DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 1

Agenda

- Class Guidelines
- Course Overview
- Textbooks and Reference books
- Introduction to DevOps
  - What is DevOps?
  - Need for DevOps
- Foundational Concepts
  - Software Development Life Cycle
  - Process Models (before Agile)
  - Agile Process Model
  - Agile Methodologies – Scrum, Extreme Programming
  - TDD, FDD and BDD in Agile context

Class Guidelines

- Overall class duration is 2 hours that is 1:30 PM to 3:30 PM with 5 minutes given at the start as buffer to join.
- Lecture will start at 1:35 PM sharp!
- There will be a break given in between for 15 minutes so we shall have two parts to the lecture:

Time slot	Session	Duration
1:35 PM – 2:25 PM	Part 1	50mins
2:25 PM – 2:40 PM	Break	15mins
2:40 PM – 3:30 PM	Part 2	50mins

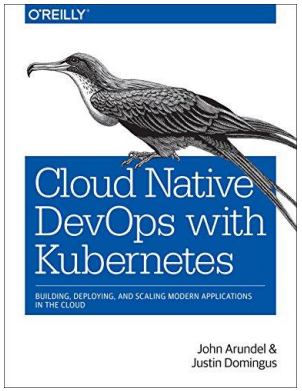
- Please keep the session interactive and ask any doubts during the lecture in chat and respond to the questions being asked.
- Do not spam the chat as it can cause hinderance to your batch mates who are trying to focus on the lecture

Course Overview

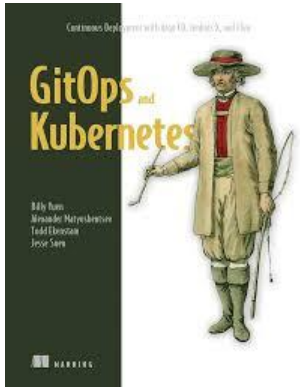
1. Foundational Concepts
2. Introduction to DevOps
3. Cloud Native Application
4. Source Code Management (Using GIT as an example tool)
5. Continuous Integration
6. Continuous Delivery using GitOps
7. Kubernetes
8. Continuous Deployment
9. IaC and Serverless CI/CD
10. Security in the DevOps lifecycle
11. Observability and Continuous Monitoring
12. MLOps
13. DataOps
14. Future trends in Cloud DevOps and Course Review

Textbooks

T1: Cloud Native DevOps with Kubernetes: Building, Deploying and Scaling Modern applications in the Cloud'' by John Arundel and Justin Domingus. Publisher: O'Reilly, 2019.

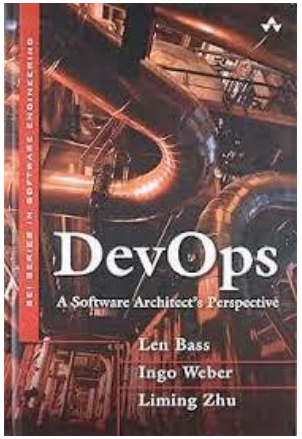


T2: GitOps and Kubernetes – Continuous Deployment with ArgoCD, Jenkins X and Flux'', by Billy Yuen et al. Publisher: Manning, 2021

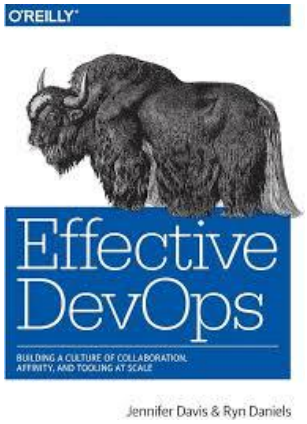


Reference books

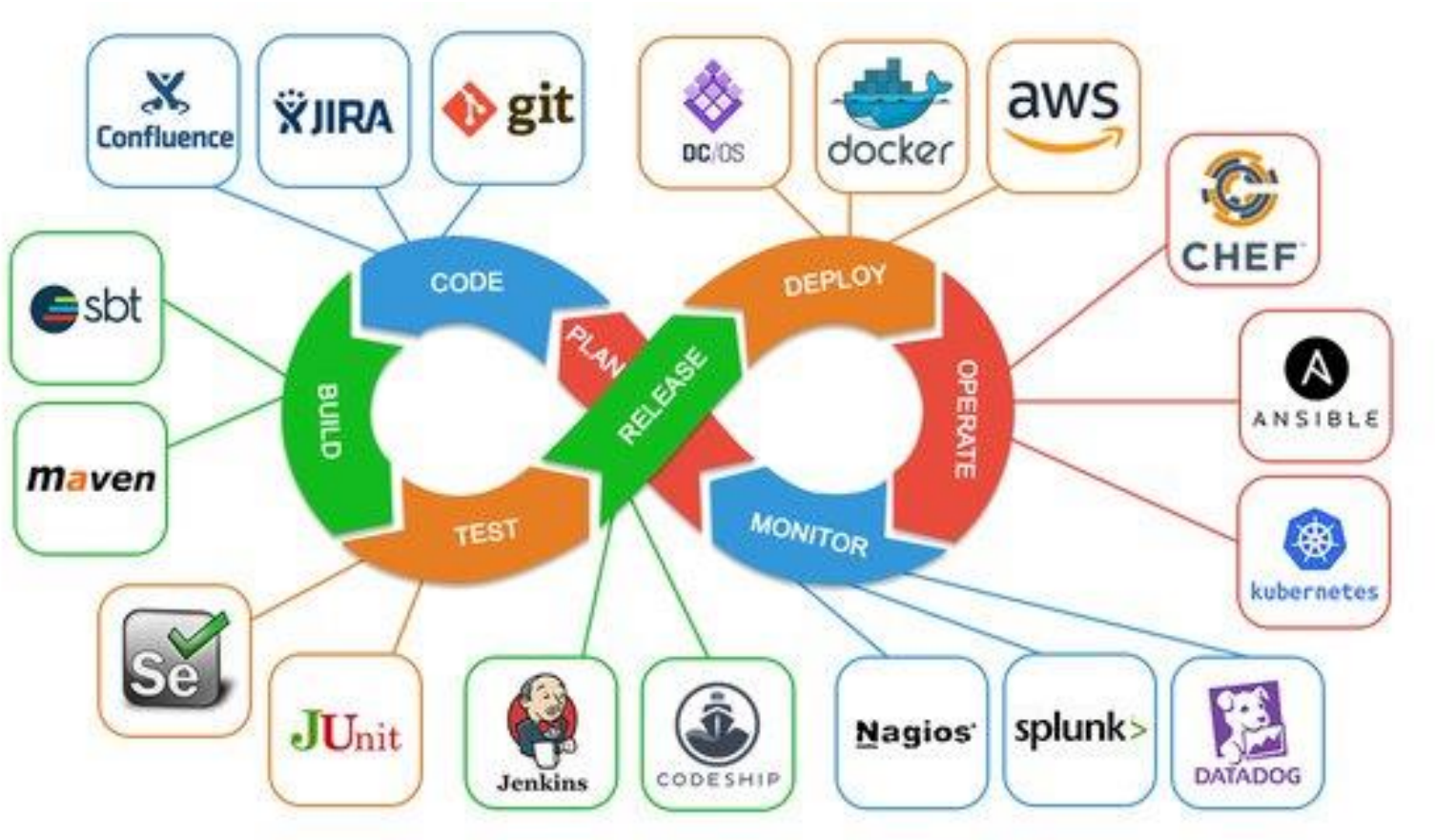
R1: DevOps: A Software Architect's Perspective (SEI Series in Software Engineering)''by Len Bass, Ingo Weber, Liming Zhu. Publisher: Addison Wesley, 2015.



R2: Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale'' by Jennifer Davis , Ryn Daniels. Publisher: O'Reilly Media, June 2016

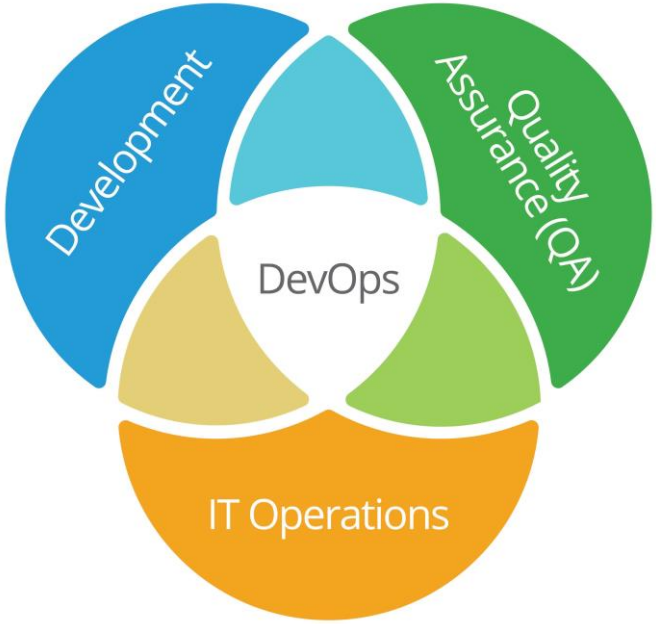


What is DevOps?



What is DevOps?

- DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).
- It aims to shorten the software development life cycle and provide continuous delivery with high software quality.



Key aspects of DevOps

- **Collaboration and Communication:**

Breaking down silos between development and operations teams.

- **Automation:**

Automating repetitive tasks such as code integration, testing, and deployment.

- **Continuous Integration and Continuous Deployment (CI/CD):**

Frequently integrating code changes and deploying them to production.

- **Monitoring and Logging:**

Continuously monitoring applications and infrastructure to detect and respond to issues quickly.

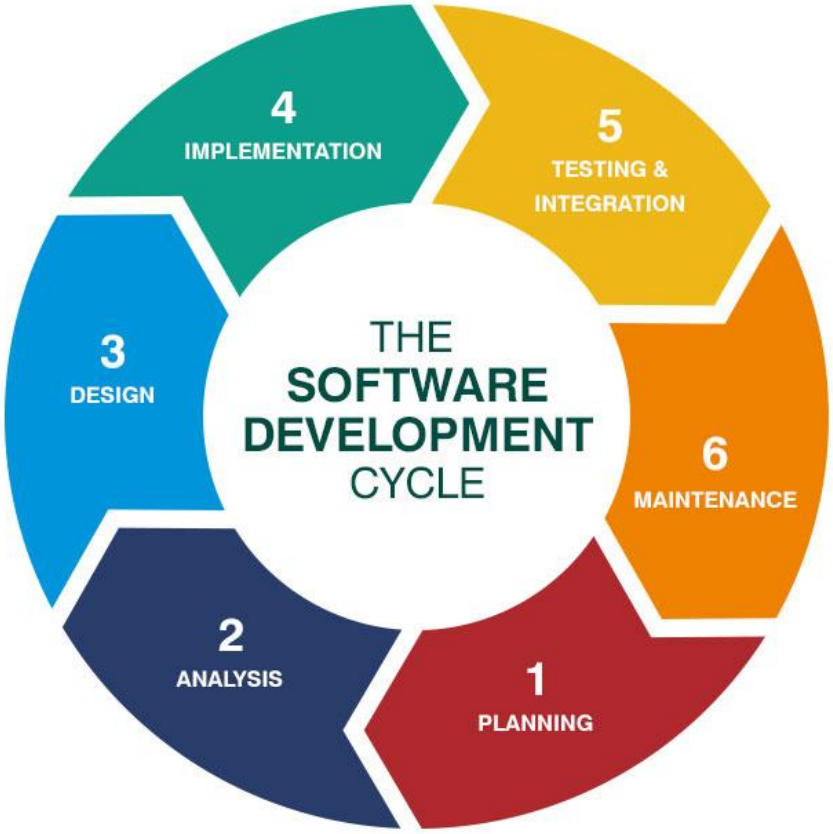
Need for DevOps

Traditional software development and IT operations often worked in silos, leading to inefficiencies and delays. The need for DevOps arose to address the following challenges:

- Slow software delivery due to lack of collaboration.
- Inefficiencies in managing infrastructure and code deployment.
- Difficulty in maintaining consistent environments across development, testing, and production.
- Slow response to changing business requirements and customer feedback.

Foundational Concepts – SDLC

- The Software Development Life Cycle (SDLC) is a systematic process for developing software through a series of phases.
- It ensures that the software meets quality standards and customer requirements.
- It typically has 6 phases.



Phases of SDLC

1) Planning:

- Identify the scope, purpose, and feasibility of the project.

2) Requirements Analysis:

- Gather and document functional and non-functional requirements.

3) Design:

- Create architectural and detailed design specifications.

4) Development (Coding):

- Write the code based on the design documents.
- Small teams , Limited co-ordination.
- Unit tests

Phases of SDLC – Contd

5a) Testing:

- Verify that the software works as intended and fix any issues.

5b) Integration and Deployment:

- Release the software to the production environment.

6) Maintenance:

- Provide ongoing support and make necessary updates.
- Responding to failures

Phases of SDLC in the context of Devops

1) **Planning:**

- Identify the scope, purpose, and feasibility of the project.

2) **Requirements Analysis:**

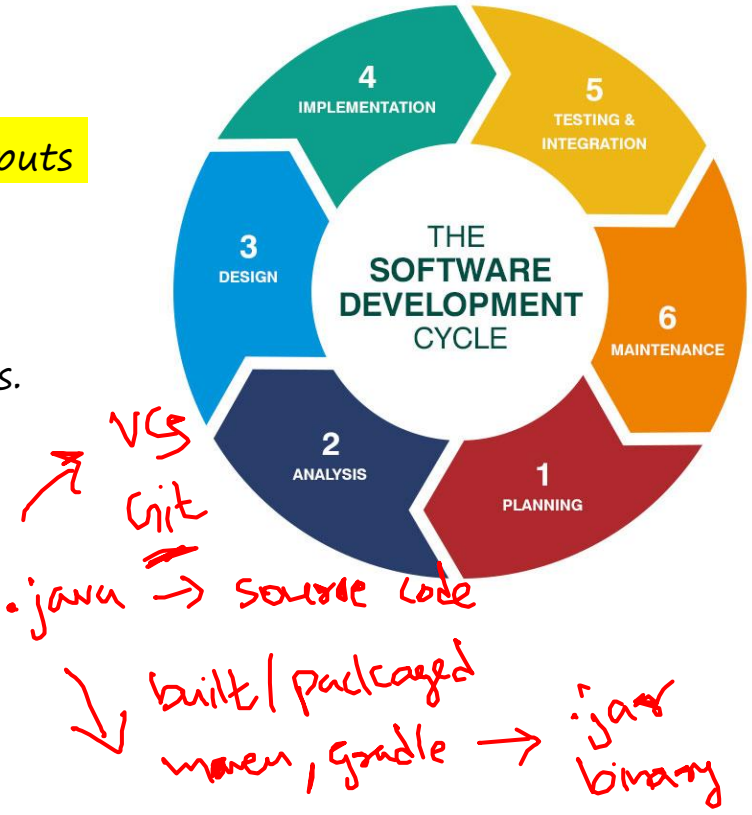
- Gather and document functional and non-functional requirements.
- Treat Ops as first-class stakeholders and get their inputs too!

3) **Design:**

- Create architectural and detailed design specifications.

4) **Development (Coding):**

- Write the code based on the design documents.
- Small teams , Limited co-ordination.
- Unit tests
- Build tools to support Continuous Integration!



Phases of SDLC in the context of Devops - Contd

innovate achieve lead

5a) Testing:

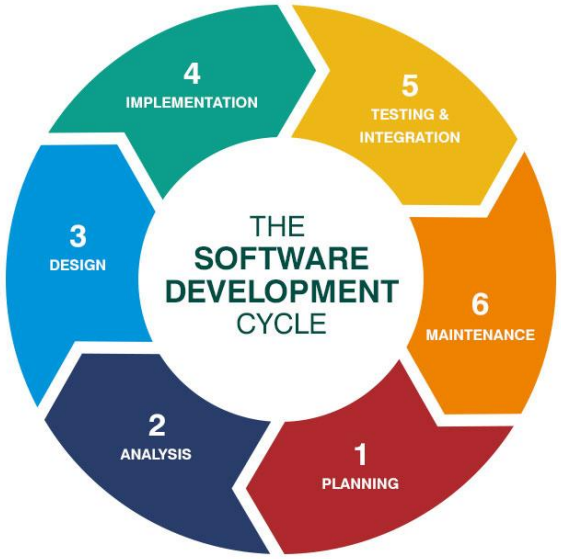
- Verify that the software works as intended and fix any issues.
- promote Automated tests to be reliable and repeatable

5b) Integration and Deployment:

- Release the software to the production environment.
- Build and Support Continuous Deployment.

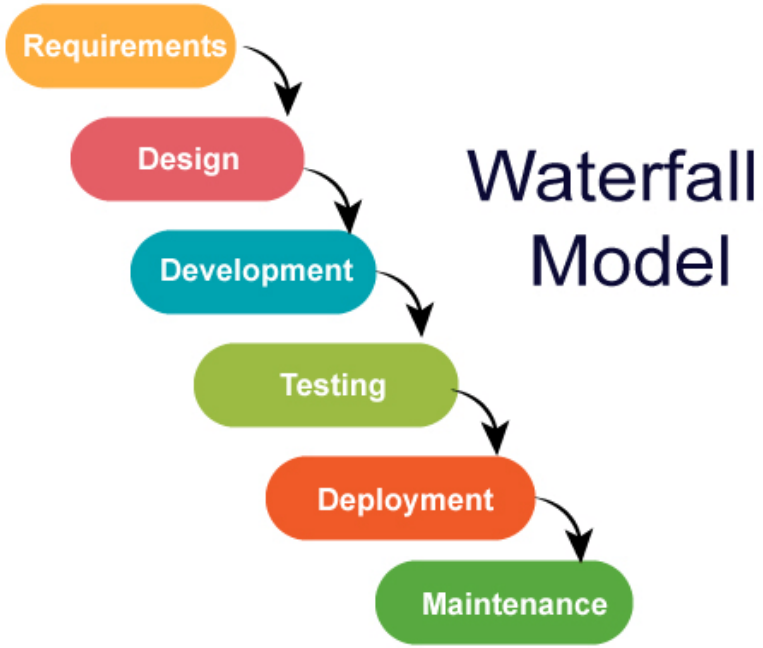
6) Maintenance:

- Provide ongoing support and make necessary updates.
- Responding to failures
- Monitoring



Process Models – Waterfall Model

- The Waterfall Model is one of the earliest and most straightforward approaches to software development.
- The term was first introduced in a paper published in 1970 by Dr. Winston W. Royce
- Waterfall Model is very simple to understand, use and has distinct endpoints or goals at each phase.



Process Models – Waterfall Model – Cycle – 1

innovate achieve lead

1a) Requirement Gathering:

In this phase, business analyst and project manager participate in the meetings with client to gather the requirements.

Outcome:

Business requirements / BRS.

1b) Analysis

Business Analyst (BA) converts the business requirements into technical requirements with the help senior team members like SMEs (Subject-matter experts) and team leads.

Outcome:

Technical Requirements SRS (Software Requirement Specification)

Process Models - Waterfall Model – Cycle – 2,3

innovate achieve lead

2) Development or Implementation Phase:

In this phase, Developers start programming with the code by following organization coding standards.

Outcome:

Source Code Document (SCD) and developed product.

3) Testing

- In this phase, the code gets tested to check whether it is working as expected or not. The developer performs the initial testing that are unit testing (UT) and/or Application Integration Testing (AIT) before handover the code to the testing team

- testing team follows the BRS document to verify the new changes or working fine or not. If anything not working, testing team raises the defect and development team has to fix it before the specified time.

Outcome:

Defect free Product and certified artifacts

Process Models - Waterfall Model – Cycle - 4

innovate achieve lead

4) Deployment Phase:

- Once the software is fully tested and has no defects or errors, then the test results and artifacts are reviewed by the client and provides approval for deployment.

- Once the software got deployed to production, then the new functionality available to the end-users who are currently using the system.

Outcome:

Usable product available to end-users

Process Models - Waterfall Model – Cycle - 5

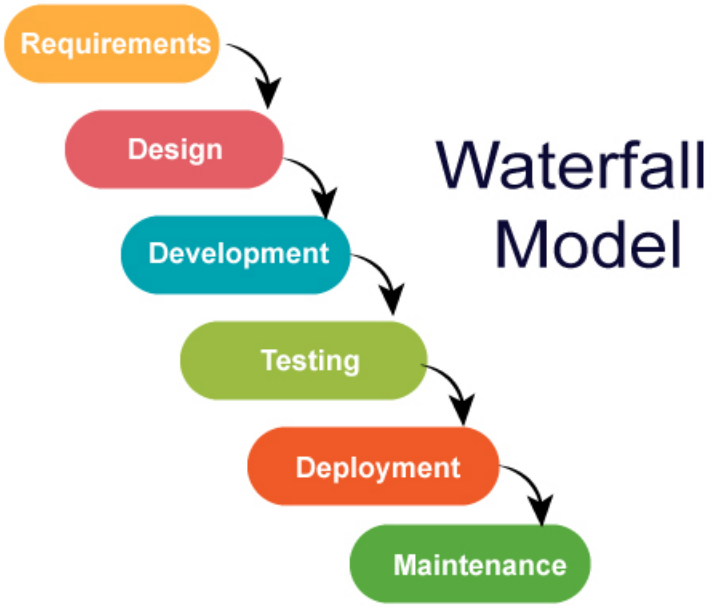
innovate achieve lead

5) Maintenance Phase:

- Once the end-user starts using the newly deployed software, there might be a possibility that the real-time issues starts coming up.
- The team has to fix these issues to avoid the loss in business if the issue has less priority or less impact.
- If the issue has high priority and has huge impact, client can take a decision to roll out or backout new changes and refine the functionalities as required.

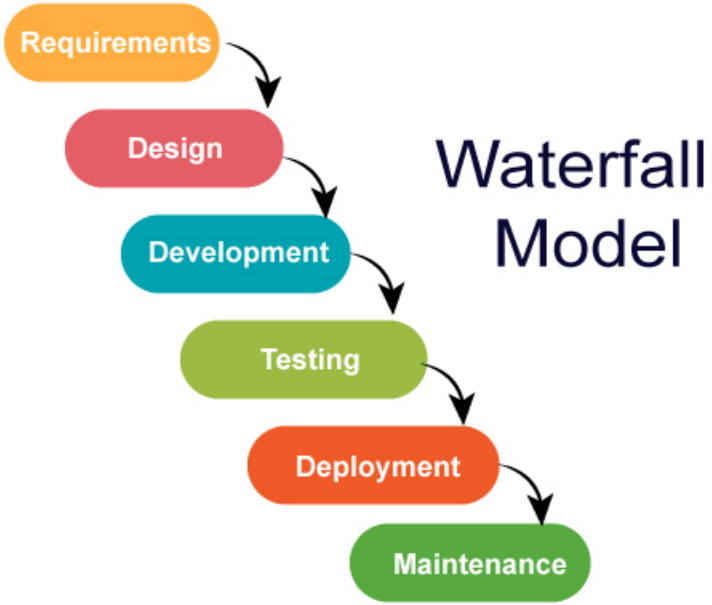
When to use Waterfall Model?

- All the requirements are clearly defined at the beginning.
- Client doesn't want to involve more in development and reviews only output.
- Working technology is clearly understandable.
- A linear and sequential approach where each phase must be completed before moving to the next.



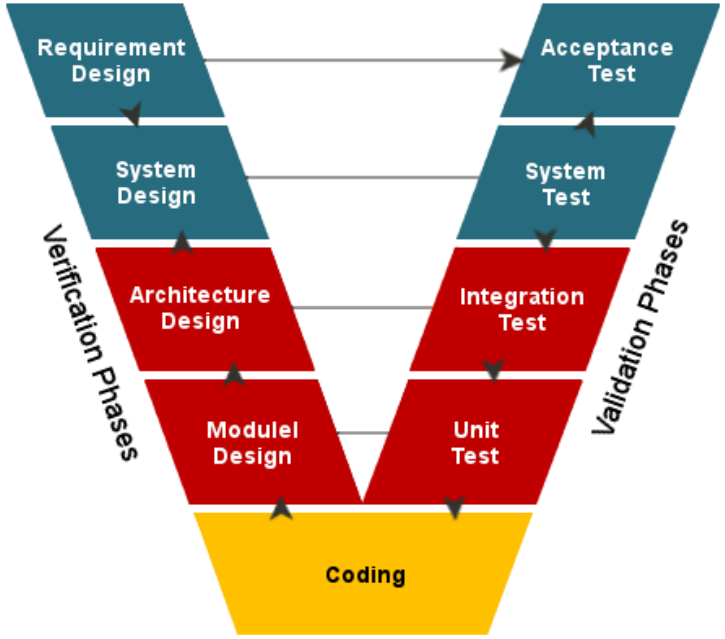
Disadvantages of Waterfall Model

- No feedback path until the end of the project.
- Measuring the progress for every stage is a real difficult.
- If the application requires some requirement changes that are found during the testing phase, it is not easy to go back and fix it
- No intermediate deliveries until the deployment of full product.
- It is inflexible, making it difficult to accommodate changes and needs Longer delivery time.



Process Models– V-Model

- V Stands for **Validation and Verification Model** and emphasizes verification and validation at each stage.
- An extension of the Waterfall Model and so is a linear model too.
- This integration of testing throughout the lifecycle is the primary distinction between the V-Model and the Waterfall Model.
- **Verification:**
- Involves in static analysis with code execution
- The evolution procedure carried out during the development to verify all the requirements covers in coding or not.
- **Validation:**
- Involves dynamic analysis (both functional and non-functional) and testing is performed by code execution.



Iterative Model

- Develops the software in repeated cycles (iterations), allowing for incremental improvements and is non linear in nature.
- Each iteration builds on the previous one.
- Feedback from each iteration is used to improve the next cycle.
- Requires more planning and management effort.

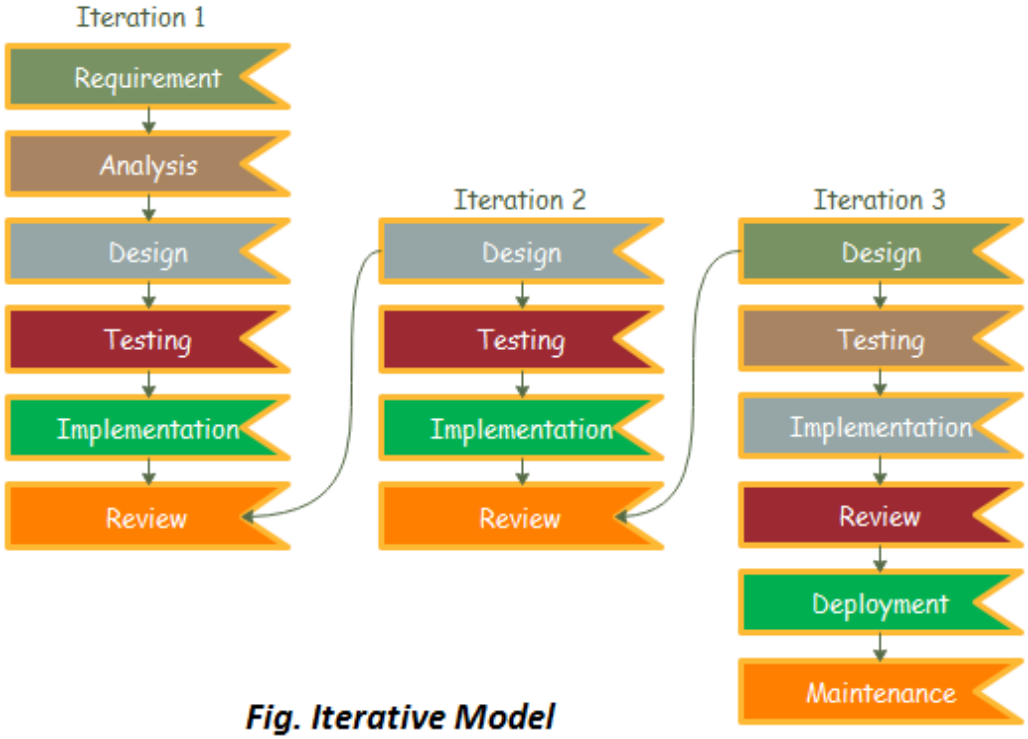
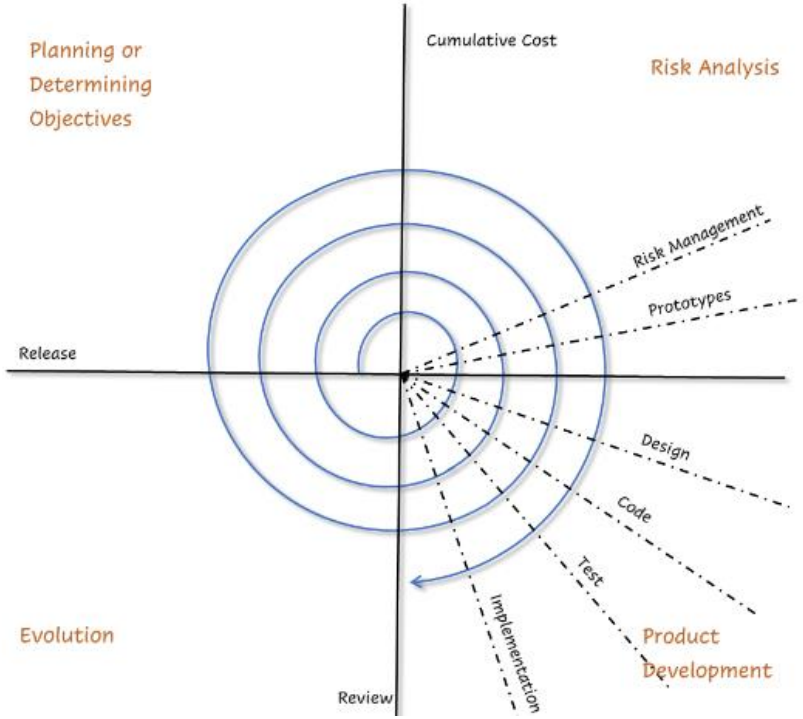


Fig. Iterative Model

Spiral Model

- A non-linear model that combines iterative development with risk management.
- Integrates the iterative model approach with waterfall model.
- Each iteration (or spiral) involves planning, risk analysis, engineering, and evaluation.
- Uses a risk-driven approach and emphasizes iterative development and refinement of prototypes.



TDD in Agile Context

- TDD stands for **Test-Driven Development.**
- A development process where tests are written before the code.
- It follows a cycle of writing a test, writing code to pass the test, and refactoring.
- TDD ensures high code quality and reduces bugs.

TDD Cycle

1) Write a test for a new feature.

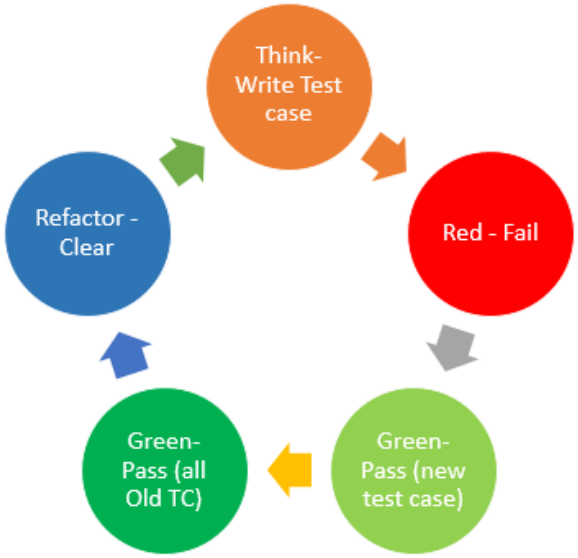
2) Run the test and see it fail (because the feature is not implemented yet).

3) Write the minimum amount of code required to pass the test.

4) Run the test again and see it pass.

5) Refactor the code to improve its structure while ensuring the test still passes.

6) Repeat the cycle for the next feature.



TDD – Example

- **Feature:** Imagine you are developing a simple calculator with a function to add two numbers.

1) What is a test case you might write?

```
def test_add():
    assert add(2, 3) == 5
```

2) Run the test and see it fail:

The add function does not exist yet, so the test fails.

3) Implement the feature (write code):

```
def add(a, b):
    return a + b
```

4) Run the test again:

The test passes.

5) Refactor (if necessary):

Ensure the code is clean and follows best practices.

