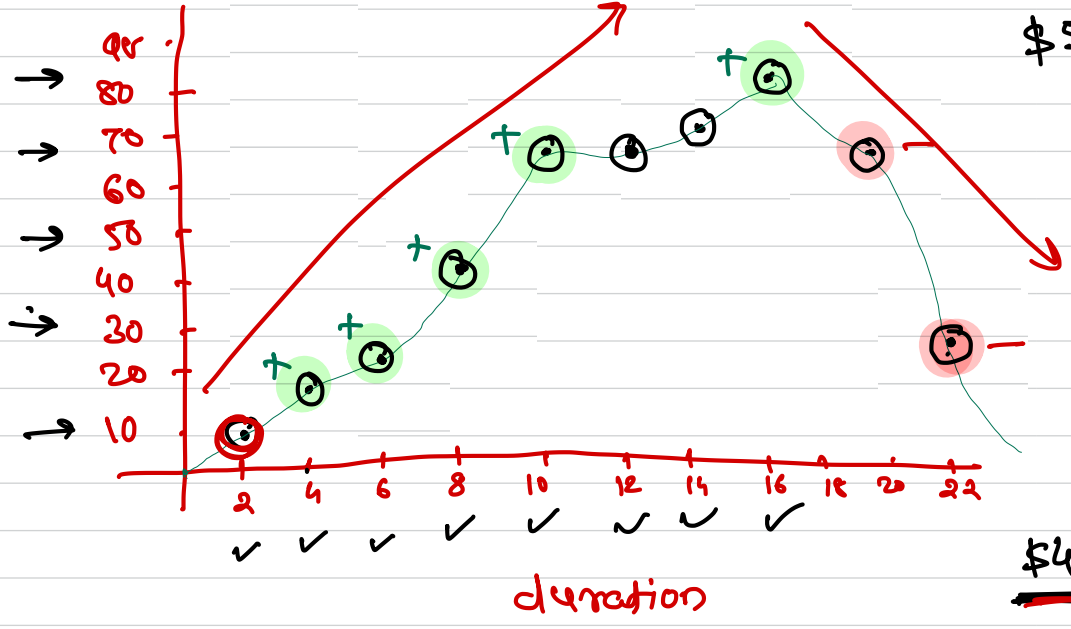
Type II



Terminologies

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





$5K Infra → (10K)

\+ 2x

3x 10K = 30K

\+ 5x

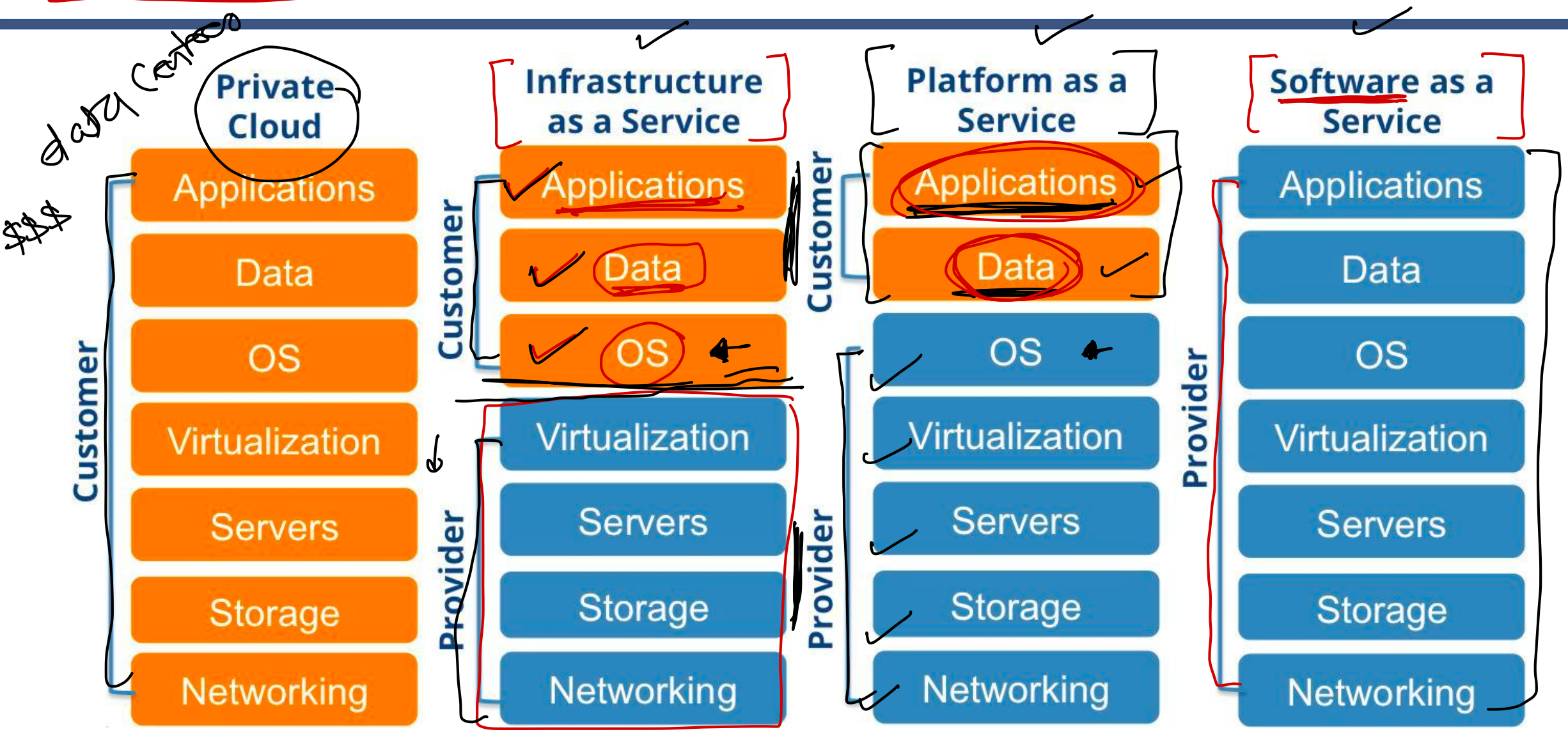
8x 10K = 80K

\+ 1x

9x 10K = 90K

$45K

Service Models



Service Models

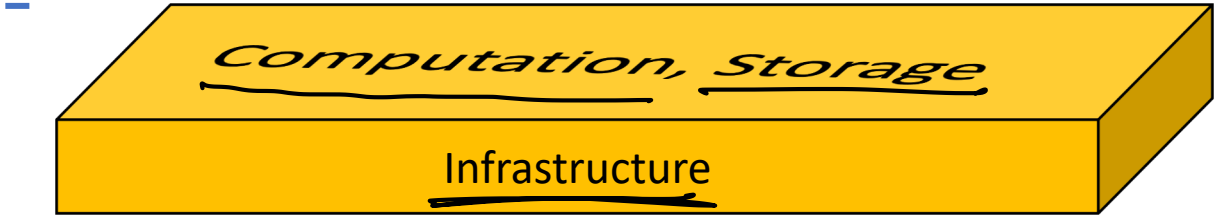
Software as a Service (SaaS)

Google Apps, Facebook, YouTube, Dropbox, Google Photos

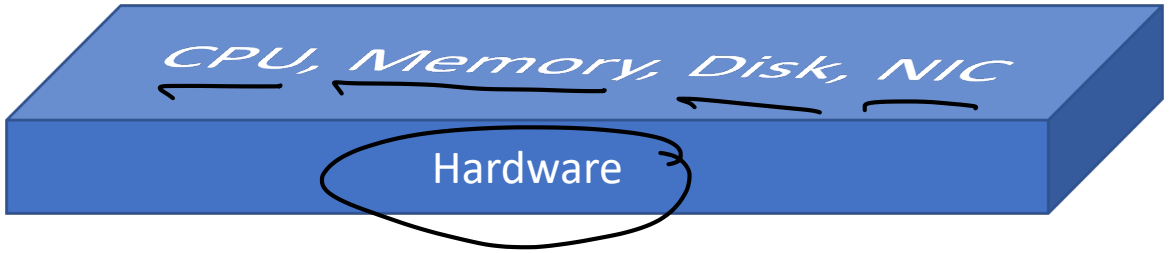
Platform as a Service (PaaS)

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

Infrastructure as a Service (IaaS)



Amazon EC2, Google Compute VM, Azure VM



Data Center

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

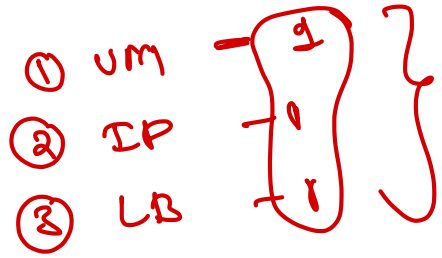
Service Models: SaaS

- 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 (automatic)
- Broad Network Access
- Location Independent Resource Sharing
- Measured Services ($$)

① VM
② IP
③ LB



Cloud Deployment Models: Public

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

Cloud Deployment Models: Private

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

Cloud Deployment Models: Hybrid

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

Cloud Services

- Compute: used to create the Virtual Machine
- 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

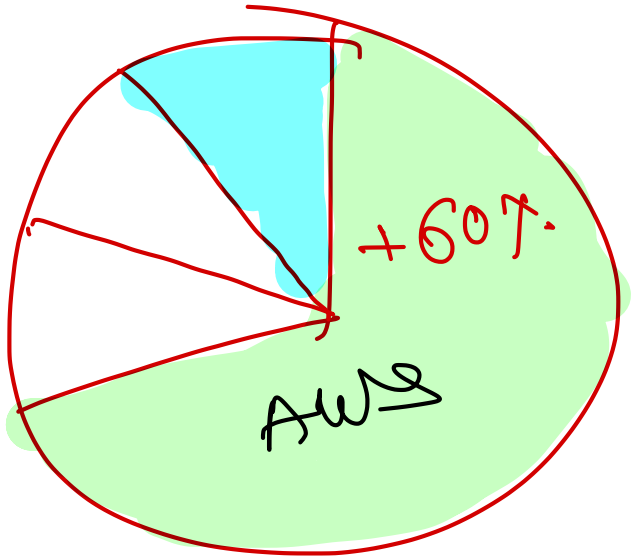
Disadvantages

- 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 provider specific

Cloud Providers

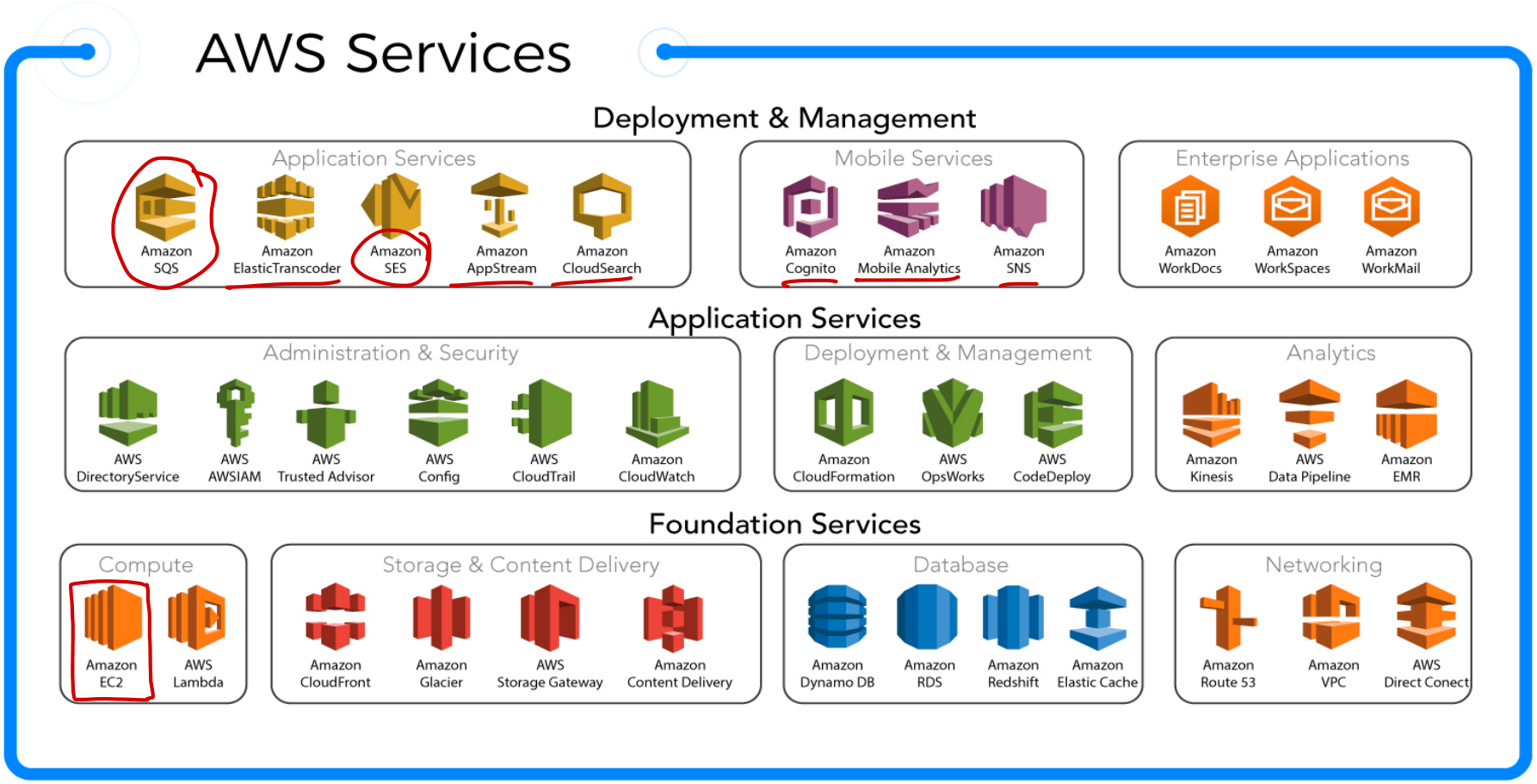
- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure
- Rackspace
- DigitalOcean
- Alibaba Cloud
- Oracle Cloud
- IBM Cloud



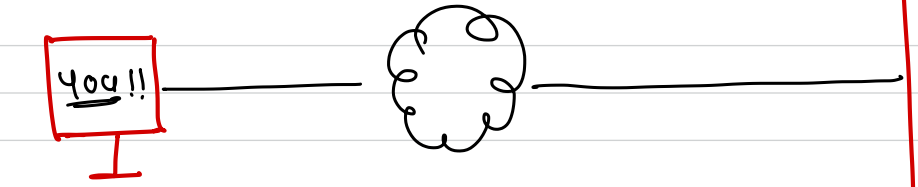
What is AWS ?

- 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

AWS Services

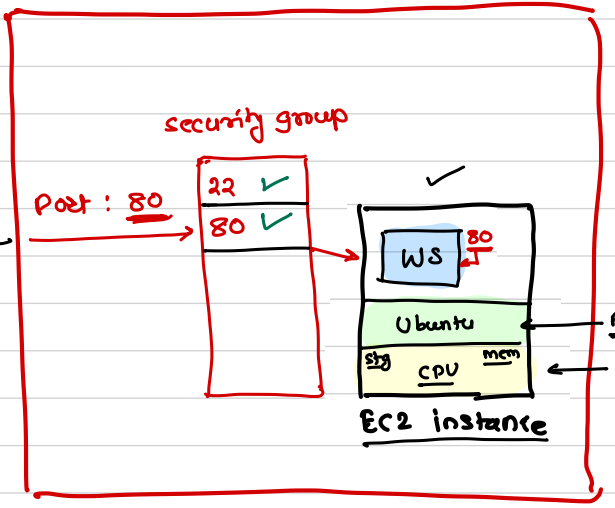


EC2 instance



> ssh -i <pem> ubuntu@<ip>

VPC : virtual Private cloud



Global Infrastructure: Region

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

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

Global Infrastructure: Edge Locations

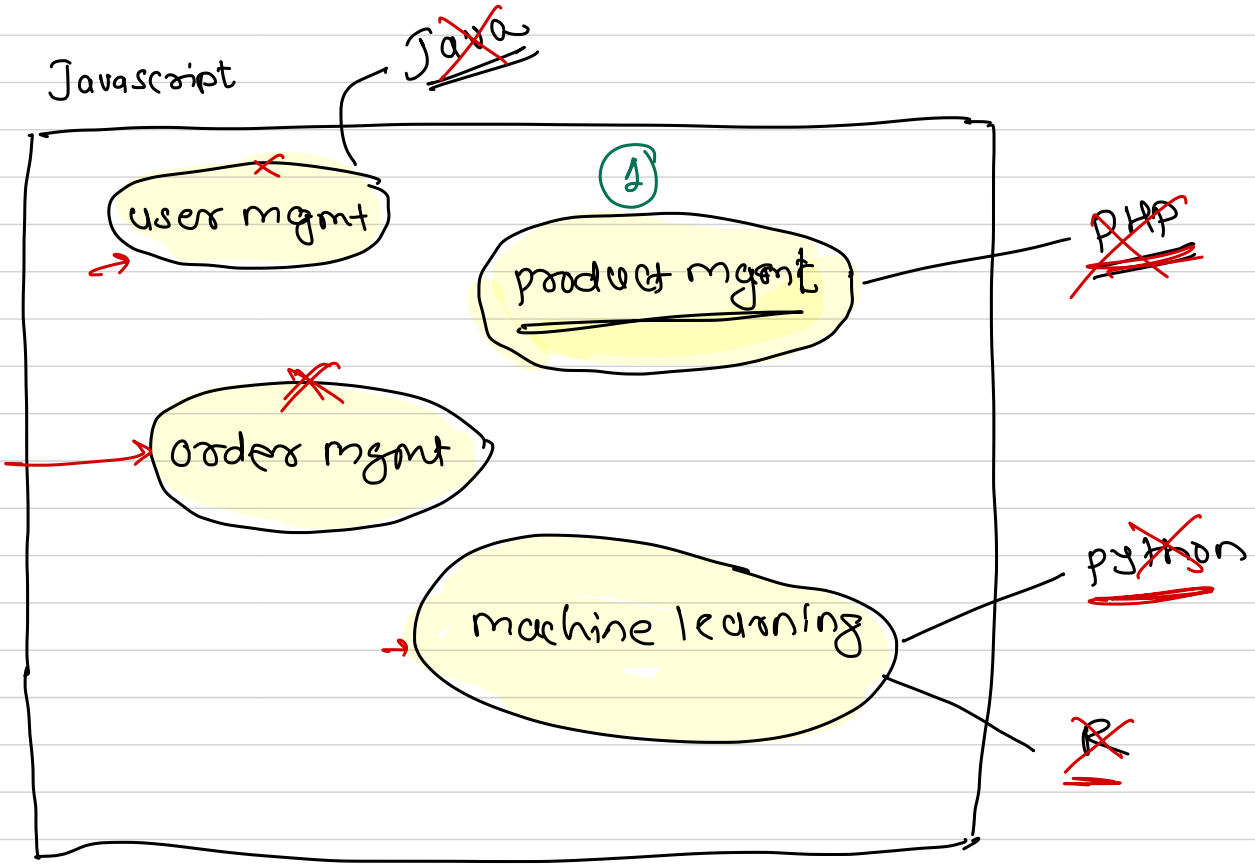
- Edge Locations are AWS sites deployed in major cities and highly populated areas across the globe
- Generally used to cache data and reduce latency for end-user access by using the Edge Locations as a global Content Delivery Network (CDN)
- Edge Locations are primarily used by end users who are accessing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

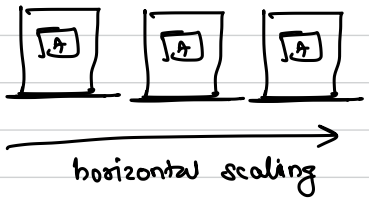
Microservices

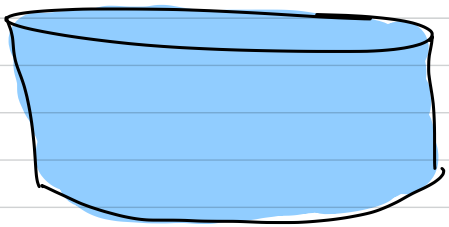
monolithic

E-commerce
- user mgmt
- product mgmt
- order mgmt
- machine learning

stack - MEAN
language - JavaScript
platform - node.js
Database - MySQL
Backend - Express







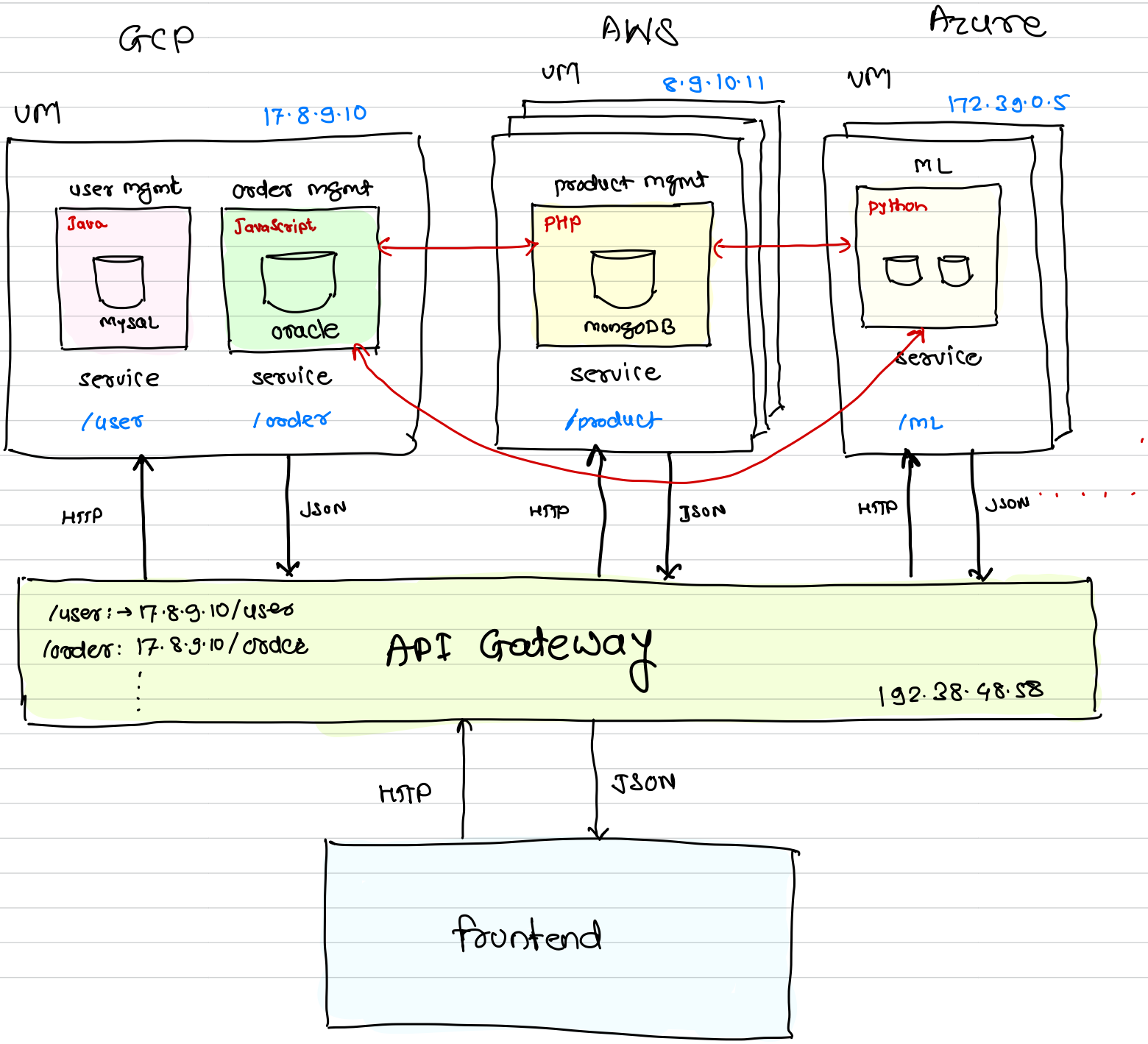
MySQL

mongoDB

Microservices

E-commerce

- user mgmt
  - Language : Java
  - database : MySQL
- product mgmt
  - Language : PHP
  - database : MongoDB
- order mgmt
  - Language : nodejs (JS)
  - database : Oracle
- machine learning
  - Language : python
  - database : —



