# DEVOPS FOR CLOUD
# (CC ZG507)

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

# Contact Session - 1

innovate achieve lead

## 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



innovate achieve lead

## 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

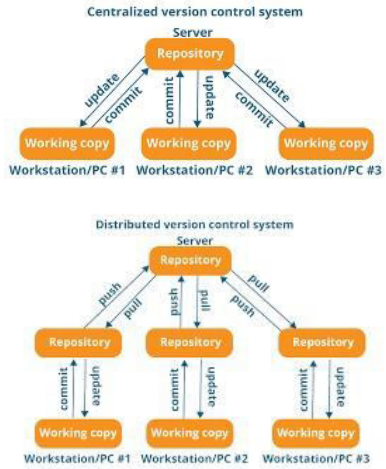
# Course Overview

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



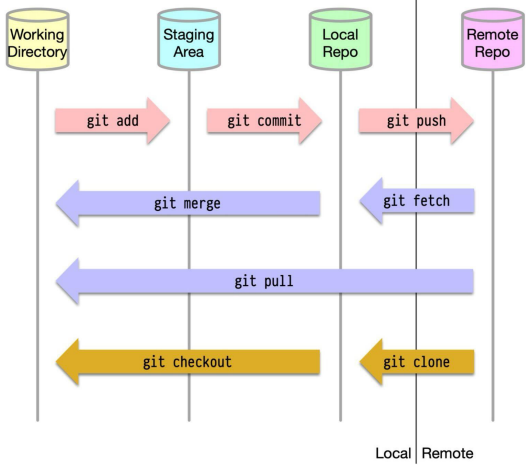
# CVCS vs DVCS

- **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.



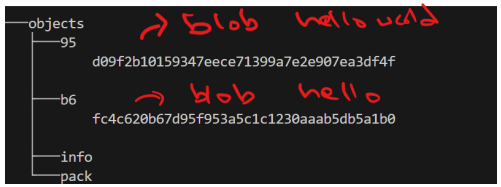


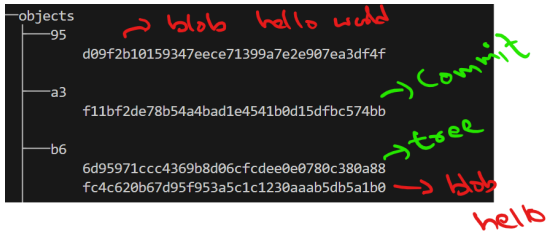

# Git Lifecycle Visualized






# Git State tracked after a commit



# DataOps Principles

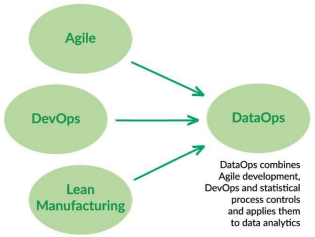
innovate achieve lead

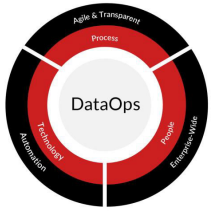
**1) Continually satisfy your customer**
*Our highest priority is to satisfy the customer through the early and continuous delivery of valuable analytic insights*

**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*

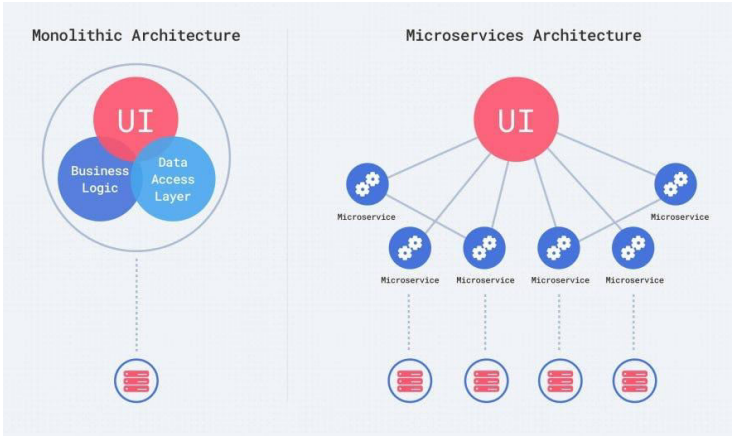





# Monolithic vs Microservices

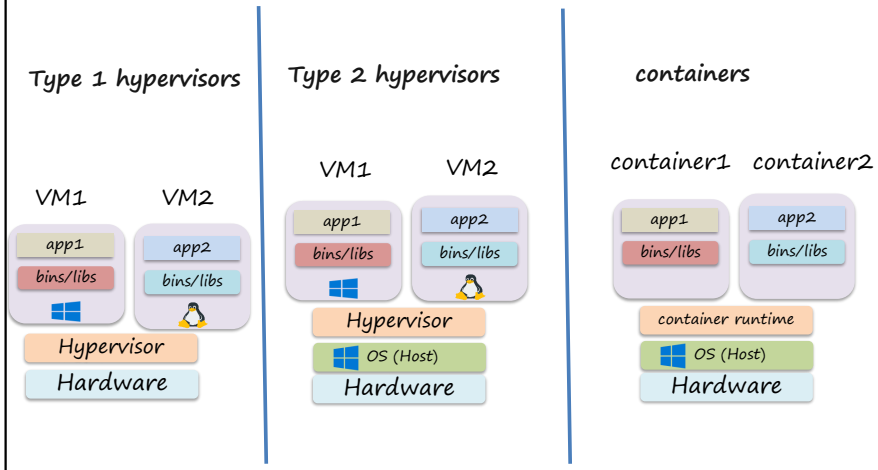
innovate achieve lead






# Hypervisors vs Containers Visualized

innovate achieve lead

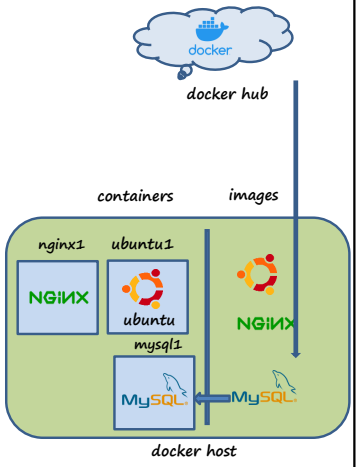



# creating our third container – mysql db

innovate achieve lead

- 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
1b5f483783c7: Pull complete
878e53a613d8: Pull complete
2a36204e79f: Pull complete
6e4d4f473cfe: Pull complete
692634634755: Pull complete
fe5e855492b2: Pull complete
5c02229ce6f1: Pull complete
7320aa32bf42: Pull complete
Digest: sha256:2d02ba6960cdf254de13f57d7265570c0675bc8bae1851e4a9806cff863006c9
Status: Downloaded newer image for mysql:latest
0c142754304f023c8ecc0ae641f015811c0be237e2c6f7bd7977f3b0fca5fd82a

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
```



# 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.

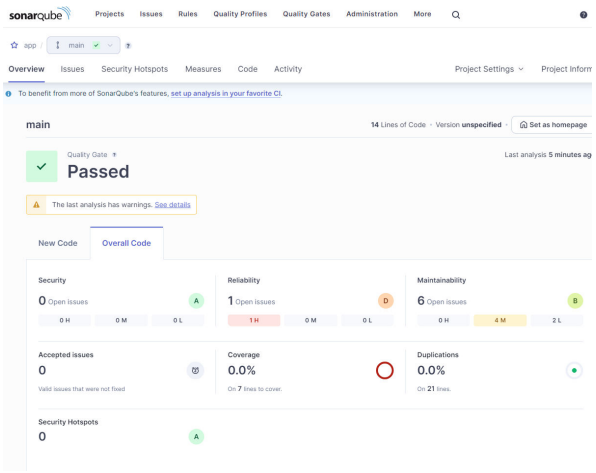
Maven

**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
```



# Integrate SonarQube with Gradle






# break time!!

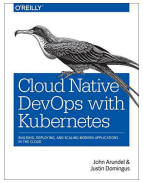
break from
2:25 PM - 2:35PM


any questions?

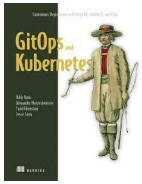


# Textbooks

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.

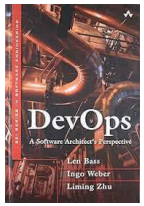



T2: GitOps and Kubernetes – Continuous Deployment with ArgoCD, Jenkins X and Flux'', by Billy Yuen et al. Publisher: Manning, 2021

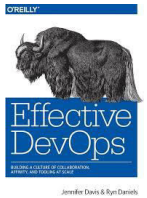


# Reference books

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

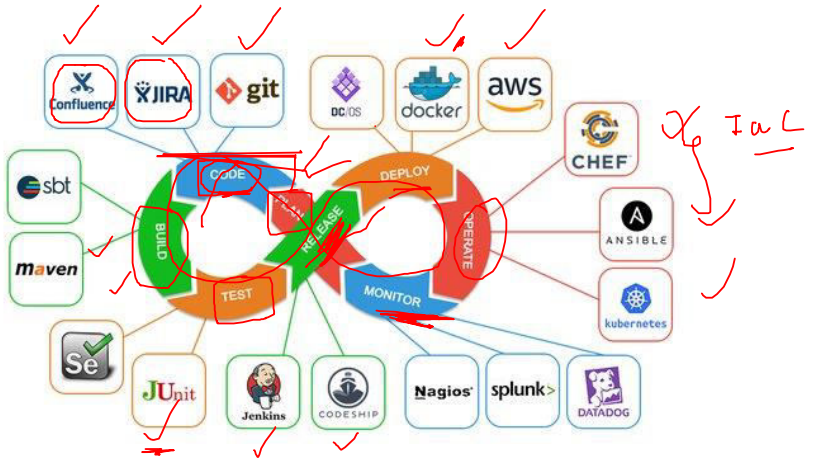



R2: Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale" by Jennifer Davis , Ryn Daniels. Publisher: O'Reilly Media, June 2016





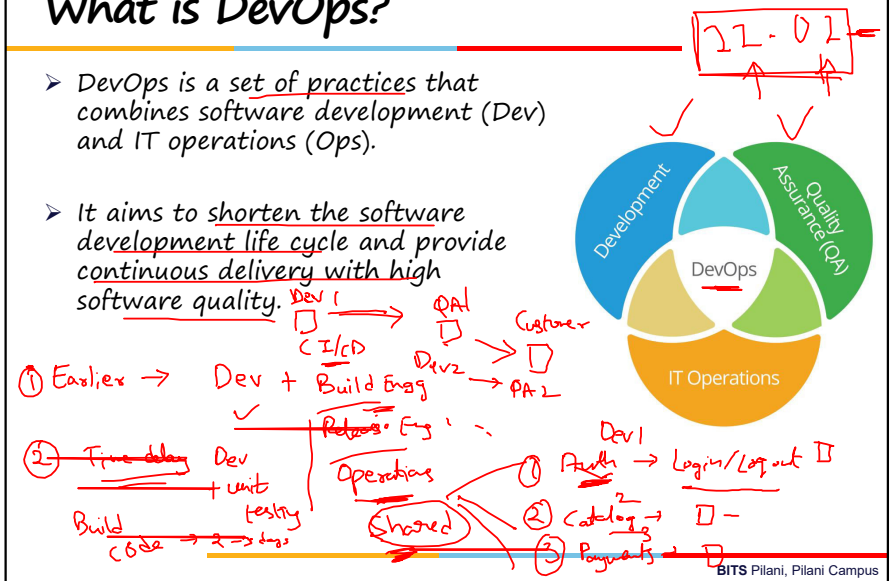

# What is DevOps?






# 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

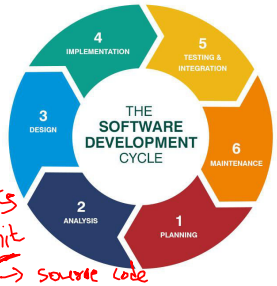
innovate achieve lead

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!




VCS
Git
.java → source code
built/packaged
maven, gradle → .jar binary



# 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.

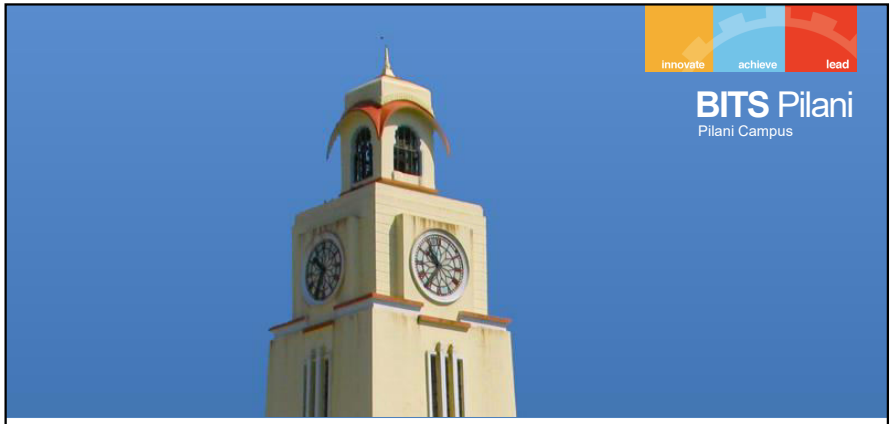
6) **Maintenance:**
- Provide ongoing support and make necessary updates.
- Responding to failures
- Monitoring




innovate achieve lead

# THANK YOU!






# Contact Session - 2

## Recap

- Foundational Concepts
  - Intro to Devops
  - Software Development Life Cycle
  - Need for DevOps
  - What is DevOps



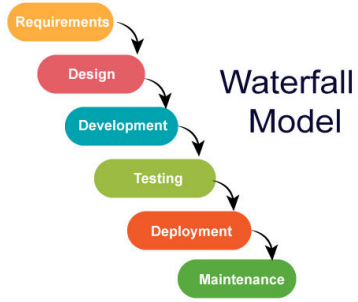
## Agenda for today's class

- Process Models (before Agile)
  - Waterfall Model
  - V Model
  - Iterative Model
  - Spiral Model
- 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
- Principles of Software Delivery
- Version control system and its types
- Introduction to GIT



## 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

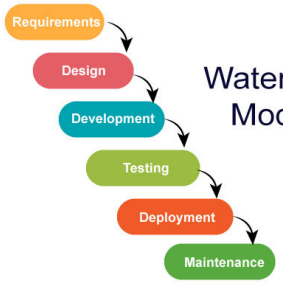
**5) Maintenance Phase:**

- 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?

innovate achieve lead

- 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.

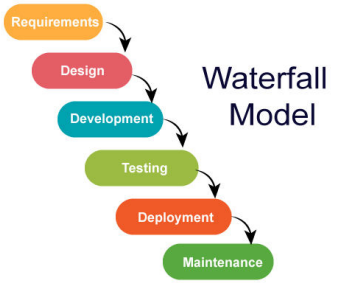





# Disadvantages of Waterfall Model

innovate achieve lead

- 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.

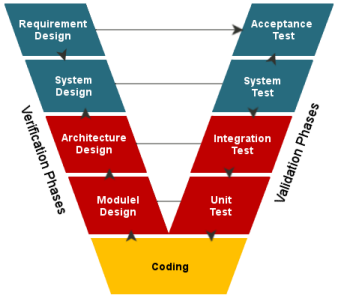



BITS Pilani, Pilani Campus

# Process Models– V-Model

innovate achieve lead

- V Stands for **Validation and Verification Model** and emphasizes verification and validation at each stage.
- An extension of the Waterfall Model introduced in 1980 and 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.




BITS Pilani, Pilani Campus

# Iterative Model

innovate achieve lead

- 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.

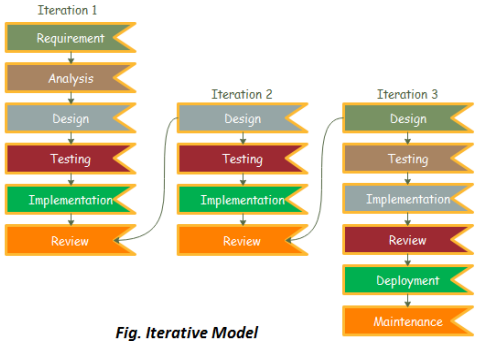
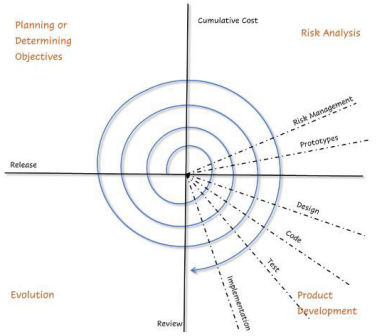



***Fig. Iterative Model***

BITS Pilani, Pilani Campus

# Spiral Model

- Introduced in 1986 by Barry Boehm.
- 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.






# 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.

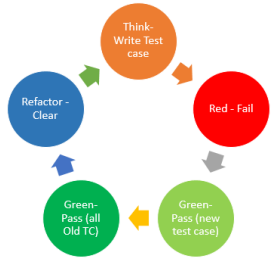
# TDD in Agile Context

- 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_add1():
    assert add(2, 3) == 5
def test_add2():
    assert add(2, -3) == -1
```

**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:**

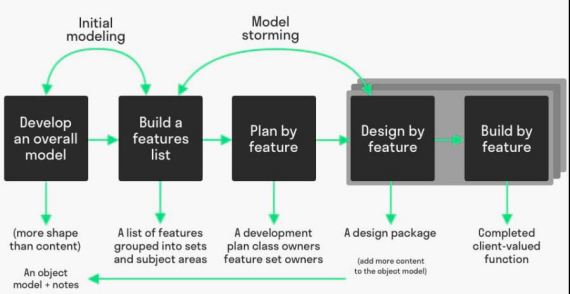


# FDD in Agile Context

- 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
}
```



# Agile Methodologies – Scrum

- 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.



# THANK YOU!

# DEVOPS FOR CLOUD
# (CC ZG507)

BITS Pilani
Pilani Campus

BITS Pilani
Pilani Campus

## Contact Session - 3

## Recap

- Process Models
  - Waterfall Model
  - V Model
  - Iterative Model
  - Spiral Model
- DevOps Lifecycle
- Agile Methodology for DevOps
  - Scrum, Extreme Programming
  - TDD, FDD and BDD in Agile context



## Agenda for today's class

- Real World examples of process models
- Three Dimensions of DevOps
- Key DevOps Practices
- Principles of Software Delivery
- CI - steps
- Version control system and its types
- Introduction to GIT
- GIT Basics commands
- Git Lifecycle Visualized
- How does Git keep track of files and changes?
  - Blob
  - Tree
  - Commit

# Class Guidelines

- There will be a break given in between for 15 minutes so we shall have two parts to the lecture:

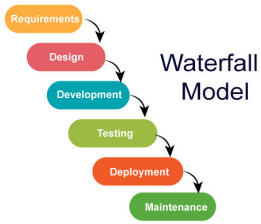
| Time slot | Session | Duration |
|---|---|---|
| 7:05 PM – 7:55 PM | Part 1 | 50mins |
| 7:55 PM – 8:10 PM | Break | 15mins |
| 8:10 PM – 9:00 PM | Part 2 | 50mins |

- 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



# Process Models - Waterfall Model - IBM

- IBM used the Waterfall Model for developing mainframe operating systems (e.g., OS/360 in the 1960s).




- Pros:

1) Clearly defined requirements ensured a structured development process
2) Extensive documentation helped maintainability for long-term projects

- Cons:

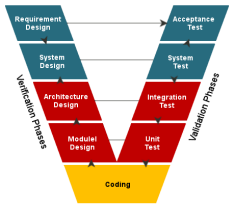
1) **Inflexible to hardware changes:** IBM's OS/360 project suffered when hardware advancements required software changes, but the rigid structure made adaptation difficult.
2) **Late-stage testing exposed design flaws:** Since testing was done only after full implementation, major design flaws were found too late, leading to cost overruns.

- Fred Brooks, the project lead, wrote about these challenges in his book "The Mythical Man-Month", showing why Waterfall doesn't work well for complex projects.



# Process Models– V-Model - Intel

- Intel follows the V-Model for developing firmware and microprocessor drivers.
- Since hardware and software must work together perfectly, Intel emphasized verification and validation at every stage to ensure reliability and performance.




- **Pros:**

1) Strict validation and verification made it ideal for hardware-software integration.
2) Caught defects early due to parallel testing at every stage.
3) Ensured high-quality output, essential for processor and chipset software.

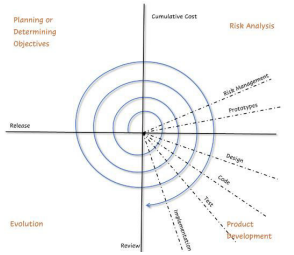
- **Cons:**

1) **Missed real-world validation step:** Intel's Pentium Floating-Point Bug caused incorrect division calculations but was only discovered after release by a professor, not during internal testing.
2) V-Model focused on predefined test cases but didn't consider unpredictable real-world scenarios, allowing the bug to slip through.
3) **Rigid structure made fixing the issue difficult:** The bug required a massive product recall, costing Intel $475 million and damaging its reputation.



# Spiral Model - Microsoft

- Microsoft used the Spiral Model for Windows Vista, but the project faced severe setbacks..




- Pros:

1) Allows flexibility and iterative improvements based on feedback.
2) Risk assessment in each cycle reduces major failures in theory.

- Cons:

1) **Feature creep from too many iterations:** Microsoft kept adding features in each Spiral cycle, delaying the release by 5+ years.
2) **Frequent risk analysis cycles led to excessive delays:** Vista took 5+ years to release. When it finally launched in 2007, the OS was slow, bloated, and hardware-incompatible.

# Agile Model – Microsoft

- After the Windows Vista failure, Microsoft switched to Agile for developing Windows 10 and Azure, embracing incremental updates and continuous integration.


- Pros:

1) **Continuous integration & delivery (CI/CD):** Windows 10 introduced Windows as a Service (WaaS), allowing small, frequent updates instead of one big release.
2) **Better adaptability:** Azure's cloud infrastructure evolved dynamically, responding to market demands and competing with AWS & Google Cloud.

- Cons:

1) **Initial struggles in shifting the mindset:** Microsoft engineers, used to Waterfall, found it hard to adapt to Agile's iterative nature, leading to internal resistance during early adoption.
2) **Lack of long-term stability for enterprise users:** Unlike older Windows versions (which had stable long-term support), Agile-driven Windows 10 forces frequent changes, making it difficult for businesses to plan upgrades.



# 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



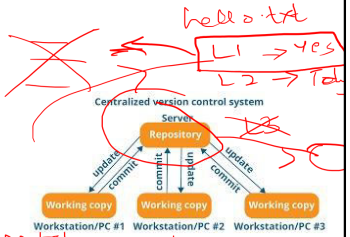
# Introduction to VCS

- 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!




# CVCS

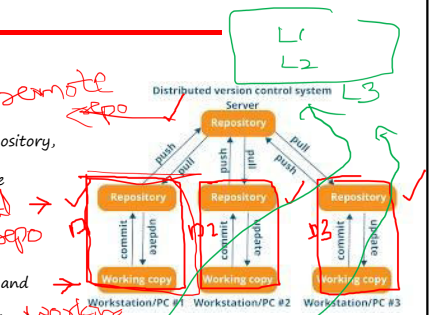
- **Examples:**
- CVS (Concurrent Versions System), Subversion (SVN)
- **It's a Single Repository:**
- CVCS uses a central server to store all versions of the code.
- Each user checks out code from this central repository and commits changes back to it.
- **History:**
- The central repository keeps the 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.





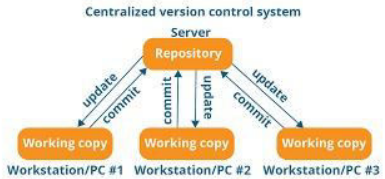

# DVCS

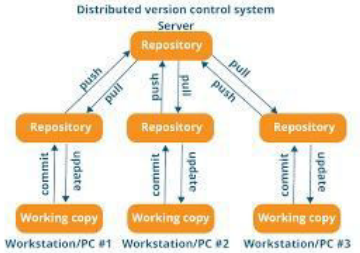
- **Examples:**
- Git, Mercurial
- **Multiple Repositories:**
- In DVCS, each user has a full copy of the entire repository, including its history.
- This means every user has a complete version of the codebase locally.
- **Collaboration:**
- DVCSs make collaboration more flexible.
- Users can commit 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.

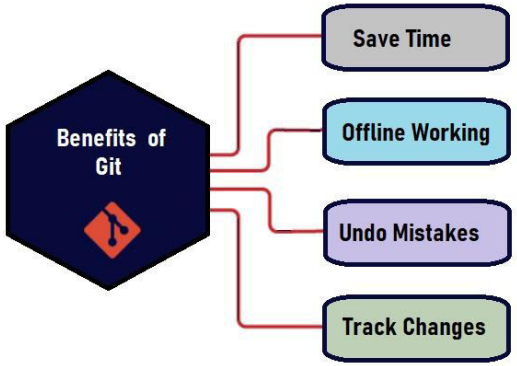
- **Development:**

  Torvalds began developing Git in April 2005.

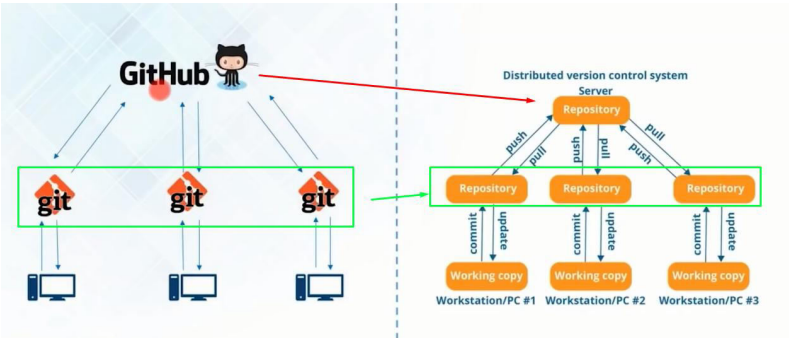
  His goal was to create a distributed version control system that was fast, efficient, and capable of handling large projects with many contributors.




# Introduction to Git – Advantages



# 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



# break time!!

break from
7:55 PM - 8:10PM


any questions?



