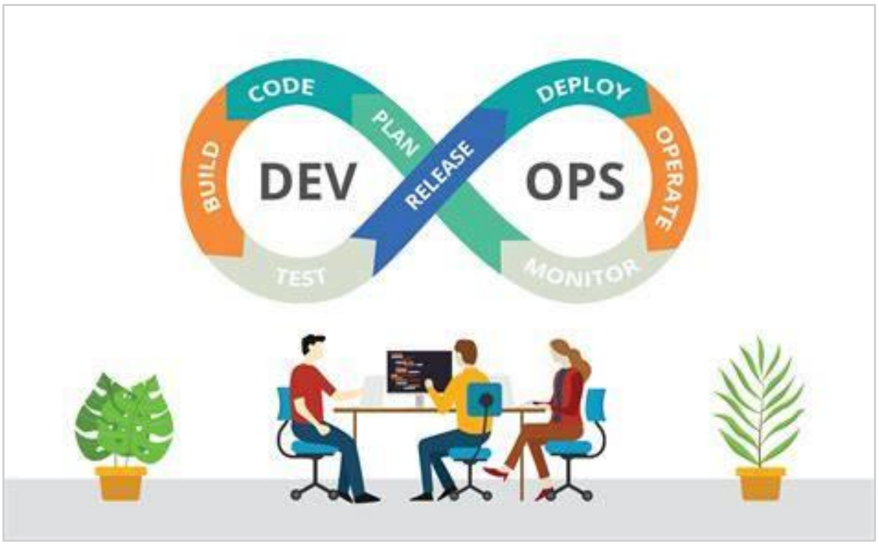
DevOPs

DevOps = Development + Operations

DevOPs is a culture/methodology/process in IT Industry to simplify the application delivery process to client/customer with high quality.

The main aim of DevOps is to establish the collaboration between the Development and Operation teams.



It's a never ending loop (CI/CD)

Development Team Phases:

1. **Plan** – Client requirements, Technology stack, Budget, Timeline
2. **Code** – Programming languages to be used acc. to the req.
3. **Build** – collecting all the files from multiple developers into Packages.
4. **Test** – To test the application for bugs and errors before deploying.

---App Release--

Operations Team Phases:

1. **Deploy** – Process of installing the application into the server.
2. **Operate** – In Use/Production.
3. **Monitor** – To check how the application is reacting. All about INCs and alerts etc.

AWS EC2 Instance

Server: Server is a powerful computer or system that provides resources, data, services, or programs to other computers. It serves the needs of other computers.

Types of Servers:

1. Web Server – Delivers web pages (Apache, Nginx, Microsoft IIS, GWS)
2. Database Server – Manages databases (MySQL, PostgreSQL, Oracle DB, MSSQL)
3. File Server – Stores and shares files (Sharepoint)
4. Application Server – Runs specific applications (Apache TOMCAT, IBM Sphere, Web Logic, F5 NGINX, Glass Fish)
5. Mail Server – Sends and receives emails (Microsoft Exchange, Postfix, Sendmail)
6. Authentication Server – Verifies user credentials and manages access control (LDAP, Radius, Active Directory (AD))
7. DNS Server (Domain Name System) – Translates Domain names into IP addresses (BIND Microsoft DNS)
8. Proxy Server – as Acts as an intermediary between clients and other servers to improve performance or security. (Content filtering, caching)
9. Firewall Server – Monitors and controls incoming/outgoing traffic based on security rules.
10. Cloud Server – Virtual servers hosted in the cloud (AWS EC2, Azure VM, Google compute engine) *on – demand computing resources.*

AWS EC2: Amazon Web Services Elastic compute cloud.

Creating an EC2 Instance = Creating a VM in AWS

Amazon EC2 (Elastic Compute Cloud) is a service provided by AWS that lets you run virtual servers in the cloud.

1. Scalable: You can increase or decrease the number of servers as needed.
2. Flexible: Choose your operating system, CPU, memory, storage, and networking.
3. Pay-as-you-go: You only pay for what you use.
4. Secure: Integrated with AWS security features like IAM (Identity and access Mgmt) and VPC (Virtual Private Cloud)

- EC2 is the most demanded service in AWS.
- EC2 is region specific.
- EC2 Instance's Minimum billing period is 1 hr.

