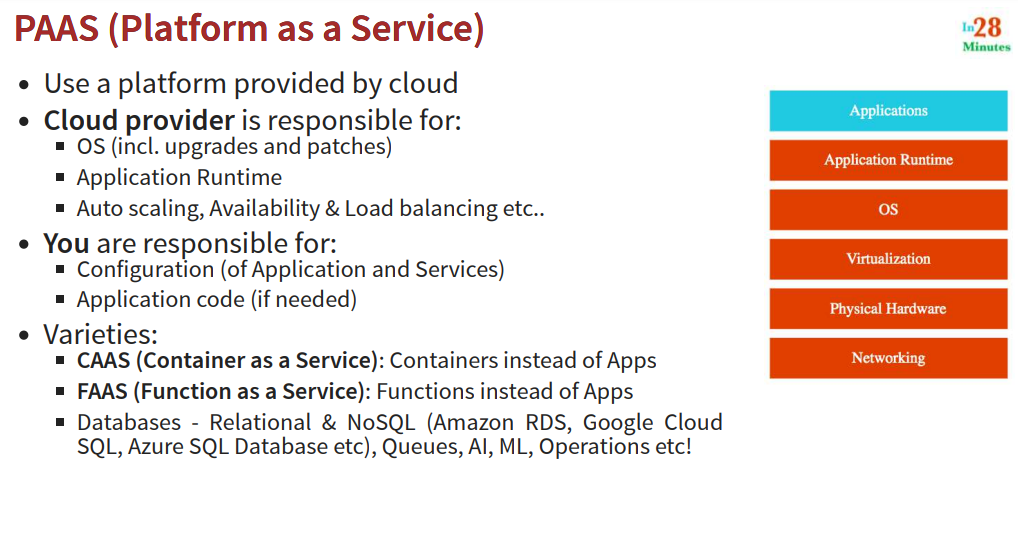
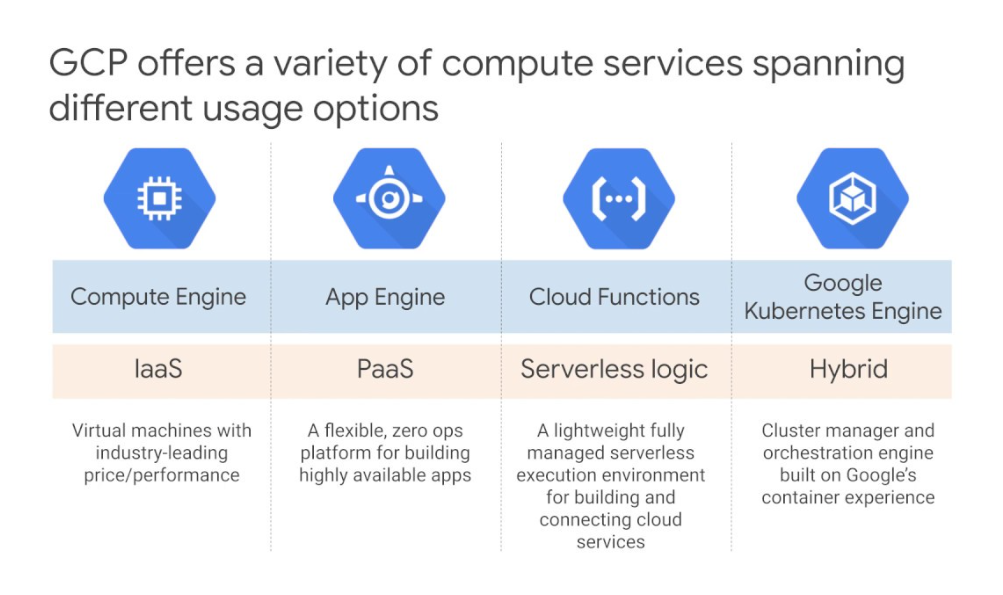
GCP





# Types of Infra in GCP



## 1.Compute Engine

* **Type**: Infrastructure as a Service (IaaS)
* **Description**: Provides raw virtual machines (VMs) with full customization options, including the OS, storage, networking, and more. You can run any application or workload with a high degree of control.
* **Use Cases**:
  + Running legacy applications or custom software
  + Workloads requiring specific OS configurations
  + Applications requiring custom hardware or high performance (e.g., GPUs for machine learning)
* **Pros**:
  + Complete control over the VM
  + Ideal for custom configurations and high-performance needs
* **Cons**:
  + More setup and maintenance required (OS patching, scaling, etc.)
* **Best for**: Full control of the infrastructure and custom configurations

## 2. App Engine

* **Type**: Platform as a Service (PaaS)
* **Description**: Managed platform for deploying applications directly. Handles scaling, load balancing, and app updates automatically. Available in both **Standard** (with built-in language runtimes) and **Flexible** (with custom runtimes in Docker containers) environments.
* **Use Cases**:
  + Web and mobile backends
  + Microservices requiring automatic scaling
  + Development teams that want to focus only on the code
* **Pros**:
  + Automatic scaling and management
  + Integrated CI/CD options
  + Rapid development with managed infrastructure
* **Cons**:
  + Limited control over the underlying infrastructure
  + Not ideal for applications needing custom configurations
* **Best for**: Web applications and services where scalability and simplicity are key

## 3. Google Kubernetes Engine (GKE)

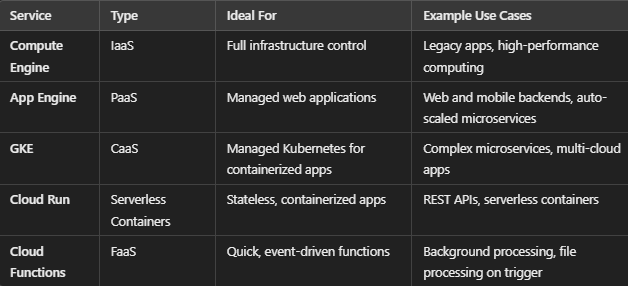
* **Type**: Container as a Service (CaaS)
* **Description**: Managed Kubernetes platform, enabling container orchestration and deployment of containerized applications. Ideal for microservices and applications requiring complex configurations.
* **Use Cases**:
  + Microservices architectures and containerized applications
  + Applications needing high scalability and portability
  + Development teams experienced with Kubernetes
* **Pros**:
  + Full control over Kubernetes environment
  + Highly scalable and resilient
  + Allows for hybrid and multi-cloud deployments
* **Cons**:
  + Steeper learning curve (requires Kubernetes knowledge)
  + Higher maintenance and configuration requirements compared to fully managed services
* **Best for**: Complex, containerized applications and microservices architectures