# Installing Git

**Windows:**

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

**Mac:**

- Install via Homebrew:
- brew install git

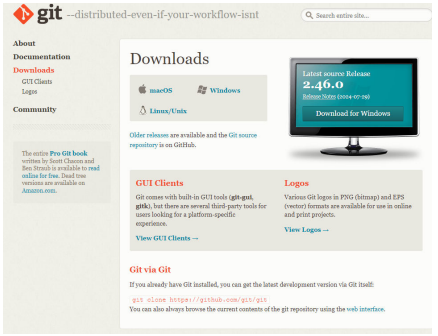
**Linux:**

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

**Verifying installation:**

git --version

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



# Installing Git

**Windows:**

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

**Mac:**

- Install via Homebrew:
- brew install git

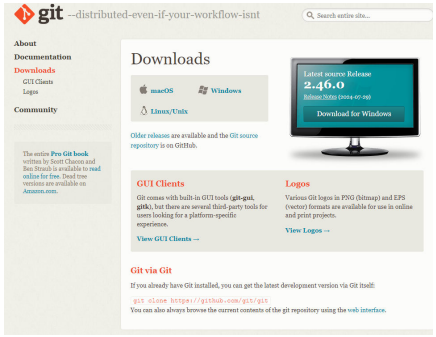
**Linux:**

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

**Verifying installation:**

```
git --version
```

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



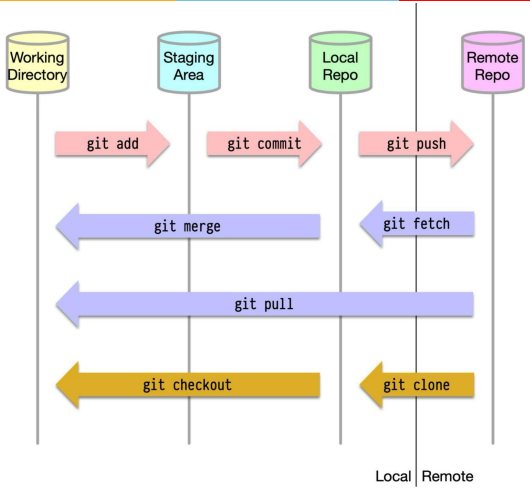



# Setting up your identity

- 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"`



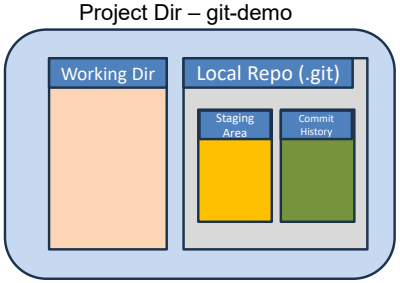
# Git Lifecycle Visualized






# Creating a New Local Repository


- 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



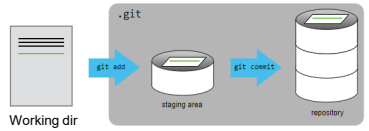
# Creating a New File

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

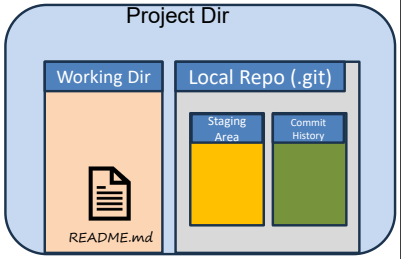
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha04/test.git
git push -u origin main
```

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

```
git remote add origin https://github.com/tanvitha04/test.git
git branch -M main
git push -u origin main
```




- Create a new file:
  - touch README.md





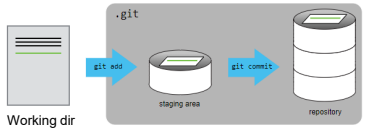

# Adding the file to staging area

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

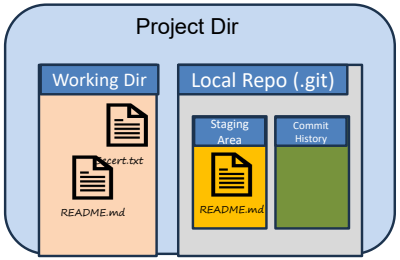
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha04/test.git
git push -u origin main
```

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

```
git remote add origin https://github.com/tanvitha04/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





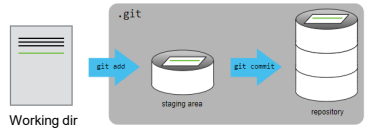

# Committing the file to local repo

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

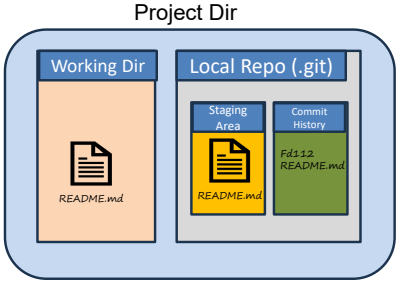
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha04/test.git
git push -u origin main
```

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

```
git remote add origin https://github.com/tanvitha04/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"



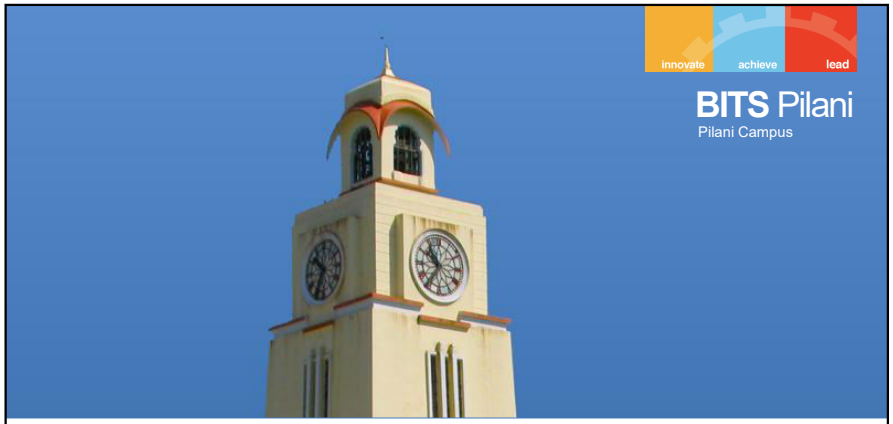
innovate achieve lead

# THANK YOU!



# DEVOPS FOR CLOUD
# (CC ZG507)

**BITS** Pilani
Pilani Campus



innovate achieve lead

**BITS** Pilani
Pilani Campus

# Contact Session - 4

innovate achieve lead

## 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

# Recap

- Git – DVCS
  - Local Repository
    - Staging area
    - Commit history
  - Remote Repository
- Git Basics commands
  - Creating Repositories, Clone
  - add , commit, push, pull
  - git status, logs
- How does Git keep track of files and changes?
  - Blob
  - Tree
  - Commit



# Agenda for today's class

- Git workflows
  - Master workflow
  - Centralized workflow
  - Feature workflow
- Feature branching
- GIT Pull requests
- Managing Conflicts
- Tagging and types
- Best Practices- clean code



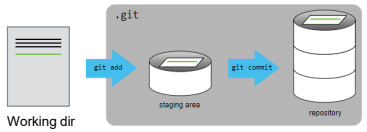
# Creating a New File

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

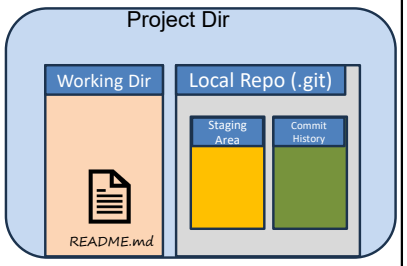
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha04/test.git
git push -u origin main
```

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

```
git remote add origin https://github.com/tanvitha04/test.git
git branch -M main
git push -u origin main
```




- Create a new file:
  - touch README.md





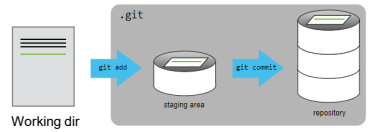

# Adding the file to staging area

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

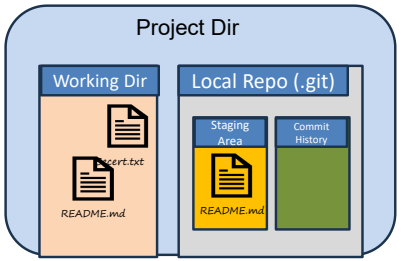
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha04/test.git
git push -u origin main
```

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

```
git remote add origin https://github.com/tanvitha04/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

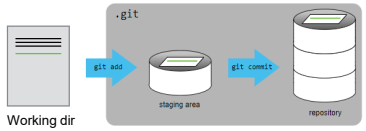


...or create a new repository on the command line

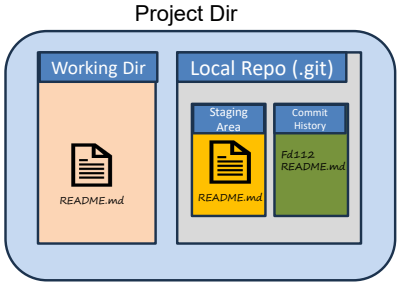
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/tanvitha94/test.git
git push -u origin main
```

...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.

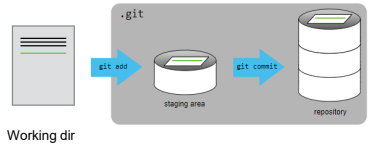
- To view commit history:
  - git log
- To view a simplified log:
  - git log --oneline
- To view changes introduced by each commit:
  - git show commit-hash

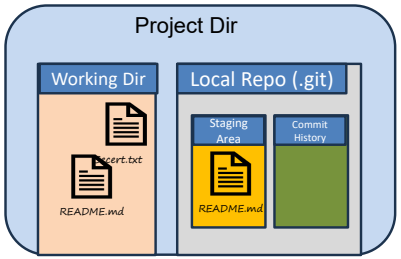


# Understanding .git directory



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






# Git add visualized



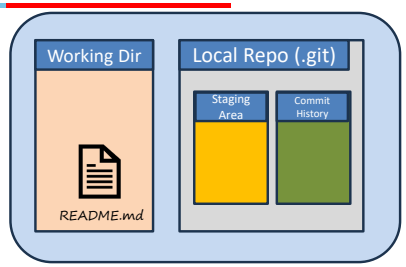

# How does Git keep track of files and changes? Blob!

**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.

- **What is SHA-1?**

SHA-1 is a cryptographic hash function that takes an input (file/directory) and generates a 40-character hexadecimal hash (160 bits).

- 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?

- 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



# 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



# Git State tracked after a commit

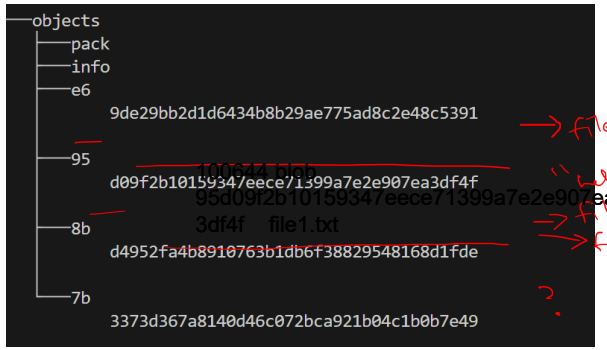

8b.. tree
100644 blob 95d09f2b10159347eece71399a7e2e907ea3df4f file1.txt

644 -> user group other
6 -> 110 -> read, write , execute
4 -> 100 -> read

# Git State tracked after a commit




# Git State tracked after a commit




a3.. Tree
100644 blob 95d09f2b10159347eece71399a7e2e907ea3df4f file1.txt
100644 blob 95d09f2b10159347eece71399a7e2e907ea3df4f file2.txt

ce.. Commit
a3 …



# break time!!

break from
2:25 PM - 2:35PM


any questions?



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

**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 file permissions, file type (blob or tree), and the SHA-1 hash of the object, filename.
- 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.



# Collaborating with Others

innovate achieve lead


- 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

**BITS** Pilani, Pilani Campus

# Git workflows

innovate achieve lead

- 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**

**BITS** Pilani, Pilani Campus

# Git workflows – Centralized Workflow

innovate achieve lead

- 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.

**BITS** Pilani, Pilani Campus

# Git workflows – Master Workflow

innovate achieve lead

- 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

innovate achieve lead

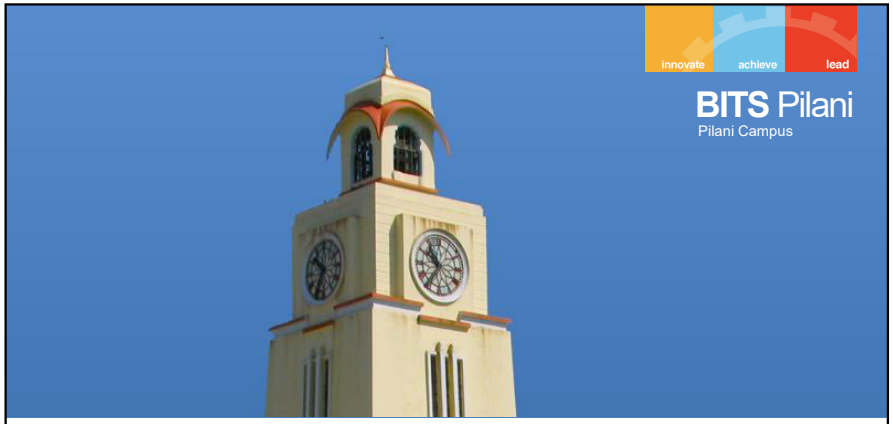
- This workflow is ideal for teams that work on multiple features concurrently.
- Each feature is developed in its own branch.
- **Branches:**
- 1) **main (or master):** The stable branch where all finished features are merged.
- 2) **feature-branches:** Separate branches for each feature being developed.
- **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.



innovate achieve lead

# THANK YOU!






# Contact Session - 5

## Recap



- Version control system and its types
- GIT Basics commands
- Git workflows



## Agenda for today's class

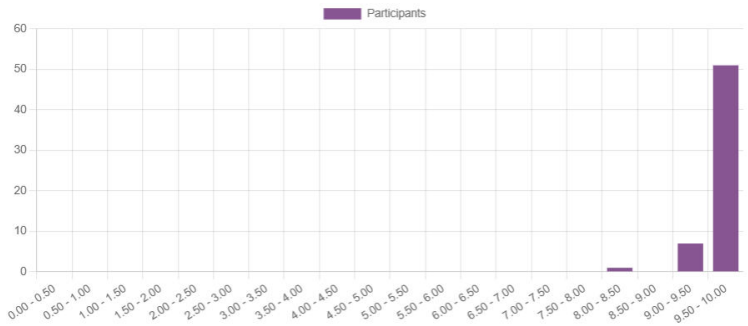


- Feature branching
- GIT Pull requests
- Managing Conflicts
- Tagging and Merging
- Git Best Practices- clean code
- Modern application requirements
- Cloud Native Applications
  - Cloud Enabled, Cloud Native Applications
- Monolithic vs Microservice



## Quiz -1 Marks - Analysis






| | |
|---|---|
| **Overall average** | **9.85 (59)** |

# Feature workflow - Merging

innovate achieve lead

- 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

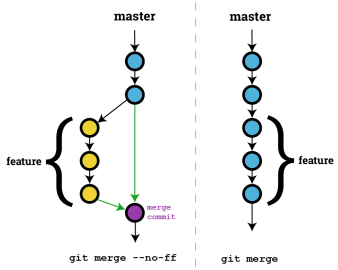
innovate achieve lead




# Types of Merging - Fast-Forward Merge

innovate achieve lead

- 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

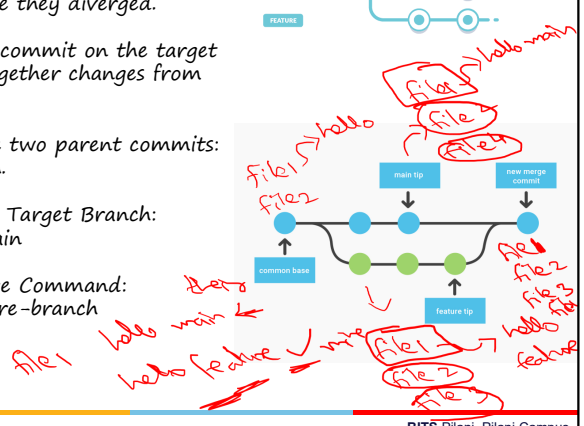






# Types of Merging - Three-Way Merge

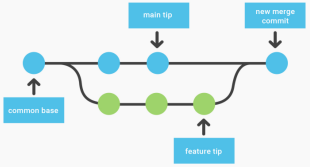
innovate achieve lead

- 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

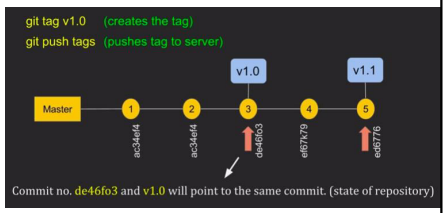






# Git Tagging - Types

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



# Git Best Practices- clean code

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

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



# 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.



# Cloud-enabled apps Example

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

Ex: Todo app -> java based and it was running tomcat on prem server and it connects to mysql db for data.
On prem – traditional

On cloud -> java based todo app will be running on tomcat based on AWS EC2 and it connects to AWS RDS for data.

# Cloud-based apps Example

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

Ex: Todo app -> deployed on AWS beanstalk with RDS for data.
Scale up and scale down

Todo -> user authentication , todo rest api c
Single codebase.
codebase is monolith or following monolithic arch



# Cloud-native apps Example

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

Ex: Todo app -> microservice arch
user authentication service -> python
todo service -> js
Multiple codebase.



# 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.



# 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.



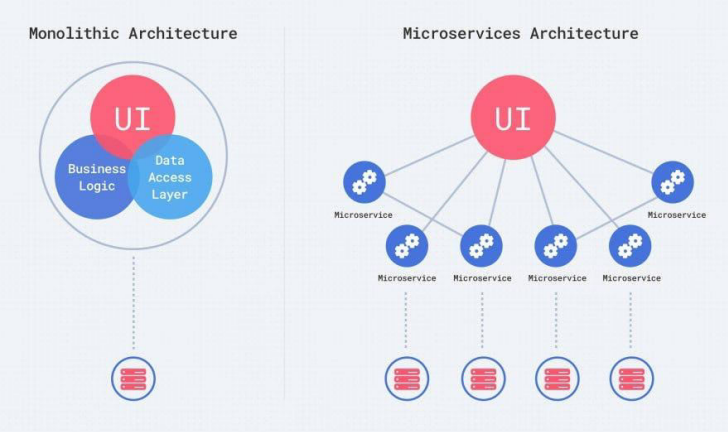
# Introducing Cloud-native software

**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

# Monolithic vs Microservices





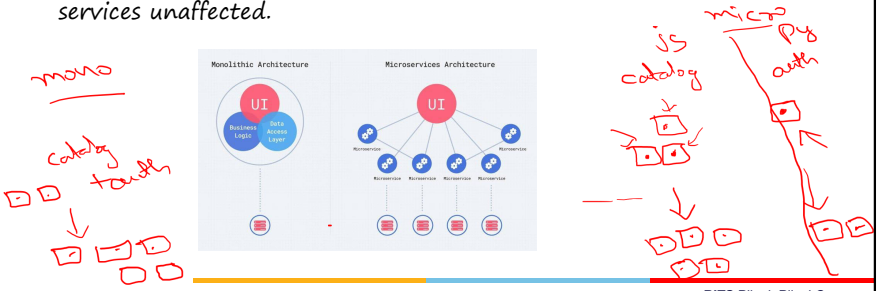

# Monolithic vs Microservices Real World Example

- You are working on an e-commerce website that was originally built using a monolithic architecture.
- The entire application, including user authentication, product catalog, shopping cart, payment processing is bundled into a single codebase and deployed as a single unit.
- As the website has grown, you've encountered issues such as difficulty in scaling specific components (e.g., payment processing), longer deployment times due to the need to redeploy the entire application for any change, and challenges in maintaining the codebase as multiple teams work on different parts of the application.
- To address these issues, your team has decided to transition from a monolithic architecture to a microservices architecture.



# Challenge#1 (Handling High Traffic for Specific Services)

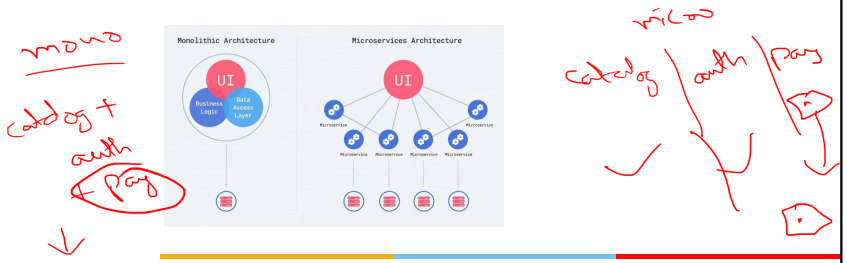
- In a monolithic architecture, the entire application must scale together, meaning even if only the Catalog Service experiences high traffic, the whole application instance must be scaled, leading to resource inefficiency.
- In a microservices architecture, only the Catalog Service can be scaled horizontally (ex: using Kubernetes or an auto-scaling group), ensuring efficient resource utilization while keeping other services unaffected.






# Challenge#2 (Frequent Updates Without Full Redeployment)

- In a monolithic system, even small changes (ex: updating the Payment Service) require a full redeployment, increasing downtime and risk of breaking unrelated services.
- With microservices, individual components like Payment Service or Authentication Service can be updated independently using CI/CD pipelines, ensuring faster deployment cycles without affecting other parts of the application.

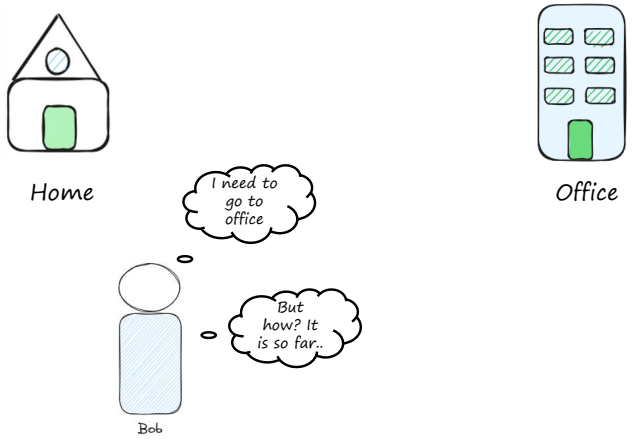


# Monolith vs Micro Service for Payments

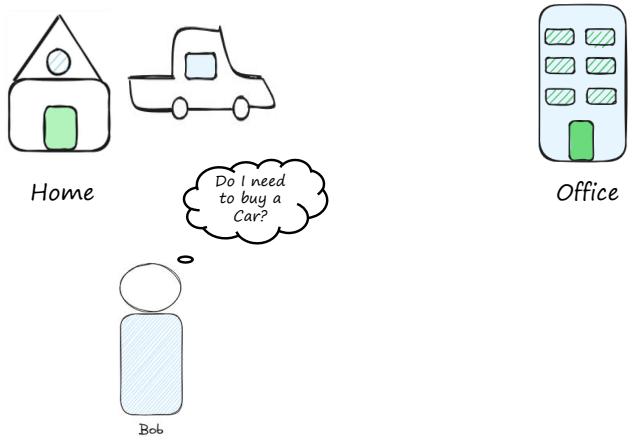
| Aspect | Advantages | Disadvantages |
|---|---|---|
| Scalability | ✓ Can scale the Payment Service independently based on transaction volume.<br>✓ Prevents unnecessary scaling of other services. | More complex load balancing and auto-scaling strategies required. |
| Resilience & Fault Isolation | ✓ Payment failures don't bring down the entire application.<br>✓ Can implement circuit breakers to prevent cascading failures. | Failure handling and retry mechanisms need careful implementation |
| Latency & Network Overhead | ✓ Service calls can be optimized using gRPC, asynchronous messaging (Kafka, RabbitMQ). | Additional network calls to external payment gateways (e.g., Stripe, PayPal) may introduce latency. |
| Technology Flexibility | ✓ Can use specialized databases (e.g., PostgreSQL for transactions, Redis for caching).<br>✓ Can choose the best programming language/framework for payments (e.g., Java, Go). | Polyglot architecture increases maintenance complexity. |
| Development & Deployment Speed | ✓ Can update the Payment Service independently without redeploying the entire system.- Enables CI/CD pipelines for rapid feature releases (e.g., adding a new payment provider). | Multiple deployments need orchestration, increasing DevOps workload. |



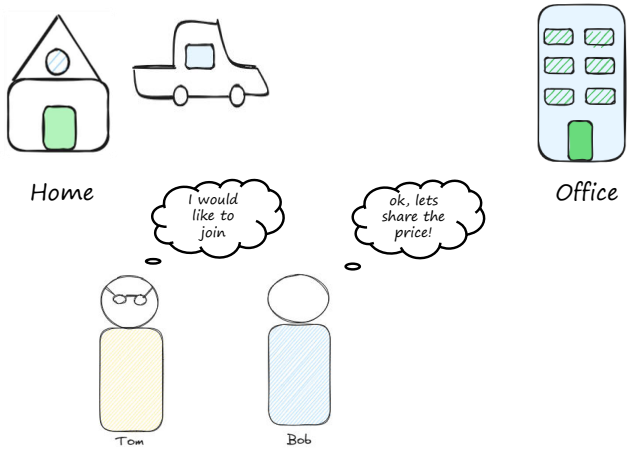
# Virtualization – Real World Analogy



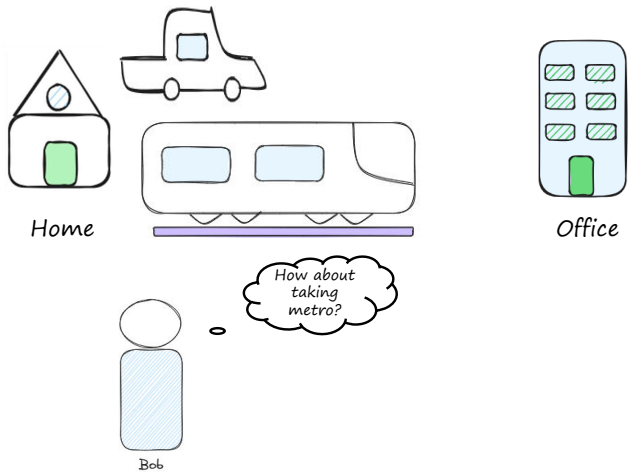

# Real World Analogy –private transport (car)




# Real World Analogy – scalability



# Real World Analogy – public transport (metro)

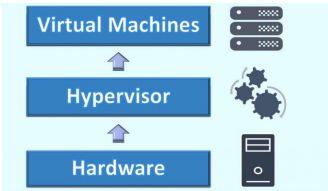



# Hypervisor-based Virtualization

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

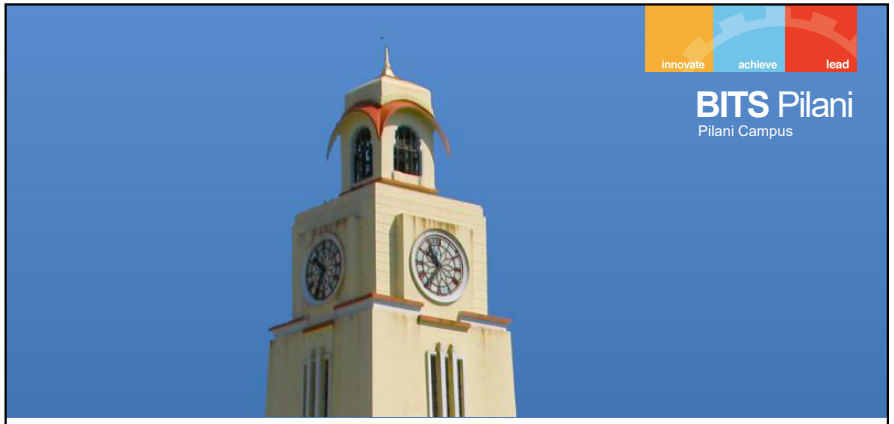
What does a hypervisor do?

- 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!






# Contact Session - 6

## Recap

- Best Practices- clean code
- Modern application requirements
- Cloud-native evolution
- Introducing Cloud-native software
- Cloud-enabled vs Cloud-based vs Cloud-native apps



## Agenda for today's class

- 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



## Cloud-native apps characteristics

- ✓ Microservices architecture instead of monolithic design.
- ✓ Containers for portability (Docker, Podman, etc.)
- ✓ Orchestration (e.g., Kubernetes, but also AWS ECS, Google Cloud Run, etc.)
- ✓ Serverless computing (AWS Lambda, Google Cloud Functions, etc.)
- ✓ Elastic scaling and automation (auto-scaling groups, managed databases)
- ✓ CI/CD Pipelines for rapid deployments

# 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:

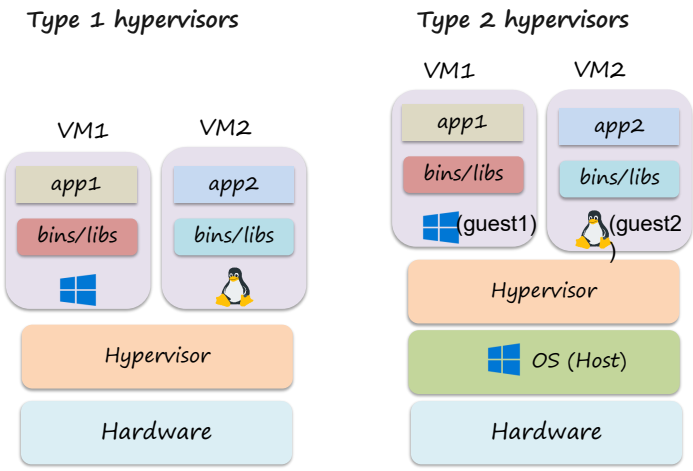
**Isolation**: Containers run isolated processes, ensuring consistency across environments.

**Efficiency**: More resource-efficient than virtual machines.

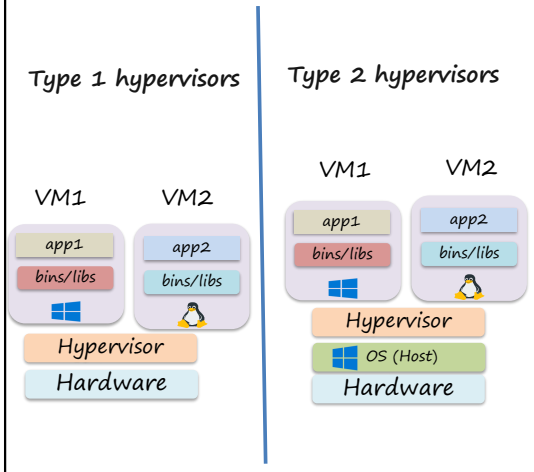
**Portability**: Run consistently on any environment with a container runtime (e.g., Docker).



# Types of Hypervisors




# Need for Containers

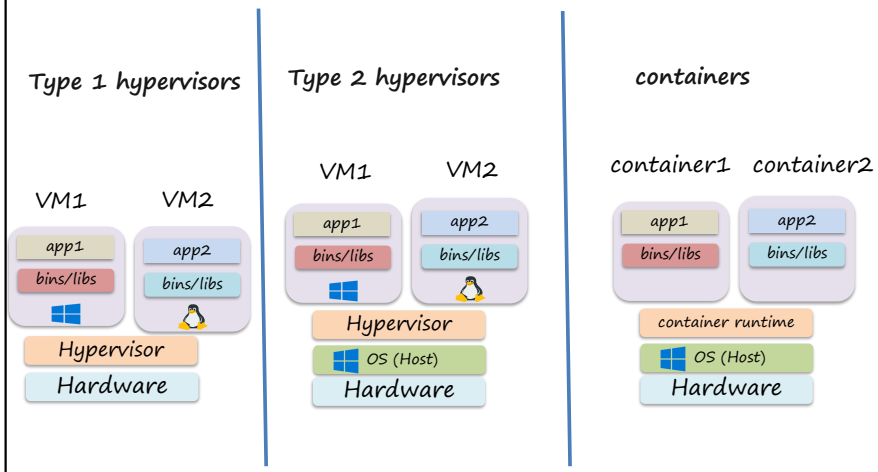



the virtual machines have an OS as well so are very heavy w.r.t file size

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

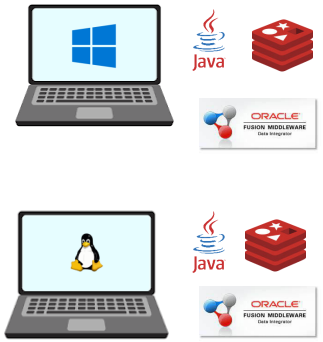
also need a lot of boot up time before we could run our app, can we do better?

# Hypervisors vs Containers Visualized




# Developer Onboarding – The Problem!

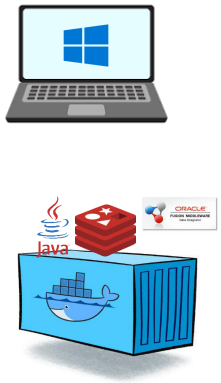
- lets say we have a dev team that works on data analytics product.
- So whenever a new developer on boards the team they need to setup all the tech stack needed for development.
- For example, an ETL tool like oracle data integrator and that needs jdk to be installed first and we also need a redis for 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!





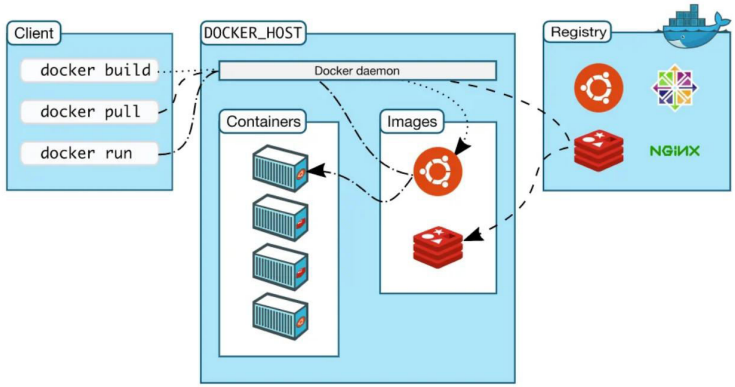

# Developer Onboarding – Docker container Accelerates!

- 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!



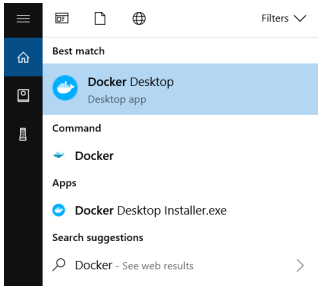



# Docker architecture



# Installing Docker on Windows 10

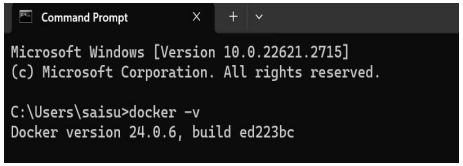
- In order to download and use docker, navigate to the official website of docker to download for windows – https://docs.docker.com/docker-for-windows/install/ and click on Docker Desktop for Windows as shown to the right.
- Once installed you should be able to find the docker desktop app in the search bar as shown to the right


# Advantages of using Docker


# 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:

# Creating our first docker container

innovate achieve lead

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

  **docker pull <image_name>**

  **Ex: docker pull ubuntu**

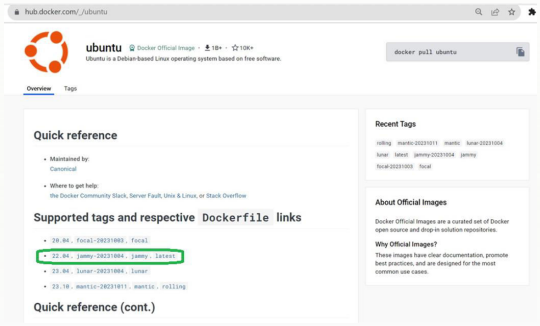
```
C:\Users\saisu>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
aece8493d397: Pull complete
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

