

# Cloud Foundation & DevOps Workshop

17<sup>th</sup>-18<sup>th</sup> May 2025

By

Siddharth Karkhanis

- 01 Virtualization
- 02 Cloud Computing
- 03 Agile Methodology
- 04 DevOps
- 05 DevOps Basic Tools

## Low Level Agenda



# 01 Virtualization

# Virtualization

What is Virtualization?

## Core of Virtualization

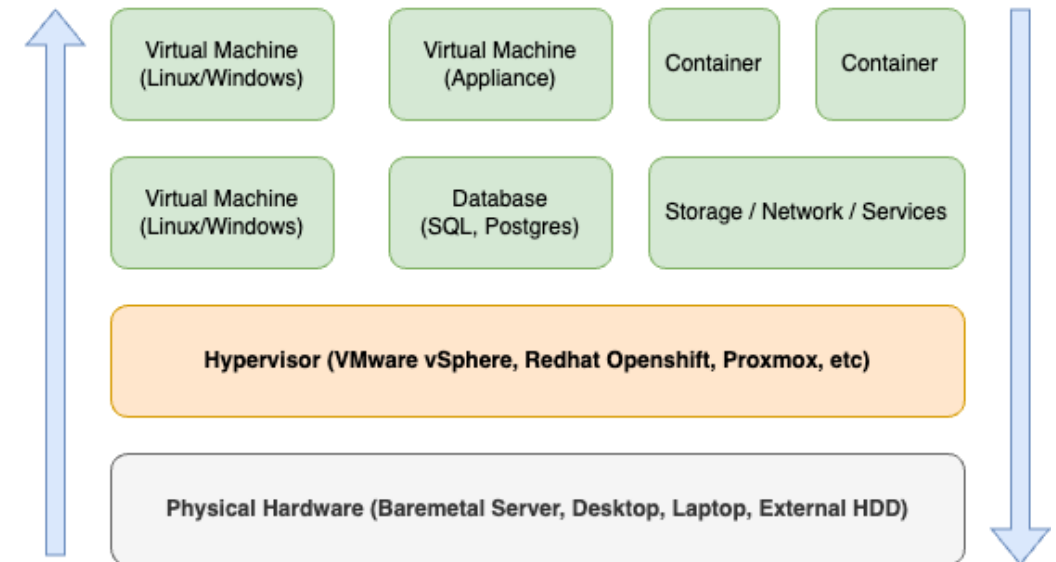
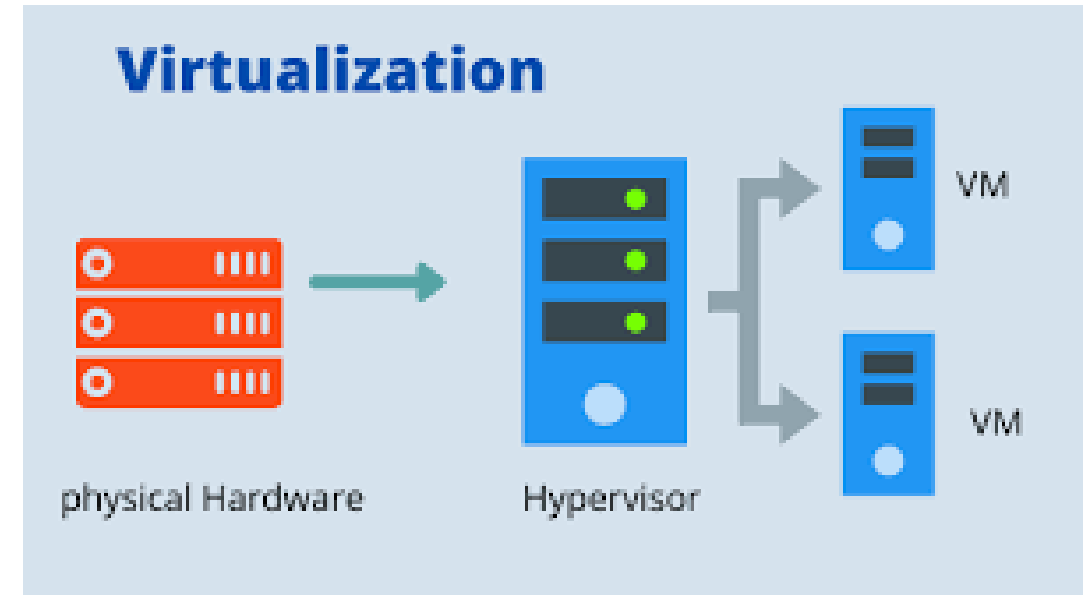
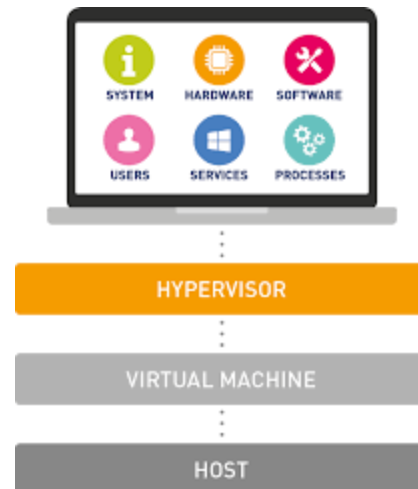
Virtualization abstracts physical resources, allowing multiple operating systems to run concurrently on a single hardware platform.

## What is it?

Virtualization is the process of creating a virtual version of physical components such as servers, storage devices, and network resources. It allows multiple operating systems and applications to run on a single physical machine by abstracting the hardware.

## Why It Matters?

- Improves hardware utilization
- Reduces operational cost
- Enables scalability and flexibility
- Provides isolation between workloads



# Hardware Level Virtualization

Traditional Virtual Machines (VMs)

## ✓ What it is:

You simulate entire hardware — CPU, memory, storage — so that each **Virtual Machine (VM)** behaves like a separate computer.

## ✓ How it works:

A software layer called a **hypervisor** sits directly on the physical machine (or on an OS) and allows you to run multiple **full operating systems** (e.g., Windows, Linux) on one physical server.

## ✓ Characteristics:

- Each VM has its own **kernel** (the core part of an OS)
- VMs are **heavier**, take more time to boot, and use more memory
- Ideal for **running multiple full systems** like test environments or legacy apps

## ✓ Examples:

VMware ESXi, Microsoft Hyper-V, KVM

# OS Level Virtualization

Containers, Desktop Virtualization

## ✓ What it is:

Instead of virtualizing hardware, you virtualize the **operating system** itself. Multiple **containers** share the same OS kernel but run in isolated spaces.

## ✓ How it works:

The host OS runs many **lightweight, standalone applications** (containers), each in its own space, without needing a full OS per app.

## ✓ Characteristics:

- Containers are **faster** and **lighter** than VMs
- They **start instantly** and use fewer resources
- Great for **microservices**, DevOps pipelines, and scalable cloud apps

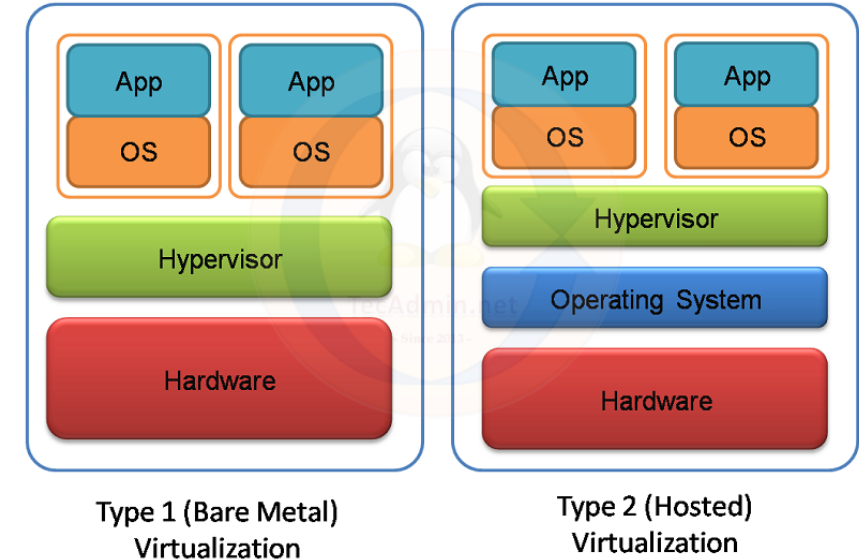
## ✓ Examples:

Docker, LXC, Podman

# Types of Virtualizations

## Type 1 & Type 2

Criteria	Type 1 hypervisor	Type 2 hypervisor
AKA	Bare-metal or Native	Hosted
Definition	Runs directly on the system with VMs running on them	Runs on a conventional Operating System
Virtualization	Hardware Virtualization	OS Virtualization
Operation	Guest OS and applications run on the hypervisor	Runs as an application on the host OS
Scalability	Better Scalability	Not so much, because of its reliance on the underlying OS.
Setup/Installation	Simple, as long as you have the necessary hardware support	Lot simpler setup, as you already have an Operating System.
System Independence	Has direct access to hardware along with virtual machines it hosts	Are not allowed to directly access the host hardware and its resources
Speed	Faster	Slower because of the system's dependency
Performance	Higher-performance as there's no middle layer	Comparatively has reduced performance rate as it runs with extra overhead
Security	More Secure	Less Secure, as any problem in the base operating system affects the entire system including the protected Hypervisor
Examples	<ul style="list-style-type: none"><li>• VMware ESXi</li><li>• Microsoft Hyper-V</li><li>• Citrix XenServer</li></ul>	<ul style="list-style-type: none"><li>• VMware Workstation Player</li><li>• Microsoft Virtual PC</li><li>• Sun's VirtualBox</li></ul>