To create a VM in AWS EC2 we require:

🔐 **Key Pair:**

- Used for **secure SSH access** to your instance.
- Consists of:
  - **Public key** (stored in AWS)
  - **Private key** (downloaded as .pem or converted to .ppk for PuTTY)
- Required for Linux (SSH) and Windows (to decrypt admin password for RDP).

🖫 **Volumes (EBS – Elastic Block Store):**

- Provides **persistent storage** for your EC2 instance.
- Acts like a **virtual hard drive**.
- You can attach multiple volumes.
- Free tier limit: Up to 30 GB.
- Max size per volume: Up to 16 TB.

🌐 **Network (VPC – Virtual Private Cloud):**

- Defines the **network environment** for your instance.
- Includes:
  - Subnets (public/private)
  - Route tables
  - Internet Gateway (for public access)
  - NAT Gateway (for private subnet internet access)

🛡 **Firewall (Security Group – SG):**

- Acts as a **virtual firewall**.
- Controls **inbound and outbound traffic**.
- Common rules:
  - Port 22 for SSH (Linux)
  - Port 3389 for RDP (Windows)
  - Port 80 for HTTP
  - Port 443 for HTTPS

Steps to create an Instance:

1. Name and tag
2. Select an AMI – Amazon Machine Image (An AMI contains the operating system, application server, and applications for your instance). AMI IDs differ from region to region. (Free tier eligible: AWS Linux)
3. Select any Instance Type (Free tier eligible: t3.micro)
4. Select the Key-Pair or create new (we'll get a .pem file downloaded)
5. Network and Security Group configurations.
6. Storage configurations and then review and launch the instance.

GIT (Global Information Tracker)

GIT is a Version-control system (VCS) that is widely used in the programming world.
It is used for tracking changes in the source code and collaborating with others during software development.
It was developed in 2005 by Linus Torvalds, the creator of the Linux operating system kernel.

GIT is also called as Software configuration Management or Source-code Management.
GIT is a 3rd Generation tool. It is a centralized VCS. Used to handle small to very large projects.

Term	Meaning
Repository (repo)	A folder where your project and its history are stored.
Commit	A snapshot of your code at a point in time.
Branch	A separate line of development (e.g., for testing new features)
Merge	Combining changes from one branch into another.
Push	Sending your changes to a remote server (like GitHub)
Pull	Getting the latest changes from the remote server.

GitHub: It is a remote repository hosting service. Other ex: Gitlab, BitBucket.

Life cycle of a GIT:

1. ***Working Directory:*** This is where you create or edit your files. Check the status of the working directory using – **git status.** You move files to the staging area using - **git add**

   ---*git add*---

2. ***Staging Area:*** Think of this as your **"ready to submit" folder**. It's a temporary holding area before officially saving the changes. Once you **commit** (save) your changes from the staging area, they go into your local repo.

   --*git commit*---

3. ***Local Repository:*** This is like your personal saved copy of the project. It keeps track of all versions you've saved so far. You **push** your local commits to the remote repo.

   --*git push*---

4. ***Remote Repository:*** You share your work with others (like pushing to GitHub).

- Clone a remote repo to your local machine using – **git clone**
- Pull latest changes from remote to local using – **git pull**
- Developers work on separate branches and raise **pull requests** to merge their changes into the main/master branch (from where the deployment starts).

Main/Master Branch:

- This is the **production-ready** branch.
- It should always contain **stable and tested code**.

Feature/Development Branches:

- Each developer works on their own branch (e.g., feature/login, bugfix/header, etc.).
- These branches are usually created from the main or develop branch.

Commits and Pushes:

- Developers commit their changes locally and push them to the remote repository (e.g., GitHub, GitLab).

Pull Requests (PRs):

- Once a feature or fix is complete, the developer raises a **Pull Request** (also called a Merge Request in GitLab).
- The PR is a request to merge their branch into the main (or sometimes develop) branch.

