Introducing Google Cloud

Cloud Computing Overview

Cloud computing is an IT model defined by five key traits:

1. **On-Demand & Self-Service**: Users can access computing resources (processing, storage, and network) via a web interface without human intervention.
2. **Internet Accessibility**: Resources are available from anywhere with an internet connection.
3. **Resource Pooling**: The cloud provider manages a large pool of resources and distributes them efficiently, reducing costs for users.
4. **Elasticity**: Customers can scale resources up or down as needed, ensuring flexibility.
5. **Pay-as-You-Go Pricing**: Users only pay for the resources they use or reserve, with no upfront investment.

Evolution of Cloud Computing

Cloud computing evolved in three major waves:

1. **Colocation (First Wave)**: Companies rented physical space in data centers instead of owning infrastructure.
2. **Virtualized Data Centers (Second Wave)**: Businesses maintained and controlled virtualized infrastructure, similar to traditional data centers but with increased flexibility.
3. **Container-Based Cloud (Third Wave)**: Google pioneered an **automated, elastic cloud** where services dynamically provision infrastructure, leading to higher efficiency and scalability.

The Future of Cloud Computing

- Google believes that all companies will become **data-driven technology companies**.
- High-quality **software and data** will be key competitive differentiators.
- Google Cloud provides access to this **third-wave cloud architecture**, allowing businesses to leverage **automation, scalability, and advanced cloud technologies**.

Cloud Service Models: IaaS, PaaS, SaaS, and Serverless

Cloud computing has evolved into different service models to meet various business needs:

1. Infrastructure as a Service (IaaS)

- Provides **virtualized compute, storage, and networking** similar to physical data centers.
- Users manage their own infrastructure.
- Example: **Google Compute Engine**.

- **Pricing Model**: Pay for allocated resources **ahead of time**.

2. Platform as a Service (PaaS)

- Provides a **managed environment** where developers focus on application logic without handling infrastructure. Let's you bind your application code to libraries that give access to the infrastructure your application needs.
- Example: **Google App Engine**.
- **Pricing Model**: Pay for **resources actually used**.

3. Serverless Computing

- Further eliminates infrastructure management by automatically provisioning and scaling resources.
- **Google Serverless Services**:
    - **Cloud Run**: Deploy containerized applications in a fully managed environment.
    - **Cloud Functions**: Execute event-driven code on a pay-as-you-go basis.

4. Software as a Service (SaaS)

- Delivers **fully managed cloud applications** that users access via the internet.
- No installation required on local devices.
- Examples: **Gmail, Google Docs, Google Drive (Google Workspace)**.

Benefits of Managed Services & Serverless Computing

- Reduces time and costs spent on infrastructure maintenance.
- Allows businesses to focus on their **core applications** and deliver products faster.
- Enhances **scalability, reliability, and efficiency**.

Cloud computing continues to evolve, enabling businesses to operate with **greater flexibility and efficiency** through managed services and serverless architectures.

Google Cloud's Global Network and Infrastructure

Google Cloud operates on **Google's own global network**, one of the largest and most advanced in the world. This infrastructure is built to deliver **high throughput and low latency** for applications, ensuring fast and reliable access to services.

Key Features of Google Cloud's Network

1. **Content Caching Nodes**:
    - Over **100 caching nodes worldwide** store frequently accessed content closer to users for faster response times.
2. **Geographic Locations & Redundant Cloud Infrastructure**:
    - Google Cloud is distributed across **five major regions**:

        - **North America**
        - **South America**
        - **Europe**
        - **Asia**
        - **Australia**
    - This ensures **high availability and durability** for cloud applications.
3. **Regions and Zones**:
    - A **region** is an **independent geographic area** that contains multiple zones.
    - A **zone** is a specific **deployment area** within a region where cloud resources (such as virtual machines) are hosted.
    - Example: **London (europe-west2)** has **three zones** for redundancy and scalability.
4. **Multi-Region Support**:
    - Some services allow deployment in **multi-region configurations**, enabling **low-latency access and disaster recovery**.
    - Example: **Spanner multi-region configuration** replicates data across zones in different regions (e.g., The Netherlands and Belgium).
5. **Global Scale & Growth**:
    - Google Cloud currently has **121 zones across 40 regions**, with new locations continuously added.
    - The latest locations can be found at **cloud.google.com/about/locations**.