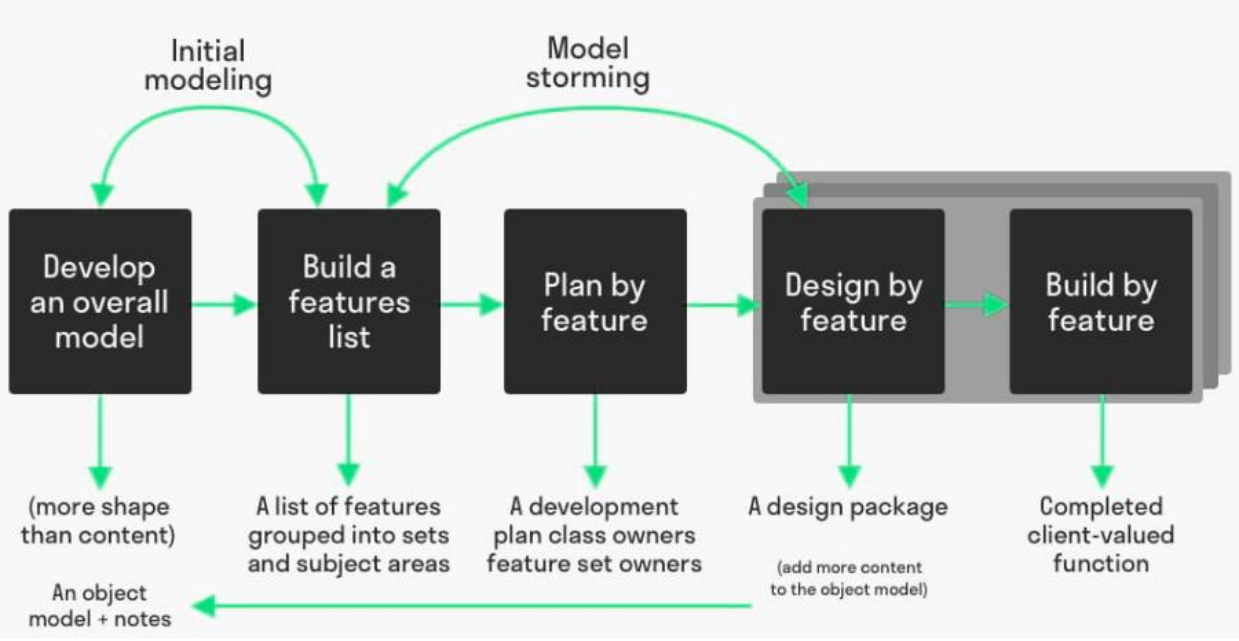
6) Repeat for the next feature:

FDD in Agile Context

- FDD stands for **Feature-Driven Development**
- An iterative and incremental software development process focusing on delivering tangible, working software features regularly..
- It involves creating a list of features, planning by feature, designing by feature, and building by feature.

FDD Cycle

1) Develop an overall model.
2) Build a feature list.
3) Plan by feature.
4) Design by feature.
5) Build by feature.



FDD – Example

- Consider a project to develop an e-commerce application.

1) Develop an overall model:

Create a high-level design of the e-commerce system, including key entities like Product, User, Order, etc.

2) Build a features list:

Identify features

1) User Registration,
2) Product Search
3) Shopping Cart
4) Checkout

3) Plan by feature:

Prioritize the features and create a plan for implementing them.

4) Design by feature:

For the **User Registration** feature, design the necessary components like user interface, backend services, and database schema..

5) Build by feature:

Implement the **User Registration** feature, test it, and ensure it is fully functional before moving to the next feature.:

BDD in Agile Context

- BDD stands for **Behavior-Driven Development**
- An extension of TDD that encourages collaboration between developers, QA, and non-technical stakeholders.
- It uses natural language descriptions of the software's behavior, typically written in the format of Given-When-Then scenarios.
- BDD ensures that all team members understand the requirements and that the software meets business needs.

BDD Cycle

1) Define scenarios using a natural language format.
2) Write tests based on these scenarios.
3) Implement the code to pass the tests.
4) Run the tests and see them pass.
5) Refactor the code while ensuring the tests still pass.

BDD – Example

- **Feature:** User Login

1) Define Scenarios:

EX1: Successful login with valid credentials

Given the user is on the login page

When the user enters a valid username and password and the user clicks the login button

Then the user should be redirected to the dashboard page

3) Implement the code:

Write the necessary code to handle user login.

4) Run the tests:

Ensure the tests pass.

5) Refactor the code:

Improve the code structure while ensuring the tests still pass.

2) Write tests:

Using a BDD framework like Cucumber, write tests based on the scenario.

```
@Given("the user is on the login page")
public void the_user_is_on_the_login_page() {
    // Navigate to login page
}

@When("the user enters a valid username and password")
public void the_user_enters_valid_credentials() {
    // Enter username and password
}

@When("the user clicks the login button")
public void the_user_clicks_login() {
    // Click login button
}

@Then("the user should be redirected to the dashboard page")
public void the_user_should_see_dashboard() {
    // Check redirection to dashboard
}
```

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 2

Recap

- Foundational Concepts
  - Software Development Life Cycle
  - Process Models (before Agile)
    - Waterfall Model
    - Iterative Model
    - Spiral Model
- Introduction to DevOps
  - Need for DevOps
  - What is DevOps

Agenda for today's class

- Agile Methodologies
  - Scrum, Extreme Programming
  - TDD, FDD and BDD in Agile context
- Three Dimensions of DevOps:
  - People, Process, Tools
  - Key DevOps Practices – CI, CT, CD, CM
- Agile Methodology for DevOps
- Principles of Software Delivery
- Version control system and its types
- Introduction to GIT

Agile Process Model

- Agile is the name given to a group of software development methodologies that are designed to be more lightweight and flexible than previous methods such as waterfall.

- Agile is an iterative and incremental approach that emphasizes flexibility, customer collaboration, and responding to change.

- It breaks down the project into small, manageable units (sprints) and delivers working software frequently.

Agile – Manifesto

- The Agile Manifesto, written in 2001 outlines its main principles as follows:

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - individuals and interactions over processes and tools working software over comprehensive documentation.
  - customer collaboration over contract negotiation responding to change over following a plan.

- That is, while there is value in the items on the right, we value the items on the left more.

Key principles of Agile

1) Customer satisfaction through early and continuous delivery of valuable software.
2) Welcoming changing requirements.
3) Delivering working software frequently.
4) Close, daily cooperation between business people and developers.
5) Motivated individuals and self-organizing teams.
6) Face-to-face conversation as the best form of communication.
7) Continuous attention to technical excellence and good design.

Agile Methodologies – Scrum

innovate achieve lead

- In the mid-1990s, Ken Schwaber and Dr. Jeff Sutherland, two of the original creators of the Agile Manifesto, merged individual efforts to present a new software development process called **Scrum**

- Scrum is a software development methodology that focuses on maximizing a development team's ability to quickly respond to changes in both project and customer requirements.

- A popular Agile framework that organizes work into sprints (usually 2-4 weeks).

Key stakeholders of Scrum

- **Product Owner:** Defines the product backlog and prioritizes features.
- **Scrum Master:** Facilitates the process and removes obstacles.
- **Development Team:** Builds the product incrementally.
- **Ceremonies:** Sprint planning, daily stand-ups, sprint reviews, and retrospectives.

Daily Scrum

- One key feature of Scrum is the daily Scrum or daily standup, a daily meeting where team members (rather rapidly) each answer three questions:

1) What did I do yesterday that helped the team meet its sprint goals?

2) What am I planning to do today to help the team meet those goals?

3) What, if anything, do I see that is blocking either me or the team from reaching their goals?

Extreme Programming (XP)

XP Focuses on improving software quality and responsiveness to changing requirements.

Key practices include:

- **Pair Programming:** Two developers work together at one workstation.
- **Continuous Integration:** Code is integrated and tested frequently.
- **Test-Driven Development (TDD):** Write tests before writing the code.
- **Refactoring:** Continuously improving the codebase without changing its functionality.

TDD vs FDD vs BDD

TDD focuses on writing tests before code and follows a cycle of writing a test, writing code, and refactoring.

FDD emphasizes developing and delivering features in an iterative manner, with a focus on tangible, working software.

BDD involves defining behavior in a natural language that can be understood by all stakeholders and using these definitions to drive development and testing.

Evolution of DevOps

The evolution of DevOps can be traced through several stages:

Agile Development:

Agile methodologies laid the foundation by promoting collaboration and iterative development.

Continuous Integration (CI):

Tools like Jenkins, Travis CI, and CircleCI automated the process of integrating code changes and running tests.

Infrastructure as Code (IaC):

Tools like Ansible, Puppet, and Terraform enabled managing infrastructure using code, promoting consistency and scalability.

Evolution of DevOps

Continuous Deployment (CD):

Extending CI to automatically deploy changes to production, reducing manual intervention and increasing release frequency.

Monitoring and Feedback:

Tools like Nagios, Prometheus, and ELK Stack provided real-time monitoring and feedback, enabling quick issue resolution.

Example of CI/CD: IMVU

- IMVU, Inc. is a social entertainment company whose product allows users to connect through 3D avatar-based experiences.
- IMVU has a thousand test files, distributed across 30–40 machines.
- IMVU does continuous integration.
  - The developers commit early and often.
  - A commit triggers an execution of a test suite andtakes about nine minutes to run.
  - Once a commit has passed all of its tests, it is automatically sent to deployment and this takes about six minutes.

Example of CI/CD: IMVU Contd

- The code is moved to the hundreds of machines in the cluster, but at first the code is only made live on a small number of machines (canaries).
- A sampling program examines the results of the canaries and if there has been a statistically significant regression, then the revision is automatically rolled back.
- Otherwise the remainder of the cluster is made active.
- IMVU deploys new code 50 times a day, on average.

Example of CI/CD: The essence of the process

- Every time a commit gets through the test suite and is rolled back, a new test is generated that would have caught the erroneous deployment, and it is added to the test suite.
- Generally, a full test suite (with the confidence of production deployment) that only takes nine minutes to run is uncommon for large-scale systems.
- In many organizations, the full test suite that provides production deployment confidence can take hours to run, which is often done overnight.
- A common challenge is to reduce the size of the test suite judiciously and remove "flaky" tests.

Three Dimensions of DevOps:

- **People:** Focuses on culture, collaboration, and communication. Successful DevOps requires a cultural shift that encourages teamwork and shared responsibilities.

- **Process:** Emphasizes streamlined workflows and automation to improve efficiency. This includes practices like continuous integration and continuous delivery (CI/CD).

- **Tools:** Involves selecting and integrating the right tools to support automation, collaboration, and monitoring.
- Examples include Jenkins for CI/CD, Docker for containerization, and Kubernetes for orchestration.

Key DevOps Practices

- **Continuous Integration (CI):** Frequent merging of code changes into a central repository, followed by automated builds and tests to detect issues early.
- **Continuous Testing (CT):** Automated testing of the software at various stages to ensure quality and performance.
- **Continuous Delivery (CD):** Automating the release process so that code changes can be deployed to production at any time.
- **Configuration Management (CM):** Managing and maintaining the consistency of the software's performance and configuration across environments.

Principles of Software Delivery

- **Customer-Centric Action:** Prioritizing customer needs and feedback.
- **Create with the End in Mind:** Focusing on delivering value from the start.
- **Automate Everything:** Emphasizing automation to reduce manual effort and errors.
- **Work in Small Batches:** Breaking work into smaller, manageable pieces for faster delivery and feedback.
- **Continuous Improvement:** Constantly seeking ways to improve processes and outcomes.

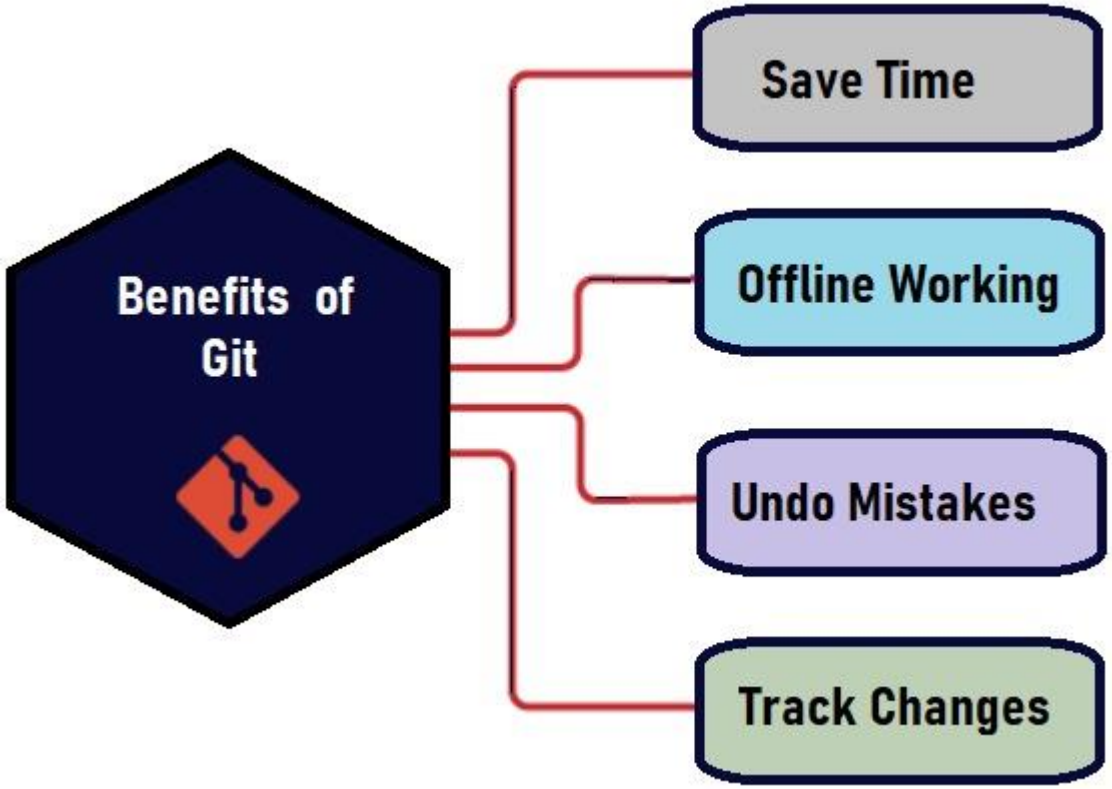
CI - steps

1) Set Up Version Control System (VCS)
2) Install and Configure CI/CD Tool
3) Create a Build Script
4) Configure the CI Tool to Trigger Builds
5) Run Unit Tests
6) Generate Code Coverage Report
7) Build Artifacts

Introduction to VCS

- What is VCS?
  - VCS stands for Version Control System.
  - Versioning: Keeps a history of changes.
  - VCS helps track changes in source code during software development.
- What is D?
  - D stands for Distributed
  - Distributed nature means code is saved in multiple locations.
- So Git is a DVCS - Distributed Version Control System!

Introduction to Git – Advantages



CVCS

- **Examples:**
- CVS (Concurrent Versions System), Subversion (SVN)

- **It's a Single Repository:**
- CVCS uses a central server to store all versions of the code.
- Each user checks out code from this central repository and commits changes back to it.

- **History:**
- The central repository keeps the complete history of all changes.
- Users only have access to this history via the central server.

- **Collaboration:** Users work directly with the central repository, which can make collaboration straightforward but also introduces a single point of failure.(SPOF!)

- **Network Dependency:** Since users need to interact with the central repository for most operations (like committing changes or retrieving updates), network connectivity is crucial.

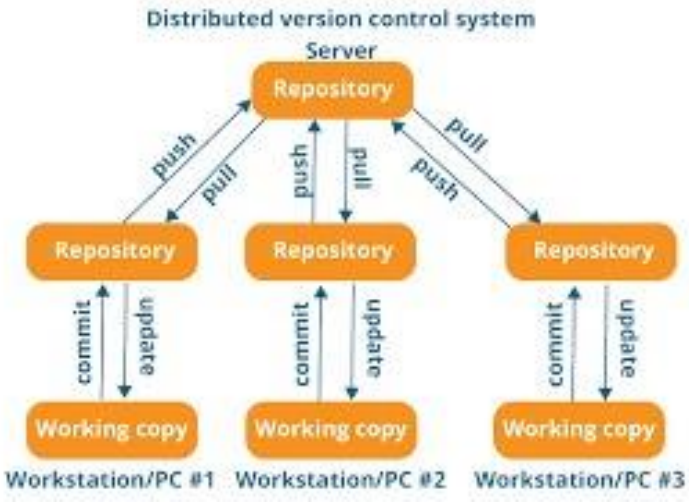
DVCS

- **Examples:**
- Git, Mercurial

- **Multiple Repositories:**
- In DVCS, each user has a full copy of the entire repository, including its history.
- This means every user has a complete local version of the codebase.

- **Collaboration:**
- DVCSs make collaboration more flexible.
- Users can commit changes to their local repository and then push updates to shared repositories.
- Branching and merging are typically more streamlined.

- **Network Independence:**
- Users can perform most operations locally without needing network access.
- Changes can be synchronized with remote repositories when convenient.



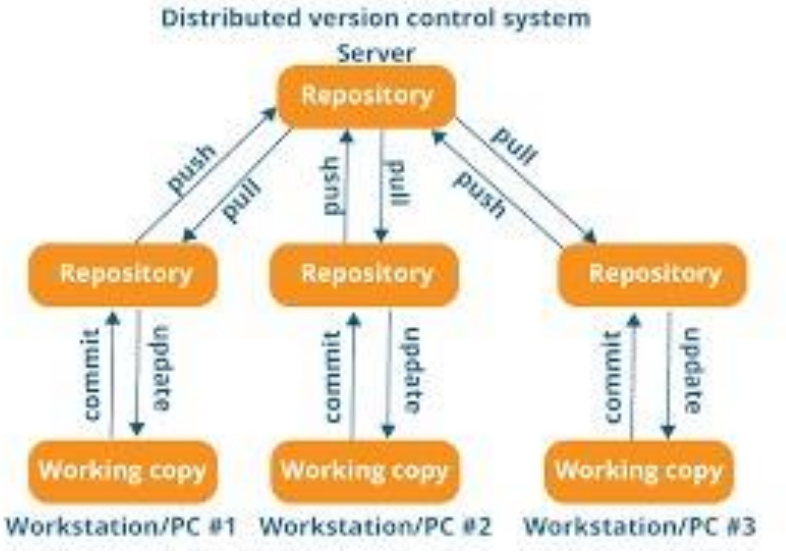
CVCS vs DVCS

- **Centralization vs. Decentralization:**
- CVCS centralizes the repository, while DVCS distributes it across multiple users.

- **Network Dependency:**
- CVCS requires constant network access to the central repository, whereas DVCS allows for extensive local work.

- **History Management:**
- DVCSs allow for more extensive local history management and offline work, whereas CVCSs rely on the central repository for history access.

- **Branching and Merging:**
- DVCSs often provide more powerful and flexible tools for branching and merging.



A brief History of VCS

- **Early Version Control Systems Before Git:**
- Version control systems were primarily centralized.
- Notable examples include: RCS, CVS, SVN

- **RCS (Revision Control System):**
  - Developed in the 1980s,
  - RCS was one of the first systems to manage file revisions.
- **CVS (Concurrent Versions System):**
  - Introduced in the late 1980s and early 1990s,
  - CVS allowed multiple developers to collaborate on a codebase by managing revisions in a central repository.
- **Subversion (SVN):**
  - Released in 2000, SVN aimed to improve upon CVS with features like atomic commits and better handling of binary files.

The Birth of Git

- Creation by Linus Torvalds in 2005.

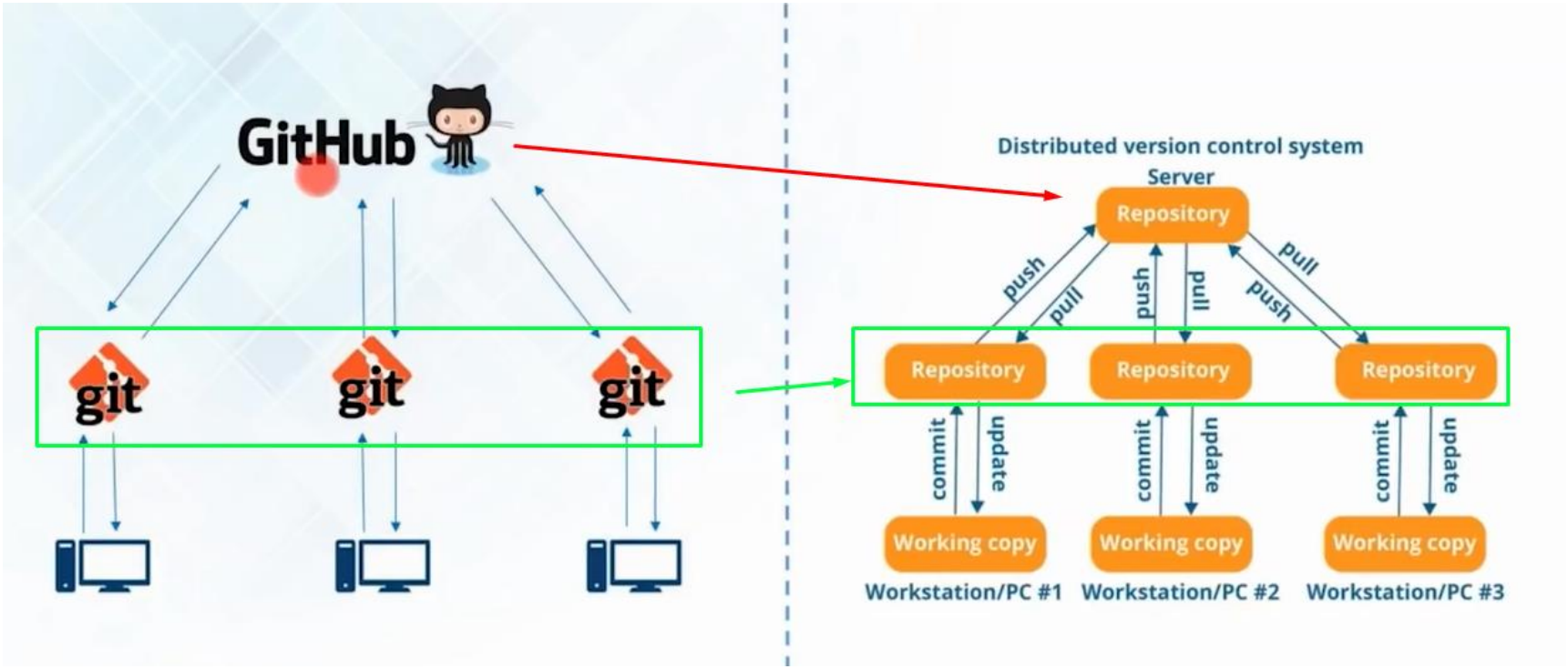
- **Motivation:**

  Linus Torvalds, the creator of the Linux kernel, needed a new version control system to handle the Linux kernel development after the Linux kernel project had outgrown the limitations of BitKeeper, a proprietary version control system.

- **Development:**

  Torvalds began developing Git in April 2005.
  His goal was to create a distributed version control system that was fast, efficient, and capable of handling large projects with many contributors.

Git vs GitHub



Git vs GitHub

Git

- Used for version control
- Installed locally on computer /laptop
- Tracks changes made to a file

GitHub

- Used for hosting git repos remotely
- Cloud based
- Provides web UI to view file changes

Installing Git

Windows:

- Download from Git for Windows Run the installer and use default settings.

Link to download:
https://git-scm.com/downloads

Mac:

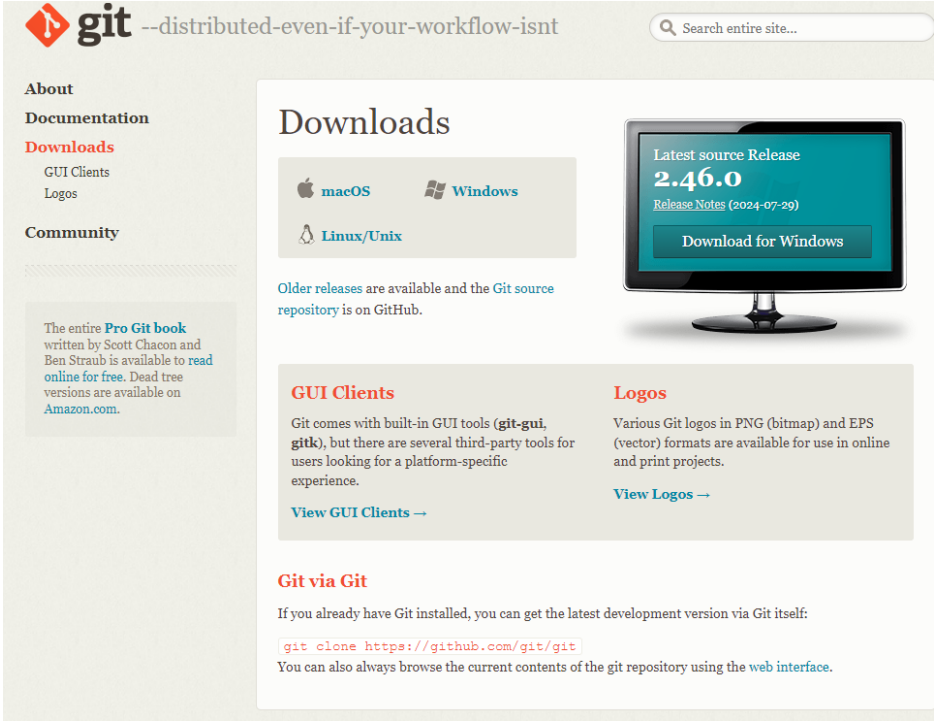
- Install via Homebrew:
- brew install git

Linux:

- Use the package manager:
- sudo apt-get install git (for Ubuntu/Debian)

Verifying installation:

git --version



THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 3

Recap

- DevOps Lifecycle
- Three Dimensions of DevOps:
  People, Process, Tools
- Key DevOps Practices – CI, CT, CD, CM
- Agile Methodology for DevOps
- Principles of Software Delivery
- Version control system and its types
- Introduction to GIT

Agenda for today's class

innovate achieve lead

- GIT Basics commands
- Git Lifecycle Visualized
- Git workflows
  - Feature workflow
  - Master workflow
  - Centralized workflow
- Feature branching
- GIT Merge requests
- Managing Conflicts
- Tagging and Merging
- Best Practices- clean code

Installing Git

Windows:

- Download from Git for Windows Run the installer and use default settings.

Link to download:
https://git-scm.com/downloads

Mac:

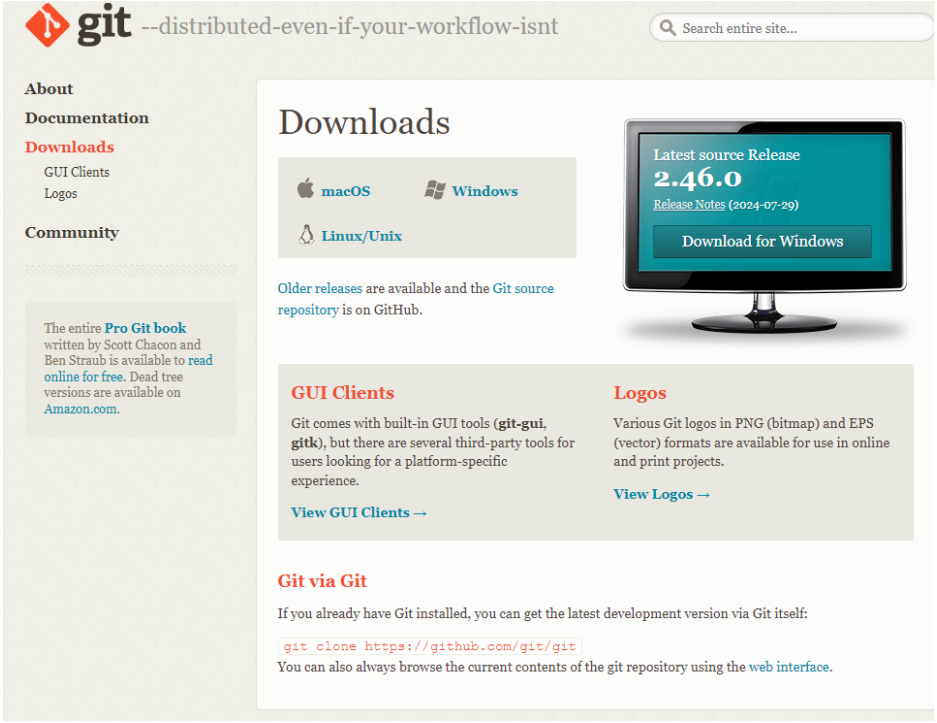
- Install via Homebrew:
- brew install git

Linux:

- Use the package manager:
- sudo apt-get install git (for Ubuntu/Debian)

Verifying installation:

git --version



Setting up your identity

- When you set up your identity in Git, you're configuring Git with your name and email address.
- This information is used to label the commits you make, so it's clear who made each change in the project.
- Properly setting up your identity ensures that your commits are correctly attributed to you and helps maintain accurate version history.

- Open Your Terminal or Command Line Interface and run

  git config –list

- To configure globally (for all repositories):

  git config --global user.name "Your Name"

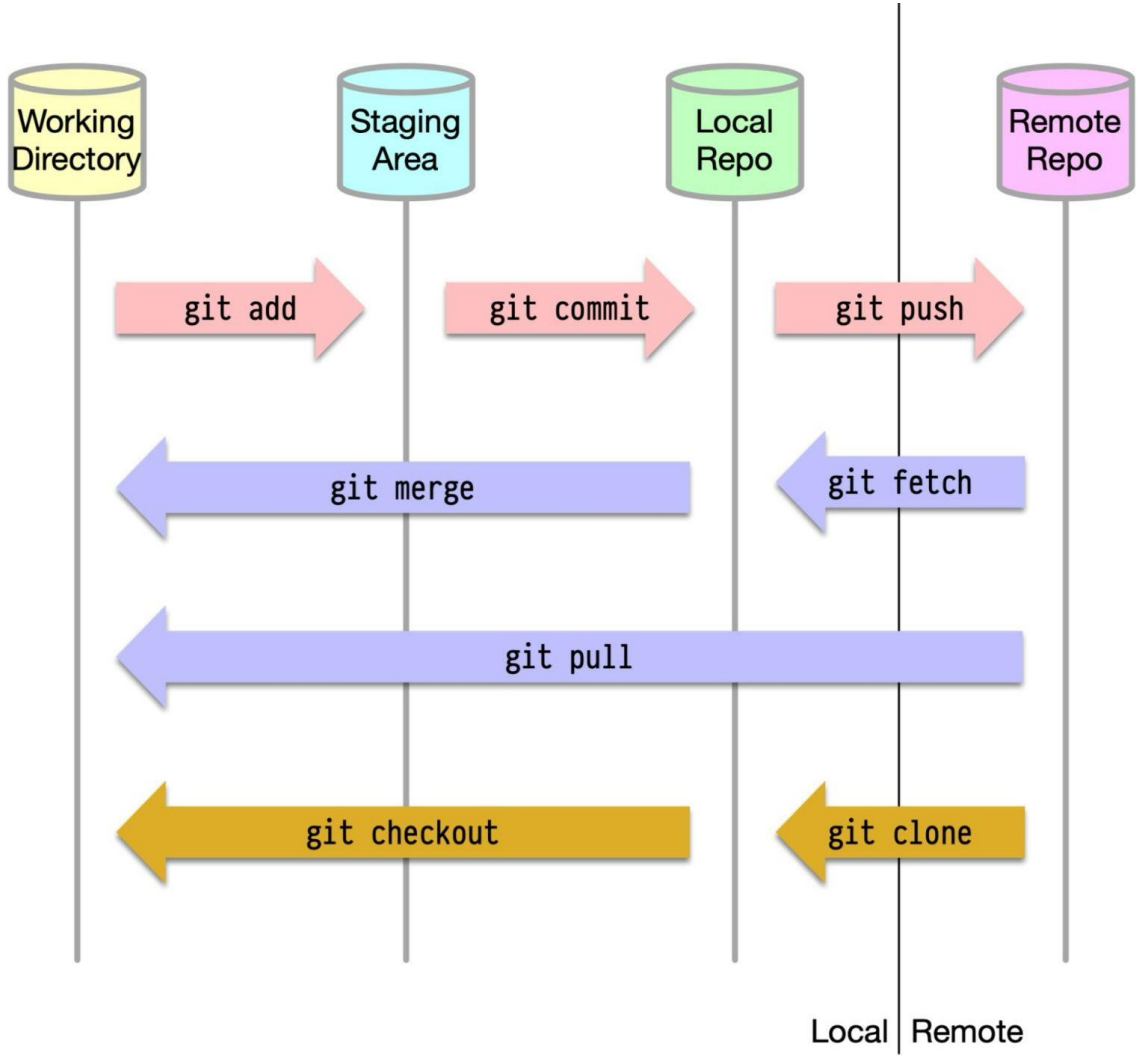
  git config --global user.email "your.email@example.com"

- To configure locally (for a specific repository):

  git config user.name "Your Name"

  git config user.email "your.email@example.com"

Git Lifecycle Visualized



Creating a New Local Repository

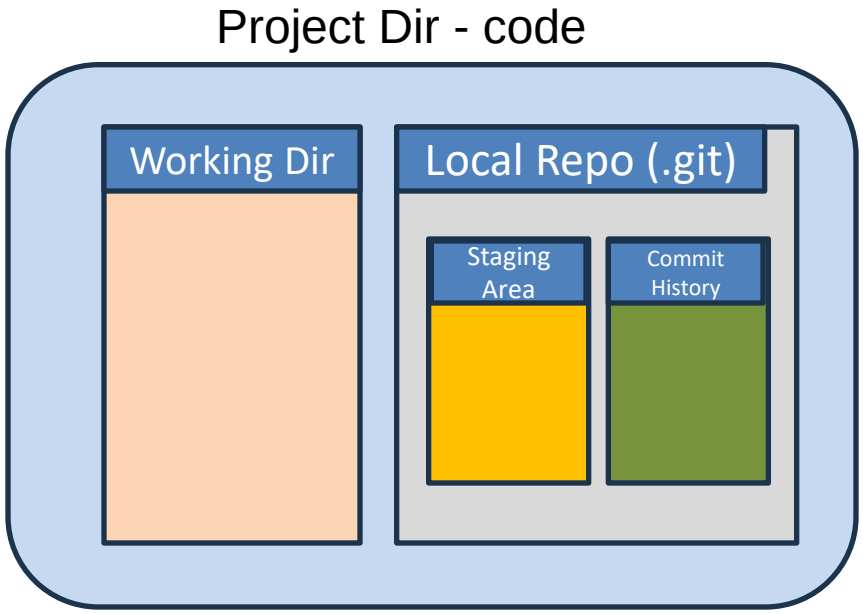
...or create a new repository on the command line

```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha94/test.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/tanvitha94/test.git
git branch -M main
git push -u origin main
```

- Initialize a new repository (local):
  git init
- This creates a default branch called master.
- To create a branch with the name main use
  git init –b main
  Or
  git switch –create main



Tracking the git repo

- The git status command is used to display the state of the working directory and the staging area in a Git repository.
- It provides important information about which changes have been staged, which haven't, and which files aren't being tracked by Git.
- Open Your Terminal or Command Line Interface and run

  git status

O/p:

On branch main

Your branch is up to date with 'origin/main'.