Code Review & CI/CD:

- The PR is reviewed by peers or leads.
- Automated tests (CI pipelines) may run to ensure code quality.
- Once approved, the PR is merged into the main branch.

Merge Strategies:

- Depending on the team's policy, merges can be:
  - **Squash merge** (combines all commits into one),
  - **Rebase and merge** (linear history),
  - **Merge commit** (preserves full history).

MAVEN

Maven = An Expert (In yiddish lang, 'meyvn' –"one who understands")

Maven is a **build automation and project management tool** primarily used for Java projects. It simplifies the process of building, packaging, managing dependencies for software projects. It uses a **Project Object Model (POM)** file (pom.xml) to manage dependencies and build configurations.

Maven tool is developed and maintained by the Apache Software Foundation (ASF)

- Builds your project (compiles code, runs tests, packages into JAR/WAR files)
- Manages dependencies (downloads libraries your project needs from a central repository)
- Standardizes project structure (uses a common directory layout)
- Supports plugins for tasks like code analysis, deployment, documentation, etc.

Key Concepts

- **POM (Project Object Model)**: The pom.xml file is the heart of a Maven project. It defines:
  - Project details (name, version, etc.)
  - Dependencies
  - Plugins
  - Build configurations
- **Repositories**:
  - **Local repository**: Where Maven stores downloaded dependencies on your machine.
  - **Remote repository**: Online sources like Maven Central.
- **Phases of Build Lifecycle**:
  - validate, compile, test, package, verify, install, deploy

JAVA: Java is a **high-level, object-oriented programming language** designed to be platform-independent. Its core philosophy is **"Write Once, Run Anywhere" (WORA)**—meaning compiled Java code can run on any system with a Java Virtual Machine (JVM).

Developed by James Gosling at **Sun Microsystems in 1990's.** Originally called **Oak**, it was renamed to **Java** after discovering "Oak" was already trademarked. The name "Java" was inspired by **Java coffee beans. (Java is an Indonesian Island)**

.java -----*to convert .java - JAVAC*----- >**bytecode (.class)** ----*to run .class – JVM*---- >**java o/p**

JAVAC – Java compiler converts the java code (high-level code) into bytecode (intermediate code) stored in .class file. [compilation process]

JVM – Java virtual Machine converts the bytecode into machine code (binary instructions) [Execution process]

Java Project: A Java project is a collection of related files and resources that together form a complete application or system written in Java.

It typically includes:

1. Source Code Files (actual Java classes and logic/ .java)
2. Compiled Files (generated by javac/.class)
3. Project Configuration Files (pom.xml – Maven), (build.gradle – Gradle)
4. Libraries / Dependencies (.jar/.war)
5. Resources (Imgs, Config files .properties/.xml)
6. Test code (unit tests written using frameworks like Junit)

JAR (Java ARchive) - Used to package Java applications, libraries, or components.
WAR (Web Application ARchive) - Used to package Java web applications for deployment on a web server (like Apache Tomcat).

Maven build lifecycle goals—these are standard phases that Maven executes in order to build and manage a Java project.

Goals	Description
Clean	Deletes the target/ directory to remove previous build artifacts.
Compile	Compiles the source code in src/main/java into .class files
Test	Runs unit tests in src/test/java using frameworks like JUnit.
Package	Packages the compiled code into a .jar or .war file as defined in pom.xml.
Install	Installs the packaged file into your local Maven repository (~/.m2/repository).
Deploy	Uploads the packaged file to a remote repository for sharing or deployment

Maven Project Creation Terms:

1. **archetype:** A template for generating a Maven project.
   Example: maven-archetype-quickstart creates a basic Java project with sample code and structure.

2. **groupId:** Represents the organization or group that the project belongs to. Usually follows a reverse domain name convention.

Example: com.example, org.apache.maven

3. **artifactId:** The name of the project or module. This becomes the name of the generated .jar or .war file.
   Example: my-app, inventory-service

4. **Packaging:** Defines the type of output Maven should produce.
   Common values: jar → for libraries or console apps
   war → for web applications
   pom → for parent projects or aggregator

<u>NEXUS</u>

Nexus is a **repository manager** developed by Sonatype, and it's widely used in DevOps pipelines to manage and distribute software artifacts like .jar, .war, .tar, Docker images, and more.