Why This Matters for Businesses?

- **Better Performance**: Users receive responses from the closest location, reducing delays.
- **Higher Availability**: Data and applications can be distributed across multiple regions for **disaster recovery**.
- **Scalability & Redundancy**: Businesses can run resources across different zones to **avoid single points of failure**.

By leveraging Google Cloud's **cutting-edge global network**, businesses can **enhance speed, reliability, and scalability** while ensuring that services are always available to users worldwide.

Google Cloud's Commitment to Sustainability

Google Cloud is not just about cutting-edge technology—it's also deeply committed to **environmental sustainability**. Data centers worldwide consume about **2% of the world's electricity**, so Google is continuously working to reduce its **carbon footprint** while helping customers achieve their own environmental goals.

Google Cloud's Sustainability Initiatives

Initiative	Description

ISO 14001 Certification	Google's data centers were the **first to achieve ISO 14001**, a globally recognized environmental management certification that ensures resource efficiency and waste reduction.
Energy-Efficient Data Centers	Facilities like **Hamina, Finland**, use **sea water cooling** from the **Bay of Finland**, reducing energy consumption—**a world-first innovation** in data center cooling.
Carbon Neutral Since 2007	Google was the **first major company** to achieve **carbon neutrality** in its **founding decade** (2007).
100% Renewable Energy Since 2017	Since 2017, Google Cloud has matched **100% of its energy consumption with renewable sources** like wind and solar.
Carbon-Free Operations by 2030	Google aims to be **completely carbon-free** by 2030, using only **clean energy 24/7**, a goal that no other major cloud provider has yet achieved.

Why This Matters for Google Cloud Customers?

- **Lower Carbon Footprint**: Running workloads on Google Cloud helps businesses reduce their own environmental impact.
- **Energy Efficiency**: Google's advanced cooling and energy-saving techniques lower power consumption, making cloud computing more sustainable.
- **Sustainable Growth**: Businesses can align with **green initiatives** without compromising performance, security, or scalability.

Google Cloud not only provides cutting-edge cloud solutions but does so **in an environmentally responsible way**, helping both businesses and the planet thrive.

Google Cloud's Multi-Layered Security Approach

Security is at the core of Google Cloud's design, ensuring **data protection, user authentication, encryption, and attack prevention** at every level. With over **a billion users relying on Google services**, security measures are constantly evolving.

Google Cloud's Security Layers

Google Cloud's security is structured in **progressive layers**, starting from **physical security** up to **operational security**.

1. Hardware Infrastructure Layer

🔹 **Custom Hardware & Security Chips**: Google designs its own **server boards and networking equipment** with **built-in security**.

🔹 **Secure Boot Stack**: Ensures only **authenticated and verified software** runs on Google machines.

🔹 **Premises Security**: Google's data centers have **strict access controls**, with **limited employee access** and additional layers of security in third-party data centers.

2. Service Deployment Layer

🔹 **Encryption of Inter-Service Communication**:

- All internal **Remote Procedure Call (RPC) traffic is encrypted**.
- **Cryptographic accelerators** are being deployed for **enhanced security** inside Google data centers.

3. User Identity Layer

🔹 **Beyond Password Authentication**:

- Google's login system evaluates **risk factors** like **device, location, and behavior**.
- Supports **2-Factor Authentication (2FA)** and **Universal 2nd Factor (U2F) security keys** to prevent phishing attacks.

4. Storage Services Layer

🔹 **Encryption at Rest**:

- All stored data is **encrypted by default** using centrally managed keys.
- Google uses **hardware encryption** in SSDs and hard drives for extra security.