# Virtualization

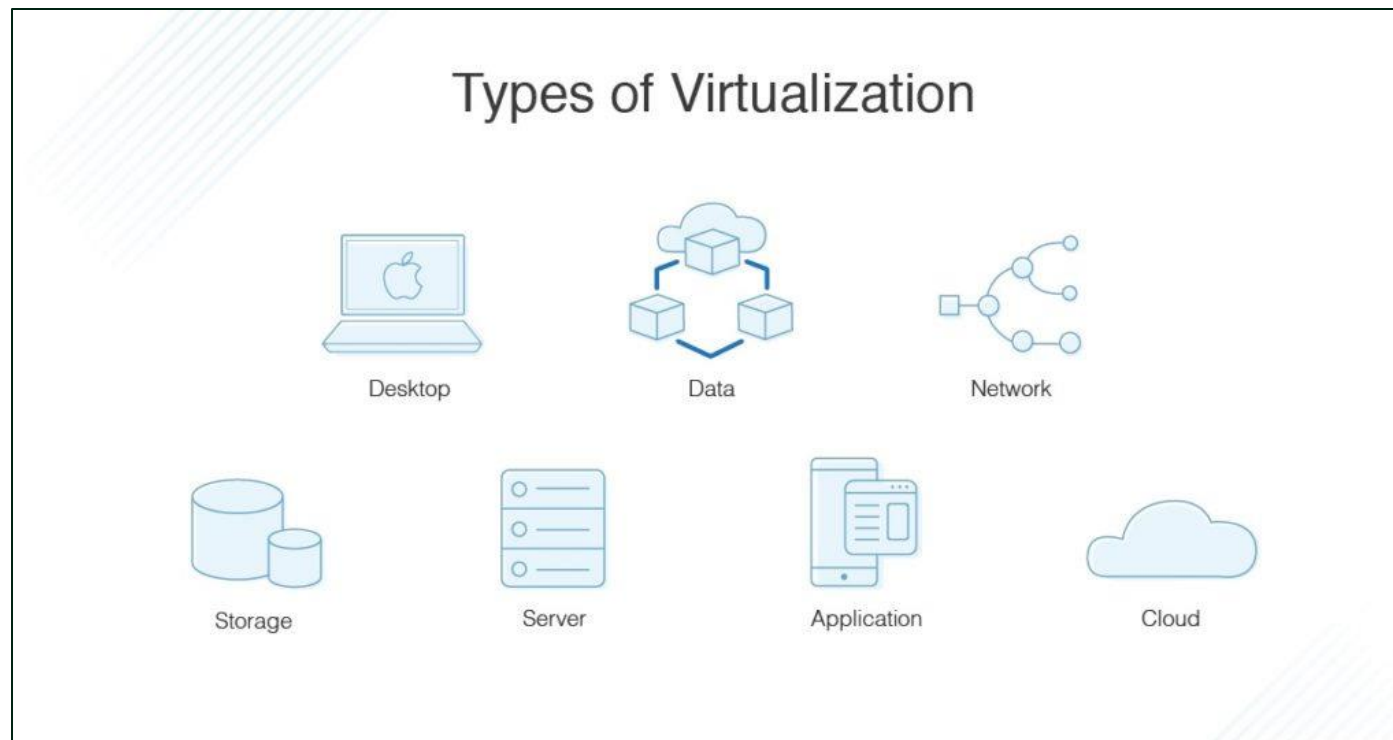
## Context of Cloud

### How Virtualization Powers Cloud?

- Virtualization is the **foundation of cloud computing** — it enables cloud providers to offer multiple isolated virtual machines on shared physical infrastructure.
- It allows **elastic resource allocation**, **multi-tenancy**, and **dynamic provisioning**.

### Why it's important for cloud?

- Enables Infrastructure as a Service (IaaS)
- Supports scalability and high availability
- Reduces cost and increases efficiency for providers and users



# Cloud Computing

# Cloud Computing

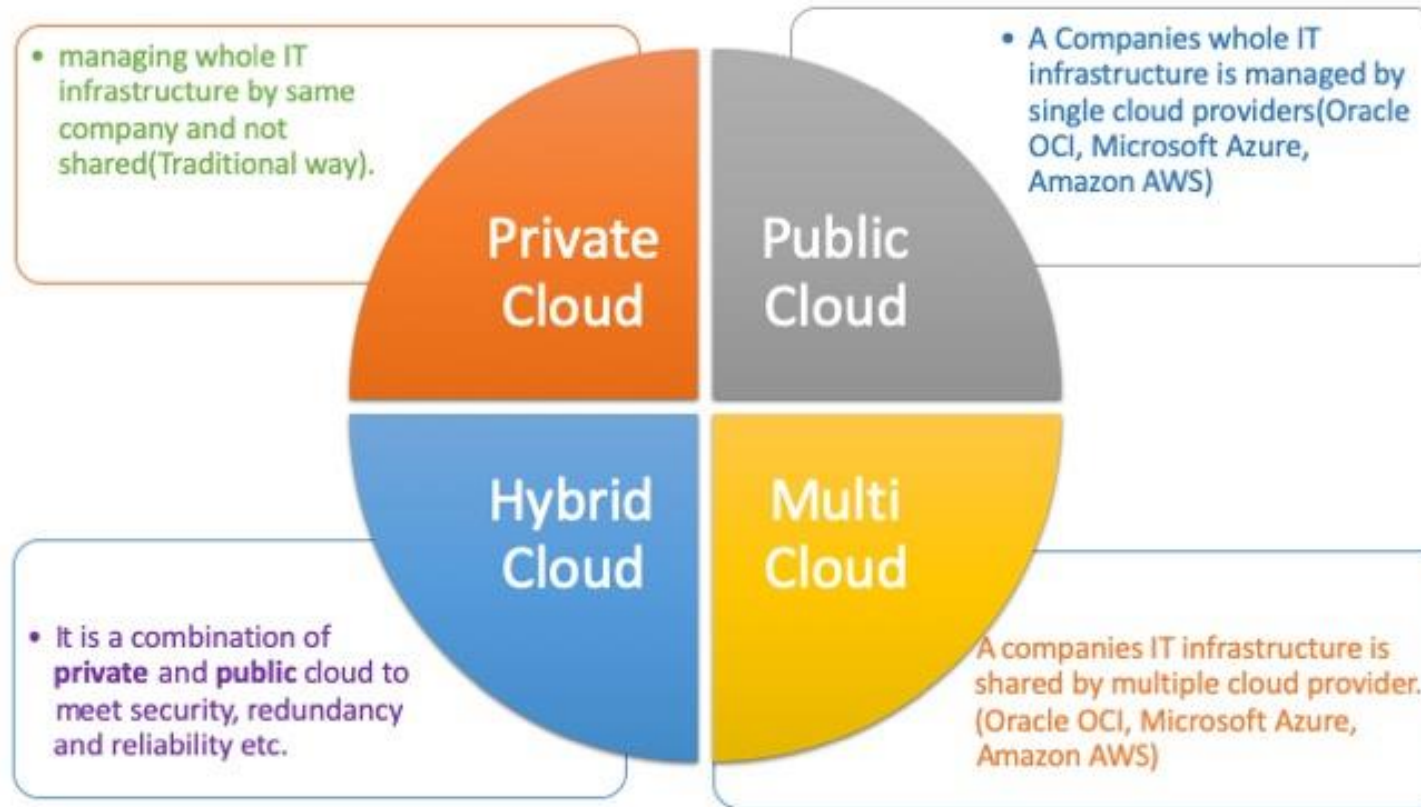
What is Cloud? And What is Cloud Computing?

## Definition:

Cloud computing is the **on-demand delivery of IT resources** over the internet with **pay-as-you-go pricing**. It allows users to rent computing services instead of buying and maintaining physical hardware.



OPENSIFT



Google Cloud



DigitalOcean

# Cloud Computing

## Deployment Models

Model	Description	Example Use Case
Public	Services delivered over internet and shared infrastructure	Startups, Scalable apps (e.g., Azure)
Private	Dedicated infrastructure for one organization	Banks, Government
Hybrid	Mix of public + private; integrated management	Disaster recovery, burst scaling
Multi-cloud	Use of multiple public cloud providers	Avoid vendor lock-in, optimize pricing

# Cloud Computing

## Cloud Service Models



### IaaS (Infrastructure as a Service)

- Provides virtualized computing resources: VMs, storage, networks
- Example: AWS EC2, Azure Virtual Machines
- You manage: OS, apps, runtime

### PaaS (Platform as a Service)

- Provides a platform to develop, run, and manage applications without managing infrastructure
- Example: Azure App Services, Google App Engine
- You manage: Apps and data

### SaaS (Software as a Service)

- Delivers software applications over the internet
- Example: Gmail, Microsoft 365, Salesforce
- You use the software; provider manages everything else

**HCL**

**tcs** TATA  
CONSULTANCY  
SERVICES



**accenture**

**kyndryl**

# Cloud Computing

## Infrastructure As Service (IAAS)

### ◆ What It Means:

You rent **infrastructure** — virtual machines, storage, networking, etc.

- You manage the OS, runtime, and applications.
- The cloud provider manages the **physical servers, virtualization, and networking**.

### 🔧 Example:

You get a **virtual server** from Azure, install your operating system, your database, and then deploy your web app on it.

### 💡 Real-World Analogy:

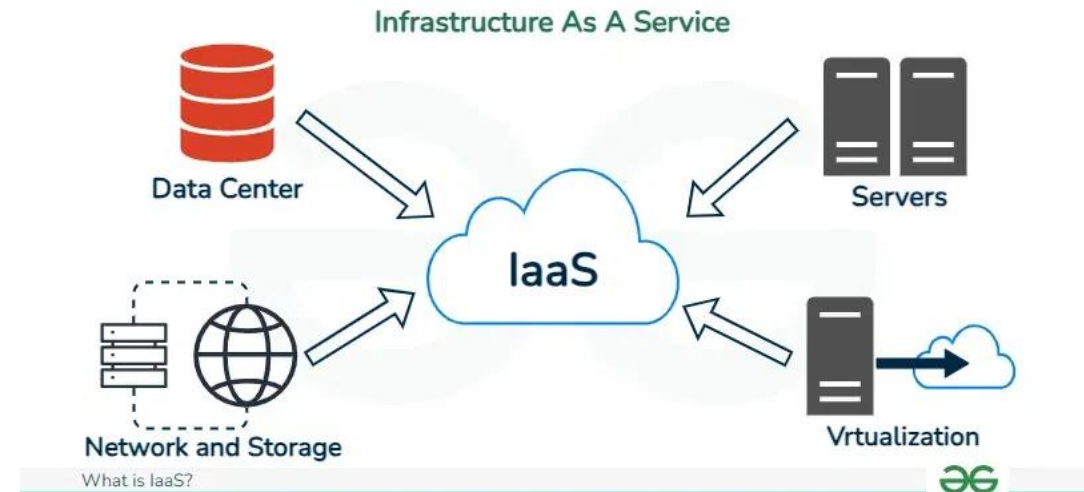
Imagine renting an empty house. You bring in your own furniture, appliances, do your own cooking and maintenance — but you don't own the building.

### ✅ Use Case:

- Developers who want full control over the server
- Hosting custom applications or legacy software

### ✂ Examples of IaaS:

- Microsoft Azure Virtual Machines
- AWS EC2 (Elastic Compute Cloud)
- Google Compute Engine



# Cloud Computing

## Platform As Service (PAAS)

### ◆ What It Means:

You rent not just infrastructure, but also a **platform** that includes the OS, runtime, and development tools.

- You only manage your **code and data**
- The provider manages the **OS, infrastructure, runtime, and tools**

### 🧰 Example:

You use **Azure App Service** to deploy a web app. You just upload your code — Azure takes care of the rest (hosting, scaling, patches).

### 💡 Real-World Analogy:

Renting a fully furnished apartment where you just bring your suitcase. You can live there and cook meals, but don't worry about repairs or appliances — it's all maintained for you.

### ✅ Use Case:

- Quick web app deployments without worrying about servers
- Focus on coding, not setup

### ✂ Examples of PaaS:

- Azure App Service
- Google App Engine
- Heroku
- AWS Elastic Beanstalk

# Cloud Computing

## Software As Service (SAAS)

### ◆ What It Means:

- You **use a complete software product** hosted and managed by someone else.
- You don't worry about the infrastructure or the platform — you just **use the application via browser or app**.

### 📁 Example:

You log into **Outlook, Gmail, or Microsoft Teams** — you're using a SaaS product. No setup, no installations.

### 💡 Real-World Analogy:

Like eating at a restaurant — everything is prepared and served to you. You just enjoy the food (features), no cooking or cleaning.

