Git

Git is a Devops Tool used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. Linus Torvalds created Git in 2005 for the development of the Linux kernel.

Scenario before Git:

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers.

Scenario after Git:

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers.

Features of Git :

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development

Docker :

Docker is a software platform that allows you to build, test, and deploy applications quickly.

Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

How Docker works

Docker works by providing a standard way to run your code. Docker is an operating system for containers. Similar to how a virtual machine virtualizes (removes the need to directly manage) server hardware, containers virtualize the operating system of a server. Docker is installed on each server and provides simple commands you can use to build, start, or stop containers.

Key Components of Docker

1. **Docker Engine**:
    - The core runtime that runs and manages containers.
    - Includes the following components:
        - **Daemon**: The background service that manages containers.
        - **CLI (Command Line Interface)**: Used to interact with the Docker daemon.
        - **REST API**: Provides programmatic access to Docker functionality.
2. **Docker Images**:
    - Lightweight, standalone, and executable packages that include the application code, runtime, libraries, and dependencies.
    - Acts as a blueprint for containers.
3. **Docker Containers**:
    - A runtime instance of a Docker image. It is a lightweight, isolated environment where the application runs.
4. **Docker Hub**:
    - A cloud-based registry where developers can store, share, and manage Docker images. It hosts both official and user-contributed images.
5. **Docker Compose**:
    - A tool to define and manage multi-container applications using a docker-compose.yml file.
6. **Docker Swarm**:

- A native clustering and orchestration tool for managing a group of Docker hosts as a single virtual system.

Key Features of Docker

1. **Portability**:
    - Containers ensure that applications can run consistently across different environments, whether on a developer's laptop, a test server, or in production.
2. **Lightweight**:
    - Unlike virtual machines, Docker containers share the host system's kernel, making them faster and more resource-efficient.
3. **Isolation**:
    - Each container runs in a separate environment, ensuring no interference between applications.
4. **Version Control**:
    - Docker images are versioned, allowing developers to track changes and roll back to previous versions if needed.
5. **Extensibility**:
    - Extensive library of pre-built images available on Docker Hub. Custom images can also be created by modifying base images.
6. **Security**:
    - Provides isolation at the container level, ensuring that containers do not access each other's resources unless explicitly allowed.

Popular Docker Commands

1. docker build: Builds a Docker image from a Dockerfile.
2. docker run: Runs a container from a Docker image.
3. docker ps: Lists all running containers.
4. docker stop: Stops a running container.
5. docker pull: Pulls an image from Docker Hub.
6. docker push: Pushes an image to Docker Hub.
7. docker exec: Executes a command inside a running container.

8. docker-compose up: Starts services defined in a docker-compose.yml file.

Kubernetes :

Kubernetes (commonly abbreviated as K8s) is an open-source container orchestration platform developed by Google, now maintained by the Cloud Native Computing Foundation (CNCF). It automates the deployment, scaling, and management of containerized applications, making it easier to run complex applications in production.

Key Concepts and Components of Kubernetes

1. Cluster Architecture

Kubernetes works by orchestrating applications in a **cluster**, which consists of:

- **Master Node**:
  Manages the cluster and handles scheduling, scaling, and monitoring of workloads. Key components include:
  - **API Server**: Provides an interface to interact with the cluster (kubectl, APIs).
  - **Controller Manager**: Manages controllers that regulate the state of the system (e.g., scaling pods).
  - **Scheduler**: Assigns workloads (pods) to appropriate worker nodes based on resources.
  - **etcd**: A distributed key-value store for cluster state and configuration.
- **Worker Nodes**:
  Execute the application workloads. Each node runs:
  - **Kubelet**: Communicates with the master node and manages pod execution.
  - **Container Runtime**: Runs containers (e.g., Docker, containerd).
  - **Kube-Proxy**: Handles networking and communication within the cluster.

2. Kubernetes Resources

Kubernetes uses declarative configurations to manage the lifecycle of applications via YAML or JSON files. Major resources include:

- **Pods**:
  The smallest deployable unit in Kubernetes, representing one or more containers running together with shared storage and networking.

- **Services**:
  Provide stable network endpoints to expose pods to other services or external clients.
- **Deployments**:
  Declarative definition for managing and scaling pods. It ensures desired state is always maintained.
- **ReplicaSets**:
  Maintain a specified number of pod replicas running at all times.
- **Namespaces**:
  Logical partitions for organizing resources in a cluster.
- **ConfigMaps** and **Secrets**:
  Manage configuration data and sensitive information (e.g., passwords, API keys).

3. Features of Kubernetes

1. **Automated Deployment and Scaling**:
   Kubernetes automatically deploys containers and scales them up or down based on resource usage or demand.
2. **Self-Healing**:
   Automatically replaces failed containers, restarts unresponsive ones, and reschedules pods if a node fails.
3. **Load Balancing**:
   Distributes traffic across multiple containers to ensure high availability.
4. **Storage Orchestration**:
   Automatically mounts storage systems like AWS EBS, Google Persistent Disks, or NFS.
5. **Resource Management**:
   Efficiently allocates resources (CPU, memory) to containers to prevent overloading.
6. **Rolling Updates and Rollbacks**:
   Ensures zero-downtime application updates. If something goes wrong, it can roll back to the previous state.
7. **Multi-Cloud Support**:
   Kubernetes is cloud-agnostic, enabling deployment on AWS, Azure, GCP, or on-premises infrastructure.

Advantages of Kubernetes

1. **Portability**:
   Kubernetes supports various container runtimes, cloud platforms, and on-premises environments, making it highly portable.
2. **Scalability**:
   Dynamically scales applications based on traffic or resource consumption.
3. **High Availability**:
   Ensures applications remain available by redistributing workloads during node or pod failures.
4. **Efficient Resource Utilization**:
   Optimizes resource usage by scheduling workloads based on node availability.
5. **Automation**:
   Automates repetitive operational tasks, reducing manual intervention.
6. **Open-Source and Community Support**:
   Kubernetes has strong community backing, with frequent updates and improvements.

Jenkins

Jenkins is an open source continuous integration /continuous delivery and deployment (CI/CD) automation software Devops tool written in the Java programming language. It is used to implement CI/CD workflows called pipelines.

CI/CD pipelines automate testing and reporting on isolated changes in a larger codebase in real time. They also facilitate the integration of disparate branches of the code into a main branch. Pipelines rapidly detect defects in a codebase, build the software, automate testing of builds, prepare the codebase for deployment and delivery, and ultimately deploy code to containers and virtual machines (VMs), as well as to bare-metal and cloud servers.

History of Jenkins

Jenkins is a fork of a project called Hudson, which was trademarked by Oracle. Hudson was eventually donated to the Eclipse Foundation and is no longer under development. Jenkins development is now managed as an open source project under the governance of the CD Foundation, an organization within the Linux Foundation.

How Jenkins Works :

1. **Jobs**:

- Jenkins defines tasks as **jobs**. A job can be as simple as running a shell script or as complex as executing a full CI/CD pipeline.

2. **Triggers**:
    - Jenkins jobs can be triggered automatically by:
        - Code changes in version control (webhooks or polling).
        - Scheduled time intervals (cron-like scheduling).
        - API calls or manual execution.
3. **Build Process**:
    - After triggering a job, Jenkins executes the defined build steps (e.g., compiling code, running tests, creating packages).
4. **Plugins**:
    - Plugins enable integration with external tools and extend Jenkins' functionality, such as Docker integration, Kubernetes support, and report generation.
5. **Pipeline Execution**:
    - Pipelines are executed step-by-step, ensuring tasks like building, testing, and deployment happen in the correct order.