Key roles of Nexus:

- Artifact Repository: Stores build outputs like .jar, .war, .zip, etc.
- Dependency Proxy: Caches external dependencies (e.g., from Maven Central) to speed up builds and reduce internet reliance.
- Centralized Sharing: Acts as a central hub for teams to share internal libraries and components.
- Supports Multiple Formats: Works with Maven, npm, NuGet, Docker, PyPI, and more.
- CI/CD Integration: Easily integrates with tools like Jenkins, GitLab CI, Azure DevOps, etc.
- Security & Access Control: Manages who can upload/download artifacts and tracks versions.

➢ Nexus server runs on port number – 8081.

Snapshot Repositories: A Snapshot Repository is used to store development versions of software artifacts. These are not final releases, but rather work-in-progress builds that may change frequently.

Release Repositories: A Release Repository in Nexus is used to store final, stable versions of software artifacts. (Ready for production)

Unlike snapshot repositories, release artifacts are immutable—once published, they cannot be overwritten.

<u>JENKINS</u>

Jenkins is an open-source automation server widely used in DevOps for Continuous Integration (CI) and Continuous Delivery (CD). It helps automate the building, testing, and deployment of software projects.

🚀 Why Jenkins is Popular in DevOps:

Automates repetitive tasks like building code, running tests, and deploying applications.
Integrates with many tools: Git, Docker, Kubernetes, Ansible, Maven, Gradle, etc.
Extensible: Over 1,800 plugins available to customize pipelines.
Scalable: Can run on a single machine or distributed across multiple nodes.

Core Concepts:

Concept	Description
Job/Project	A task Jenkins performs (e.g., build a Java app).
Build	The execution of a job.
Pipeline	A script that defines the steps of CI/CD.
Node/Agent	A machine Jenkins uses to run jobs.
Executor	A slot for running builds on a node.

<u>ANSIBLE</u>

Ansible is an open-source automation tool used for **configuration management**, **application deployment**, **orchestration**, and **provisioning** of IT infrastructure. It's especially popular in DevOps because it simplifies complex tasks and helps manage large-scale environments efficiently.

Orchestration: If automation is about doing a single task automatically (e.g., installing a package), **orchestration** is about managing **multiple automated tasks** in a **specific order** across **multiple systems** to achieve a larger goal.

What is Ansible?

Agentless: No need to install anything on the target machines. It uses SSH to connect and run tasks.
Declarative Language: You describe the desired state of your systems using YAML in **Playbooks**.

Idempotent: Running the same playbook multiple times won't change the system if it's already in the desired state.

Key Components:

- **Inventory**: A list of servers (hosts) Ansible will manage.
- **Modules**: Units of work (e.g., install a package, start a service).
- **Playbooks**: YAML files that define tasks and roles.
- **Roles**: A way to organize playbooks into reusable components.
- **Facts**: System information gathered from hosts.

Ansible can orchestrate tasks like:

- Setting up a web server cluster
- Deploying code to multiple environments
- Rolling updates with zero downtime

You define these workflows in **Playbooks**, and Ansible ensures everything runs in the right order, on the right machines.

DOCKER

Docker is a key tool in DevOps and cloud-native development. It helps you **package applications** and their dependencies into **containers**, making them portable, consistent, and easy to deploy.

What is Docker?
Docker is a platform that uses containerization to run applications in isolated environments. A container includes everything needed to run an app: code, runtime, libraries, and system tools.

Why Docker is Useful in DevOps:

Consistency: Works the same in dev, test, and prod.
Portability: Runs on any system with Docker installed.
Isolation: Each container runs independently.
Speed: Containers start faster than virtual machines.
Scalability: Works well with orchestration tools like Kubernetes.

Key Docker Concepts:

Concept	Description
Image	A snapshot of an application and its environment.
Container	A running instance of an image.
Dockerfile	A script to build Docker images.
Docker Hub	A public registry to share Docker images.
Volumes	Persistent storage for containers.
Networks	Communication between containers.