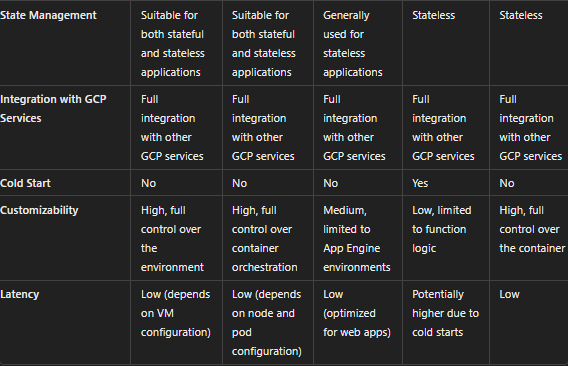
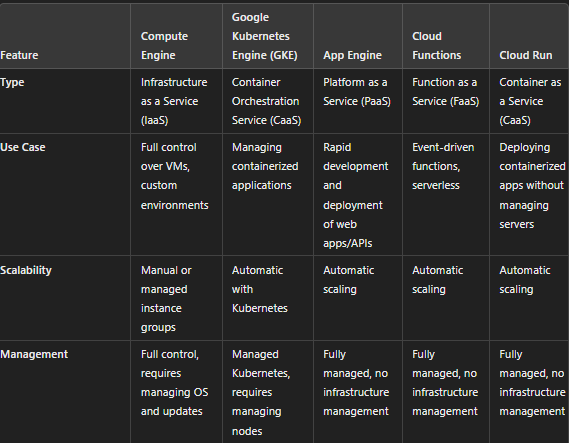
## 4. Cloud Run

* **Type**: Serverless Container Platform
* **Description**: Fully managed service to run stateless containers on a serverless platform. Automatically scales to handle incoming requests, making it suitable for rapid deployments without needing Kubernetes expertise.
* **Use Cases**:
  + Stateless, HTTP-driven applications or APIs
  + Event-driven processing (e.g., background jobs)
  + Microservices that can be containerized
* **Pros**:
  + Serverless scaling with a pay-per-use model
  + Simple to deploy containers without managing infrastructure
  + Easy integration with other Google Cloud services
* **Cons**:
  + Limited to stateless workloads and HTTP-based containers
  + Less control compared to GKE
* **Best for**: Stateless applications and microservices needing fast, serverless deployments

## 5. Cloud Functions

* **Type**: Function as a Service (FaaS)
* **Description**: Serverless compute service designed to run single-purpose, event-driven functions. Ideal for quick and lightweight tasks, with automatic scaling based on events.
* **Use Cases**:
  + Event-driven workloads (e.g., file uploads, database changes)
  + Background processing tasks
  + Real-time processing with minimal setup
* **Pros**:
  + Event-driven, with no need to manage servers
  + Low cost and pay-per-execution model
  + Good integration with Google Cloud services and events
* **Cons**:
  + Limited to short-running tasks and specific languages
  + Not suitable for complex or stateful applications
* **Best for**: Quick, event-driven functions or background tasks requiring minimal infrastructure







# Compute Engine

Google Compute Engine (GCE) is a part of Google Cloud Platform (GCP) that offers virtual machine (VM) instances for computing resources in the cloud.

Google Compute Engine allows you to:

* Host and run websites and applications.
* Process and analyze large amounts of data.
* Develop, test, and deploy software.
* Run legacy and high-performance applications.
* Scale resources based on demand.
* Host game servers and virtual desktops.
* Provide backup and disaster recovery solutions.

**Disks**:

* **Boot Disks**: The disk from which the VM boots.
* **Persistent Disks**: Durable and high-performance storage for VM data.
* **Local SSDs**: High-speed, temporary storage

## Virtual Machine (VM)

**Definition**: A virtual machine (VM) is a virtualized instance of a computer system that runs on GCP's infrastructure.

**Use**: VMs are used to run applications, host websites, process data, and perform a wide variety of computing tasks.

**Example**: When you create a VM instance in Google Compute Engine (GCE), you specify the machine type, operating system, and other configurations.

A Virtual Machine (VM) is a compute instance that emulates a physical computer, allowing you to run applications and software in a virtual environment.

These VMs can be customized with different operating systems, CPUs, memory, and storage to meet specific needs.

* **General Purpose**: Balanced resources (e.g., N1, N2 instances).
* **Compute-Optimized**: High CPU performance (e.g., C2 instances).
* **Memory-Optimized**: High memory capacity (e.g., M2 instances).
* **Accelerator-Optimized**: GPU/TPU support for machine learning (e.g., A2 instances).
* **Shared-core**: Cost-effective with shared CPUs (e.g., F1-micro).

## Template

An instance template is a configuration file that defines the settings for VM instances. It includes parameters like machine type, disk types, network configurations, metadata, and other instance properties.

**Definition**: A VM instance template is a configuration file that defines the settings and parameters for VM instances.

**Use**: Templates are used to create multiple VM instances with the same configuration, ensuring consistency and simplifying the creation process.

**Example**: You can use a template to launch a group of instances in a managed instance group with predefined settings such as machine type, disk images, and network configurations.

## Snapshot

A snapshot is a point-in-time copy of a disk, typically a persistent disk. Snapshots are used primarily for backup and recovery purposes.

**Definition**: A snapshot is a point-in-time copy of a VM's persistent disk. Unlike images, snapshots can be incremental, meaning they only store changes made since the last snapshot.

**Use**: Snapshots are used for backup and recovery purposes. They allow you to restore a disk to a previous state or create a new disk with the same data.

**Example**: You might take regular snapshots of a VM's disk to ensure you can recover data in case of failure or corruption.

## Image

An image is a stored copy of a VM instance's boot disk or operating system that can be used to create new instances.

**Definition**: An image is a static snapshot of a VM's disk at a specific point in time, including the operating system and installed software.

**Use**: Images are used to create new VM instances. They provide a way to quickly launch VMs with a preconfigured OS and software setup.