### ✅ Use Case:

- Email, file storage, CRM, team collaboration
- End users and businesses who need ready-to-use tools

### ✂ Examples of SaaS:

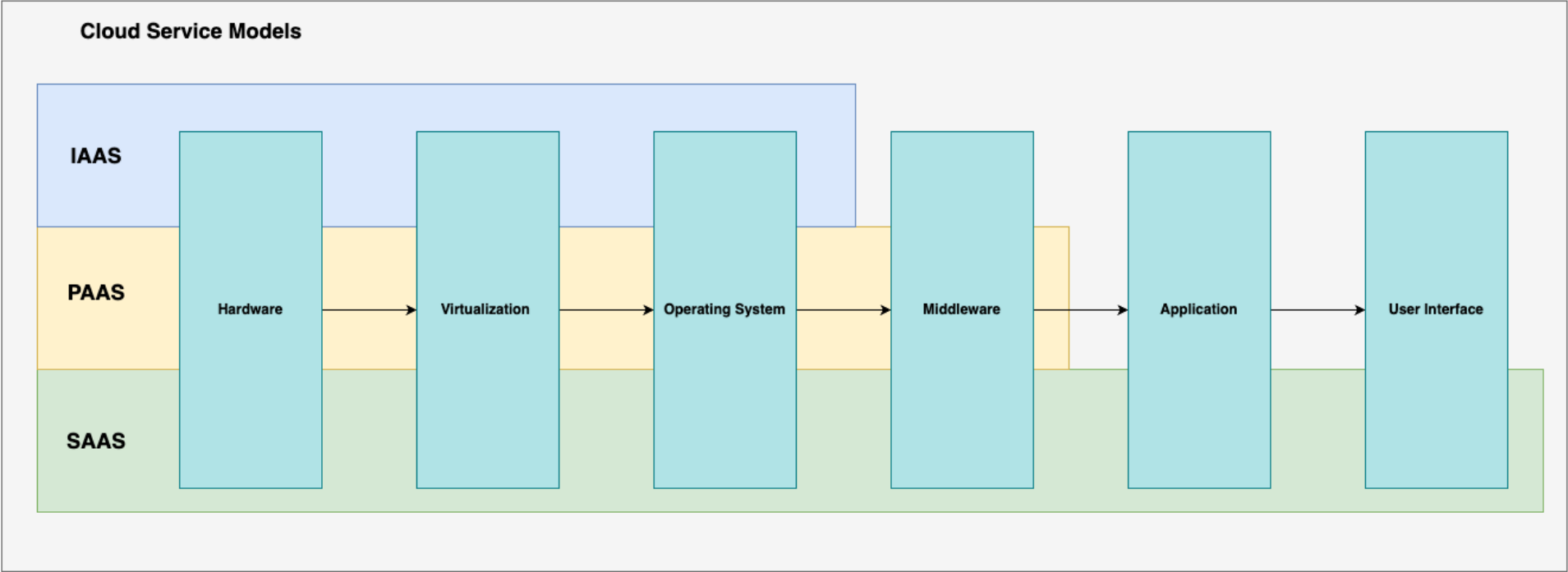
- Microsoft 365
- Google Workspace
- Dropbox
- Salesforce
- Zoom



# Cloud Computing

## Cloud Service Models

Feature	IaaS	PaaS	SaaS
Managed By You	App, data, runtime, OS	App & data	Just using the software
Control Level	Highest	Medium	Lowest
Setup Time	Long	Medium	Almost none
Flexibility	Maximum	Moderate	Minimal
Example	Azure VM	Azure App Service	Microsoft 365



# Cloud Computing

## Cloud Security – Service Level Agreements (SLA)

### What is it?

An SLA is a **formal agreement** between the cloud provider and the customer that defines:

- **Uptime (availability)** expectations
- **Response time** for issues or outages
- **Support commitments** and escalation paths

### Example:

Azure offers a **99.9% uptime SLA** for Virtual Machines.

That means the VM is guaranteed to be available at least **99.9% of the time** in a given month (less than ~45 minutes of downtime).

### Why SLAs matter:

- Helps businesses plan for redundancy (e.g., use multiple Availability Zones)
- Defines accountability and compensation if guarantees are not met



# Cloud Computing

## Cloud Security – Identify & Access Management (IAM)

### Identity & Access Management (IAM)

#### What is IAM?

- IAM helps organizations **securely manage who has access** to what resources in the cloud. It's the gatekeeper of your cloud environment.

#### Key IAM Concepts:

##### Role-Based Access Control (RBAC)

- Assigns permissions to users based on their **role** (e.g., Admin, Developer, Reader)
- You don't give access directly to users — you give it through **roles**
- Example: A "Reader" can view a VM but can't stop or delete it

##### Principle of Least Privilege

- Users should have **only the minimum access needed** to do their job — nothing more
- Helps reduce the risk of mistakes or breaches

##### Multi-Factor Authentication (MFA)

- Adds an **extra layer of security** beyond just username & password
- User must also verify via:
  - SMS code
  - Mobile app (e.g., Microsoft Authenticator)
  - Biometric login (e.g., fingerprint)

#### Why IAM is Critical

- Prevents **unauthorized access**
- Supports compliance (e.g., ISO, GDPR)
- Essential for secure collaboration in DevOps teams



# Cloud Computing

## Identity & Access Management



### What is IAM (Identity and Access Management)?

- Framework to manage **who can access what** in the cloud.
- Ensures **secure, controlled, and auditable access** to resources.
- Manages:
  - **Identities** (users, apps, services)
  - **Authentication** (verify identity)
  - **Authorization** (grant/deny access)



### Why IAM is Critical in Cloud:

- Cloud is accessible from anywhere → strong access control is a must.
- Prevents unauthorized access, accidental deletions, security breaches.
- Helps with **compliance, accountability, and operational security**.



### Core Components of IAM:

Component	Purpose
<b>User</b>	Individual identity (e.g., developer)
<b>Group</b>	Collection of users with similar permissions
<b>Role</b>	Defined set of permissions for tasks
<b>Policy</b>	Rules that allow/deny actions
<b>Permissions</b>	Fine-grained controls (e.g., read/write/delete access)



### Example:

#### Rahul, a developer:

- Identity created
- Added to “Dev Team” group
- Assigned a role that allows code push but blocks production deletes

# Cloud Computing

## Identity & Access Management

### What is IAM (Identity and Access Management)?

- **Azure:** Uses Azure AD + RBAC (roles like Reader, Contributor)
- **AWS:** JSON-based policies for users, groups, and roles
- **GCP:** Project-level roles (basic, predefined, custom)

### IAM Best Practices:

- ✓ Grant **least privilege** access
- ✓ Use **roles** over individual permissions
- ✓ Enable **Multi-Factor Authentication**
- ✓ **Review** access regularly
- ✓ Use **managed identities/service accounts** for apps
- ✓ Monitor access logs for suspicious activity

### Summary:


IAM Concept	Purpose
Authentication	Who you are
Authorization	What you're allowed to do
RBAC	Role-based access
MFA, Logs	Boost security and visibility

# Cloud Computing


## Cloud Offerings

Category	Services / Examples	Platform(s)
Compute	Azure VM, AWS EC2, AKS, EKS, Azure Functions, Lambda	Azure, AWS
Storage	Azure Blob, S3, Azure Files, EBS (block storage)	Azure, AWS
Database	Azure SQL, Cosmos DB, RDS, DynamoDB	Azure, AWS
Developer Tools	Azure DevOps, GitHub, Jenkins, CI/CD Pipelines	Azure, GitHub, Jenkins
Security	IAM, Azure Key Vault, AWS KMS, NSGs, DDoS Protection	Azure, AWS
Networking	Azure VNet, Load Balancers, VPN Gateway, AWS VPC	Azure, AWS
Media	Azure CDN, AWS CloudFront, Encoding Services	Azure, AWS
Mobile	Azure Notification Hub, Firebase Cloud Messaging (FCM)	Azure, Firebase
Integration	Azure Logic Apps, Service Bus, AWS Step Functions, EventBridge	Azure, AWS


## Cloud Services Overview




**Compute**  
VMs (Azure VM, S2)  
Containeres  
Serveriess  
(Azure Functions, Lambda)




**Storage**  
Object (BloB)  
File  
Block




**Database**  
SQL (Azure)  
NoSQL  
Cosmos DB




**Developer Tools**  
DevOps Services  
GitHub  
CI/CD Pipelines




**Security**  
IAM  
Key Vaults  
Firewalls  
DDoS Protection



**Networking**  
VNet  
Load Balancers  
VPN Gateway



**Mobile**  
Push  
Notification Hub



**Integration**  
Logic Apps  
Service Bus

# 03 Agile

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

Agile Project Management

# Agile





# Agile

## Introduction to Agile

### What is Agile?

- Agile is a project management framework that emphasizes iterative development, continuous improvement, and collaboration to adapt to changes quickly
- It's a way of working that focuses on delivering working software in short cycles (sprints) and integrating user feedback throughout the process
- Agile prioritizes individuals and interactions, working software over documentation, customer collaboration, and responding to change over following a strict plan.

### What does Agile Manifesto Say?

*“We are uncovering better ways of developing software by doing it and helping others do it.”*

***Individuals and interactions*** over processes and tools

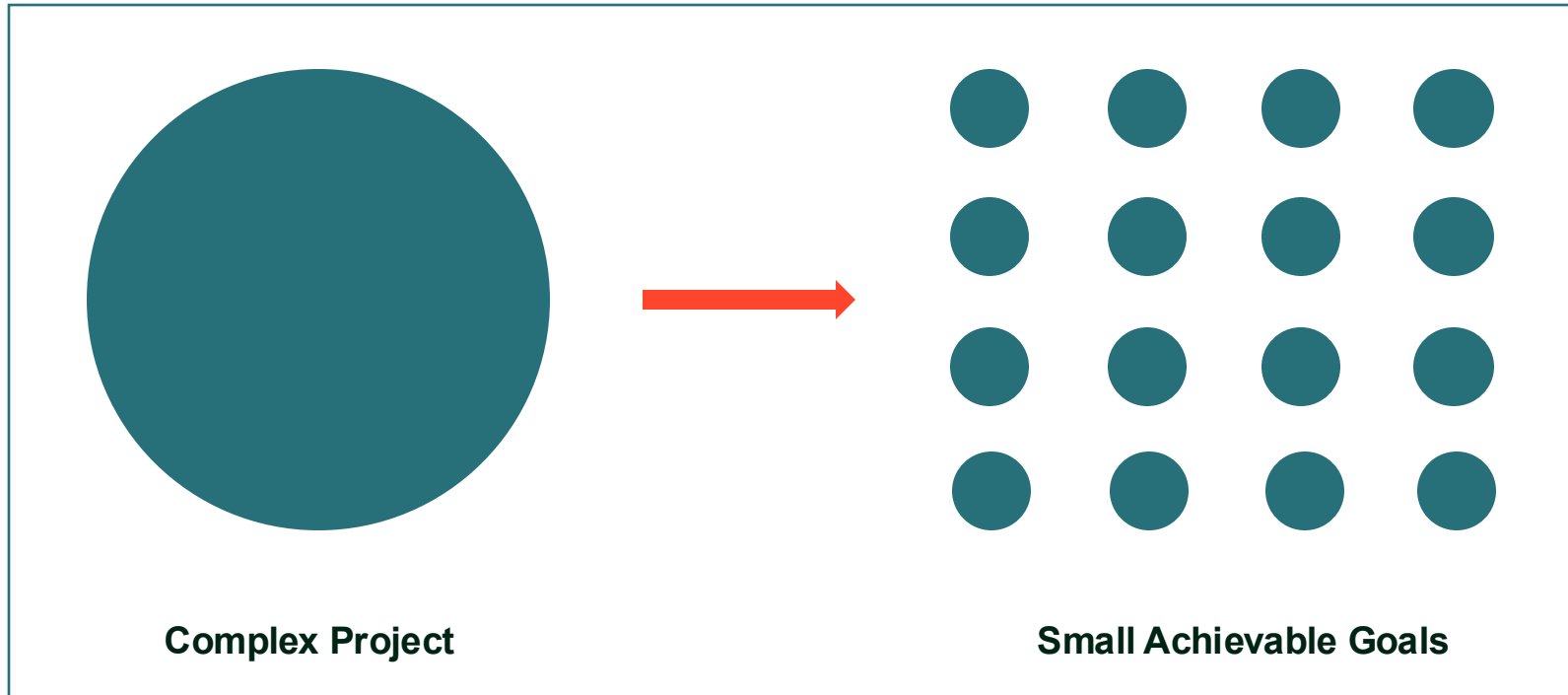
***Working software*** over comprehensive documentation

***Customer collaboration*** over contract negotiation

***Responding to change*** over following a plan

# What is Agile Software Development?

- It is an umbrella It is an umbrella term for a set of frameworks and practices that break down complex projects into small manageable goals.