Overview

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

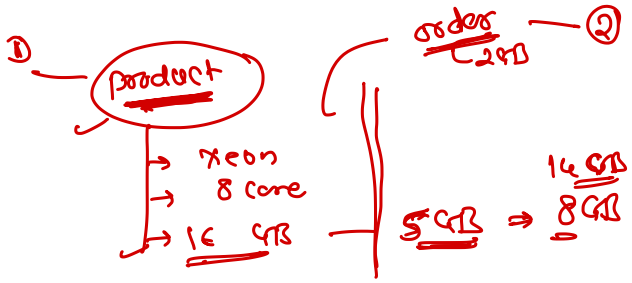
Benefits

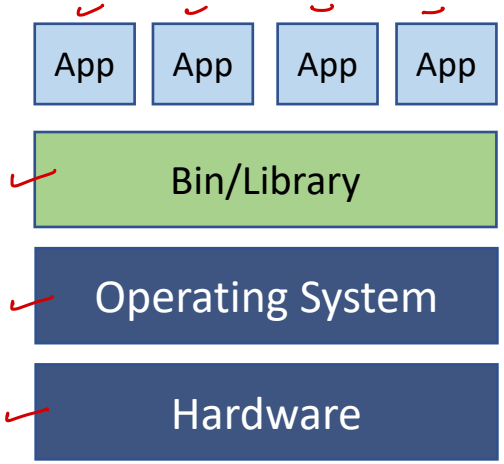
- Easier to Build and Maintain Apps
- Improved Productivity and Speed
- Code for different services can be written in different languages
- Services can be deployed and then redeployed independently without compromising the integrity of an application
- Better fault isolation; if one microservice fails,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

Containerization

Traditional Deployment

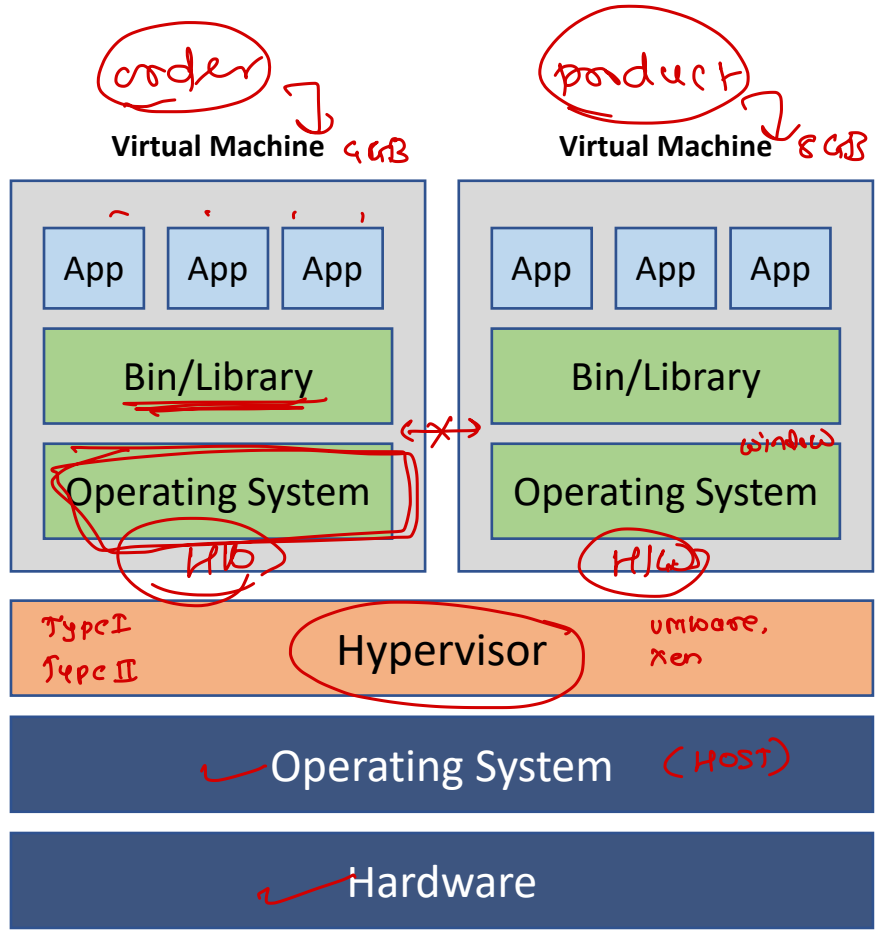
- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application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





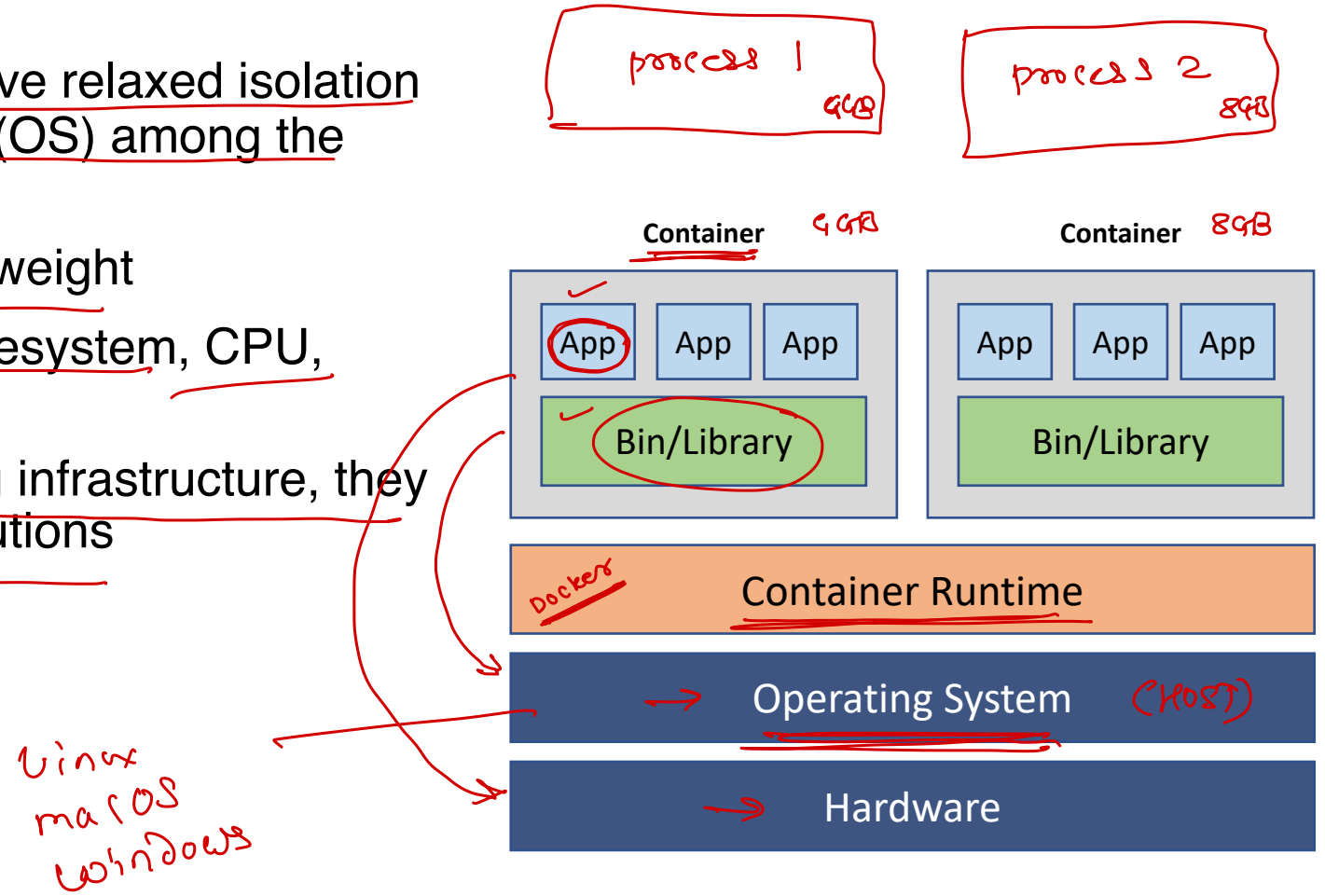
Virtualized Deployment

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



Container deployment

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



Containerization

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

Development

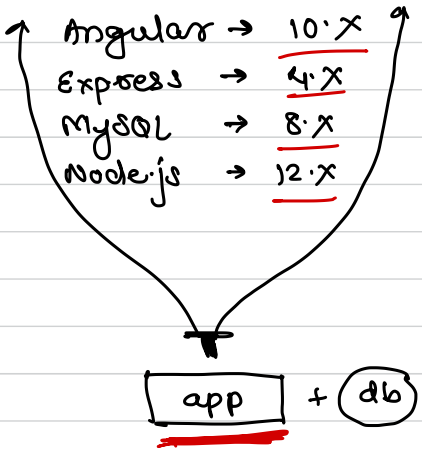
developers

① understand requirements

② develop the application

③ unit test application

eg.



testers

① understand requirement

② write test cases

③ test application

Ops

① managing resource

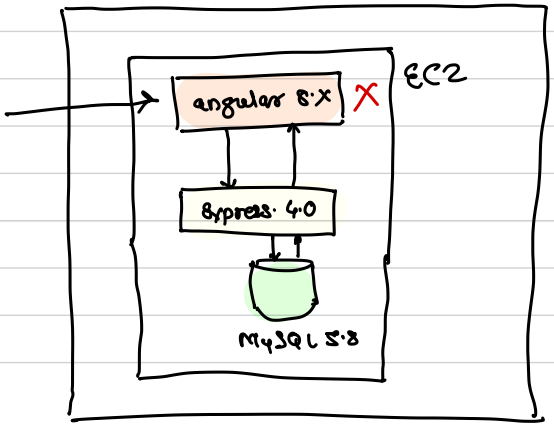
- creating VMs
- updating OS
- patches security updates

② managing networking

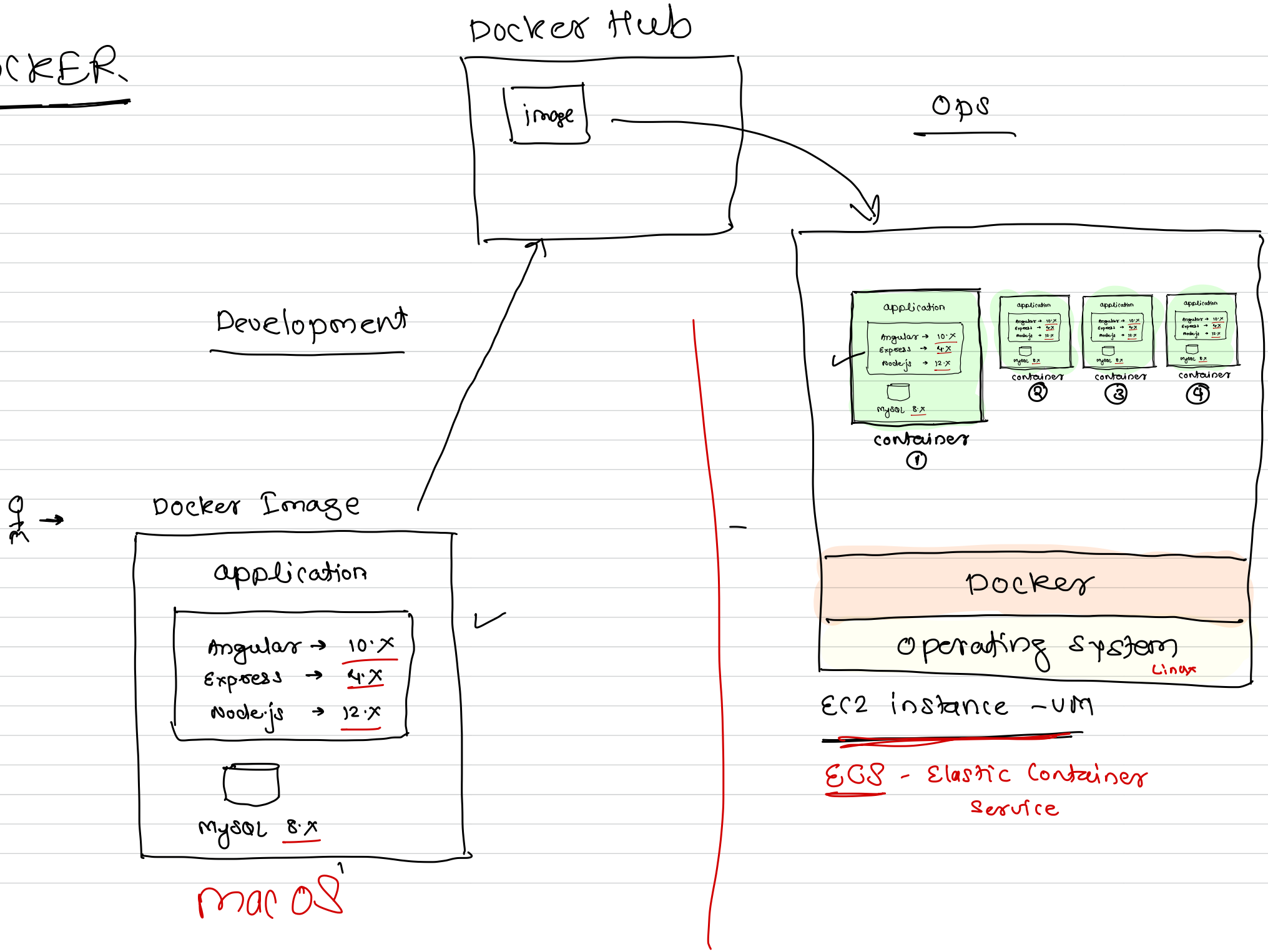
③ deploying application

angular = 8.X
mysql = 5.8
Express = 4.X
Node.js = 10.X

AWS



DOCKER


Docker Hub
image
Ops
Development
Docker Image
application
Angular → 10.X
Express → 4.X
Node.js → 12.X
MySQL 8.X
mac OS
application
Angular → 10.X
Express → 4.X
Node.js → 12.X
MySQL 8.X
container 1
container 2
container 3
container 4
docker
Operating system
Linux
EC2 instance - VM
ECS - Elastic Container Service

Containers vs Virtual machines

	Virtual Machine	Container
1	Hardware level virtualization	OS virtualization
2	Heavyweight (bigger in size) [full OS]	Lightweight (smaller in size) [shares OS]
3	Slow provisioning	Real-time and fast provisioning
4	Limited Performance [virtualized hardware]	Native performance [share h/w with physical machine]
5	Fully isolated [two VMs cannot talk to each other]	Process-level isolation [shared OS]
6	More secure	Less secure
7	Each VM has separate OS	Each container can share OS resources
8	Boots in minutes	Boots in seconds
9	Pre-configured VMs are difficult to find and manage	Pre-built containers are readily available
10	Can be easily moved to new OS	Containers are destroyed and recreated [immutable]
11	Creating VM takes longer time	Containers can be created in seconds

Advantages

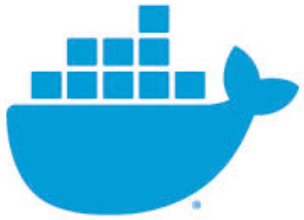
- **Portability**
  - A container creates an executable package of software that is abstracted away from (not tied to or dependent upon) the host operating system, and hence, is portable and able to run uniformly and consistently across any platform or cloud
- **Agility**
  - The open source Docker Engine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

Advantages

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

Popular container platforms

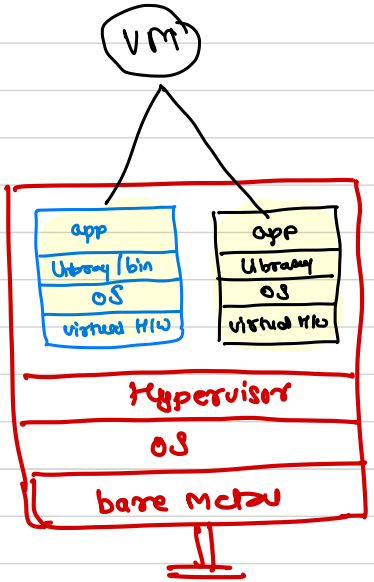
- Linux Containers (LXC)
- Docker
- Windows Server
- CoreOS rkt

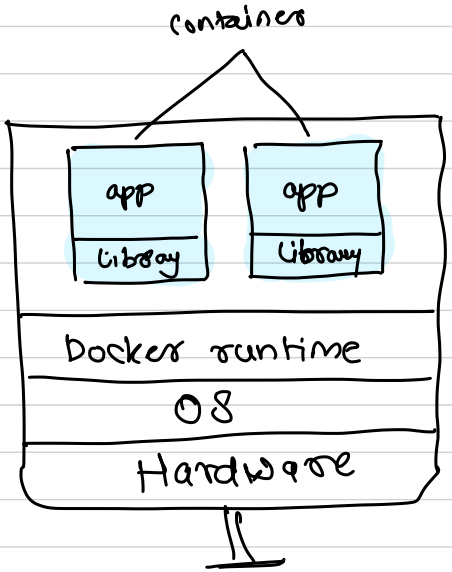


Docker

Overview

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



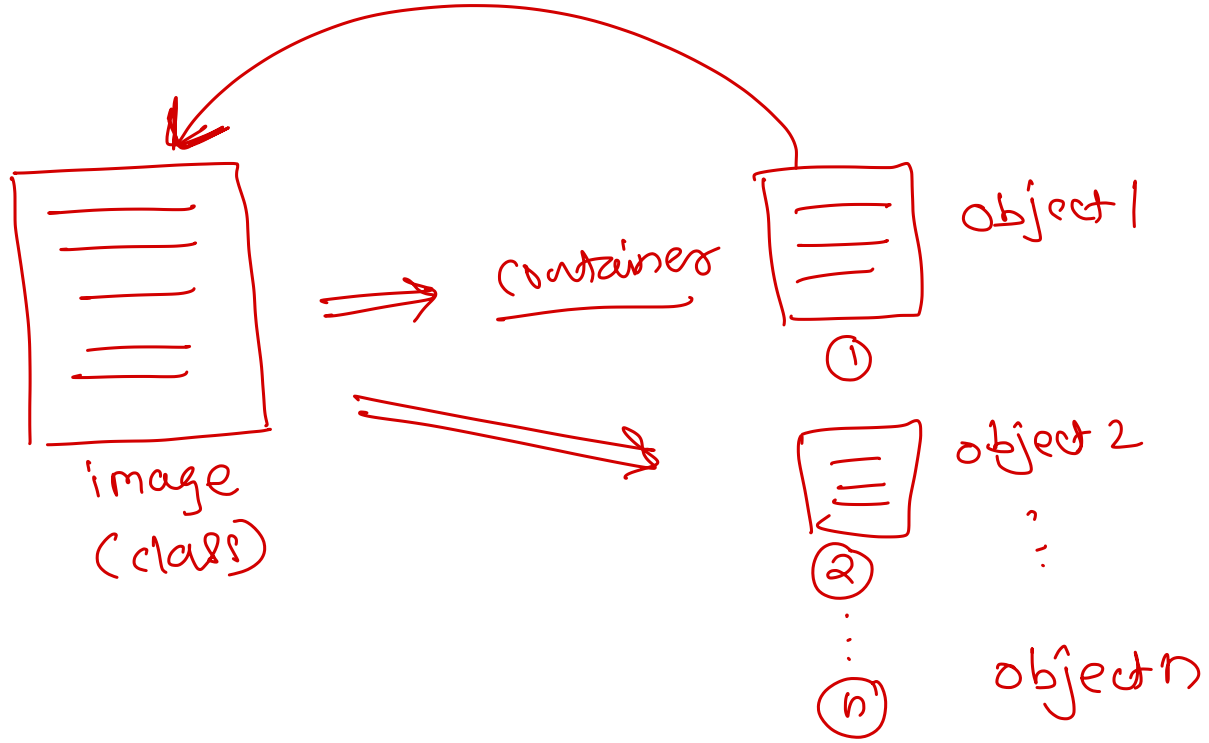


Container

- your application
  - executable
  - website code
- libraries / runtime / binaries