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

innovate achieve lead

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



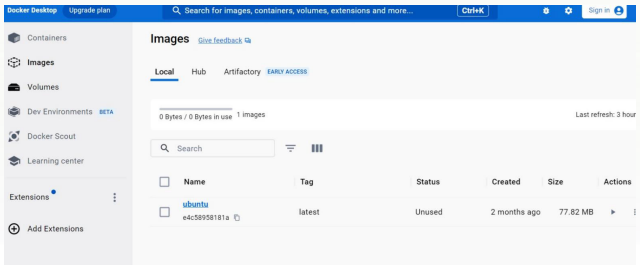

# Creating our first docker container

innovate achieve lead

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

innovate achieve lead

- 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:

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

innovate achieve lead

- 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

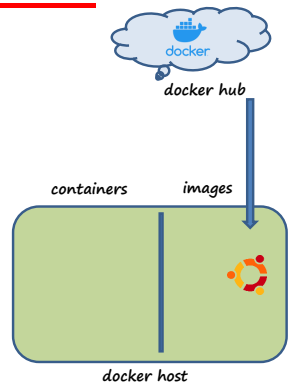
innovate achieve lead

- 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
ace8093d397: 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.

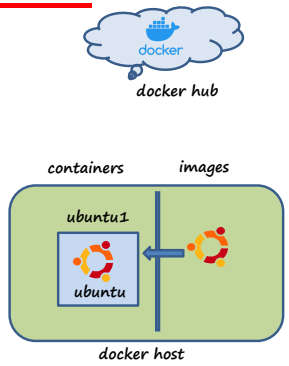
# Creating our first docker container – ubuntu

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




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

C:\Users\saisu>
```

# Where is our ubuntu container?

innovate achieve lead

- 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
```

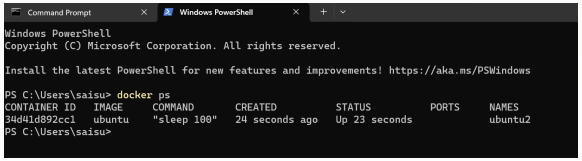
# Why is the ubuntu container not running?

innovate achieve lead

- 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**

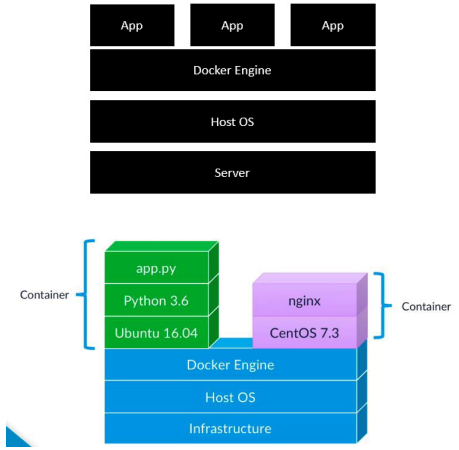
this will hang the current window prompt for 100 seconds ,let's open another command prompt and run docker ps




# Docker Containers – An example

innovate achieve lead

- 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

innovate achieve lead

- 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

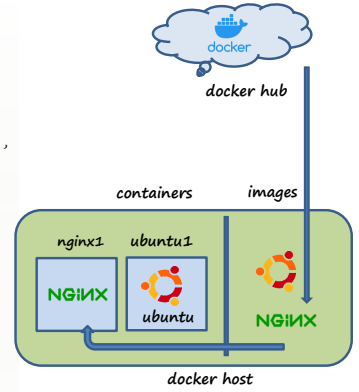
innovate achieve lead

- 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**


- 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

innovate achieve lead

- 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>
```

# Executing commands in docker container

innovate achieve lead

- 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

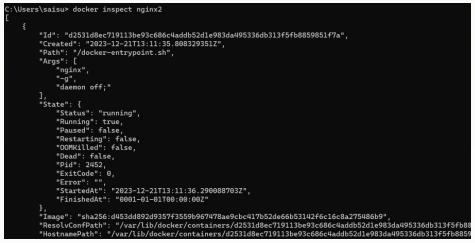
innovate achieve lead

- 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

innovate achieve lead

- 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**

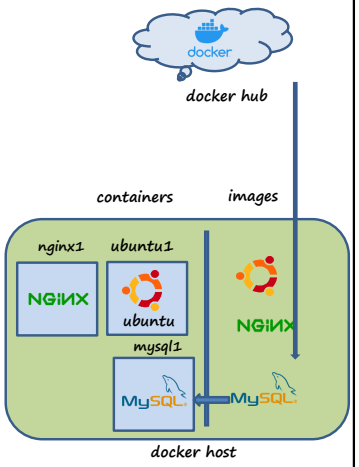
```
C:\Users\saisu>docker logs nginx2
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: 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

innovate achieve lead

- 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
bce831bc522d: Pull complete
cf7e9f463619: Pull complete
185f483783c7: Pull complete
378e53a613d8: Pull complete
2a362044e79f: Pull complete
6e4df4f73cfe: Pull complete
692636363755: Pull complete
fe5e855492202: Pull complete
5c02229ce6f1: Pull complete
7328aa32bf42: Pull complete
Digest: sha256:2d82ba6966cdf254de13f57d7265570c0675bc8bae1051e4a9806cff863006c9
Status: Downloaded newer image for mysql:latest
0c142754330f023c8ecc0ae641f015011c0be237e2c6f7bd7977f3b0fca5fd82a

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

innovate achieve lead

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
0c142754330f   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
3d4d1d892cc1   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

innovate achieve lead

```
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

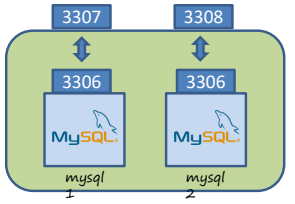
innovate achieve lead

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>**

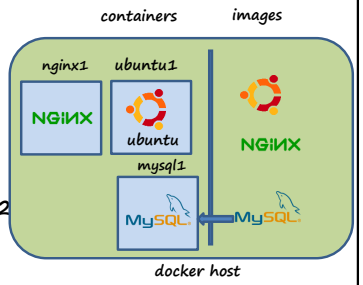
Ex: docker stop mysql1

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>**

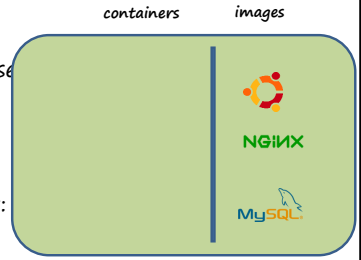
Ex: docker rmi mysql

- 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

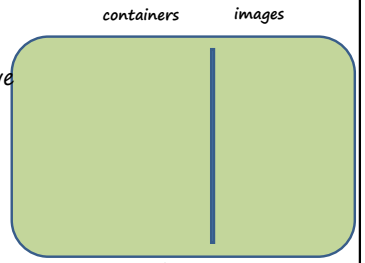


- this brings us back to original state we started with

- 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.


docker hub




docker host

# Creating a custom docker image



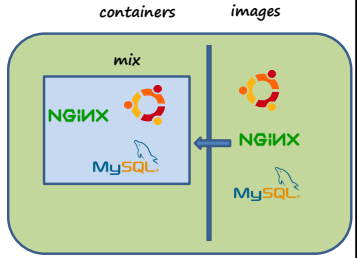
- Dockerfile specifies all the steps defining what all are needed to be done on the container once created.

- 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

# 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)
```

# THANK YOU!





# 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

- Containers Continued
  - Creating a custom docker image – Dockerfile
  - Building and Running Custom Image
- Traditional CI setup
- Building the application



# 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

| 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 |

# 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

- 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.

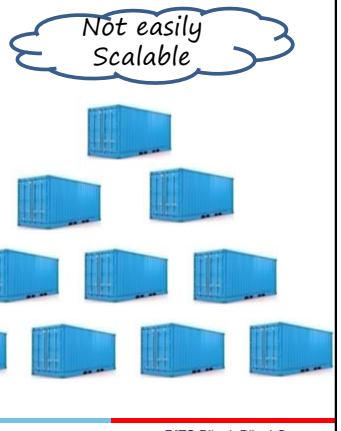


# The need for Orchestration

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

kubernetes

HashiCorp Nomad

minikube

OPENSHIFT

CLOUDIFY



# 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.



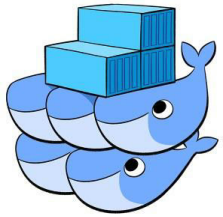
# History of Container Orchestration Tools – 2

- **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.

kubernetes

# Docker Swarm

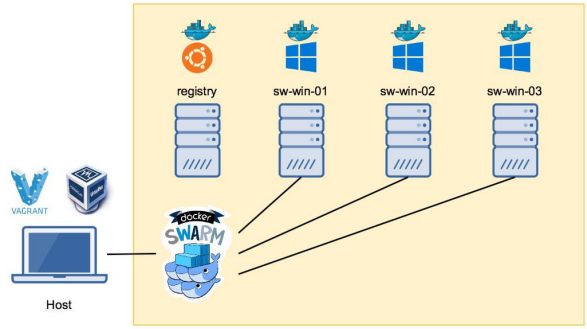
- Docker Swarm is an orchestration management tool that runs on Docker applications.
- While Docker is great for running containers on a single machine, Swarm turns several Docker hosts into a cluster, coordinating their efforts.
- Docker Swarm allows you to manage multiple containers on multiple hosts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





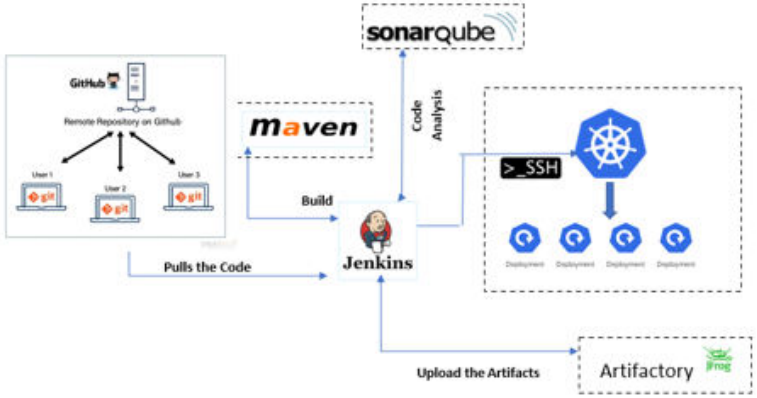
# Docker Swarm

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






# Traditional CI setup






# Key Components of a Traditional CI Setup

**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

innovate achieve lead

**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

innovate achieve lead

**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



# 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

# 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>
```

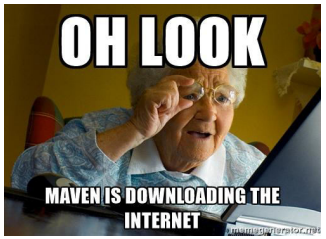
```
    <build>
        <plugins>
            <plugin>

                <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

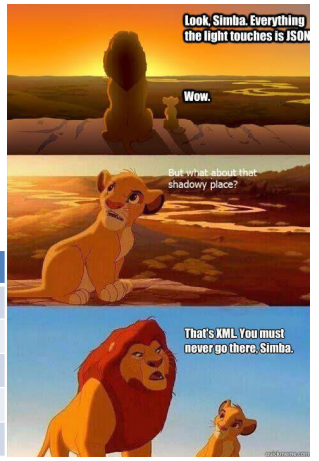
1) **mvn clean:** Removes the target directory to start with a clean build.
2) **mvn compile:** Compiles the source code in the src/main/java directory.
3) **mvn test-compile:** Compiles the test source code in the src/test/java directory.
4) **mvn test:** Executes the unit tests.mvn package: Compiles 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.



# Maven drawbacks , Gradle!




| Feature | Maven | Gradle |
|---|---|---|
| Incremental Compilation | No | Yes |
| Build Caching | No | Yes (--build-cache) |
| Parallel Execution | No | Yes (--parallel) |
| Automatic Rebuild | No | Yes (--continuous) |
| Task Up-to-Date Check | No | Yes |






# 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.





# THANK YOU!

# DEVOPS FOR CLOUD
# (CC ZG507)

BITS Pilani
Pilani Campus

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

# Gradle steps for a Java project

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!'
    }
}
```

```
my-java-project/
├── src/
│   ├── main/
│   │   └── java/
│   │       └── com/
│   │           └── example/
│   │               └── App.java
│   └── test/
│       └── java/
│           └── com/
│               └── example/
│                   └── AppTest.java
├── build.gradle
└── settings.gradle
```



# Gradle steps for a Java project – impl vs testImpl

1) **Implementation:**

- Used for main application dependencies.
- These dependencies are required for compiling and running the main application. (src/main/java)
- They are not available to test code. (src/test/java)

2) **testImplementation:**

- Used for dependencies required only for testing.
- These dependencies are available only in test code (src/test/java).
- They are not included in the final build of the application.



# Java plugin - tasks

```
:build
+--- :assemble
|    \--- :jar
|         \--- :classes
|              +--- :compileJava
|              \--- :processResources
:check
\--- :test
     +--- :classes
     |    +--- :compileJava
     |    \--- :processResources
     \--- :testClasses
          +--- :compileTestJava
          |    \--- :classes
          |         +--- :compileJava
          |         \--- :processResources
          \--- :processTestResources
```

depends on

= 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

innovate achieve lead

- 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.
- To generate token navigate to your profile on dashboard (top-right corner) → My Account. Go to the Security tab. Under "Generate Tokens": Enter a name (e.g., GradleToken). Click Generate. Copy the generated token (you will not be able to see it again!).



# Integrate SonarQube with Gradle

innovate achieve lead

Add the following code to your build.gradle:

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

BITS Pilani, Pilani Campus

# 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:

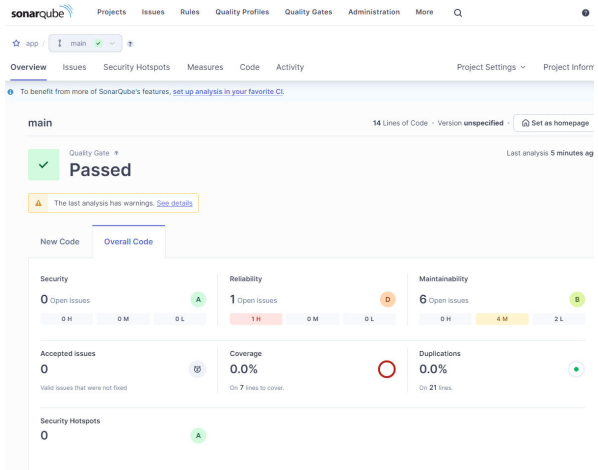
```
./gradlew sonarqube
```

## 4) Analyze Results on SonarQube Dashboard

BITS Pilani, Pilani Campus

# Integrate SonarQube with Gradle

innovate achieve lead




BITS Pilani, Pilani Campus

# 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 Question Paper format and Topics

- 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!

# Kubernetes

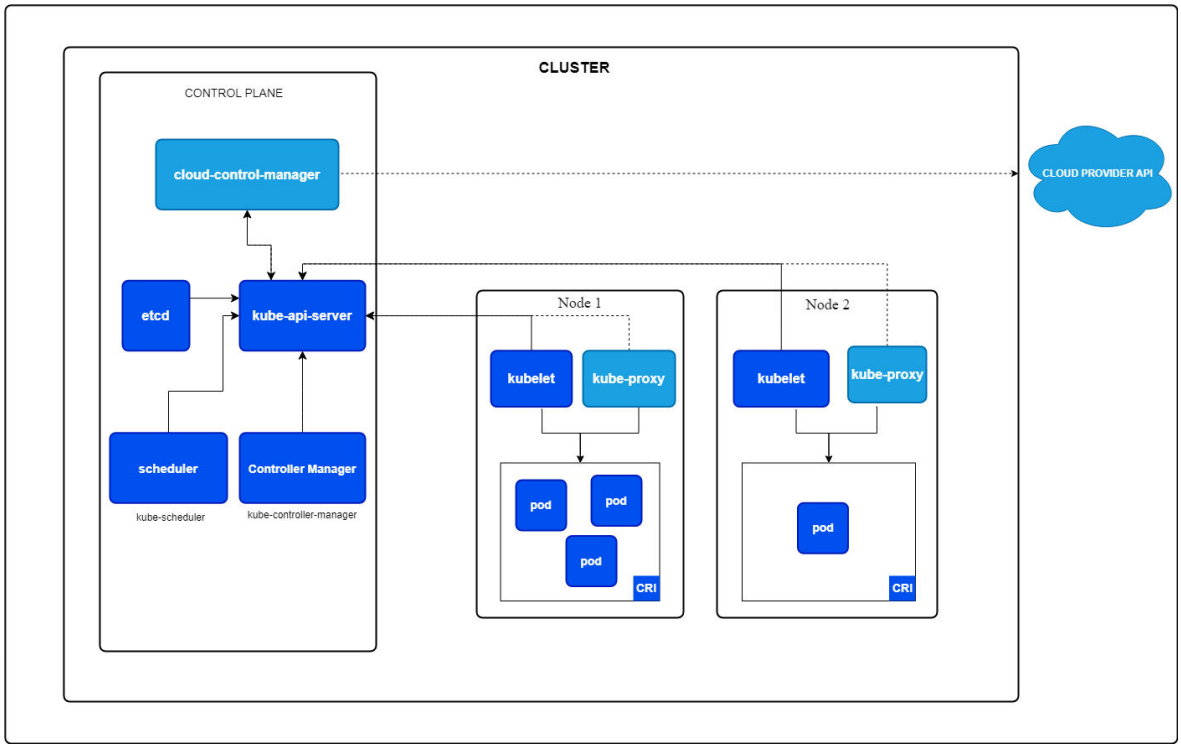
- It is an open-source container orchestration tool that was originally developed and designed by engineers at Google.
- Kubernetes orchestration allows you to build application services that span multiple containers, schedule containers across a cluster, scale those containers, and manage their health over time.



# Kubernetes Cluster Architecture – Details

- 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.

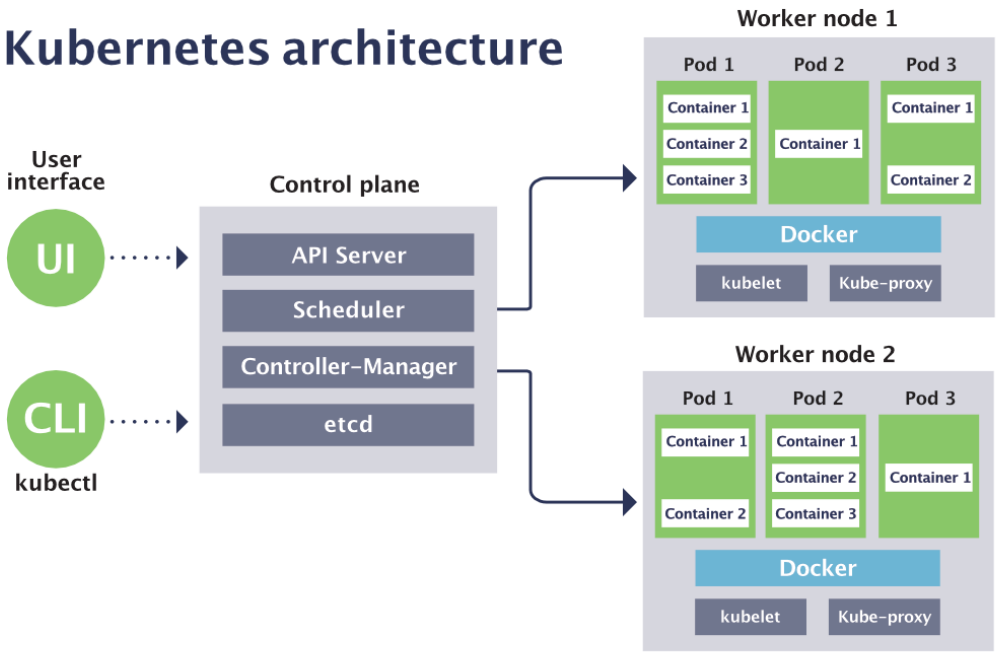
# Kubernetes Cluster Architecture






innovate achieve lead

# Kubernetes Cluster Architecture




BITS Pilani, Pilani Campus

# Minikube Setup on Windows – 1

- 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

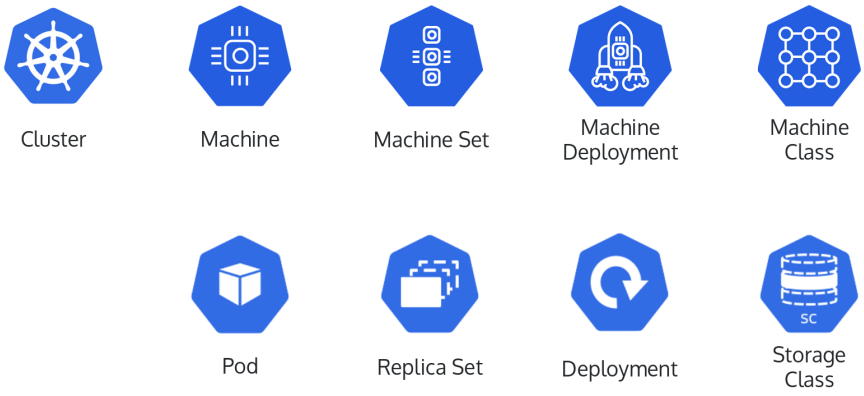
**Step3: Start Minikube**

- 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

**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

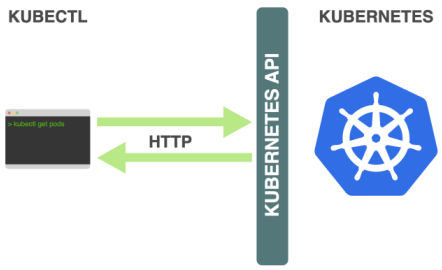
**2) help**
**kubectl help**

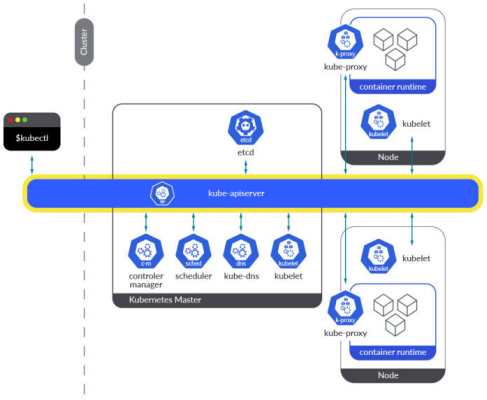
- to get commands help

**3) nodes**
# to get the list of nodes running
**kubectl get nodes**

# to get list of multiple components
**kubectl get pods,svc**

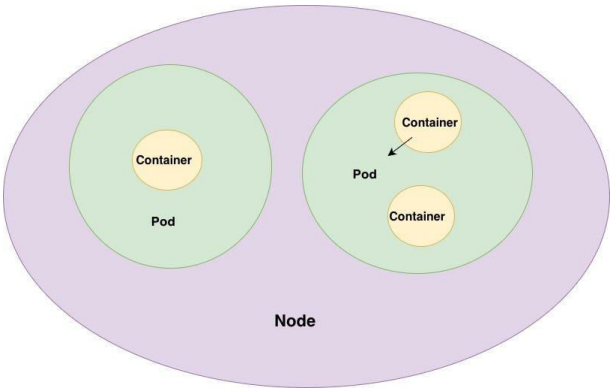
# for more verbose info use
**kubectl get nodes -o wide**



# 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.




innovate achieve lead

# 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
```



innovate achieve lead

# 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)

# Running a Pod

**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**

| XML | JSON | YAML |
|---|---|---|
| <Servers><br><Server><br><name>Server1</name><br><owner>John</owner><br><created>123456</created><br><status>active</status><br></Server><br></Servers> | {<br>Servers: [<br>{<br>name: Server1,<br>owner: John,<br>created: 123456,<br>status: active<br>}<br>]<br>} | Servers:<br>- name: Server1<br>owner: John<br>created: 123456<br>status: active |

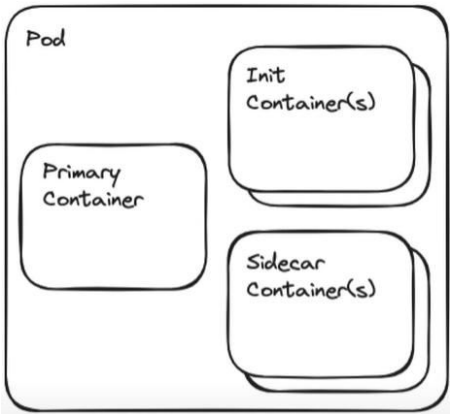


innovate achieve lead

# Init and Sidecar containers

- **Init Container:** runs a setup task before the primary container starts.
- **Primary Container:** the main application running in the Pod.
- **Sidecar Container:** runs alongside the primary container, providing auxiliary services.
- Ex:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  initContainers:
  - name: init-container
    image: busybox
    command: ["sh", "-c", "echo 'Initializing...'; sleep 5; echo 'Init Complete'"]

  containers:
  - name: primary-container
    image: nginx
    ports:
    - containerPort: 80

  - name: sidecar-container
    image: busybox
    command: ["sh", "-c", "while true; do echo 'Sidecar running...'; sleep 10; done"]
```



# 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

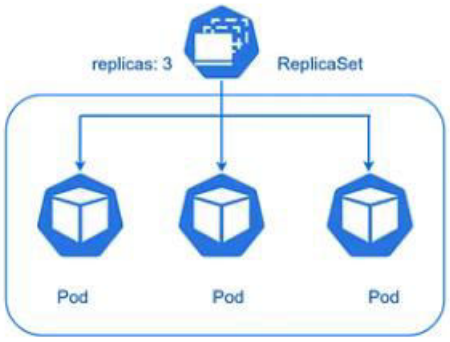



Fig: ReplicaSet



innovate achieve lead

# 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
```

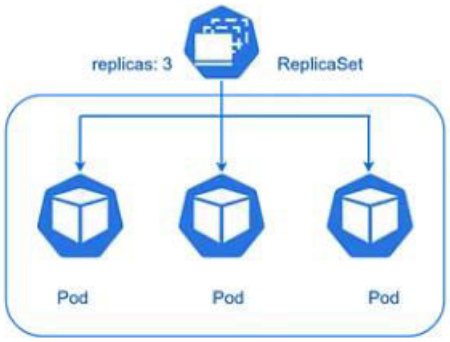



Fig: ReplicaSet

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

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

# Namespace

- A namespace in Kubernetes is a virtual cluster within a physical cluster. It helps logically separate and organize resources like pods, services, and deployments.

