Dev Ops

**1. CI/CD – Continuous Integration / Continuous Deployment (or Delivery)**

* **Continuous Integration (CI)**: Developers frequently merge code into a shared repository. Automated tests verify the changes, ensuring stability and quick feedback.
* **Continuous Delivery (CD)**: The application is automatically built, tested, and prepared for release. Deployments can be triggered manually.
* **Continuous Deployment (CD)**: Every change that passes tests is automatically released to production with no manual steps.
* **Benefits**:
  + Faster development cycles.
  + Quick detection and fixing of bugs.
  + Enhanced collaboration and automation.

**2. Containerisation**

* A method to **package an application and its dependencies** into a single container.
* Containers run consistently across different environments (dev, test, production).
* Lightweight, portable, and use fewer resources than traditional virtual machines.
* Improves scalability and simplifies deployment.

**3. Virtual Machines vs Containers**

|  |  |  |
| --- | --- | --- |
| Feature | Virtual Machines | Containers |
| Resource Usage | Heavy (includes full OS) | Lightweight (shares host OS) |
| Boot Time | Minutes | Seconds |
| Isolation | Strong (full OS separation) | Moderate (process-level) |
| Portability | Less portable | Highly portable |
| Performance Overhead | Higher | Lower |

**4. Docker vs Kubernetes**

|  |  |  |
| --- | --- | --- |
| Feature | Docker | Kubernetes |
| Isolation | Provides isolated containers | Manages containers across clusters |
| **Scalability** | Limited to single host | Automatically scales across nodes |
| **Response Management** | Manual restart & scaling | Self-healing and auto-restarts |
| **API Driven** | Uses Docker CLI and REST API | Declarative config & API |
| **Container Runtime** | Provides runtime for containers | Orchestrates container runtimes |
| **Control over Resources** | Limited resource control | Advanced resource limits and quotas |
| **Complexity** | Easier to learn and use | More complex but powerful |
| **Popular Use Case** | Application packaging and testing | Managing production-scale deployments |

**Docker** is an open-source platform designed to **simplify application development, deployment, and execution** by using **containers**.

**Key Concepts:**

* **Container**: A lightweight, standalone, executable package that includes everything needed to run a piece of software—code, runtime, libraries, and system tools.
* **Docker Engine**: The runtime that builds and runs Docker containers.
* **Docker Image**: A read-only template used to create containers. It includes the application and all its dependencies.
* **Docker Container**: A running instance of a Docker image. Containers are isolated from each other and from the host system.

**Benefits:**

* **Portability**: Containers can run consistently across any environment (development, testing, production).
* **Efficiency**: Containers use less memory and start faster than virtual machines.
* **Scalability**: Easy to scale services up or down using orchestration tools like Kubernetes.
* **Isolation**: Applications run in isolated environments, reducing conflicts between software versions.

**Common Use Cases:**

* Microservices architecture
* Continuous integration/continuous deployment (CI/CD)
* Testing and development environments

**Kubernetes** (often abbreviated as **K8s**) is an open-source container orchestration platform developed by Google. It is used to **automate the deployment, scaling, and management of containerized applications**.

**Key Concepts:**

* **Cluster**: A set of machines (nodes) that run containerized applications.
* **Node**: A single machine in the cluster (can be physical or virtual).
* **Pod**: The smallest deployable unit in Kubernetes. A pod can contain one or more containers that share the same network and storage.
* **Deployment**: Manages the rollout and scaling of pods and updates to application versions.
* **Service**: An abstraction that defines a logical set of pods and a policy to access them.
* **Namespace**: A way to divide cluster resources between multiple users or projects.

**Core Features:**

* **Self-Healing**: Automatically restarts failed containers, replaces them, and reschedules them on healthy nodes.
* **Load Balancing**: Distributes traffic evenly across pods.
* **Scaling**: Automatically scales applications up or down based on demand.
* **Rolling Updates**: Gradually rolls out changes to your application with zero downtime.
* **Storage Orchestration**: Mounts local or cloud storage as required.

**Benefits of Kubernetes:**

* Efficient **resource utilization** across clusters.
* Supports **declarative configuration** and automation.
* Portable across **on-premise, hybrid, and cloud** environments.
* Integrates well with **CI/CD pipelines**.

**Common Use Cases:**

* Managing large-scale **microservices** applications.
* Running **cloud-native** applications.
* Automating infrastructure in **DevOps** pipelines.
* Ensuring **high availability and fault tolerance**.