Images

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



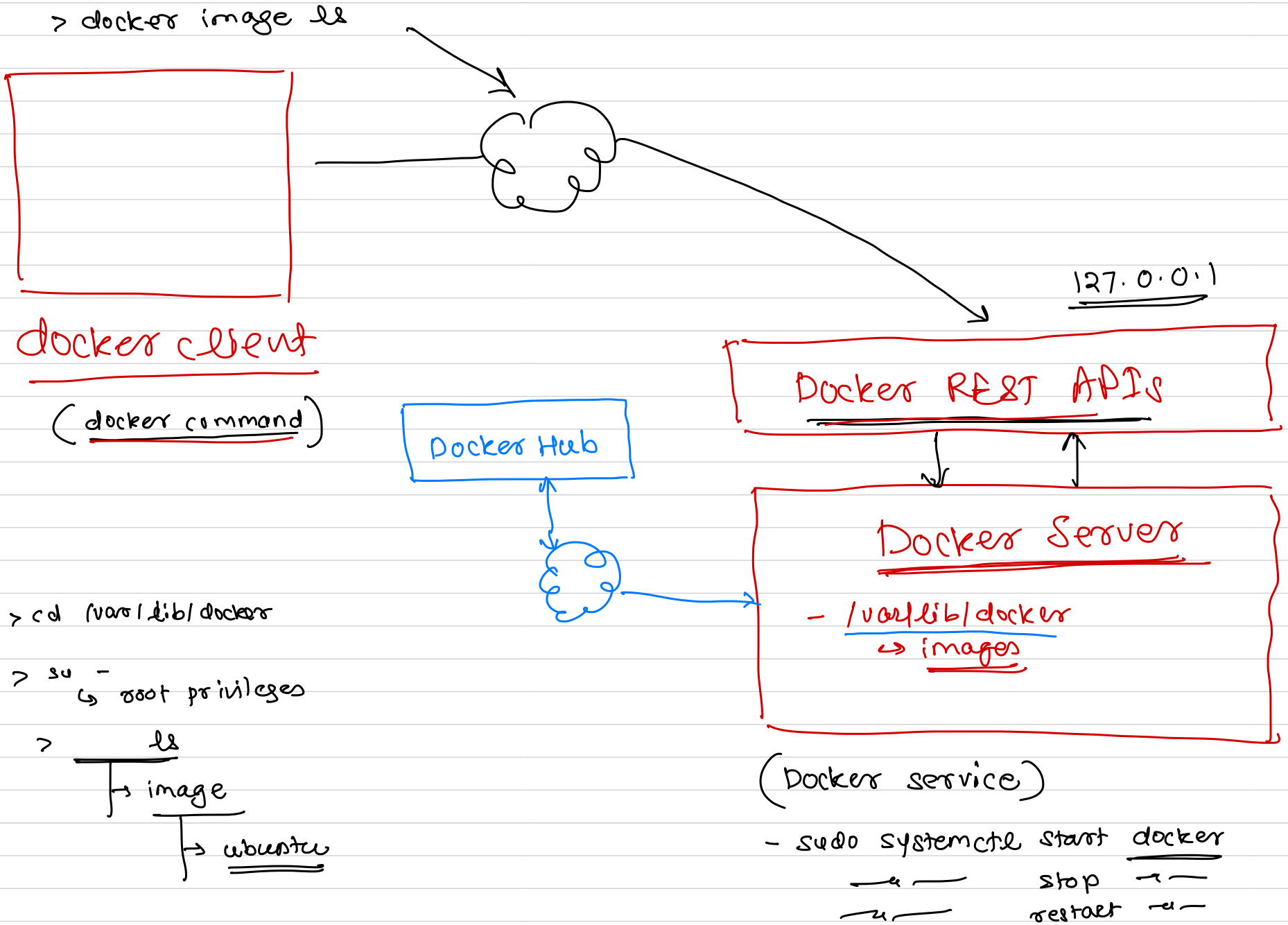
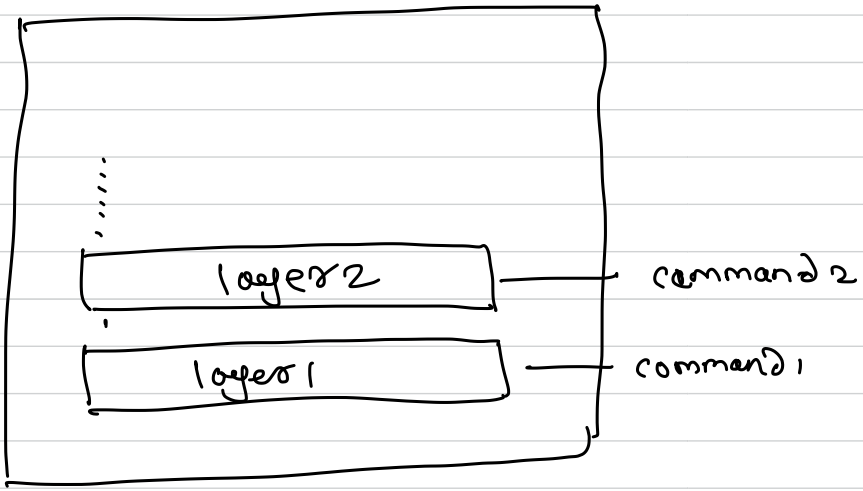

> docker image ls
docker client
(docker command)
127.0.0.1
Docker REST APIs
Docker Hub
Docker Server
- /var/lib/docker
↳ images
(Docker service)
- sudo systemctl start docker
stop
restart
> cd /var/lib/docker
> su -
↳ root privileges
> ls
image
ubuntu

image - layers



Commands related to images

- **List all the images**

  > docker image ls

- **Download an image from docker hub**

  > docker image pull <image name>

- **Get the details of selected image**

  > docker image inspect <image name>

- **Delete an image**

  > docker image rm <image name>

Commands related to images

- **Push the image to docker hub**

  > docker image push <image name>

- **Tag an image**

  > docker image tag <image name> <tag>

- **Build an image with Dockerfile**

  > docker image build <Dockerfile>

Containers

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

Commands related to containers

- **List the running containers**

  > docker container ls

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

  > docker container ls –a

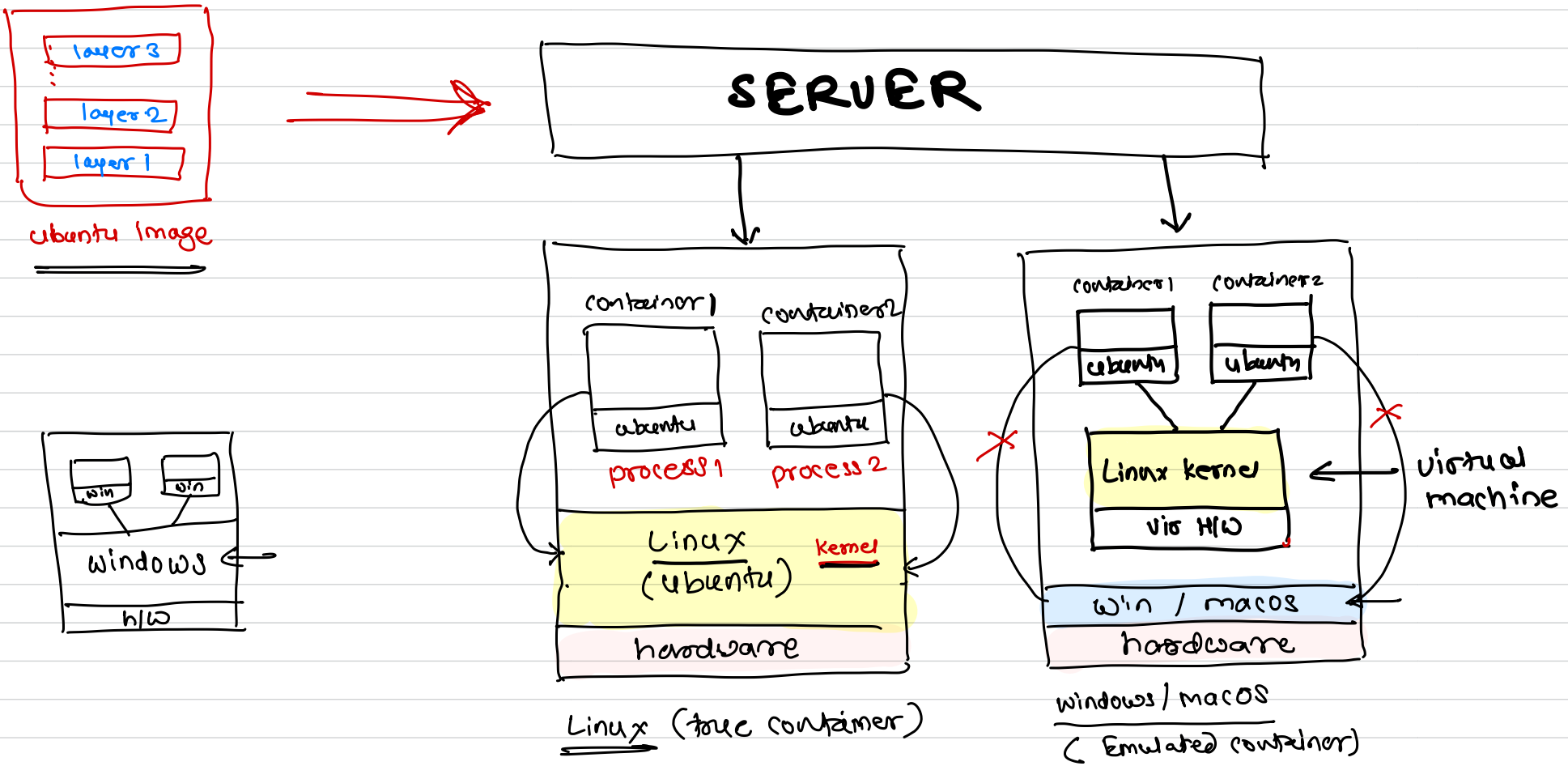
- **Create and start container**

  > docker container run <image>

- **Start a stopped/created container**

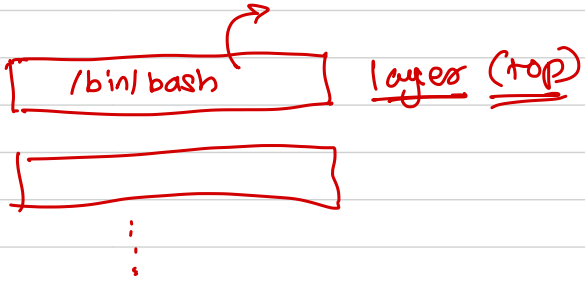
  > docker container start <container>

> docker container run ubuntu



> docker container run ubuntu

↳ detached



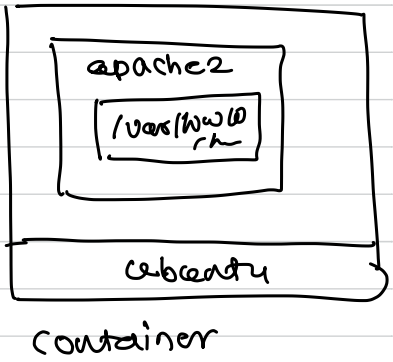
> docker container run -it ubuntu

↳ attached

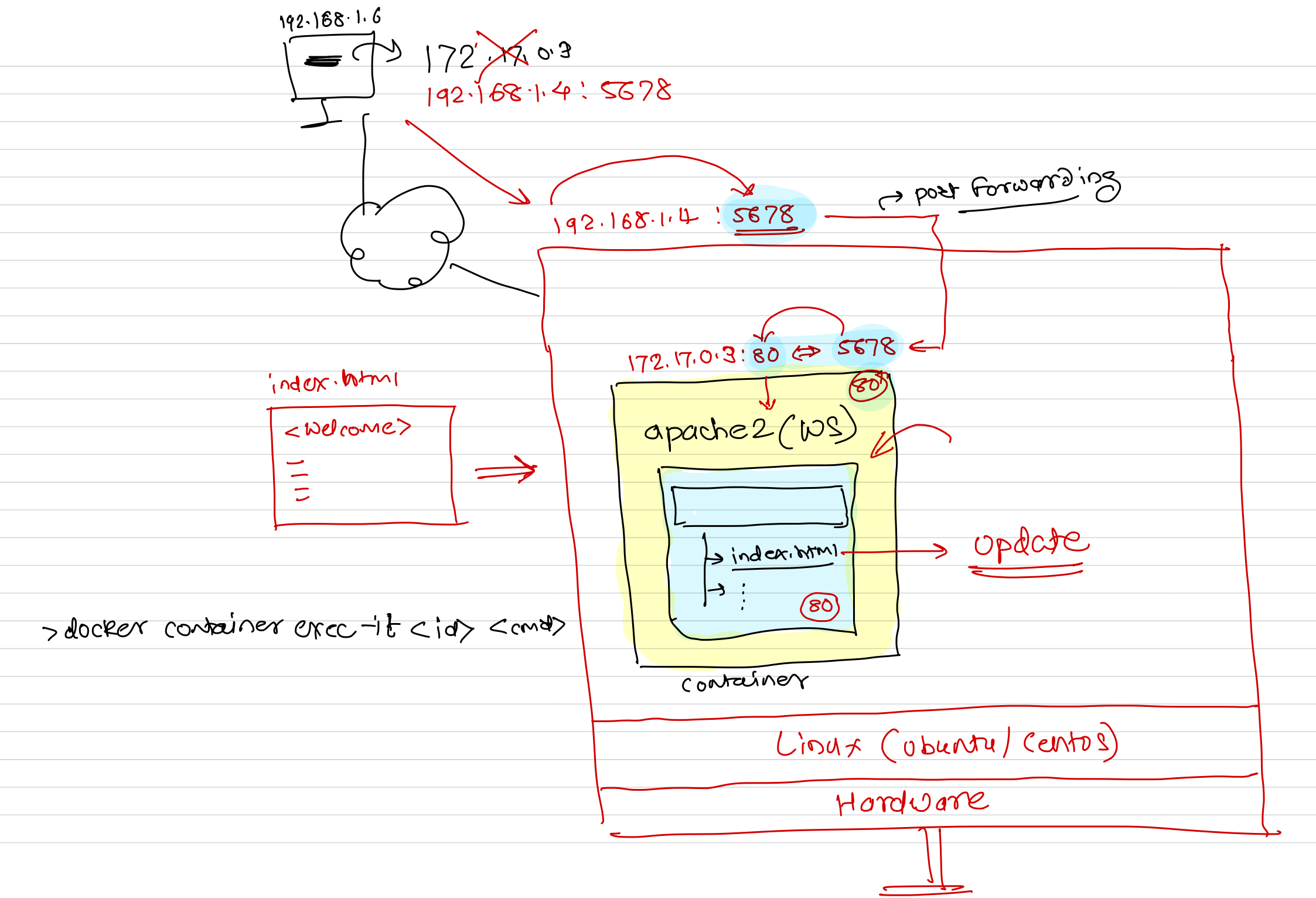
→ interactively (i)

→ teletype (t)

↳ terminal



container


192.168.1.6
172.17.0.3
192.168.1.4 : 5678
192.168.1.4 : 5678
post forwarding
172.17.0.3 : 80 ⇔ 5678
80
apache2 (WS)
index.html
<welcome>
update
80
container
Linux (ubuntu / centos)
Hardware
> docker container exec -it <id> <cmd>

Building Docker Image

- Dockerfile
  - → ADD
  - → CMD
  - → COPY
  - → RUN
  - ⋮

Commands related to containers

- **Stop a container**

  > docker container stop <container>

- **Remove a container**

  > docker container rm <container>

- **Get the stats of selected container**

  > docker container stats <container>

- **Execute a command in a container**

  > docker container exec <container>

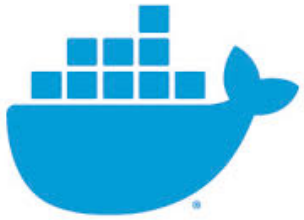
Commands related to containers

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

  > docker container commit <container>

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

  > docker container top <container>



Docker Compose

Microservices

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

Docker Compose

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

Features

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

Installation

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

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

- Apply executable permissions to the binary:

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

Start using docker compose

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

```
version: ‘3’
services:
  web:
    build: .
    ports:
      - “9090: 80”
```

Build and run the application

- To run the application use

  > docker-compose up

- To stop the containers

  > docker-compose stop

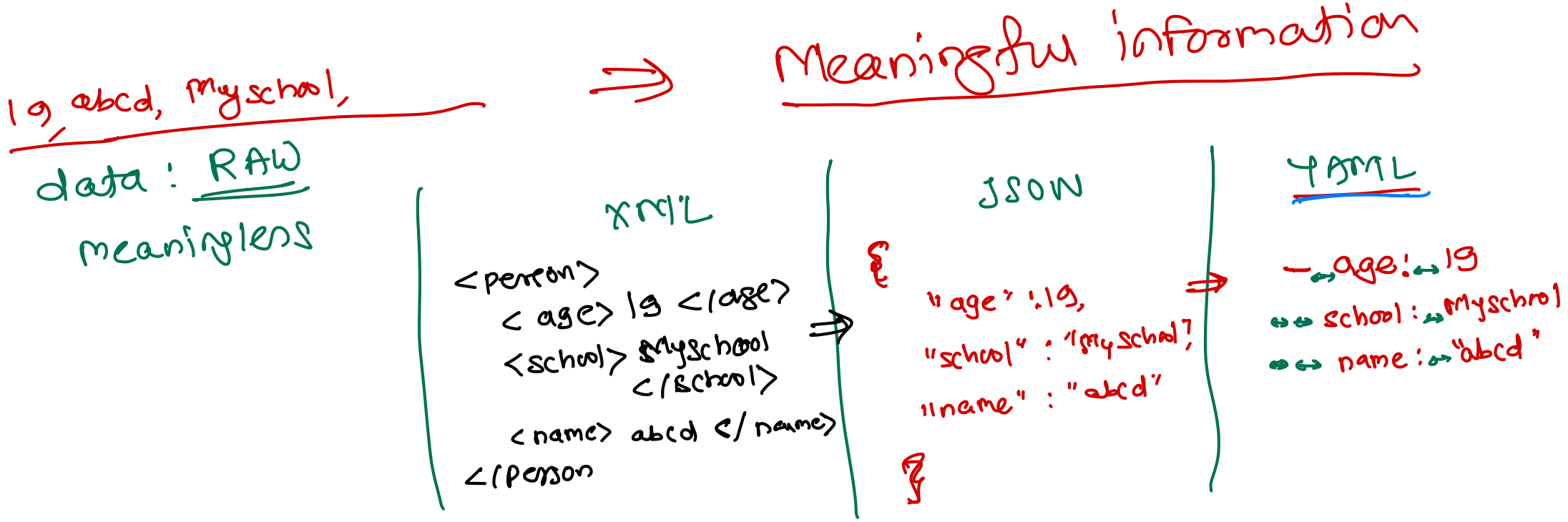
- To remove the containers

  > docker-compose down

YAML

Overview

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



Features

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

Basics

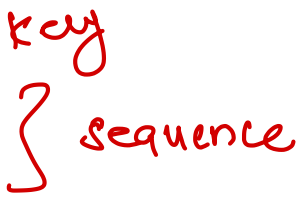
- YAML is case sensitive
- The files should have **.yaml** or **.yml** as the extension
- YAML does not allow the use of tabs while creating YAML files; spaces are allowed instead
- Comment starts with #
- Comments must be separated from other tokens by whitespaces.

Scalars

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

Mapping

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



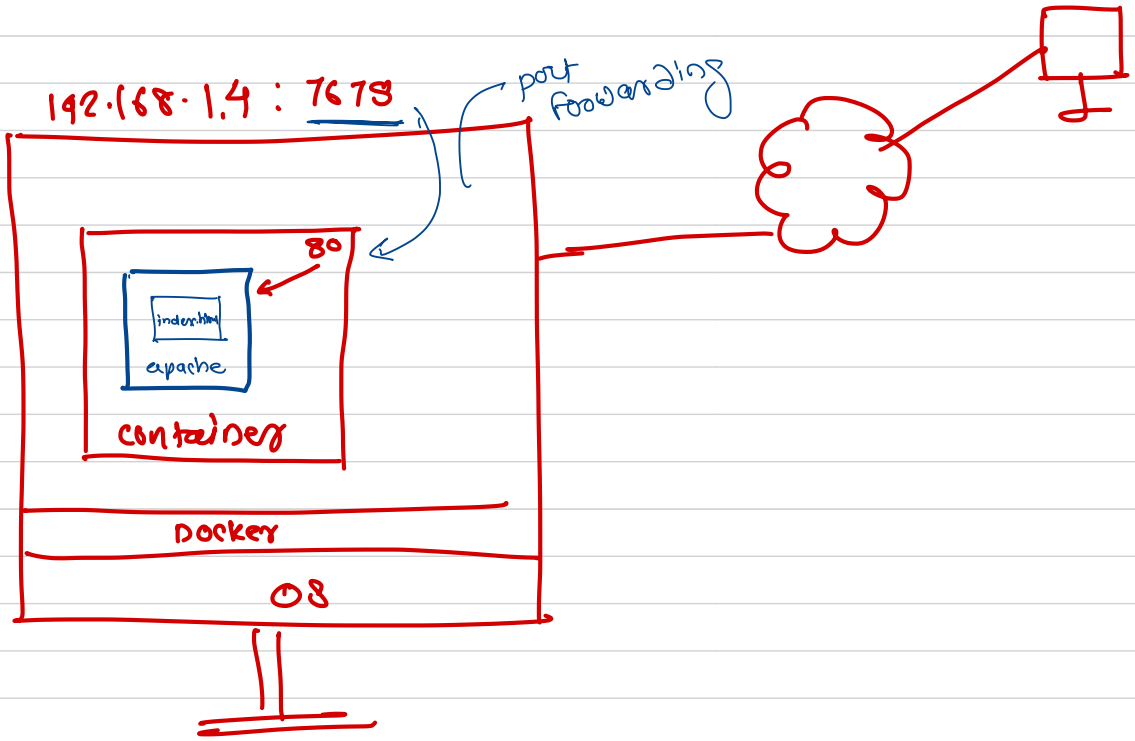
Sequence (List)

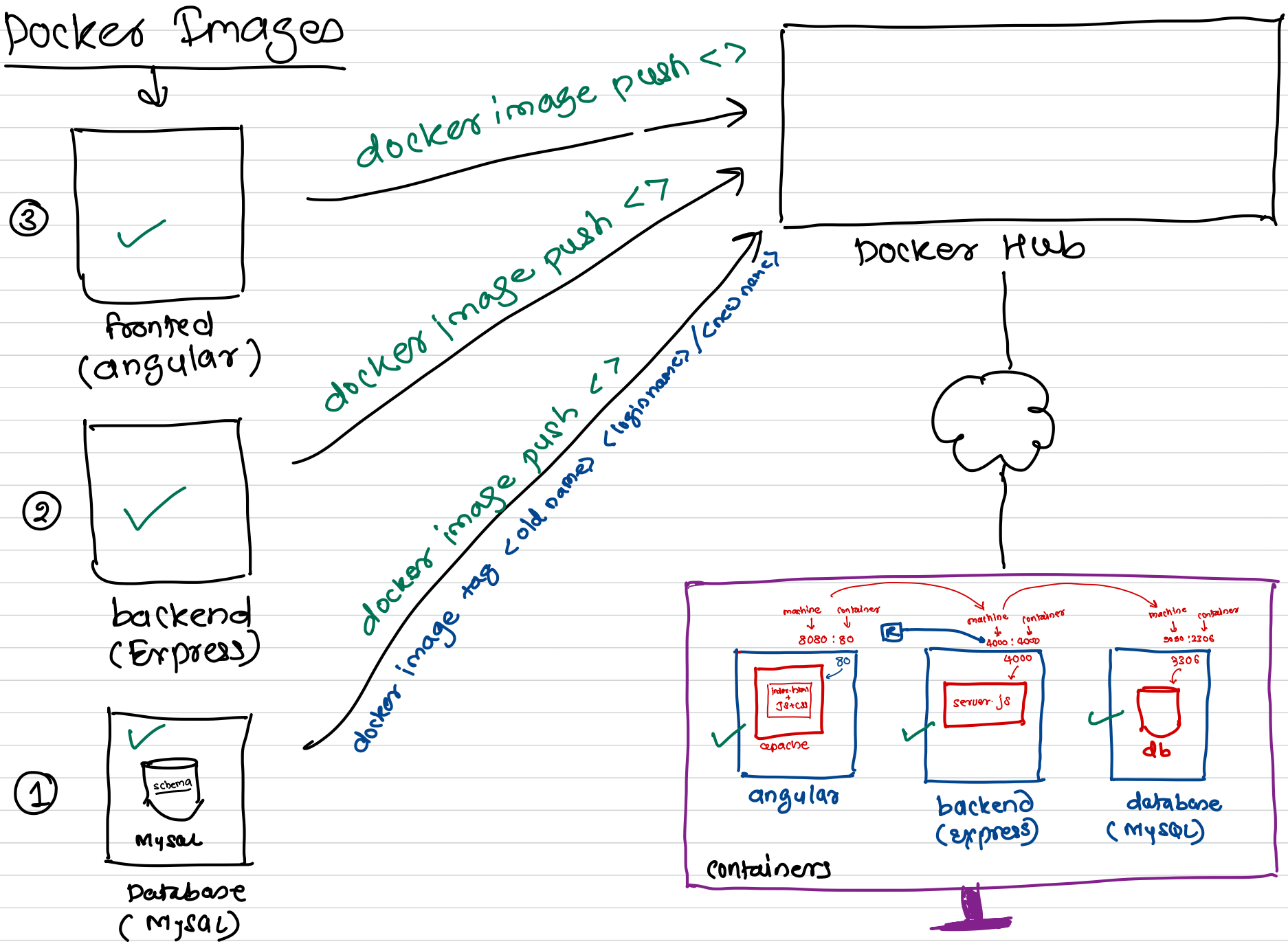
- Represents list of values
- Must be written on separate lines using dash and space
- Please note that space after dash is mandatory
- E.g.
  - # pet animals
  - - cat
  - - dog