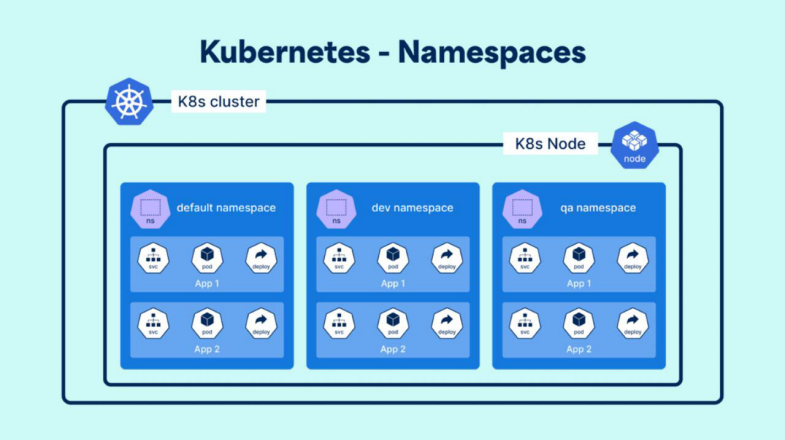
- **Creation:**

kubectl create namespace <name>
Or
kubectl apply -f Namespace.yaml

- **Deletion:**

kubectl delete namespace <name>
Or
kubectl delete -f Namespace.yaml

Ex:
```
apiVersion: v1
kind: Namespace
metadata:
  name: dev-namespace
```




Common Namespaces:
1) default: The default namespace if none is specified.
2) kube-system: Reserved for Kubernetes system components.
3) kube-public: Public resources accessible to all.
4) kube-node-lease: Manages node heartbeats.



innovate achieve lead

# Namespace – Naming Guidelines

1) **Use lowercase letters:** Kubernetes namespaces are case-sensitive and should be lowercase.

2) **Use hyphens (-) as word separators instead of underscores (_):** Hyphens improve readability.

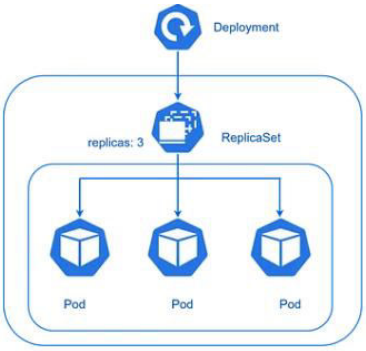
3) **Keep it short but meaningful:** Avoid overly long names.

4) **Avoid special characters:** Stick to alphanumeric characters and hyphens.

5) **Use a standard prefix if needed:** Useful for multi-team clusters (Ex: team1-app, team2-app).

# Deployment

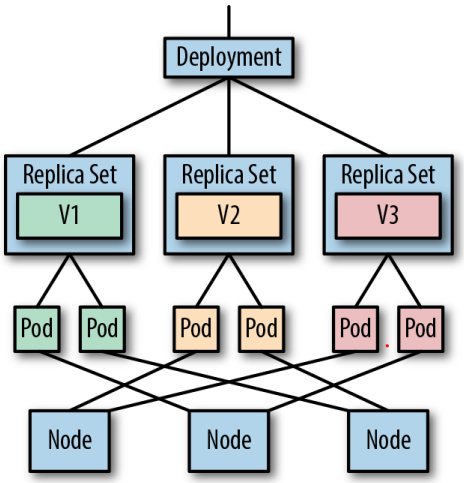
- Wrapper on replicasets
- 1 replicatSet would be created for 1 deployment.
- provides declarative updates to applications for roll over updates and rollbacks.
- When using the RollingUpdate strategy, Kubernetes gradually replaces old Pods with new ones, instead of stopping all Pods at once.
- This ensures high availability of your application during updates and zero downtime upgrades!






# Deployment vs ReplicaSet vs pod

- Let's say your app is running with v1, and you want to upgrade to v2.
- You can create a Deployment for v1 that initially runs 5 replicas.
- When you want to update the app to v2, you update the Deployment configuration.
- If your Deployment has 5 replicas and maxSurge is set to 3, Kubernetes can create up to 8 Pods (5 + 3) during the update to ensure that v1 Pods are gradually replaced by v2 Pods, without downtime.

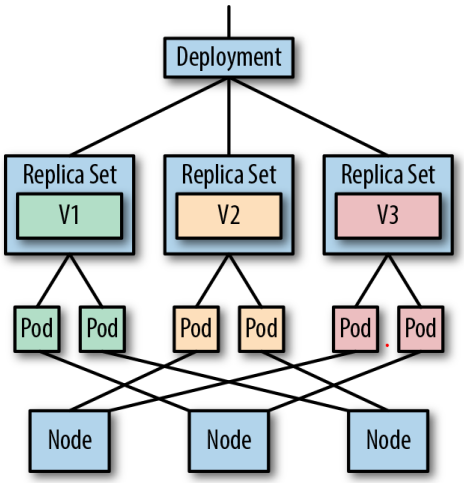



- Ex:

```
replicas: 5
strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 3 # Allow up to 3 extra pods during update
      maxUnavailable: 1  # Ensure at least 4 pods are running during update
```

# Deployment vs ReplicaSet vs pod

- V1 V1 V1 V1 V1
- V2 V2 V2 V1 V1 V1 V1 V1 V1
- V2 V2 V2 V2 V1 V1 V1 V1 V1
- V2 V2 V2 V2 V2 V1 V1 V1 V1
- V2 V2 V2 V2 V2 V2 V1 V1 V1
- V2 V2 V2 V2 V2 V2




```
replicas: 5
strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 3 # Allow up to 3 extra pods during update
      maxUnavailable: 1  # Ensure at least 4 pods are running during update
```



# Deployment definition

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-dep
spec: (contains template of pod to be replicated)
  template:
          <your_pod-def>
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 3
```

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!

# Custom frontend App Deployment on k8s


1) Create frontend folder

2) Create index.html in it

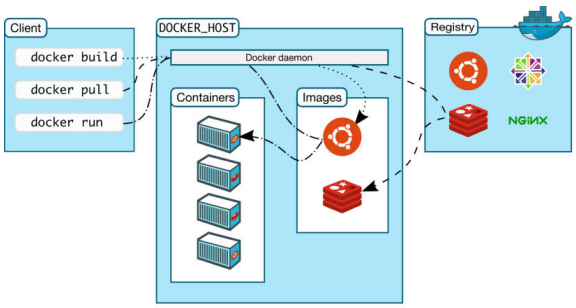
3) Create a Dockerfile

4) Build the Docker Image

5) Run and test the Docker Container Locally

6) Push the image to docker registry (public – dockerhub or private – container registries)

6) Deploy the Container to Kubernetes
   a. create Configmaps, Secrets
   b. create Deployment
   c. create Service to expose pod to external world






# Steps 1-2) Create front end folder and index.html


```
<!DOCTYPE html>
<html>
    <head>
        <title>Front end app</title>
    </head>

    <body>
        <h1 id="app-title"></h1>
        <p>Version: <span id="app-version"></span></p>
        <p>API key: <span id="api-key"></span></p>
    </body>
</html>
```

# Steps 3-5) Dockerfile, building and running the custom image


**Create a Dockerfile:**

```
# Use the official Nginx image as the base image
FROM nginx:alpine

# Copy the HTML file to the Nginx web server directory
COPY ./index.html /usr/share/nginx/html/index.html

# Copy the entrypoint script
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# Expose port 80
EXPOSE 80

# Set the entrypoint to run the script
ENTRYPOINT ["/entrypoint.sh"]
```

**Build the Docker Image:**

```
docker build -t frontend-app .
```

**Run and test the Docker Container Locally:**

```
docker run -d -p 8080:80 hello-world-app
```




# ConfigMap

- ConfigMap is an API object that allows you to store non-sensitive configuration data as key-value pairs.
- It can be consumed by pods or other objects.
- Ex:

**frontend-app-config.yaml**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-app-config
data:
  APP_TITLE: "Hello World Frontend"
  APP_VERSION: "1.0"
```

**Deploy the config map**

```
kubectl apply -f frontend-app-config.yaml
```

**Verify the deployment:**

```
kubectl get configmap
or
kubectl get cm
```

# Secret

- Secret is an API object that allows you to store sensitive configuration data as key-value pairs.
- It can be consumed by pods or other objects.
- Ex: to generate base64 encoded value for API_KEY
- echo -n 'my api key pass' | base64

**app-secret.yaml**

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
type: Opaque
data:
  API_KEY: cGFzc3dvcmQ=
# base64-encoded value for "password"
```

**Deploy the config map**

kubectl apply -f db-secret.yaml

**Verify the deployment:**

kubectl get secret



# Deploy the Container to Kubernetes

**frontend-app-dep.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-app-dep
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend-app
  template:
    metadata:
      labels:
        app: frontend-app
    spec:
      containers:
      - name: frontend-app-container
        image:
        ports:
        - containerPort: 80
```

```
env:
  - name: APP_TITLE
    valueFrom:
      configMapKeyRef:
        name: frontend-app-config
        key: APP_TITLE
  - name: APP_VERSION
    valueFrom:
      configMapKeyRef:
        name: frontend-app-config
        key: APP_VERSION
  - name: API_KEY
    valueFrom:
      secretKeyRef:
        name: app-secret
        key:  API_KEY
```

**Deploy the pod**

kubectl apply -f frontend-app-dep.yaml

**Verify the deployment:**

kubectl get pods

# Entrypoint

## entrypoint.sh

```
#!/bin/sh
# Replace the placeholder for APP_VERSION in index.html

sed -i "s/<span id=\"app-version\"><\/span>/<span id=\"app-
version\">$APP_VERSION<\/span>/" /usr/share/nginx/html/index.html

# Replace the placeholder for API_KEY in index.html

sed -i "s/<span id=\“api-key\"><\/span>/<span id=\"db-password\">$API_KEY<\/span>/"
/usr/share/nginx/html/index.html


# Start Nginx or another web server
nginx -g "daemon off;“
```

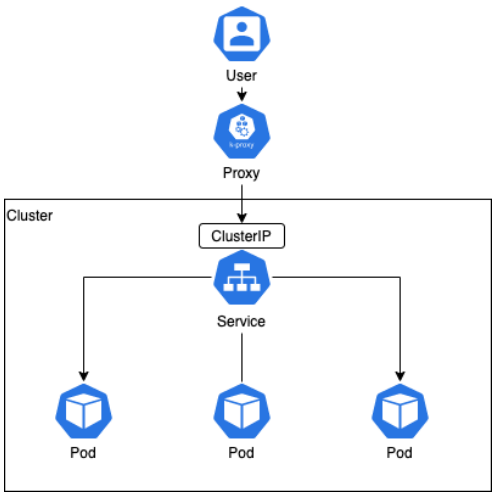
**Note: Decoding password if needed:**

```
echo "c2VjdXJlX3Bhc3N3b3Jk" | base64 --decode
```



# Service

- the ip addr associated with pod may change as and when pod crashes and new one replaces it.
- To ensure we have static name to refer to a deployment, we have a service.
- Service acts as a load balancer and proxy.




**use cases:**

- 1) to expose one pod to another pod via endpoint (service name Ex: redis-db for redis database pod) in the k8 cluster.
- 2) to expose pod to external client outside k8 cluster

# Deploy the Container to Kubernetes - Service

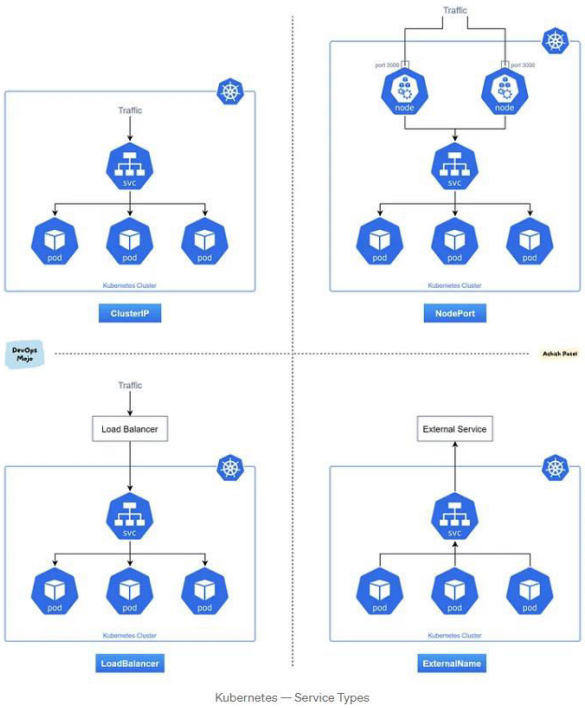
**frontend-app-svc.yaml**

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-app-svc
spec:
 type: ClusterIP
 selector:
    app: frontend-app
 ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```



# Service types

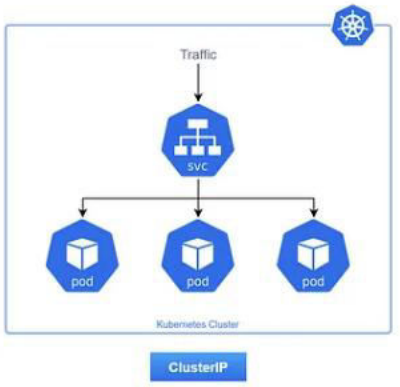
- Kubernetes provides several types of Services to facilitate communication between Pods and external users.
- Each Service type serves a different purpose in managing access and load balancing.
- Here are the primary types of Kubernetes Services:
- ClusterIP,
- NodePort,
- LoadBalancer and
- ExternalName




Kubernetes — Service Types

# Service types – ClusterIP

- Default Service Type.
- Exposes the Service on a cluster-internal IP.
- Only accessible within the cluster and not accessibl from outside.
- Useful for internal communication between Pods.

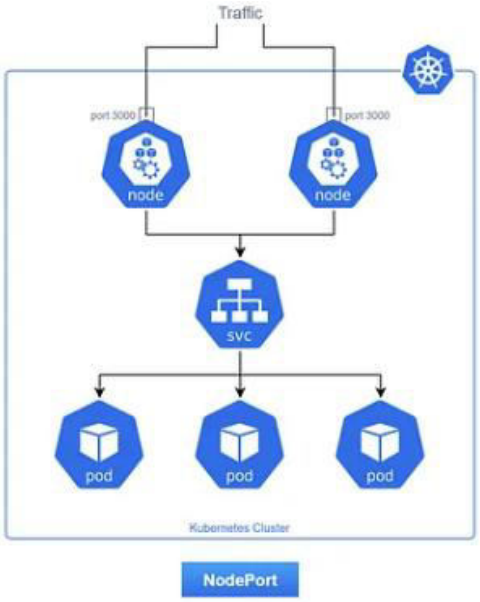



```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```



# Service types – NodePort

- Exposes the service externally on a specific static port on all nodes in the cluster.
- Accessible from outside the cluster by <NodeIP>:<NodePort>.
- Routes traffic to the appropriate Pod(s) within the cluster.
- Useful for simple external access to Services for testing or when you want direct access from outside the cluster.



# Service definition example for Nodeport Service Type

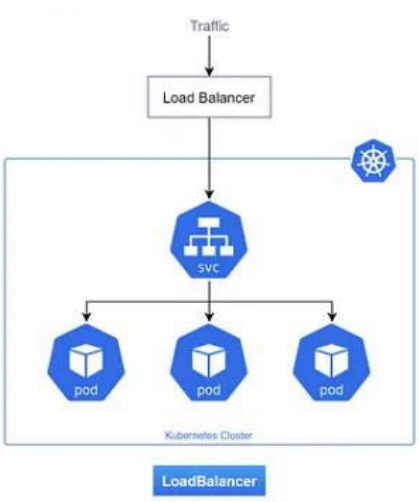

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec: (contains template of pod to be replicated)
  type: NodePort
  ports:
  - port: 80 #(service port)
    targetPort: 8080 #(container port)
    nodePort: 30008 #(node port)
  selector: (to link what all pods this service will be linked to)
    name: redis-pod
```

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!



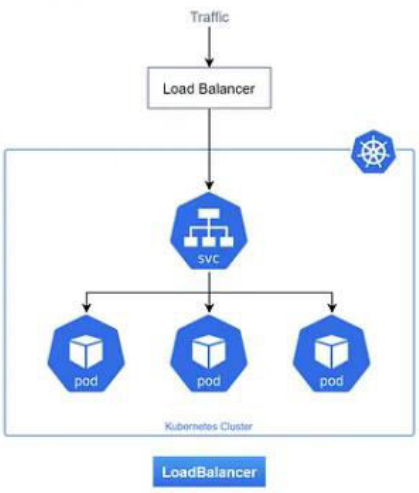

# Service types – LoadBalancer

- Exposes the Service externally using a cloud provider's load balancer.
- Automatically assigns a public IP address for the Service.
- Routes traffic to the corresponding NodePort or ClusterIP Service.
- Useful for production environments where you need reliable external access.



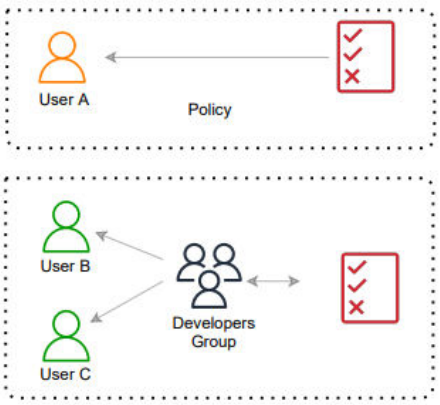
# Service type – LoadBalancer Example

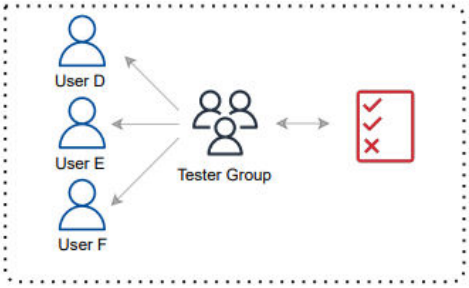
```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx-app
  ports:
    - protocol: TCP
      port: 80        # Exposed port
      targetPort: 80  # Port inside the Pod
  type: LoadBalancer

# External load balancer, exposes service to the
internet
```






# AWS – IAM Users , Policies, Groups




AWS Policies – AWS managed vs custom

# Clusterless Container services

- Clusterless container services refer to container platforms or frameworks that allow users to run and manage containers without directly managing the underlying cluster infrastructure.
- Infrastructure management means taking care of nodes, networking, and scaling.
- These services abstract away the complexities of orchestrating and managing a cluster like Kubernetes does.
- Ex:
  - AWS Fargate (for ECS and EKS)
  - Azure Container Instances (ACI)
  - Google Cloud Run
- **Benefits:**
  - **No Infrastructure Management:** You don't need to manage servers, clusters, or nodes.
  - **Automatic Scaling:** Services can automatically scale containers up or down based on demand.
  - **Cost-Efficiency:** Pay only for the compute resources you use, avoiding over-provisioning.



# AWS Fargate Example

**1) Create a Docker image:**
This can be any simple application, for example, a Python web app.

2) **Define ECS Task:**
- You define a task with a container definition in Amazon ECS.
  - CPU: 256 MB
  - Memory: 512 MB
  - Container Image: Your image from ECR (my-python-app)
  - Port Mappings: Expose port 80 (or any other)
  - Any other config like configmap , secrets etc

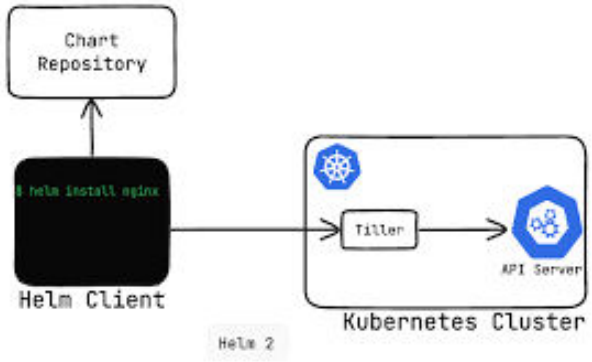
**3) Create a Service:**
- In ECS, define a Fargate service that runs your task:
  - Desired number of tasks: 1
  - Networking: VPC, Subnet, and Security Groups
  - Auto-scaling options, if required.

**4) Run the Service:**
- AWS Fargate will launch your container without you needing to manage any infrastructure (like EC2 instances).

**5) Access the Application:**
- Once the service is up and running, AWS Fargate will assign a load balancer or public IP (depending on your setup) so you can access your application.

# What is Helm?

- Helm is a package manager for Kubernetes, designed to simplify the process of deploying, managing, and maintaining Kubernetes applications.
- Helm enables us to define, install, and upgrade complex Kubernetes applications using **Helm charts**.
- Helm charts are collections of pre-configured Kubernetes resources.





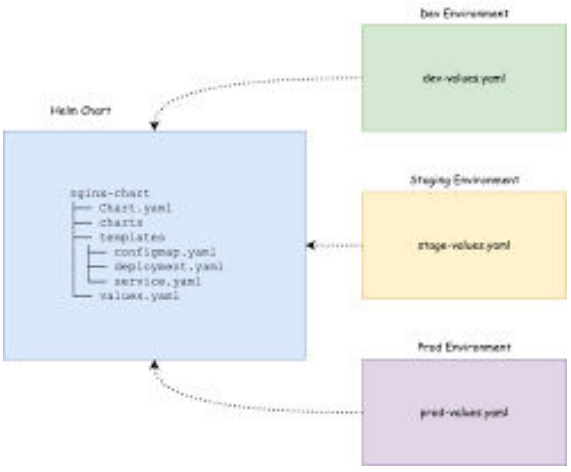

# Helm – 3 key use cases!

**1) Templating Engine:**

Use a single configuration file (values.yaml) to manage parameters like the number of replicas, environment variables, secrets, and more.

Ex: **Deploy Applications:**

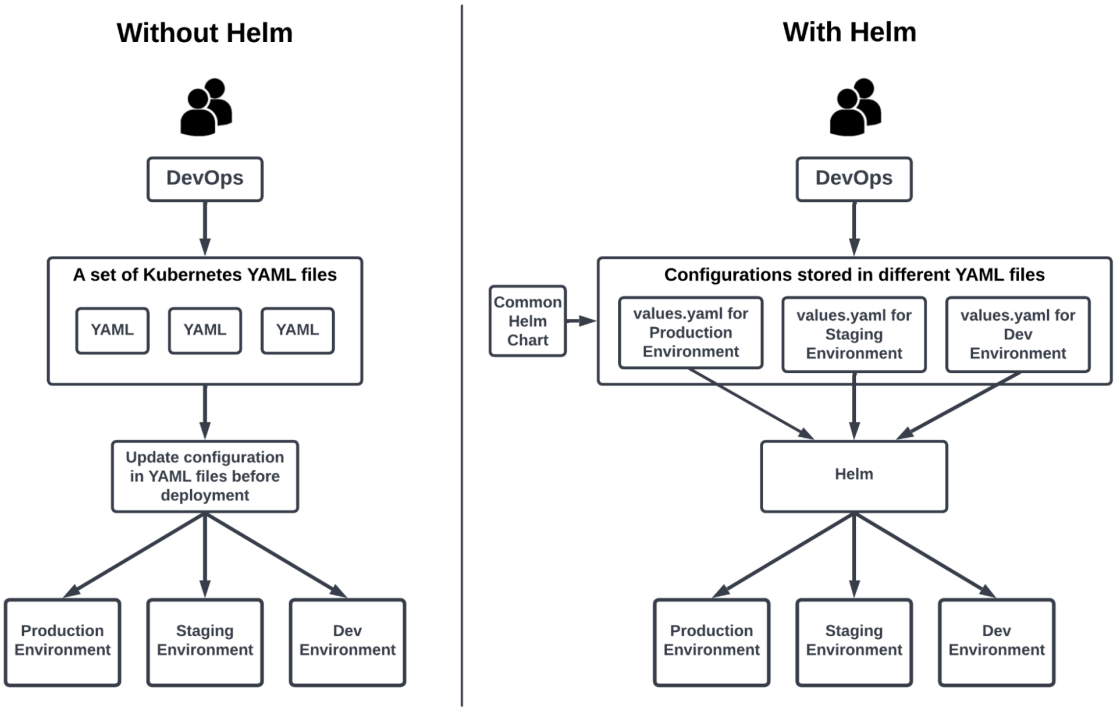
Deploy the same application across different environments (dev, staging, production) with a single command, while applying environment-specific configuration.




**2) Release Management and Version Control:**

Version Helm charts, allowing you to roll back to previous versions if necessary.

# Why Helm?



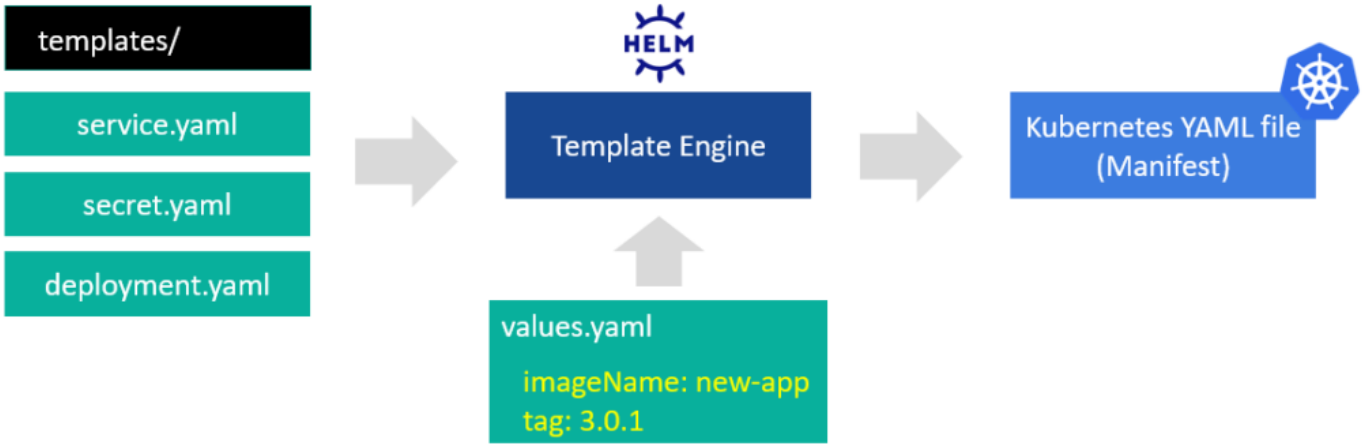



# Helm chart folder structure?

**$ helm create mychart**

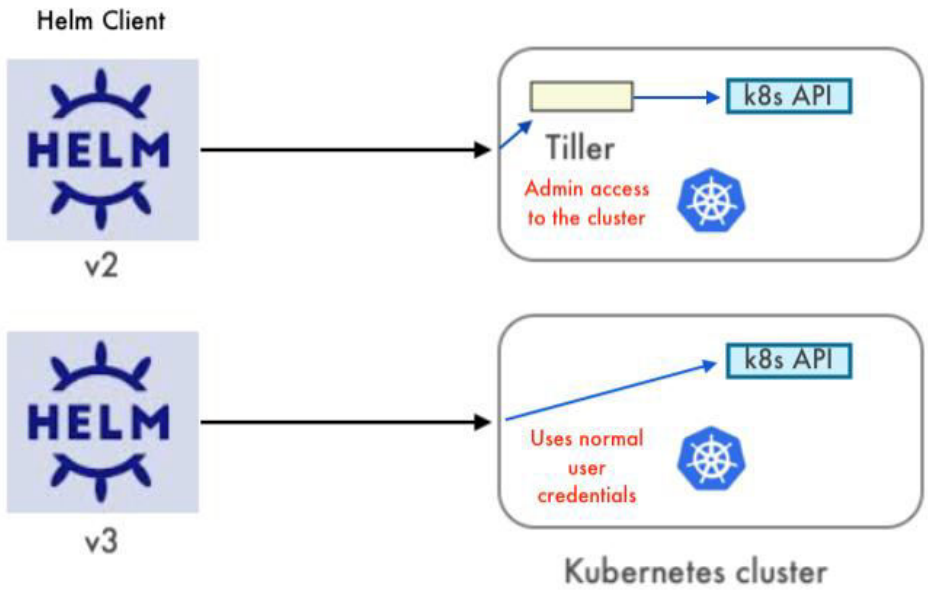
```
mychart
├── Chart.yaml          # Information about your chart, metadata, version and dependency
├── charts              # Charts that this chart depends on
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml         # The default values for your templates
```

# Helm – Templating Engine






# Helm 2.x vs 3.x?



# Configuration management with Helm Example


**1) Install Helm using Chocolatey on Windows:**

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

```
choco install kubernetes-helm
```

**2) Create a Helm Chart:**

```
helm create mysql-chart
```

**3) Add Helm Stable Repository:**

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

**4) Install MySQL Chart:**

- We can deploy MySQL using the official Bitnami Helm chart:

```
helm install mysql-chart-release1 bitnami/mysql –namespace mysql-helm –create-namespace
```

- This command deploys MySQL with default settings.
- Helm will create a Kubernetes deployment, service, and persistent volume claim for the database



# Configuration management with Helm Example


**5) Check the Release and Status**

```
helm list
helm status mysql-chart-release1
```

**6) Check All Kubernetes Objects Deployed by Helm for my-mysql**

```
helm get manifest mysql-chart-release1
```

**Deep Dive of folder structure of a helm chart:**

```
helm pull bitnami/mysql --untar
tree mysql/
```

# Configuration management with Helm Example


**7) Customize MySQL Configuration:**

- To customize the deployment, we create a values.yaml file with desired configuration (such as root password, database name, storage settings).