**Example**: You can use a public image provided by GCP (like a basic Ubuntu or Windows image) or create your own custom image from an existing VM.

| **Aspect** | **VM** | **Template** | **Image** | **Snapshot** |
| --- | --- | --- | --- | --- |
| **Purpose** | Running applications and services | Defining configurations for creating consistent VMs | Providing a reusable OS and software setup | Backup and recovery of disk data |
| **State** | Active, running instances | Static configuration file | Static copy of a disk | Point-in-time copy of a disk |
| **Use Case** | Deploy and manage applications | Automate creation of similar VMs | Quickly launch new VMs with a predefined setup | Restore disks to previous states, protect against data loss |
| **Creation** | Created through GCE with specified configurations | Created to standardize VM configurations | Created from existing VMs or disks | Created from existing persistent disks |
| **Storage** | Persistent disks | Text-based configuration | Stored as images in GCP | Stored as snapshots in GCP |
| **Modification** | Running VMs can be modified (resize, attach/detach disks, etc.) | Templates can be edited and updated | Images are immutable once created | Snapshots are immutable; new changes require new snapshots |
| **Recovery** | NA | NA | NA | Used to recover disks to the state at the time of snapshot |

**Tags**: Primarily used for network security by defining which resources can communicate with VM instances.

**Labels**: Used for organization, management, and operational purposes to categorize and identify resources.

RAM, CPU and GPU

RAM - Random Access Memory

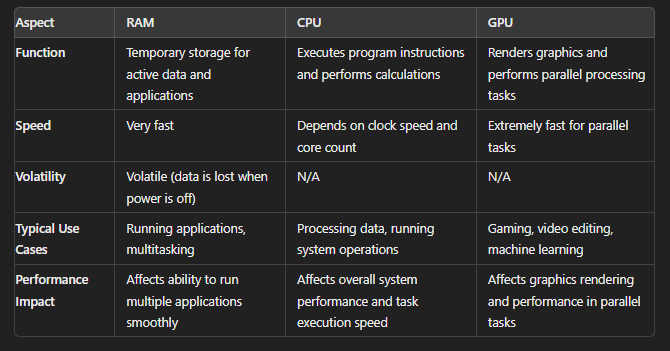
Use Cases:

* Running Applications: Active processes and applications are loaded into RAM to ensure quick access and execution.
* Multitasking: More RAM allows a computer to handle multiple applications at the same time without slowing down.

CPU – Central Processing Unit

Use Cases:

* Executes instructions from programs and applications.
* Performs calculations and logical operations.
* Handles tasks such as running applications, processing data, and managing system operations.



Gcloud

**Gcloud** is the command-line interface (CLI) for Google Cloud Platform.

It allows you to manage GCP resources and services directly from the command line, providing a powerful tool for automation and scripting.

Gsutil

**Gsutil** is specialized for interacting with Google Cloud Storage, providing powerful capabilities for managing objects and buckets

INSTANCE GROUPS

Instance groups are collections of VM instances that you can manage as a single entity. They are useful for scaling applications, ensuring high availability, and simplifying management tasks. GCP offers two types of instance groups: managed instance groups (MIGs) and unmanaged instance groups.

Managed Instance Groups (MIGs) :

* Group of VMs with created with same configuration or with same template
* Automatically manage the lifecycle of VM instances based on a specified instance template.
* Support auto-scaling, auto-healing, rolling updates, and multi-zone deployments.
* Ideal for stateless, scalable workloads like web servers and application servers.

**Advantages**:

Automation of scaling, healing, and updates; multi-zone deployment for high availability; integrates with managed load balancing.

**Best For**:

Stateless applications, auto-scaling needs, applications requiring high availability and fault tolerance.

Unmanaged Instance Groups :

* Group of VMs with different configuration
* Simply group together existing VM instances without automatic management features.
* Useful for grouping and load balancing a set of instances that you manage individually.

**Advantages:**

Full control over individual instances.

suitable for fixed-size deployments or legacy applications.

**Best For:**

Applications with static workloads, specific management requirements, or that cannot easily be refactored for automation.

Rolling update

Rolling restart/replace

# App Engine

**Fully Managed Service**:

You don't need to manage servers, storage, or networks.

Google handles all the infrastructure, including scaling, load balancing, and security patches.

**Automatic Scaling**:

Your application automatically scales up and down based on traffic. If your app gets more traffic, App Engine adds more resources; if traffic decreases, it scales down to save costs.

**Supports Multiple Languages**:

You can write your applications in various programming languages such as Python, Java, Node.js, Go, Ruby, PHP, and .NET.

**Integrated Developer Tools**:

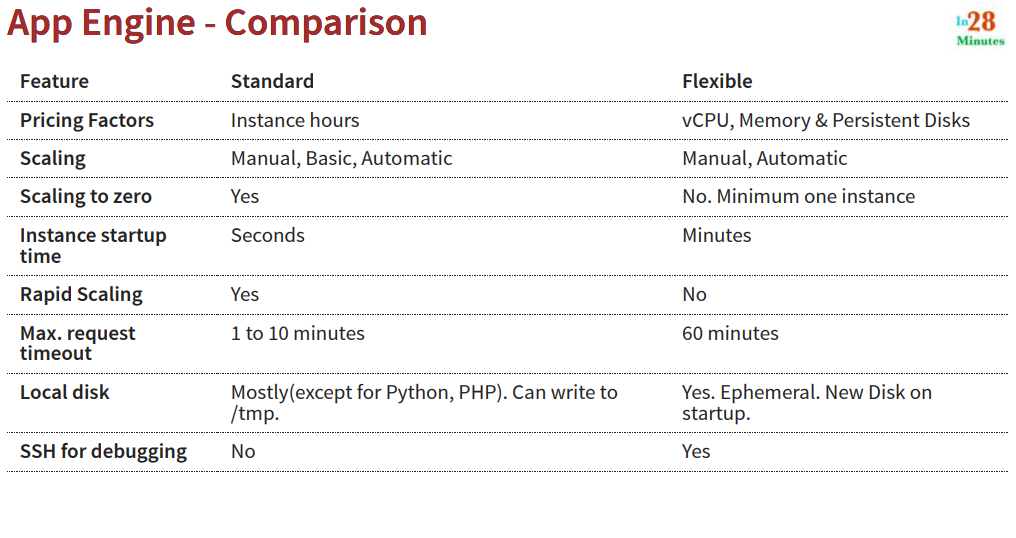
Tools for continuous integration and delivery (CI/CD) are built-in. You can easily deploy updates and manage your application’s lifecycle.

**Standard Environment**:

* Runs in a sandboxed environment.
* Supports automatic scaling.
* Limited to certain runtime versions and language features.

**Flexible Environment**:

* Runs on customizable Compute Engine VMs.
* Supports any runtime and custom libraries.
* Provides more control over the environment but requires managing scaling configurations.



# Containerization

**Monolithic Architecture**

In a Monolithic architecture, the entire application is built as a single, indivisible unit. All components of the application are interconnected and interdependent.