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

Sequence

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


192.168.1.4 : 7679
port forwarding
80
index.html
apache
container
Docker
OS


Docker Images
docker image push <>
docker image push <>
docker image push <>
docker image tag <old name> <login name>/<new name>
Docker Hub
③
frontend
(angular)
②
backend
(Express)
①
schema
MySql
Database
(MySQL)
machine
container
8080 : 80
4000 : 4000
3306 : 3306
80
4000
3306
index.html + JS+CSS
server.js
db
apache
angular
backend
(express)
database
(MySQL)
containers

custom image always needs a base image

To build an image

① create a file Dockerfile

② add required instructions

FROM : base image
ENV : set environment variable
COPY : used to copy file from local machine to container
ADD : used to copy file from local machine to container
EXPOSE : used to expose port from container
CMD : used to execute command when container starts

③ build the image

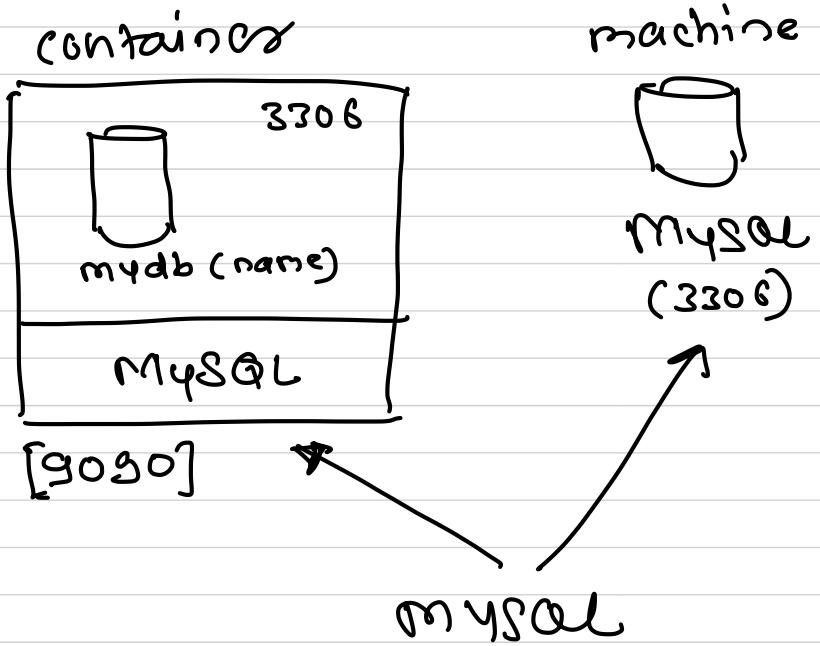
```
> docker image build . -t <image name>
```

Database image

① database : MySQL :→ Base Image [Docker Hub]

② database schema
- → tables
- → store procedures
- → views
- → functions

③ environment variables
- → MYSQL_ROOT_PASSWORD
- → MYSQL_DATABASE

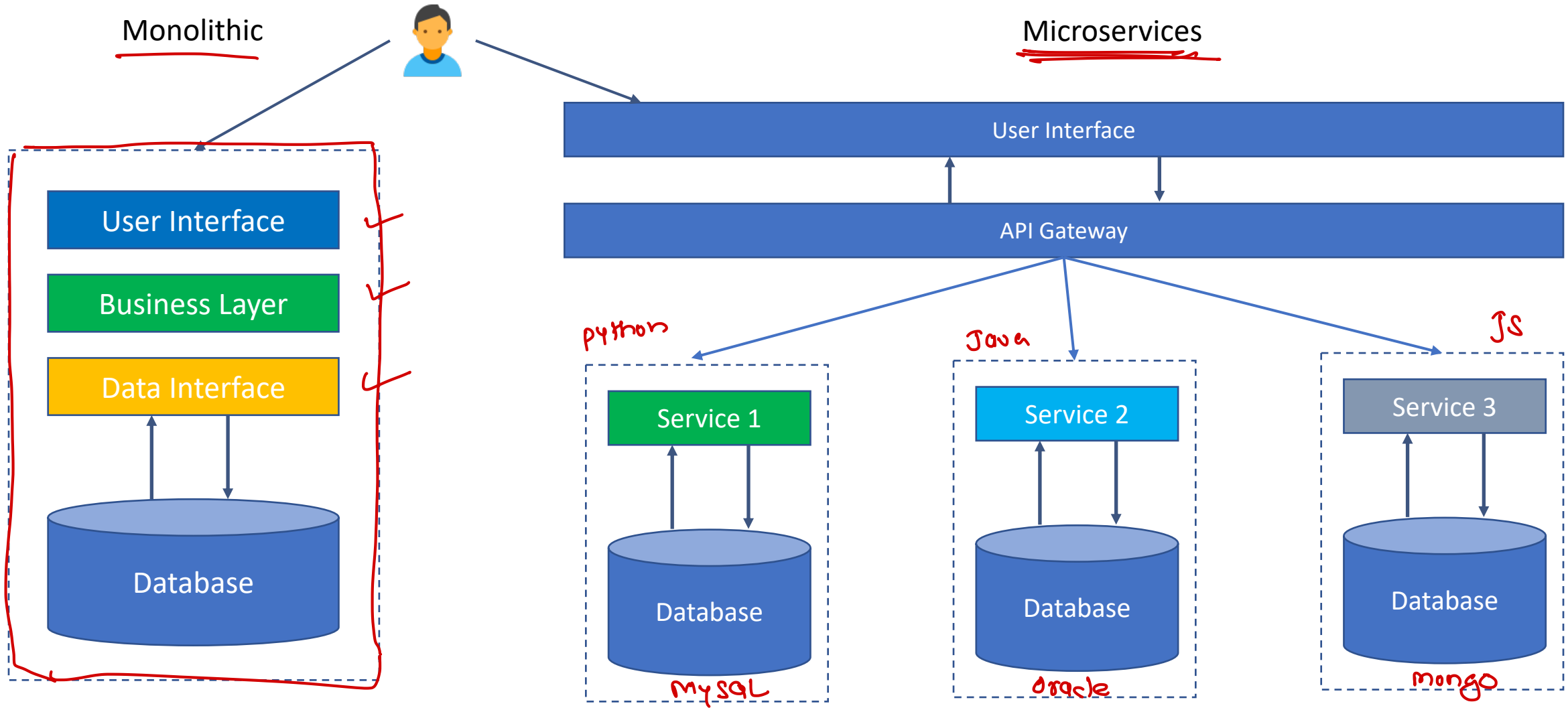


kubernetes

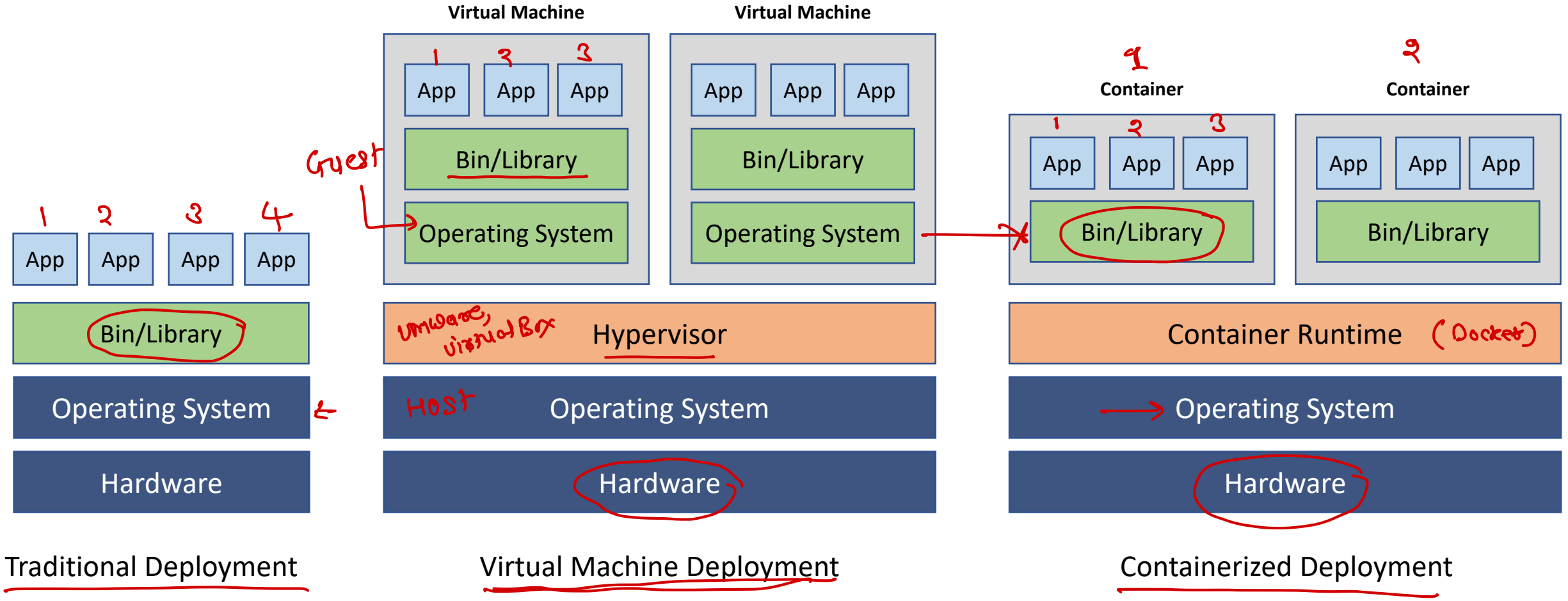
Microservice

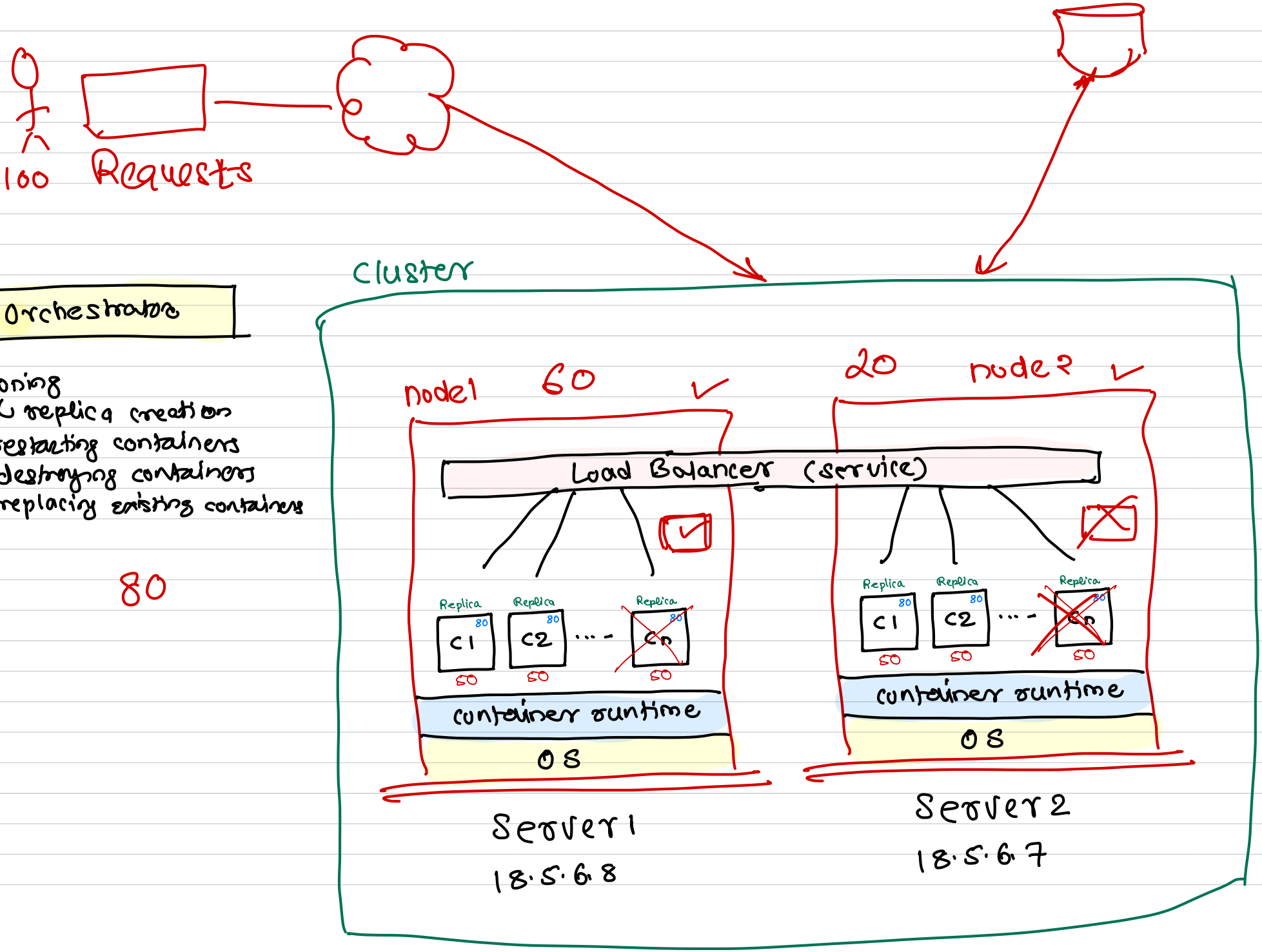
- Distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations
- Is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled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

Monolithic vs Microservice



Deployment options



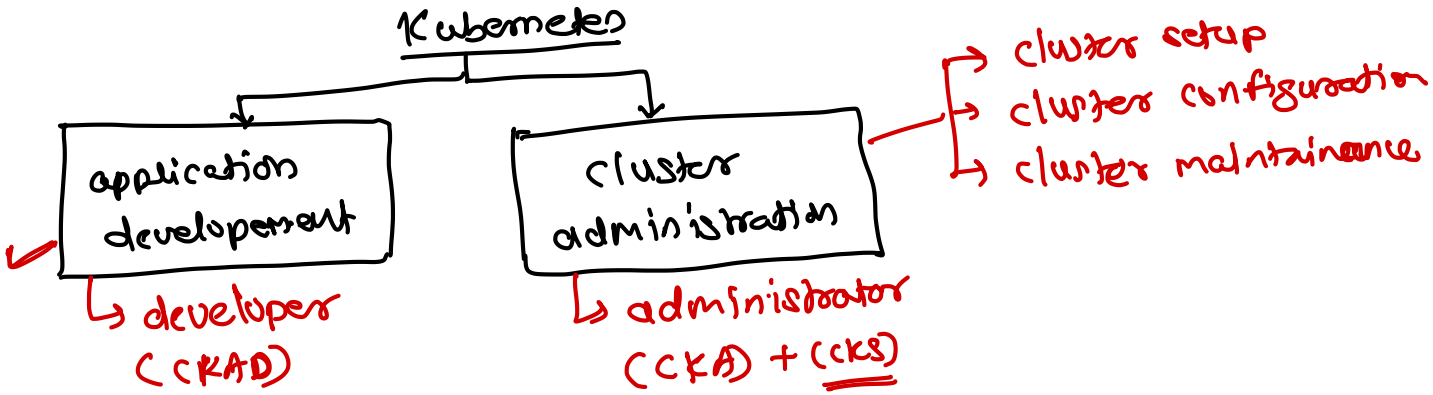

100
Requests
Cluster
Orchestrator
① cloning
↳ replica creation
② restarting containers
③ destroying containers
④ replacing existing containers
80
node1
60
node2
20
Load Balancer (service)
Replica
C1
C2
Cn
80
50
container runtime
OS
Server1
18.5.6.8
Server2
18.5.6.7

Container orchestration

- Container orchestration is all about managing the lifecycles of containers
- 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
  - 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
  - Movement of containers from one host to another if there is a shortage of resources, or if a host dies
- Tools
  - Docker Swarm → <10,000
  - Kubernetes → >10,000
  - Apache Mesos and Marathon

Introduction

- Kubernetes is an open source system for automating deployments, scaling and management of containerized applications
- It is a container orchestration system
- It enables organizations to automate deployment and manage the containers, thus helping them to streamline and simply the day to day routines
- If you are using Kubernetes that means your application is following the microservices architecture and is already containerized

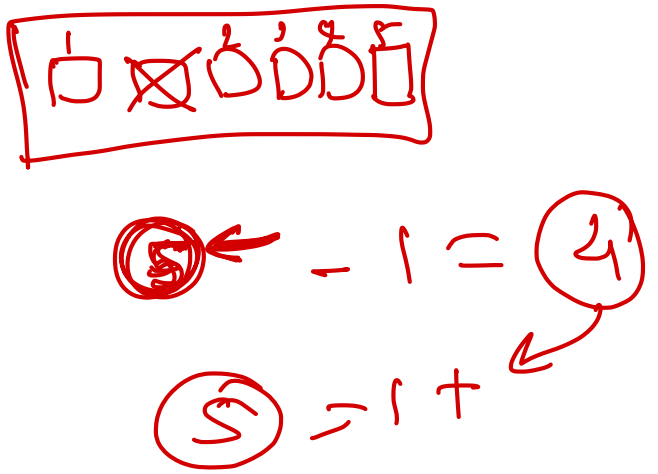


History

- Kubernetes was originally developed at Google having emerged from project Borg
- Now it has been taken over by Cloud Native Computing Foundation (CNCF)
- It is not only open source but it is managed by open community

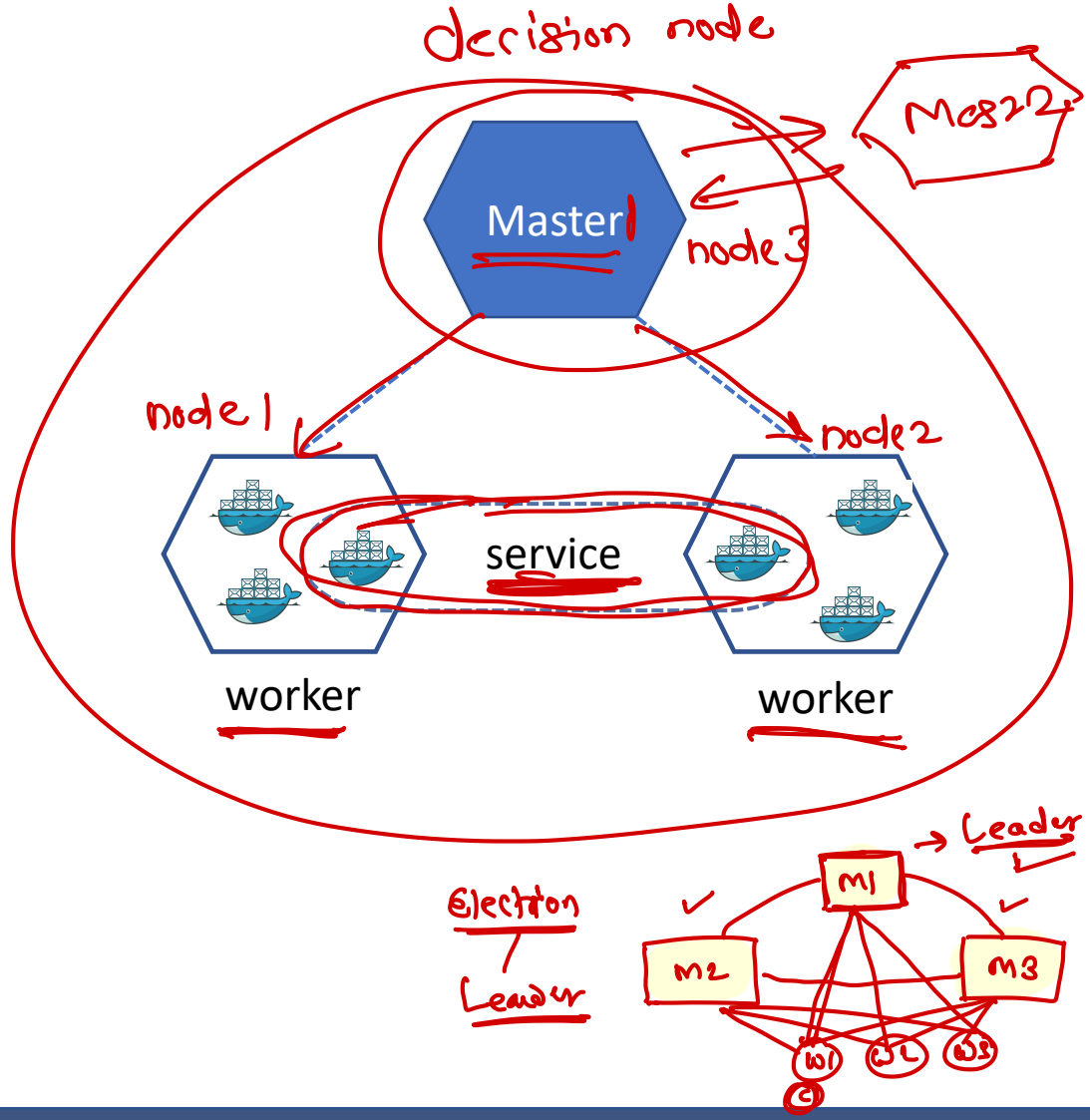
Kubernetes Features

- Provisioning and deployment of containers
- Self healing
- Service discovery and load balancing
- Storage orchestration
- Auto scaling
- Run anywhere
- Automated rollouts and rollbacks
- Secrets and configuration management
- Scale
- REST APIs at its core
- Security