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: file1.txt

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: file2.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

README.md

Creating a New File

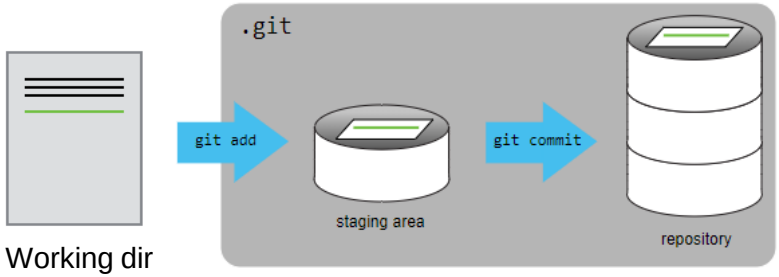
...or create a new repository on the command line

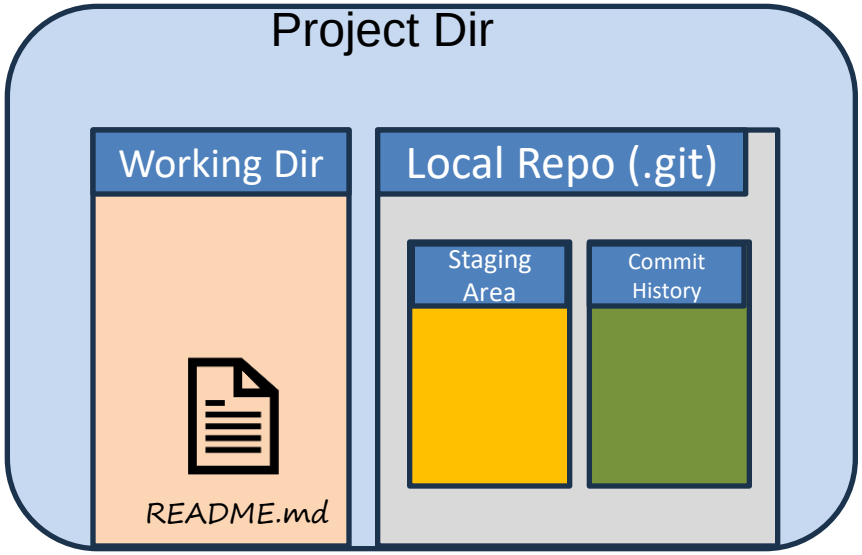
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha94/test.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/tanvitha94/test.git
git branch -M main
git push -u origin main
```

- Create a new file:
  - touch README.md





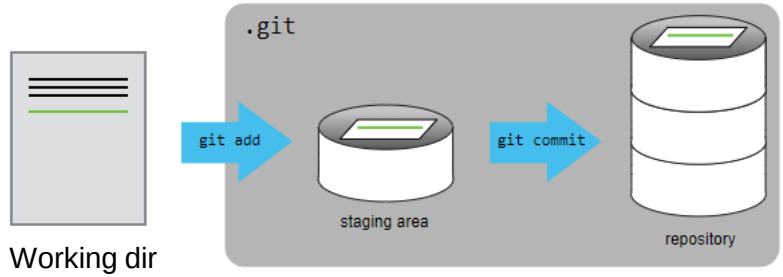
Adding the file to staging area

...or create a new repository on the command line

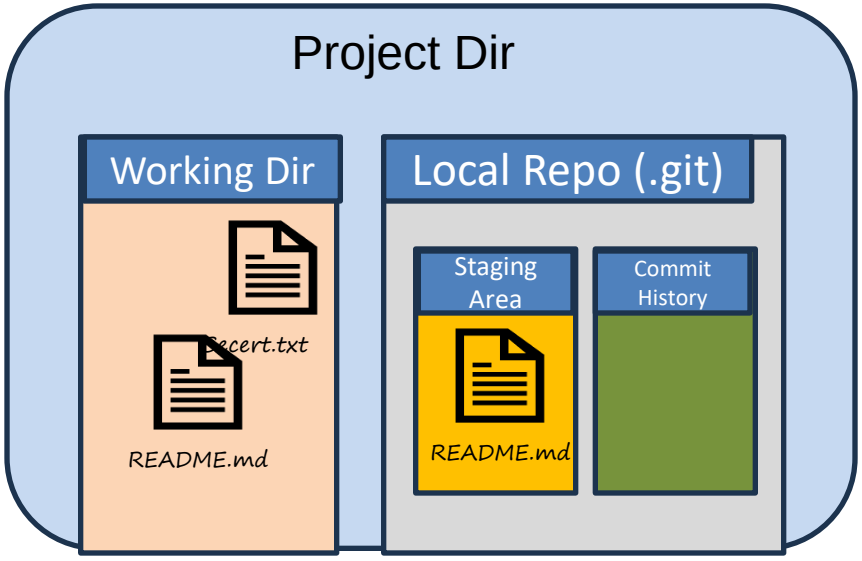
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha94/test.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/tanvitha94/test.git
git branch -M main
git push -u origin main
```



- Create a new file:
  - touch README.md
- Add the file to the staging area:
  - git add README.md
- Staging area:
  - index file in .git folder



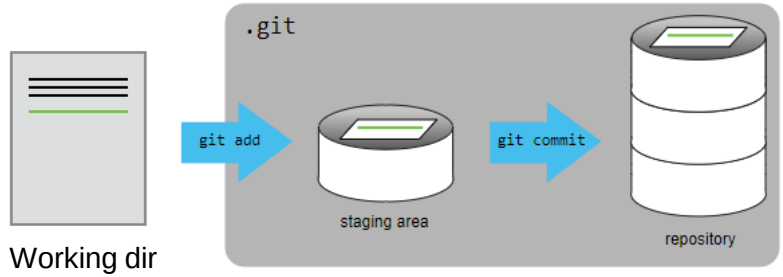
Committing the file to local repo

...or create a new repository on the command line

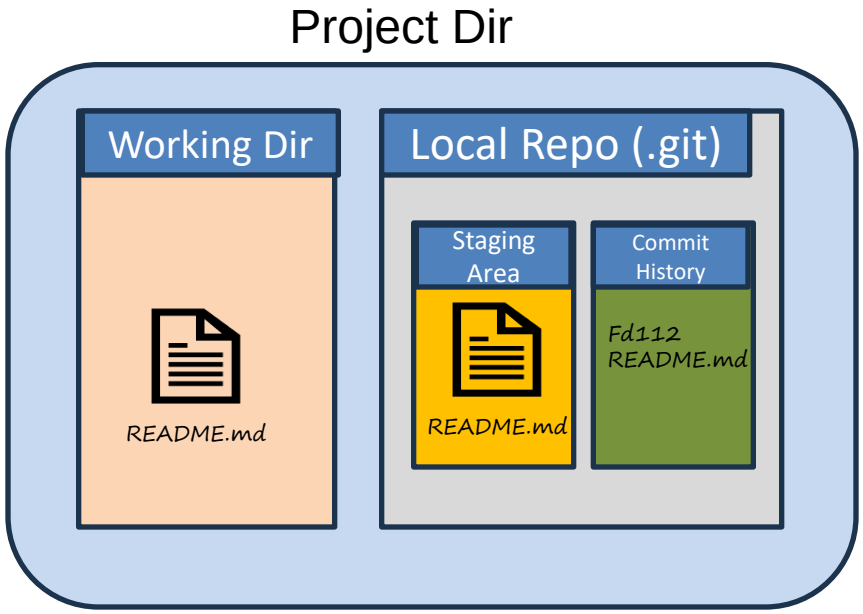
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha94/test.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/tanvitha94/test.git
git branch -M main
git push -u origin main
```



- Create a new file:
  - touch README.md
- Add the file to the staging area:
  - git add README.md
- Commit the file to the repository:
  - git commit -m "Initial commit"



Viewing the Commit History

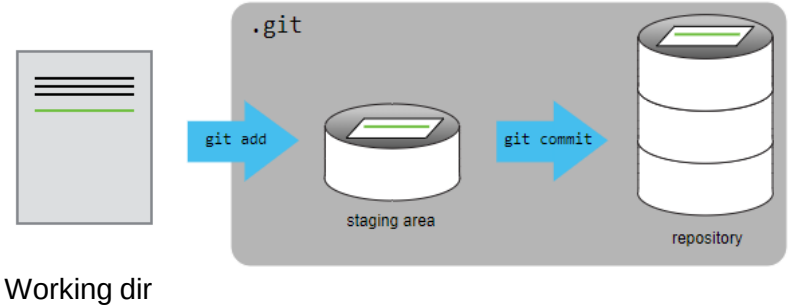
- The git log command is used to view the commit history in a Git repository.
- It shows a list of all the commits made in the repository, starting with the most recent one.

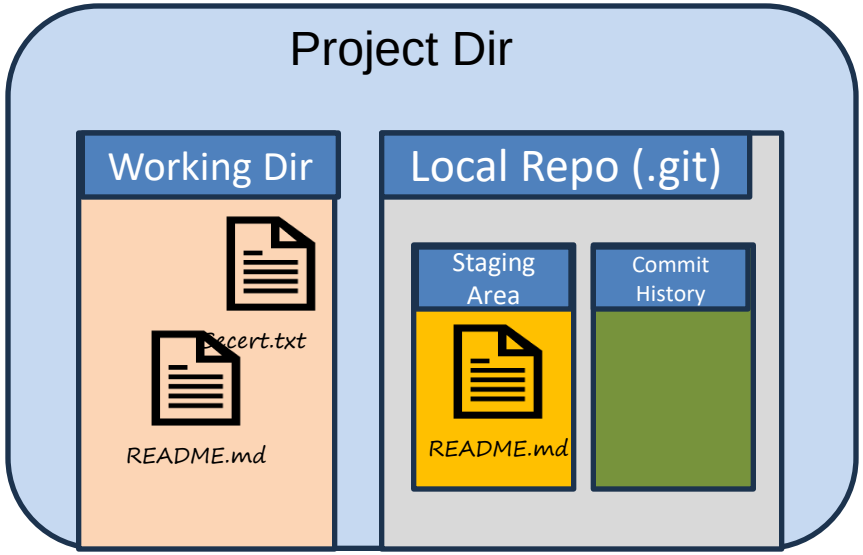
1) Commit Hash: A unique identifier (SHA-1 hash) for the commit.
2) Author: The name and email of the person who made the commit.
3) Date: The date and time when the commit was made.
4) Commit Message: A short description of what changes were made in the commit.

- To view commit history:
  - git log
- To view a simplified log:
  - git log --oneline
- To view changes introduced by each commit:
  - git show commit-hash

Understanding .git directory

- **Staging Area:**
  .git/index
- **Commit History:**
- Commits: .git/objects/
- Branch References: .git/refs/heads/
- Tag References: .git/refs/tags/
- Current Branch/Commit: .git/HEAD





Collaborating with Others

...or create a new repository on the command line

```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha94/test.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/tanvitha94/test.git
git branch -M main
git push -u origin main
```

- To view remote repositories:
  - git remote -v
- To add a remote repository:
  - git remote add origin https://github.com/<username>/<repository>.git
- Push changes to the remote repository:
  - git push origin main
- Pull changes from the remote repository:
  - git pull origin main

Git workflows

- Git workflows provide structured ways to manage code in a project.
- Different workflows cater to different types of development teams and project needs.
- The following are the three workflows that are popular:

    1) **Centralized Workflow**
    2) **Master Workflow**
    3) **Feature Workflow**

Git workflows – Centralized Workflow

- This workflow mimics the traditional version control systems like Subversion, where there is a single central repository.
- It's suitable for teams transitioning from a centralized VCS to Git.
- **Branches:**
- 1) **main (or master):** The single, central branch where all changes are pushed.
- **Process:**
- Developers clone the central repository and create their own local branches if needed.
- They make changes and commit them locally.
- When ready, they push their changes directly to the main branch on the central repository.
- Conflicts are resolved by the developer before pushing.

Git workflows – Master Workflow

- In this workflow, all development happens directly on the master branch.
- It's suitable for small teams or solo developers where the overhead of managing multiple branches might not be necessary.
- **Branches:**
- 1) **main (or master):** The only branch, where all commits are made
- **Process:**
- Developers make changes and commit directly to the master branch.
- If needed, tags can be used to mark release points or versions.

Git workflows – Feature Workflow

- This workflow is ideal for teams that work on multiple features concurrently.
- Each feature is developed in its own branch.

- **Branches:**
- 1) **main (or master):** The stable branch where all finished features are merged.
- 2) **feature-branches:** Separate branches for each feature being developed.

- **Process:**
- Developers create a new branch from the main branch for each new feature.
- They develop the feature on this branch.
- Once the feature is complete and tested, the branch is merged back into the main branch, often via a merge request(MR).
- The feature branch can be deleted after merging.

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 4

Recap

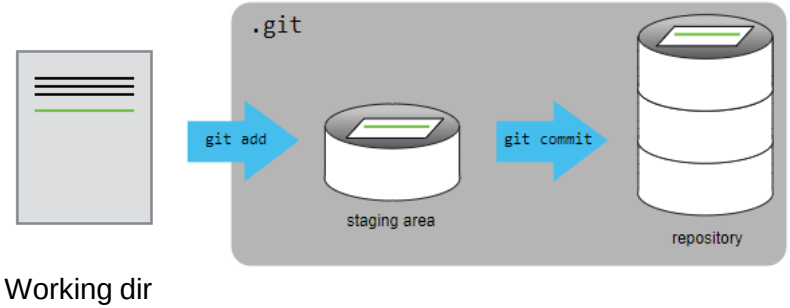
- Git – DVCS
  - Local Repository
    - Staging area
    - Commit history
  - Remote Repository
- Git Basics commands
  - Creating Repositories, Clone
  - add , commit, push, pull
  - git status, logs
- Git workflows
  - Master workflow
  - Centralized workflow
  - Feature workflow

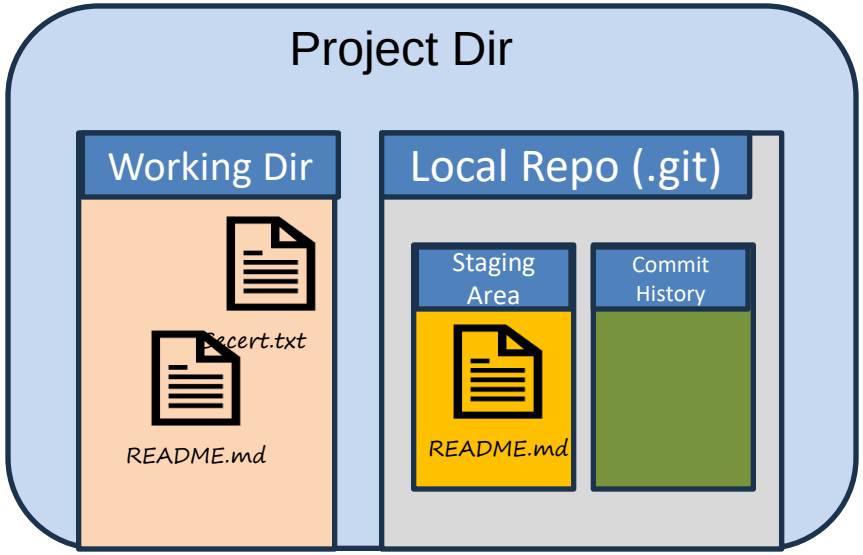
Agenda for today's class

- How does Git keep track of files and changes?
  - Blob
  - Tree
  - Commit
- Feature branching
- GIT Pull requests
- Managing Conflicts
- Tagging and types
- Best Practices- clean code
- Overview of Xops
- Case Studies - Netflix, Facebook

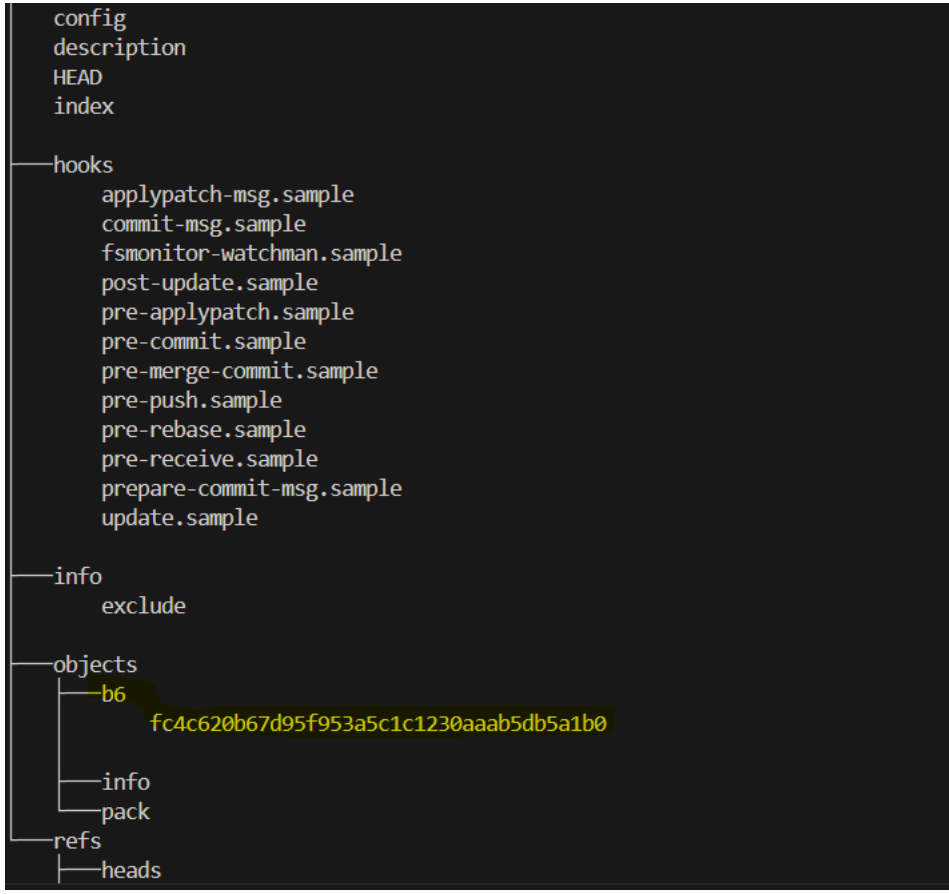
Understanding .git directory

- **Staging Area:**
  .git/index
- **Commit History:**
- Commits: .git/objects/
- Branch References: .git/refs/heads/
- Tag References: .git/refs/tags/
- Current Branch/Commit: .git/HEAD





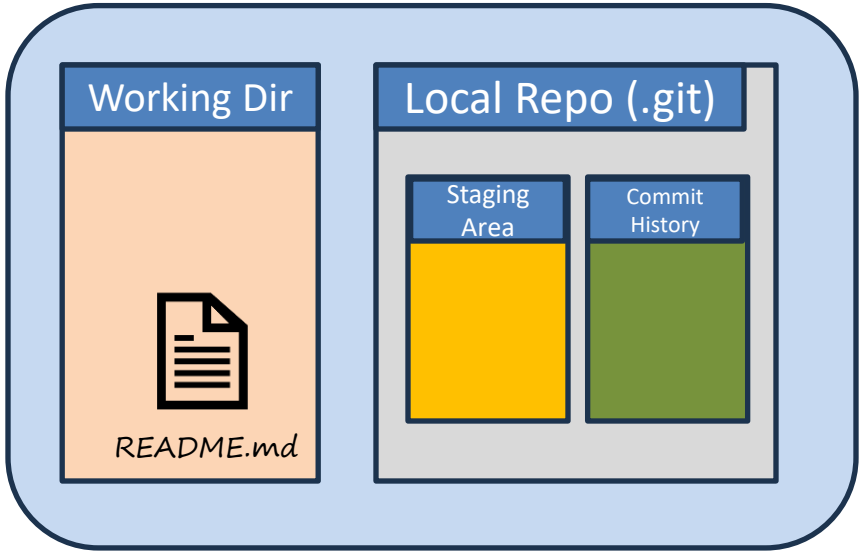
Git add visualized



How does Git keep track of files and changes? Blob!

Blob:

- Stands for **B**inary **L**arge **OB**ject.
- Represents the content of a file.
- Blobs are stored as binary data.
- Identified by a unique SHA-1 hash.
- Blobs do not contain any metadata about the file, like its name or permissions, they contain just the content itself.




