**1. Google Cloud Platform (GCP) Introduction**

**What is GCP?**

Google Cloud Platform is a suite of **cloud computing services** offered by Google. It provides infrastructure as a service (IaaS), platform as a service (PaaS), and serverless computing environments to build, deploy, and scale applications.

**Key Services of GCP:**

| **Category** | **Services** |
| --- | --- |
| **Compute** | Compute Engine, App Engine, Kubernetes Engine (GKE), Cloud Functions |
| **Storage** | Cloud Storage, Persistent Disks, Cloud SQL, Firestore, BigQuery |
| **Networking** | Virtual Private Cloud (VPC), Cloud Load Balancing, Cloud CDN |
| **Big Data** | BigQuery, Dataflow, Dataproc, Pub/Sub |
| **AI & ML** | AI Platform, Vertex AI, AutoML, Dialogflow |
| **Identity & Security** | IAM, Cloud Identity, VPC Service Controls, Cloud Armor |
| **Monitoring** | Cloud Monitoring (formerly Stackdriver), Cloud Logging |

**Global Infrastructure:**

* 35+ **regions**
* 100+ **zones**
* 200+ **edge locations**
* Highly available and redundant infrastructure

**Key Features:**

* Auto-scaling and load balancing
* Serverless options (Cloud Functions, Cloud Run)
* Integrated ML/AI tools
* Deep integration with Google Workspace and Android
* Pay-as-you-go pricing

**2. Multi-cloud Strategies**

**What is Multi-cloud?**

A **multi-cloud strategy** involves using **multiple cloud service providers (CSPs)** (e.g., AWS, Azure, GCP) to run different workloads.

**Why use Multi-cloud?**

* **Avoid vendor lock-in** – leverage services from multiple vendors
* **High availability and resilience** – distribute workloads
* **Compliance and data sovereignty** – meet regional or industry regulations
* **Optimized performance** – choose best-in-class services

**Common Multi-cloud Architectures:**

| **Strategy** | **Description** |
| --- | --- |
| **Redundant Multi-cloud** | Run the same app on multiple clouds for failover |
| **Split-stack** | Frontend in one cloud, backend/database in another |
| **Workload distribution** | Distribute different services to different clouds based on their strengths |
| **Geo-specific** | Use cloud services based on regional performance/availability |

**Tools Enabling Multi-cloud:**

* **Anthos** (by GCP): Manage workloads across GCP, on-premises, and other clouds
* **Terraform**: Infrastructure as Code (IaC) supporting multiple providers
* **Kubernetes**: Container orchestration that can span multiple clouds
* **HashiCorp Vault / Consul**: For secrets and service mesh management

**Best Practices:**

* Unified monitoring/logging across providers
* Abstract the provider-specific logic
* Use cloud-agnostic tools (e.g., containers, serverless frameworks)
* Manage IAM and security consistently across platforms

**3. Cloud Cost Optimization**

**Why Cloud Cost Optimization Matters:**

* Cloud costs can scale rapidly and unexpectedly
* Unused or underutilized resources can result in wasted spend
* Budgeting and forecasting are harder in dynamic environments

**Cost Optimization Areas:**

**a) Right-Sizing Resources**

* Identify oversized VMs, storage, and databases
* Use tools like **GCP’s Recommender** to suggest optimal instance sizes

**b) Auto-scaling**

* Configure auto-scaling to adjust resources based on demand
* Use GKE, Cloud Run, or App Engine with scaling policies

**c) Use Committed and Sustained Use Discounts**

* **Committed Use Discounts (CUDs)**: Long-term usage commitments in exchange for lower prices
* **Sustained Use Discounts**: Automatically applied for sustained usage of VMs in a month

**d) Turn Off Idle Resources**

* Use automation or policies to shut down unused VMs, test environments, or dev clusters during off-hours

**e) Storage Lifecycle Management**

* Move infrequently accessed data to **Coldline** or **Archive** storage
* Use lifecycle rules to auto-delete old data

**f) Monitor and Track Usage**

* Use **Cloud Billing Reports**, **Cost Table Reports**, and **Budgets & Alerts**
* Integrate **Cloud Monitoring** and **Cloud Logging** for visibility

**g) Tagging and Labeling Resources**

* Use labels to group resources by project, team, or environment
* Helps in chargeback/showback and usage analysis