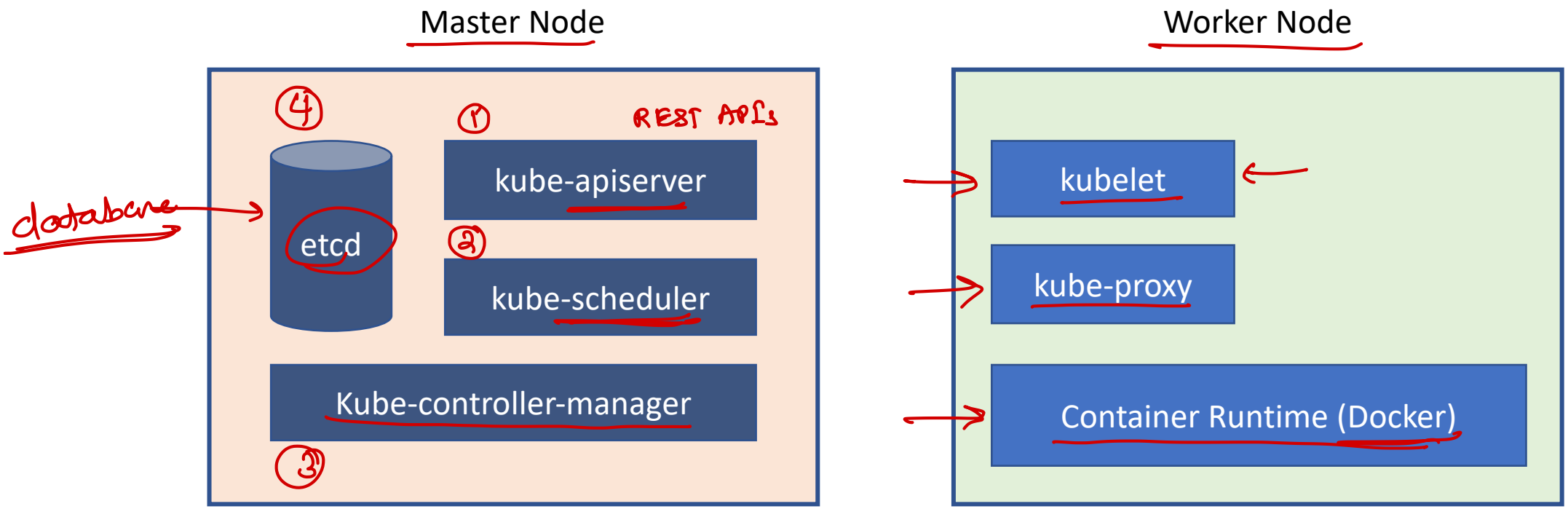


Kubernetes Cluster

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

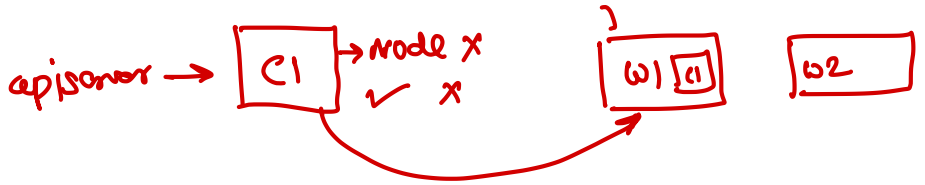


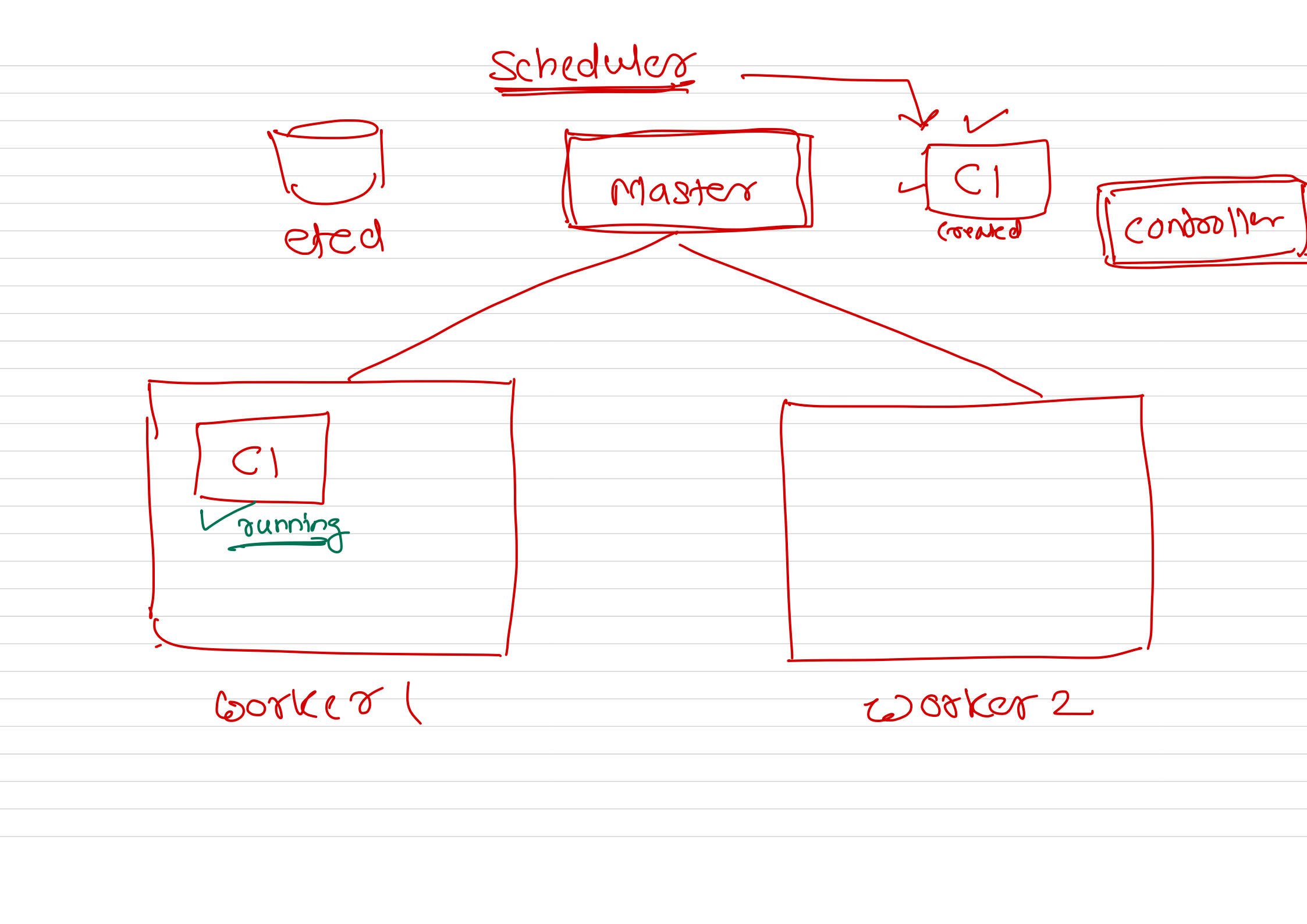
Kubernetes Components



Master Components

- Master components make global decisions about the and they detect and respond to cluster events
- Master components can be run on any machine in the cluster
- **kube-apiserver**
  - The API server is a component that exposes the Kubernetes REST API
  - The API server is the front end for the Kubernetes
- **etcd**
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
  - The configuration data the etcd stores represents the cluster state
  - e.g. which nodes are running the containers, which applications are running etc
- **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




Scheduler
etcd
Master
C1
created
controller
C1
running
worker 1
worker 2

Master Components

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

Node Components

[both master & worker run these components]

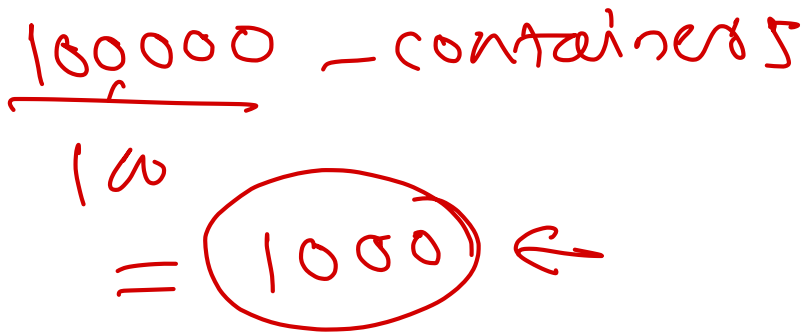
- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment
- **kubelet**
  - An agent that runs on each node in the cluster
  - It makes sure that containers are running in a pod

] worker kubelet updates the current status to master

- **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.

Designing a cluster

- Application components and the number of replicas required to run the application smoothly
- Kubernetes v1.16 has following restrictions
  - 5000 nodes
  - 150000 pods
  - 300000 containers
  - 100 pods per node
- Resource requirements of the pods
  - Processor, Memory and HDD requirements for the pods
  - Application Data Storage
- Co-located or external etcd
- Cluster monitoring
- Infrastructure: on-premises, bare metal or managed cloud cluster
- Security



AKS, EKS

Designing a cluster

- Worker nodes requirements
- Number of workers required to run the pods
- Configuration of worker nodes
- https://kubernetes.io/docs/setup/best-practices/cluster-large/

Nodes	Configuration
1-5 nodes	2 vCPU / 4 GB RAM
6-10 nodes	2 vCPU / 7.5 GB RAM
11-100 nodes	4 vCPU / 15 GB RAM
101-250 nodes	8 vCPU / 30 GB RAM
251-500 nodes	16 vCPU / 30 GB RAM
More than nodes	36 vCPU / 60 GB RAM

Cluster node requirements

- Cluster can be installed on physical or virtual machine
- Following operating systems are supported with v1.16
  - Ubuntu 16.04+
  - Debian 9+
  - Centos 7+
  - RHEL 7+
  - Fedora 25+
  - Container Linux
  - HypriotOS 1.0.1+
- Minimum configuration on each node is 2CPU and 2 GB RAM
- Control plane ports needed to open
  - 6443, 2379-2380, 10250, 10251, 10252
- Worker nodes ports needed to open
  - 10250, 30000-32767

High Availability Cluster

- No single point of failure
- Kube-apiserver is exposed to worker nodes using a load balancer
- Stacked etcd or External etcd

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 I sudo apt-key add -

> cat <<EOF I 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

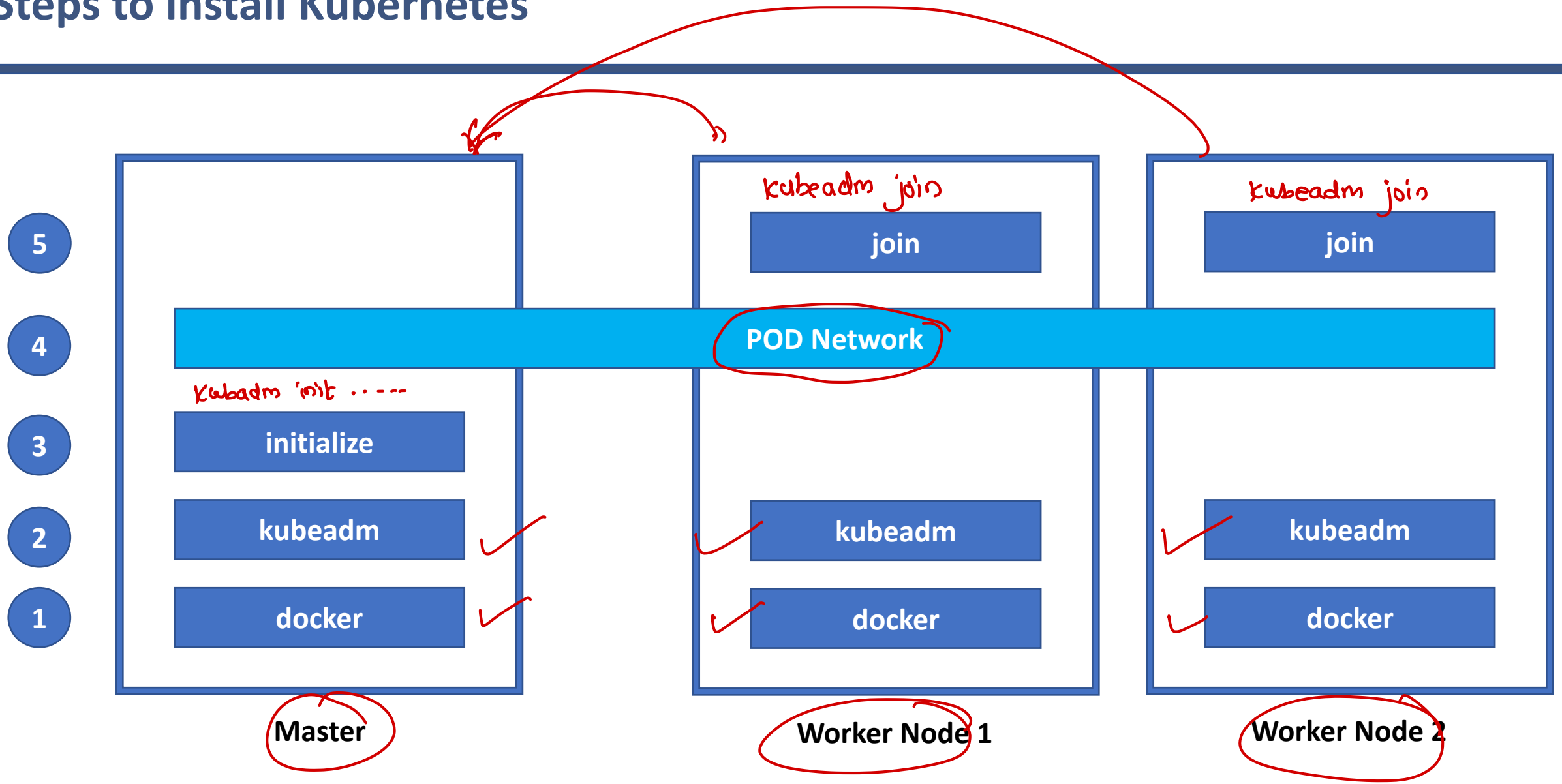
```
> kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml
```

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



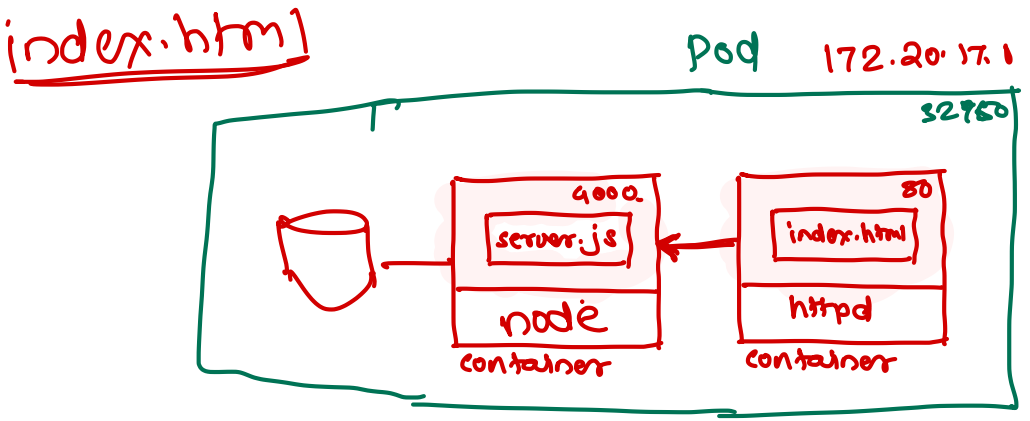
Core Concepts

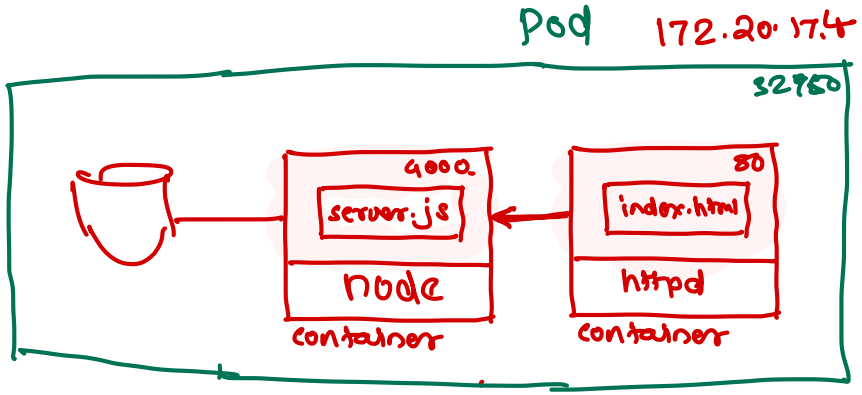
Kubernetes objects

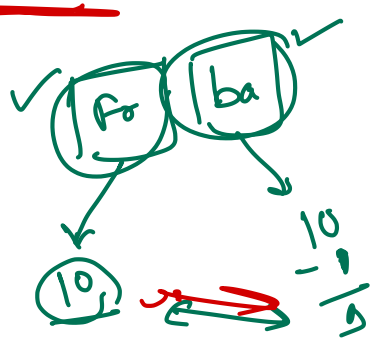
- Persistent entities expressed in YAML format
- Represent the desired state of the cluster
- Describes the following
  - Containerized application running in the cluster
  - Resources made available to these applications
  - Policies for these applications
- Objects can be created using command line utility (kubectl)
- You can also use client libraries to make the API calls direcly
  - C#, java, JS, python → REST

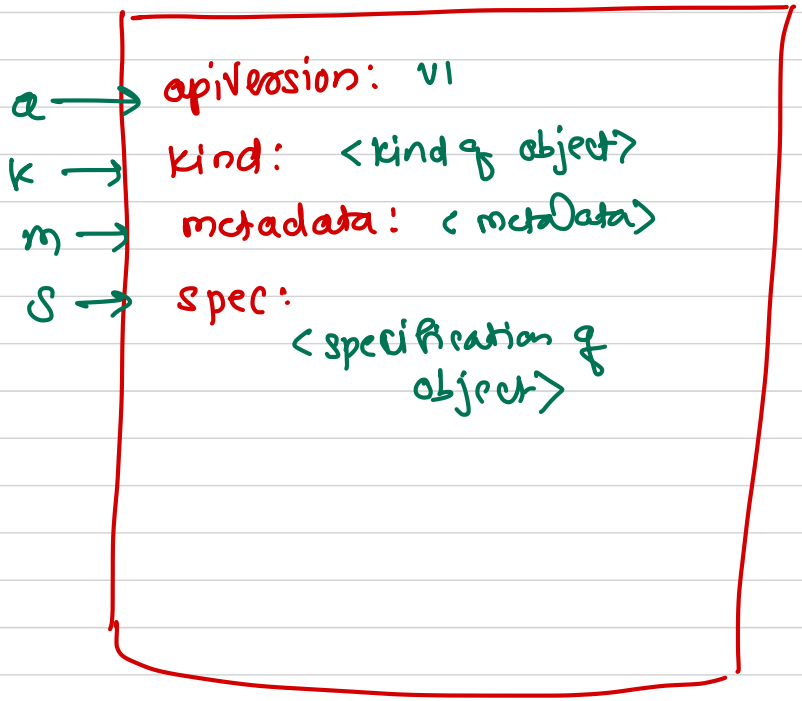
Pods

- Smallest deployable unit in the Kubernetes
- A pod encapsulates
  - Single or multiple containers (Init or App containers)
  - Storage resources
  - Unique Network IP
  - Options governing how the containers should run
  - Ephemeral and disposable entities
- Task
  - Create your first pod using yaml file








a
k
m
s
apiVersion: v1
kind: <kind of object>
metadata: <metaData>
spec:
<specification of object>

pod.yml

Phases of Pod

- Phase of a Pod is a simple high level summary showing what state the Pod is in
- Pending
  - Accepted by the cluster
  - Waiting to be scheduled
- Running
  - Scheduled on one of the worker nodes
  - In running/starting/restarting state
- Succeeded
  - All containers in the Pod are terminated successfully and will not be restarted
- Failed
  - At least one container terminated with failure state
- Unknown
  - For some reason the state is not known (may be due to the node communication error)