```
git hash-object -w <file_path>
```

-w writes to .git/objects

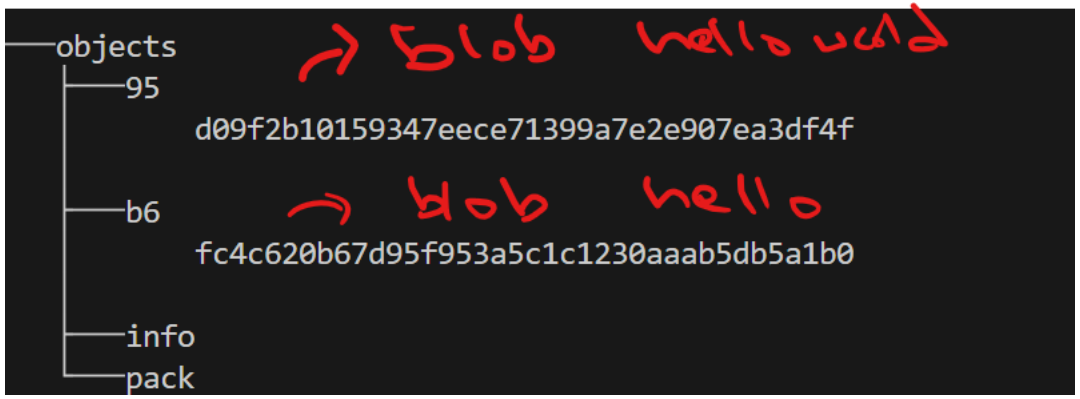
Ex: `git hash-object -w README.md`

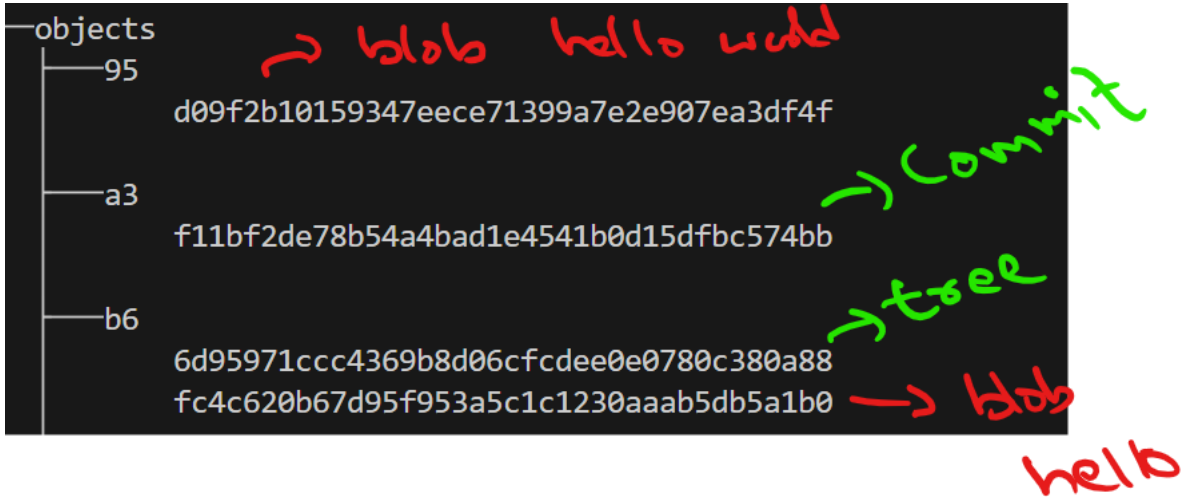
How does Git keep track of files and changes?

innovate achieve lead

- To list the current directory including subdirectories and files in cmd use
  - tree /f <dir_name>
  - Ex: tree /f .git
- To view type and content of a file in .git folder use:
  - git cat-file -t <hash>
  - git cat-file -p <hash>
  - Ex: git cat-file –p fef011e610096c75bc420592c1e57fe47594ab86

Git State tracked after a commit





Blob vs Tree

PS C:\Users\saisu\bits\Devops for cloud\git-demo2> git cat-file -t b66d95971ccc4369b8d06cfcdee0e0780c380a88

tree

PS C:\Users\saisu\bits\Devops for cloud\git-demo2> git cat-file -p b66d95971ccc4369b8d06cfcdee0e0780c380a88

100644 blob b6fc4c620b67d95f953a5c1c1230aaab5db5a1b0 file1.txt

100644 blob b6fc4c620b67d95f953a5c1c1230aaab5db5a1b0 file2.txt

100644 blob 95d09f2b10159347eece71399a7e2e907ea3df4f file3.txt

100644

- ➔ 100 -> file
- ➔ 644 -> permissions

040000

- ➔ 040 -> folder
- ➔ 000 -> permissions

How does Git keep track of files and changes? Tree!

innovate achieve lead

Tree:

- Represents a directory structure.
- A tree object contains pointers to blob objects (files) and other tree objects (subdirectories).
- Each entry in a tree object includes the filename, file type (blob or tree), and the SHA-1 hash of the object.
- *A tree is also identified by a hash like blob objects.*

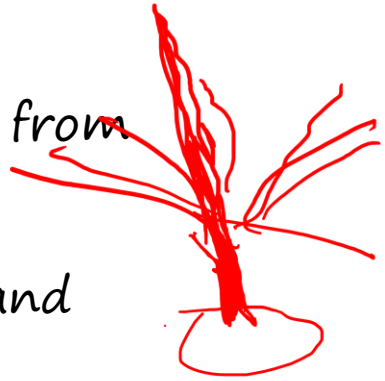
How does Git keep track of files and changes? Commit!

innovate achieve lead

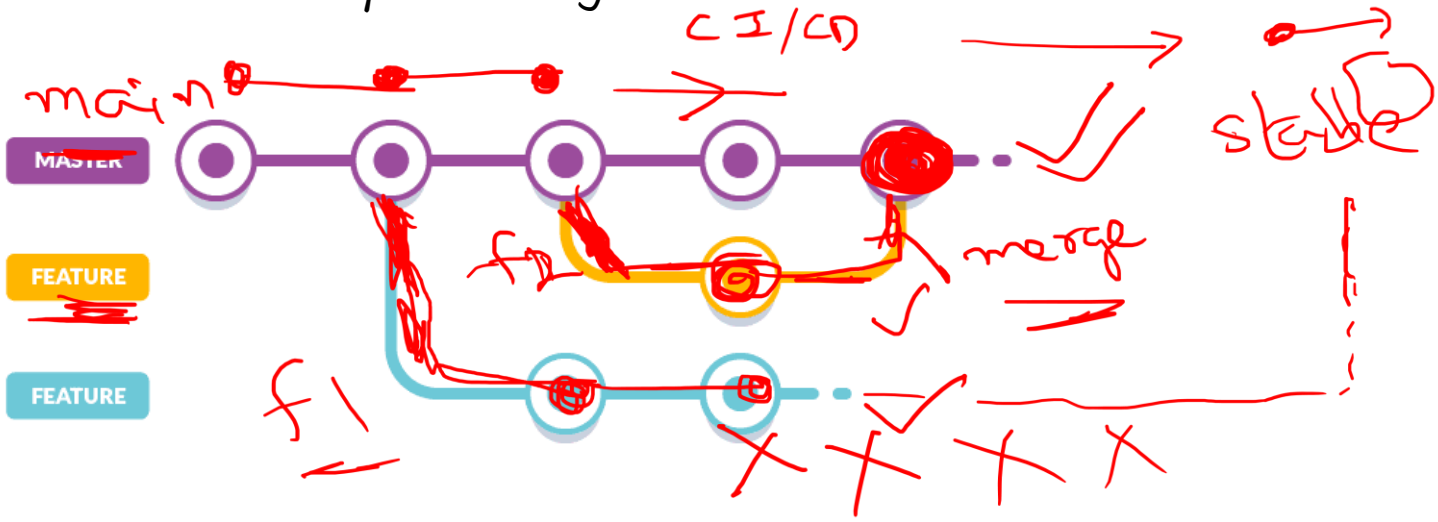
Commit:

- A commit object ties together the tree structure (representing the project state at a certain point).
- It contains a commit message, the author, and a pointer to one or more parent commits
- Each commit is identified by a unique SHA-1 hash like blobs, trees.

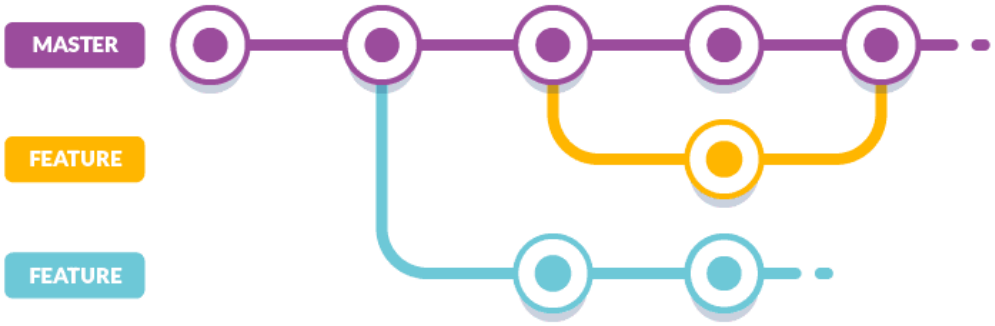
Feature workflow – Merging



- The git merge command is used to integrate changes from one branch into another branch.
- It allows you to take the content of a source branch and combine it with the branch you're currently on.
- This is a crucial operation in Git for bringing together work that has been done independently in different branches.

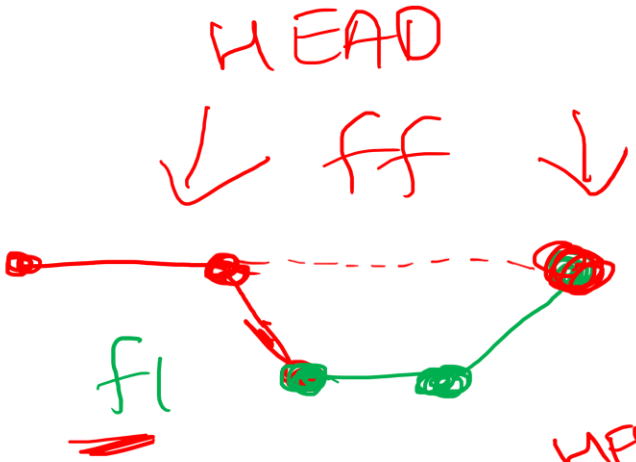


Feature workflow – Merging



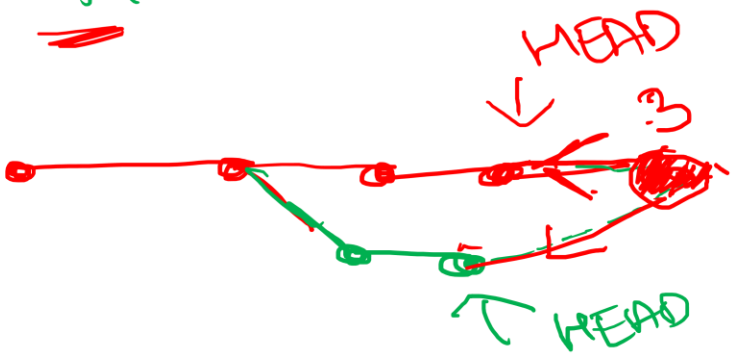
Case 1:

main



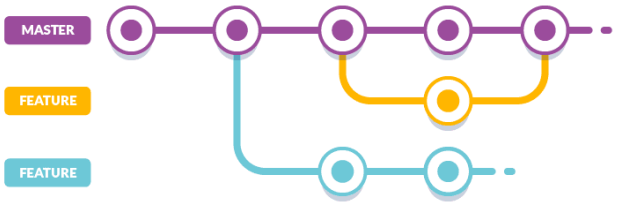
Case 2:

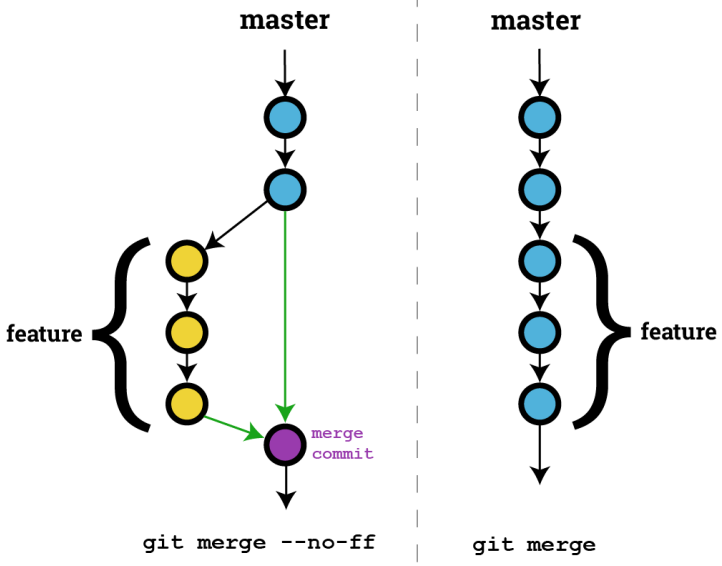
main



Types of Merging - Fast-Forward Merge

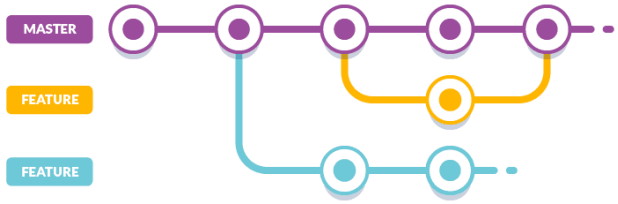
- A fast-forward merge happens when the branch being merged has a direct linear path from the current branch.
- In other words, no additional commits have been made on the target branch since the source branch was created.
- No new merge commit is created; history remains linear.
- The branch pointer is simply moved forward to the latest commit of the source branch.
- Step 1: Switch to the Target Branch:
  git checkout main
- Step 2: Run the Merge Command:
  git merge feature-branch

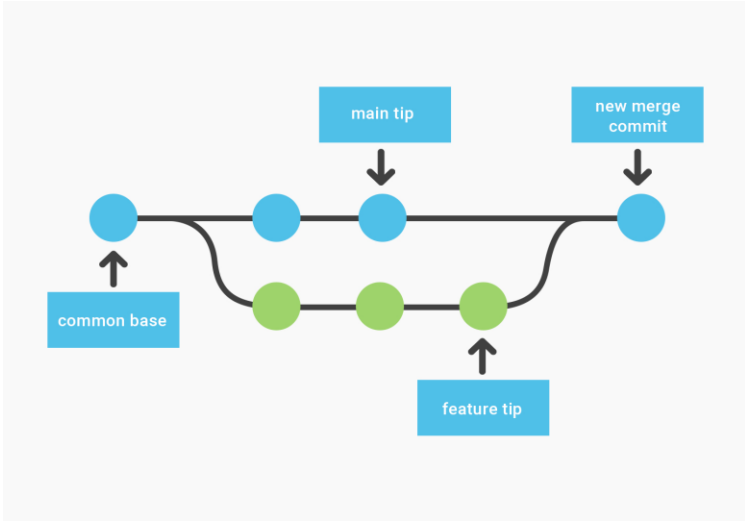




Types of Merging – Three-Way Merge

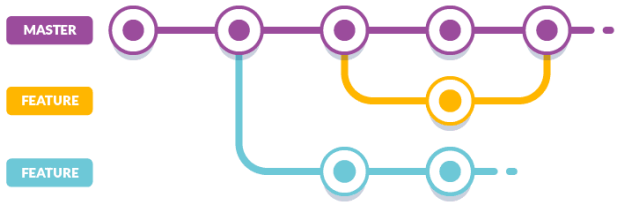
- A three-way merge happens when the target branch has diverged from the source branch, meaning there have been commits on both branches since they diverged.
- Git will create a new commit on the target branch that brings together changes from both branches.
- This commit will have two parent commits: one from each branch.
- Step 1: Switch to the Target Branch:
  git checkout main
- Step 2: Run the Merge Command:
  git merge feature-branch

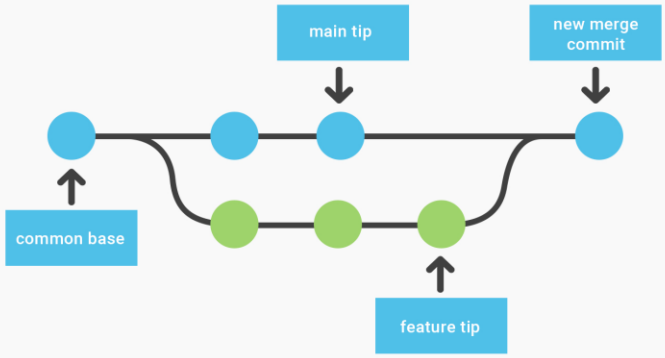




Git Tagging

- Git tags are a useful feature for marking specific points in a repository's history.
- Typically used to denote important milestones like releases.
- Tags are similar to branches, but unlike branches, they are usually static and do not change once created.
- To view tags use:
  - git tag
- To see details about a specific tag use:
  - git show <tag>
  - Ex: git show v1.0

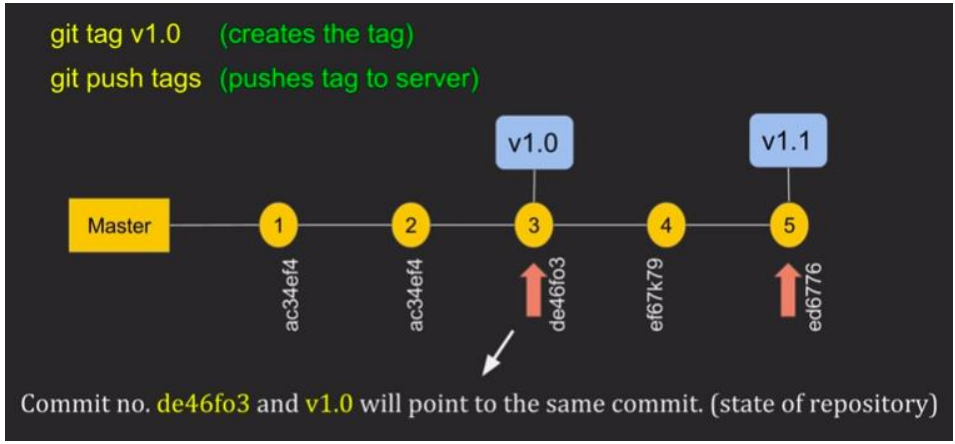




Git Tagging – Types

1) **Lightweight Tags:**
   - A lightweight tag is simply a pointer to a specific commit.
   - It doesn't contain any additional information, such as a tag message or author.
   - Ex1: git tag v1.0
   - Ex2: git tag v1.0 <commit_hash>

2) **Annotated Tags:**
   - An annotated tag is a full object in the Git database, stored as an object in the .git directory.
   - It contains a tag message, tagger's name, email, date etc
   - Ex: git tag -a v1.0 –m "Version 1.0 release"



Git Tagging – Pushing Tags to a Remote Repository

- By default, tags are not automatically pushed to remote repositories when you push commits.
- To push a specific tag:
  - git push origin v1.0
- To push all tags:
  - git push –tags
- **Deleting Tags:**
  - git tag -d v1.0
  - git push origin --delete v1.0

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 5

Recap

- Version control system and its types
- GIT Basics commands
- Git workflows
- Feature branching
- GIT Pull requests
- Managing Conflicts
- Tagging and Merging

Agenda for today's class

innovate achieve lead

- Best Practices- clean code
- Overview of Xops
- Case Studies - Netflix, Facebook
- Modern application requirements

Git Best Practices- clean code

innovate achieve lead

1) **Write Clear Commit Messages**
   - Use a consistent format for commit messages.
   - A common convention is:
     - Subject line: Short summary (50 characters or less).
     - Body (optional): Detailed description if needed, with a blank line separating it from the subject line.

2) **Make Atomic Commits**
   - Each commit should represent a single logical change. Avoid bundling unrelated changes together.

3) **Use Branches Effectively**
   - Create branches for new features or bug fixes.
   - Follow naming conventions for branches.

4) **Review Code Before Merging**
   - Create a merge request (MR).

5) **Avoid Committing Sensitive Information**
   - Add files and directories to .gitignore that should not be tracked by Git, such as build artifacts, temporary files, and IDE-specific files.

.gitignore

- The .gitignore file in a Git repository specifies which files and directories should be ignored by Git.
- These ignored files are not tracked in the repository, meaning
  - they won't be included in commits,
  - won't be pushed to remote repositories and
  - won't clutter up the working directory status.
- The .gitignore file is usually placed at the root of your repository, but you can have multiple .gitignore files in different directories if needed.

.gitignore examples

innovate achieve lead

1) Ignoring a Specific File:
secret.txt

2) Ignoring All Files with a Specific Extension:
Ex: if we wanted git to ignore all log files
*.log

3) Ignoring a Directory:
/build/

4) Ignoring All Files Inside a Directory:
logs/*

5) Ignoring Everything Except One File:
*.log
!important.log

6) Comments:
Lines starting with # are considered comments and are ignored.
Ex:
#Ignore log files
*.log

7) Wildcards:

a) * - match any number of chars
b) ? matches single char
c) ** - match multiple directory levels

8) Common Use Cases in .gitignore

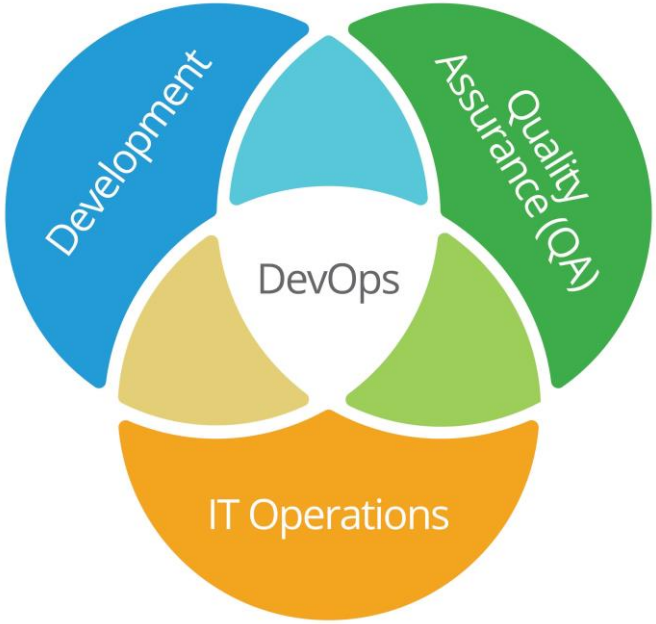
a) Node.js Projects
node_modules/
npm-debug.log

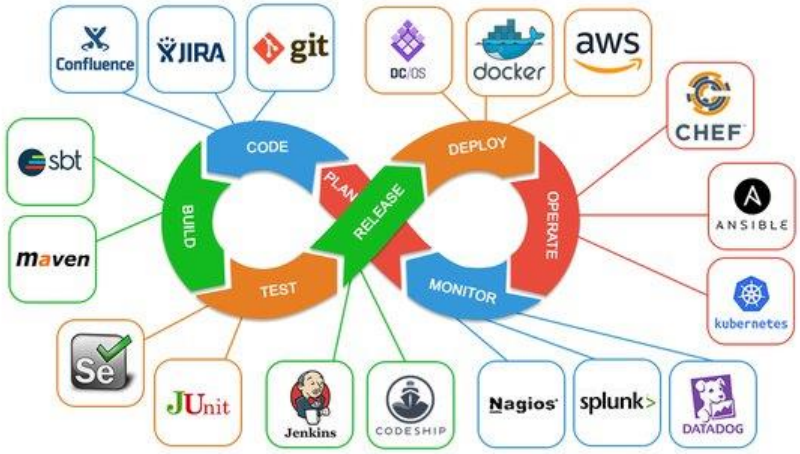
b) Python Projects
__pycache__/
*.pyc
*.pyo
.venv

Recap – What is DevOps?

- DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).

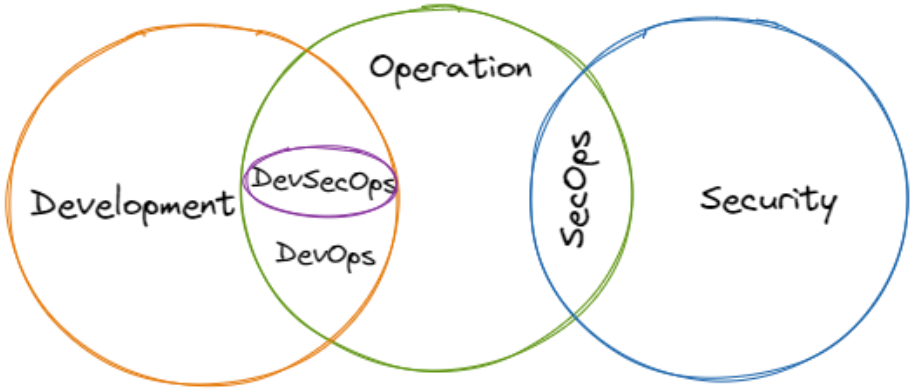
- It aims to shorten the software development life cycle and provide continuous delivery with high software quality.





Overview of XOps

- XOps stands for cross-operations.
- Xops extends the principles of DevOps to various operational domains, aiming to bring the same benefits of improved collaboration, automation, and efficiency.



- Here are some of the key types of XOps:
  - **DataOps**
  - **SecOps**
  - **AIOps**
  - **MLOps**

Overview of Xops : DataOps

Focus:

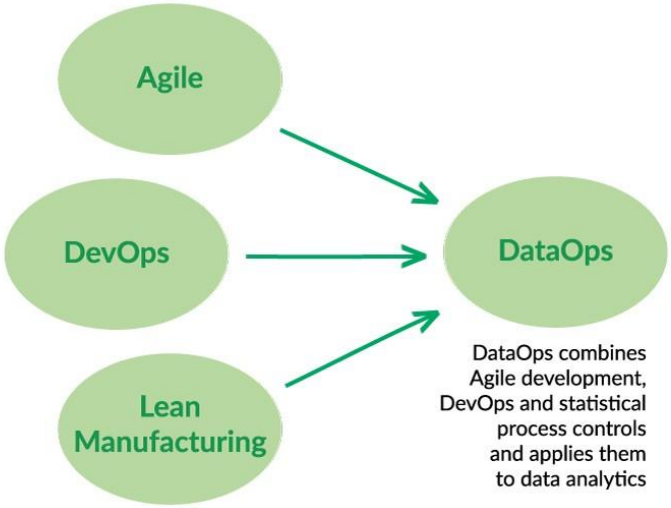
- Enhances the speed, quality, and reliability of data analytics and management processes.

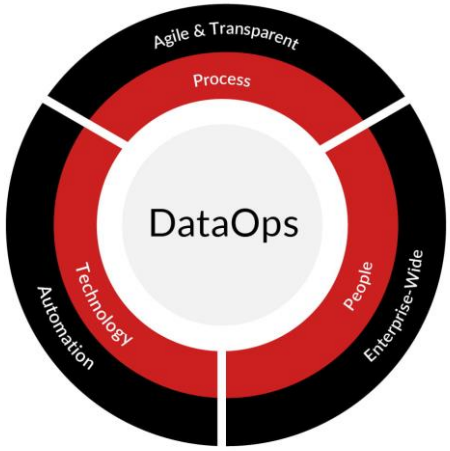
Practices:

- Data pipeline automation
- Continuous integration
- Continuous deployment of data-related processes
- Version control for data and analytics code.

Tools:

- Apache NiFi
- Apache Airflow
- DBT (Data Build Tool).





DataOps Stack

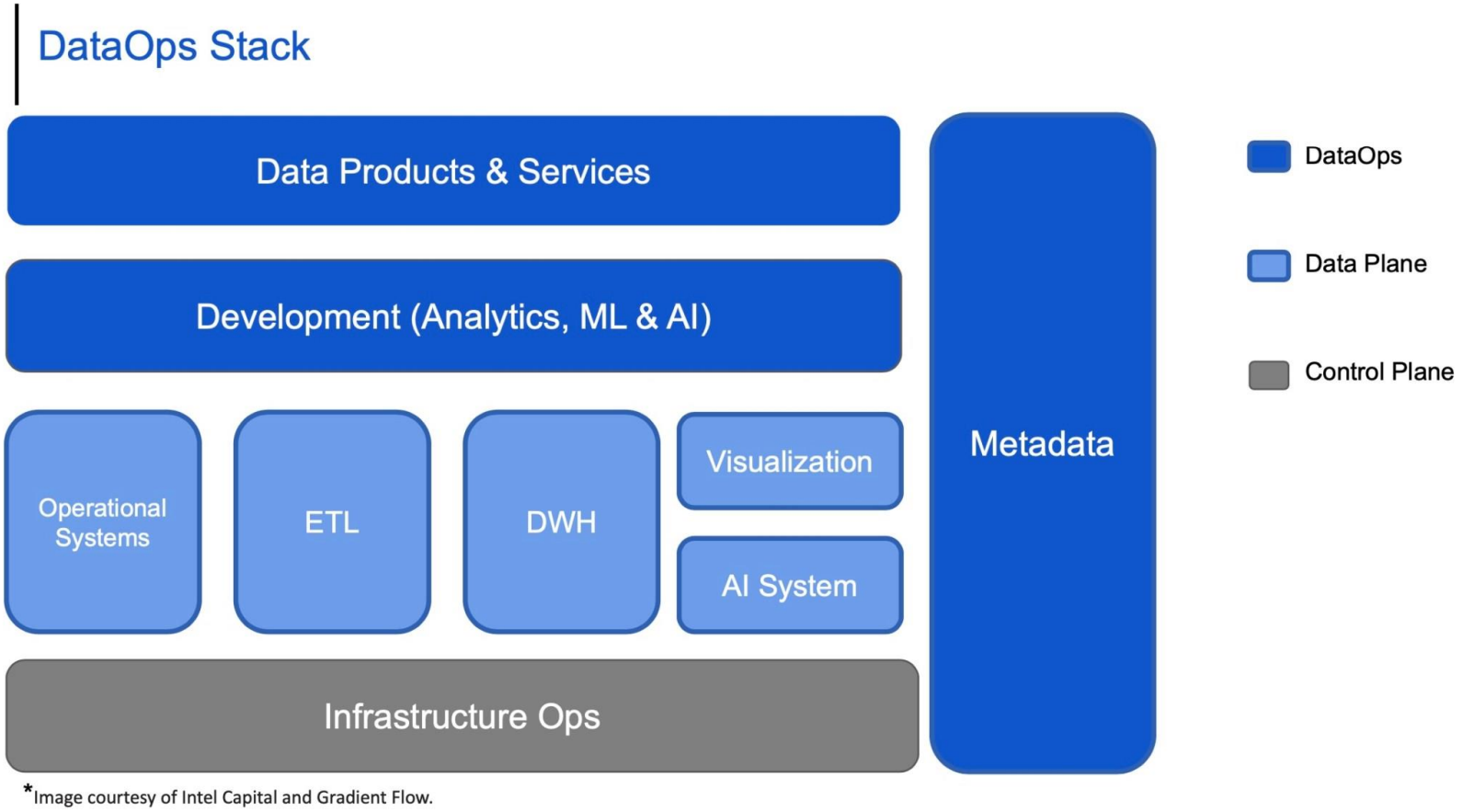


*Image courtesy of Intel Capital and Gradient Flow.

DataOps Principles

1) Continually satisfy your customer

Our highest priority is to satisfy the customer through the early and continuous delivery of valuable analytic insights

2) Value working analytics

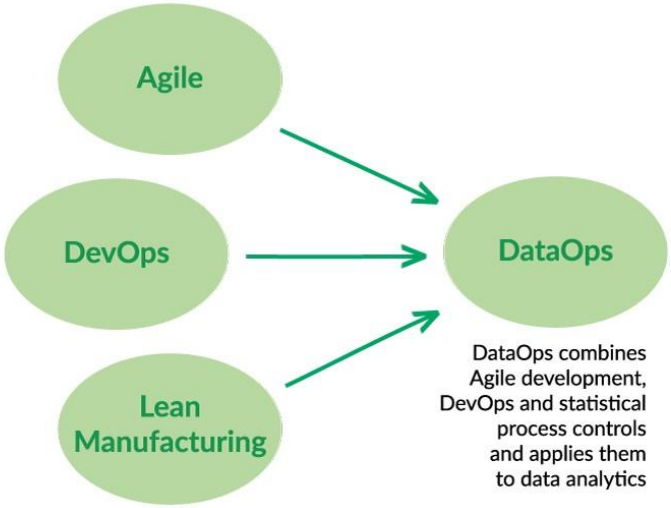
The primary measure of data analytics performance is the degree to which insightful analytics are delivered, incorporating accurate data.

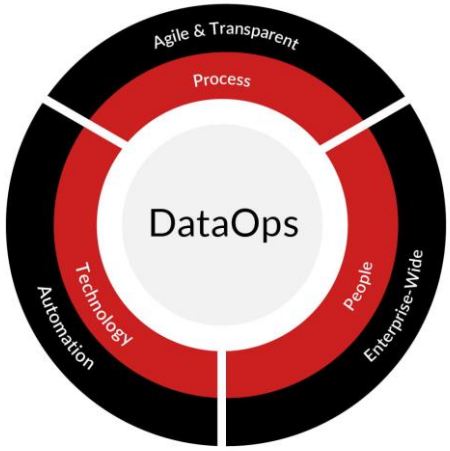
3) Daily interactions

Customers, analytic teams, and operations must work together daily throughout the project.

4) Data orchestration

Automates and optimizes the flow of data through the pipeline, including extracting, transforming, cleaning, and loading data





Overview of Xops : SecOps

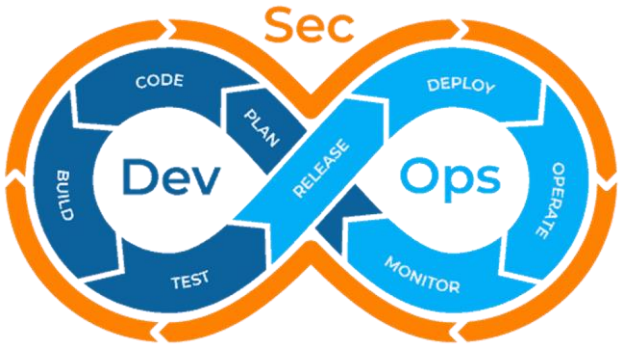
- SecOps stands for security-operations.

Focus:

- Integrates security practices into the DevOps process to ensure security is addressed throughout the software development lifecycle.

Practices:

- Automated security testing
- Continuous monitoring for vulnerabilities
- Integrating security tools into CI/CD pipelines.



Tools:

- OWASP ZAP – open-source security tool designed to help find vulnerabilities in web applications during the development and testing phases

Overview of Xops : AIOps

- **Focus**: Utilizes artificial intelligence and machine learning to enhance IT operations through proactive problem resolution and automation.
- **Practices:** Predictive analytics, anomaly detection, automated root cause analysis.
- **Tools:** Splunk, Dynatrace, Moogsoft.

Overview of Xops : MLOps

- **Focus:** Streamlines the deployment, monitoring, and management of machine learning models in production.
- **Practices:** Continuous integration/continuous deployment for machine learning, model versioning, automated model retraining.
- **Tools:** Kubeflow, MLflow, TFX (TensorFlow Extended).

Case Studies – Netflix

Problem:

- Netflix needed to provide a highly scalable, reliable, and performant streaming service to millions of users worldwide.
- The company faced challenges related to scalability, rapid feature deployment, and maintaining high availability.

Solution:

- **Microservices Architecture:**
- Netflix transitioned from a monolithic architecture to a microservices architecture.
- This allowed them to break down their application into smaller, independent services that could be developed, deployed, and scaled independently.

Case Studies – Netflix

- **CI/CD Pipelines:** Implemented robust CI/CD pipelines to automate the integration and deployment of code changes. This ensured that new features and updates could be released rapidly and reliably.

- **Tools:**

1) Jenkins:

- Used for continuous integration, Jenkins helps in automating the building and testing of code changes.

2) Spinnaker:

- Netflix developed Spinnaker, an open-source, multi-cloud continuous delivery platform that automates the process of releasing software changes.
- It integrates with Jenkins and other CI tools to facilitate automated deployment pipelines, enabling Netflix to release updates to production frequently and reliably.

Case Studies – Netflix

- **Infrastructure as Code (IaC):**

1) Terraform:

- Netflix uses Terraform for defining and provisioning infrastructure using code.
- This allows them to manage their AWS resources in a declarative way, ensuring that infrastructure is version-controlled and repeatable.

2) AWS CloudFormation:

- Utilized for automating the deployment of AWS resources, although Netflix has largely standardized on Terraform for its flexibility and multi-cloud support.

Case Studies – Netflix

- **Monitoring and Logging:**

1) Atlas: Netflix developed Atlas, an in-house telemetry and monitoring platform that scales to handle the massive amount of data generated by their services.

- It helps in real-time monitoring and alerting.

2) ELK Stack:

The ELK stack is used for centralized logging, allowing Netflix to aggregate logs from various services and analyze them for troubleshooting and operational insights.

- **Cloud-Based Infrastructure:** Migrated their infrastructure to the cloud (AWS), enabling elastic scalability and high availability.

Case Studies – Netflix – Cloud-Based Infrastructure

1) Amazon EC2 (Elastic Compute Cloud):

Netflix uses Amazon EC2 instances to run its various applications and services, including the backend for streaming, recommendation algorithms, and big data processing.

2) Amazon S3 (Simple Storage Service):

Amazon S3 is the primary storage solution for Netflix's vast library of video content, including movies, TV shows, and user-generated data.

3) Amazon RDS (Relational Database Service):

Netflix uses Amazon RDS for managing relational databases that store structured data, such as user account information, billing data, and metadata associated with video content.

Case Studies – Netflix – Key AWS Services used

4) Amazon DynamoDB:

DynamoDB is used for managing large-scale, low-latency data operations, such as session management, user activity tracking, and storing recommendation data.