**Micro service Architecture**

Micro services architecture breaks down an application into smaller, independent services that are loosely coupled and can be developed, deployed, and scaled independently.

**Container**

A container includes everything needed to run the application, such as code, runtime, system tools, libraries, and settings.

Imagine you have a web application. Instead of installing all the necessary software and dependencies directly on a server, you package the application along with its environment (like the web server, database connectors, etc.) into a container.

This container can then be easily moved and run on any machine that supports containers, ensuring consistent behavior regardless of where it's deployed.

## Use Cases:

**Microservices**: Ideal for building and deploying microservices architectures where applications are divided into smaller, independent services.

**Continuous Integration/Continuous Deployment (CI/CD)**: Containers streamline the process of building, testing, and deploying applications in automated pipelines.

**Cloud-Native Applications**: Containers are widely used in cloud environments for building scalable and resilient applications.

**Helm** is a package manager for Kubernetes applications that simplifies the process of installing, managing, and deploying software applications on Kubernetes clusters. Here's a simplified explanation:

* **Purpose**: Helm helps manage Kubernetes applications by packaging them into charts. These charts include all the Kubernetes resources and configurations needed to run an application.
* **Charts**: A chart is a collection of files that describe a set of Kubernetes resources. It includes YAML files for deployments, services, ingress rules, and any other necessary configurations

# Cloud Source Repository

Cloud Source Repositories is a Git version control service provided by Google Cloud Platform (GCP). Here’s a straightforward explanation:

* **Git Repository**: Cloud Source Repositories allows you to create and manage Git repositories for storing and versioning your source code.
* **Integration**: It seamlessly integrates with other Google Cloud services and tools, making it easy to deploy code to services like Google Kubernetes Engine (GKE), Compute Engine, and App Engine.
* **Collaboration**: Developers can work together on projects, sharing and reviewing code changes, tracking issues, and managing project tasks.

# Google Cloud Registry

# **GCP Kubernetes**

**GCP Kubernetes**: Creates a cluster of machines (nodes) that can run containers. These nodes work together to run and manage your Docker containers.

**Clusters**:

* **Nodes**: The basic units of a Kubernetes cluster. Nodes can be either master (control plane) or worker (where applications run).
* **Pods**: The smallest deployable units in Kubernetes, encapsulating one or more containers. Pods manage containers, storage, and networking.

## Key Features of GKE

**Maintenance**: Google handles the management and maintenance of the Kubernetes control plane, including updates and patches.

**Auto-Scaling**: GKE can automatically scale the number of nodes in your cluster based on resource usage and demand.

**Google Cloud Services**: Seamlessly integrates with other GCP services like Cloud Storage, Cloud SQL, and Stackdriver for monitoring and logging.

**IAM Integration**: Leverages Google Cloud’s Identity and Access Management (IAM) for access control.

**GKE Autopilot**: Provides a fully managed Kubernetes experience where Google takes care of infrastructure management and operations.

**Internal Load Balancing**: Provides internal load balancing and network policies to control traffic flow within your cluster.

Create and Connect to Kube cluster

gcloud container clusters create my-cluster --num-nodes 3 --zone us-central1-a --project solid-course-258105

gcloud container clusters get-credentials my-cluster --zone us-central1-a --project solid-course-258105

Increase No of Nodes

gcloud container clusters resize my-cluster --node-pool my-node-pool --num-nodes 5

Create deployment & service using kubectl commands

kubectl create deployment hello-world-rest-api --image=in28min/hello-world-rest-api:0.0.1.RELEASE

kubectl get deployment

kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080

kubectl get services

Increase number of instances of your microservice:

kubectl scale deployment hello-world-rest-api --replicas=3

Setup auto scaling for your microservice:

kubectl autoscale deployment hello-world-rest-api --max=10 --cpu-percent=70

Setup auto scaling for your Kubernetes Cluster

gcloud container clusters update cluster-name --enable-autoscaling --min-nodes=1 -- max-nodes=10

Add some application configuration for your microservice

Config Map –

kubectl create configmap todo-web-application-config –from-literal=RDS\_DB\_NAME=todos

Add password configuration for your microservice

Kubernetes Secrets –

kubectl create secret generic todo-web-application-secrets-1 -- from-literal=RDS\_PASSWORD=dummytodos

Delete the Microservices Delete service

kubectl delete service

Delete the Cluster

gcloud container clusters delete

# Cloud Run

**Google Cloud Run** is a service provided by Google Cloud Platform (GCP) that allows you to run your applications as containers. It makes deploying and managing applications easy and efficient, especially if you want to avoid dealing with infrastructure details.

* You don’t need to manage servers. Google handles the infrastructure for you.
* Focus on writing your code and let Cloud Run take care of deployment and scaling.
* You pay for the CPU usage based on the amount of time your container is processing requests.

# Cloud Funtion

With **Cloud Functions**, you write and deploy small pieces of code (functions) without managing the underlying infrastructure.

Google automatically manages scaling, provisioning, and runtime environments.

Functions are triggered by events from GCP services like Cloud Storage, Pub/Sub, Firestore, and HTTP requests.

Imagine you have an e-commerce website. You can use Cloud Functions to automatically send confirmation emails to customers whenever they make a purchase. The function would be triggered by a database change event (like a new order entry), process the order details, and send an email using an email service API

# PUB/SUB

Google Cloud Pub/Sub is a messaging service that allows applications to send and receive messages reliably, asynchronously, and at scale.

* You might use Pub/Sub to send a message from one service that processes customer orders to another service that handles inventory updates.
* The processing service subscribes to the "weather-data" topic to receive and process the data.

# Load Balancer

Load balancing distributes incoming network traffic across multiple servers to ensure no single server becomes overwhelmed, improving performance and reliability.