Ex: values.yaml

```
mysql:
  rootPassword: "root_password"
service:
  type: ClusterIP
persistence:
 enabled: true
 accessMode: ReadWriteOnce
 size: 1Gi
```

**8) Applying the Custom Configuration:**

```
helm install mysql-chart-release1 mysql-chart --values dev-values.yaml
helm install mysql-chart-release1 mysql-chart --values prod-values.yaml
```



# Traditional CD - Continuous Deployment


- Traditional CD automates the entire release process after code has passed the build and test phases.

- **Anatomy of a Deployment Pipeline:**
- A deployment pipeline automates the workflow from code commit to production.

- It includes stages like:

- **Commit Stage:** Running unit tests on the latest commit.
- **Automated Test Stage:** Running integration and end-to-end tests.
- **Staging Environment:** A replica of the production environment to test changes.
- **Production Deployment:** Final stage, deploying to live users.

# Example: Jenkins pipeline for a simple web app


```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'mvn clean install'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('Deploy to Staging') {
            steps {
                sh 'kubectl apply -f k8s/staging.yaml'
            }
        }
        stage('Deploy to Production') {
            steps {
                input message: 'Deploy to production?', ok: 'Deploy'
                sh 'kubectl apply -f k8s/production.yaml'
            }
        }
    }
}
```



# Jenkins pipeline using docker image


### 1) Pull the Jenkins Docker Image and Start Jenkins

docker pull jenkins/jenkins:lts
docker run -d --name jenkins_container -p 8080:8080 -p 50000:50000
8080 -> Jenkins UI
50000 -> Jenkins agent

### 2) Access Jenkins by going to http://localhost:8080 in your browser.

### 3) Jenkins will ask for an initial administrator password. You can find this in the Docker logs:

docker exec jenkins_container cat /var/jenkins_home/secrets/initialAdminPassword

### 4) Install Pipeline Plugins:

Go to Manage Jenkins → Manage Plugins.
In the Available tab, search for Pipeline and install the Pipeline plugin.

### 5) Set Up a Simple Jenkins Pipeline

a) Create a New Pipeline Job
b) Go to Jenkins UI → New Item.
c) Enter the job name (e.g., MyPipeline) and select Pipeline as the job type.
d) Click OK.
e) You can use a Jenkinsfile (Pipeline as Code) or define the pipeline directly in the Jenkins UI.

# Environment-based Release Patterns

- These patterns manage deployments across environments:

1) **Blue-Green Deployment:** Runs two environments, switching traffic between them.

- **Blue:** The current application version is running in this environment.
- **Green:** The new application version is running in this environment.
- Test the green environment.
- Once the green environment is ready, redirect live application traffic to the green environment.
- Deprecate the blue environment

2) **Canary Release:** Incrementally deploying to users.

- Releases new software to a small group of users before rolling it out to the entire user base.
- This gradual approach allows for more control over the release process and immediate feedback from users.
- If issues are detected, the rollout can be stopped without affecting the majority of users

3) **Feature Toggles:** Enabling/disabling features without redeploying code.



# Deployment Pipeline Practices

- The Best practices for deployment pipeline include:

1) **Automating Everything:**
    - Each step, from build to deploy, should be automated.
    - Human-free deployments aim for full automation.
    - Ensures no human intervention is required for successful deployments to production.

2) **Frequent Releases:** Encourages smaller, frequent deployments to reduce risk.

3) **Canary Releases:** Deploying to a subset of users before full rollout.

# Blue-Green Deployment in Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app-green
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: web-app-green-pod
    spec:
      containers:
        - name: web-container
          image: web-app:v2
---
apiVersion: v1
kind: Service
metadata:
  name: web-app-service
spec:
  selector:
    app: web-app-green
```



# GitOps CD

- GitOps CD leverages Git as the single source of truth for both infrastructure and applications, automating deployments by syncing Git changes with the desired state of Kubernetes clusters.

- **Operational Benefits of GitOps:**

1) **Version Control:** All deployment configurations are versioned in Git.

2) **Auditing and Compliance:** Git logs provide a detailed history of changes.

3) **Reproducibility:** Clusters can be easily recreated by applying Git configurations

**4) Declarative vs. Imperative Object Management in Kubernetes:**

1) Declarative Management: You declare the desired state in files (e.g., YAML), and Kubernetes ensures the actual state matches.

2) Imperative Management: You directly issue commands to change the current state.

Ex: kubectl run my-app --image=nginx --replicas=3

**5) Kubernetes with GitOps:** Tools like ArgoCD or Flux are used to sync the Kubernetes cluster state with the Git repository.

# What is IAC?

- Infrastructure as Code (IaC) is the process of managing and provisioning infrastructure through machine-readable scripts or code, instead of manual hardware configurations or interactive configuration tools.
- IaC automates the setup and management of infrastructure components such as servers, networks, and databases, reducing the need for manual intervention.
- Some popular IaC tools include Terraform , AWS CloudFormation, Ansible, Chef etc



# History of IAC

**Early 2000s:**

- The concept of configuration management tools began to emerge with the rise of virtualization.
- **Puppet** was developed in 2005 to automate the management of system configurations.
- **Chef** was created by Opscode in 2009, furthering the concept of IaC by allowing developers to define infrastructure using Ruby DSL (Domain-Specific Language).
- **Ansible** was released in 2012, emphasizing simplicity and ease of use in configuration management.
- It allowed users to describe their infrastructure in YAML files without requiring an agent on managed nodes.

# History of IAC

**2010-2020:**

- **Terraform**, developed by HashiCorp in **2014**, was introduced as a tool to provide a declarative way to manage infrastructure across multiple cloud providers.
- It allowed users to define their infrastructure using configuration files, making it easier to manage complex deployments.
- **AWS CloudFormation**, an existing service since 2011, gained significant traction as an IaC tool, enabling users to create and manage AWS resources using templates.

**2020s and beyond:**

- The adoption of IaC practices grew significantly, fueled by the rise of container orchestration tools (like Kubernetes) and serverless architectures, prompting further innovations in IaC tooling and practices.



# Types of IAC

**1) Declarative IaC:**

- Focuses on **defining** the **desired end state** of the infrastructure.
- The system figures out how to achieve that state.
- We describe what the infrastructure should look like.
- Ex: Terraform, AWS CloudFormation.

**2) Imperative IaC:**

- Focuses on the **steps needed to reach the desired state.**
- We **explicitly describe** how to achieve the target configuration.
- Ex: Ansible, Chef.

# Terraform vs Chef – provisioning a Nginx server on AWS EC2 Example

**Chef Recipe (webserver.rb):**

Chef doesn't directly provision infrastructure (like creating EC2
package "ngininstances)

```
# Update the package manager cache
execute "update_apt" do
  command "apt-get update"
  action :run
end

# Install the Nginx package
x" do
  action :install
end

# Ensure Nginx service is started and enabled to run on boot
service "nginx" do
  action [:enable, :start]
end
```

**Terraform Code (main.tf)**

```
# Define the provider (AWS)
provider "aws" {
  region = "us-east-1"
}

# Provision a basic EC2 instance
resource "aws_instance" "web" {
  ami           = "ami-0c55b159cbfafe1f0"  # Ubuntu AMI
  instance_type = "t2.micro"

  # User data script to install and start Nginx
  user_data = <<-EOF
              #!/bin/bash
              apt-get update
              apt-get install -y nginx
              systemctl start nginx
              EOF

  tags = {
    Name = "nginx-web-server"
  }
}
```



# What is the new Infra trend?

## Earlier

- Server creation and destruction took long time.
- Infrastructure provisioned by system admins and is manual.
- App servers were long lived and mutable (patching).

## Now

- Server creation and destruction happens within seconds.
- Infrastructure provisioned via code and is automated.
- App servers (containers now) are short-lived and immutable.

# Key IAC Principles

1) Idempotency:

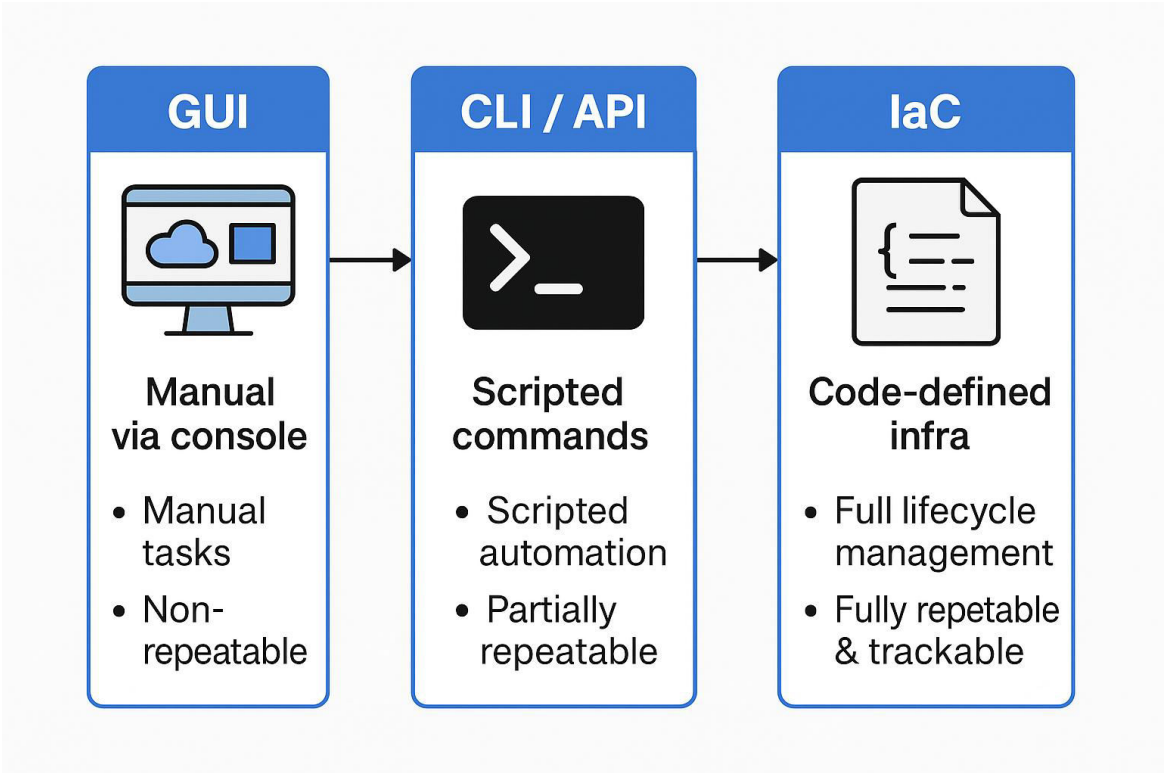
- Running the same code multiple times should yield the same result.
- This ensures consistency in infrastructure state.

2) Version Control:

- Like application source code, IaC scripts are managed in version control system like Git.
- This enables us to backup, track and revert changes.



# Provisioning cloud resources



# Key IAC Principles

**3) Automation:**

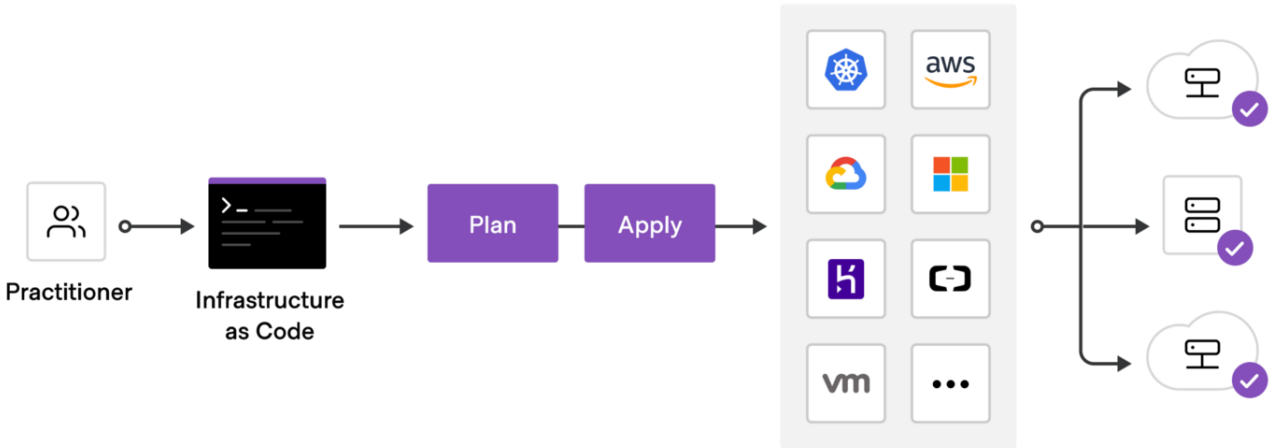
- IaC enables full automation of provisioning and scaling infrastructure using CI/CD pipelines.

**4) Consistency:**

- Infrastructure is consistent across environments Ex: dev, staging, production
- This eliminates configuration drift.



# IAC Workflow



# IAC Workflow – part1

**1) Write IaC Code:**

Define the desired infrastructure (Ex: compute instances, databases, networks) using tools like Terraform, AWS CloudFormation, or AWS SAM.

**2) Version Control:**

Store the IaC code in a Git repository for tracking changes, rollback, and collaboration.

**3) CI Pipeline:**

**a) Linting:** Ensure code follows best practices.

**b) Testing:** Test infrastructure in a sandbox environment to validate provisioning.



# IAC Workflow – part2

**4) CD Pipeline:**

**a) Review Changes:** IaC tools usually provide a plan of changes before applying them.

**b) Apply the Changes:** Execute the IaC code to create/update the infrastructure.

**5) Monitor and Audit:**

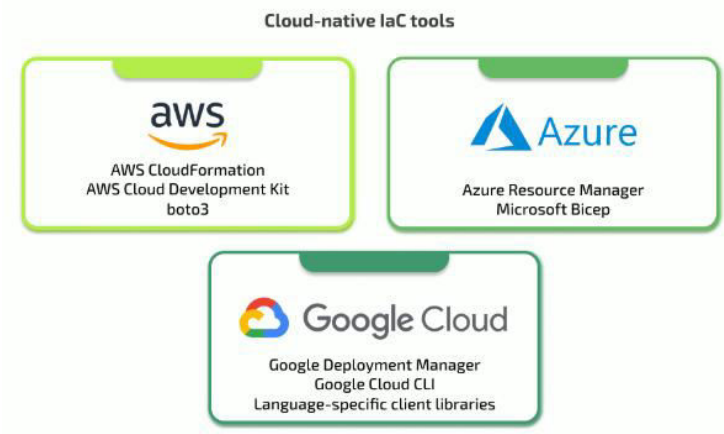
- Use monitoring tools to track infrastructure and detect drifts.

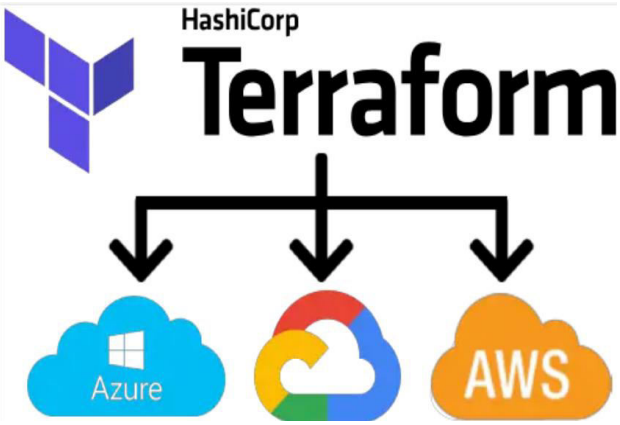
# Terraform

- Terraform is an **open-source Infrastructure as Code** (IaC) tool developed by **HashiCorp**.
- It enables users to define, provision, and manage infrastructure resources (such as servers, databases, networks etc) **across multiple cloud providers** and on-premises environments in a declarative configuration language called HashiCorp Configuration Language (HCL).
- The Configuration Files are written in a file with extension as .tf to indicate it is a terraform script.
- In a nutshell, terraform is a tool **for building, modifying and versioning Infrastructure safely and efficiently!**

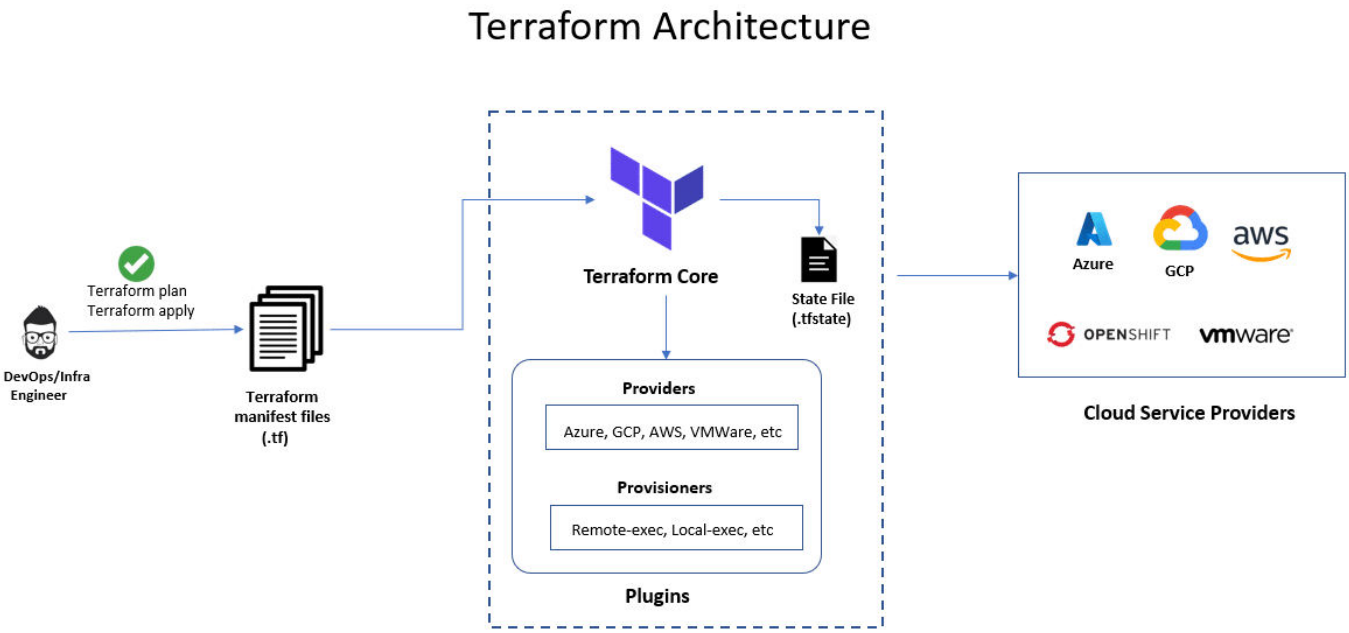


# Terraform is cloud agnostic



# Terraform Architecture






# Terraform – Setup

## Step 1: Install Terraform

1. Download the appropriate Terraform binary from https://www.terraform.io/, specifically https://developer.hashicorp.com/terraform/install?product_intent=terraform#windows for windows.
2. Extract the binary and add it to your system's PATH so that you can run Terraform commands from the terminal.
3. To confirm installation, run:
   - `terraform -v`

## Step 2: Create Terraform Configuration Files

- Create a file named main.tf

# Terraform – Providers

- In Terraform, providers are plugins that allow Terraform to interact with different infrastructure platforms and services.
- Providers are essential because they define what resources Terraform can manage and how it interacts with those resources, typically via APIs provided by cloud platforms or other service vendors.
- Each provider in Terraform has its own set of resource types and data sources it can manage.
- **Ex1**: The AWS provider manages resources like EC2 instances, S3 buckets, and IAM roles.
- **Ex2**: The Azure provider manages resources like Virtual Machines, Resource Groups, and Storage Accounts.



# Terraform – Provider Syntax and example

**syntax:**

```
provider “<cloud_provider_name>" {
# Configuration options
}
```

**example:**

```
provider "aws" {
  region = “ap-south-1"
}
```

# Terraform – Provider Source and version


**syntax:**

```
required_providers {
<cloud_provider_name> = {
source = "hashicorp/ <cloud_provider_name>"
version = "~<version>"
}
```

**example:**

```
required_providers {
  aws = {
        source = "hashicorp/aws"
        version = "~>3.0"
        }
}
```




# Terraform – Resource

- In Terraform, a resource is a fundamental component that represents a specific piece of infrastructure or service managed by a provider.
- Resources define what you want to create, modify, or delete in your infrastructure, such as a virtual machine, storage bucket, load balancer, or database.
- Each resource has 3 things:

1) **Resource Type:** Defined by the provider

Ex: aws_instance, google_compute_instance.

2) **Resource Name:** A unique name you assign within your configuration to identify the resource.

3) **Arguments and Attributes:** Parameters and settings that specify properties for the resource, such as the machine type, size, or region.

# Terraform – Resource Syntax and example


**syntax:**

```
resource "<provider>_<resource_type>" "<resource_name>" {
  # Arguments for the resource configuration
  attribute1 = "value1"
  attribute2 = "value2"
  # ...additional configuration
}
```

**example:**

```
resource "aws_instance" "web_server" # resource name in tf
{
  ami  = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  tags = {
     Name = "MyWebServer"  # resource name in aws
  }
}
```



# Terraform – Workflow and Commands


**1) Initialize: terraform init**

- Run terraform init to set up plugins required to interact with providers (e.g., AWS, Azure).

**2) Lint: terraform fmt**

- Check code with terraform fmt for formatting issues

**3) Plan: terraform plan**

- Run terraform plan to see what changes Terraform will make to achieve the desired state.
- This is like a dry run and shows what resources will be created, modified, or destroyed.

**4) Apply: terraform apply**

Run terraform apply to provision the defined resources.

**5) Destroy: terraform destroy**

- Run terraform destroy to remove all resources defined in the configuration file, which is helpful for cleaning up temporary or test environments.

# Terraform – AWS example

- Let us try to create, update, and destroy an AWS S3 bucket using Terraform.
- Can you recall the high level steps to be done?



# Terraform – AWS S3 example

1) Create a main.tf

```
# Configure the AWS provider
provider "aws" {
  region = "us-west-2"  # Specify the AWS region
}
# Create an S3 bucket
resource "aws_s3_bucket" "ssd-s3-bucket-test1" {
  bucket = "ssd-s3-bucket-test1"
}
```

2) terraform fmt
3) terraform init
4) terraform plan
5) terraform apply
6) terraform destroy

# Multi Cloud IaC – Example

- Let us create the following infrastructure using terraform:

1) A Serverless Lambda Function on GCP

2) A Virtual Machine on Azure

3) Integrate Lambda (GCP) and VM (Azure)



# 1) A Serverless Lambda Function on GCP

- Google Cloud offers Cloud Functions for serverless functions, which are similar to AWS Lambda.
- Official documentation of google terraform providers – https://github.com/hashicorp/terraform-provider-google
- https://registry.terraform.io/providers/hashicorp/google/latest/docs/guides/provider_reference

**High Level Steps:**

1) Set up GCP provider
2) Storage Bucket Creation
3) Upload Source Code
4) Deploy Cloud Function
5) HTTP Trigger Setup

**Step 1: Set up GCP provider:**

```
provider "google" {
  project = "project-id"
  region  = "us-east1"
}
```

# 1) A Serverless Lambda Function on GCP

**main.py**

```
from fastapi import FastAPI
from mangum import Mangum

app = FastAPI()
@app.get("/")
def greet():
    return {"message": "Hello, FastAPI on Google Cloud Functions!"}

# ASGI-compatible entry point for Google Cloud Functions
handler = Mangum(app)
```

**requirements.txt:**

```
fastapi==0.95.1
mangum==0.14.0
uvicorn==0.22.0
functions-framework==3.3.0
```



# 1) A Serverless Lambda Function on GCP

**Step 2: Create a Google Cloud Storage bucket to store the source code of the Cloud Function:**

```
resource "google_storage_bucket" "ssd_gcp_bucket" {
  name     = "function-source-bucket"
  location = "US"
}
```

This resource creates a Google Cloud Storage bucket named ssd_gcp_bucket to store the source code of the Cloud Function.

**Step 3: Create a Google Cloud Storage bucket object for the source code zip of the Cloud Function:**

```
resource "google_storage_bucket_object" "ssd_gcp_lambda_code_object" {
  bucket = "source.zip"
  source = "path_to_source.zip"
}
```

This resource uploads source.zip into the bucket ssd_gcp_bucket.

BITS Pilani, Pilani Campus

# 1) A Serverless Lambda Function on GCP

- Mangum converts the FastAPI app into an ASGI-compatible handler that Google Cloud Functions can invoke.
- ASGI frameworks like FastAPI, Starlette, and Django with Channels are asynchronous and optimized for modern web applications.
- However, platforms like AWS Lambda and Google Cloud Functions expect specific synchronous HTTP interfaces.
- Mangum bridges this gap, enabling ASGI applications to run seamlessly in these serverless environments.
- Create a source.zip that contains

├── main.py # FastAPI application code
├── requirements.txt # Dependencies



# 1) A Serverless Lambda Function on GCP

**Step 4: Create a Google Cloud Function:**

```
resource "google_cloudfunctions_function" "ssd_gcp_lambda_function" {
  name        = " ssd_gcp_lambda_function "
  runtime     = "python311"
  entry_point = "app" # fastapi entry point

  source_archive_bucket = google_storage_bucket.ssd_gcp_bucket
  source_archive_object = google_storage_bucket_object. ssd_gcp_lambda_code_object.bucket

  trigger_http = true
}
```

- A public HTTPS endpoint is automatically generated for the function

Ex: https://REGION-PROJECT_ID.cloudfunctions.net/ssd_gcp_lambda_function

BITS Pilani, Pilani Campus

# 2) A Virtual Machine on Azure

- Azure provides the azurerm provider for provisioning virtual machines.
- Link: https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs

**Step 1: Set up Azure provider**

```
provider "azurerm" {
  features {}
}
```

**Step 2: Create the Virtual Machine**

```
resource "azurerm_linux_virtual_machine" "ssd_ma_vm" {
name = " ssd_ma_vm "
location = "East US"
...
admin_username  = "adminuser"
admin_password  = "AdminPassword123!"
network_interface_ids = [azurerm_network_interface.nic.id]}}
```



# 3) Integrate Lambda (GCP) and VM (Azure)

- On the Azure VM, we can write a script to make an HTTP request to the Cloud Function URL.

**Step1: Create a script to invoke google cloud function:**

```
invoker.sh
#!/bin/bash
curl -X GET "https://REGION-PROJECT_ID.cloudfunctions.net/ssd_gcp_lambda_function"
```

- We can provision this script on the VM using Terraform's azurerm_linux_virtual_machine_extension to run the script upon VM creation.

**Step2: Create a script to invoke google cloud function:**