Kubernetes Object Management

- The kubectl supports 3 ways to manage the objects
- Imperative commands
  - Operates directly on the live objects
  - Simple with single step commands
  - Do not provide template for creating new objects
  - E.g. kubectl run nginx --image nginx
- Imperative object configurations
  - Specify the file containing the full definitions of obejcts
  - Configuration is simple to understand and can be source controlled
  - Requires additional step of creating YAML file
  - E.g. kubectl create -f config1.yml
- Declarative object configuration
  - User does not define the operations to be taken
  - Create, update and delete operations are automatically detected per-object by kubectl
  - Harder to debug and understand results
  - E.g. kubectl apply –f <directory>/

kubernetes

Pod Replication

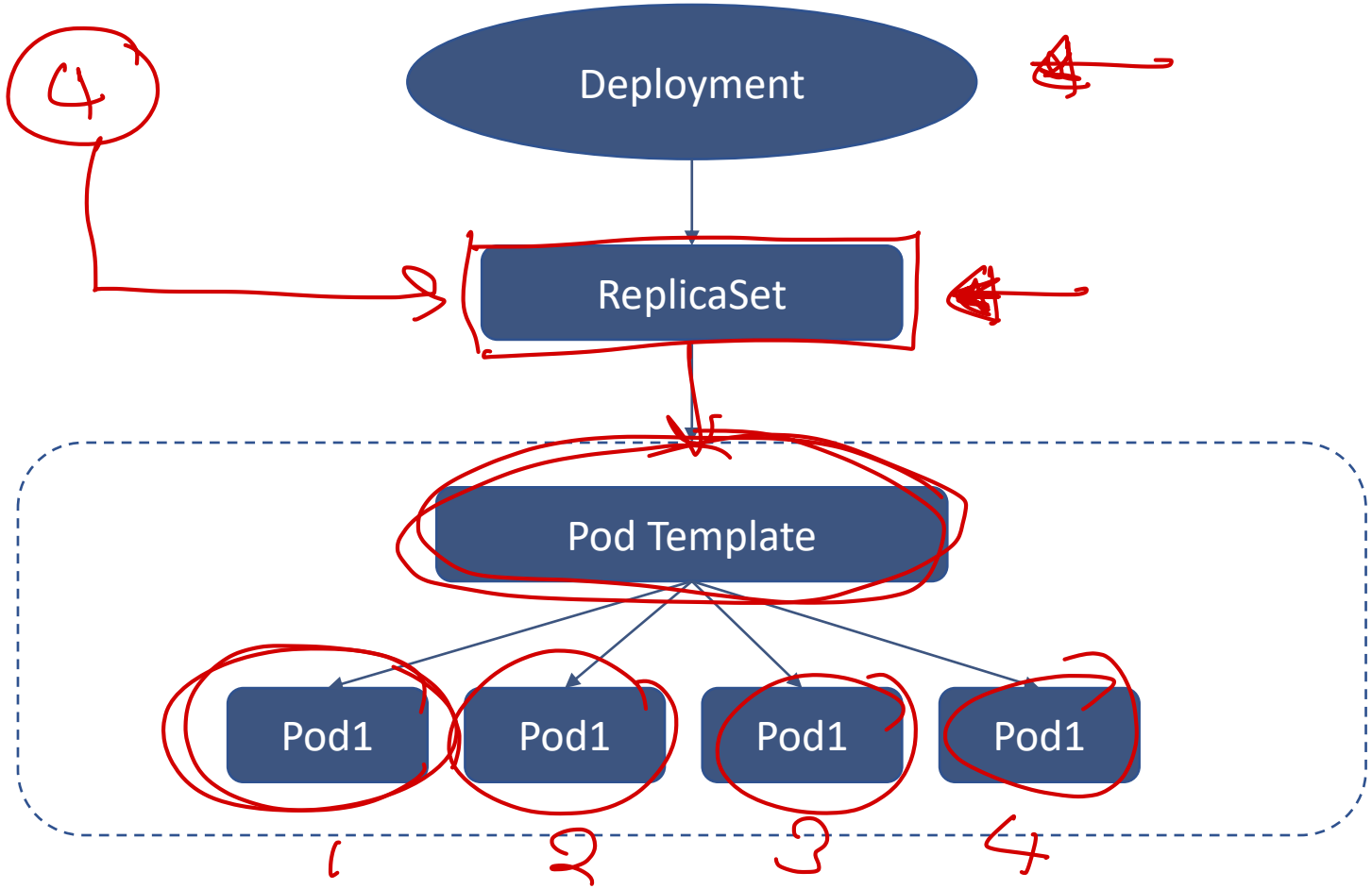
Replication Controller & ReplicaSet

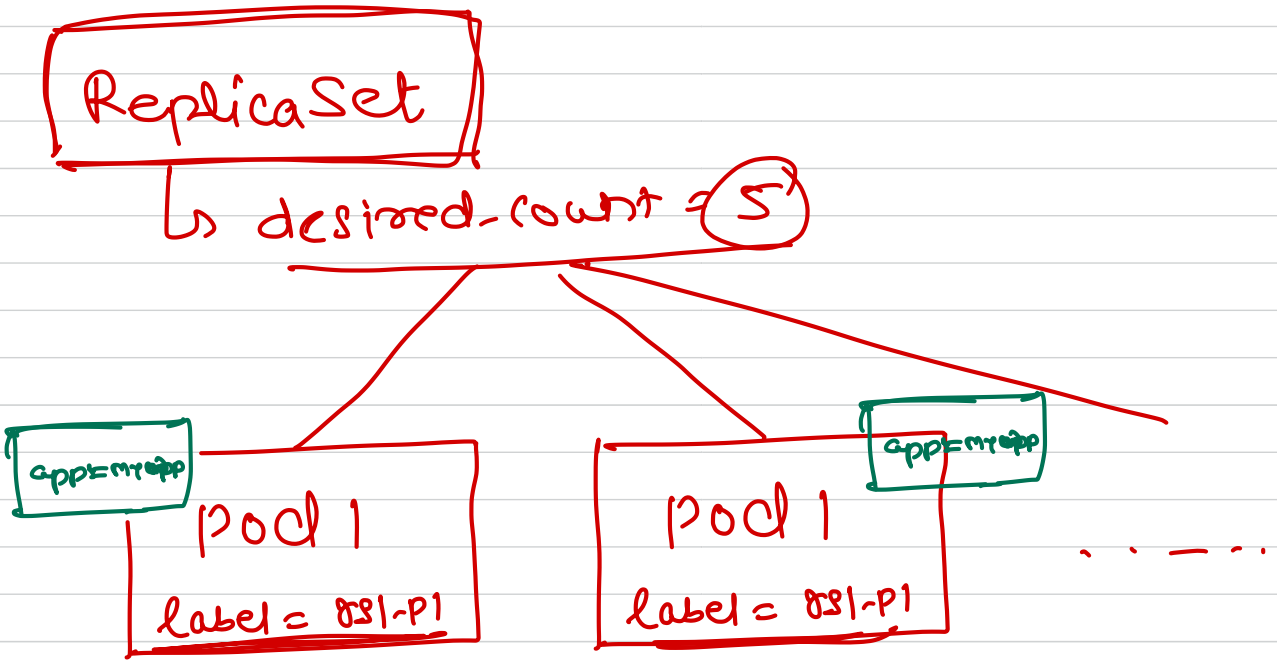
- Both of them do the similar job
- You specify the number of pods (desired count) you want to run, and RC or RS will make sure that those many pods are running at any given time
- If pod crashes RC/RS will start new pods to match the desired count
- To create the new pod, RC/RS will use the template specified in the yml definition
- Replication Controller supports only equality based selectors
- ReplicaSet supports both equality based as well as set based selectors

Deployments

- Describes the desired state of a component of the application
- In a simplest case, deployment involves one more more ReplicaSets to create pod replicas
- User does not require to use ReplicaSet separately as deployment will use it internally
- A single deployment is similar to a single microservice in the application
- Result of each deployment is a ReplicaSet which then manage the pods in the microservice
- You can create deployment
  - Using command
  - Using yml definition file

Deployment


Deployment
ReplicaSet
Pod Template
Pod1
Pod1
Pod1
Pod1
1
2
3
4


ReplicaSet
desired-count = 5
app=myapp
pod 1
label = dsl-p1
app=myapp
pod 1
label = dsl-p1

pod8 pod10

Services

Need of Services

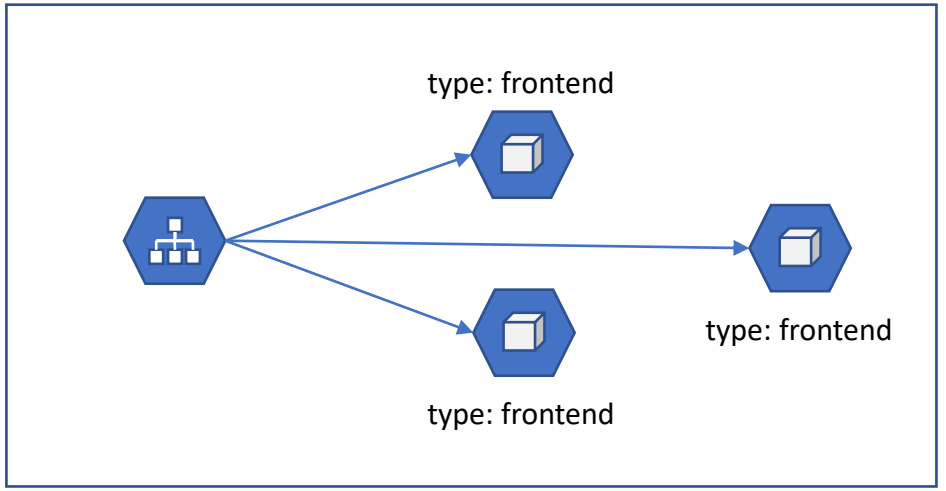
- Kubernetes pods are mortal, means they can die because of any reason
- Kubernetes provide different controllers to manage the lifecycle of these pods.
- Kubernetes gives a pod its own unique IP address, but if the pod dies then the IP address changes
- It becomes very difficult to keep track of which IP address to connect to access the application
- The problem increases in multi-tier applications

Service

- Service provides an abstraction for a set of Pods and a policy to access them
- The set of Pods targeted by a service are determined by selector
- Service is used to expose an application running in set of pods
- It provides a single DNS name and can load balance across them
- Represented by Kubernetes API object and it is namespaced

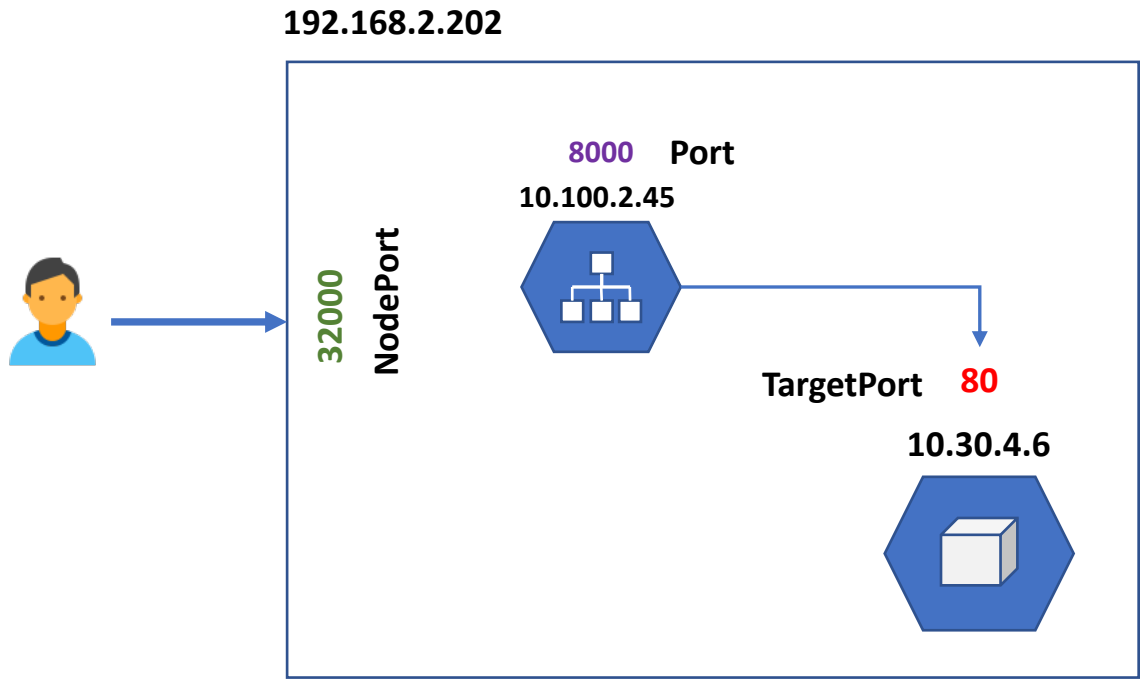
Service Type: ClusterIP

- Exposes the service within the cluster
- It uses cluster internal IP which is not reachable from external network
- Service maps any incoming port to a targetPort



Service Type: NodePort

- Exposed on each node's IP address
- NodePort service can be accessible from external network
- Service provides the NodePort on which the application is accessible
- ClusterIP service gets created automatically



Service Types

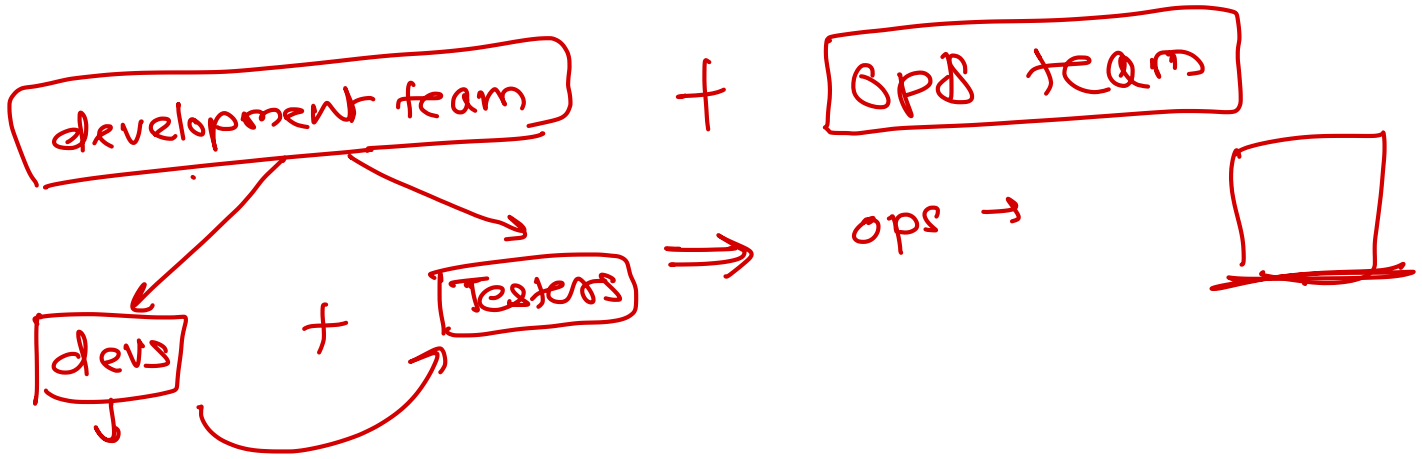
- LoadBalancer
  - Exposes the service externally using a cloud provider's load balancer
  - Along with the LoadBalancer, ClusterIP and NodePort services are created automatically
  - External load balancer routes the traffic through these pods
- ExternalName
  - The service does not contain the selectors, instead it uses DNS names
  - It maps the pod to the external name like test.sunbeaminfo.com
  - Mainly used to expose and object using cname record

Jenkins

DevOps

Overview

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



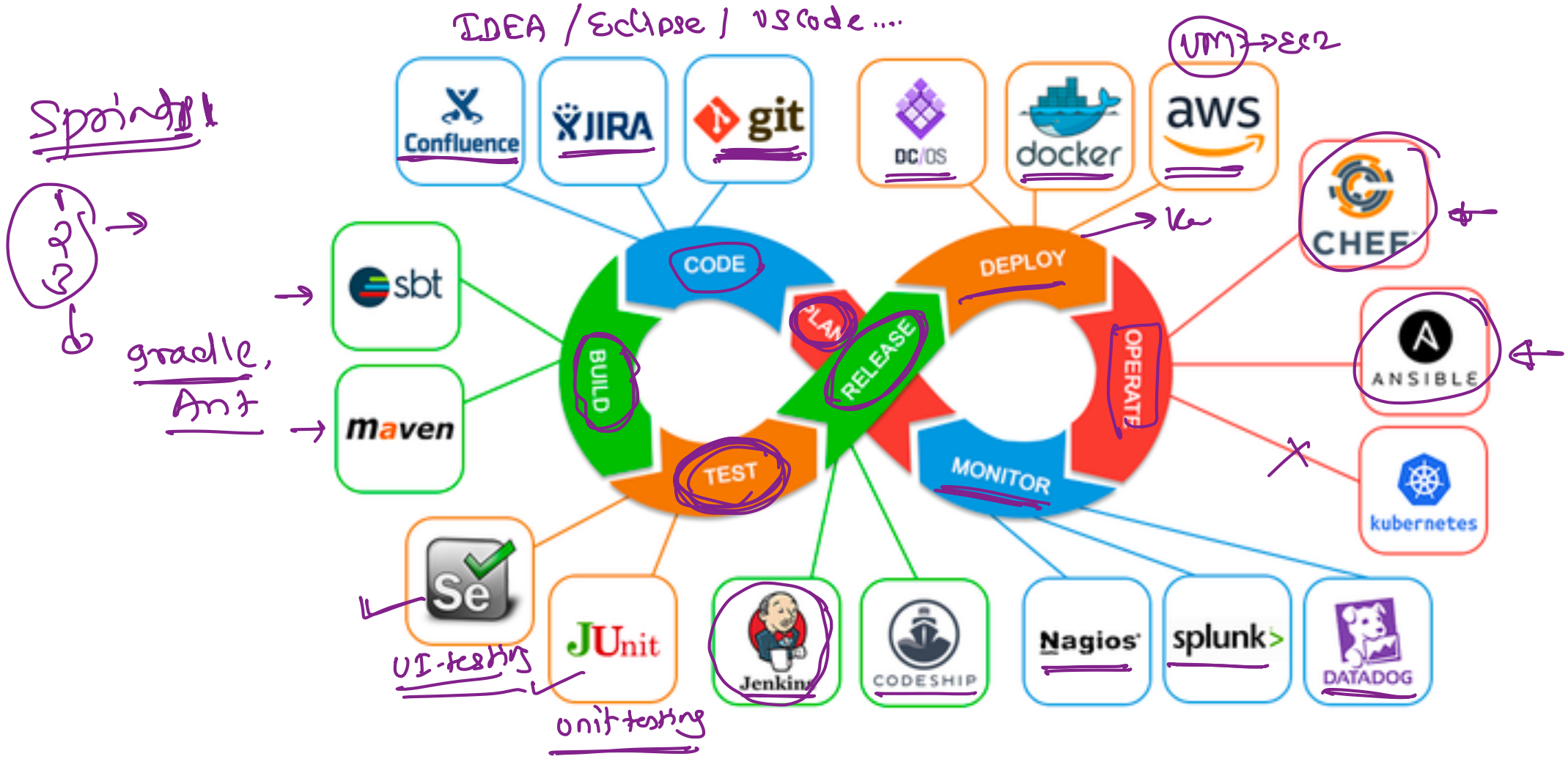
Why DevOps is Needed?

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

What is DevOps ?

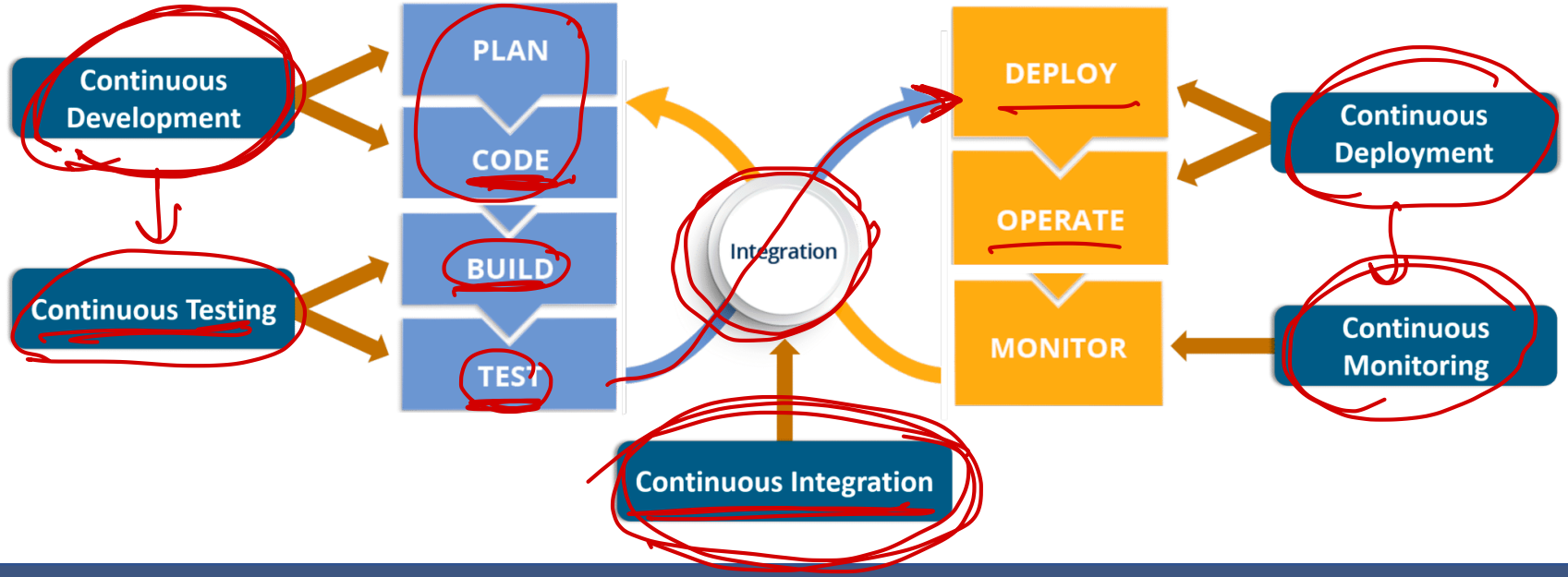
- DevOps is a never-ending process of continuous improvement
- It integrates Development and Operations teams
- It improves collaboration and productivity by
  - Automating infrastructure
  - Automating workflow
  - Continuously measuring application performance

DevOps Lifecycle



DevOps Terminologies

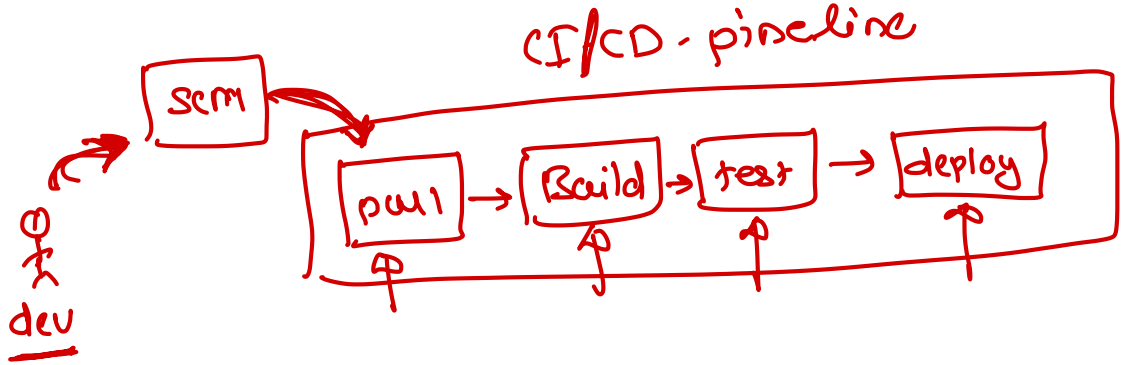
- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Continuous Monitoring



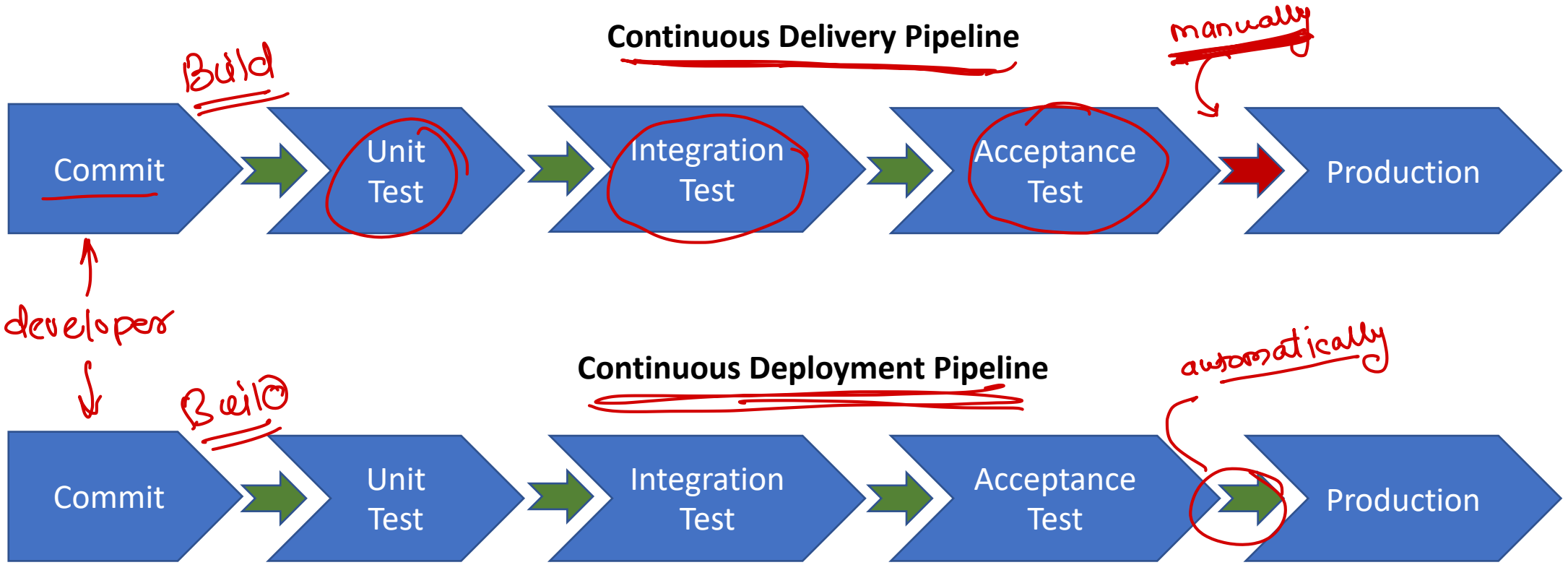
Continuous Integration

Overview

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



CI/CD Pipeline



Importance

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

Popular CI tools

Jenkins

TeamCity

Bamboo

GitLab CI

Travis CI


Jenkins

What is Jenkins ?