5) Amazon CloudFront:

CloudFront is utilized by Netflix as a CDN to distribute content (videos, images, etc.) to users across the globe with minimal latency.

6) Amazon Route 53:

Route 53 is used for DNS management, directing user traffic to the nearest or most optimal data center, which is crucial for performance and availability.

Case Studies – Netflix

- **Outcome:**

- Achieved a highly available and resilient streaming service.
- Reduced downtime and improved user experience.
- Enabled rapid and frequent deployment of new features, enhancing their competitive edge.

Case Studies – Facebook

- **Problem:**
- Facebook required a method to deploy new features and updates rapidly without disrupting the service for its billions of users.

- They needed a solution that could support their massive scale and maintain high performance and reliability.

- **Solution:**
- **Continuous Integration:** Integrated CI practices to ensure that code changes were automatically tested and merged frequently, catching issues early and improving code quality.

- **Automated Deployment:** Implemented automated deployment processes to streamline the release of new features. This minimized manual intervention and reduced the risk of human error.

Case Studies – Facebook

- **Continuous Integration: Custom CI/CD Pipelines**
- Facebook has built its own CI/CD pipeline tools tailored to its specific needs.
- The company emphasizes frequent small releases, deploying code to production multiple times a day.

- **Phabricator:** An open-source suite of tools originally developed at Facebook.
- Phabricator is used for code review, repository hosting, and continuous integration, although Facebook has evolved beyond using it for CI/CD in recent years.

Case Studies – Facebook

- **Blue-Green Deployments**:
- Utilized blue-green deployment strategies to minimize downtime during releases. This involved running two identical production environments (blue and green) and switching traffic between them during deployments.

- **Infrastructure as Code (IaC):**
- Facebook manages its infrastructure through highly customized internal tools rather than relying on public IaC frameworks like Terraform.
- These tools are designed to work seamlessly with Facebook's custom-built servers and network equipment, offering tight integration with its unique environment.

Case Studies – Facebook

- **Robust Monitoring**:
- Established comprehensive monitoring and logging systems to track the performance and health of their applications in real-time.
- This allowed for quick detection and resolution of issues.

- **Scuba**:
- Facebook uses Scuba, a real-time data analysis tool for monitoring and troubleshooting its vast infrastructure.
- It allows engineers to query and visualize operational data in real-time.

- **LogDevice**:
- An internal distributed log storage system developed by Facebook, LogDevice is used to manage and analyze the huge volumes of logs generated across Facebook's infrastructure.

Cloud Native Application
Modern Application Requirements

Modern applications must meet several key requirements to ensure they are robust, scalable, and adaptable:

- **Scalability**: Ability to handle increasing loads by scaling up or down.
- **Resilience**: Capability to recover from failures and continue operating.
- **Agility**: Ease of deploying updates and new features quickly.

Cloud Native Application
Modern Application Requirements

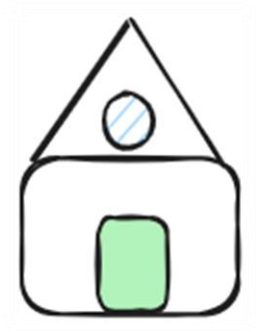
- **Portability**: Flexibility to run across different environments and cloud providers.
- **Security**: Robust security measures to protect data and ensure compliance.
- **Automation**: Integration of automated processes for deployment, monitoring, and management.

Cloud-Native Evolution

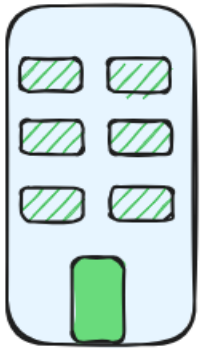
Cloud-native computing has evolved to address the limitations of traditional monolithic applications, promoting more modular, scalable, and resilient software architectures:

- **Monolithic to Microservices:** Transition from monolithic applications to microservices, where each service is independently deployable and scalable.
- **Virtual Machines to Containers:** Moving from virtual machines to containers for better resource efficiency and portability.

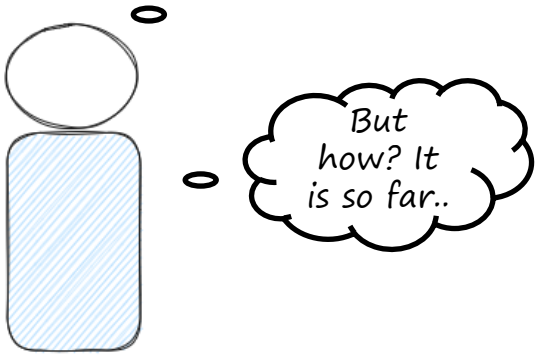
Virtualization – Real World Analogy



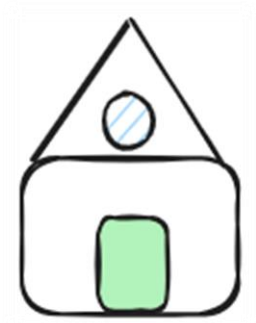
Home

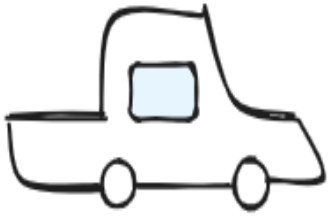


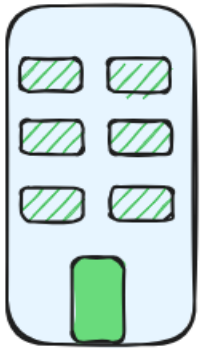
Office



Real World Analogy -private transport (car)



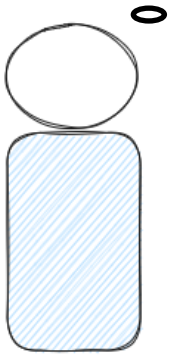




Home

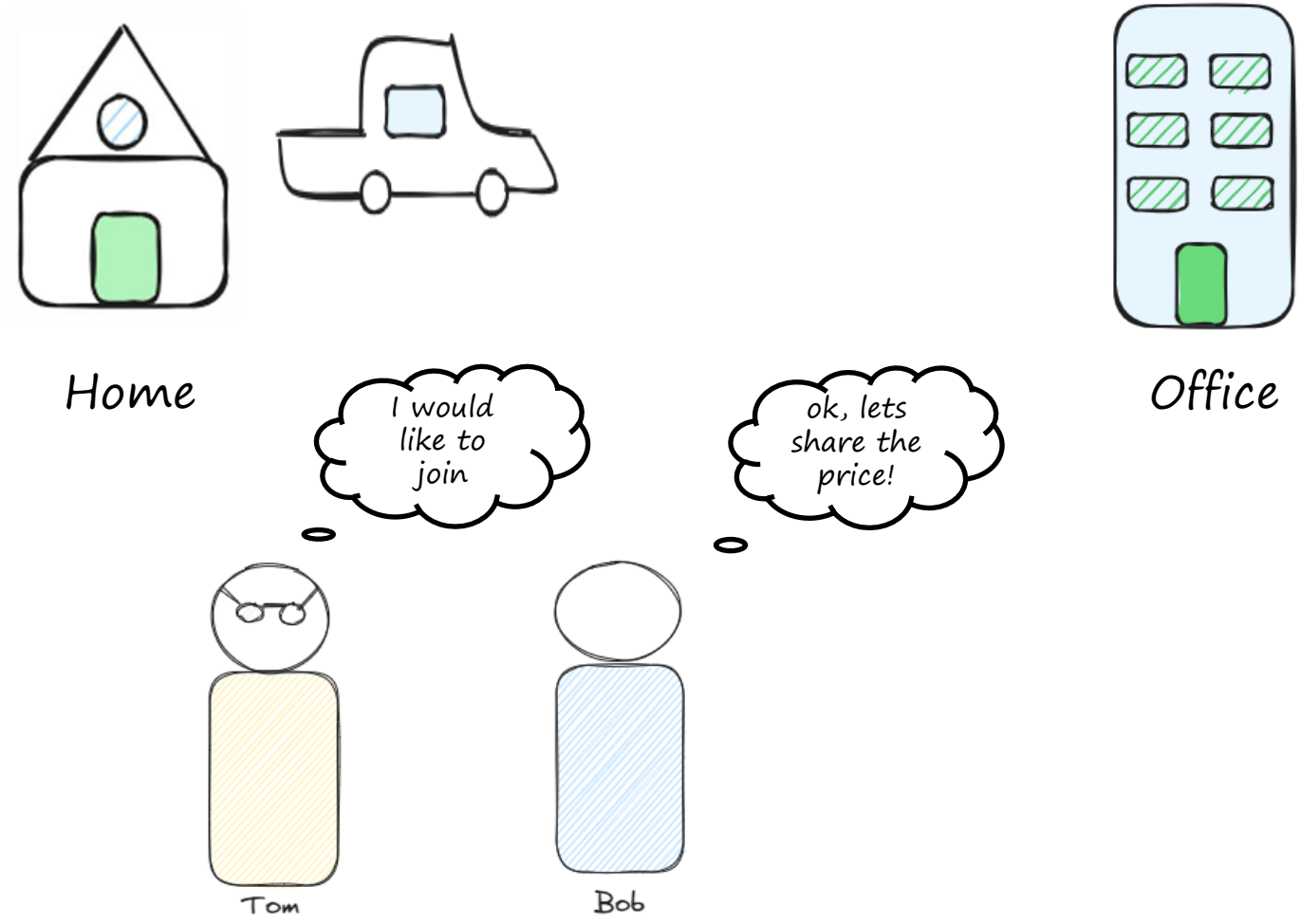
Do I need to buy a Car?

Office

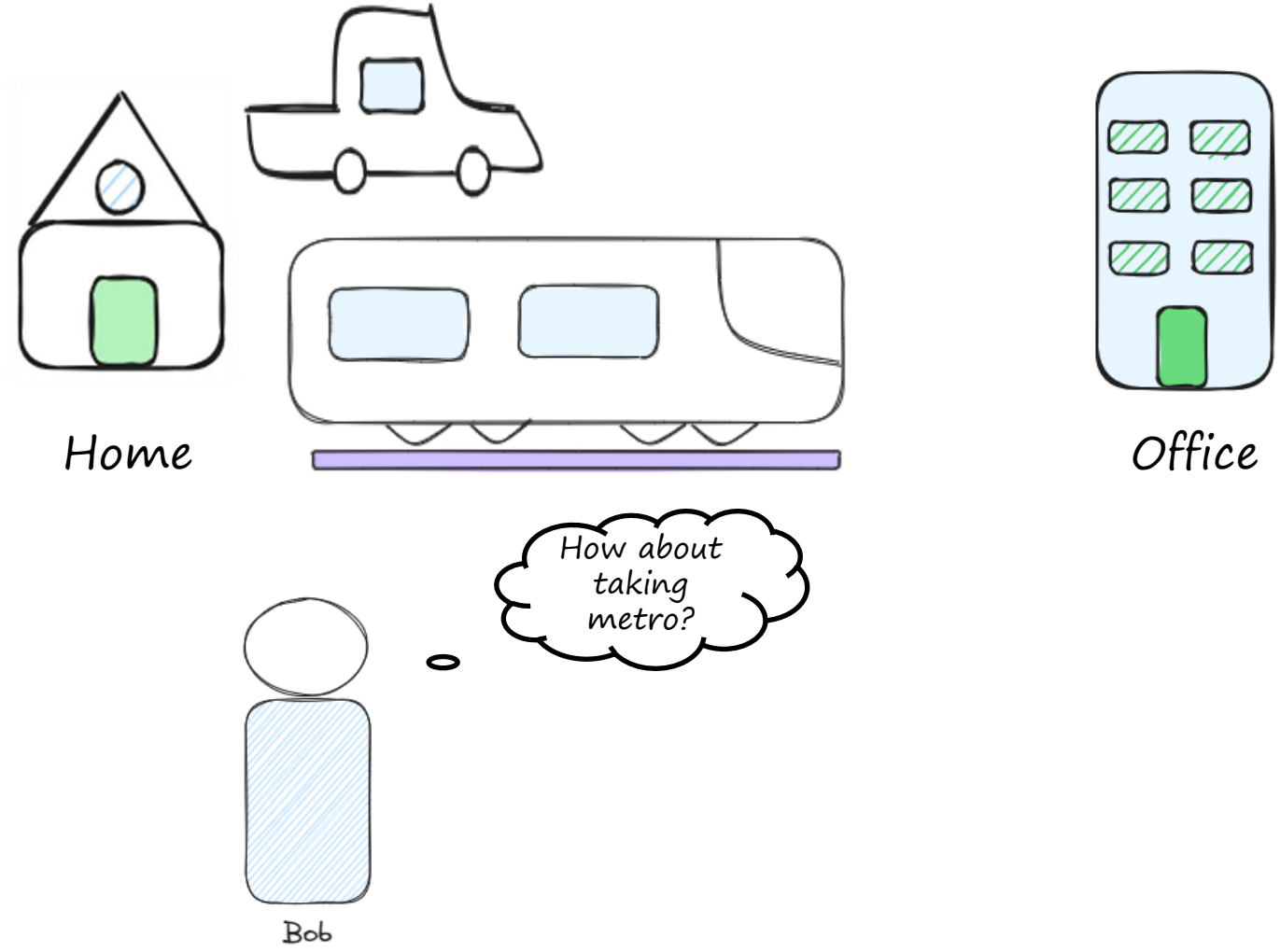


Bob

Real World Analogy - scalability

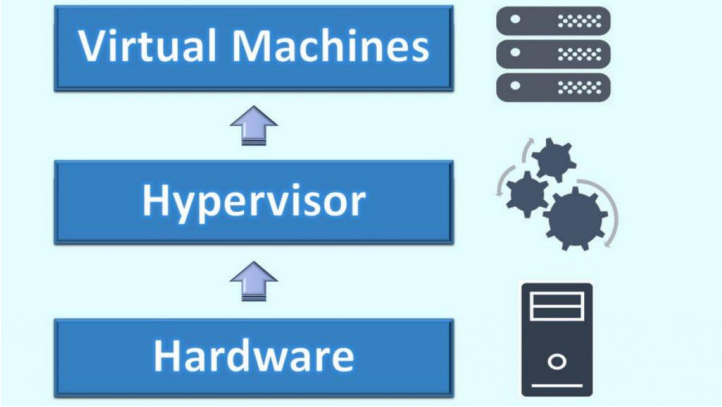


Real World Analogy – public transport (metro)



Hypervisor-based Virtualization

A hypervisor is a software layer that serves as an interface between the virtual machine (VM) and the underlying physical hardware(PM).



What does a hypervisor do?

- Ensures that each VM has access to the physical resources it needs to execute.
- It also ensures that the VMs don't interfere with each other by impinging on each other's memory space or compute cycles.

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 6

Recap

- Best Practices- clean code
- Overview of Xops
- Case Studies - Netflix, Facebook
- Modern application requirements

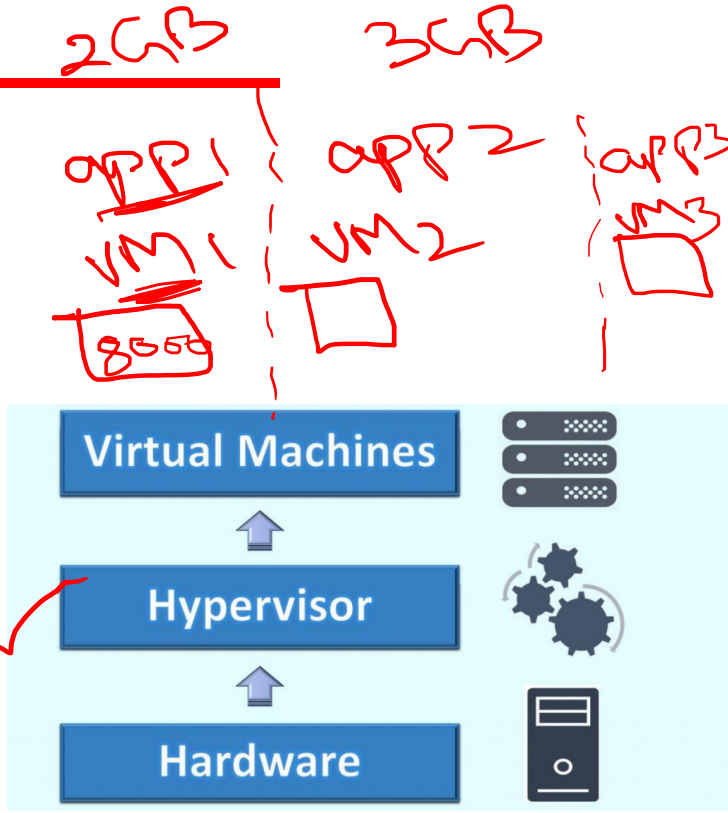
Agenda for today's class

innovate achieve lead

- Cloud-native evolution
- Introducing Cloud-native software
- Cloud-enabled vs Cloud-based vs Cloud-native apps
- Obstacles & enablers for Cloud-native apps
- CNCF Landscape
- Overview of Cloud-native ecosystem
  - Cloud DevOps
  - Microservices
  - Containers
- Building a Container
- Container Images
- Container Registries
- Dockerfile and Docker Compose
- Kubernetes Cluster Architecture

Hypervisor-based Virtualization

A hypervisor is a software layer that serves as an interface between the virtual machine (VM) and the underlying physical hardware(PM).



What does a hypervisor do?

- Ensures that each VM has access to the physical resources it needs to execute.
- It also ensures that the VMs don't interfere with each other by impinging on each other's memory space or compute cycles.

Cloud-Native Evolution

- **Manual to Automated Deployment:** Adopting CI/CD pipelines for automated and continuous delivery of applications.
- **On-Premises to Cloud:** Shifting from on-premises infrastructure to cloud environments for greater flexibility and scalability.

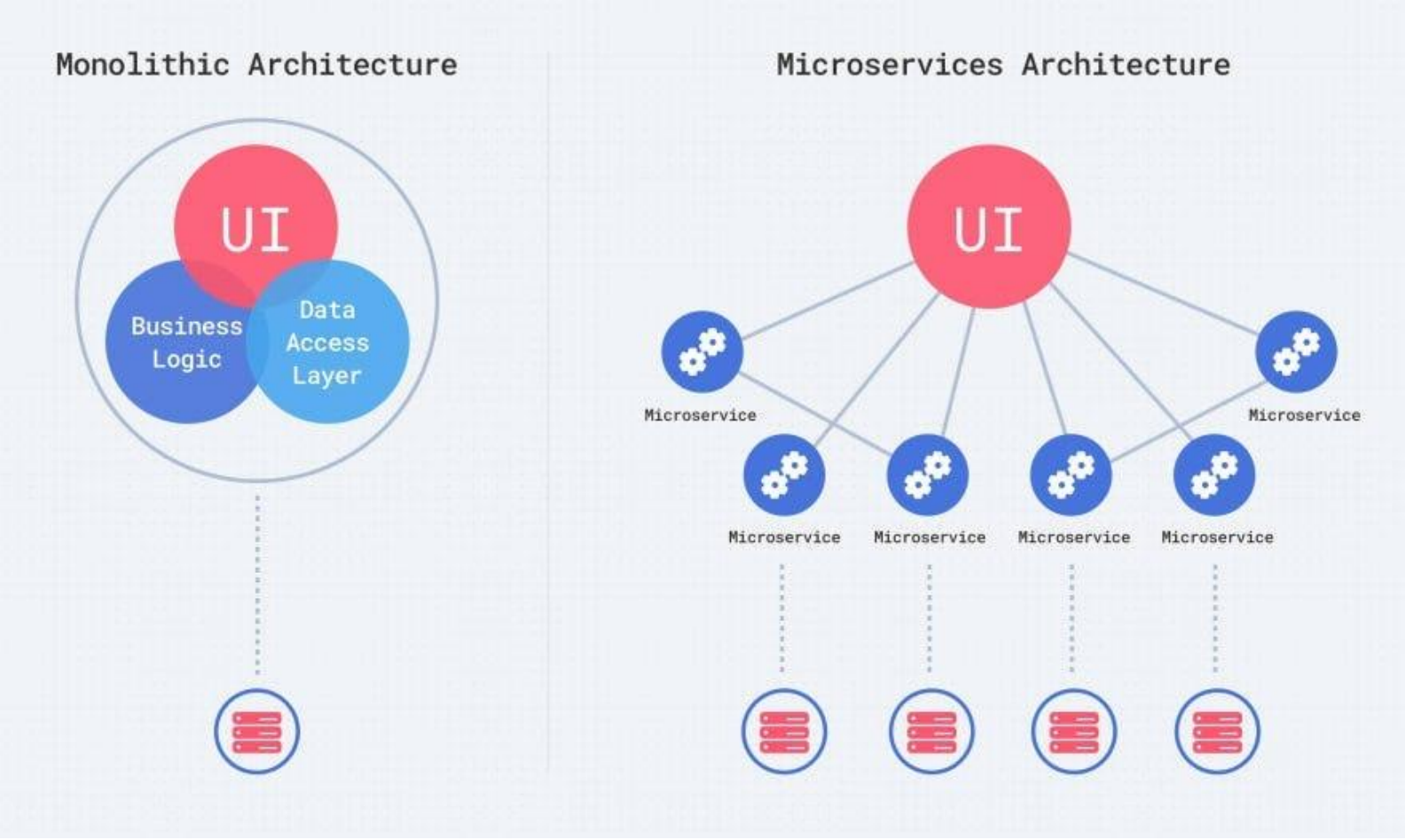
Introducing Cloud-native software

Cloud-native software is designed to leverage cloud infrastructure, providing enhanced agility, scalability, and resilience.

Key characteristics include:

- **Microservices Architecture:** Applications are composed of small, independent services that communicate through APIs.
- **Containers:** Each microservice runs in its own container, ensuring consistency across environments.

Monolithic vs Microservices



Introducing Cloud-native software

innovate achieve lead

Orchestration: Tools like Kubernetes manage container deployment, scaling, and operations.

DevOps Practices: Integration of development and operations teams to improve collaboration and efficiency.

Serverless Computing: Execution of code without managing servers, enabling automatic scaling and cost efficiency

Cloud-enabled vs Cloud-based vs Cloud-native apps

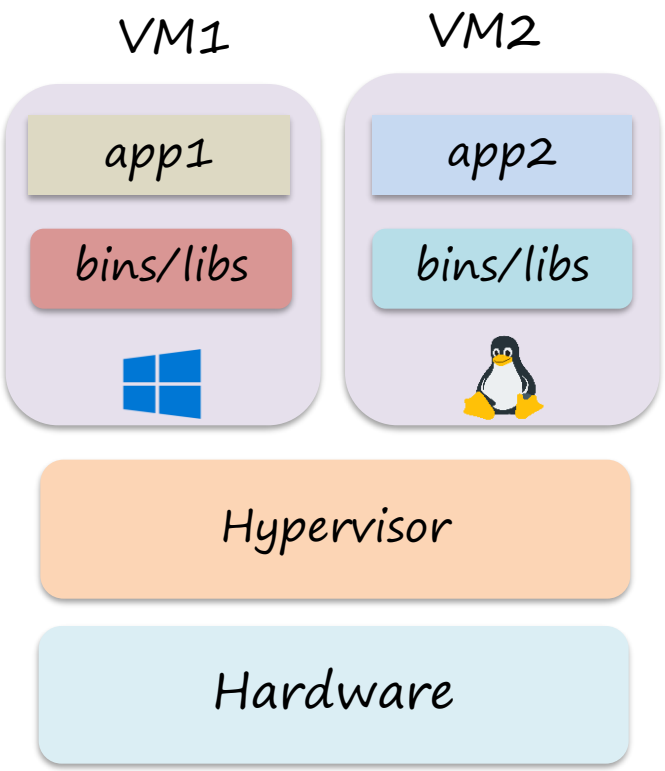
Cloud-Enabled Apps: Traditional applications modified to run on the cloud but not designed specifically for it. Limited scalability and flexibility.

- **Cloud-Based Apps:** Applications that primarily run in the cloud but may not fully leverage cloud-native principles. Better scalability than cloud-enabled apps.

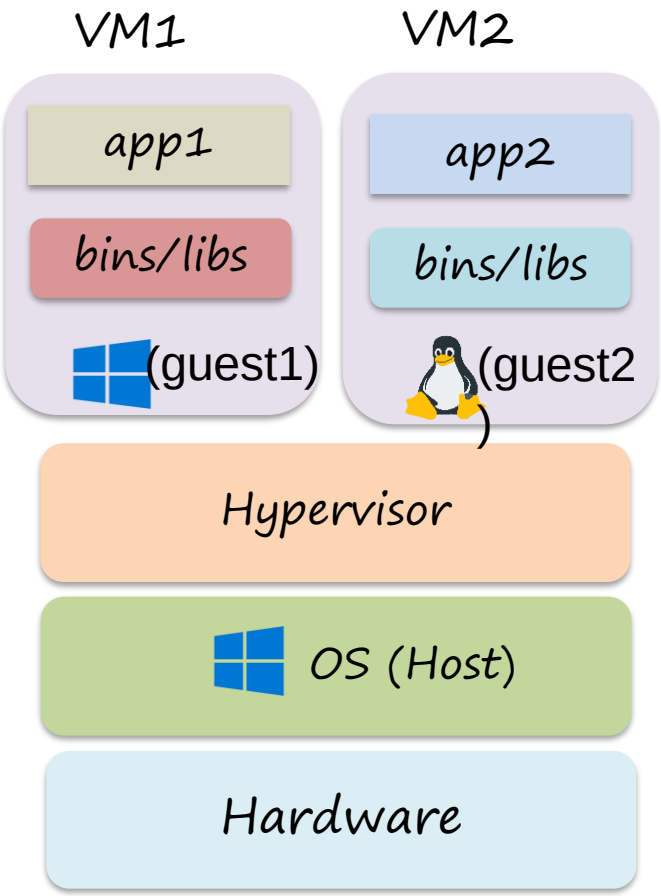
- **Cloud-Native Apps:** Applications designed from the ground up to run in the cloud, fully utilizing cloud services, scalability, and resilience.

Types of Hypervisors

Type 1 hypervisors

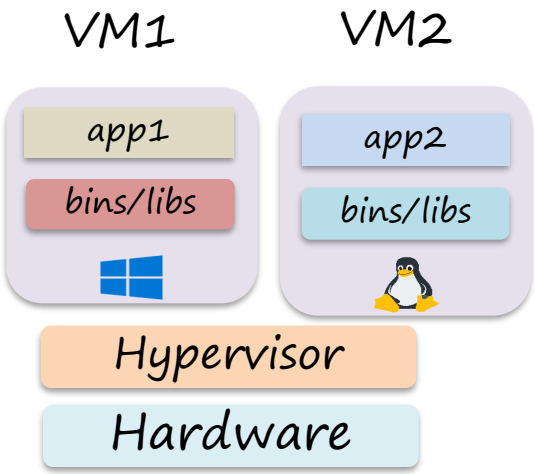


Type 2 hypervisors

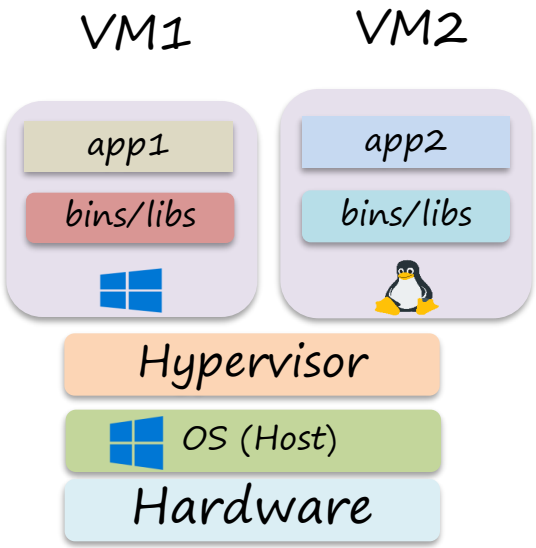


Need for Containers

Type 1 hypervisors



Type 2 hypervisors



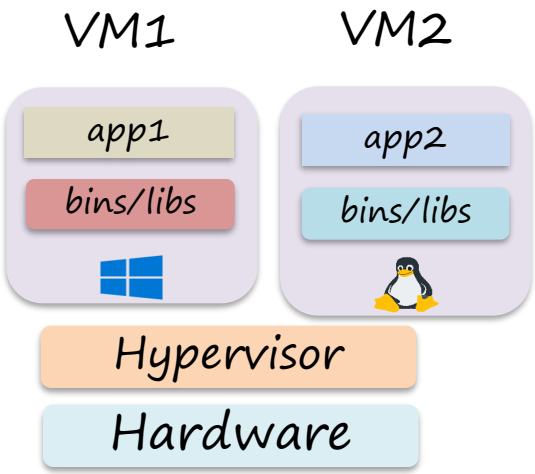
the virtual machines have an OS as well so are very heavy w.r.t file size

Ex: ubuntu image we downloaded last time was around 4GB

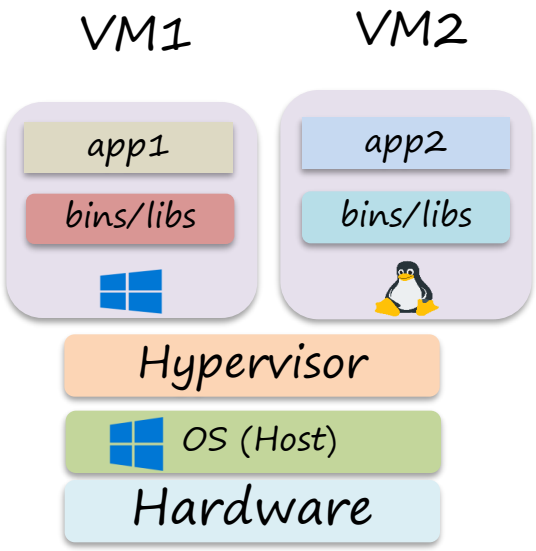
also need a lot of boot up time before we could run our app, can we do better?

Hypervisors vs Containers Visualized

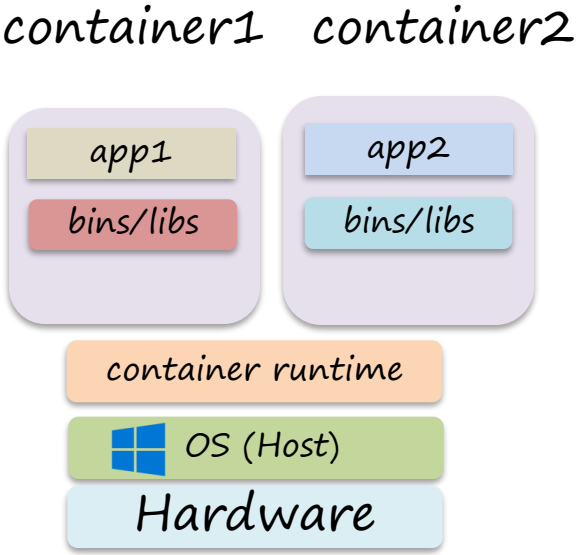
Type 1 hypervisors



Type 2 hypervisors

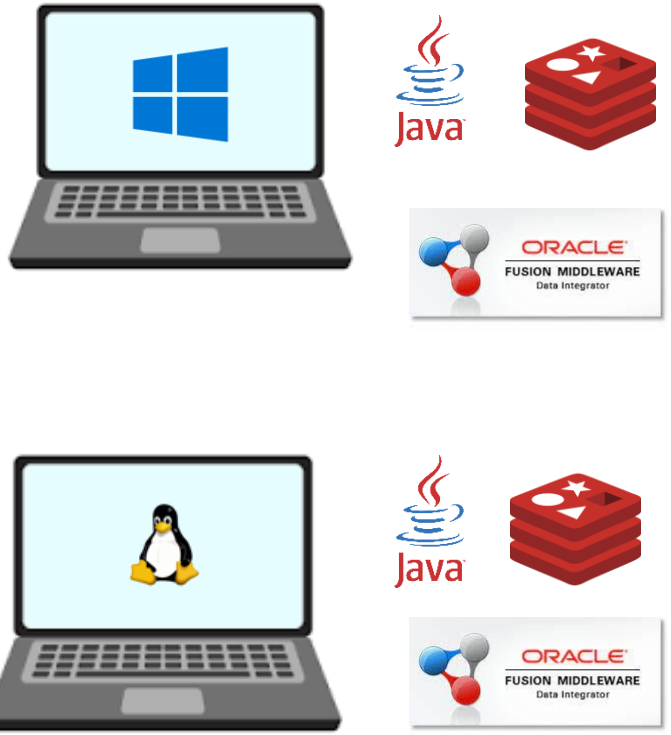


containers



Developer Onboarding – The Problem!

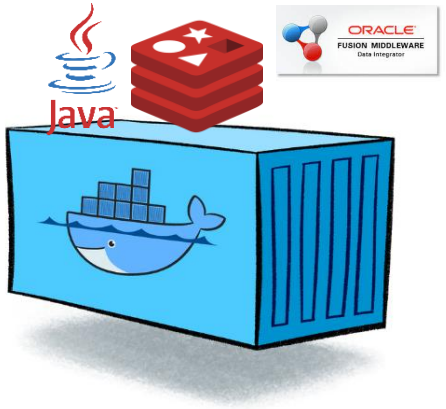
- lets say we have a dev team that works on data analytics product.
- So whenever a new developer on boards the team they need to setup all the tech stack needed for development.
- For example, an ETL tool like oracle data integrator and that needs jdk to be installed first and we also need a redis for caching.
- Now the steps needed to install these are specific to each operating system given to the developer and also there are many tools and technologies that one needs to manually install and configure and is error prone especially when they are new to the team!



Developer Onboarding – Docker container Accelerates!

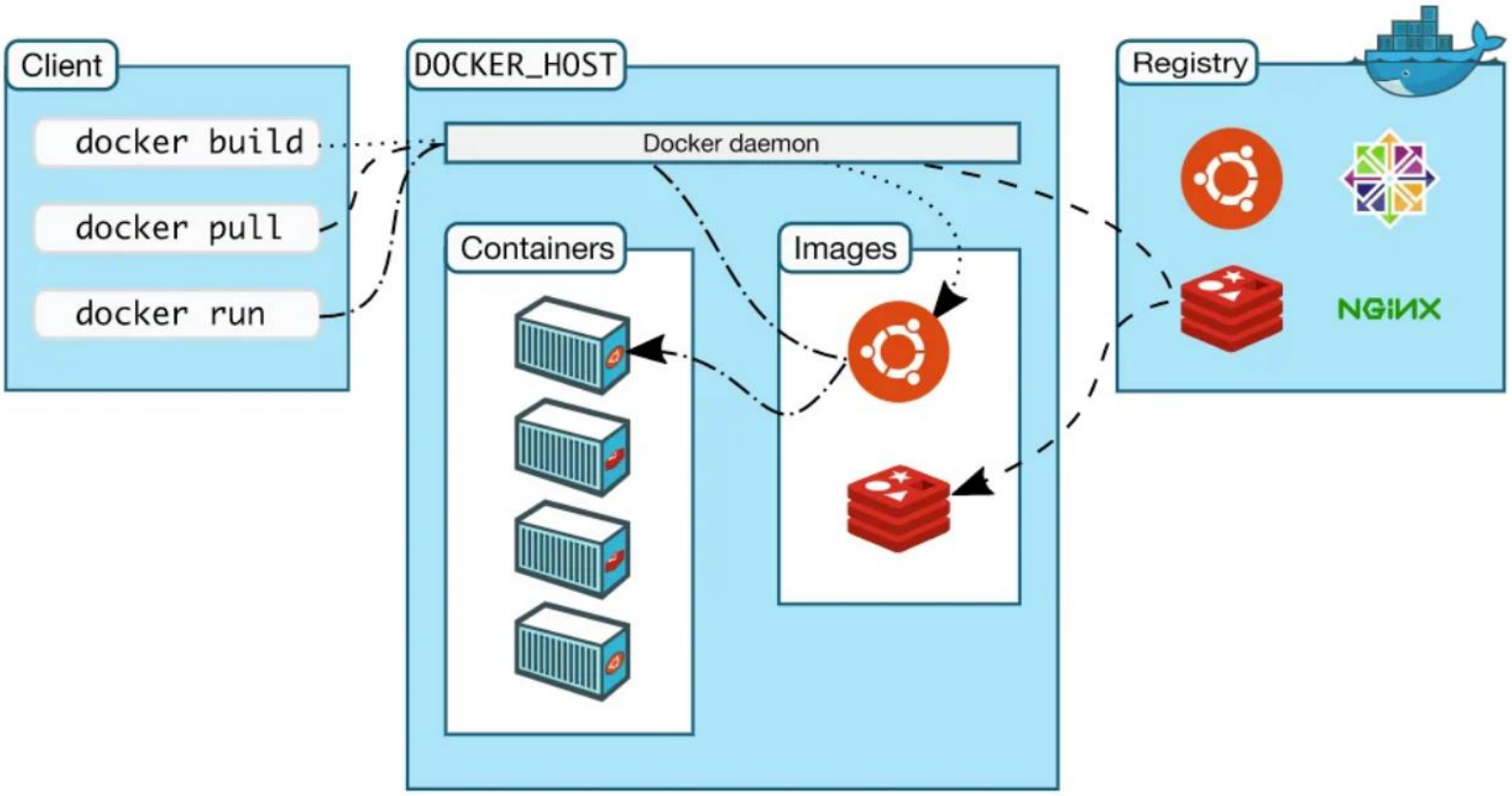
- with docker container , we can build one custom image containing all the tools and configurations needed and all developers need to do is to install docker runtime and then simply download and run these images.

- It accelerates the on boarding time needed wrt the setup of these complex tools
- It gives unified and consistent experience across all environments!



- be it dev or QA or customer prod environment!

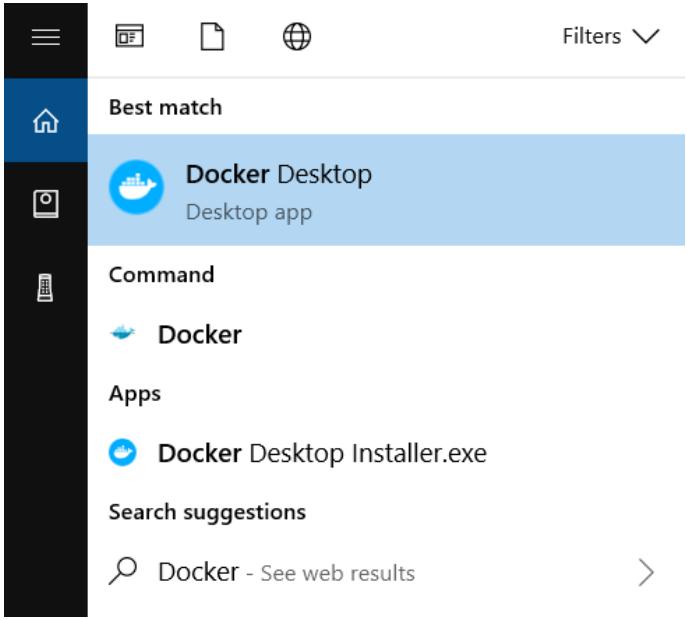
Docker architecture



Installing Docker on Windows 10

- In order to download and use docker, navigate to the official website of docker to download for windows - https://docs.docker.com/docker-for-windows/install/ and click on Docker Desktop for Windows as shown to the right.

- Once installed you should be able to find the docker desktop app in the search bar as shown to the right

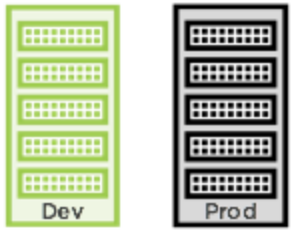


Advantages of using Docker

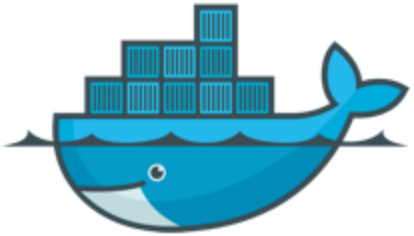
Docker Benefits
(for Web Developers)

Accelerate Developer Onboarding

Eliminate App Conflicts



Environment Consistency



Ship Software Faster

Cloud-native apps - Obstacles

- **Complexity**: Designing and managing microservices can be complex.
- **Cultural Shift**: Requires significant changes in organizational culture and processes.
- **Legacy Systems**: Integrating with or migrating from legacy systems can be challenging.
- **Security**: Ensuring security in a distributed and dynamic environment.

Cloud-native apps enablers

Automation Tools: CI/CD pipelines, infrastructure as code (IaC), and monitoring tools.

Container Orchestration: Platforms like Kubernetes streamline container management.

DevOps Culture: Collaboration between development and operations teams.

Cloud Services: Utilization of managed services and serverless platforms.

CNCF Landscape

The Cloud Native Computing Foundation (CNCF) hosts various open-source projects that form the backbone of the cloud-native ecosystem. Key projects include:

- **Kubernetes**: Container orchestration platform.
- **Prometheus**: Monitoring and alerting toolkit.
- **Envoy**: Service proxy for cloud-native applications.
- **Fluentd**: Open-source data collector for unified logging.

Overview of Cloud-native ecosystem – Cloud DevOps

Combines cloud computing with DevOps practices to automate and streamline software development, deployment, and operations:

- **CI/CD Pipelines:** Automate code integration and deployment.
- **Infrastructure as Code (IaC):** Manage infrastructure using code (e.g., Terraform, Ansible).
- **Monitoring & Logging**: Tools like Prometheus and ELK stack for real-time insights and troubleshooting.

Overview of Cloud-native ecosystem – Microservices

Architecture where applications are composed of small, loosely coupled services:

Independence: Each microservice can be developed, deployed, and scaled independently.

Communication: Services communicate via lightweight protocols like HTTP/REST or messaging queues.

Flexibility: Easier to update and maintain compared to monolithic architectures.

Overview of Cloud-native ecosystem – Container

Lightweight, portable units that package an application and its dependencies:

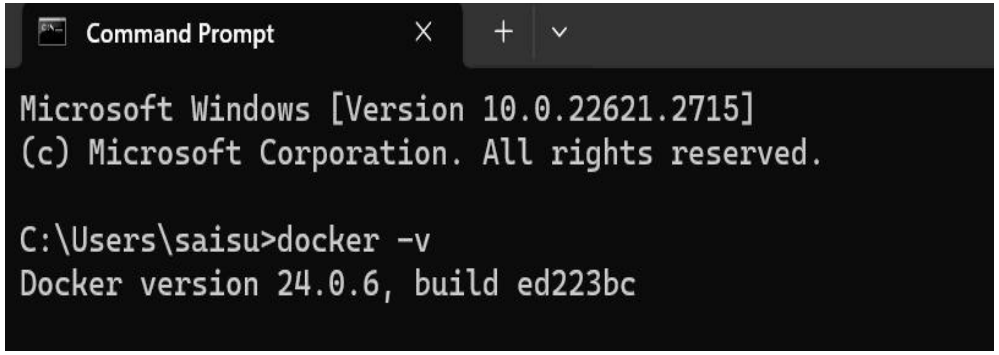
Isolation: Containers run isolated processes, ensuring consistency across environments.

Efficiency: More resource-efficient than virtual machines.

Portability: Run consistently on any environment with a container runtime (e.g., Docker).

Verifying Version of docker

- In general, there are no GUIs when we use docker on production environments which are based on linux operating systems.
- So, in order to practice the docker commands, we shall use command prompt from now on to create docker images and so on.
- To verify the installation of docker open command prompt from search menu and type docker -v to get the version of docker installed as shown



Our first docker command

- To check the list of docker images that are present in our current docker environment, we can run the below command:

docker images

- In our case as we haven't created any custom images or downloaded or run any OOTB(out of the box) images, it would show empty as shown below:

```
C:\Users\saisu>docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE

C:\Users\saisu>
```

Creating our first docker container

- To download an image from docker registry (docker hub), we can use the below command:

  docker pull <image_name>

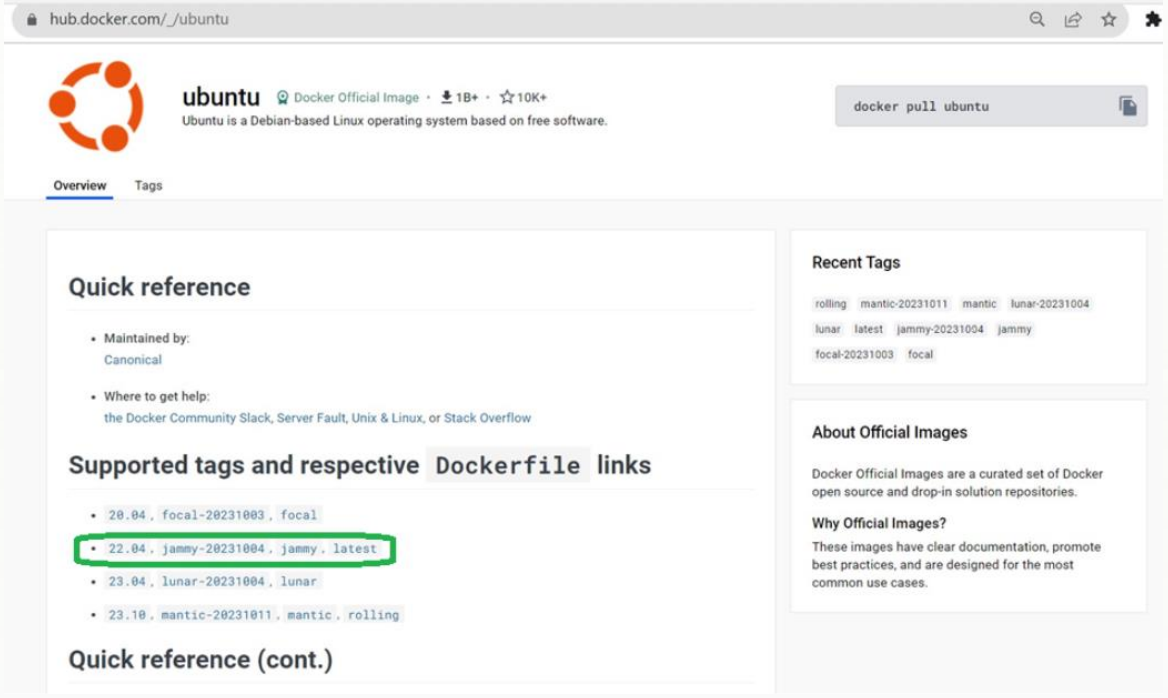
  Ex: docker pull ubuntu

```
C:\Users\saisu>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
aece8493d397: Pull complete
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu

C:\Users\saisu>
```

The first line says using default tag which is latest as we haven't specified any tag, it would try to download the latest version available that is 22.04 as shown above in green.

Creating our first docker container

This would download the docker image of ubuntu at docker hub - https://hub.docker.com/_/ubuntu

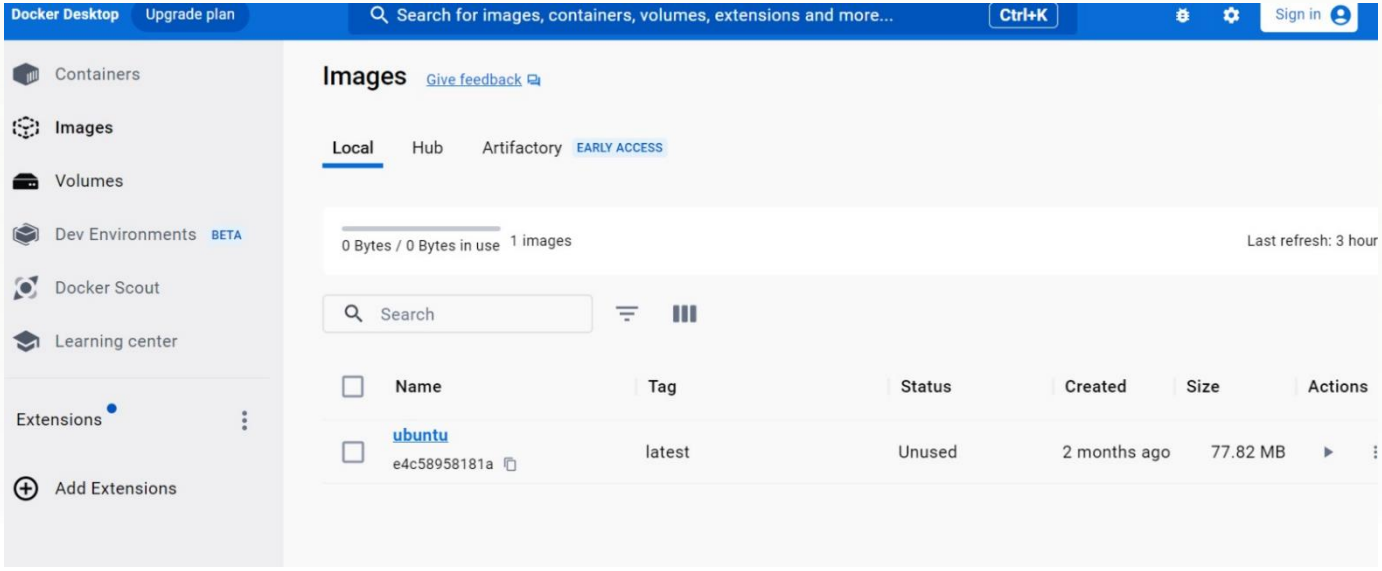


Creating our first docker container

We can re-run **docker images** to verify that the image got downloaded as shown below:

```
C:\Users\saisu>docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       latest    e4c58958181a   8 weeks ago    77.8MB
```

We can also verify the same in docker desktop app in windows under images section as shown below:



Creating our first docker container

- This only downloads the ubuntu image but doesn't run anything!
- We can use the following command to get list of containers:

  docker ps

```
C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\saisu>
```

- As expected this doesn't show any containers yet, to create a container (runtime instance of an image) using below command:

  docker run --name <container_name> <image_name>

Creating our first docker container – ubuntu

- Let's create a container that runs ubuntu operating system by running the ubuntu docker image we downloaded earlier

  docker run --name ubuntu1 ubuntu

- this would create a container from the image **ubuntu** with the name of the container as **ubuntu1**
- we can check the same again using docker ps

```
C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\saisu>
```

Where is our ubuntu container?

- docker ps list only running containers, to get the list of all containers lets use -a option in the docker ps command as shown below:

  docker ps -a

- This shows a list of containers that are both running and stopped.

```
C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\saisu>
```

Exercises:

innovate achieve lead

1. How to install LXC on Ubuntu?
    1. Visit https://ubuntu.com/server/docs/containers-lxc
    2. Follow the instruction given to install LXC

2. What is docker registry?
    1. Visit the Docker Hub and understand it's features: https://hub.docker.com/
    2. Understand more from Docker documentation: https://docs.docker.com/docker-hub/

docker vs VMs wrt compatibility

- we can run virtual machine image of any operating system on any of the operating system hosts.
- Ex: we downloaded and ran ubuntu os image on windows host using oracle virtual box.
- On the other hand we cannot run linux os based docker images on windows host.
- Ex: windows 7 or 8

docker vs VMs wrt compatibility

- we say a docker container can run natively on a particular operating system when the kernel of the host operating system supports running the docker image.
- recent versions of windows started supporting running of docker images natively!
- Ex: windows 10,11

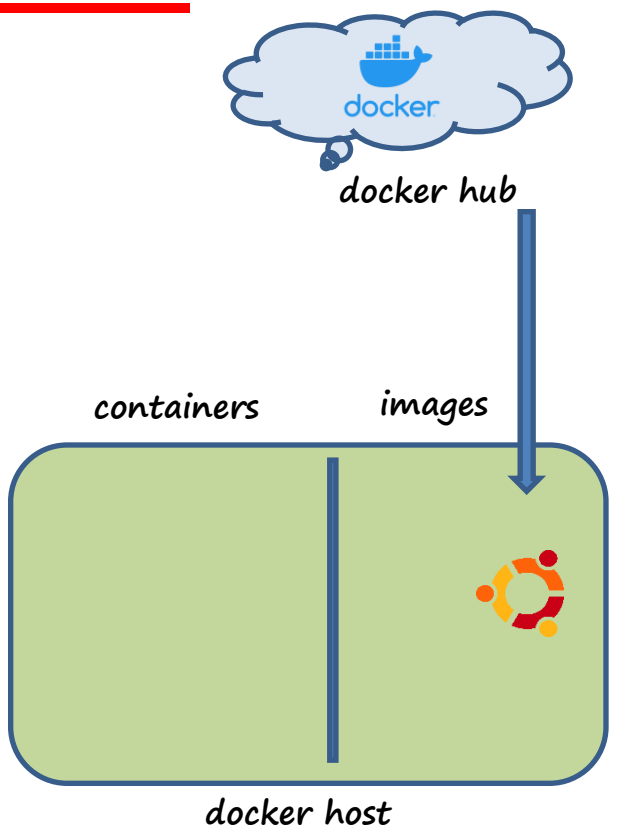
Creating our first docker container

- To download an image from docker registry (docker hub), we can use the below command:

  docker pull <image_name>

  Ex: docker pull ubuntu

```
C:\Users\saisu>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
aece8493d397: Pull complete
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu

C:\Users\saisu>
```



The first line says using default tag which is latest as we haven't specified any tag, it would try to download the latest version available that is 22.04 as shown above in green.

Creating our first docker container – ubuntu

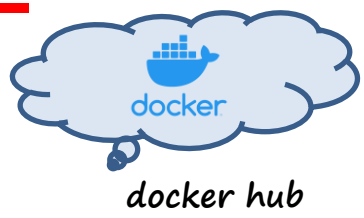
innovate achieve lead

- Let's create a container that runs ubuntu operating system by running the ubuntu docker image we downloaded earlier

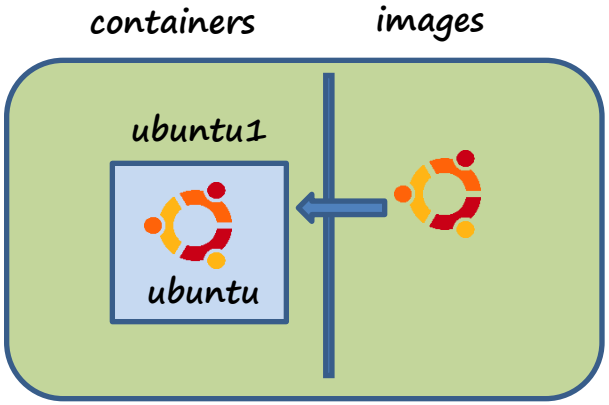
  docker run --name ubuntu1 ubuntu

- this would create a container from the image ubuntu with the name of the container as ubuntu1

- we can check the same again using docker ps



docker hub



docker host

```
C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\saisu>
```

Where is our ubuntu container?

- docker ps list only running containers, to get the list of all containers lets use -a option in the docker ps command as shown below:

  docker ps -a

- This shows a list of containers that are both running and stopped.

```
C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\saisu>
```

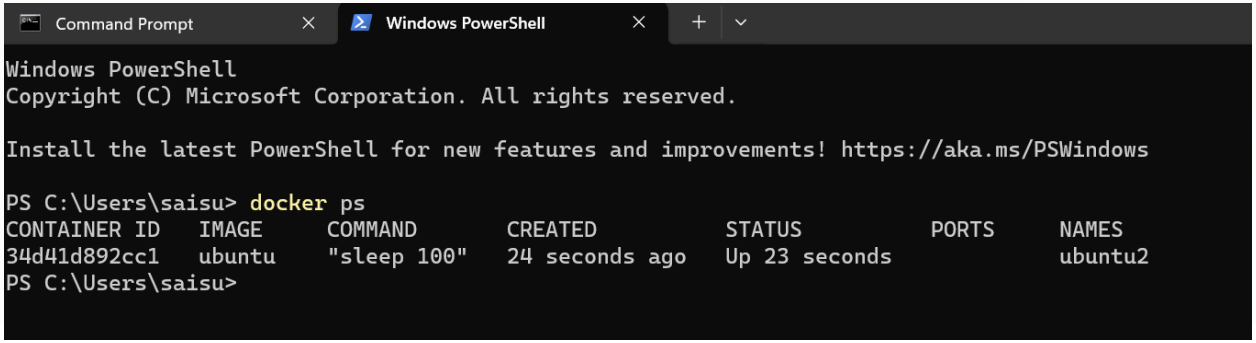
```
C:\Users\saisu>docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
c0f7ff0bb4a4   ubuntu    "/bin/bash"   22 seconds ago   Exited (0) 21 seconds ago             ubuntu1
```

Why is the ubuntu container not running?

- container runs only till underlying process runs, in this case there is no running process after starting the os.
- Also, containers are not meant to host os unlike VMs, they are meant to host web server or database or app server etc
- Let's use a trick to keep the container alive for some time! sleep!
- We can use sleep command to keep the container alive , lets use sleep for 100 seconds

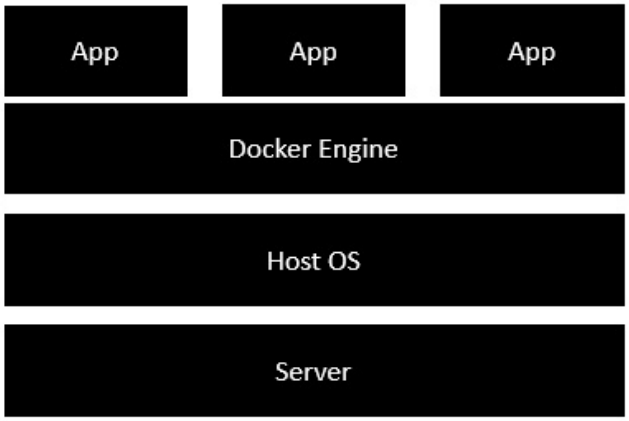
docker run --name ubuntu2 ubuntu sleep 100

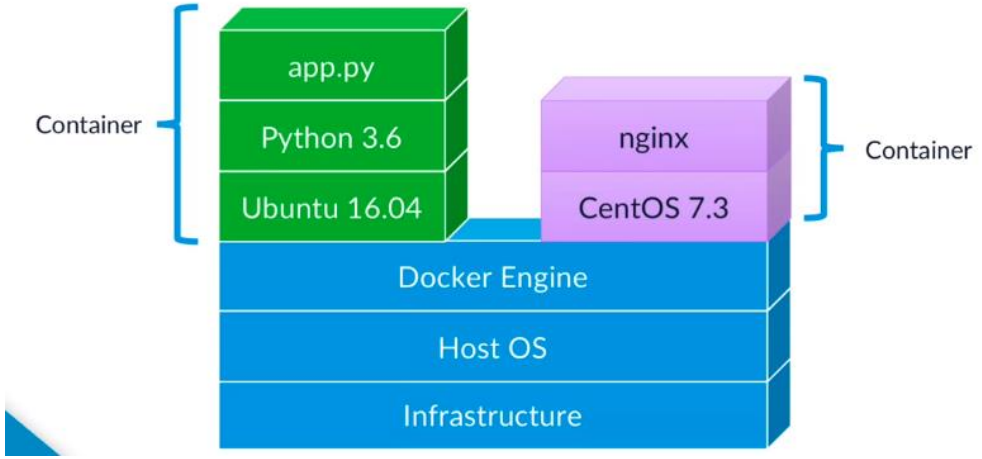
this will hang the current window prompt for 100 seconds ,let's open another command prompt and run docker ps



Docker Containers – An example

- The server is the physical server that is used to host multiple virtual machines
- The Host OS is the base machine such as Linux or Windows.
- Now comes Docker engine. This is used to run the operating system (which earlier used to be virtual machines).
- All of the Apps now run as Docker containers.





running a container in background

- As seen earlier sometimes depending on the container process the current window prompt would hang (for example when wd did sleep 100 , it hanged for 100 seconds)

- In order to run the container in background so that it doesnt block our command prompt when running using docker run we can use option **-d** which stands for **detach**.

docker run --name ubuntu3 -d ubuntu sleep 100

Now the container runs in background and we get back a prompt immediately with the container id that gets created (2bcf... as shown above) and can run docker ps to check in the same command prompt!

Creating our second docker container - nginx

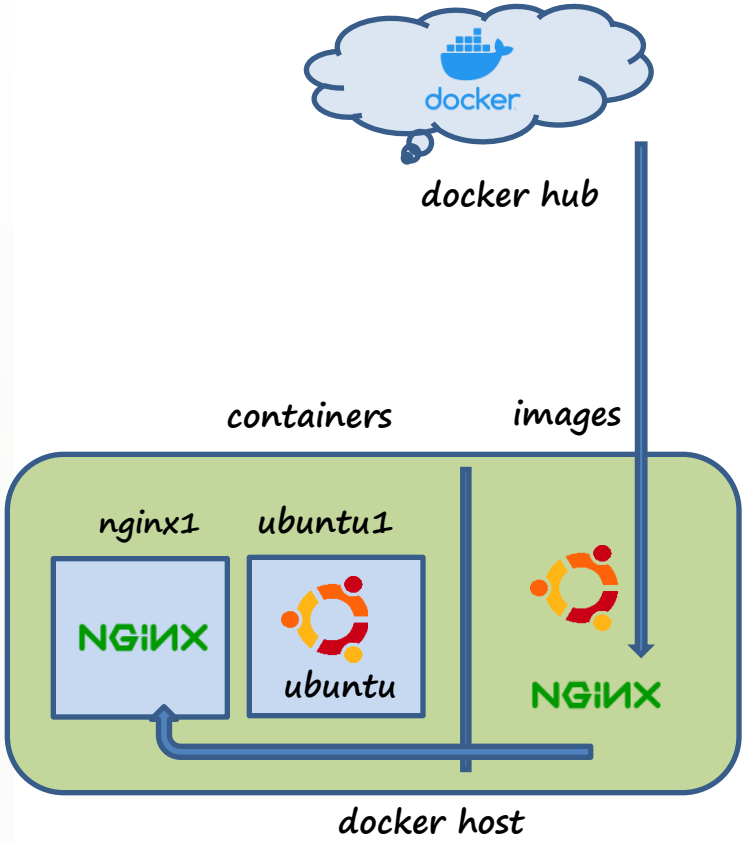
- Nginx (pronounced "engine-x") is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server (origin server)
- Lets download the nginx image using docker pull and run a container out of it using docker run
- In fact we can pull and run an image using docker run itself , let's do it for getting nginx image and spinning up a container with this image and name as nginx1:

  docker run --name nginx1 nginx

```
C:\Users\saisu>docker run --name nginx1 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
af107e978371: Downloading [==================================>                ]  22.83MB/29.13MB
336ba1f05c3e: Download complete
8c37d2ff6efa: Download complete
51d6357098de: Download complete
782f1ecce57d: Download complete
5e99d351b073: Download complete
7b73345df136: Download complete
```

- Notice the first line where it tries to find the image nginx locally and since it wasn't downloaded , it would download from docker registry and then this will start the nginx container named nginx1 and block the command prompt

  docker run --name nginx2 -d nginx



Executing commands on docker container

- To run some command on the container from command line interface (cli), we can use the below command:

docker exec <container_name> <command_to_be_run>

- Ex: docker exec -it nginx2 /bin/bash
  docker exec nginx cat /etc/os-release

```
C:\Users\saisu>docker exec nginx2 cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
NAME="Debian GNU/Linux"
VERSION_ID="12"
VERSION="12 (bookworm)"
VERSION_CODENAME=bookworm
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

C:\Users\saisu>
```

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 7

Recap

- Virtualization – VMs vs Containers
- Introducing Cloud-native software
- Cloud-Native Evolution
- Monolithic vs Microservices
- Hypervisors and Container Runtime
- Container Images
- Container Registries
- Creating docker containers
  - ubuntu
  - nginx

Agenda for today's class

innovate achieve lead

- Containers Continued
  - Executing commands
  - Inspecting containers
  - Running MYSQL containers and port binding
  - Clean up of containers
  - Creating a custom docker image – Dockerfile
  - Building and Running Custom Image
- Traditional CI setup
- Building the application

Executing commands in docker container

- To login to container we can use -it option which stands for interactive terminal:

  docker run -it nginx echo "hello"

```
C:\Users\saisu>docker run -it nginx echo "hello"
hello

C:\Users\saisu>
```

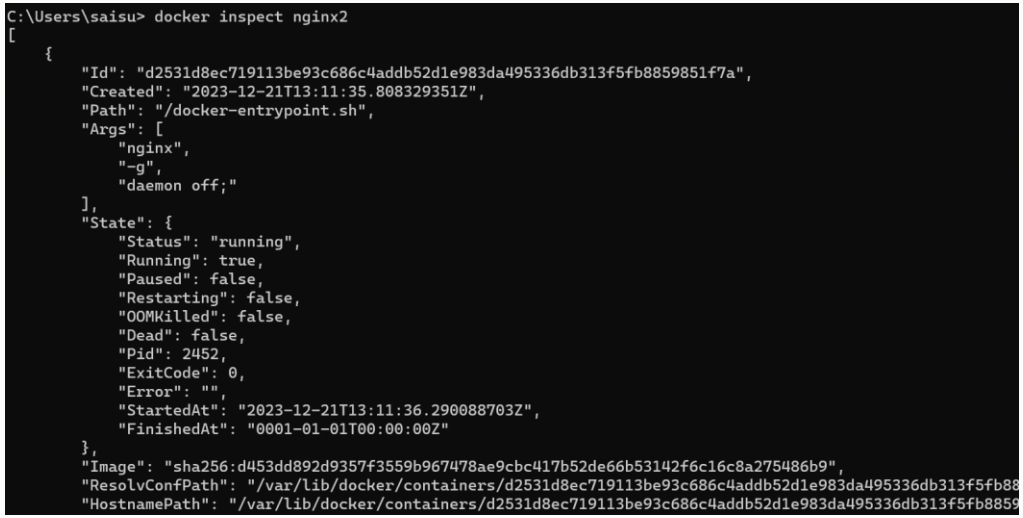
inspecting our docker container – getting metadata

- To inspect the docker container that got created, we can use docker inspect as shown below:

  docker inspect <container_name>

  Ex: docker inspect nginx2

- This prints all the details (metadata) about the container.



getting logs from our docker container

- To check the logs of the docker container that got created, we can use docker logs as shown below:

  docker logs <container_name>

  Ex: docker logs nginx2

```
C:\Users\saisu>docker logs nginx2
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/12/21 13:11:36 [notice] 1#1: using the "epoll" event method
2023/12/21 13:11:36 [notice] 1#1: nginx/1.25.3
2023/12/21 13:11:36 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2023/12/21 13:11:36 [notice] 1#1: OS: Linux 5.15.133.1-microsoft-standard-WSL2
2023/12/21 13:11:36 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/12/21 13:11:36 [notice] 1#1: start worker processes
2023/12/21 13:11:36 [notice] 1#1: start worker process 29
2023/12/21 13:11:36 [notice] 1#1: start worker process 30
2023/12/21 13:11:36 [notice] 1#1: start worker process 31
2023/12/21 13:11:36 [notice] 1#1: start worker process 32
2023/12/21 13:11:36 [notice] 1#1: start worker process 33
2023/12/21 13:11:36 [notice] 1#1: start worker process 34
2023/12/21 13:11:36 [notice] 1#1: start worker process 35
2023/12/21 13:11:36 [notice] 1#1: start worker process 36

C:\Users\saisu>
```

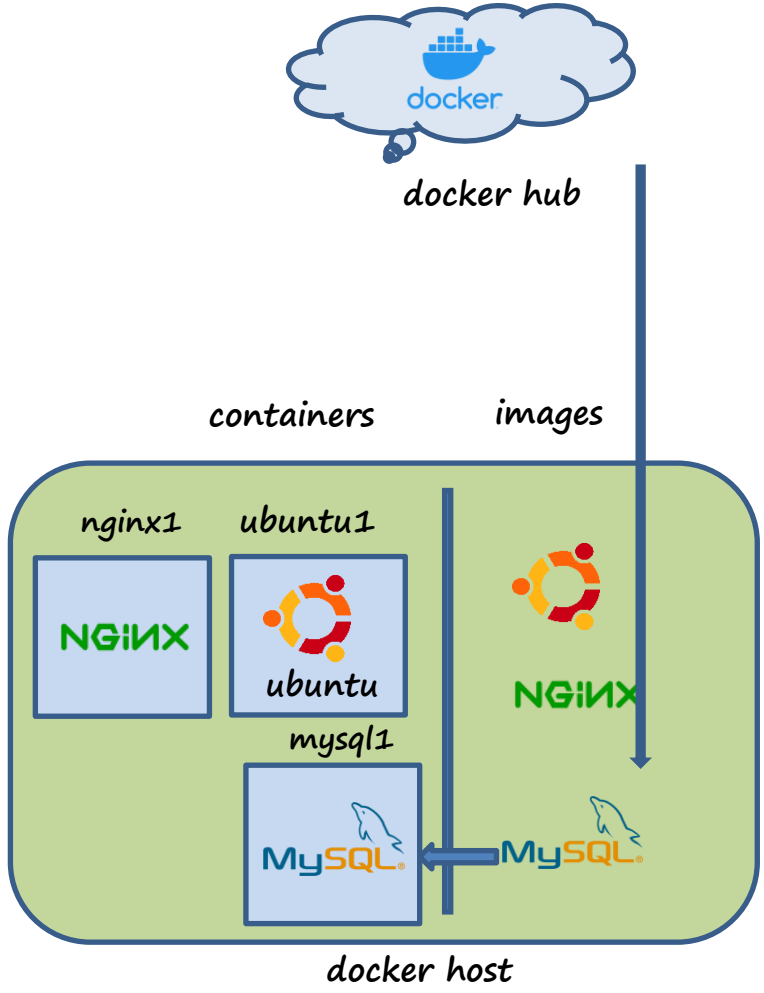
creating our third container – mysql db

- Let us repeat the above steps to create a container that runs mysql

  docker run –name mysql1 -d mysql

- this will again download the mysql image as it cant find the image locally and run a container with the name mysql1 but notice that the container exits immediately, Why?