- Ability to adapt and change
- Very flexible
- Quickly adapts to changes
- Quick launch of product
- Enables fast decision-making
- Improved communication
- Emphasis on continuous improvement

# Agile

## How does Agile Work?

Now that we know the basics of Agile development, let's take a more in-depth look at how it works. We can break the Agile process down into three main stages:

1. Preparation
2. Sprint Planning
3. Sprint

### Preparation

In the preparation stage, the product owner creates a backlog of features they want to include in the final product. This is known as the product backlog. Then, the development team estimates how long each feature will take to build.

### Sprint planning

The sprint planning meeting is where the team decides which features from the product backlog they are going to work on during the sprint.

A sprint is a set period (usually two weeks) during which the development team must achieve a specific goal. The team also decides how many of each type of task they can complete during the sprint.

For example, the team may decide they can complete three coding tasks, two testing tasks, and one documentation task during the sprint. This information is then added to the sprint backlog.

### Sprint

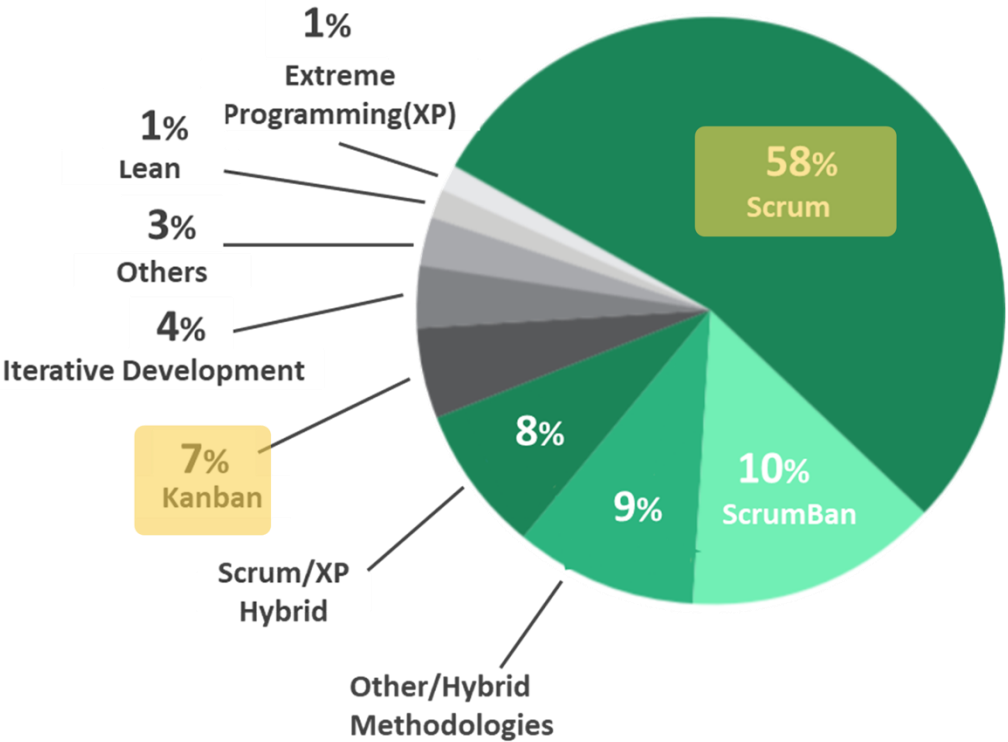
During the sprint, the team works on completing the tasks in the sprint backlog. They may also come across new issues to address. If this happens, they will add these issues to the product backlog and prioritize them accordingly. At the end of the sprint, the development team should have completed all features in the sprint backlog.

If not, the team will carry them over to the next sprint. The team then holds a sprint review meeting where they demo completed features to the product owner and stakeholders. They also discuss what went well during the sprint and how they could improve their next one.

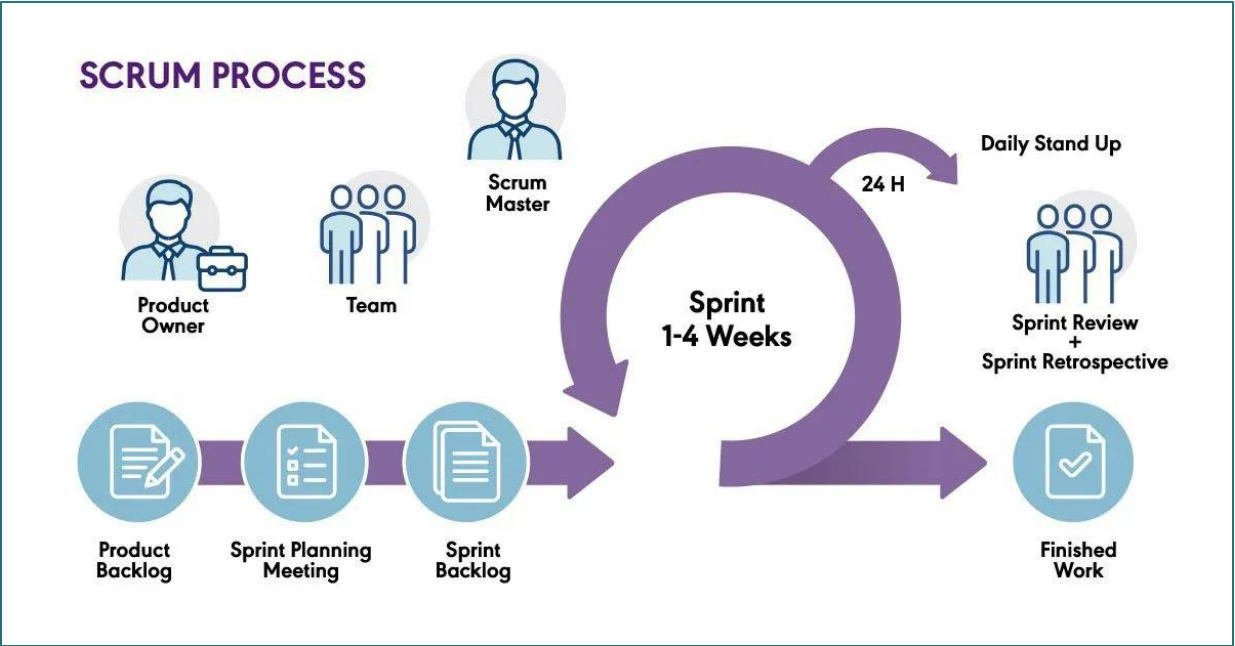
Finally, the team holds a retrospective meeting, where they reflect on what went well and what didn't go so well during the sprint. They then create a plan of action for addressing these issues in future sprints. This feedback loop helps to ensure that each sprint is more successful than the last.

# Agile

## Agile Frameworks & Agile Methodologies



Source: VersionOne's 14<sup>th</sup> Annual State of Agile Report



# Agile

## Scrum & Kanban – Similarities

### Pull System

With pull system you create a workflow where work is pulled only if there is a demand for it.

### Limit Work-in-progress (WIP)

Limiting WIP will allow teams to identify bottlenecks and improves throughput

### Break Down Complex Tasks

They allow large and complex tasks to be broken down and completed efficiently

### High Value on Continual Improvement

They place a high value on continual improvement, optimization of the work and the process



**Pull System**



**Limit WIP**



**Break down  
complex tasks**



**Value Continual  
Improvement**

# Agile

## SCRUM Process

# The SCRUM Process

*The term 'Scrum' was first introduced by professors Hirotaka Takeuchi and Ikujiro Nonaka in their article "The New New Product Development Game" at Harvard Business Review.*

*They borrowed the name 'Scrum' from the game of rugby, to stress the importance of teamwork to deal with a complex problem.*

In Scrum, you break down the phases of your project into smaller pieces that can be completed by a cross-functional team within a prescribed time period (called a **sprint**).

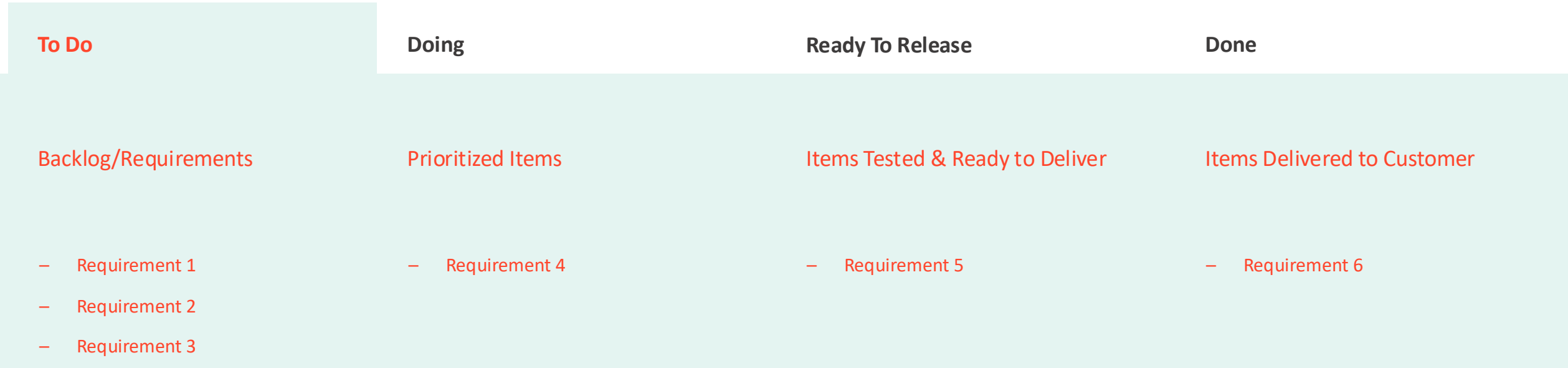
Once the sprint begins, you aren't allowed to add any new requirements.



The duration of the sprint in scrum is anywhere from **2 to 4 weeks to almost a month**

# Agile

## Kanban Process



Kanban ***focuses on maintaining a continuous task flow and continuous delivery***. At the same time, the team is never given more work than it can handle. This is accomplished through the two primary principles of Kanban:

**Visualize your work, Limiting Work-in-progress**

# Knowledge Check





# Time for some Fun!!

Quick Knowledge Check

<https://www.menti.com/al1uy64auit6>



# 04 DevOps

# DevOps

## What is DevOps

DevOps is not a tool or a software, it's a culture that you can adopt for continuous improvement. It will help you to bring your Developer Team and Operations Team on the same page, allowing them to work together with ease.