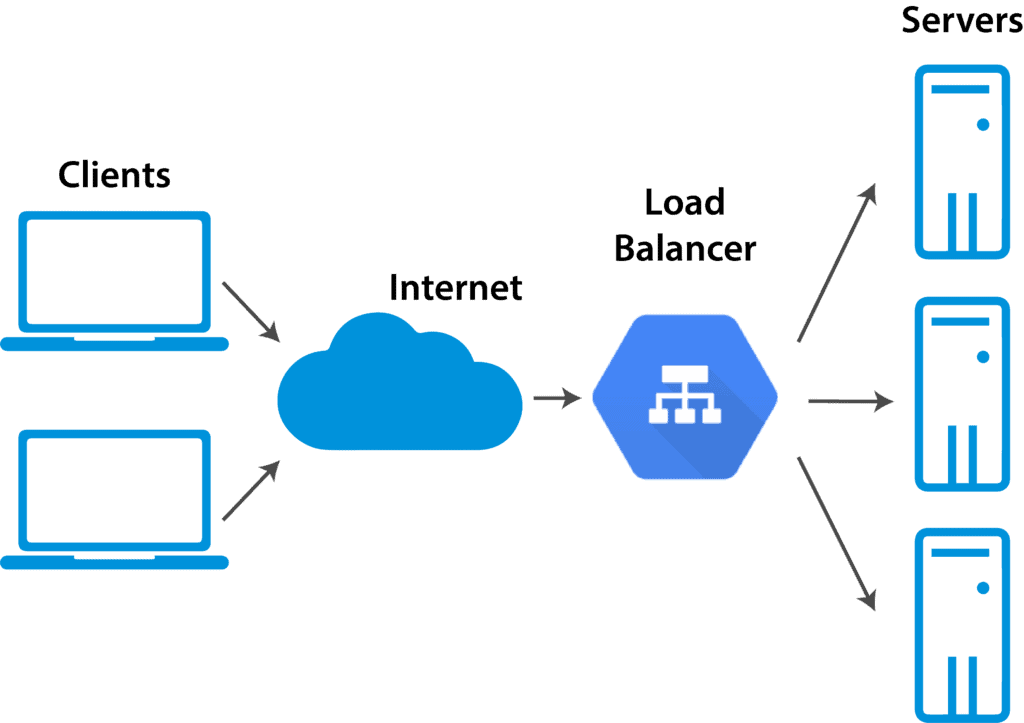
**HTTP(S) Load Balancer**: For web traffic, global, Layer 7.

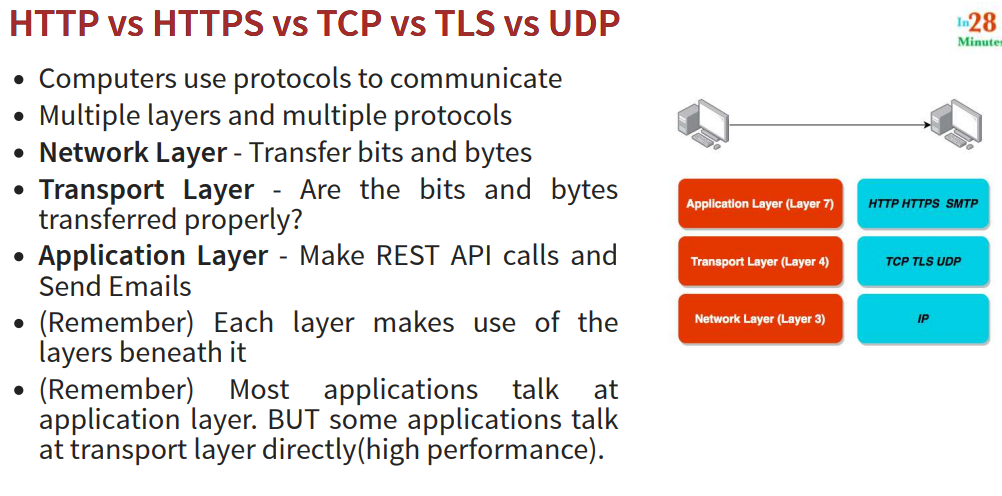
**TCP/SSL Proxy Load Balancer**: For TCP/SSL traffic, global, Layer 4.

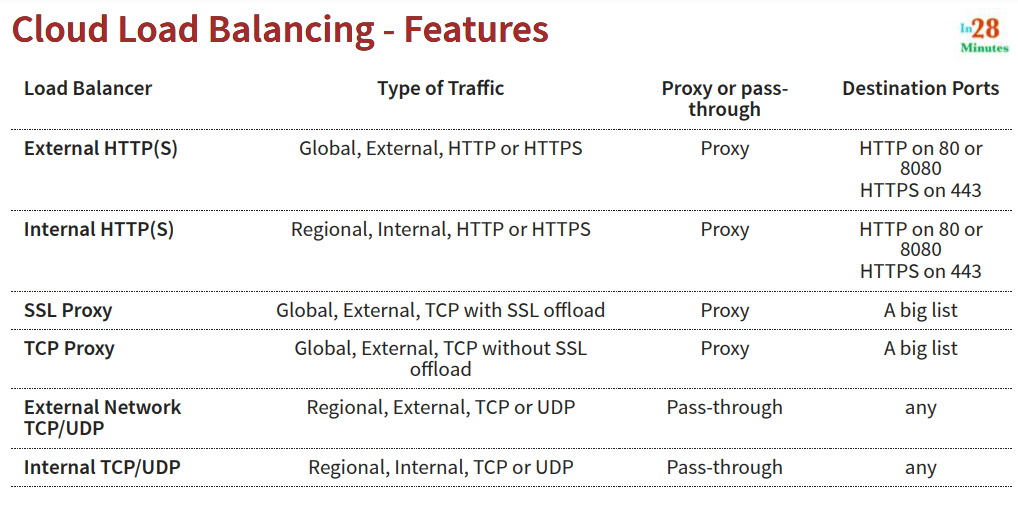
**Network Load Balancer**: For raw TCP/UDP traffic, regional, Layer 4.

**Internal HTTP Load Balancer**: For internal web traffic, regional, Layer 7.

**Internal TCP/UDP Load Balancer**: For internal TCP/UDP traffic, regional, Layer 4







# VPC

A **Virtual Private Cloud (VPC)** in Google Cloud is like your own private network on the internet where you can run your applications securely.

## IP Addresses:

* **Definition:** An IP address is like the address of a house. It uniquely identifies a device (like a computer or a smartphone) on a network.
* **Example:** An IP address looks like this: 192.168.1.1 for IPv4, or 2001:0db8:85a3:0000:0000:8a2e:0370:7334 for IPv6.
* **Purpose:** When data is sent over the internet, it needs to know where to go. The IP address tells it the destination, like sending a letter to someone's home.

## Ports:

* **Definition:** A port is like a specific room in a house. Each device has many ports, and each port can be used for a different type of communication.
* **Example:** Common port numbers include 80 for HTTP (web traffic) and 443 for HTTPS (secure web traffic).
* **Purpose:** Ports allow different types of services and applications to run on the same device without interfering with each other. For example, web browsing and email services can run simultaneously on the same device because they use different ports.