```
C:\Users\saisu>docker run --name mysql1 -d mysql
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
bce031bc522d: Pull complete
cf7e9f463619: Pull complete
105f403783c7: Pull complete
878e53a613d8: Pull complete
2a362044e79f: Pull complete
6e4df4f73cfe: Pull complete
69263d634755: Pull complete
fe5e85549202: Pull complete
5c02229ce6f1: Pull complete
7320aa32bf42: Pull complete
Digest: sha256:2d82ba6960cdf254de13f57d7265570c0675bc8bae1051e4a9806cff863006c9
Status: Downloaded newer image for mysql:latest
0c14275430f023c8ecc0ae641f015011c0be237e2c6f7bd7977f3b0fca5fd82a

C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
d2531d8ec719   nginx     "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   80/tcp    nginx2
```



debugging our third container – mysql db

We shall use docker logs command to check the logs to see if we can find the reason for it exiting: **docker logs mysql1**

```
C:\Users\saisu>docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
0c14275430f0   mysql     "docker-entrypoint.s…"   14 seconds ago   Exited (1) 8 seconds ago              mysql1
d2531d8ec719   nginx     "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes               80/tcp    nginx2
5d6d229c9dda   ubuntu    "sleep 100"              15 minutes ago   Exited (0) 13 minutes ago             ubuntu3
dd2e85b6afb8   nginx     "/docker-entrypoint.…"   4 hours ago      Exited (0) 15 minutes ago             nginx1
34d41d892cc1   ubuntu    "sleep 100"              4 hours ago      Exited (0) 4 hours ago                ubuntu2
c0f7ff0bb4a4   ubuntu    "/bin/bash"              4 hours ago      Exited (0) 4 hours ago                ubuntu1

C:\Users\saisu>docker logs mysql1
2023-12-21 13:22:21+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.2.0-1.el8 started.
2023-12-21 13:22:22+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2023-12-21 13:22:22+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.2.0-1.el8 started.
2023-12-21 13:22:22+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
    You need to specify one of the following as an environment variable:
    - MYSQL_ROOT_PASSWORD
    - MYSQL_ALLOW_EMPTY_PASSWORD
    - MYSQL_RANDOM_ROOT_PASSWORD

C:\Users\saisu>
```

logs show that the image expects environment variables to be passed namely: **MYSQL_ROOT_PASSWORD**

to pass the environment variables docker run provides -e option,
lets pass mysql123 as password

docker run –name mysql2 -d -e MYSQL_ROOT_PASSWORD=mysql123 mysql

running mysql db container in background


```
C:\Users\saisu>docker run --name mysql2 -d -e MYSQL_ROOT_PASSWORD=mysql123 mysql
09aa4373eff688ba881e4354d40644dfedfcb2d08d0305b918aef3d3690b36ed
```

Now if we check the running containers using docker ps ,
we can see the mysql2 container running!

```
C:\Users\saisu>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                 NAMES
09aa4373eff6   mysql     "docker-entrypoint.s…"   9 seconds ago    Up 7 seconds    3306/tcp, 33060/tcp   mysql2
d2531d8ec719   nginx     "/docker-entrypoint.…"   18 minutes ago   Up 18 minutes   80/tcp                nginx2
```

Notice that the database runs on port 3306 by default as shown under ports of the docker ps output above and is the same port mapped to the host machine (your laptop) as well.

Lets now understand how docker container ports map to host ports

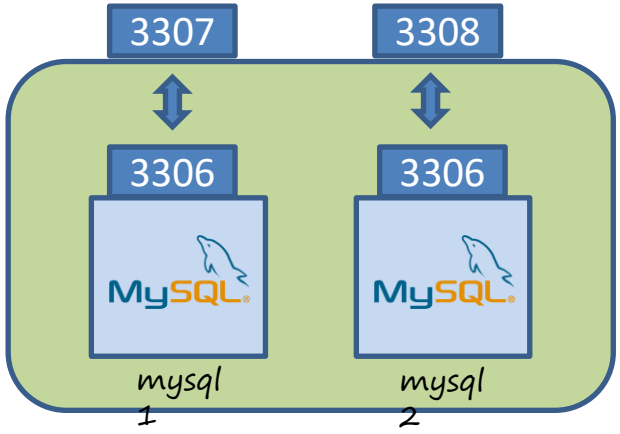
binding container port to host port

in order to connect to our mysql db container from a tool on host like mysql work bench , we need to map/bind container port to host port.

By default port 3306 is the container port for mysql db.

If we wanted to map the 3306 port of container to port of the host say 3307 we can use -p option followed by host_port:container_port of docker run as

docker run --name <container_name> -d -e <key1> <val1> -p <host_port>:<container_port> <image_name> <cmd> <args>



clean up of containers

In order to remove containers from docker , we can run docker rm command using the container name.

But before that we need to stop if the container is running using

docker stop <container_name>

docker rm <container_name>

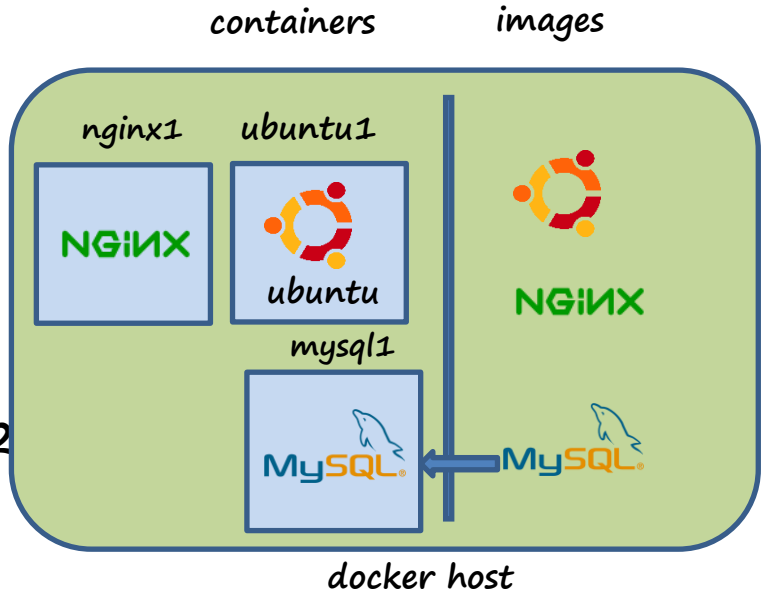
Ex: docker stop mysql1

docker rm mysql1

We can also remove multiple containers at once using rm command as shown below:

docker stop nginx1 nginx2 ubuntu1 ubuntu2
docker rm nginx1 nginx2 ubuntu1 ubuntu2

docker hub



docker host

clean up of images

- In order to remove images from docker , we can run docker rmi command using the container name.

docker rmi <image_name>

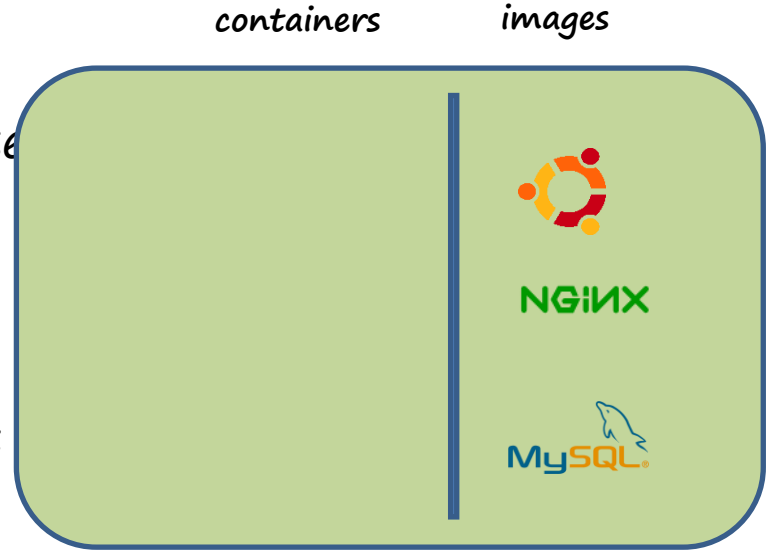
Ex: docker rmi mysql

- this removes latest version of image if exists for mysql
- To remove an image of specific version use

docker rmi <image_name>:<version>

- We can also remove multiple images at once using rm command as shown below:
  docker rmi nginx ubuntu

docker hub



docker host

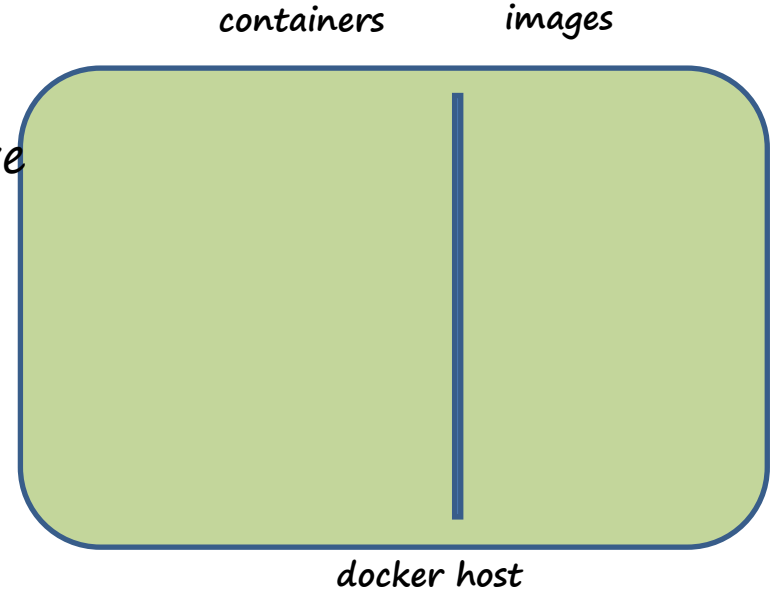
clean up and custom image

- this brings us back to original state we started with

docker hub

- What if we wanted to create a single container that has all the above softwares running like ubuntu, nginx, mysql?

- we do not have a direct image that has combination of these on docker hub , but we can create a custom image in this case by creating a Dockerfile.



Creating a custom docker image

- Dockerfile specifies all the steps defining what all are needed to be done on the container once created.

- To create a Dockerfile use below command

  touch Dockerfile

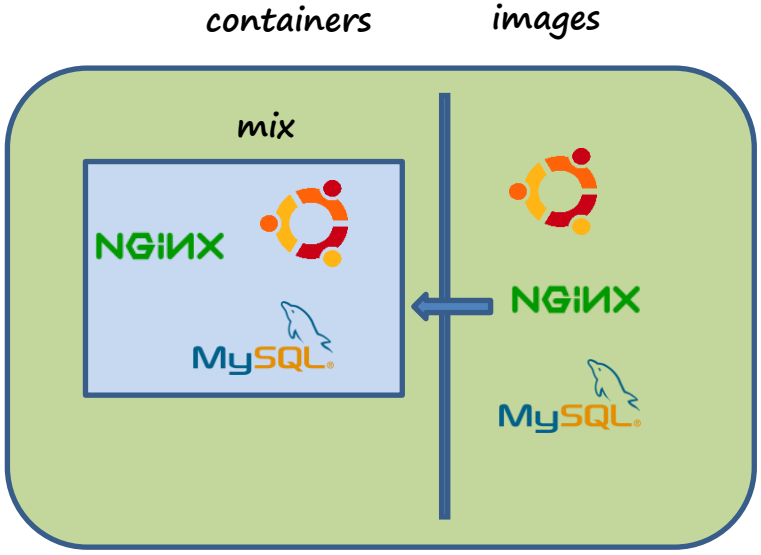
- Format of lines in Dockerfile:

  <instruction> <argument>

  Ex: FROM Ubuntu

- Here instruction is FROM and argument is Ubuntu

docker hub



docker host

Creating a custom docker image - Dockerfile

Command	Description	Example
FROM	Specifies the base image to use for the Docker image	FROM python:3.9-slim
WORKDIR	Sets the working directory inside the container	WORKDIR /app
COPY	Copies files or directories from the host system to the container	COPY . /app
RUN	Executes a command during the image build process. Typically used to install dependencies.	RUN apt-get update && apt-get install -y git
ENV	Sets environment variables inside the container	ENV FLASK_APP=app.py
CMD	Defines the default command to run in the container when it starts.	CMD ["python", "app.py"]
EXPOSE	Informs Docker that the container will listen on the specified network ports at runtime	EXPOSE 5000

Create a Simple Python Web App

- lets create a hello world python web app named app.py using flask.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def greet():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Creating a custom docker image - Dockerfile

we can use # for adding comments

```
# Use an official Python runtime as a base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages
RUN pip install --no-cache-dir Flask

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Building and Running Custom Image

innovate achieve lead

- Once we create the Dockerfile, we can build the image using the following command:

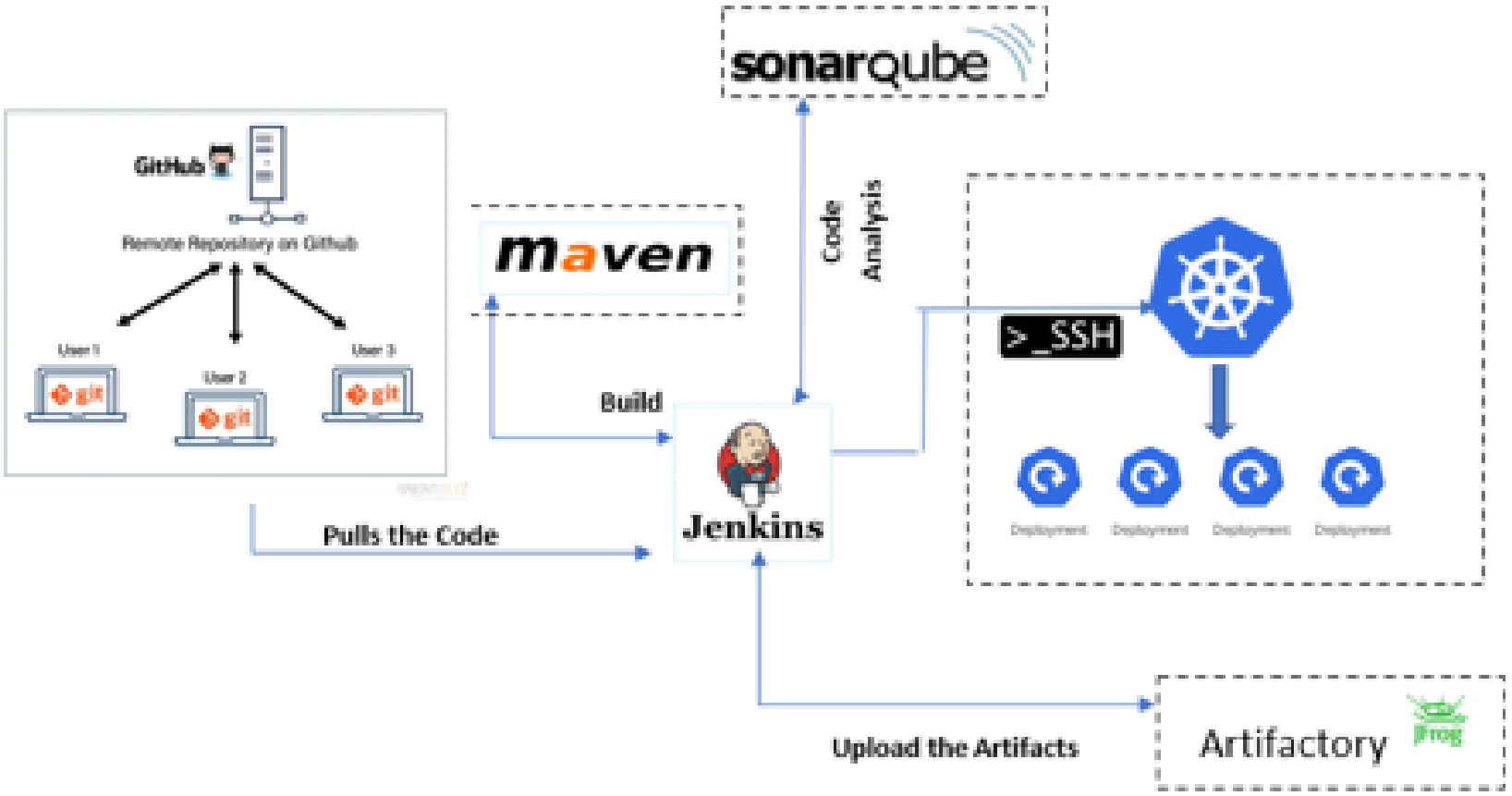
docker build -t <custom-image-name> .

- To run the container based on the newly created image:

docker run -d -p 8080:80 <custom-image-name>

- This will map port 8080 on your host machine to port 80 on the container, allowing us to access the Nginx web server via http://localhost:8080.

Traditional CI setup



Key Components of a Traditional CI Setup

innovate achieve lead

1) **Version Control System (VCS):**

- A VCS like SVN or Git is used to track code changes.
- Developers commit their code changes to a shared repository frequently
- Example: GitHub, GitLab, Bitbucket.

2) **CI Server**

- The CI server is the core of the setup responsible for detecting code changes, triggering automated builds, and running tests.
- Popular CI servers include Jenkins, Travis CI,TeamCity, Bamboo etc.

Key Components of a Traditional CI Setup

3) Build Scripts

- The CI server uses build scripts to compile the application, package the code, and run automated tests.
- The build process can be automated using tools like Maven , Gradle for Java projects , Ant/Make for C/C++ projects.

4) Automated Tests

- Automated testing is a critical part of CI.
- Tests can include unit tests, integration tests, and functional tests.
- These are run after every build to catch issues early.
- Testing frameworks can include JUnit for Java projects, pytest for Python Projects etc.

Key Components of a Traditional CI Setup

5) Artifact Repository

- After a successful build, artifacts such as binaries, packages, or libraries are stored in an artifact repository for further consumption by QA/Customer.
- Popular ones include Jfrog Artifactory

6) Notification System

- CI systems send notifications (via email, Slack, or other messaging tools) to notify developers of the build status—whether the build passed or failed.

Building a Java application

innovate achieve lead

- The following are the general steps we perform to package a java application:

1) Define dependencies

Ex: logger apache log4j - logger.info , logger.debug

2) Compile Java classes from .java to .class files

3) Manage resources that are need by java files

Ex: configuration – log4j2xml/json and properties files - .properties

4) Run Tests – junits

5) Create a jar containing all the above files

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 8

Recap

- Docker Containers
- Hands-on exercises using Docker Desktop and Docker Hub
- Containerizing a custom app
- Traditional CI Setup

Agenda for today's class

- Build Tools - Maven, Gradle
- Continuous code inspection - Code quality
- Code quality analysis tools – Sonarqube
- Managing Components & Dependencies
- Essential CI Practices
- Mid Sem Regular Question Paper format and Topics

Building a Java application

- The following are the general steps we perform to package a java application:

1) Define dependencies

Ex: logger apache log4j - logger.info , logger.debug

2) Compile Java classes from .java to .class files

3) Manage resources that are need by java files

Ex: configuration – log4j2xml/json and properties files - .properties

4) Run Tests – junits

5) Create a jar containing all the above files

Overview of Build Tools- Maven

- Maven is a build automation tool primarily for Java projects.
- Maven is known for its dependency management and build lifecycle management.
- It uses XML for configuration and focuses on convention over configuration.
- Meaning it assumes a standard project structure and configuration unless explicitly overridden.

maven java project convention

```
my-java-project/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── example/
│   │   │           └── App.java
│   │   └── resources/
│   │       └── application.properties
│   └── test/
│       ├── java/
│       │   └── com/
│       │       └── example/
│       │           └── AppTest.java
│       └── resources/
│           └── test-config.properties
├── target/
│   └── (compiled classes, JARs, etc.)
├── .gitignore
└── pom.xml
```

Key Features of Maven

1) Project Object Model (POM):

- Maven uses a pom.xml file to define project structure, dependencies, and plugins.
- The POM file is used to manage project configuration and dependencies.

2) Dependency Management:

- Maven manages project dependencies through the **dependency** element in the POM file.
- Dependencies are downloaded from central repositories (Maven Central) and are stored in the local Maven repository.

Key Features of Maven

3) Build Lifecycle:

- Maven follows a well-defined build lifecycle consisting of phases like compile, test, package, install, and deploy.
- Each phase has specific goals and plugins that are executed in sequence.

4) Plugins:

- Maven uses plugins to perform specific tasks during the build process (e.g., compiler plugin).
- Plugins are configured in the POM file.

- **What is a POM?**
- A Project Object Model or POM is the fundamental unit of work in Maven.
- It is an XML file that contains information about the project and configuration details used by Maven to build the project.

Example pom.xml Configuration


```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>my-java-project</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>

                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Maven Commands

1) **mvn clean:** Removes the target directory to start with a clean build.

2) **mvn compile:** Compiles the source code in the src/main/java directory.

3) **mvn test-compile:** Compiles the test source code in the src/test/java directory.

4) **mvn test:** Executes the unit tests.mvn package: Compiles code, runs tests, and packages the application into a JAR or WAR file.

5) **mvn install:** Packages the code and installs the artifact into the local Maven repository.

6) **mvn dependency:tree:** Displays the project's dependency tree.

7) **mvn clean install:** Cleans the project and then installs the artifact to the local repository.

Overview of Build Tools- Gradle

- Gradle is a versatile build automation tool used for building, testing, and deploying software projects.
- It is designed to be more powerful and flexible than earlier build tools like Maven and Ant while providing better support for complex project configurations and dependencies.
- It also follows convention over configuration like maven project.

Gradle steps for a Java project

innovate achieve lead

1) **Initialize Project:** Start by creating a new Java project using the gradle init command.

2) **add maven repo**

```
repositories {
    mavenCentral()
}
```

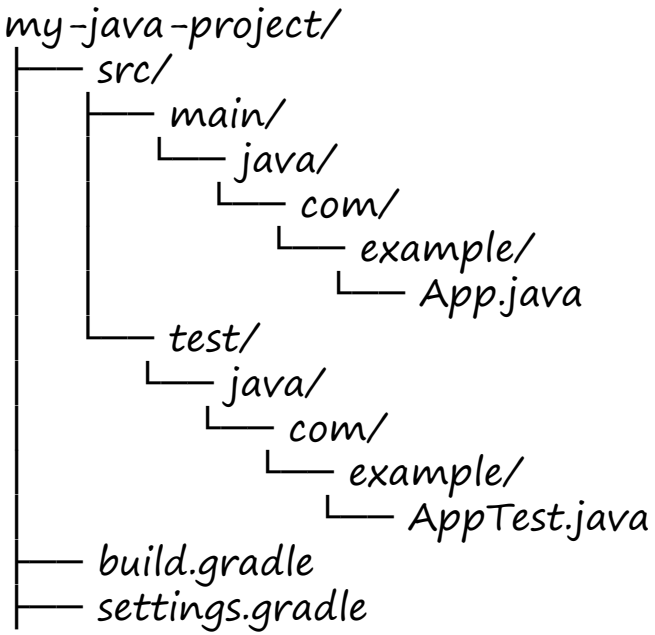
3) **add java plugin:** The Java plugin adds Java compilation along with testing and bundling capabilities to a project

```
plugins {
    java
}
```

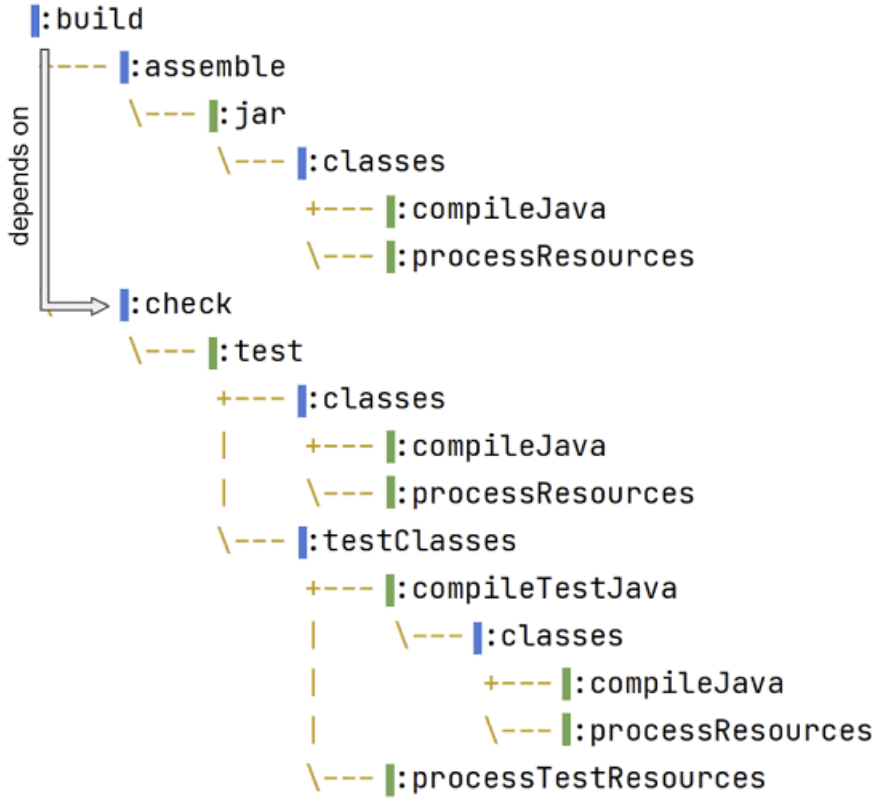
4) **build the project:** using ./gradlew build

Gradle tasks for a Java project


```
plugins {
    id 'java'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework:spring-
core:5.3.8'
    testImplementation 'junit:junit:4.13.2'
}

task hello {
    doLast {
        println 'Hello, Gradle!'
    }
}
```



Java plugin - tasks



= tasks which perform an action

= aggregate tasks

Java plugin – tasks

1) **compileJava:**
- Compiles the source code located in the src/main/java directory.
- Output is stored in the build/classes/java/main directory.

2) **processResources:**
- Copies non-Java resources (e.g., properties files) from src/main/resources to build/resources/main.

3) **classes:**
- Aggregates compileJava and processResources tasks to produce the compiled classes and resources.

4) **jar:**
- Packages the compiled code and resources into a JAR file.
- Output JAR is located in the build/libs directory.

Java plugin – tasks

5) compileTestJava:

- Compiles the test source code in src/test/java. Output is stored in build/classes/java/test.

6) processTestResources:

- Copies test resources from src/test/resources to build/resources/test.

7) testClasses:

- Aggregates compileTestJava and processTestResources tasks to prepare the test classes.

8) test:

- Runs the unit tests using the compiled test classes.
- Generates test reports in the build/reports/tests directory.

Java plugin – tasks

9) clean:

Deletes the build directory, removing all generated files.

10) build:

- Aggregates the tasks for compiling, testing, and packaging the project into a JAR.
- Runs the following sequence:
  - clean
  - compileJava
  - processResources
  - test
  - jar

SonarQube

- SonarQube is a powerful tool for continuous inspection of code quality.
- It automatically reviews your code for bugs, vulnerabilities, and code smells, and it integrates seamlessly into your development pipeline.

SonarQube – Terminology

1) Projects:

- SonarQube organizes code analysis into projects.
- Each project corresponds to a codebase that you want to analyze.

2) Issues:

- These are the problems SonarQube detects in your code, such as bugs, code smells, and vulnerabilities.

3) Quality Gates:

- Quality Gates define the criteria that your code must meet to pass the quality check.
- For example, a Quality Gate might require that there be no critical issues in the code.

SonarQube Setup on laptop

1) Step 1: Download and Install SonarQube:
   Visit SonarQube's official website and download the appropriate version for your operating system.

2) Extract the downloaded ZIP file to a directory of your choice.

3) Start SonarQube:
   1) For linux/mac:
      cd <path_to_sonarqube>/bin/macosx-universal-64./sonar.sh start
   2) For Windows:Go to <path_to_sonarqube>\bin\windows-x86-64\Double-click StartSonar.bat.

4) Access SonarQube Dashboard:
   - open your web browser and navigate to http://localhost:9000.
   - The default credentials are:
     - Username: admin
     - Password: admin
   - You can change the admin password after logging in for the first time.

SonarQube Example

```
public class App {
    public void example() {
        //1) Code smell: unused variable
        String unusedVariable = "I am not used anywhere";
        //2) Bug: Division by zero
        int divideByZero = 10 / 0;
        //3) Code smell: Always true condition
        if (true) {
                System.out.println("This will always print.");
        }
    }
}
```

Gradle vs SonarQube

- **Gradle** is a build automation tool used to compile, package and manage dependencies for your code.
- **SonarQube** is a code quality and security analysis tool that helps you identify bugs, vulnerabilities, code smells, and other quality issues in your codebase.
- Even though **Gradle** builds the project successfully, it doesn't necessarily mean the code is clean or secure.
- **SonarQube** analyzes your code for potential quality issues, helping you improve maintainability, reduce technical debt, and detect security flaws early.

Integrate SonarQube with Gradle

- SonarQube acts as a server that stores and analyzes the code quality reports.
- Gradle communicates with this server to upload the results of the analysis.

1) Add the following code to your build.gradle:

```
plugins {
    id "org.sonarqube" version "4.3.0.3225"
}
sonar {
    properties {
        property "sonar.projectKey", "your-project-key"
        property "sonar.host.url", "http://localhost:9000"
        property "sonar.token", "your-sonarqube-token"
    }
}
```

Integrate SonarQube with Gradle

innovate achieve lead

2) Configure Project in SonarQube:

- In the SonarQube dashboard, create a new project. Choose a project key (which is added in build.gradle) and generate a token for authentication.

- Add the token in build.gradle under the sonar.login property.

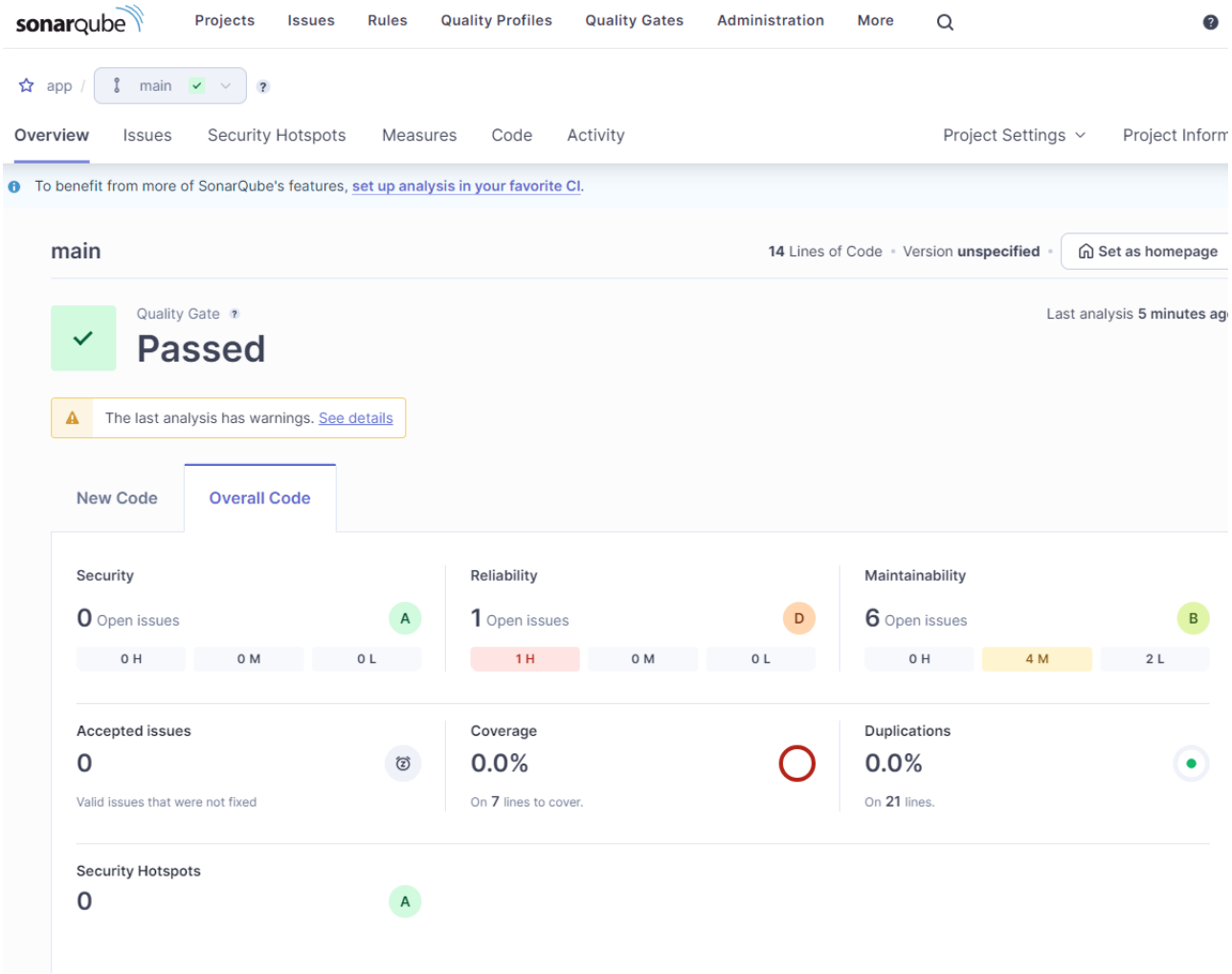
3) Run SonarQube Analysis with Gradle:

- Use the following command to trigger SonarQube analysis on your project:

```
./gradlew sonarqube
```

4) Analyze Results on SonarQube Dashboard

Integrate SonarQube with Gradle



Essential CI Practices - 1

1) **Automated Build:** Automatically compile and build code changes with each commit.

2) **Automated Testing:** Run unit, integration, and end-to-end tests automatically to validate changes.

3) **Code Quality Checks:** Use tools to perform static code analysis and linting to ensure code quality.

4) **Continuous Feedback:** Provide immediate feedback on build and test results to developers.

5) **Frequent Commits:** Encourage regular, small commits to catch issues early and integrate smoothly.

Essential CI Practices - 2

6) **Build Artifacts Management:** Store and version build artifacts for consistent deployments and rollbacks.

7) **Fail Fast and Fail Early:** Detect and address issues promptly by failing builds early when problems are found.

8) **Secure CI Pipeline:** Protect the CI system and codebase with access controls and secure handling of credentials.

9) **Documentation and Reporting:** Maintain clear documentation and generate reports on build status, test results, and code quality.

Mid Sem Regular Question Paper format and Topics

innovate achieve lead

- There will be 5 questions with each question carrying 6 marks for a total of 5*6 = 30.

Q1) SDLC , Process Models and Devops Practices

Q2) Version Control System

Q3) Virtualization and Containers

Q4) Cloud Native Applications

Q5) Continuous Integration

THANK YOU!

DEVOPS FOR CLOUD (CC ZG507)

BITS Pilani
Pilani Campus

Sai Sushanth Durvasula
Guest Faculty

innovate achieve lead

BITS Pilani

Pilani Campus

Contact Session - 9

Recap

- Build Tools - Maven, Gradle
- Continuous code inspection - Code quality
- Code quality analysis tools – Sonarqube
- Managing Components & Dependencies
- Essential CI Practices
- Mid Sem

Agenda for today's class

innovate achieve lead

- Custom Image and Container creation
- Container Orchestration
  - What is it?
  - Why do we need it?
  - History of Container Orchestration Tools
- Kubernetes Cluster Architecture
- Minikube
- Working with Kubernetes Objects
  - pods,
  - replicasets,
  - deployment
- Hands-on exercises using Minikube / cloud platforms

Creating a custom docker image

- Dockerfile specifies all the steps defining what all are needed to be done on the container once created.

- To create a Dockerfile use below command

  touch Dockerfile

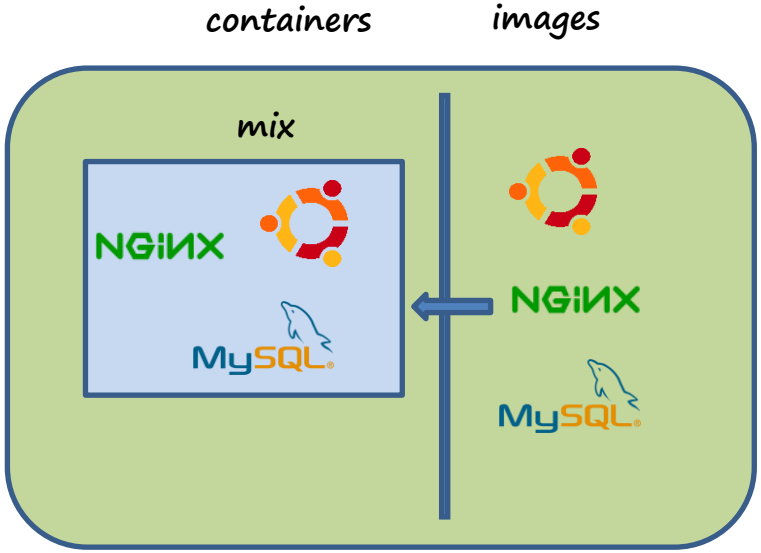
- Format of lines in Dockerfile:

  <instruction> <argument>

  Ex: FROM Ubuntu

- Here instruction is FROM and argument is Ubuntu

docker hub



docker host

Creating a custom docker image - Dockerfile

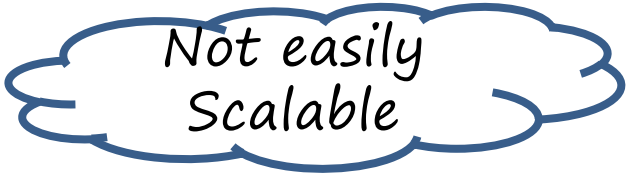
we can use # for adding comments

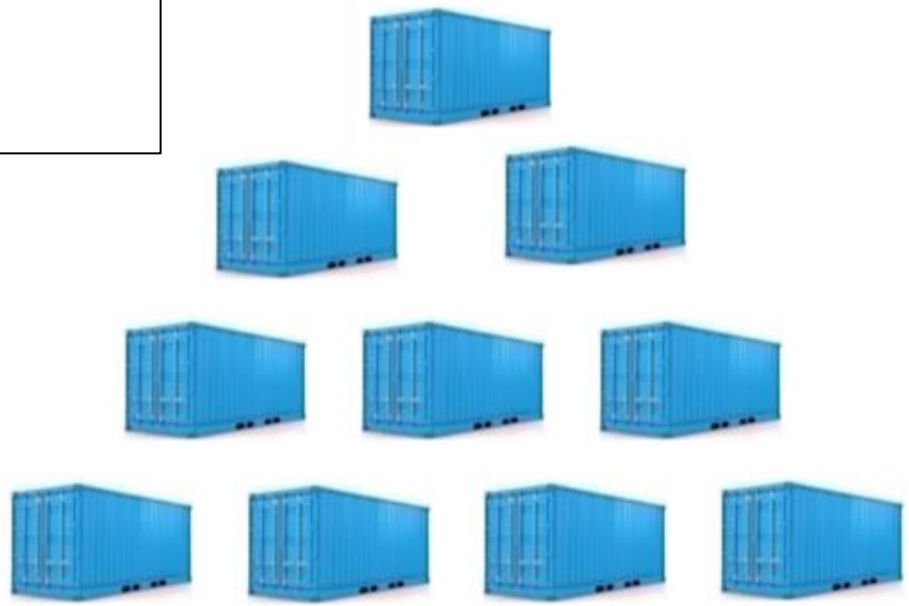
```
# Use an official Python runtime as a base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages
RUN pip install --no-cache-dir Flask

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py

# Run app.py when the container launches
CMD ["python", "app.py"]
```

The need for Orchestration

Both Linux Containers and Docker Containers

- Isolate the application from the host





Why Do We Need Container Orchestration?

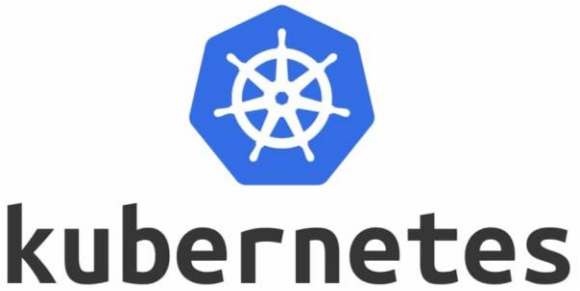
- Imagine you're running a website.
- Initially, you have one server, and everything works fine.
- As your website grows, you add more servers, and instead of running everything on the server directly, you start using containers (like Docker) to package your apps and their dependencies.

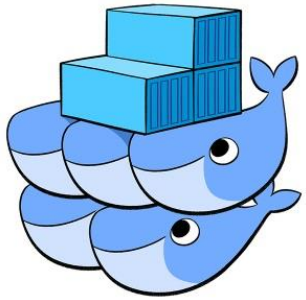
- **Challenges** arise when you manage multiple containers:
  - **Scaling**:
    - What if a lot of users are using your app?
    - You need to spin up more containers.
  - **Resilience:**
    - What if a container crashes?
    - You need a system to restart it.
  - **Distribution:**
    - You may want to run containers across multiple servers.
  - **Management:**
    - Keeping track of all your running containers, scaling, updating them, etc.

Container Orchestration

Container orchestration automates the deployment, management, scaling, and networking of containers.

Container Orchestration Tools





History of Container Orchestration Tools – 1

- As containerized apps became more common, orchestration tools evolved to help manage them:

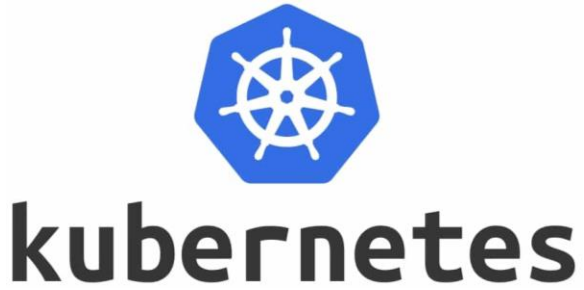
- **2013 – Docker Containers**
- Docker – Containers were popularized with Docker, but Docker alone could not handle complex systems
- Ex: scaling or restarting crashed containers.

- **2014 – Mesos and Marathon**
- Mesos was used by companies like Twitter for resource management, and
- Marathon helped manage containers on Mesos.

- **2015: Docker Swarm**
- Docker's native orchestration tool aimed at solving container management but lacked advanced features.

History of Container Orchestration Tools – 2

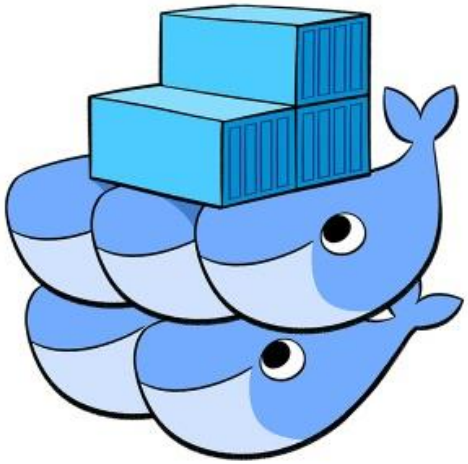
- **2014-2015: Kubernetes (K8s)**
- Developed by Google, based on its internal system (Borg), it became the most popular solution for managing containers.
- It's open-source and supports
  - advanced scheduling
  - scaling
  - self-healing and more.



- **2018:**
- Kubernetes became the de-facto industry standard, overshadowing most alternatives.

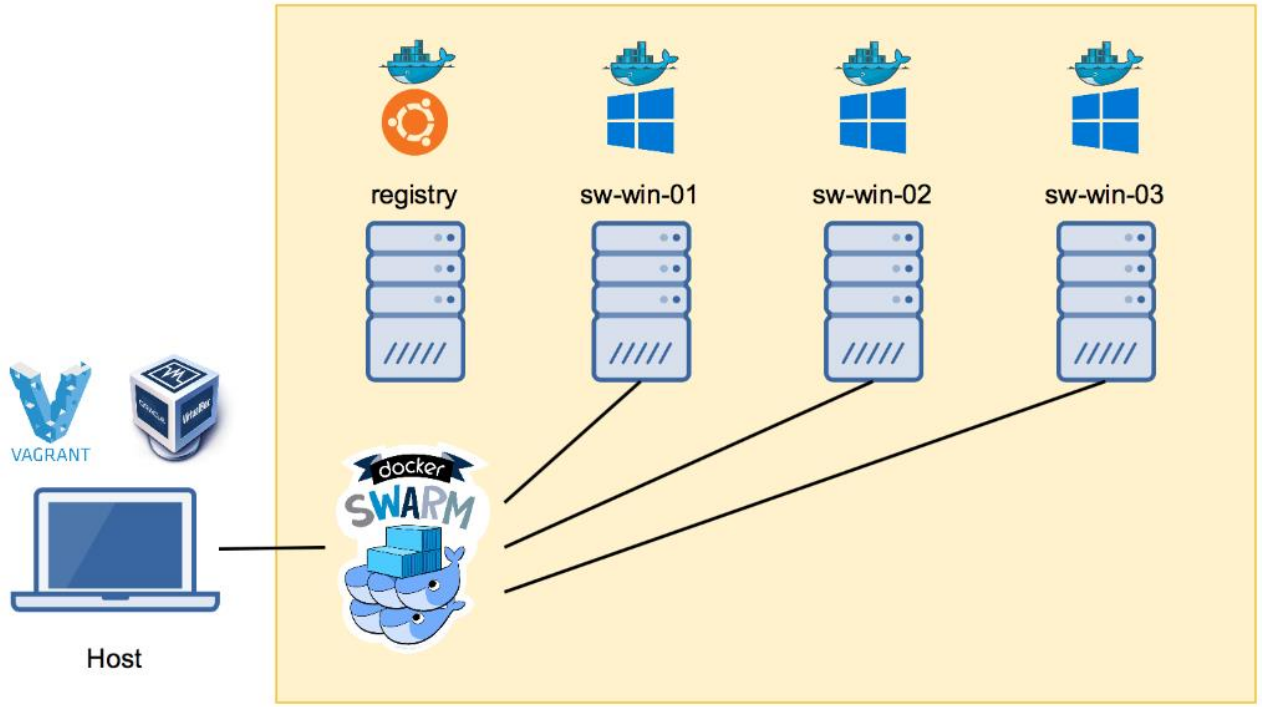
Docker Swarm

- Docker Swarm is an orchestration management tool that runs on Docker applications.
- While Docker is great for running containers on a single machine, Swarm turns several Docker hosts into a cluster, coordinating their efforts.
- Docker Swarm allows you to manage multiple containers on multiple hosts as a single system.
- Each node of a Docker Swarm is a Docker daemon, and all Docker daemons interact using the Docker API.



Docker Swarm

- Each container within the Swarm can be deployed and accessed by nodes of the same cluster.



Kubernetes

- It is an open-source container orchestration tool that was originally developed and designed by engineers at Google.

- Kubernetes orchestration allows you to build application services that span multiple containers, schedule containers across a cluster, scale those containers, and manage their health over time.

Kubernetes Cluster Architecture – Details

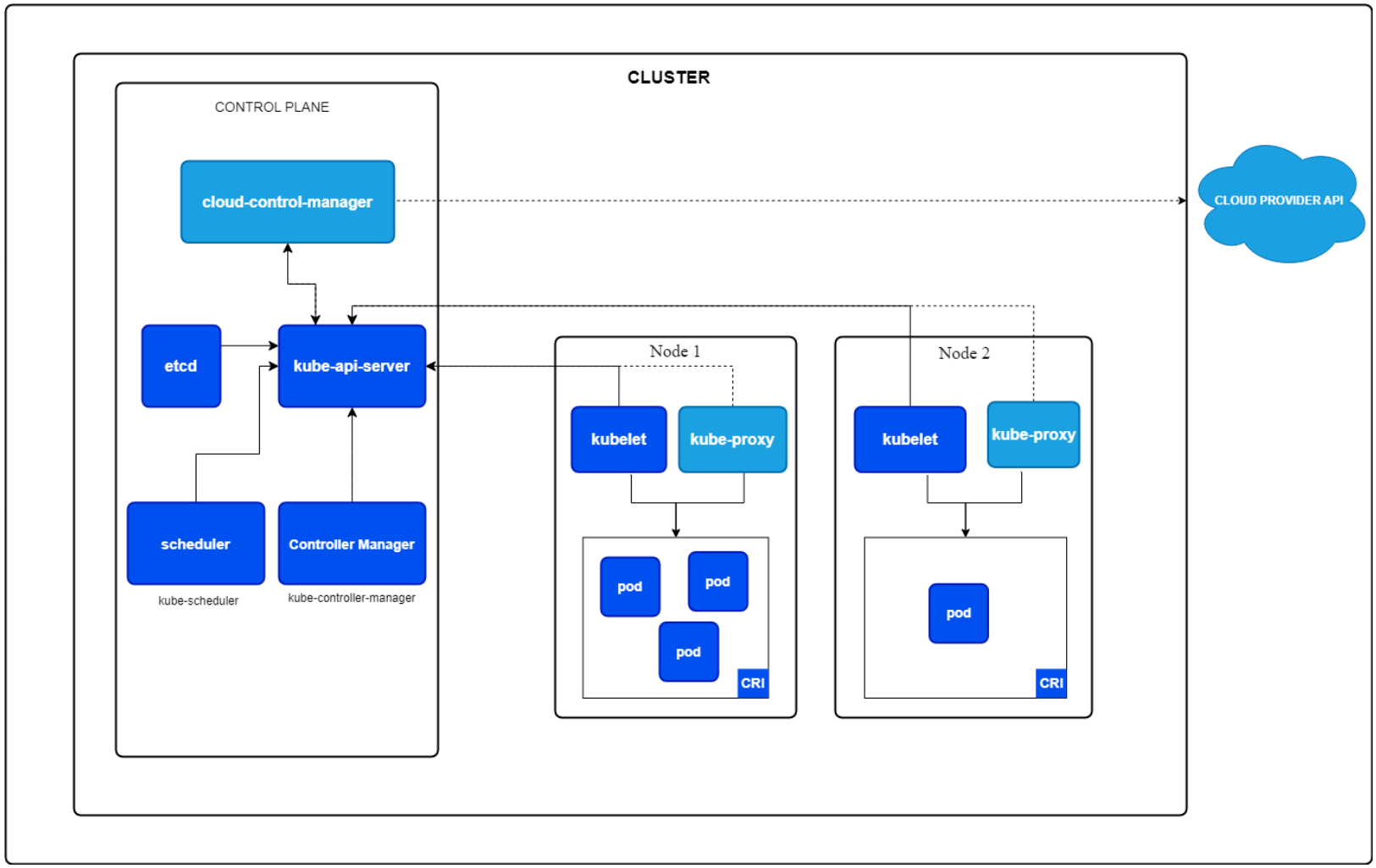
innovate achieve lead

- Kubernetes uses a **master-worker** architecture:

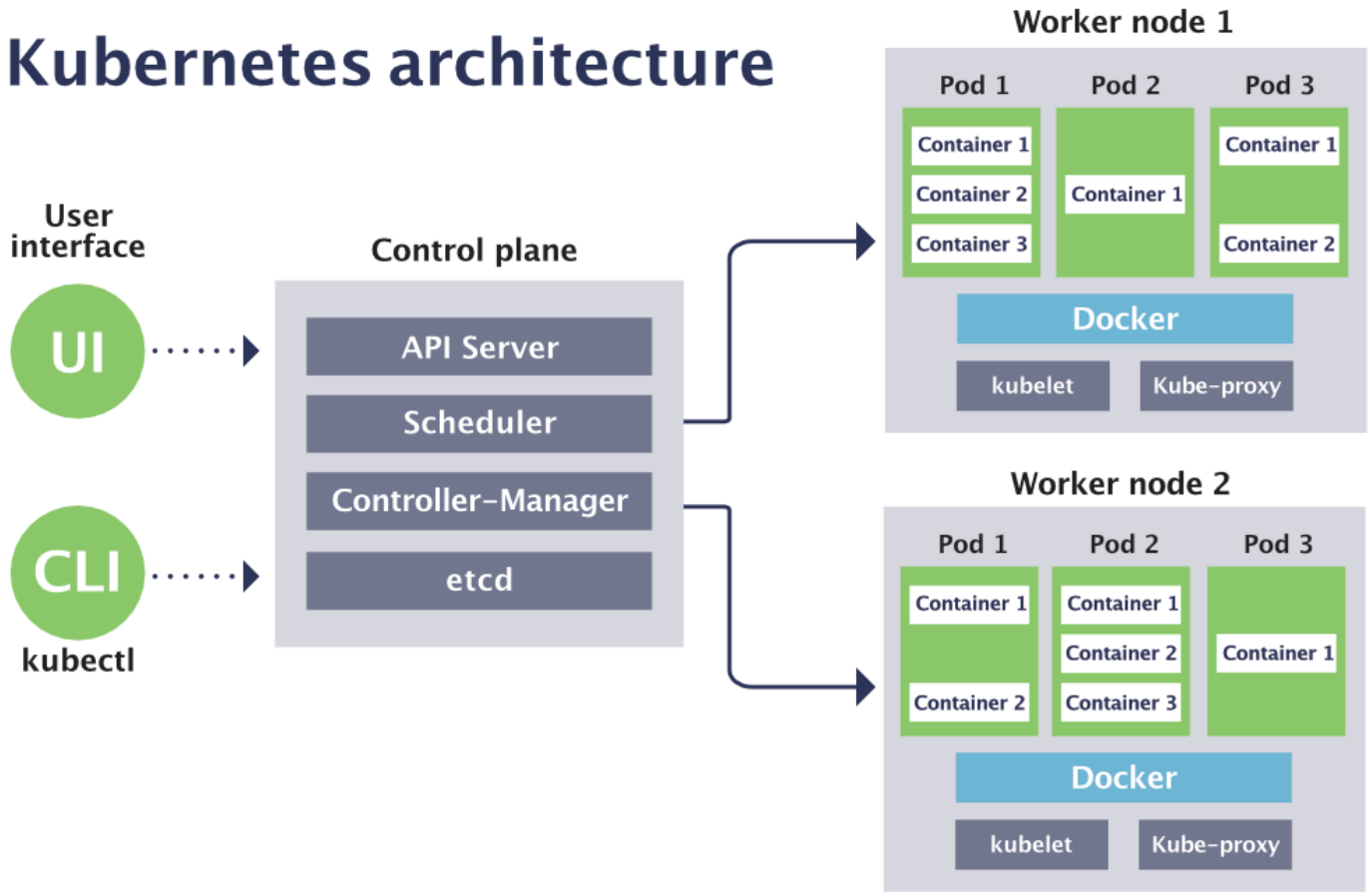
- **Master Node**: The control plane responsible for managing the cluster.
  - **API Server**: Interacts with Kubernetes users. Validates requests, authenticates users.
  - **Controller Manager**: Ensures that the desired state of the system is maintained.
  - **Scheduler**: Assigns workloads to worker nodes.
  - **etcd**: A key-value store for cluster data.

- **Worker Nodes**: The machines that run your applications.
  - **CRI**: Container run time interface – **docker**
  - **Kubelet**: Manages containers on the node by talking to master.
  - **Kube Proxy**: Handles network traffic between pods and expose pods (your applications) to end users.

Kubernetes Cluster Architecture



Kubernetes Cluster Architecture



Minikube Setup on Windows - 1

innovate achieve lead

- Minikube is a tool that lets you run Kubernetes locally, perfect for learning and testing.

Step 1: Install Virtualization Software:

- We need a hypervisor like Hyper-V or Oracle VirtualBox.
- Minikube requires virtualization to run the Kubernetes cluster locally
- To check if virtualization is enabled, you can go to the Task Manager > Performance tab > CPU. There should be a section for Virtualization: Enabled.
- To enable Hyper-V, open PowerShell (Admin) and run the following command and restart the computer after that:
  - Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V –All
- Verify the installation using VBoxManage --version

Step 2: Install kubectl (Kubernetes Command-line Tool)

- Download kubectl by following steps at https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/
- Add kubectl.exe to your system path and verify using:
  - minikube version

- We now have a single-node Kubernetes cluster running locally!

Minikube Setup on Windows - 2

innovate achieve lead

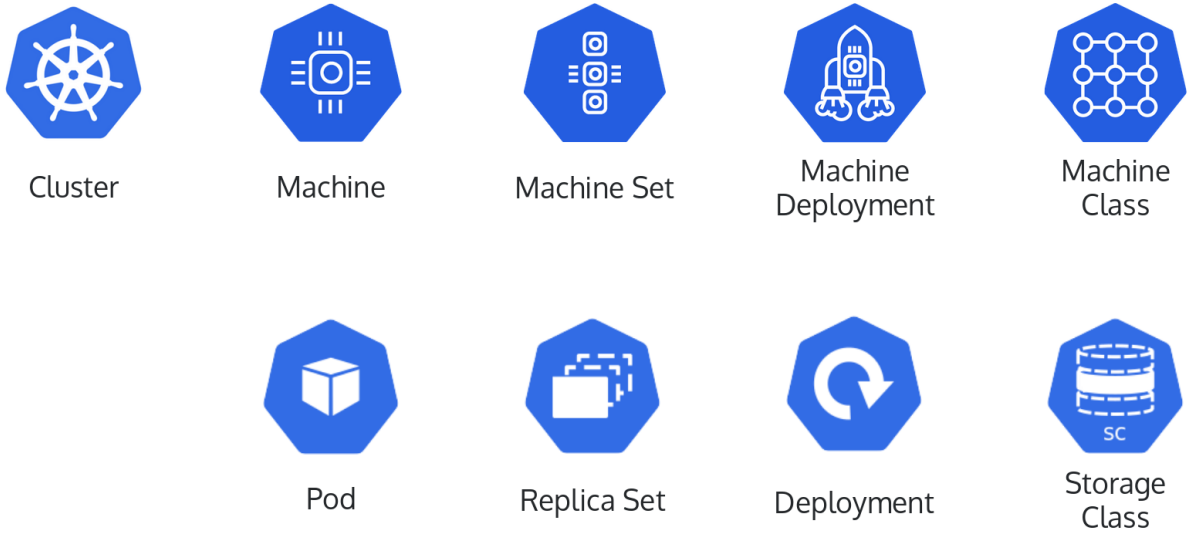
Step3: Start Minikube

- Open PowerShell as Administrator.
- To start a Minikube cluster, run the following command:
  - minikube start --driver=hyperv or
  - minikube start --driver=virtualbox
- Minikube will download the necessary Kubernetes components and start the cluster.
- Verify Minikube is running by checking the status:
  - minikube status

- Once Minikube starts, verify your cluster is running by checking the node status:
  - kubectl get nodes

Kubernetes Objects

1. Pod
2. ReplicaSet
3. Deployment
4. Service
5. Job
6. ConfigMap
7. Secret
8. Namespace
9. Volume etc



kubectl commands - 1

innovate achieve lead

1) client and server versions
   kubectl version

- lists the versions of client and server

kubectl version --client for getting version of client
kubectl version --server for getting version of server

2) help
   kubectl help
- to get commands help

kubectl commands - 2

3) nodes

to get the list of nodes running

```
kubectl get nodes
```

to get list of multiple components

```
kubectl get pods,svc
```

for more verbose info use

```
kubectl get nodes -o wide
```

kubectl commands - 3

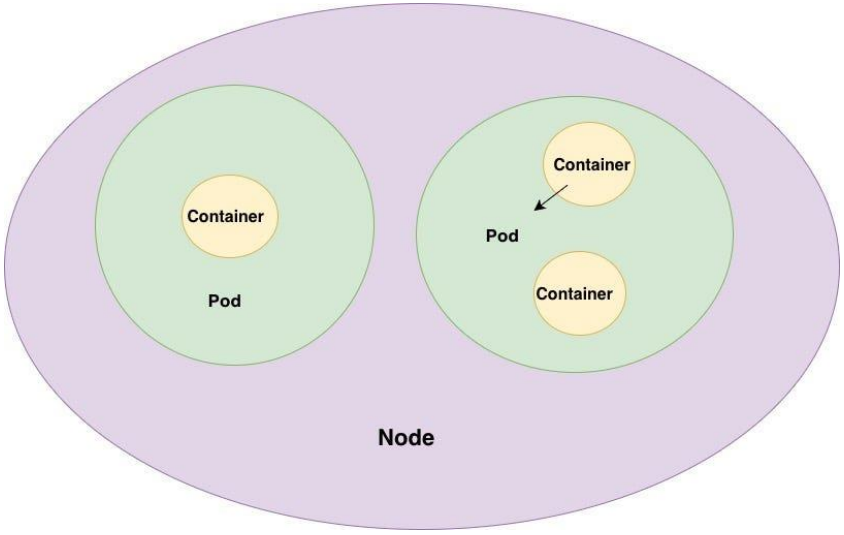
innovate achieve lead

4) yaml file configuration

```
# to create yaml file for given config
  kubectl run nginx --image=nginx --dry-run=client
   -o yaml > nginx-pod.yaml

# to create many at once from many config files
  kubectl create -f pod1-def.yaml,pod2-def.yaml

# to delete many at once
  kubectl delete pod1,pod2
```

What is a Pod?

- A **Pod** is the smallest deployable unit in Kubernetes.
- A pod represents a single instance of an application.
- A pod can contain one or more containers.
- A pod is defined in a YAML file.



Pod definition

apiVersion: v1 #(version of kubernetes api to be used to create this object)

kind: Pod (type of object being created)

metadata: (key value pairs)
    name: myapp-pod
    labels: (any key value pairs)
        app: myapp
spec: (specifies whats inside the component)

    containers:
    - name: nginx-container
      image: nginx

api versions for various objects

kind	version
Pod	v1
ReplicaSet	apps/v1
ConfigMap	v1
Job	v1
Deployment	apps/v1
Service	v1

Pod definition - example

Let us create a pod named **nginx-pod** running a container named **nginx-container** based on **nginx** image.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
        app: myapp
        tier: frontend
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```

Running a Pod

1) directly using image

syntax: kubectl run <pod_name>

runs a pod by downloading the nginx image from docker repo

Ex: kubectl run redis-pod --image=redis

2) using a pod-def yaml file

syntax: kubectl create/apply-f <pod-defn_yaml_file>

kubectl create -f pod-def.yml (first time)

kubectl apply -f pod-def.yml (after updates/changes)

ReplicaSet

- Pods are not self-healing, meaning if one crashes, it won't restart on its own.
- **ReplicaSet** ensures that a specified number of identical pods are always running.
- If a pod fails, replicaSet automatically replaces it.
- **ReplicaSet** is used for High Availability of the applications.
- Monitors the pods based on the selector

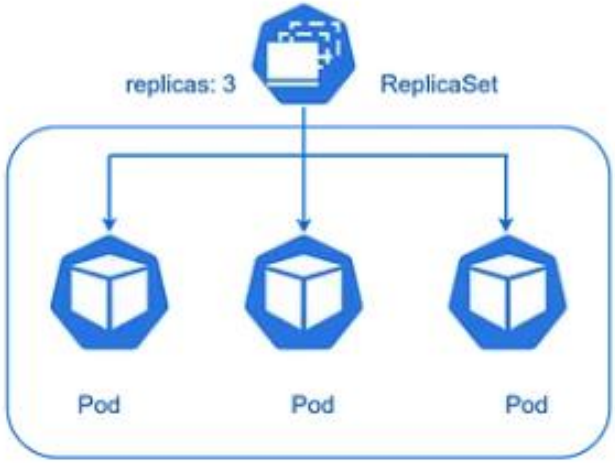


Fig: ReplicaSet

ReplicaSet definition


```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaSet
spec: (contains template of pod to be replicated)
  template:
          <your_pod-def>
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
```

Notes:

======

1) the selector matchLabels is used to match the pods running in kubernetes cluster having labels as tier: frontend and this replicaset will be applicable to all of them

2) the pod being defined inside the replicaset needs to always match wrt label selector!

Running a ReplicaSet

1) create replicaSet using the config yml file

syntax: kubectl create/apply -f repset-defn.yml

2) to get the list of replicaset running

kubectl get replicaset or kubectl get rs

3) to delete a replicaset

kubectl delete replicaset <replicaset_name>

4) to update a replicaset

kubectl edit replicaset <replicaset_name>

kubectl replace -f repset-defn.yml

Running a ReplicaSet

innovate achieve lead

5) to scale a replicaset to change replicas

kubectl scale -replicas=6 -f repset-defn.yml

6) to get more details about a replicaset

kubectl describe replicaset <replicaset_name>

kubectl explain replicaset

THANK YOU!