@rahuldighe

01

Flow : The principles of Flow, which accelerate the delivery of work from Development to Operations to our customers

02

Feedback : The principles of Feedback, which enable us to create ever safer system of work

03

Continual Learning and Experimentation : which foster a high-trust culture and a scientific approach to organizational improvement as part of our daily work

# First Way

# FLOW

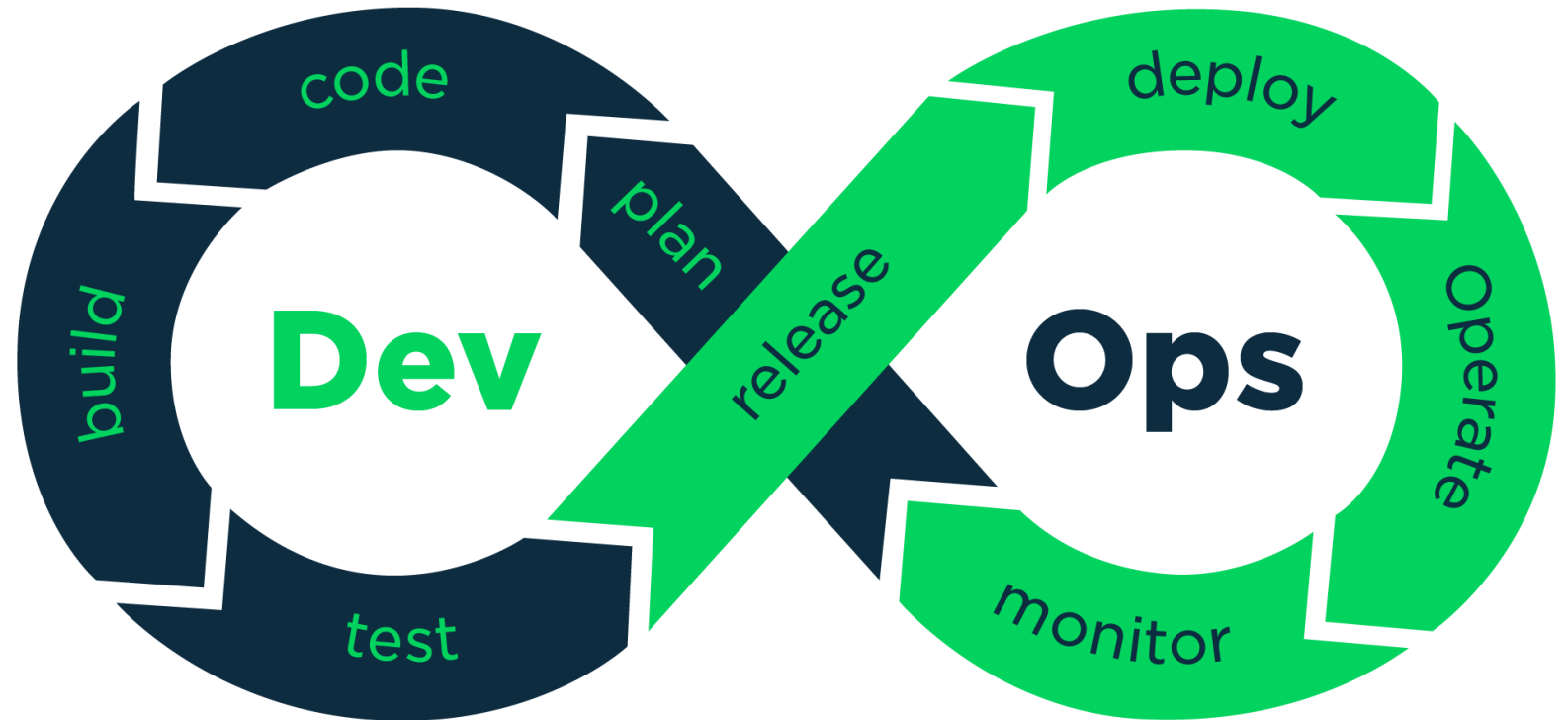
The first principle/way says we need to accelerate the work from development to operation, and then to our customers.

# DevOps

The First Way: Flow



- Understand the flow of work
- Always seek to increase flow
- Never pass a known defect downstream
- Never allow local optimization to cause global degradation
- Achieve a profound understanding of the system



# DevOps

## Continuous Integration (CI)

Continuous integration is a development practice that requires developers to integrate code into a shared repository on a daily basis.

### What is Continuous Integration (CI)?

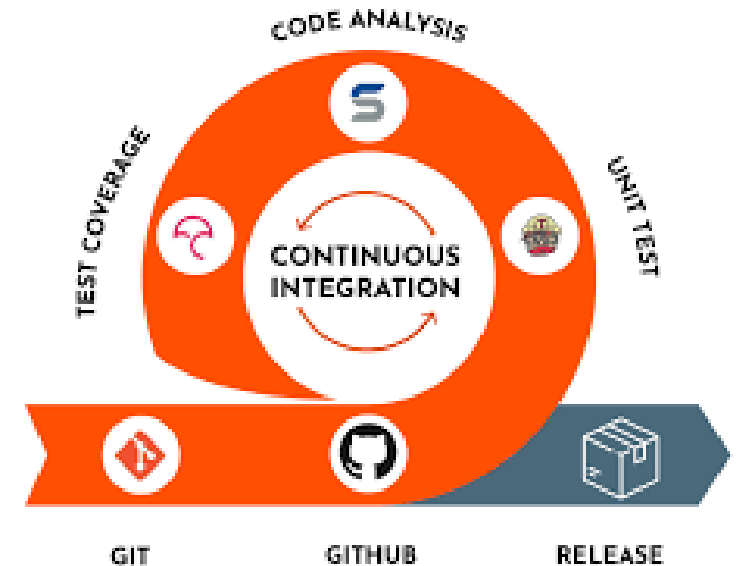
**Continuous Integration (CI)** is a **software development practice** where developers **frequently integrate** (merge) their code changes into a central repository — often multiple times a day.

Each time code is integrated, automated builds and tests are run to verify that the new code doesn't break the existing system.

### Why is CI important?

- Catches bugs early in development
- Encourages small, incremental changes instead of risky large updates
- Reduces integration headaches later in the process
- Keeps code in a “working” state at all times

**Analogy:** Think of CI like editing a shared Google Doc — everyone makes changes regularly, and the system checks immediately if something breaks the document formatting.



# DevOps

## Continuous Delivery (CD)

Continuous delivery is a methodology that focuses on making sure software is always in a releasable state throughout its lifecycle.

### What is Continuous Delivery (CD)?

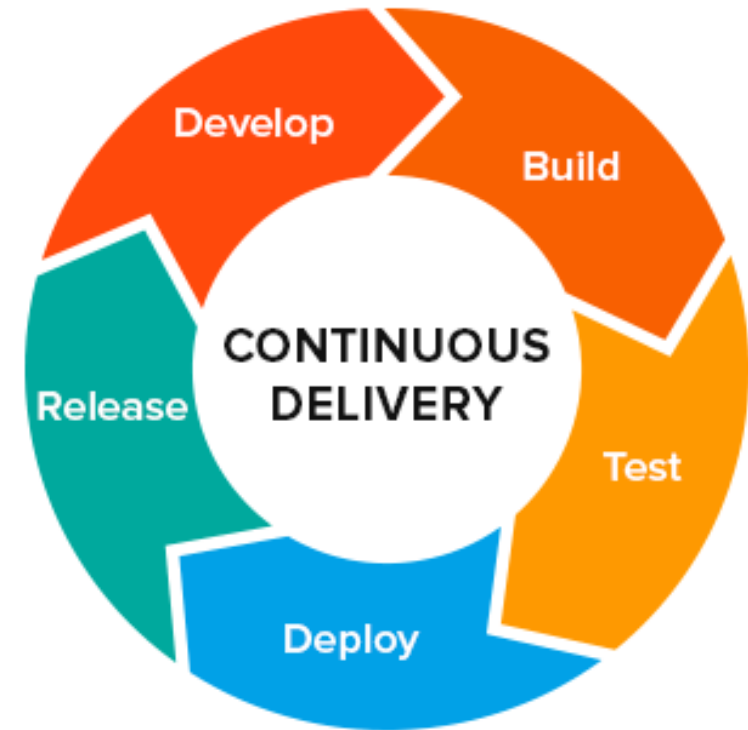
**Continuous Delivery** is the **next step after CI**. It means:

- Once your code passes tests in CI,
- It is **automatically packaged and made ready for deployment**
- You can deploy to production at **any time** with just a click

In simpler terms:

- CI = Code is always **tested**
- CD = Code is always **deployable**

With Continuous Delivery, you don't always deploy immediately, but your system is always in a deploy-ready state.





# DevOps

## Continuous Deployment

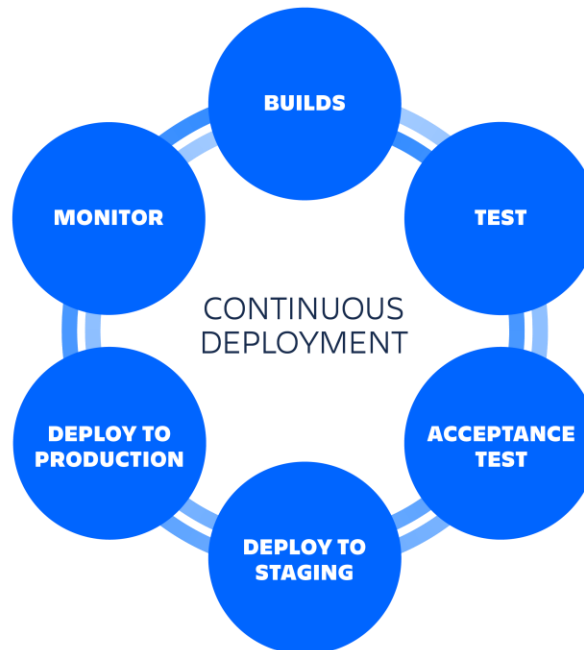
## ✓ And What is Continuous Deployment?

While **Continuous Delivery** prepares code for deployment, **Continuous Deployment** takes it one step further — **it actually deploys the code automatically** to production after passing tests, without any manual approval.

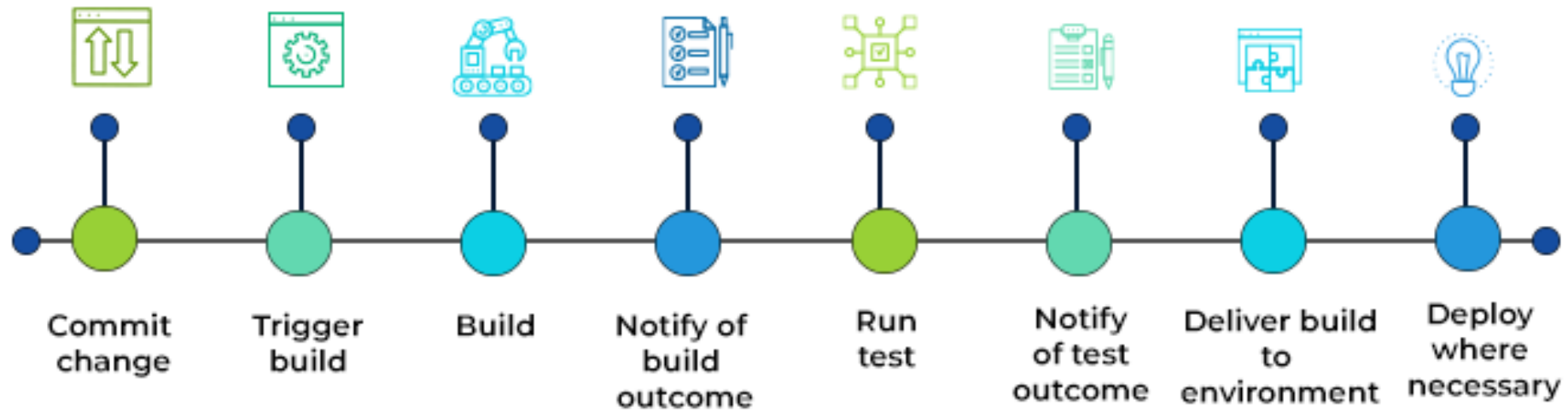
So:

- **CI** = Test the code regularly
- **CD (Delivery)** = Prepare the code for deployment
- **CD (Deployment)** = Automatically deploy to live environment

**Note:** Some teams stop at Continuous Delivery because they want a human to approve production changes. Others go full auto with Continuous Deployment.