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

① website
② desktop app
③ mobile app
④ console app

Where is it came from ?

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

Features

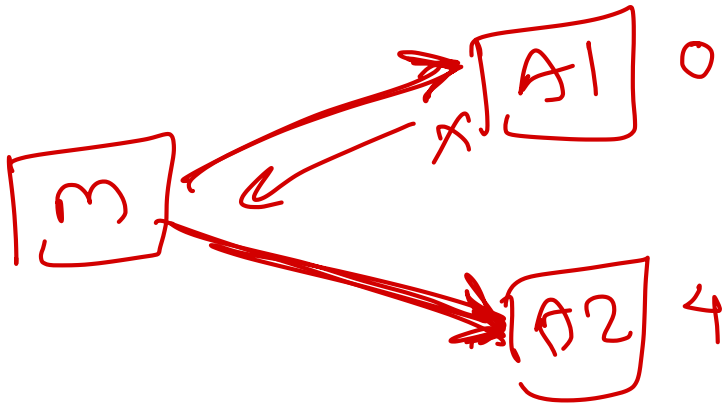
- Easy installation on different operating systems
- Supports pipelines as code that uses **domain-specific language** (**DSL**) to model application delivery pipelines as code
- Easily extensible with the use of third-party plugins
- Easy to configure the setup environment in the user interface
- Master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

Terminologies

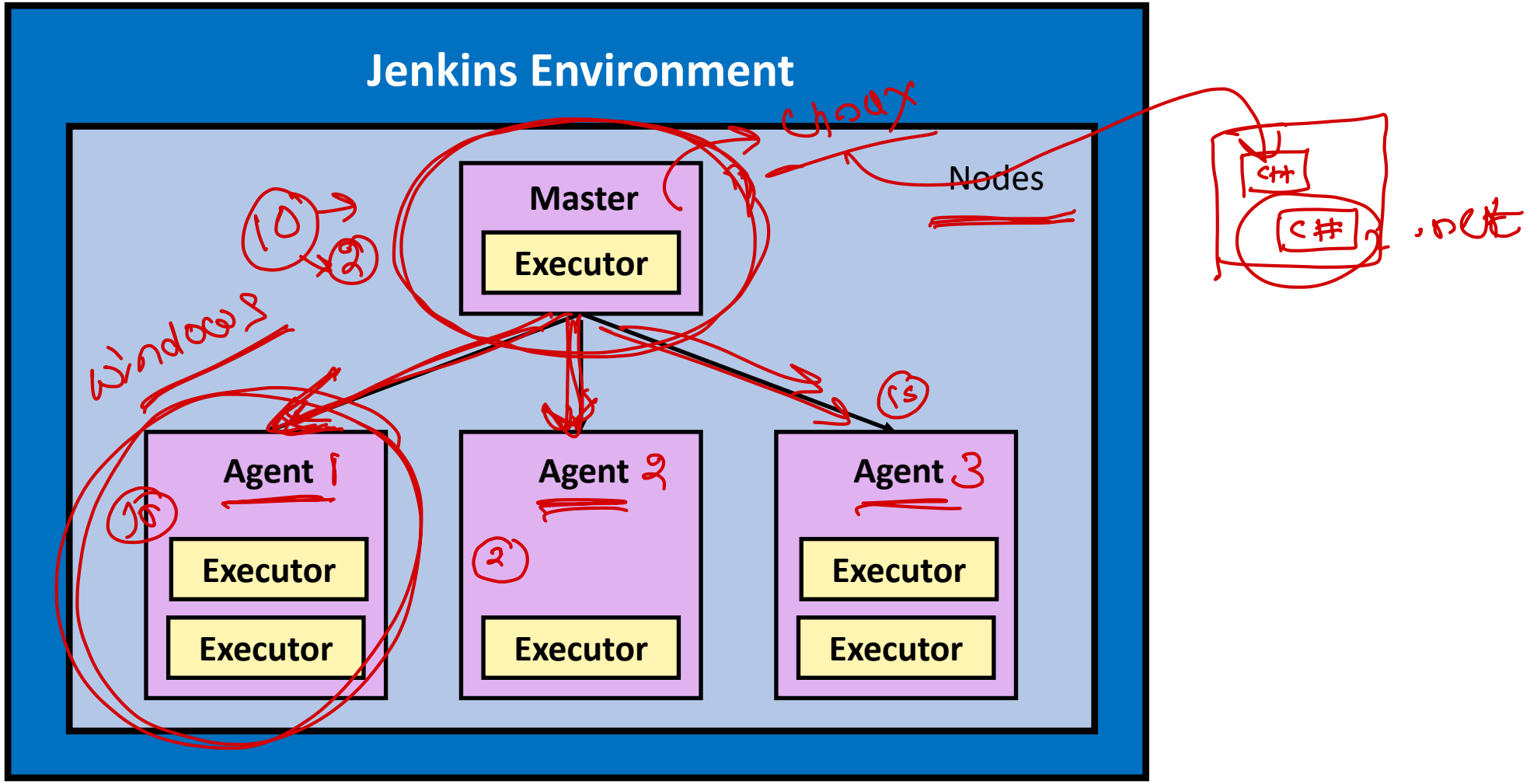
- **Node**
  - Node is the generic term that is used in Jenkins to mean any system that can run Jenkins jobs
  - This covers both masters and agents, and is sometimes used in place of those terms
  - Furthermore, a node might be a container, such as one for Docker
- **Master**
  - A Jenkins *master*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

Terminologies

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

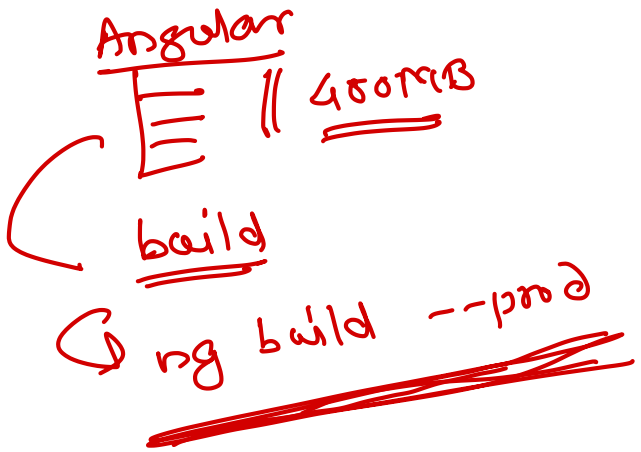


Jenkins Environment



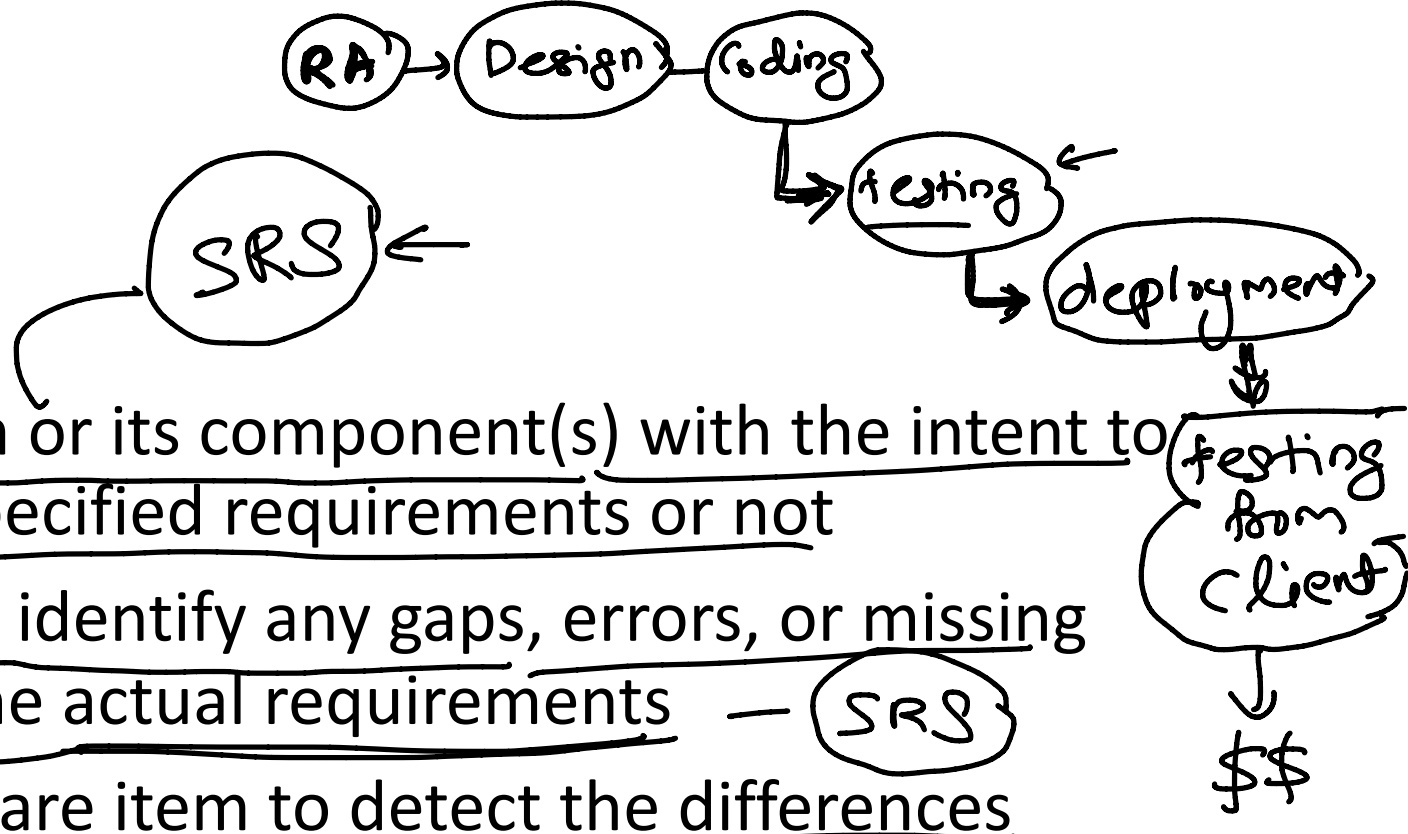
Job

- Also known as Project
- It represents the steps used to build the code
- To create a new job, use option “new item”
- Project in Jenkins has different types
  - Freestyle Project
  - Pipeline
  - Multi-configuration Project
  - Folder
  - GitHub Organization
  - Multibranch Pipeline



Application Testing

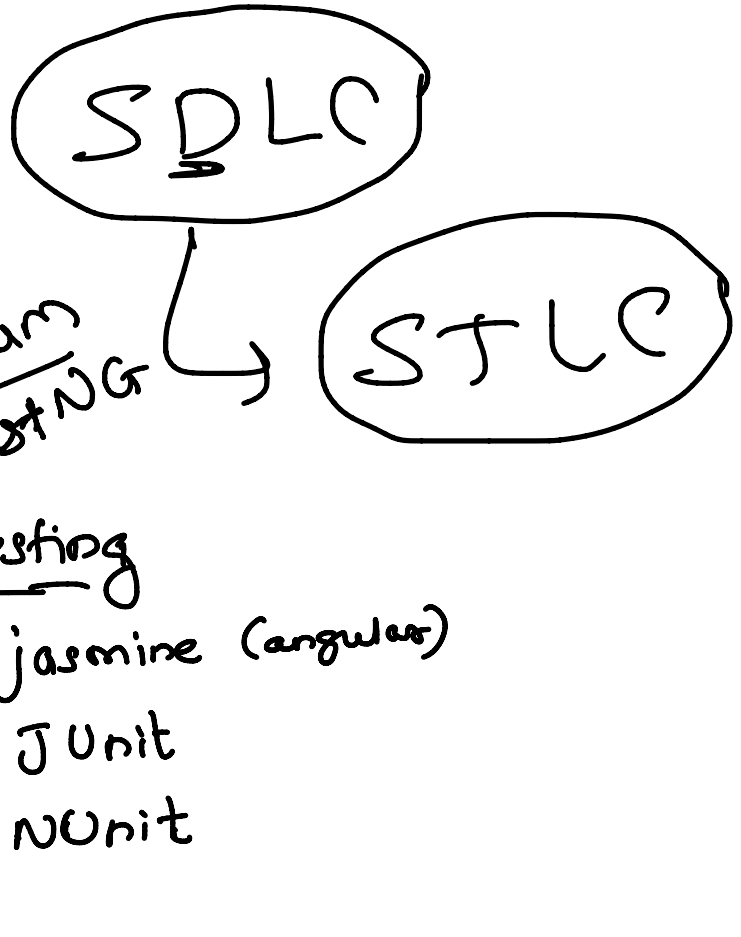
What is testing?



- process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not
- executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements
- A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item

Who does testing?

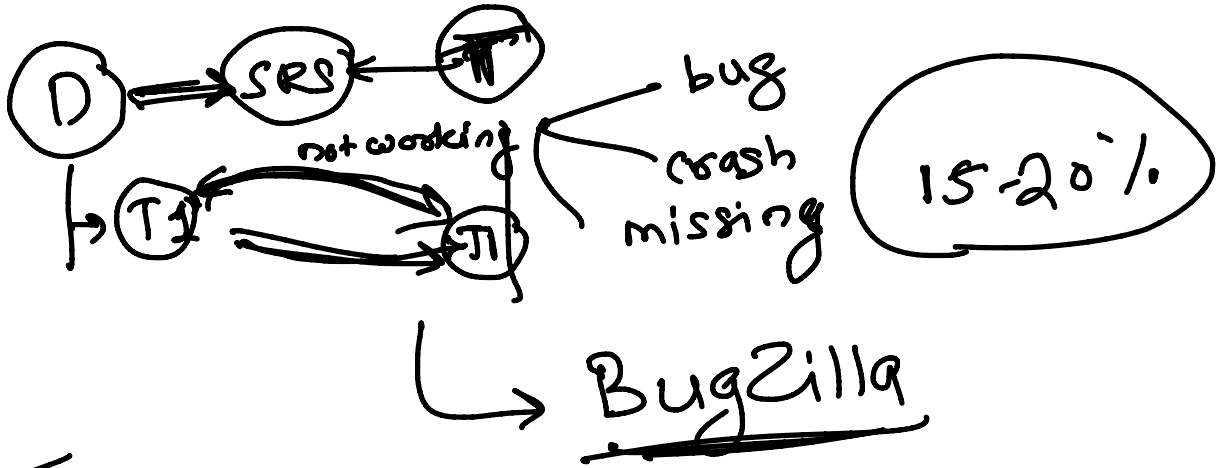
- Testing will be done by
  - Software Tester
  - Software Developer
  - Project Lead/Manager
  - End User / Client



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 (unit testing)

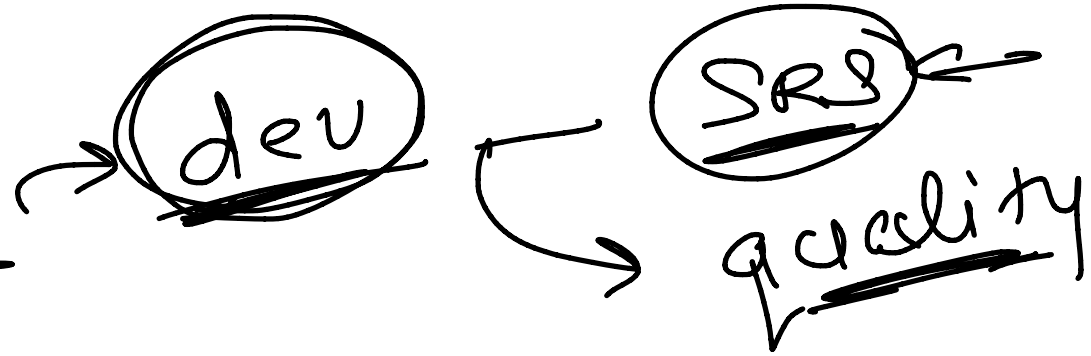
When to Stop Testing?



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

Docker

Verification



- 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

Validation → Testers

- 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

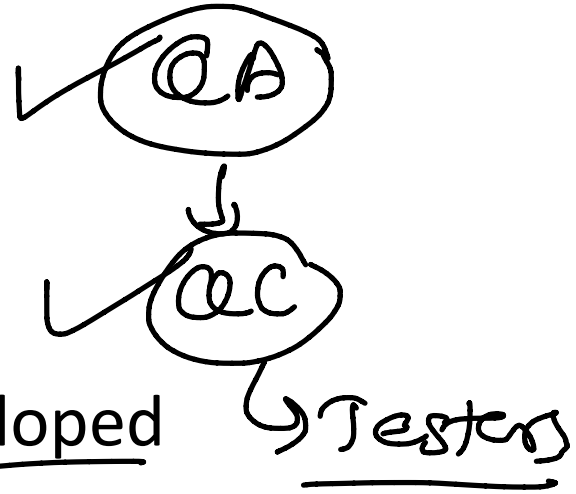
Testing, Quality Assurance and Quality Control

- Bit confusing to understand
- They are interrelated and to some extent, they can be considered as same activities
- But there exist distinguishing points that set them apart

Quality Assurance → set process

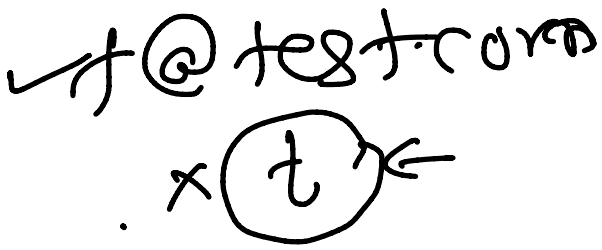
- QA includes activities that ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements
- Focuses on processes and procedures rather than conducting actual testing on the system
- Process-oriented activities
- Preventive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

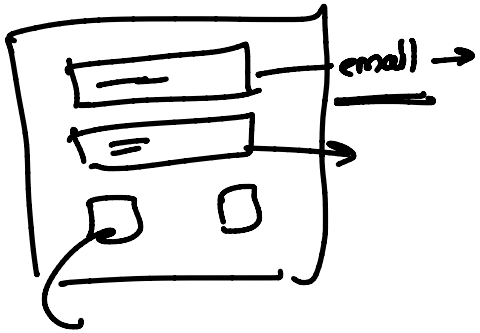
Quality Control



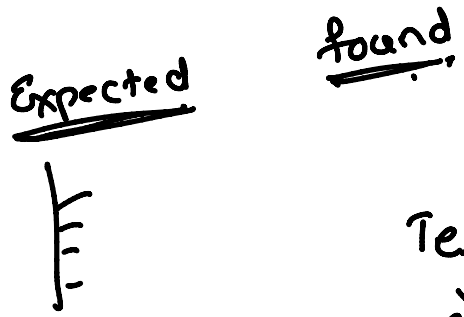
- It includes activities that ensure the verification of a developed software with respect to documented requirements
- Focuses on actual testing by executing the software with an aim to identify bug/defect through implementation of procedures and process
- Product-oriented activities
- It is a corrective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

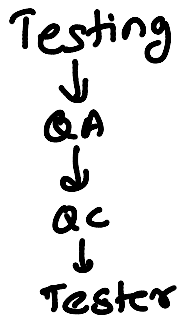
Testing





- It includes activities that ensure the identification of bugs/error/defects in a software
- Focuses on actual testing
- Product-oriented activities
- It is a preventive process
- Testing is the subset of Quality Control

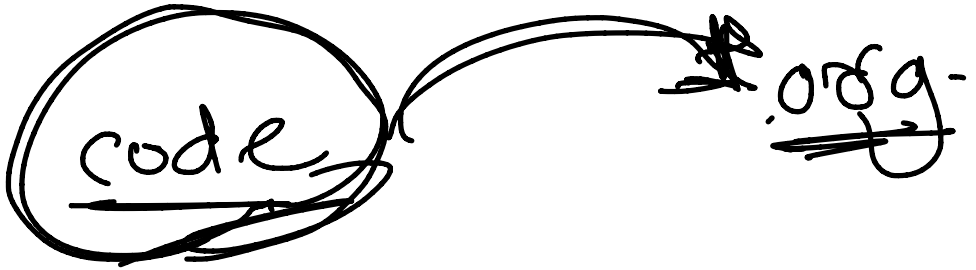




Audit

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

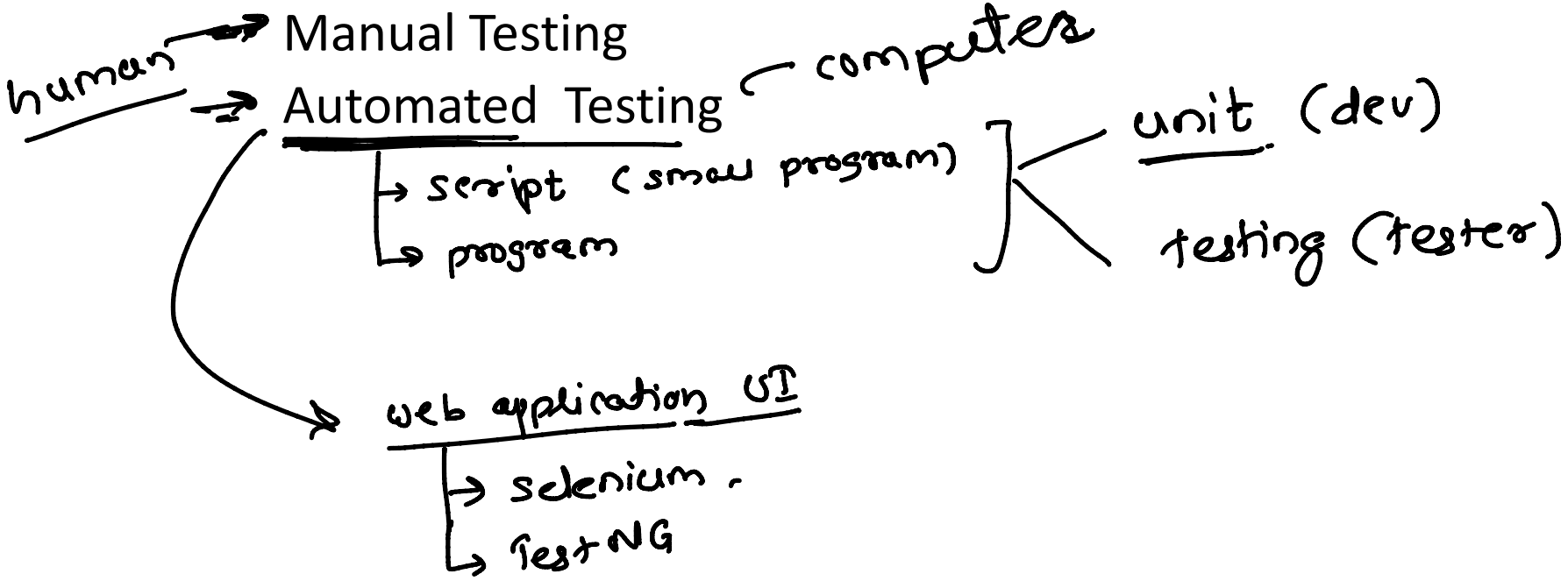
Inspection



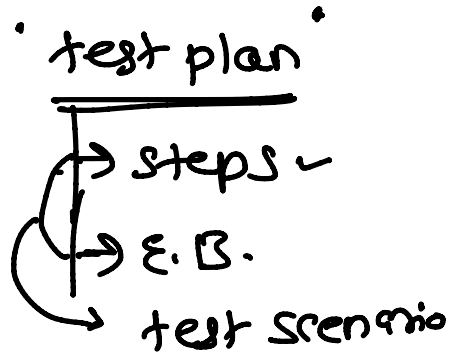
- Inspection is a formal evaluation technique in which software requirements, designs, or codes are examined in detail by a person or a group other than the author to detect faults, violations of development standards, and other problems.