## Protocols:

* **Definition:** A protocol is like a language that devices use to communicate with each other. It sets the rules for how data is transmitted and received.
* **Example:** Common protocols include:
  + **HTTP (Hypertext Transfer Protocol):** Used for browsing websites.
  + **FTP (File Transfer Protocol):** Used for transferring files.
  + **TCP (Transmission Control Protocol):** Ensures reliable data transmission.
  + **UDP (User Datagram Protocol):** Used for faster, but less reliable, transmissions, like video streaming.
* **Purpose:** Protocols ensure that data is exchanged in a way that both the sending and receiving devices understand, ensuring effective communication.

**Subnets** are smaller sections of your VPC network. Think of them as different floors in a building, each with its own range of IP addresses.

* **Definition**: Subnets are subdivisions of a larger VPC network in GCP. They allow you to segment your network into smaller, more manageable parts.
* **Purpose**: Subnets help in organizing and controlling how your virtual machines (VMs), services and other resources are connected and communicate within your cloud environment.

**Auto Mode VPC**: Google automatically creates subnets for you in every region.

**Custom Mode VPC**: You decide where and how to create your subnets, giving you more control.

**Private Google Access** allows your resources to communicate with Google services without needing a public internet IP, making it more secure.

**Shared VPC** lets multiple projects within your organization share a common VPC, so different teams can work together more easily.

**VPC Peering** connects two VPCs privately, so they can share resources without using the public internet.

**Hybrid Cloud** connects your existing data centers to Google Cloud. For example, if you have a data center in your office, you can connect it to your Google Cloud VPC to extend your resources and capabilities

**Cloud VPN**: Securely connects your on-premises network to your Google Cloud VPC over the internet.

## Accessing from other Network

### Identity-Aware Proxy (IAP)

* Secure access to web applications and resources by verifying user identity and context
* **Access Control** : User-level, based on identity and context.
* **Best For**: Securing access to web applications and cloud resources with granular, user-level control.
* **Strengths**: High security with context-aware access, easy management, suitable for remote work without VPN.
* **Use Cases**: Protecting internal web apps, providing secure access to cloud resources.

### VPN

* Connect on-premises networks to GCP securely over the internet
* **Access Control** : Network-level, allows access to entire on-premises network.
* **Best For**: Securely connecting on-premises networks to GCP for full network access.
* **Strengths**: Good for extending on-premises networks to the cloud, lower initial cost compared to direct connections.
* **Use Cases**: Site-to-site connections, hybrid cloud setups, secure network extension.

### Cloud Interconnect

* Establish a high-bandwidth, low-latency connection between on-premises networks and GCP
* **Access Control** : Network-level, allows access to entire on-premises network.
* **Best For**: High-bandwidth, low-latency connections between on-premises networks and GCP.
* **Strengths**: High performance, suitable for bandwidth-intensive and latency-sensitive applications.
* **Use Cases**: Large-scale data migration, disaster recovery, hybrid cloud architectures.

### Choosing the Right Solution :

* **Use IAP** if you need secure, user-level access to web applications and resources with minimal management overhead.
* **Use VPN** if you need to securely extend your on-premises network to GCP and can manage the network and security configurations.
* **Use Cloud Interconnect** if you require high performance, low latency, and reliable connections for large data transfers and hybrid cloud setups.

| **Aspect** | **VPN** | **IAP** |
| --- | --- | --- |
| **Purpose** | Securely connect networks (site-to-site) | Securely access web applications and VMs |
| **How It Works** | Encrypted tunnel between networks | User authentication and authorization |
| **Security** | Encrypts all data through the tunnel | Requires user login, enforces access controls |
| **Access Method** | Network-based access | Identity-based access |
| **Use Cases** | Hybrid cloud, extending on-premises network | Securing web apps, remote VM access |
| **Setup Complexity** | Generally more complex, network configuration | Generally simpler, configure access policies |
| **Example Use Case** | Secure on-premises database access from GCP | Restricting web app access to authorized users |

### Summary

* **VPN** is ideal for securely connecting entire networks, allowing seamless integration between on-premises and cloud resources. It encrypts all traffic between the connected networks.
* **IAP** is best for securing access to specific web applications and VMs in GCP based on user identity and permissions. It does not require a VPN and provides fine-grained access control.

# VPC Firewall Rules

In a **Virtual Private Cloud (VPC) firewall**, ingress and egress rules control the flow of traffic into and out of your VPC. Think of them as gates that you can open or close to allow or block certain types of traffic.

**Control Access**: Firewall rules help you control which IP addresses or networks can access your VMs. For example, you can allow only specific IP addresses to access your web servers.

**Block Unauthorized Traffic**: You can block traffic from untrusted sources, protecting your VMs from malicious attacks.

**Ingress Rules:**

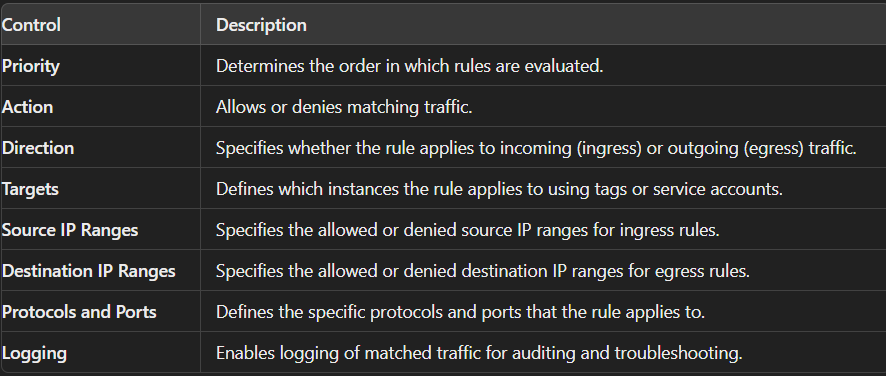
* **Definition:** Ingress rules control incoming traffic.
* **Purpose:** These rules determine what kind of traffic is allowed to enter your VPC from the outside world.
* **Example:** If you have a web server in your VPC, you might set up an ingress rule to allow traffic on port 80 (HTTP) and port 443 (HTTPS) so that people can access your website.

**Egress Rules:**

* **Definition:** Egress rules control outgoing traffic.
* **Purpose:** These rules determine what kind of traffic is allowed to leave your VPC and reach the outside world.
* **Example:** If your servers need to access external services, like fetching updates from the internet, you would set up an egress rule to allow traffic on the necessary ports, such as port 80 (HTTP) for web access.