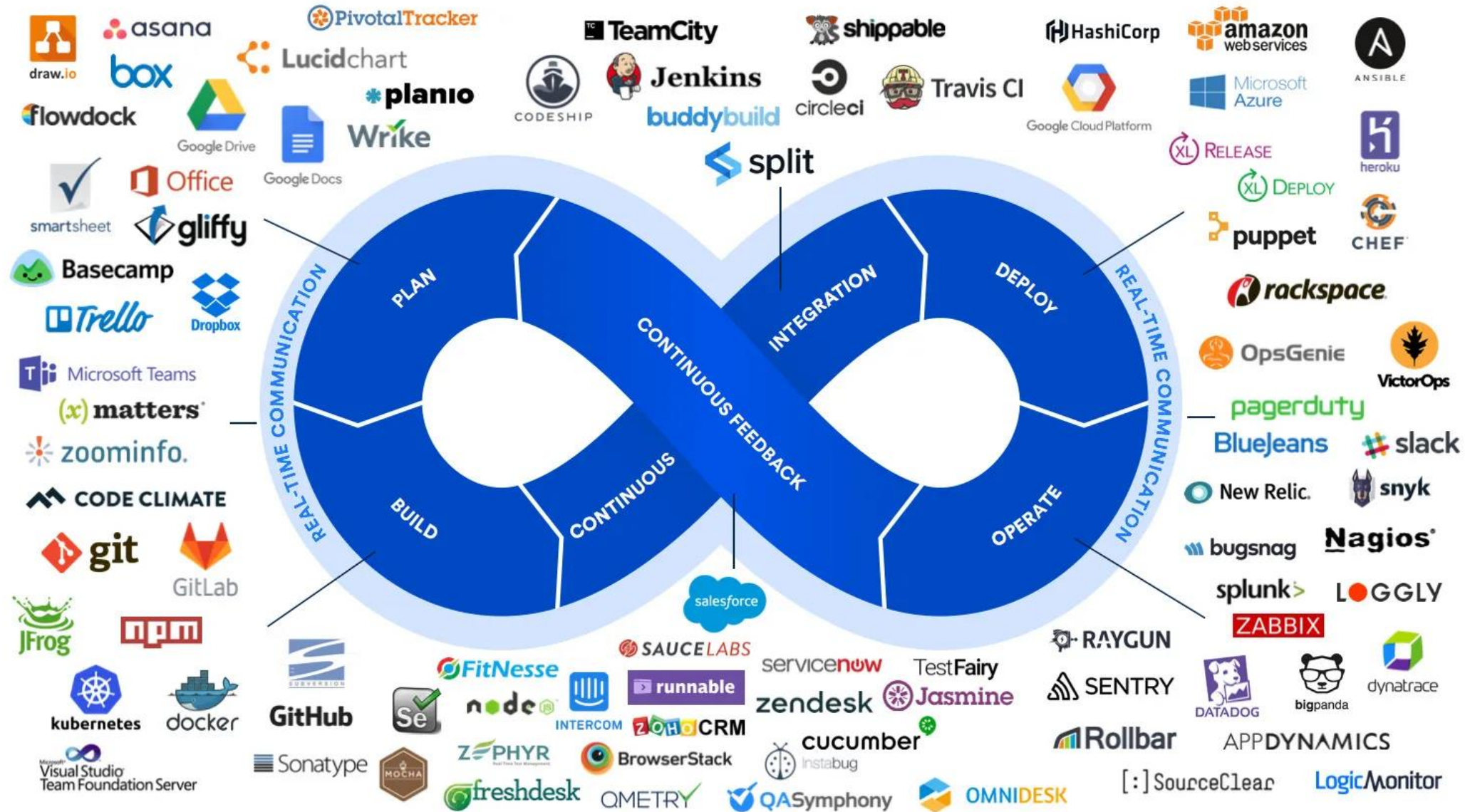


## CI/CD PIPELINE



# DevOps

## Tools



# 05 DevOps Basic Tools

# DevOps Basic Tools

## Source Code Management

### Source code management

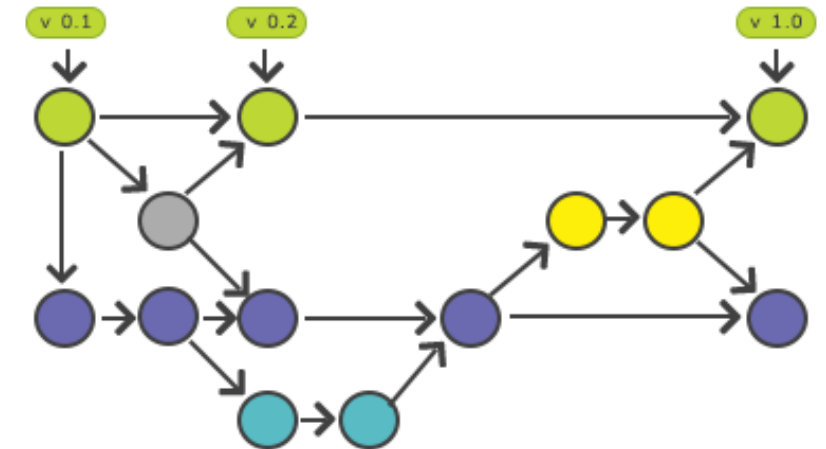
Source code management (SCM) is used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. SCM is also synonymous with Version control.

As software projects grow in lines of code and contributor head count, the costs of communication overhead and management complexity also grow. SCM is a critical tool to alleviate the organizational strain of growing development costs.

### The importance of source code management tools

When multiple developers are working within a shared codebase it is a common occurrence to make edits to a shared piece of code. Separate developers may be working on a seemingly isolated feature, however this feature may use a shared code module. Therefore developer 1 working on Feature 1 could make some edits and find out later that Developer 2 working on Feature 2 has conflicting edits.

Before the adoption of SCM this was a nightmare scenario. Developers would edit text files directly and move them around to remote locations using FTP or other protocols. Developer 1 would make edits and Developer 2 would unknowingly save over Developer 1's work and wipe out the changes. SCM's role as a protection mechanism against this specific scenario is known as Version Control.



# DevOps Basic Tools

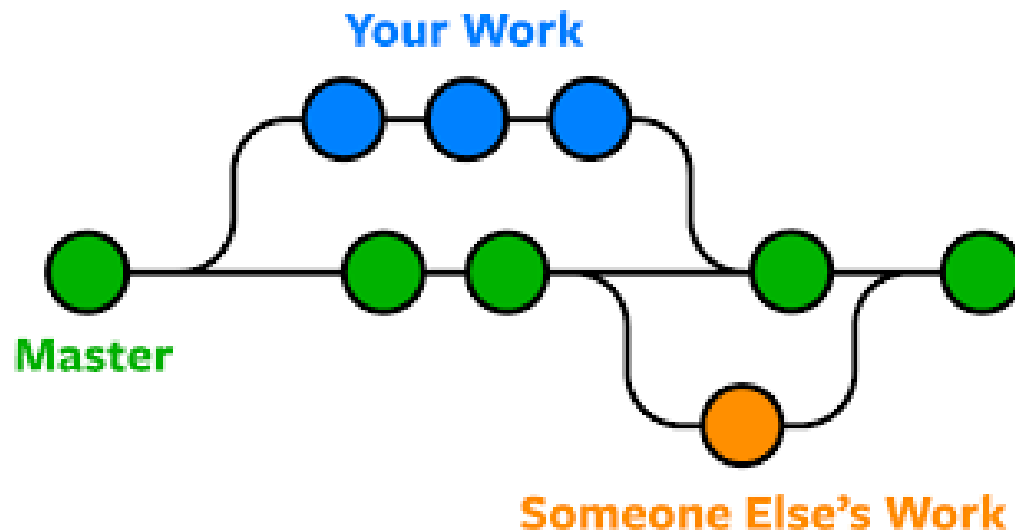
## GIT



## What is Git?

*"By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel."*

- A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).
- Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.
- In addition to being distributed, Git has been designed with performance, security and flexibility in mind.

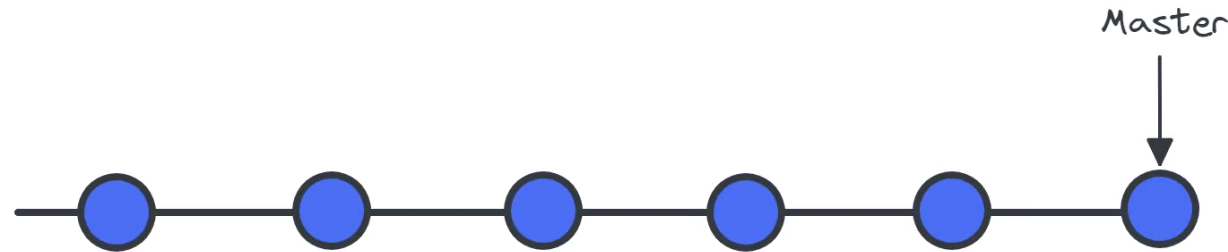


# DevOps Basic Tools

## GIT

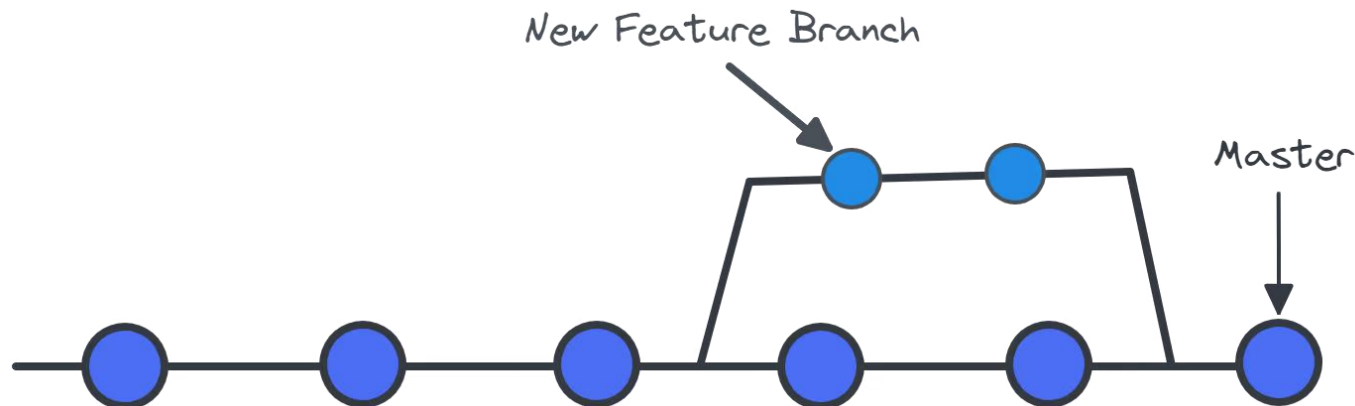
### How Does Git Work?