```
resource "azurerm_linux_virtual_machine_extension" "init_script" {
    name = "init-script"
    virtual_machine_id   = azurerm_linux_virtual_machine.ssd_ma_vm.id
    publisher  = "Microsoft.Azure.Extensions"
    type       = "CustomScript"
    type_handler_version = "2.0"
    script = "./invoker.sh"
 }
```

- This will run the script on the Azure VM after it is provisioned, which will call the GCP Cloud Function.

# Terraform vs Ansible vs Helm

1. Terraform -> IAC

➢ Provision and manage cloud infrastructure as code.

Ex: Spin up Infra (EKS/GKE, VPCs, Load Balancers, etc.)

2. Ansible -> CAC

➢ Automate configuration and software setup across systems.

Ex: Provision Nodes (Install Docker, Kubeadm, Helm CLI, etc.)
apt install helm

3. Helm -> App packaging and deployment

➢ Deploy and manage Kubernetes applications using packaged charts.

Ex: Deploy App into K8s (e.g., helm install myapp ./chart)




# Terraform – State and locking

**State:**

➢ Terraform maintains a file called terraform.tfstate, which acts as the source of truth for:
  ✓ What resources have been created
  ✓ Their current configurations and metadata
  Ex: instance IDs, Ips
  ✓ Dependencies between resources

➢ **Without the state file, Terraform has no idea what's already deployed!**

**Locking:**

➢ Imagine two developers running terraform apply at the same time — they could corrupt the state or race on creating resources.

➢ State locking prevents that.State locking ensures only one process can modify the state at a time.

# Terraform – State and locking remote backend

- If we are working in a team or using CI/CD,
- we don't want our terraform.tfstate stored locally.

Instead:

We shall use s3 bucket to store our state file.
Ex: dev, staging, prod
dev/tf.state
Prod/tf.state

| Feature | Why use it? |
|---|---|
| **S3 bucket** | Centralized, versioned, reliable state storage |
| **DynamoDB table** | Enables **state locking** to prevent race conditions |



# Terraform – State and locking – s3 remote backend example

**Create a s3 bucket as follows:**

```
terraform {
  backend "s3" {
    bucket         = "my-terraform-state-bucket"
    key            = "prod/terraform.tfstate"
    region         = "us-east-1"
    dynamodb_table = "terraform-locks"
    encrypt        = true
  }
}
```

Create a dynamodb table as follows:

| Table Name | Primary Key |
|---|---|
| terraform-locks | LockID (string) |


```
{
  "ID": "prod/terraform.tfstate",
  "Operation": "OperationTypeApply",
  "Info": "Lock for terraform apply",
  "Who": "sushanth@laptop",
  "Version": "1.7.0",
  "Created": "2025-04-27T11:45:12.320Z",
  "Path": "prod/terraform.tfstate"
}
```




# Serverless IaC – Tools

Several tools are specifically designed for Serverless IaC:

1) **AWS CloudFormation:**
- AWS's native IaC tool.
- Can be used to manage AWS infrastructure, including serverless resources like Lambda and API Gateway.

2) **AWS Serverless Application Model (SAM):**
- Built on CloudFormation, SAM simplifies the definition of serverless resources like Lambda functions and API Gateway by using a shorthand syntax.

3) **Terraform:**
- Supports managing AWS Lambda functions and other serverless components.

4) **Serverless Framework:** An open-source framework that simplifies the deployment of serverless applications across multiple cloud providers (AWS, Azure, Google Cloud).

# Automated deployment using AWS Serverless Application Model (SAM) example

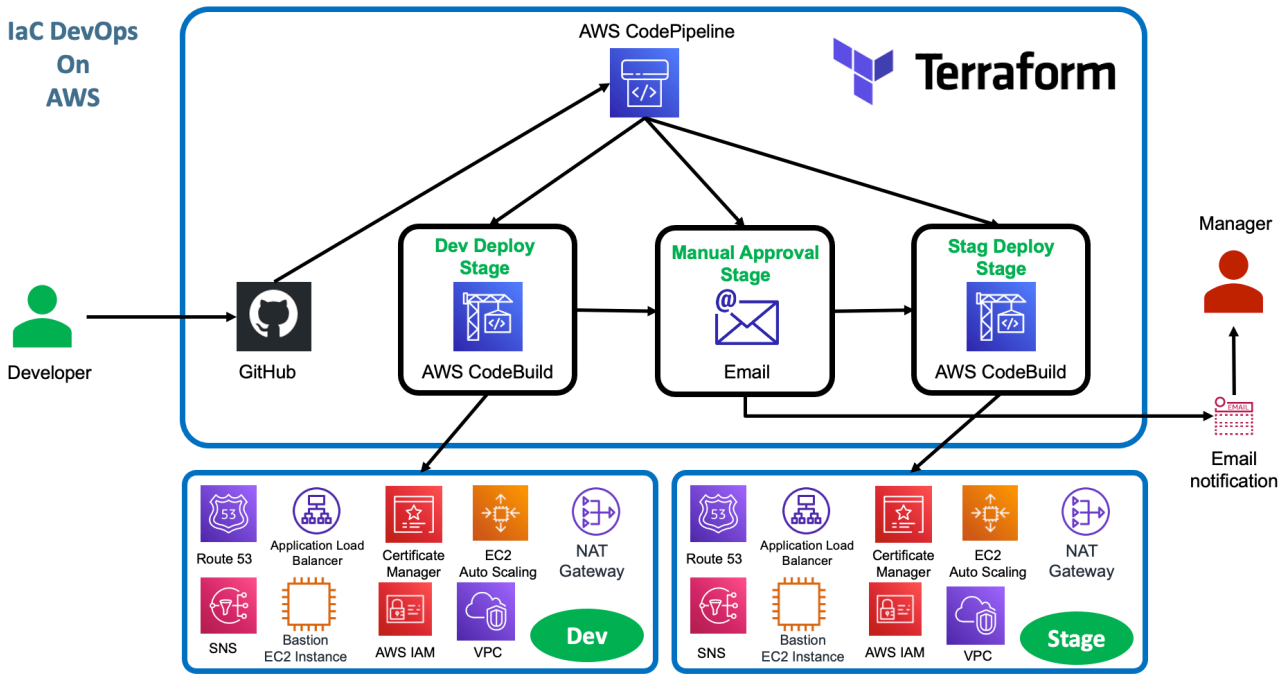
Create a template.yaml

```
AWSTemplateFormatVersion: '2024-10-16'
Transform: AWS::Serverless-2016-10-31
Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: app.lambdaHandler
      Runtime: python3.8
      CodeUri: ./
      Events:
        ApiEvent:
          Type: Api
          Properties:
            Path: /greet #endpoint
            Method: GET
```

```
sam build
sam deploy –guided
curl https://<api-id>.execute-api.<region>.amazonaws.com/Prod/greet
```



# Release pipeline for Serverless deployment

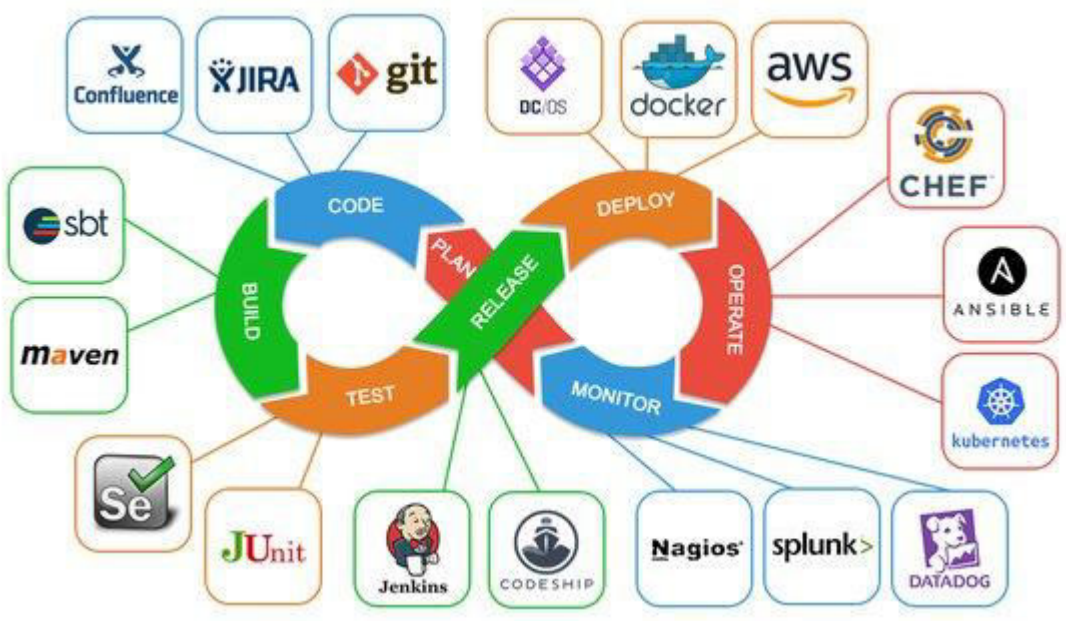


# Release pipeline for Serverless deployment – AWS Example

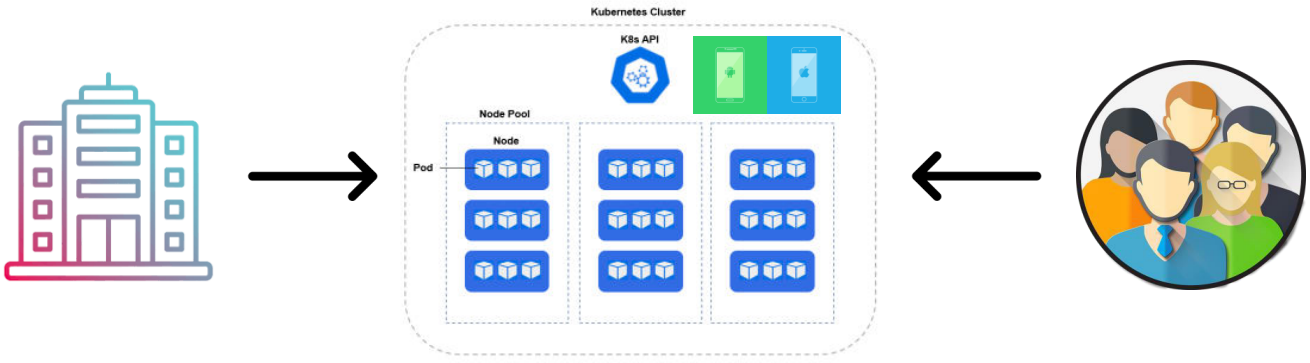
- Create an AWS and Terraform Backend
  - Create a DynamoDB table
  - Create an S3 bucket
- Create Repository and AWS Terraform files
  - Create AWS Codecommit
  - Create Terraform files
- Automate CI/CD deployment to AWS with Terraform
  - Create the CI/CD Pipeline using AWS CodePipeline
  - Create a CodeBuild Role using AWS IAM.
  - Deploy resources with CodePipeline
- More details can be found
- https://spacelift.io/blog/terraform-tutorial



# Devops lifecycle – Observability and Continuous Monitoring



# Need for Continuous Monitoring



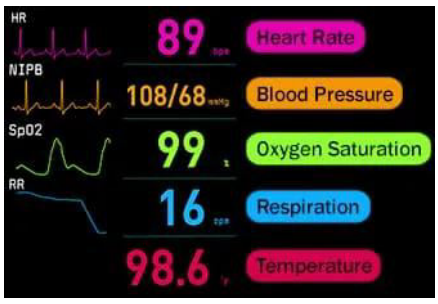

Organization App deployed on k8s Customers

- Would you find the live issues first or your Customers?
- As systems scale, complexity increases, making it harder to identify and diagnose issues!



# Observability and Continuous Monitoring – Real World Example

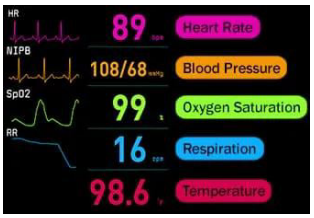
- Electronic vital sign monitors have been common in hospitals for more than 40 years
- Small sensors attached to your body carry information to the monitor.
- Some sensors are patches that stick to your skin, while others may be clipped on one of your fingers
- If one of your vital signs rises or falls outside healthy levels, the monitor will sound a warning. This usually involves a beeping noise and a flashing color. Many will highlight the problem reading in some way.



# What is Observability and Continuous Monitoring

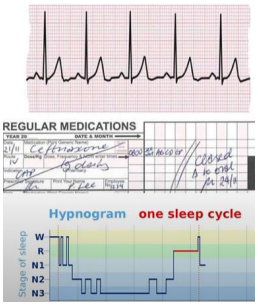
## Continuous Monitoring

- These signals are called metrics and monitoring system help in tracking them and alert the hospital staff whenever some metric is outside the normal range.
- Continuous monitoring is the ongoing process of collecting, analyzing, and alerting on data to ensure systems' health and reliability in real-time.




## Observability

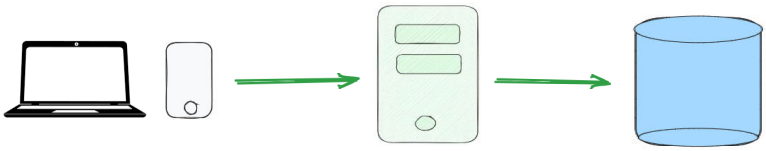
- Doctor rushes to the patient where the device is beeping and asks nurse to provide health records to further diagnose.
- Observability in DevOps is about understanding the internal state of systems by gathering telemetry data like metrics, logs, and traces.






# Observability and Continuous Monitoring – Real World Example2

- Lets say you are using a banking application for checking your account balance.
- You get the result after 10 mins, would that be cool?
- You would instead stop using such app and give a bad rating and switch to another app!
- Can banks identify this? – need to deploy Continuous Monitoring tools!
- Can they know why the query is running slow? - need observability tools!



# Observability and Continuous Monitoring

- Continuous monitoring is the ongoing process of collecting, analyzing, and alerting on data to ensure systems' health and reliability in real-time.
- Observability in DevOps is about understanding the internal state of systems by gathering metrics that reflect the same.




# Three Pillars of Observability

- The three main pillars of observability are Logs, Metrics, and Traces.
- Each pillar provides a different type of insight into the system's health and performance:

**1) Logs:**

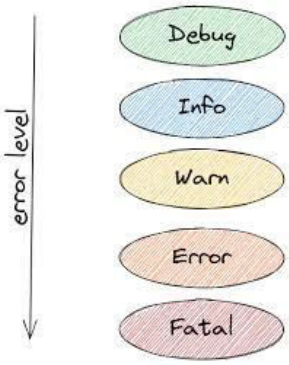
- Logs are timestamped records of discrete events that occur within an application or system.
- They provide detailed context about what happened at a particular time.
- Logs are essential for understanding specific events and errors in detail, especially when debugging or conducting root-cause analysis.
- Ex1: An error log detailing a failed transaction.
- Ex2: An informational log that records a user's login activity.

# Types and Levels of Logs

1) **System Logs**: OS-level events like kernel messages.

2) **Application Logs**: Events specific to application functions.

3) **Audit Logs**: Track access to resources and user actions for compliance.






# Three Pillars of Observability

**2) Metrics**

- Metrics are numerical representations of data measured over intervals of time.
- They typically represent system states or resource utilization, like CPU load, memory usage, or request rate.
- Metrics provide real-time and historical data to identify trends, detect anomalies, and monitor overall system health and performance.
- Ex1: CPU usage percentage over time.
- Ex2: Number of active users or HTTP requests per second.
- 100 active users 2:00PM
- 200 active users 3:00PM
- Time Series Data

# Metrics and Its Types

- Metrics measure system performance and are critical for observability. Common metric types include:

1) **System Metrics**: CPU, memory, network usage.

2) **Application Metrics**: Request count, latency, errors.

3) **Business Metrics**: Conversion rates, revenue, etc.

- What metrics should we use?



# Classification of Metrics

- Metrics are classified as **counters**, **gauges**, **histograms**, and **summaries**:

1) **Counters**: Monotonically increasing values, like a request count.

2) **Gauges**: Variable metrics that can increase or decrease, such as memory usage.

3) **Histograms**: Represent the distribution of values, such as response times.

4) **Summaries**: Similar to histograms but can be pre-aggregated.

# Three Pillars of Observability

**3) Traces**

- Traces are representations of the flow of a request as it traverses through various services in a distributed system.
- They record details of each operation in a transaction.
- Traces help visualize the path of a request through multiple services, making it easier to detect latency, failures, and bottlenecks across the entire request flow.
- Ex1: A trace showing the execution time for each microservice in a single user transaction.
- Ex2: A trace highlighting where a request failed in a service chain.



# Components of a Monitoring Stack

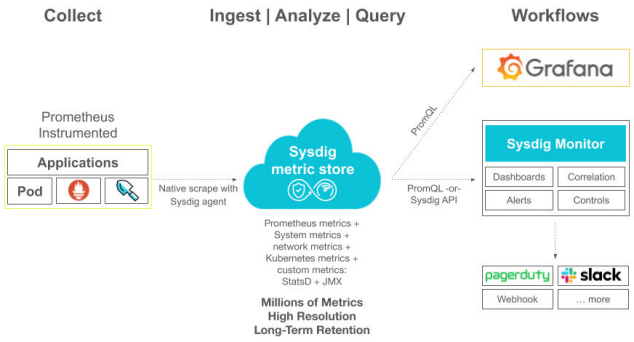
- An effective monitoring stack typically consists of:

**1) Data Collection**: Tools like Prometheus for metrics, Fluentd for logs.

**2) Storage**: Long-term data storage, such as InfluxDB or TimescaleDB, for historical metrics.

**3) Visualization**: Tools like Grafana provide visual dashboards for real-time monitoring.

**4) Alerting**: Systems like Alertmanager and PagerDuty notify relevant teams when certain thresholds are breached.



# Setting Up Alerts and Alert Management

- Alerts notify teams when system behavior deviates from expected parameters.
- Effective alerting helps prevent alert fatigue and ensures critical issues are noticed promptly.
- **Alert Types:**

1) **Threshold-based:** Triggered when metrics exceed set limits.

2) **Anomaly-based:** Triggered when unusual patterns are detected.



# 4 Golden Signals of Continuous Monitoring

- Google came up with 4 **Golden Signals** that are a standard approach to monitoring the health of distributed systems and microservices.


- These Golden Signals help focus on critical metrics to prevent service degradation, ensuring a more systematic response to issues.

# 4 Golden Signals of Continuous Monitoring

1) **Latency**: Time between request sent by client and response sent by server.

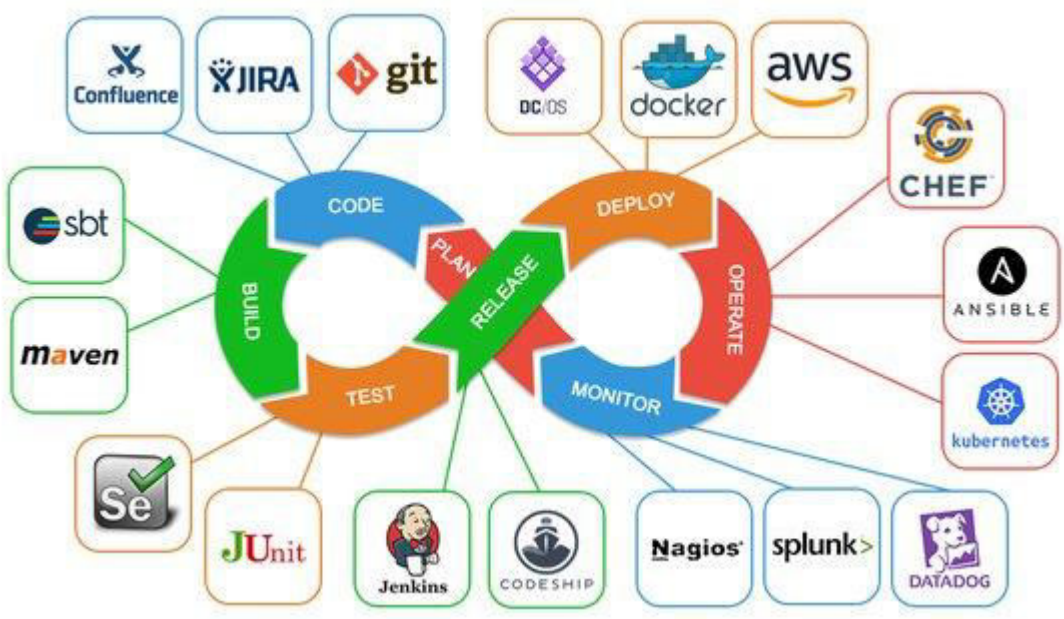
2) **Traffic**: The demand on the system, often measured in requests per second.

3) **Errors**: The rate of requests that fail.
Ex: keep track of http status codes – 500.

4) **Saturation**: The system's resource capacity like CPU, RAM, Disk usage. High saturation indicates that resources may be nearing full usage.




# Devops lifecycle – Observability and Continuous Monitoring



# Tools for Observability and Monitoring

- **AWS CloudWatch**:
- Monitors AWS resources (e.g., EC2, S3) and custom metrics.
- Logs events and alarms for infrastructure health monitoring.
- **Prometheus**:
- Open-source monitoring and alerting toolkit for collecting metrics.
- It scrapes metrics from instrumented jobs and stores them for analysis.
- **Grafana**:
- Visualization tool that integrates with Prometheus and other data sources.
- Allows users to create dashboards for real-time insights into system metrics.



# Prometheus Setup

- Download minikube for windows from https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download
- Start the minikube cluster
  - minikube start --driver=docker
- Install helm using choco
  - choco install kubernetes-helm
- Add helm chart for prometheus
  - helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
  - helm repo update
- Create a namespace in Kubernetes for Prometheus services:
  - kubectl create namespace prometheus
- Installing Prometheus using Helm-charts
  - helm install prometheus prometheus-community/prometheus -n prometheus

# Prometheus Setup

- List all pods running in Prometheus namespace
  - kubectl get pods -n prometheus
- Forwading 80 to 9090 host port (laptop)
  - kubectl port-forward --address 0.0.0.0 svc/prometheus-server -n prometheus 9090:80
  - Then open your browser and visit: http://localhost:9090
- Configure Prometheus via the Helm values
  - helm get values prometheus -n Prometheus
  - helm show values prometheus-community/prometheus > prometheus-values.yaml
  - Add your custom scrape job under the prometheus.prometheusSpec.scrapeConfigs section:

Ex:

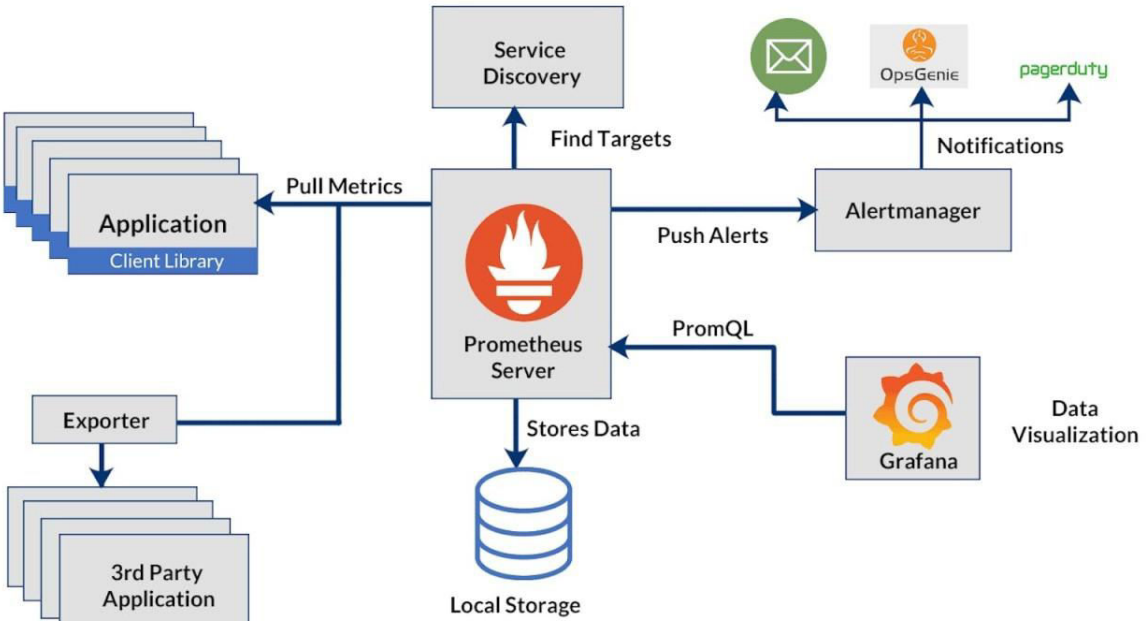
```
prometheus:
  prometheusSpec:
    scrapeConfigs:
      - job_name: 'long_running_job'
        static_configs:
          - targets: ['localhost:8000']
```

  - Apply the modified configuration:

```
helm upgrade prometheus prometheus-community/prometheus -n prometheus -f prometheus-values.yaml
```

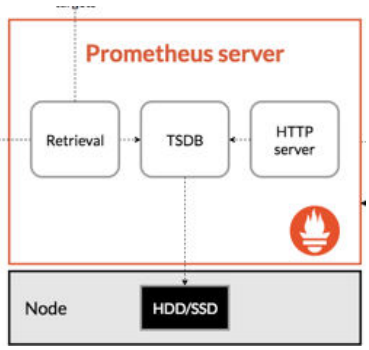


# Prometheus Architecture - 1



# Prometheus Service types

**1)Prometheus Server**




- Prometheus's direct scraping service metrics (Ex: CPU, memory, and disk usage) allows teams to respond quickly to potential issues.
- Ideal for Long-lived, persistent services.
- Ex: long running batch jobs.

**2) Instrumentation:** process of adding code to your application to expose metrics in a format Prometheus can scrape and monitor.
Ex: custom applications need to be instrumented to create metrics.

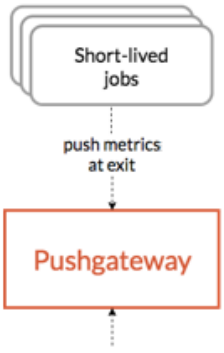
To use Prometheus with Python, the most popular library is prometheus-client.

- pip install prometheus-client



# Prometheus Service types

**3) Pushgateway:**

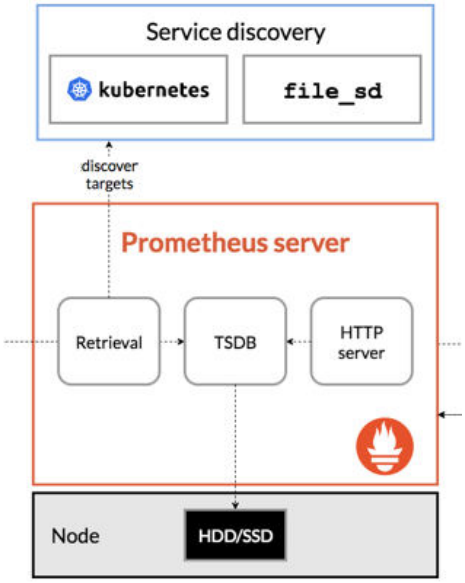



- Allows ephemeral or batch jobs to push metrics that would otherwise be missed between scrapes.
- Ideal for short-lived services, batch jobs, or any workload where instances exist only briefly, as these types of jobs might end before Prometheus can scrape them.
- **Ex:** short lived status mail job(e.g., build time, status) to the Pushgateway when triggered by CI/CD tools like Jenkins.
- Prometheus server pulling metric -> 5 seconds
- T0 -> Prometheus scraps , finds nothing
- T1 -> Time Jenkins job – 2 seconds
- T2 -> job ends -> push using pushgateway
- cant be tracked by Prometheus!

# Prometheus Service types

### 4) Service Discovery

- In modern architectures like microservices and Kubernetes, services are constantly scaling up and down, moving across nodes, or being replaced, making static configuration impractical.
- Service Discovery automates the process of tracking these changes, ensuring that Prometheus always has an updated list of targets (services and instances) to monitor without manual intervention.
- kube-state-metrics service is the one that exposes cluster-level metrics like deployments, nodes, and pods in a format Prometheus can scrape.

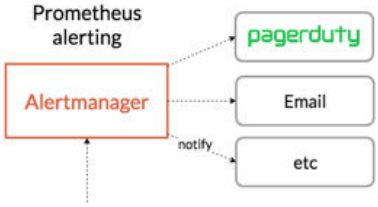





# Prometheus Service types

### 5) Alertmanager

- Timely notifications to the DevOps team can ensure faster response, reducing risks to service availability and data safety.
- Ex: For healthcare applications, Alertmanager sends alerts for high-priority incidents, like API response errors in patient-data handling services.




### 6) Federation

- Each region has a dedicated Prometheus instance, and Federation consolidates essential metrics (e.g., response time, error rates) into a global view.
- Ex: Aggregation in Large Enterprise Environments where there are Distributed services across regions.

# Prometheus Service types

## 7) Exporters

- Not all applications or systems expose metrics in the format that Prometheus requires.
- Exporters serve as a bridge, translating native system metrics into a format that Prometheus understands.
- Examples of Common Prometheus Exporters:
- **a) Node Exporter:** Collects and exposes system metrics (CPU, memory, disk I/O) from Linux and other Unix-based systems.
- **b) Database Exporters:** Specialized exporters for databases, like MySQL Exporter , PostgreSQL Exporter, MongoDB Exporter etc.

Ex: By default, Nginx exposes its metrics through the stub_status endpoint. This endpoint provides basic metrics such as: Number of active connections, Total number of requests

- The raw output from Nginx's stub_status page typically looks like this:

Active connections: 1

- The Nginx Prometheus Exporter acts as a middleware between Nginx's raw metrics and Prometheus.
- It scrapes the raw metrics from Nginx's stub_status endpoint and reformats them into a Prometheus-compatible format.

nginx_connections_active 1



# Prometheus Service types

## 7) Exporters – Example

| Raw Data (Before) | Transformed Data (After) |
|---|---|
| Active connections: 1 | # HELP nginx_connections_active Current number of active client connections.<br># TYPE nginx_connections_active gauge<br>nginx_connections_active 1 |
| server accepts handled requests 1 2 3 | # HELP nginx_connections_accepted Total accepted client connections.<br># TYPE nginx_connections_accepted counter<br>nginx_connections_accepted 1<br><br># HELP nginx_connections_handled Total handled client connections.<br># TYPE nginx_connections_handled counter<br>nginx_connections_handled 2<br><br># HELP nginx_http_requests_total Total number of HTTP requests.<br># TYPE nginx_http_requests_total counter<br>nginx_http_requests_total 3 |

BITS Pilani, Pilani Campus

# Prometheus Service types

**8) Prometheus Web UI – more details:**

- The Prometheus web UI is a built-in user interface that allows users to explore and interact with the metrics data collected by Prometheus.
- It offers several functionalities, including running queries, visualizing metrics, checking alert statuses, and monitoring the health of Prometheus components
- Use the **Graph** tab to visualize metrics.
- Use the **Targets** tab to see which endpoints Prometheus scrapes.
- Use the **Status → TSDB Status** to view:
  - Number of time series.
  - Disk space used.
  - Head block and chunk details.
  - Ingestion rate.



# Prometheus Service types

**8) Prometheus Web UI – more details:**

- **Query Time Series Data with PromQL:** In the Prometheus UI (under the **Graph** tab), we can run queries like:

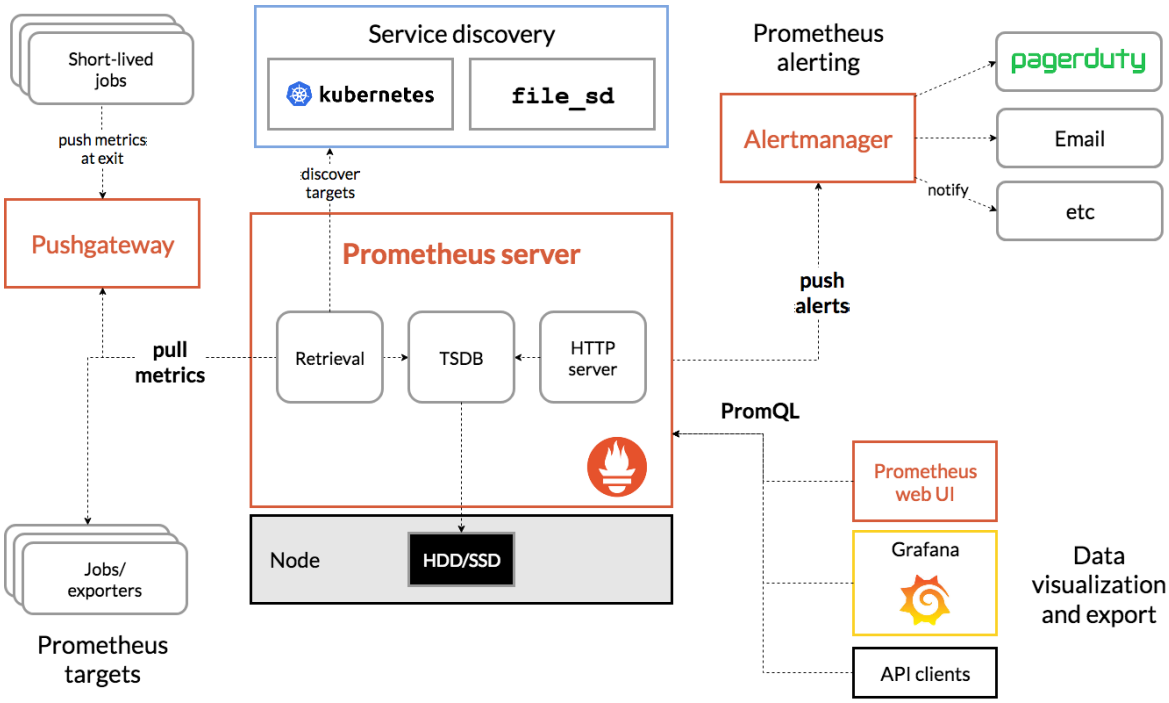
  a) **up:** This is a built-in metric in Prometheus that tells whether the target is up (1) or down (0).

  b) **node_cpu_seconds_total:** This metric counts the total cumulative time the CPU spends in different modes (user, system, idle, etc.) per core.

  c) **rate(http_requests_total[5m]):** This is using method chaining and calculates the per-second average rate of increase of http_requests_total over the past 5 minutes.

**BITS** Pilani, Pilani Campus

# Prometheus Architecture - 2






# Prometheus Client Setup in Python

- Prometheus server can scrape service metrics exposed on /metrics endpoint.
- Custom applications need to be instrumented to create metrics.
- To use Prometheus with Python, the most popular library is prometheus-client.
  - pip install prometheus_client

**Metric Types:**

- In the prometheus_client Python library, the primary metric types are Counter, Histogram, and Gauge, each serving different purposes for monitoring and instrumentation.

# Prometheus Client – Metric Types – Counter

1. **Counter:**

- A Counter is a cumulative metric that only increases and doesn't decrease (except when reset to zero).
- It is used to count discrete events like number of requests served, tasks completed, or errors occurred etc.

- **syntax:**

```
Counter(metric name, help text, labelnames=(), namespace='', subsystem='', unit='')
```

- **Ex:**

```
from prometheus_client import Counter

# Create a counter to track requests
request_counter = Counter('http_requests_total', 'Total number of HTTP requests')

# Increment the counter by 1
request_counter.inc()

# Increment the counter by a specific value
request_counter.inc(5)
```



# Prometheus Client – Metric Types – Histogram

2. **Histogram:**

- A Histogram measures the distribution of a set of continuous values, such as request durations or response sizes.
- It divides these values into configurable buckets and tracks counts for each bucket.
- Includes default buckets, but custom buckets can be defined.

- **syntax:**

```
Histogram(metric name, help text, labelnames=(), namespace='', subsystem='', unit='', buckets=(0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5, 10))
```

- **Ex:**

```
from prometheus_client import Histogram

request_duration = Histogram('http_request_duration_seconds', 'Histogram of HTTP request durations in seconds', buckets=(0.1, 0.5, 1.0, 2.5, 5.0))

# Record a duration of 1.5 seconds for a GET request to /home endpoint
request_duration.labels(method='GET', endpoint='/home')
request_duration.observe(1.5)
```

# Prometheus Client – Metric Types – Gauge


**3. Gauge:**

- A Gauge is a metric that represents a single numerical value that can arbitrarily go up or down.
- Can be used to track values that can increase and decrease, such as current memory usage, number of active threads, or CPU utilization.

- **syntax:**

```
Gauge(metric name, help text, labelnames=(), namespace='', subsystem='', unit)
```

- **Ex:**

```
from prometheus_client import Gauge

# Create a gauge to track memory usage
memory_usage = Gauge('memory_usage_bytes', 'Current memory usage in bytes')

# Set the gauge to a specific value
memory_usage.set(12345678)

# Increment the gauge
memory_usage.inc(5000)

# Decrement the gauge
memory_usage.dec(2000)
```



# Instrumentation Example – Prometheus Client


**1) Create a new python file named main.py and define a fastapi app and metrics**

```
from prometheus_client import Counter, Gauge, Histogram, generate_latest
from fastapi import FastAPI

app = FastAPI()

# metrics definitions

# Traffic (request count)
REQUEST_COUNT = Counter("fastapi_request_count", "Total number of requests")

# Latency (request latency)
REQUEST_LATENCY = Histogram("fastapi_request_latency_seconds", "Request latency")

# Errors (count of error responses)
ERROR_COUNT = Counter("fastapi_error_count", "Total number of error responses")

# Saturation (e.g., CPU utilization as a Gauge)
CPU_UTILIZATION = Gauge("fastapi_cpu_utilization", "CPU utilization in percentage")
```

# Instrumentation Example – Prometheus Client


**2) Define Middleware to collect Prometheus metrics for each request.**

- Add prometheus_middleware to process metrics for every request.
- This ensures metrics are collected regardless of which endpoint is accessed.

```
@app.middleware("http")
async def prometheus_middleware(request: Request, call_next):

    # Start timer
    start_time = time.time()

    # Process the request
    response = await call_next(request)

    # Update metrics
    REQUEST_COUNT.inc()  # Increment request count
    if response.status_code >= 500:
        ERROR_COUNT.inc()  # Increment error count for 5xx responses

    latency = time.time() - start_time

    REQUEST_LATENCY.observe(latency)  # Record latency
    CPU_UTILIZATION.set(psutil.cpu_percent(interval=None)) # Update CPU utilization

    return response
```



# Instrumentation Example – Prometheus Client


**3) Define metrics end point which is by default: /metrics**

```
@app.get("/metrics")
async def metrics():

    return Response(content=generate_latest(), media_type="text/plain")
```

**4) Create Prometheus's configuration file (prometheus.yml)**

```
global:
    scrape_interval: 5s  # How often to scrape targets by default

scrape_configs:
  - job_name: "fastapi_app"
    static_configs:
      - targets: ["localhost:8000"]
```

**5) Start Prometheus server as**

```
prometheus --config.file=prometheus.yml
```

Prometheus will periodically scrape metrics from http://localhost:8000/metrics

**6) Access App and Metrics:**

Application root: http://localhost:8000/greet

Prometheus metrics: http://localhost:8000/metrics

# Monitoring in Kubernetes

- Kubernetes introduces unique challenges for monitoring due to its distributed nature.
- In Kubernetes, observability focuses on metrics at multiple levels:

1) **Cluster Health:** Monitor the control plane, nodes, and network components.

2) **Deployment Health:** Check replica counts, rollout status, and service availability.

3) **Container Health:** Focus on container resource usage (CPU, memory).

4) **Application Health:** Service-level metrics, error rates, response times.



# Metrics in Kubernetes

- Kubernetes metrics are generally categorized as follows:

1) **Cluster Health Metrics:** Availability and performance of nodes and control plane components (API server, etcd).

2) **Deployment Metrics:** Status of Pods and Deployments, including replica counts and rollout status.

3) **Container Metrics:** CPU and memory usage, restarts, and liveness/readiness probe results.

4) **Application Metrics:** Business and service metrics related to application functionality.

5) **Runtime Metrics:** Resource usage over time to detect anomalies and trends.

# Example Workflow: Observability and Monitoring Setup

- **Collect Metrics**:
- Set up Prometheus on a Kubernetes cluster to scrape metrics from the Kubernetes API server, containers, and applications.
- Use various service types from Prometheus for various types of services we are running.

- **Store and Visualize Metrics**:
- Integrate Prometheus with Grafana for visualizing cluster health metrics.
- Set up Grafana dashboards for Golden Signals (latency, traffic, errors, saturation).



# Example Workflow: Observability and Monitoring Setup

- **Alerting**:
- Define Prometheus alert rules for critical metrics (e.g., high latency or error rates).
- Configure Alertmanager to send notifications based on these rules.

- **Continuous Monitoring**:
- Use CloudWatch for monitoring AWS resources if running Kubernetes on EKS.
- Set up CloudWatch alarms to notify based on specific criteria (e.g., node failures).

# Delivering Secure Software through Continuous Delivery

- Continuous Delivery (CD) emphasizes automation, fast feedback loops, and frequent releases, which are challenging when integrating security due to the complexity and the potential to slow down the pipeline.

**Secure software delivery can be achieved by:**

- **Automated Testing:** Integration of security-focused automated tests in the pipeline, including static analysis (SAST), dynamic analysis (DAST), and dependency scanning.

- SAST examples: SonarQube, Fortify SCA, ESLint
- DAST examples: OWASP ZAP, Burp Suite

- **Secured Environments:** Using infrastructure-as-code (IaC) to create reproducible and isolated environments, ensuring that configurations are secure and consistent across development, testing, and production.

- **Release Gates:** Establishing security checks as mandatory gates in the pipeline, such as requiring code scanning or penetration testing before deployment.



# Fortify Scan Issue – Example - SQL Injection

MERN
React – frontend – login page which takes username and password s input from user
<> ,<>

Express + Node.js – apis / endpoints – js

M - mysql db
select *
from auth
where username = <>
and password = <>;

**Typical flow: user – abc , password – xyz**
select *
from auth
where username = "abc"
and password = "xyz";

**SQL Injection: user = abc**
**password – "w\" or 1=1"**
select *
from auth
where username = "abc"
and password = "w" or 1=1 ;

# Example – XSS

- **Cross-Site Scripting (XSS)** is a web security vulnerability that allows attackers to inject malicious scripts (usually JavaScript) into content that is then rendered and executed in a user's browser.

Ex:

1) User input form

```
<form action="/search">
  <input type="text" name="query">
  <button type="submit">Search</button>
</form>
```

2) URL after user submits search:

https://example.com/search?query=hello

3) Server Response:

```
<p>You searched for: hello</p>
```

4) Hacker modifies URL to inject JS:

https://example.com/search?query=login>

5) Attacker shares this link with a target (victim) — via email, social media, or embedded in a webpage.

6) Victim clicks the link and opens it in their browser:

- The script executes in the victim's browser context.
- So, document.cookie returns the victim's session cookie for example.com.
- Attacker steals the user's session cookie and impersonates them.
- **They can access your account without needing a password!.**



# Continuous Delivery using AWS CodePipeline

- **Step 1: Set Up Your CI/CD Pipeline:**
- Create a codepipeline and connect your source repository (ex: AWS CodeCommit, GitHub, etc.) where your application code is stored.

- **Step 2: Add Build and Test Stages:**
- Add a Build stage that compiles your code and runs tests.
- Ensure that you include tests that must pass before a release can proceed (e.g., unit tests, integration tests).

- **Step 3: Implement the Release Gate:**
- Determine the conditions that must be met for a release to proceed.
- This could include:
  - Successful completion of previous stages (build, tests).
  - Compliance checks (e.g., adherence to coding standards, security policies).
  - Manual approval from specific users or groups.

- **Step 4: Configure Notifications and Monitoring:**
- CloudWatch Alarms/SNS Notifications:

# Security and Compliance Challenges and Constraints in DevOps

1) **Speed vs. Security:** DevOps pushes for rapid releases, often compromising security reviews that could slow the pipeline.

2) **Lack of Security Knowledge:** DevOps teams may lack deep security knowledge, making it hard to implement best practices.

3) **Toolchain Complexity:** Integrating various tools can expose vulnerabilities in the toolchain itself.

4) **Compliance Requirements:** For regulated industries (like finance or healthcare), compliance frameworks such as HIPAA, GDPR, and SOC2 require stringent controls, making it challenging to maintain a fast-paced CI/CD process.

- To overcome these challenges, DevOps teams integrate **Security as Code** (i.e., using code to define security policies), enhance their security skills.
- Apply automation to achieve both compliance and security without compromising on speed.



# Injecting Security into DevOps

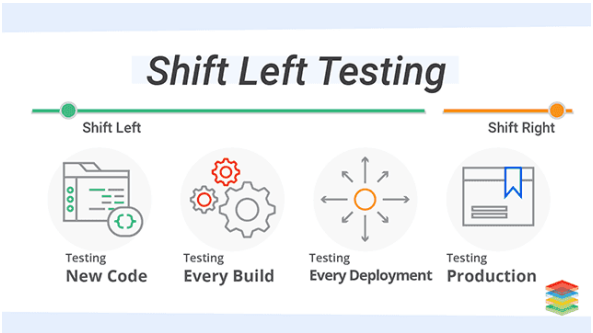
1) **Shift Security Left:** Moving security checks earlier in the development process, such as in code reviews and continuous integration.

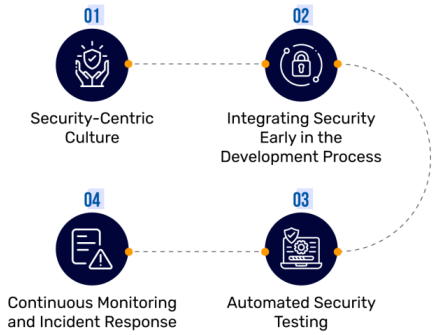
2) **Security Tools in CI/CD:** Integrating tools like static code analysis, vulnerability scanning, and dependency management within the CI/CD pipeline.

3) **Secrets Management:** Ensuring that sensitive information (e.g., API keys, passwords) is managed securely, using tools like AWS Secrets Manager or HashiCorp Vault.

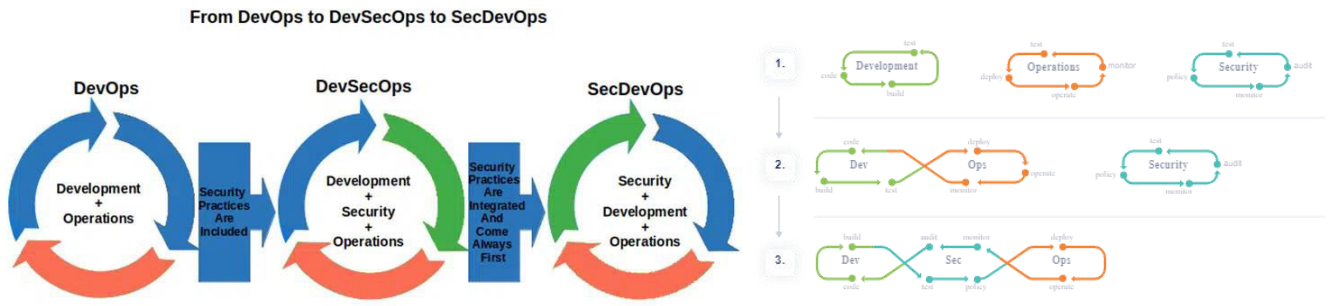
4) **Real-time Monitoring:** Implementing logging, monitoring, and alerting to detect any security issues in real time.

5) **Automation:** Automating security tasks where possible, reducing the risk of human error while ensuring continuous security testing.



# DevSecOps vs DevOpsSec vs SecDevOps




1) **DevOpsSec:**
- Starts with DevOps practices, with security added as a layer on top, often later in the pipeline.
- This approach is more reactive, security as DevOps practices mature

2) **SecDevOps:**
- Emphasizes security as the core foundation, followed by development and operational practices.
- This approach places security as the primary focus, even if it impacts speed.

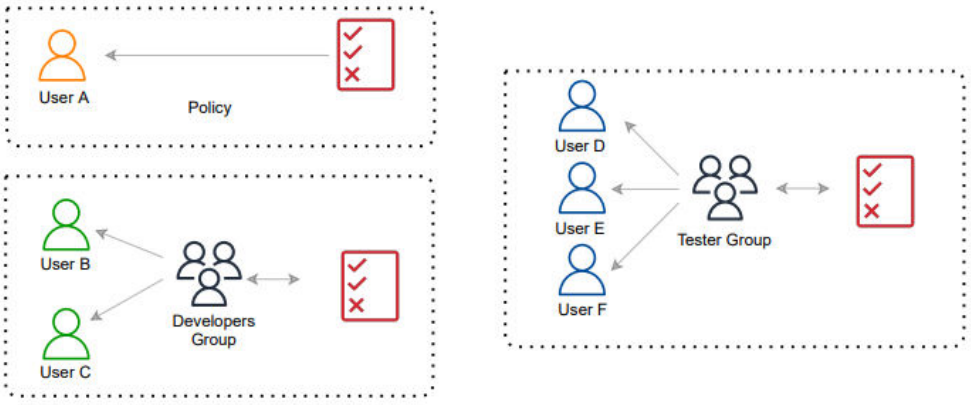
3) **DevSecOps:**
- Integrates security at each stage of DevOps, promoting **security as everyone's responsibility.**
- Security is treated as a part of development and operations from the start.
- In practical terms, **DevSecOps** is widely adopted due to its balanced approach. Security is embedded into each phase without overshadowing the goals of DevOps, unlike SecDevOps, which might prioritize security over other objectives



# AWS – IAM

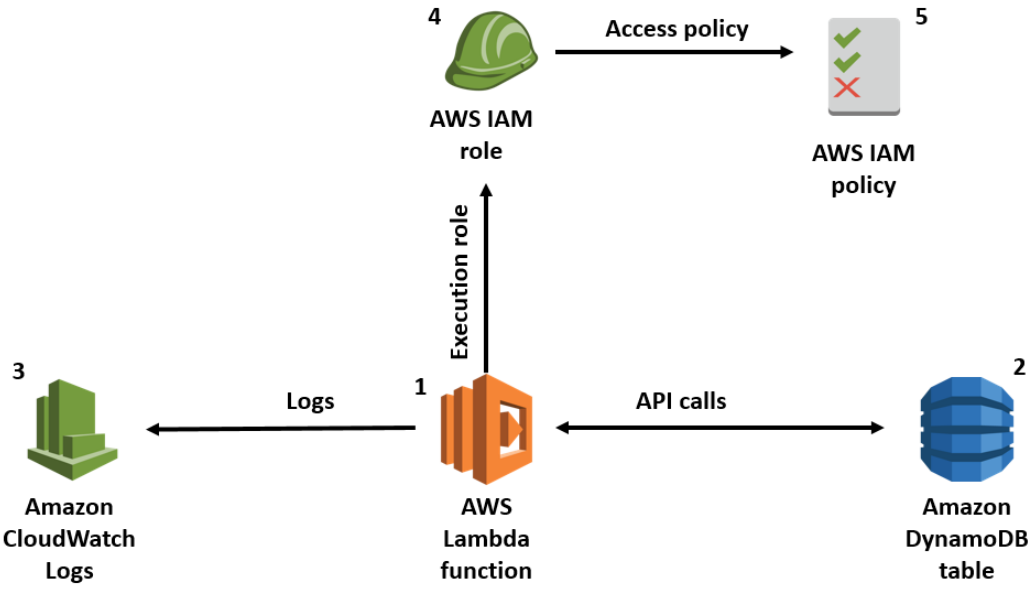
- IAM stands for Identity and Access Management.
- It is a web service that helps you securely control access to AWS resources for your users.
- **Terminology:**
- **IAM Users:** root (admin) user , other (non-admin) users
- **IAM Groups:** to manage permissions for multiple users more efficiently.
- **IAM Policies:** to enforce security best practices and restrict access.
- **IAM Roles:** to grant permissions to AWS services.

# AWS – IAM Users , Policies, Groups




AWS Policies – AWS managed vs custom

# AWS – IAM Roles



# Authentication

**Definition:**

process of verifying the identity of a user or system attempting to access a resource.

**Purpose:**

It ensures that the entity trying to access the resource is who it claims to be.

**Methods:**

Common authentication methods include passwords, biometric authentication (fingerprint, facial recognition), security tokens, and multi-factor authentication (MFA).

# Authorization

**Definition:**

process of determining what actions an authenticated user or system is allowed to perform on a resource.

**Purpose:**

It enforces access control policies to ensure that only authorized entities can perform specific actions.

**Methods:**

Authorization is typically implemented using permissions, roles, and policies that define the level of access granted to authenticated entities.

# Authentication vs Authorization

**Authentication:**
Users authenticate using their username and password or access keys to verify their identity.

**Authorization:**
After authentication, AWS IAM determines what actions the authenticated user is allowed to perform based on the policies attached to their user or group.

**authentication verifies identity, while authorization determines access rights based on that identity.**

**Example:**
**Authentication** - A user logging into an application using a username and password.

**Authorization** - After logging in, a user is allowed to view, edit, or delete specific resources based on their permissions

# AWS - IAM Best Practices

- **Principle of Least Privilege:**
Assign only the permissions required to perform a task.

- **Regular Audits:**
Regularly review IAM policies and permissions to ensure they are up-to-date and secure.

# Security as Code

- Security as Code is a DevOps practice that leverages IaC and security tools to automate security practices:

1) **Static Application Security Testing (SAST):** Tools like SonarQube analyze code for vulnerabilities during development.

2) **Dynamic Application Security Testing (DAST):** Tools like OWASP ZAP or Burp Suite run tests against live applications, simulating attacks to detect vulnerabilities.

3) **Dependency Scanning:** Tools such as Snyk or Dependabot check for vulnerabilities in libraries and dependencies.

4) **Infrastructure Security:** Using IaC tools (like Terraform) with security best practices baked in (e.g., ensuring IAM roles have the least privilege).



# SAST vs DAST

| Aspect | SAST | DAST |
|---|---|---|
| Testing Approach | Static (analyzes code without execution) | Dynamic (analyzes app during execution) |
| Type of testing | White-box | Black-box |
| Vulnerabilities | Code logic and syntax | Runtime behavior and configurations |
| Example Vulnerabilities detected | SQL Injection, Hardcoded credentials, Insecure API Usage | Authentication issues, Cross-site scripting (XSS) |
| Speed | Fast | Slower |
| False positives | More likely | Less likely |

# Security as Code using AWS IAM

- AWS Identity and Access Management (IAM) plays a crucial role in implementing security as code (SaC) practices by providing robust tools and features for managing permissions, roles, and policies in a programmable manner:

1) **Fine-Grained Access Control:** IAM allows you to create detailed policies that define what actions are allowed or denied on specific AWS resources.

2) **Infrastructure as Code (IaC) Integration:** Tools like AWS CloudFormation or Terraform enable you to define IAM roles, policies, and permissions as code. This means you can version control your security configurations alongside your application code, promoting consistency and ease of management.

3) **Role-Based Access Control (RBAC):** IAM supports the creation of roles that can be assumed by AWS services (e.g., EC2, Lambda).

This allows you to securely grant permissions to applications and services without hardcoding credentials, enhancing security in automated deployments.



# Security as Code using AWS IAM Policy Example

- The following SAC grants read-only access to an S3 bucket named ssd-bucket:

```
{
  "Version": "2025-05-11",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::ssd-bucket",
        "arn:aws:s3::: ssd-bucket/*"
      ]
    }
  ]
}
```

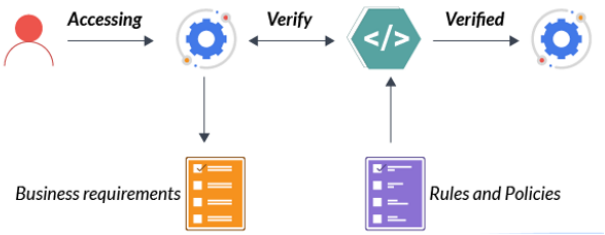
- **Effect:** "Allow" - grants permission.
- **Action:** Lists allowed actions.

s3:GetObject = read files

s3:ListBucket = list objects

- **Resource:** Defines the objects the actions apply to (here it is the S3 bucket)

# Compliance as Code




- Compliance as Code applies IaC concepts to ensure compliance requirements are met automatically through code:

1) **Policy-as-Code:** Define and enforce policies within IaC. For example, using tools like Open Policy Agent (OPA) and AWS Config to enforce rules such as all storage buckets must be encrypted.

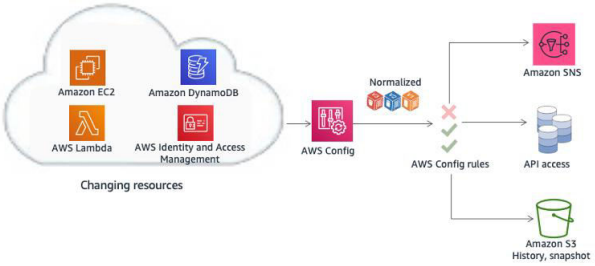
2) **Automated Auditing:** Compliance frameworks (e.g., CIS benchmarks for cloud services) are enforced programmatically, ensuring configurations comply with policies and automatically flagging any deviations.

3) **Automated Documentation and Reporting:** Compliance checks can generate audit-ready reports, reducing the burden of manual documentation and ensuring up-to-date compliance.



# Compliance as Code using AWS Config

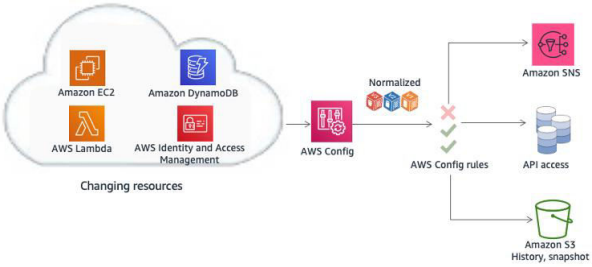
- We can use AWS Config to monitor AWS resources continuously.
- We can define rules to evaluate whether your AWS resource configurations comply with your defined policies.
- AWS Config can trigger notifications or automated remediation when resources are non compliant.
- **Managed Rules:** AWS Config offers managed rules for common compliance frameworks.
- Ex: AWS Foundations Benchmark.
- We can enable these rules to check compliance automatically.



## Compliance as Code using AWS Config – Example

- To define that all S3 buckets must have server side encryption enabled:
- We have **AWS Managed AWS Config rule:** s3-bucket-server-side-encryption-enabled
- We define a JSON for use via CloudFormation or AWS CLI:

```
{
  "ConfigRule": {
    "ConfigRuleName": "s3-bucket-encryption-check",
    "Description": "Ensure all S3 buckets have server-side encryption enabled.",
    "Source": {
      "Owner": "AWS",
      "SourceIdentifier": "S3_BUCKET_SERVER_SIDE_ENCRYPTION_ENABLED"
    },
    "Scope": {
      "ComplianceResourceTypes": [
        "AWS::S3::Bucket"
      ]
    },
    "InputParameters": {}
  }
}
```

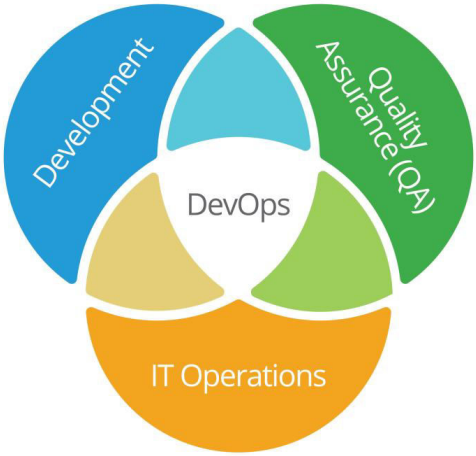


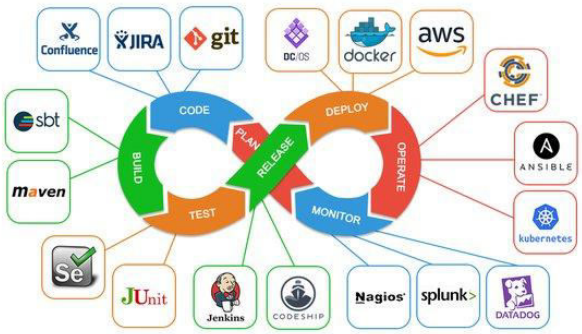





# Recap – What is DevOps?

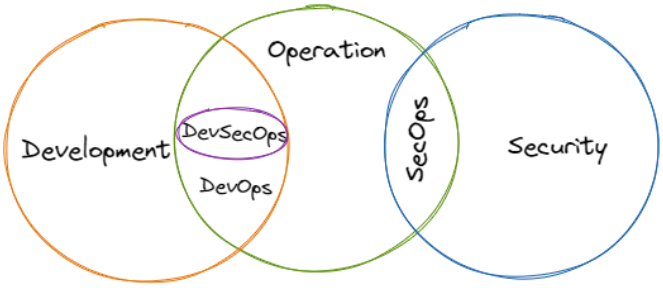
- DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).
- It aims to shorten the software development life cycle and provide continuous delivery with high software quality.



# 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**



innovate achieve lead

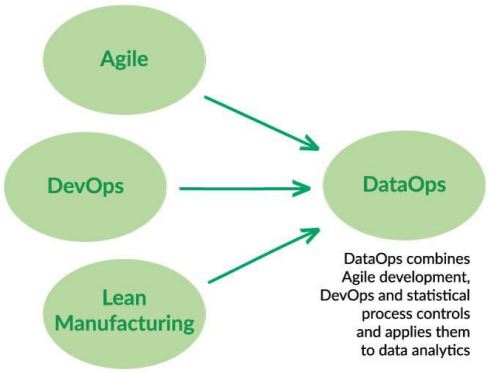
# Overview of Xops : DataOps

**Focus:**

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

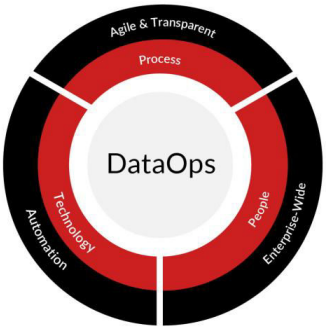
**Practices:**

- 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).




BITS Pilani, Pilani Campus

# DataOps Stack

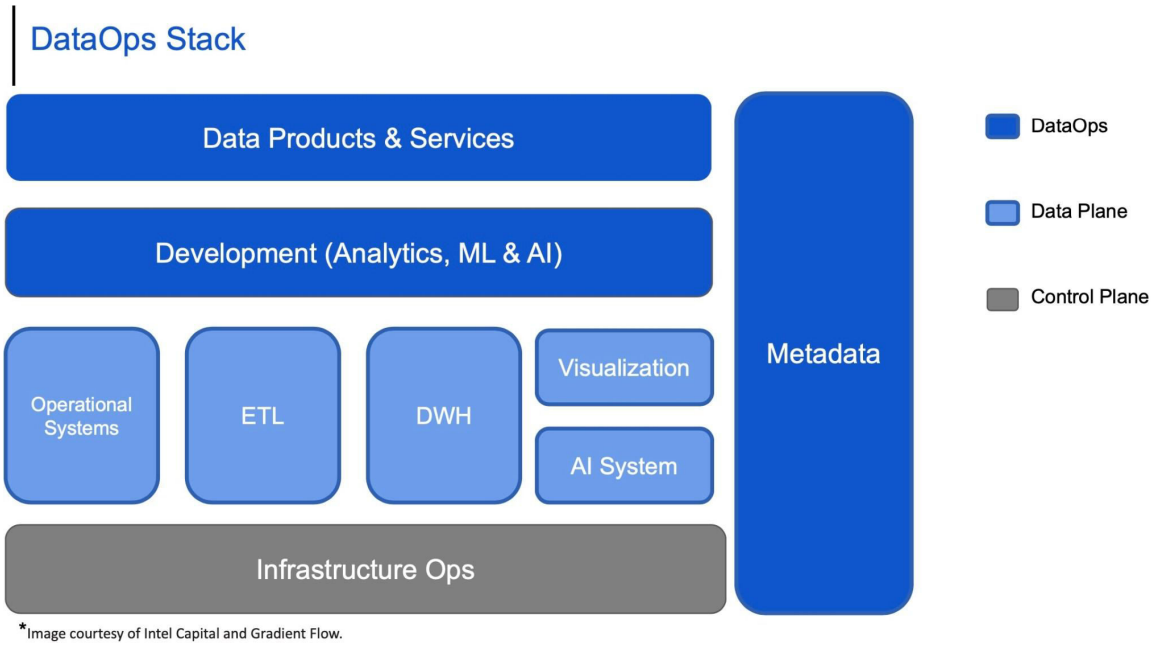



*Image courtesy of Intel Capital and Gradient Flow.



# 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**

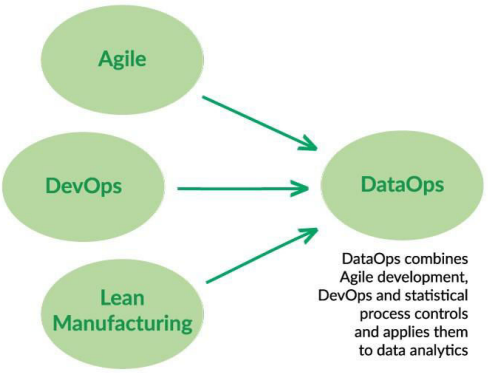
The primary measure of data analytics performance is the degree to which insightful analytics are delivered, incorporating accurate data.

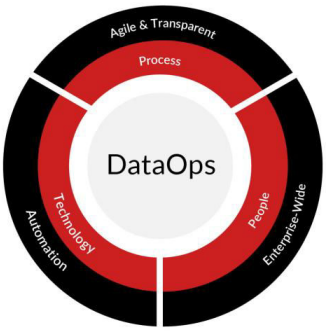
**3) Daily interactions**

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

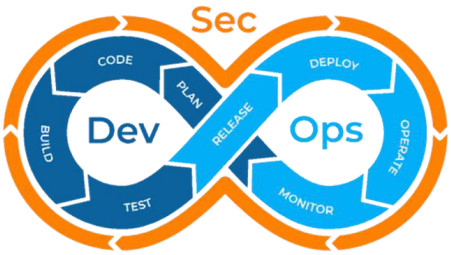
- SecOps stands for security-operations.

**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.

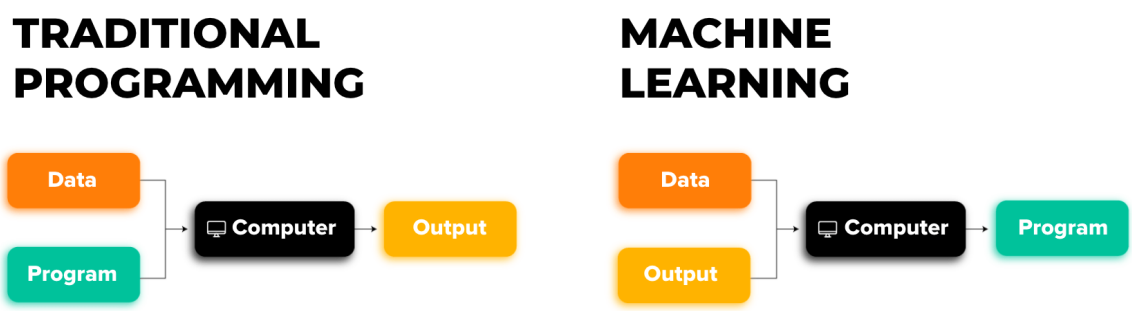
# Overview of Xops : MLOps

- **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).



# What is a Machine Learning Model?

- An ML model is a mathematical representation that learns patterns from historical data to make predictions or decisions.
- It's created through training, where algorithms use labeled data to identify features that contribute to accurate predictions.



# What is a Machine Learning Model?

Traditional Algo :
i/p: x =[1,2,3]
Given to system

System gives out
o/p: y = [3,4,5]

Program:
Given to system
y = x + 2

ML Algo:
i/p: x =[1,2,3]
o/p: y = [3,4,5]
Given to system

Form: (ML algo – linear regression)
y = mx + c (form)

Training data:
[1,3]
[2,4]
[3,5]
Test data:
[8] => ? Expect 10

Machine learns
And comes up with
m = 1, c = 1.8
y = x + 1.8
Predict:
X = 8, Y = 9.8



# Key components of an ML model

- **Algorithm**: The underlying mathematical framework used to build the model (e.g., linear regression, decision trees).
- **Features**: Input variables that the model uses to make predictions.
- **Parameters**: Values that the model optimizes during training to fit the data.
- **Hyperparameters**: Configurations set before training to influence the learning process (e.g., learning rate, batch size).

# ML Artifacts

- **Models:** The trained ML models themselves.
- **Datasets:** The data used for training and validation.
- **Code Artifacts:** Scripts and code for preprocessing, training, and inference.
- **Metadata:** Information about experiments, hyperparameters, and training configurations.

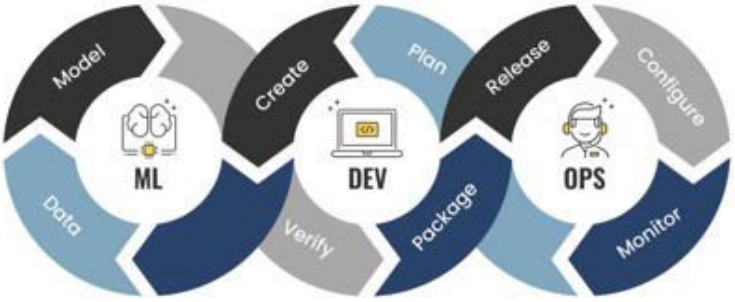


# MLOps

- MLOps stands for Machine Learning Operations.
- MLOps is a set of practices combining machine learning, DevOps, and data engineering to automate and enhance the deployment, monitoring, and lifecycle management of ML models in production.
- It focuses on integrating ML systems with software engineering principles to enable seamless collaboration between data scientists, ML engineers, and DevOps teams.

# MLOps

- MLOps stands for Machine Learning Operations.
- MLOps is a set of practices combining machine learning, DevOps, and data engineering to automate and enhance the deployment, monitoring, and lifecycle management of ML models in production.
- It focuses on integrating ML systems with software engineering principles to enable seamless collaboration between data scientists, ML engineers, and DevOps teams.






# Challenges in MLOps

- **Complex Workflows:** ML involves diverse workflows, including data processing, model training, testing, and deployment.
- **Model and Data Drift:** Over time, models can lose accuracy due to changes in data distribution (data drift) or evolving business contexts (concept drift).
- **Scalability:** Scaling model training, deployment, and serving is challenging, especially with large datasets. Large-scale MLOps implementations involve multiple pipelines for feature engineering, model training, and real-time inference serving.
- **Collaboration:** MLOps requires coordination among multiple teams with different expertise.
- **Monitoring:** Continuous monitoring of ML models is necessary to ensure they perform as expected over time.

# People of MLOps

- The key stakeholders in MLOps are:

1) **Product Managers:** They define the business requirements, monitoring model performance and ensuring alignment with business goals

2) **Data Scientists:** They develop and experiment with ML models, focusing on model accuracy and feature engineering.

3) **ML Engineers:** They handle model optimization, deployment, and ensure models can scale and meet latency requirements.

4) **Data Engineers:** They prepare and manage the data infrastructure, ensuring high-quality and consistent data for ML models.

5) **DevOps Engineers:** They ensure the infrastructure is automated, scalable, and secure, enabling the seamless deployment of ML pipelines.



# Key MLOps Features

- **Automated ML Pipelines:** Automating data preprocessing, model training, and deployment pipelines to streamline the workflow.
- **Continuous Integration/Continuous Deployment (CI/CD):** Automated workflows for integrating code changes, testing, and deploying ML models.
- **Model Versioning and Tracking:** Version control for ML models, data, and experiments for reproducibility and traceability.
- **Monitoring and Retraining**: Tools to monitor model performance in production and trigger retraining when performance degrades.
- **Data and Model Governance:** Ensuring compliance with data privacy and governance policies, managing data lineage, and access control.
- **Experiment Management**: Tracking experiments to optimize hyperparameters, algorithms, and preprocessing steps efficiently.

# Types of ML Algorithms

Learning => given x,y => model learns mapping function f which is y = f(x)

X => f(x) -> prediction

y -> ground truth

1) Supervised Learning ( y is given)

Ex: predicting revenue for next year using historical data

2) Unsupervised Learning ( y is not given)

Ex: Customer segmentation using clustering

get patterns from data

product -> t- shirt -> what all sizes? 3 GROUPS -> G1 , G2 , G3

S , M, L

K = 3

3) Reinforcement Learning

Ex: self driving car



# Containerization of ML Models

- Docker and Kubernetes are commonly used to package ML models with dependencies, ensuring that they run consistently across environments.
- Containerization helps in scaling models across multiple nodes and allows easy rollback in case of model failure.
- The following are the steps of containerizing and deploying an ML algo.

1) **Train the Model**
2) **Save the Model**
3) **Create a Prediction Script (ML algo)**
4) **Create a Dockerfile**
5) **Build and Run the Docker Container**
6) **Optimize the Container**
7) **Deploy the container**
8) **Monitor and Scale**

# Containerization of ML Models

**1) Train the Model:**

- Train model using any ML library (e.g., TensorFlow, PyTorch, scikit-learn).

**2) Save the Model:**

- Export the trained model in a suitable format:
- Ex:

  TensorFlow: .h5 or SavedModel format.

  PyTorch: .pth or .pt.

  Scikit-learn: Use joblib or pickle.

**Code:**

```
import joblib
model = ...  # Train your model
joblib.dump(model, 'model.pkl')
```



# Containerization of ML Models

**3) Create a Prediction Script (ML algo)**

- Create a Python script to load the model and expose an API for predictions.
- Use a framework like FastAPI or Flask.

```
from fastapi import FastAPI, HTTPException
import joblib
import numpy as np
from pydantic import BaseModel

app = FastAPI()

# Load the model
model = joblib.load("model.pkl")

# Define the input schema
class InputData(BaseModel):
    feature1: float
    feature2: float
    feature3: float
```

```
@app.post("/predict")
def predict(data: InputData):
    try:
        # Convert input to a NumPy array
        input_data = np.array([[data.feature1,
data.feature2, data.feature3]])
        prediction = model.predict(input_data)
        return {"prediction": prediction.tolist()}
    except Exception as e:
        raise HTTPException(status_code=500,
detail=str(e))
```

# Containerization of ML Models

### 4) Create a Dockerfile

- Create a dockerfile that defines the container's environment and how to run the application.

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

# Set the working directory
WORKDIR /app

# Copy the local files to the container
COPY . .

# Install dependencies
RUN pip install --no-cache-dir fastapi uvicorn joblib numpy scikit-learn

# Expose the API port
EXPOSE 8000

# Command to run the application
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```



innovate achieve lead

# Containerization of ML Models

### 5) Build and Run the Docker Container

**Build the Docker Image:**

```
docker build -t ml-model-api .
```

**Run the Container:**

```
docker run -d -p 8000:8000 ml-model-api
```

**Test the API:**

Use a tool like curl or Postman to test the API:

```
curl -X POST "http://localhost:8000/predict" \ -H "Content-Type: application/json" \ -d '{"feature1": 1.5, "feature2": 2.3, "feature3": 3.7}'
```

# Containerization of ML Models

6) Optimize the Container

a) **Reduce Image Size:** Use a lightweight base image like python:3.9-alpine.

b) **Use Multi-stage Builds:** Split the Dockerfile into build and runtime stages to exclude unnecessary build dependencies.

c) **Add a .dockerignore File:** Exclude unnecessary files (e.g., training data, logs) from the image.

```
*.pyc
__pycache__/
*.log
data/
```



# Containerization of ML Models

7) Deploy the Container

- Deploy the container to a production environment:

a) **Docker Compose:** Orchestrate multi-container deployments.

b) **Kubernetes:** For scalable, production-grade deployments.

c) **Cloud Services:**
- Use AWS ECS, Azure AKS, or Google Kubernetes Engine (GKE) for container orchestration.
- Use AWS SageMaker or Google AI Platform for model hosting.

# Containerization of ML Models

8) Monitor and Scale

a) Monitoring:

- Integrate monitoring tools like Prometheus and Grafana to track container metrics.
- Use application performance monitoring (APM) tools for API response time and error tracking.

b) Scaling: Use horizontal scaling (multiple replicas) with Kubernetes or a cloud service to handle high traffic.



# Key components of MLOps

1) Continuous Integration (CI) and Continuous Deployment (CD):
   - Incorporates code versioning, testing, and experiment tracking, ensuring that model code is reproducible and compliant with quality standards.
   - Ensures automated deployment of models, with proper checks to transition models from development to production seamlessly.
   - Tools like Jenkins, GitHub Actions, GitLab CI, Kubeflow Pipelines can be used.

a) Version Control for Models and Data:

- Implementing version control (using tools like Git + DVC) ensures that both models and datasets are tracked over time.
- This enables the team to reproduce experiments, compare different versions of the models, and manage changes in datasets, which is crucial for maintaining model reliability and reproducibility.
- This contributes to model stability and continuous improvement.
- This is a well known devops practice called **Data Versioning and Lineage Tracking**

# DVC

- DVC (Data Version Control) is an open-source version control system specifically designed for machine learning (ML) projects.
- It complements Git by handling large datasets, model artifacts, and other ML-specific assets that Git struggles with.
- https://dvc.org/

**a) Initialize a Project with Git and DVC:**

```
# Initialize Git and DVC
git init
dvc init

# Add DVC to Git tracking
git add .dvc .gitignore
git commit -m "Initialize Git and DVC"
```



# DVC

**b) Add a Dataset**

```
dvc add data/train.csv

# This creates a .dvc file pointing to the dataset
git add data/train.csv .dvc
git commit -m "Add dataset to DVC"
```

- DVC moves data/train.csv to a cache directory. A .dvc file is created, which contains metadata (like hash, size, etc.).
- The dataset itself is excluded from Git tracking (avoiding Git's size limits).

# DVC

### c) Push Dataset to Remote Storage:

- We can Configure a remote storage (e.g., AWS S3, Google Drive, or a local server).

```
dvc remote add -d myremote s3://mybucket/path

dvc push
```

- The dataset (data/train.csv) is uploaded to the remote storage.
- The .dvc file stays in the Git repo, acting as a pointer



innovate achieve lead

# Key components of MLOps

### 2) Automated Model Testing and Validation within CI/CD pipeline:

- Establishing automated testing pipelines (unit tests, integration tests, and performance validation) ensures that models meet performance and accuracy standards before deployment.
- By automatically validating model performance, you can avoid errors in production that could negatively affect user experience or lead to system downtime.
- Track performance shift, data drift, latency, errors.

### 3) Model Monitoring and Logging

- Tools like Prometheus + Grafana, Seldon, Evidently AI etc can be used to Track performance drift, data drift, latency, errors, and predictions.
- Detects when model performance degrades or when inference data changes, enabling proactive retraining or rollback

# Key components of MLOps

### 4) Automated Retraining Pipelines:

- Tools like Airflow, Kubeflow Pipelines can be used to automatically retrain models on new data or when performance degrades.
- Keeps models fresh without manual intervention.
- Ex:

```
from airflow import DAG
from datetime import datetime
import train_pipeline  # your training logic here

def check_new_data():
    # e.g., check if new file exists in S3 or DVC has new version
    return True

def retrain_model():
    train_pipeline.run()  # call your model training function
```

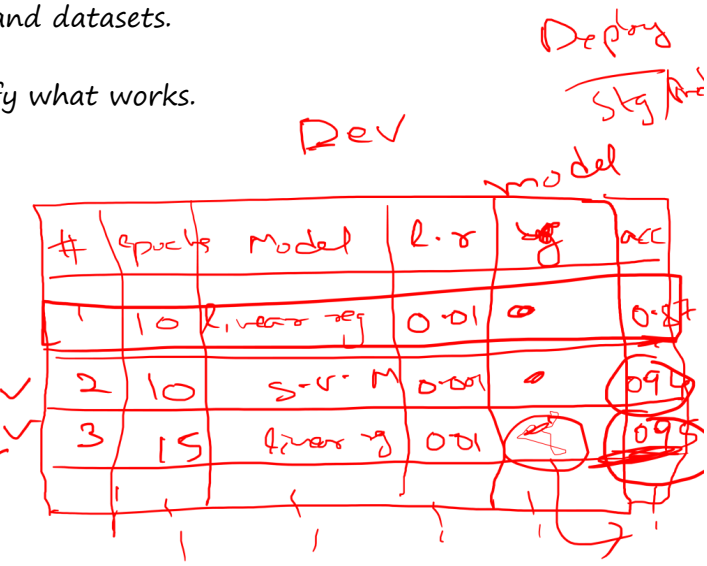


# Key components of MLOps

### 5) Experiment Tracking

- Tools like MLflow, Weights and BiasesPractice can be used to track hyperparameters, code versions, metrics, and datasets.
- Rapidly iterate on experiments and identify what works.
- Ex:

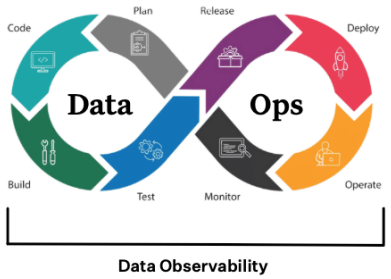
```
import mlflow
mlflow.start_run()
mlflow.log_param("learning_rate", 0.01)
mlflow.log_param("epochs", 10)
mlflow.log_metric("accuracy", 0.873)
# Save the model
mlflow.sklearn.log_model(model, "model")
mlflow.end_run()
```




BITS Pilani, Pilani Campus

# DataOps

- DataOps stands for Data Operations.
- DataOps is a collaborative data management practice focused on improving the quality, speed, and reliability of data-related operations.
- It applies principles of Agile, DevOps, and lean manufacturing to data analytics and engineering, enabling organizations to deliver data-driven insights quickly and efficiently.




**DevOps Users & Tools** — Software Engineers, comfortable with coding and complexity of multiple languages, tools, and hardware/software.

**DataOps Users & Tools** — Data Scientists, Engineers, and Analysts who want to just analyze data and build models.



# Key components of DataOps

1. **Automated Data Validation:**
   - Implement scripts or tools to validate incoming data for missing values, schema mismatches, or outliers.
   - Ensures the input data to the ML algorithm is clean and consistent.
   - Example Tools: Talend, Apache Griffin Deequ.

2. **Data Lineage and Provenance:**
   - Track the origin and transformations of data to ensure transparency and reliability.
   - Helps in debugging issues and understanding the data flow through the pipeline.
   - Example Tools: Informatica Enterprise Data Catalog, Microsoft Purview

- X -> y →f3(f2(f1(x)))
- Y = composite function = y3(y2(y1(x)))

# Key components of DataOps

**3. Automated Data Profiling**

- Tools like Pandas Profiling, Databricks Data Profile etc can be used to analyze distributions, outliers, and anomalies regularly.
- Helps detect drift or unexpected patterns in incoming data.

**4. Incremental Data Processing**:

- Process only the new or changed data instead of reprocessing the entire dataset.
- Improves efficiency and ensures that only new data is used for periodic model retraining, reducing computational overhead.
- Example Tools:

1) Apache Kafka – Supports incremental processing by allowing consumers to read only new data.
2) Delta Lake – Handles incremental data processing with features like MERGE INTO for upserts.



# MLOps vs DataOps

| Aspect | MLOps | DataOps |
|---|---|---|
| Definition | Practices for deploying, monitoring, and maintaining ML models in production. | Practices for managing data workflows, ensuring data quality, and facilitating collaboration. |
| versioning | machine learning models, their parameters, hyperparameters, and the associated code. | datasets, their schemas, transformations, and data pipelines. |
| Key Components | Model versioning, retraining, deployment, monitoring, CI/CD for ML pipelines. | Data versioning, data quality checks, lineage, and pipeline automation. |
| Tools | MLflow, Kubeflow, TFX, SageMaker, Azure ML. | DVC, Apache NiFi, Talend, Great Expectations, dbt, Airflow. |

# MLOps in Practice: Deployment Strategies

**Shadow Deployment:**

- Running the new model in the background without affecting the production model, used for testing performance.
- new version passively receives a copy of real user requests to process in parallel with the current version.

**Canary Deployment:**

- Gradually rolling out the model to a small percentage of users before a full rollout.

**A/B Testing:**

- Deploying multiple models to compare their effectiveness based on real-world feedback.



# MLOps in Practice: Deployment Strategies

**Blue Green Deployment:**

- One environment (Blue) runs the current production version, while the new version is deployed to the other environment (Green).
- Traffic is then switched from Blue to Green once the new version is validated.

**Distributed Model Training and Inference:**

- Using distributed computing frameworks (e.g., TensorFlow, PyTorch) allows training the model on large datasets across multiple machines.
- For inference, deploying the model in a distributed manner across multiple instances or microservices ensures that the system can process high volumes of data in real time without performance degradation.

# MLOps in Practice: Marketing Recommendation Engines Case Study

- A marketing recommendation engine suggests products or content to users based on their preferences and behaviors.
- Using MLOps principles can significantly improve the efficiency and scalability of this use case.
- Steps in the MLOps Pipeline for a Recommendation Engine:

**1) Data Collection and Processing:**

- Collect data on user behavior, product views, and purchases.
- Preprocess the data using tools like Apache Spark or Pandas.

**1) Model Training and Experimentation:**

- Train collaborative filtering or neural network-based recommendation models.
- Track experiments, hyperparameters, and results using tools like MLflow or Weights & Biases.



# MLOps in Practice: Marketing Recommendation Engines Case Study

**3) Model Validation and Testing:**

- Evaluate models using validation metrics like mean squared error or mean average precision.
- Conduct A/B testing to assess model effectiveness in real-world scenarios.

**4) Continuous Integration and Deployment:**

- Package the model using Docker and deploy it to a staging environment.
- Run tests for data quality, model accuracy, and latency requirements.
- Use Kubernetes for scalable deployment and model management.

BITS Pilani, Pilani Campus

# MLOps in Practice: Marketing Recommendation Engines Case Study

**5) Monitoring and Retraining:**

- Continuously monitor the model's performance using tools like Prometheus and Grafana.
- Set up alerts for model drift and trigger automatic retraining pipelines using AWS Sagemaker or Google AI Platform.

**6) Feedback and Improvement:**

- Collect feedback from marketing teams and customers to refine the model.
- Integrate user feedback into retraining datasets to improve personalization.



# 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.



innovate achieve lead

# 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.

BITS Pilani, Pilani Campus

# 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.

BITS Pilani, Pilani Campus

# 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.

BITS Pilani, Pilani Campus