## Firewall Rules Configuration

Firewall rules in GCP help you control traffic to and from your virtual machine (VM) instances.



|  |  |
| --- | --- |

### 1. **Name and Description**:

* **Name**: A unique identifier for the firewall rule.
* **Description**: A brief explanation of what the rule does or why it exists.

### 2. **Network**:

* **Network**: Specifies which Virtual Private Cloud (VPC) network the rule applies to. A VPC network is like a private, isolated network in the cloud.

### 3. **Priority**:

* **Priority**: A number that determines the rule's importance relative to other rules. Lower numbers indicate higher priority. If multiple rules apply to the same traffic, the one with the lowest priority number is applied first.

### 4. **Direction of Traffic**:

* **Ingress**: Rules for incoming traffic to instances.
* **Egress**: Rules for outgoing traffic from instances.

### 5. **Action on Match**:

* **Allow**: Permits the specified traffic.
* **Deny**: Blocks the specified traffic.

### 6. **Targets**:

* **All instances in the network**: The rule applies to all VM instances in the network.
* **Specified target tags**: The rule applies only to instances with certain tags. Tags are labels you can assign to instances.
* **Specified service accounts**: The rule applies only to instances with certain service accounts.

### 7. **Source Filter (for Ingress Rules)**:

* **IP ranges**: Specifies the IP address ranges of incoming traffic.
* **Source tags**: Applies the rule to traffic from instances with specific tags.
* **Source service accounts**: Applies the rule to traffic from instances with specific service accounts.

### 8. **Destination Filter (for Egress Rules)**:

* **IP ranges**: Specifies the IP address ranges of outgoing traffic.
* **Destination tags**: Applies the rule to traffic going to instances with specific tags.
* **Destination service accounts**: Applies the rule to traffic going to instances with specific service accounts.

### 9. **Protocols and Ports**:

* **Protocols**: Specify which network protocols (e.g., TCP, UDP, ICMP) the rule applies to.
* **Ports**: Define which ports the rule applies to. For example, port 80 for HTTP traffic or port 443 for HTTPS traffic.

# IAM

**IAM** (**Identity and Access Management**) is like a security system that helps you control who can do what with your GCP resources.

It ensures that the right people have the right access to the right resources

## Key Concepts:

**Identity:** Represents who is trying to access your resources. It could be a user (like you or your colleague), a group of users, or a service account (a special account for applications and services).

**Roles:** Define what actions (permissions) an identity can perform on your resources. There are three types of roles:

* **Primitive Roles:** Basic roles like Owner, Editor, and Viewer.
* **Predefined Roles:** More granular roles designed for specific tasks (e.g., Storage Admin, Compute Admin).
* **Custom Roles:** Roles you create to tailor permissions precisely to your needs.

**Resources:** The objects you manage in GCP, like virtual machines, databases, storage buckets, etc.

**Policy:** A collection of rules that define who has what type of access to which resources. Each policy binds one or more roles to one or more identities.



## Service Account

A **Service account** in Google Cloud Platform (GCP) is a special kind of account used by applications and virtual machines (VMs) to interact with GCP services. Here’s a simple breakdown:

**What is a Service Account?**

**Purpose**: Instead of a human user, it's for applications or VMs that need to access GCP resources.

**Identity**: Acts like an identity for your app or VM to perform tasks within your project.

**VMs**: When you run a VM, you can attach a service account so it can access GCP resources (like Cloud Storage or Cloud SQL).

**Applications**: When your application runs on platforms like Kubernetes or App Engine, it uses a service account to interact with other GCP services.

**Types of Service Accounts**

1. **User-managed**: Created and managed by you. You control the keys and roles.
2. **Google-managed**: Created and managed by GCP for specific services. You don't handle keys for these accounts.

**How It Works**

1. **Create**: You create a service account in the GCP Console or via command-line tools.
2. **Assign Roles**: You assign necessary roles to the service account, defining what it can do.
3. **Generate Keys**: You generate a key (typically a JSON file) that your application uses to authenticate as the service account.
4. **Use**: Your application uses this key to make authenticated API calls to GCP services.

**Example Scenario**

Imagine you have an application running on a VM that needs to upload files to Cloud Storage. Here’s what happens:

**Create a Service Account** : You create a service account with the necessary permissions (e.g., Storage Object Creator).

**Assign to VM** : You assign this service account to your VM.

**Authenticate** : Your application on the VM uses the service account to authenticate and interact with Cloud Storage.

# Storage in GCP

## Block Storage

### Google Cloud Persistent Disks

* A type of storage that provides raw block-level storage.
* Acts like a physical hard drive but is virtualized.
* **Durability**: Data is stored redundantly to ensure reliability.
* **Performance**: High performance for databases and applications needing fast access.
* **Flexibility**: Can be used with Google Compute Engine VMs.

**Types**:

1. **Standard Persistent Disks**:
   * Good for regular workloads.
   * More cost-effective.
2. **Balanced Persistent Disks**:
   * A balance between performance and cost.
   * Suitable for a variety of workloads.
3. **SSD Persistent Disks**:
   * High performance for I/O-intensive applications.
   * Faster than standard disks but more expensive.

**How to Use It**:

* Running a MySQL or PostgreSQL database on a Compute Engine VM, where the database needs fast read/write access to the disk.
* When creating a VM, you can attach a persistent disk.
* You can resize the disk without downtime.
* Create snapshots for backup and recovery.

## File Storage

### Google Cloud Filestore

* A managed file storage service.
* Provides a file system interface that VMs can mount and use.
* **Managed Service**: GCP handles maintenance and scaling.
* **Compatibility**: Works with standard file system protocols (NFS).
* **Performance**: High performance for applications needing shared file systems.

**Use Cases**:

* **Shared Storage**: When multiple VMs need access to the same data.
* **Legacy Applications**: Applications designed to work with file systems.
* **Content Management**: Storing large numbers of files that need to be accessed by different services.