**GCP Cost Tools:**

* **GCP Pricing Calculator** – Estimate monthly costs
* **Billing Reports and Budgets**
* **Recommender** – Suggests cost-saving opportunities
* **Cost Table Report** – Breaks down usage and pricing
* **BigQuery for Billing Export** – Analyze large-scale billing data

**✅ Summary Table:**

| **Topic** | **Key Points** |
| --- | --- |
| **GCP Intro** | Wide service range, global infra, strong AI/ML support |
| **Multi-cloud** | Increases flexibility and reliability, avoid lock-in |
| **Cost Optimization** | Focus on right-sizing, auto-scaling, discounts, and monitoring |

**1. Docker Networking**

**What is Docker Networking?**

Docker networking allows **containers to communicate** with each other, the host machine, and external networks (like the internet). Each container is given its own network namespace and IP address.

**Types of Docker Networks**

| **Network Type** | **Description** | **Use Case** |
| --- | --- | --- |
| **bridge** (default) | Creates an isolated network for containers to talk to each other | Standalone container setups |
| **host** | Shares the host’s network namespace with the container | High-performance needs; access host ports |
| **none** | Disables all networking for the container | Completely isolated containers |
| **overlay** | Connects containers across multiple Docker hosts (Swarm mode) | Multi-host, distributed apps |
| **macvlan** | Assigns a MAC address to containers and makes them appear like physical devices | When containers need to be on the same network as host |
| **custom networks** | Created using docker network create for fine-grained control | Enables service discovery and DNS resolution |

**Bridge Network (Default)**

* Containers get a private IP.
* Docker automatically sets up a virtual switch to manage the communication.
* You can use container names to connect using DNS within the same bridge network.

# Create a user-defined bridge network

docker network create my-bridge

# Run containers in that network

docker run -dit --name web --network my-bridge nginx

docker run -dit --name app --network my-bridge alpine

**Host Network**

* Container uses the host’s IP directly.
* Useful when you need high performance or access to host-level ports.

docker run --rm --network host nginx

**Overlay Network**

* Used in Docker Swarm (multi-host).
* Allows containers on different Docker hosts to communicate.

docker network create -d overlay my-overlay

**✅ Key Networking Commands**

docker network ls # List networks

docker network inspect <network-name> # Details of a network

docker network create <name> # Create custom network

docker network connect <network> <container> # Connect container to a network

**2. Docker Volumes**

**What is a Docker Volume?**

Volumes are used to **persist data** generated by and used in Docker containers. Volumes are stored **outside the container's filesystem**, in a special part of the host filesystem managed by Docker (/var/lib/docker/volumes/).

**Why Use Volumes?**

* Data persistence even if the container is removed
* Easy backup and migration
* Sharing data between containers
* Separation of concerns (app vs. data)
* Better performance than bind mounts (especially on Docker Desktop)

**Types of Docker Storage:**

| **Storage Type** | **Description** |
| --- | --- |
| **Volumes** | Managed by Docker, stored in Docker's data directory |
| **Bind Mounts** | Link container path to a host path directly |
| **tmpfs** | Stored in RAM; data is lost on container stop |

**✅ Working with Volumes**

**Create a volume:**

docker volume create my-data

**Use a volume in a container:**

docker run -d -v my-data:/app/data --name myapp myimage

**Inspect a volume:**

docker volume inspect my-data

**Remove a volume:**

docker volume rm my-data

**Example: Shared Volume Between Containers**

docker volume create shared-vol

docker run -dit --name container1 -v shared-vol:/data busybox

docker run -dit --name container2 -v shared-vol:/data busybox

* Both containers can read/write to /data, which maps to the same volume.

**Best Practices**

* Use **volumes** for database and application data.
* Avoid using bind mounts in production unless necessary.
* Use **named volumes** for better manageability.
* Clean up unused volumes:

docker volume prune

**✅ Summary Table**

| **Feature** | **Docker Networking** | **Docker Volumes** |
| --- | --- | --- |
| Purpose | Connect containers and external systems | Persist and share data across containers |
| Types | bridge, host, none, overlay, macvlan | named volumes, bind mounts, tmpfs |
| Managed by | Docker Engine | Docker Engine |
| Isolation | Different networks provide isolation | Data is isolated from container lifecycle |
| Common Use | Microservices, service discovery | Databases, logs, configs, backups |