Git stores your files and their development history in a local repository. Whenever you save changes you have made, Git creates a commit. A commit is a snapshot of current files. These commits are linked with each other, forming a development history graph, as shown below. It allows us to revert back to the previous commit, compare changes, and view the progress of the development project. The commits are identified by a unique hash which is used to compare and revert the changes made.



### Branches

The branches are copies of the source code that works parallel to the main version. To save the changes made, merge the branch into the main version. This feature promotes conflict-free teamwork. Each developer has his/her task, and by using branches, they can work on the new feature without the interference of other teammates. Once the task is finished, you can merge new features with the main version (master branch)

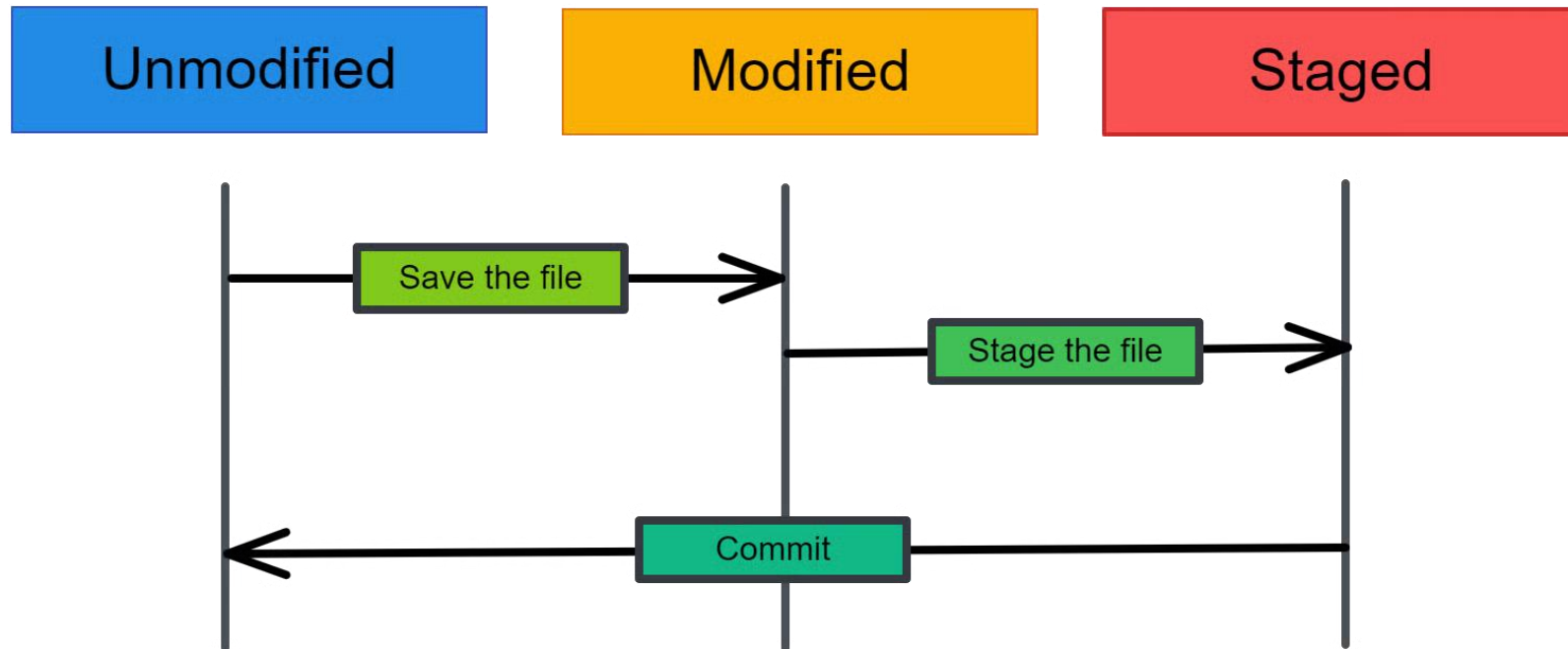


# DevOps Tools

## GIT

### Commits

There are three states of files in Git: **modified**, **staged**, and **commit**. When you make changes in a file, the changes are saved in the local directory. They are not part of the Git development history. To create a commit, you need to first stage changed files. You can add or remove changes in the staging area and then package these changes as a commit with a message describing the changes.





# DevOps Tools

## GITHUB

### What is GitHub?

GitHub is a for-profit company that offers a cloud-based Git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration.

GitHub's interface is user-friendly enough so even novice coders can take advantage of Git. Without GitHub, using Git generally requires a bit more technical savvy and use of the command line.

GitHub is so user-friendly, though, that some people even use GitHub to manage other types of projects – [like writing books](#).

Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

As a company, GitHub makes money by selling hosted private code repositories, as well as other business-focused plans that make it easier for organizations to manage team members and security. We utilize Github extensively at Kinsta to manage and develop internal projects.



# DevOps Tools

## GIT vs GITHUB

Sr.No.	Git	GitHub
1	Git is a software.	GitHub is a service.
2	Git is a command-line tool	GitHub is a graphical user interface
3	Git is installed locally on the system	GitHub is hosted on the web
4	Git is maintained by linux.	GitHub is maintained by Microsoft.
5	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7	Git was first released in 2005.	GitHub was launched in 2008.
8	Git has no user management feature.	GitHub has a built-in user management feature.
9	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11	Git provides a Desktop interface named Git Gui.	GitHub provides a Desktop interface named GitHub Desktop.
12	Git competes with CVS, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, Azure DevOps Server, etc.

# DevOps Tools

## Jenkins

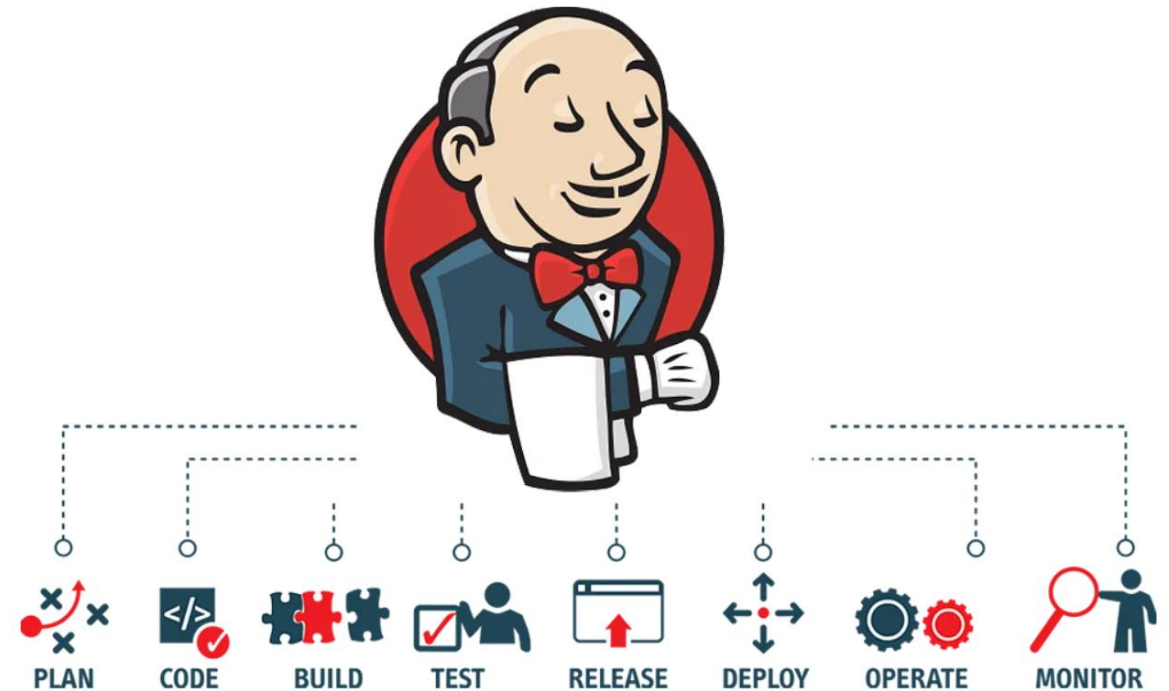
### What is Jenkins?

**Jenkins** is an **open-source automation server** widely used in software development. Its main job is to help automate the parts of software delivery: **building**, **testing**, and **deploying** code.

In short, it makes sure that developers can deliver high-quality software **faster and more reliably**. It's like a **robot assistant** for developers.

Every time someone makes a change in the code (e.g., pushes it to GitHub), Jenkins can automatically:

1. Pull the new code
2. Build it (e.g., compile if needed)
3. Run tests
4. Deploy the application to a test or production server



# DevOps Tools

## Jenkins

### Why Jenkins?

Before Jenkins (or similar tools), teams had to:

- Manually compile and build code
- Run tests by hand
- Deploy applications step-by-step

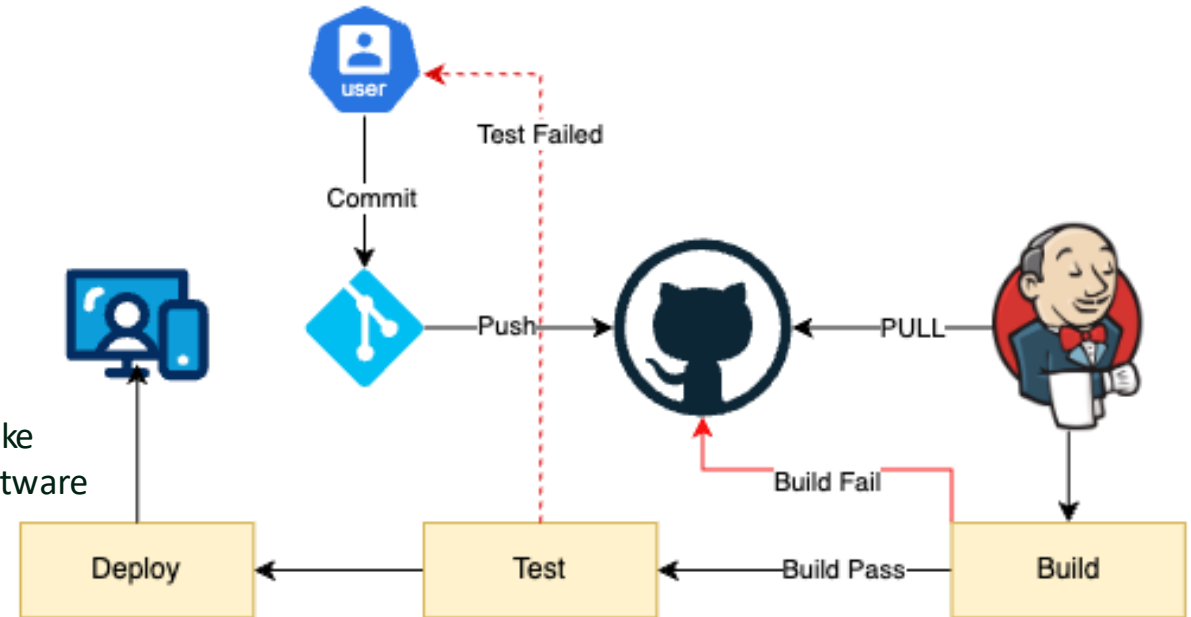
This manual process was:

- **Time-consuming**
- **Error-prone**
- Not scalable for large teams

Jenkins solves all this by offering **automation + flexibility**. It connects with tools like GitHub, Docker, AWS, Azure, etc., and uses “pipelines” to automate the entire software delivery process.