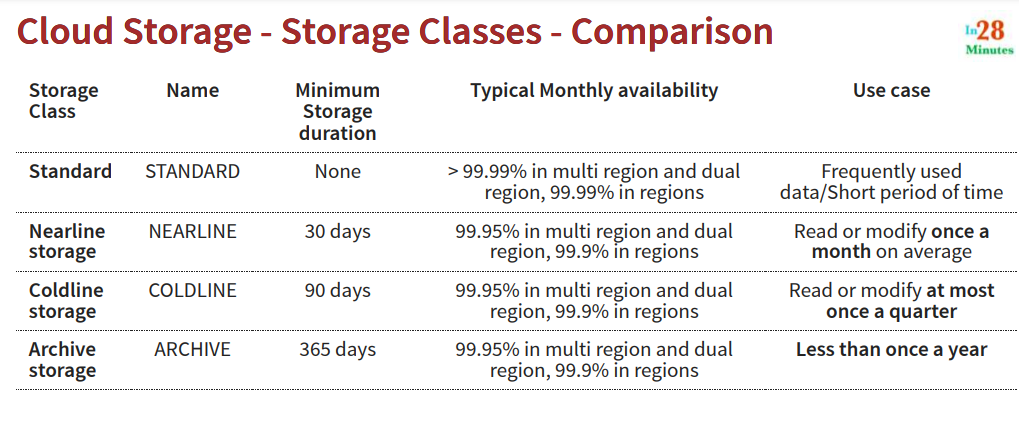
## Object Storage

### Google Cloud Storage

* **Type of Storage**: Object storage, which means data is stored as objects (files), rather than blocks or files within a file system.
* **Purpose**: Used for storing any type of data such as images, videos, backups, logs, and more.
* **Scalability**: Automatically scales to handle vast amounts of data.

**Key Features**

1. **Buckets**: Containers for your objects (files).
   * **Creating a Bucket**: You create a bucket to store your objects. Each bucket has a globally unique name.
   * **Bucket Settings**: Configure settings like location (region/multi-region), storage class, and access controls.
2. **Objects**: The individual pieces of data (files) you store.
   * **Uploading**: You upload objects into buckets.
   * **Metadata**: Each object can have associated metadata like content type and size.
3. **Storage Classes**: Different tiers for balancing cost and access needs.
   * **Standard**: For frequently accessed data.
   * **Nearline**: For data accessed less than once a month.
   * **Coldline**: For data accessed less than once a year.
   * **Archive**: For long-term storage, rarely accessed data.

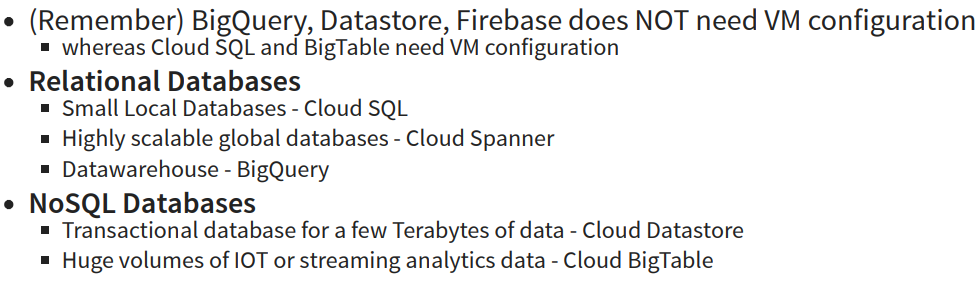


1. **Access Control**: Manage who can view or manage your data.
   * **IAM Roles**: Assign roles to users, groups, or service accounts to control access at the bucket or project level.
   * **Access Control Lists (ACLs)**: Fine-grained control at the object level.
2. **Security**: Data encryption and management.
   * **Encryption**: Data is encrypted at rest and in transit.
   * **Bucket Policies**: Enforce security rules, such as requiring certain conditions for object access.
3. **Lifecycle Management**: Automate data retention policies.
   * **Rules**: Set rules to move objects between storage classes, delete objects after a certain time, etc.

**Common Use Cases**

1. **Storing Media**: Store images, videos, and other media files for websites and applications.
2. **Backups**: Keep backups of your databases and applications.
3. **Big Data Analytics**: Store large datasets for analysis with GCP's data processing tools.
4. **Archives**: Long-term storage of infrequently accessed data.

# Google Cloud Databases

****

## Google Cloud Storage

* **Type**: Object Storage
* **Use Cases**: Storing unstructured data like images, videos, backups, and logs.
* **Features**: Highly scalable, different storage classes (Standard, Nearline, Coldline, Archive), lifecycle management, data encryption.

## Cloud SQL

* **Type**: Managed Relational Database
* **Use Cases**: Relational databases with support for MySQL, PostgreSQL, and SQL Server.
* **Features**: Automated backups, high availability, vertical scaling.

## Google Cloud Spanner

* **Type**: Horizontally Scalable Relational Database
* **Use Cases**: Global-scale applications needing relational database consistency and horizontal scalability.
* **Features**: Global distribution, strong consistency, automatic sharding.

## Google BigQuery

* **Type**: Serverless Data Warehouse
* **Use Cases**: Analyzing large datasets using SQL, business intelligence, real-time analytics.
* **Features**: Fully managed, scalable, supports standard SQL, built-in machine learning capabilities.

## Google Firestore

* **Type**: NoSQL Document Database
* **Use Cases**: Your application requires real-time data synchronization and offline capabilities. Real-time data synchronization for mobile, web, and server applications.
* **Features**: Serverless, real-time updates, offline support.

## Google Cloud Bigtable

* **Type**: NoSQL Wide-Column Database
* **Use Cases**: Large analytical and operational workloads, time-series data, IoT data.
* **Features**: High throughput, low latency, scalable.

## Google Cloud Datastore

* **Type**: NoSQL Document Database
* **Use Cases**: Web and mobile applications requiring structured data at scale.
* **Features**: Fully managed, scalable, ACID transactions.

## Google Cloud Memorystore

* **Type**: Managed In-Memory Data Store
* **Use Cases**: Caching, real-time analytics, session management.
* **Features**: Supports Redis and Memcached, fully managed, high availability.

# Data Processing and Analytics Services

## Google Dataflow

* **Type**: Stream and Batch Data Processing
* **Use Cases**: Real-time analytics, ETL (extract, transform, load), data pipelines.
* **Features**: Serverless, unified programming model, integration with Apache Beam.

## Google Pub/Sub

* **Type**: Messaging Service
* **Use Cases**: Event-driven systems, data streaming, real-time analytics.
* **Features**: Global message delivery, real-time message processing, scalability.

## Google Dataproc

* **Type**: Managed Spark and Hadoop Service
* **Use Cases**: Big data processing using Apache Hadoop and Spark.
* **Features**: Easy cluster management, integration with other GCP services, cost-effective.

## Google Data Studio

* **Type**: Data Visualization and Reporting
* **Use Cases**: Creating interactive dashboards and reports.
* **Features**: Connects to multiple data sources, customizable visualizations, collaborative features.

## Google Composer

* **Type**: Managed Workflow Orchestration Service
* **Use Cases**: Orchestrating data pipelines and workflows.
* **Features**: Based on Apache Airflow, integrated with GCP, supports complex workflows.