Types of Testing

- Testing can be done in one of the ways
  - Manual Testing
  - Automated Testing

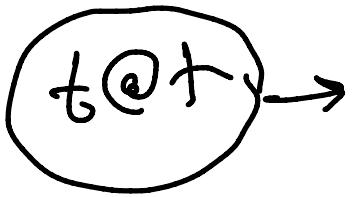


{ User is logging in } → (end-user)

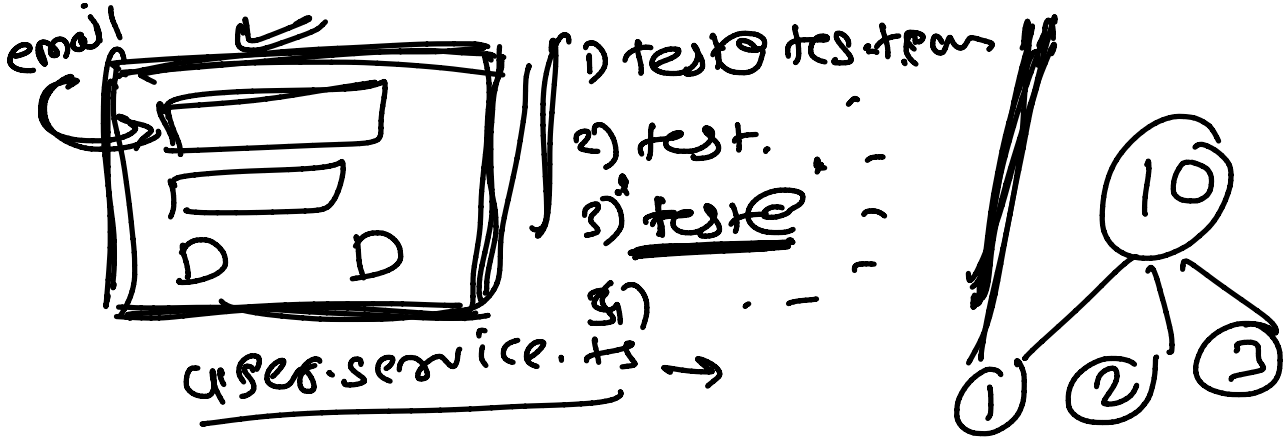


Manual Testing

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

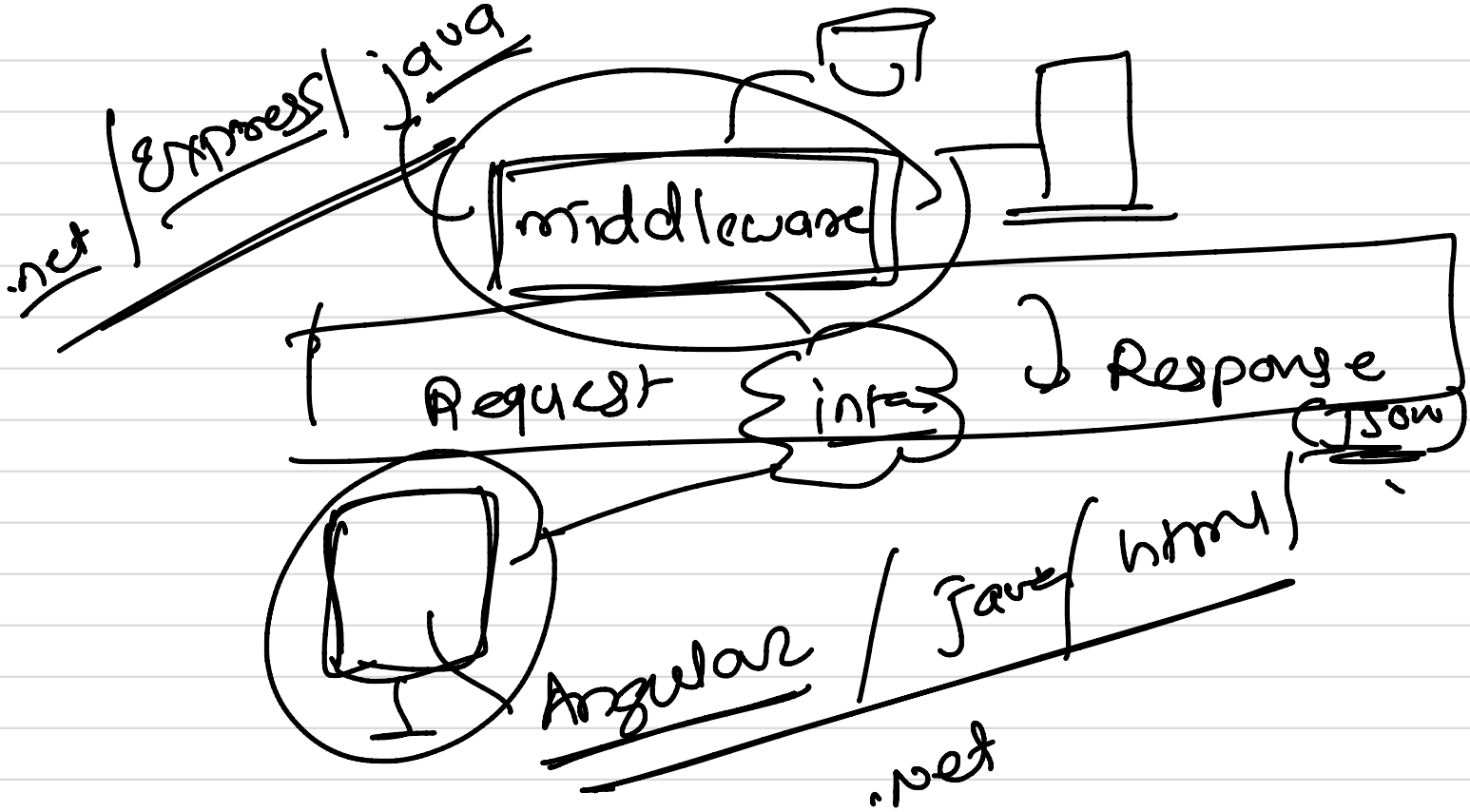


Automation Testing



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

code coverage



Request
Response

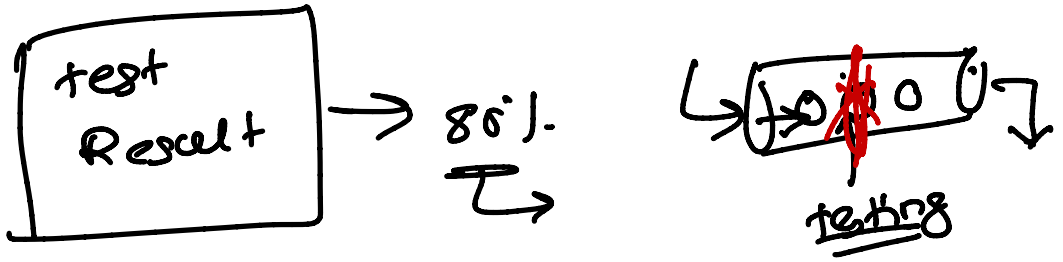
What to Automate?

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

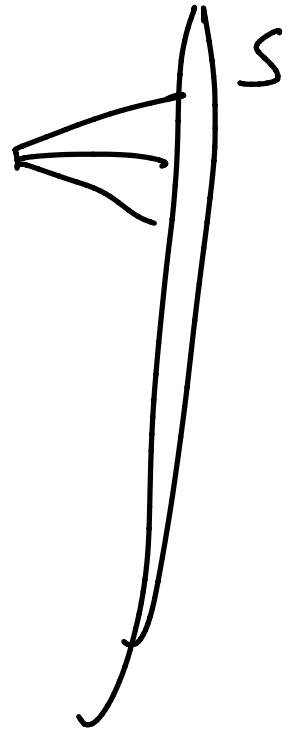
When to Automate?

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

How to Automate?



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



Software Testing Tools

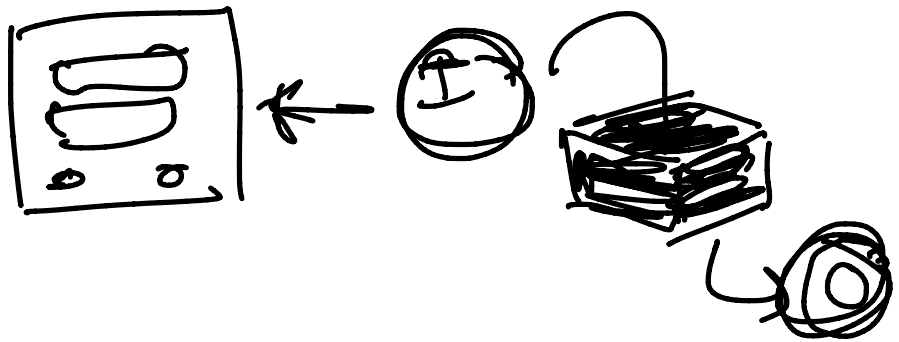
- HP Quick Test Professional
- Selenium → web UI
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR

Methods

- There are different methods that can be used for software testing
  - Black-Box Testing
  - White-Box Testing
  - Grey-Box Testing

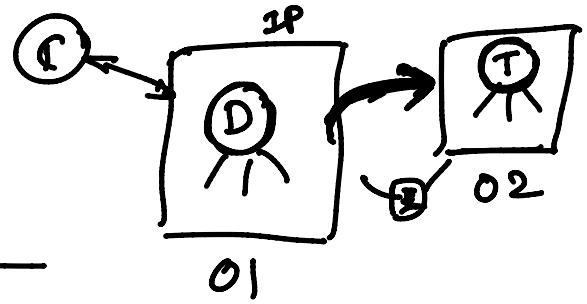
Vue.js

Black-Box Testing



- The technique of testing without having any knowledge of the interior workings of the application
- The tester is oblivious to the system architecture and does not have access to the source code
- Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon

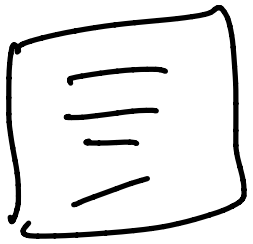
Black-Box Testing - Advantages



- 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

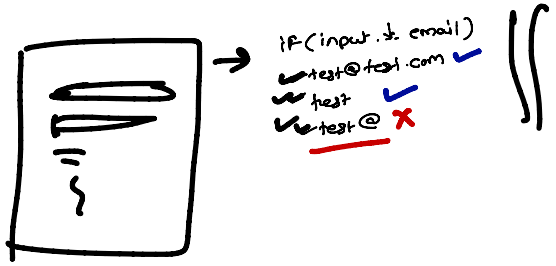
COST

Black-Box Testing - Disadvantages



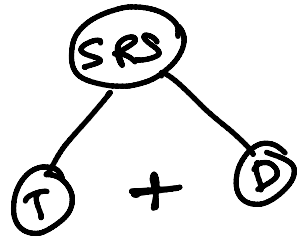
- Limited coverage, since only a selected number of test scenarios is actually performed
- Inefficient testing, due to the fact that the tester only has limited knowledge about an application
- Blind coverage, since the tester cannot target specific code segments or errorprone areas
- The test cases are difficult to design

White-Box Testing



- Detailed investigation of internal logic and structure of the code
- Is also called **glass testing** or **open-box testing**
- A tester needs to know the internal workings of the code
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately

White-Box Testing – 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

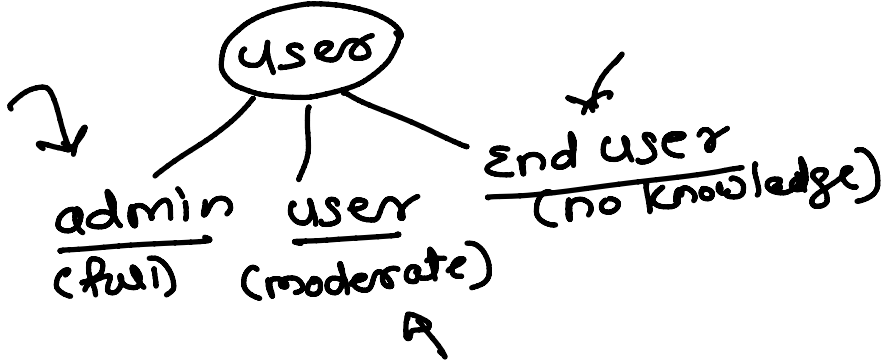
White-Box Testing – Disadvantages

- 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
- Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan

Grey-Box Testing - Advantages



- Offers combined benefits of black-box and white-box testing wherever possible
- Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications
- Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling
- The test is done from the point of view of the user and not the designer

Grey-Box Testing - Disadvantages

- Since the access to source code is not available, the ability to go over the code and test coverage is limited
- The tests can be redundant if the software designer has already run a test case
- Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested

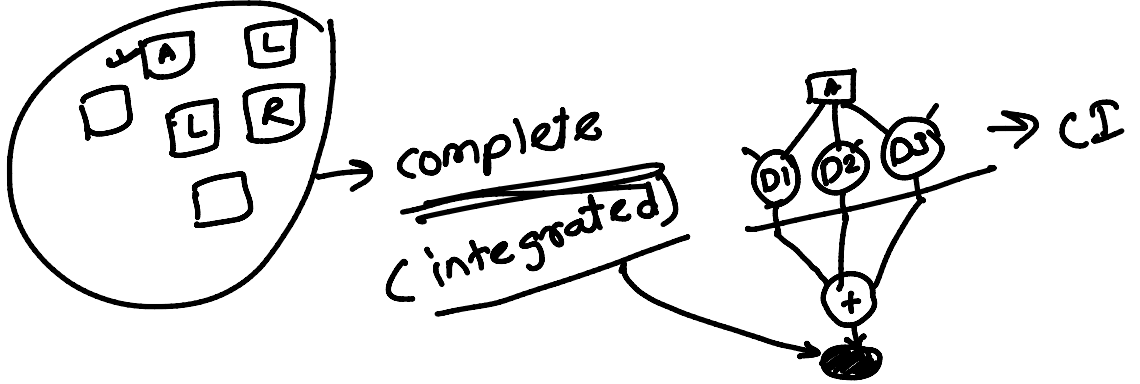
Black vs Grey vs White

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

Testing Levels

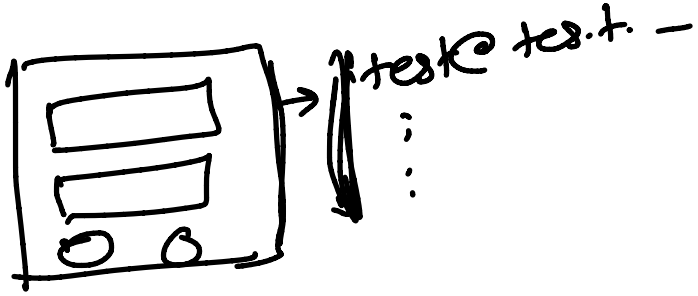
- Levels of testing include different methodologies that can be used while conducting software testing
- Levels include
  - Functional Testing
  - Non-functional Testing

Functional Testing



- This is a type of black-box testing that is based on the specifications of the software that is to be tested (SRS)
- 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

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

Functional Testing

CI / CT / CD

- Includes
  - Unit Testing
  - Integration Testing
  - System Testing
  - Regression Testing
  - Acceptance Testing
  - Alpha Testing
  - Beta Testing

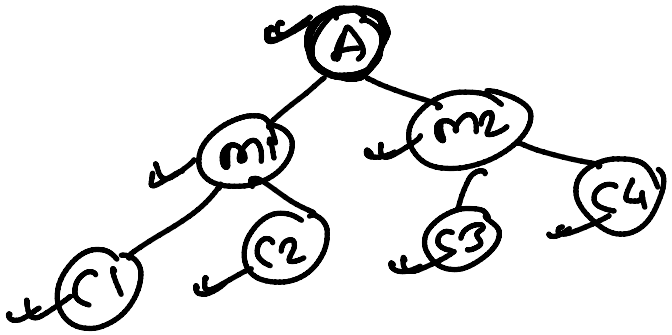
Unit Testing

- Is performed by developers before the setup is handed over to the testing team to formally execute the test cases
- Unit testing is performed by the respective developers on the individual units of source code assigned areas
- The developers use test data that is different from the test data of the quality assurance team
- The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality

Unit Testing – Limitations

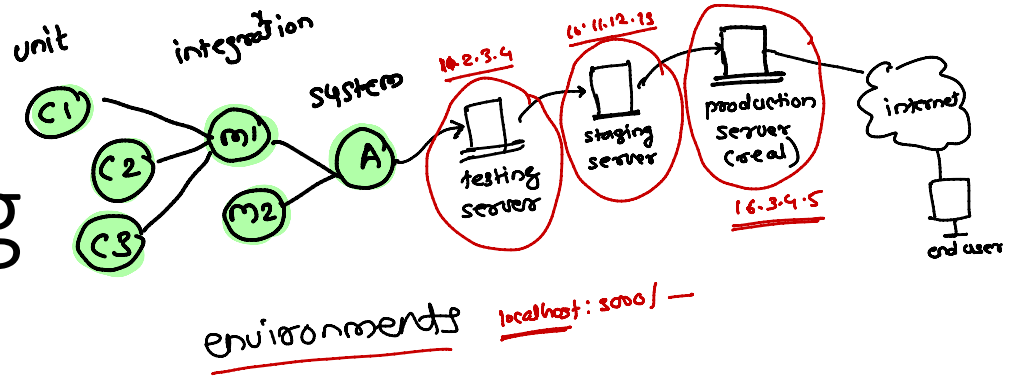
- Testing cannot catch each and every bug in an application
- It is impossible to evaluate every execution path in every software application
- There is a limit to the number of scenarios and test data that a developer can use to verify a source code
- After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units

Integration Testing



- 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 (staging) where the application will be deployed
  - System testing enables us to test, verify, and validate both the business requirements as well as the application architecture

Regression Testing

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

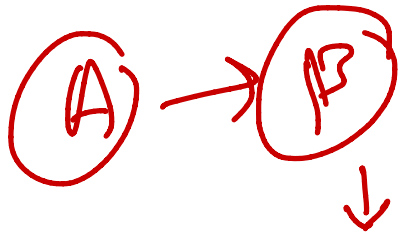
Acceptance Testing (UAT)

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

Alpha Testing

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

Beta Testing



- This test is performed after alpha testing has been successfully performed
- In beta testing, a sample of the intended audience tests the application
- Beta testing is also known as **pre-release testing**
- Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release

Beta Testing

- 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
    - stress
  - Usability Testing
  - Security Testing
    - portability

Performance Testing

- It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software
- There are different causes that contribute in lowering the performance of a software
  - Network delay
  - Client-side processing
  - Database transaction processing
  - Load balancing between servers
  - Data rendering

Performance Testing

- 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

Stress Testing

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

Usability Testing

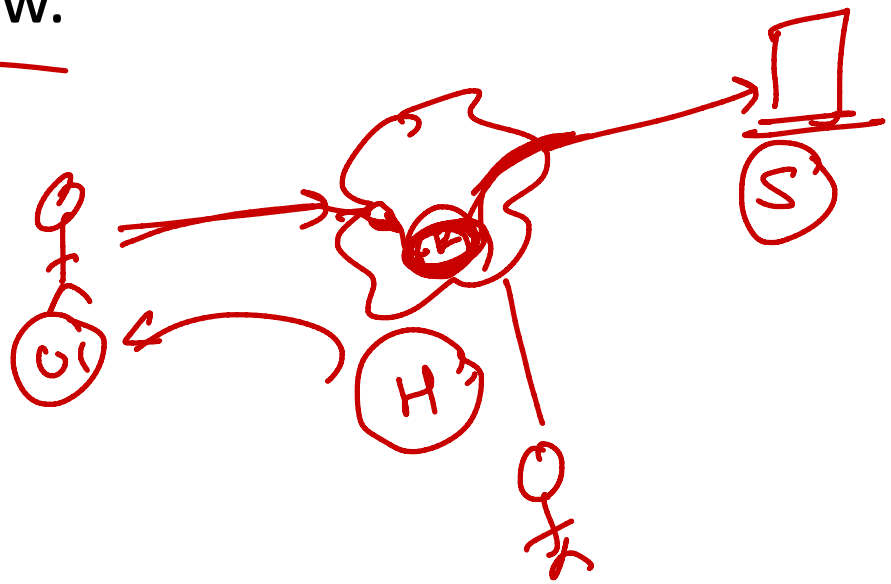
- Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation
- Can be defined as
  - efficiency of use
  - learn-ability
  - memory-ability
  - errors/safety
  - Satisfaction
- UI testing can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

Security Testing

Express → passport (token)

js → bcrypt

- Testing a software in order to identify any flaws and gaps from security and vulnerability point of view.
- Involves
  - Confidentiality
  - Integrity
  - Authentication
  - Availability
  - Authorization
  - Non-repudiation
  - Input checking and validation
  - SQL insertion attacks



email

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