### Key reasons why Jenkins is widely used:

- **Automates repetitive tasks** like builds and deployments
- **Integrates with hundreds of tools** (Git, Maven, Docker, etc.)
- **Open source** and free
- **Extensible via plugins** – you can customize it to do almost anything in the CI/CD lifecycle



# DevOps Tools

## Containers

### What is a Container?

A container is a lightweight, standalone, and executable unit of software that includes everything needed to run an application:

- The code
- Runtime
- Libraries
- Dependencies
- System tools/settings

Think of a container as a mini computer bundled with an app — it runs the same anywhere, regardless of the environment.

### Example:

You build an app that runs perfectly on your laptop, but when you give it to a teammate, it crashes on their system due to missing packages or different OS. With a container, the entire environment is packaged, so it runs identically on:

- Your laptop
- Your colleague's machine
- Testing servers
- Cloud infrastructure (AWS, Azure, etc.)



# DevOps Tools

## Containerization



### What is Containerization?

**Containerization** is the process of packaging software code with all its dependencies into a container.

Before containerization, we had **Virtual Machines (VMs)**. But VMs are:

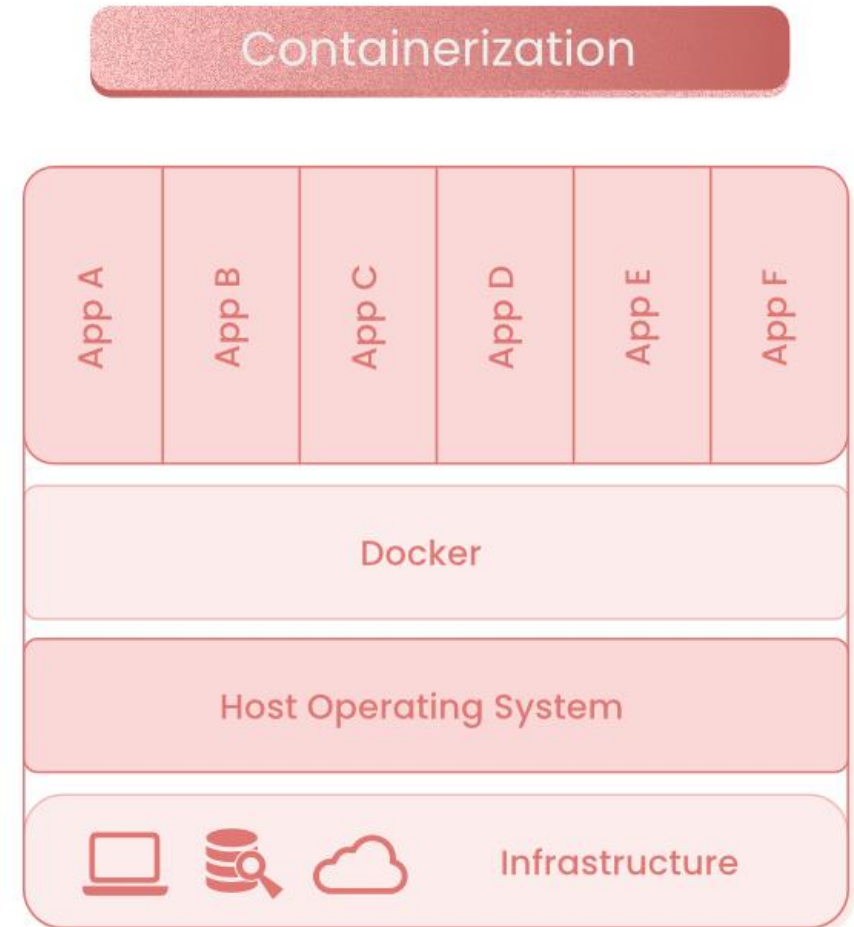
- Heavyweight (entire OS per app)
- Slower to start
- Resource-intensive

### Containers, in contrast:

- ✓ Share the host OS kernel
- ✓ Are faster and more lightweight
- ✓ Can be spun up or down in seconds

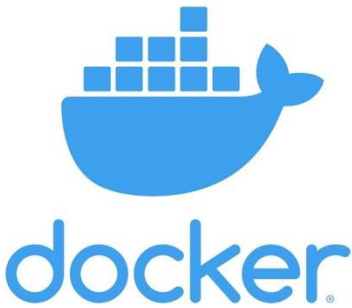
### VM vs Container Analogy:

Think of VMs like full apartments (with their own kitchens and bathrooms). Containers are like rooms in a shared flat — they have all essentials, but share the foundation (OS).



# DevOps Basic Tools

## Docker



### What is Docker?

**Docker** is the **most popular containerization platform**. It simplifies how developers:

- Create containers
- Manage them
- Distribute them (like sharing an app on a flash drive)

### So what does Docker do?

1. Lets you **build** an image (a blueprint of your app + environment)
2. **Runs containers** from that image
3. Helps you **share images** easily (via Docker Hub)
4. **Deploys containers** across machines, VMs, or cloud environments

### Benefits of Docker

#### Consistency Across Environments

Runs the same on dev, test, and production.

#### Lightweight

No full OS required — faster and less resource-hungry than VMs.

#### Portability

Build once, run anywhere: your laptop, servers, or cloud.

### Key Terms in Docker:

Term	Description
Image	A blueprint for creating containers (like a recipe)
Container	A running instance of an image (like a cooked dish)
Dockerfile	A set of instructions to create an image
Docker Hub	A cloud registry to store and share images
Docker Engine	The runtime that builds and runs containers

### Faster Development & Deployment

Containers start in seconds. Perfect for modern CI/CD pipelines.

#### Isolation

Each app runs in its own container — avoids conflicts and makes debugging easier.

#### Scalability

Docker works well with orchestration tools like Kubernetes to scale apps automatically.



# DevOps Tools

## Docker

### Real-World Examples of Docker Use

#### 1. Game Development Studio

Uses Docker to:

- Provide each developer the same build environment
- Avoid “it works on my machine” bugs
- Test across systems quickly

#### 2. FinTech Company

Runs multiple services (user auth, payments, statements) as **microservices** in separate Docker containers. Helps:

- Isolate failures
- Update one service without breaking others

#### 3. DevOps & QA Teams

Use Docker to:

- Quickly spin up test environments
- Run automated tests on different setups
- Deploy same container across dev → staging → prod

#### 4. Freelance Developer

Packages a web app with Flask + PostgreSQL + Nginx in Docker and shares the image with a client. They run it with **one command**, without installing anything else.



Dockerfile

build



Docker Image

run



Docker Container

 GEEK FLARE

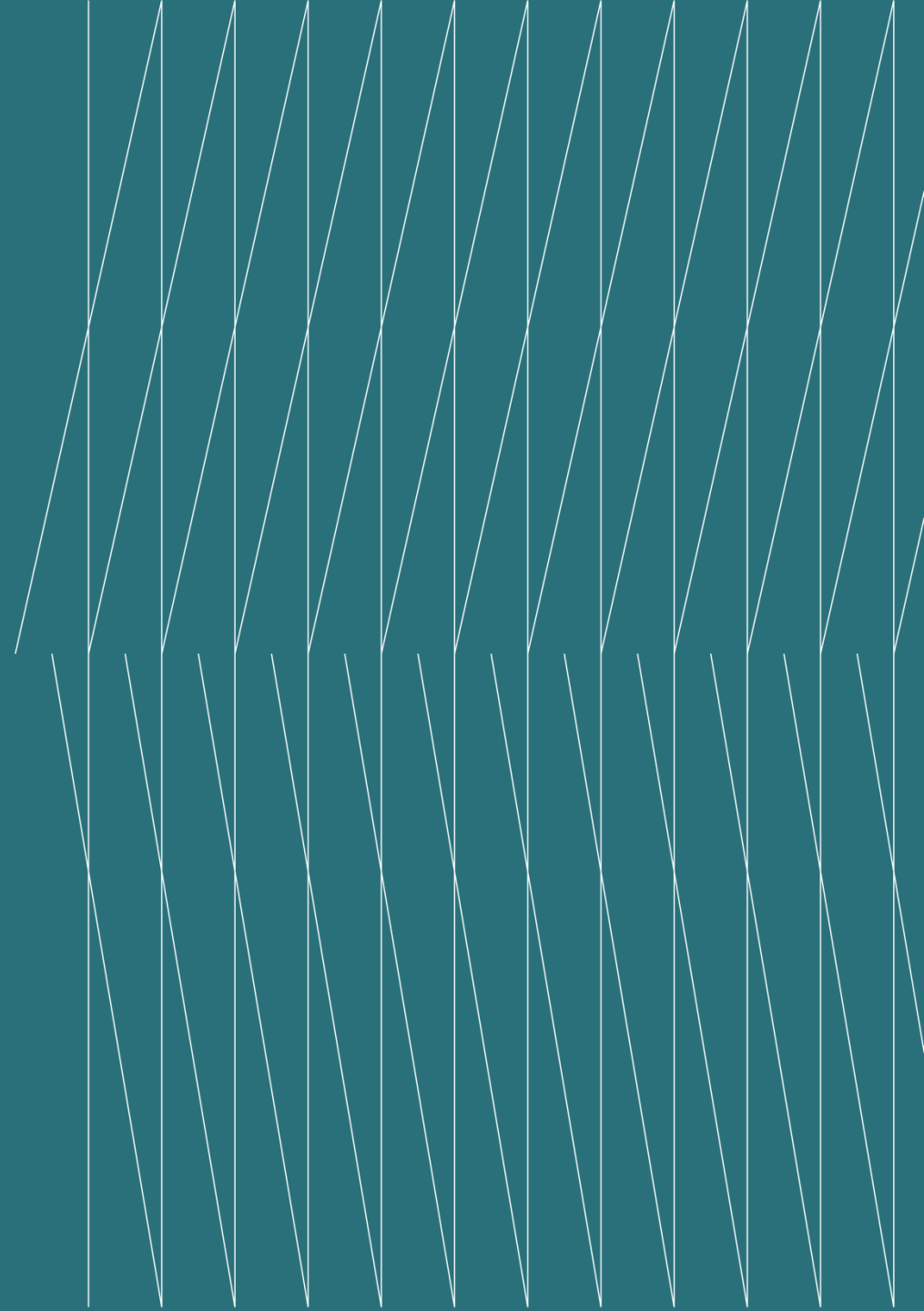


# DevOps Tools

## Docker

Concept	Purpose
<b>Container</b>	Isolated package with app + dependencies
<b>Containerization</b>	Process of creating containers
<b>Docker</b>	Tool to build, run, and manage containers
<b>Why Docker</b>	Fast, consistent, portable, and scalable
<b>Use Cases</b>	Dev/test environments, microservices, CI/CD, production apps

# Q&A



## Workshop Feedback Form