5. Internet Communication Layer

🔹 **Google Front End (GFE)**:

- All TLS (HTTPS) connections use **public-private key pairs** and **X.509 certificates**.
- Supports **Perfect Forward Secrecy (PFS)** to protect past communications even if encryption keys are compromised.

🔹 **Denial-of-Service (DoS) Protection**:

- Google's **massive infrastructure** absorbs **large-scale DDoS attacks**.
- Multi-tier **DoS protection mechanisms** minimize attack risks.

6. Operational Security Layer

- **Intrusion Detection**:
  - **Machine learning-based monitoring** and **Red Team exercises** to test security.
- **Insider Threat Protection**:
  - Google **strictly limits employee access** to infrastructure.
  - Active **monitoring of privileged accounts**.
- **Secure Development Practices**:
  - **Code reviews** by multiple engineers prevent security vulnerabilities.
  - **Automated security libraries** help developers write safe code.
- **Vulnerability Rewards Program**:
  - Google **pays security researchers** who find and report **bugs** in its infrastructure.

Why Google Cloud Security Stands Out

1. **Built-in, End-to-End Security** → From **hardware to cloud applications**, security is integrated at every level.
2. **Multi-Layered Defense** → Physical security, **encryption, access control, and attack prevention** ensure **robust protection**.
3. **Scale & Automation** → Google's **AI-driven security** detects and responds to threats **faster than traditional security models**.
4. **Security Investment** → Google invests **billions of dollars** in security research and infrastructure upgrades.

With **Google Cloud**, businesses get **enterprise-grade security** without the burden of managing it themselves.

Google Cloud's Commitment to Avoiding Vendor Lock-In

Many organizations fear **vendor lock-in** when moving to the cloud, but **Google Cloud provides flexibility** for customers to migrate or run workloads elsewhere if needed.

How Google Ensures Interoperability

1. **Open-Source Technology**:
   - Google **publishes key technologies as open source** to give customers alternatives beyond Google.
   - **Example**: **TensorFlow**, a machine learning library, fosters a strong open-source ecosystem.

2. **Cross-Cloud Compatibility**:

    - **Kubernetes & Google Kubernetes Engine (GKE)** enable customers to **run microservices across multiple cloud providers**.
    - **Google Cloud Observability** helps monitor workloads **across different cloud environments**.

By prioritizing **openness and flexibility**, Google Cloud ensures customers are not restricted to its platform, allowing **easy migration and multi-cloud strategies**.

Google Cloud Pricing & Cost Management

Google Cloud offers flexible pricing with various cost-saving features to help businesses manage their cloud expenses efficiently.

Key Pricing Features

1. **Per-Second Billing**
    - Google was the **first major cloud provider** to introduce per-second billing.
    - Now available for **Compute Engine, Kubernetes Engine, Dataproc (Hadoop as a service), and App Engine flexible VMs**.
    - You only pay for what you use—**no rounding up to the nearest minute or hour**.
2. **Automatic Discounts**
    - **Sustained-Use Discounts**: If you run a **virtual machine (VM) for more than 25% of a month**, you automatically get discounts on every extra minute.
    - **Custom VM Types**: Instead of using pre-set machine types, you can **fine-tune CPU and memory** to match your needs and avoid overpaying.

3. **Cost Estimation & Budget Control**
    - **Pricing Calculator**: Helps estimate costs before you commit (Try it here).
    - **Budgets & Alerts**:
        - Set spending limits at the **billing account or project level**.
        - Example: If your budget is **$20,000**, you can set an alert at **90% ($18,000)** to get notified before hitting the limit.
        - Custom alerts can also be set at **50%, 90%, or 100% of your budget**.
    - **Reports in Google Cloud Console**: A visual tool to track spending across different projects or services.
4. **Quota System to Prevent Unexpected Costs**
    - **Why?** To protect you from errors or malicious attacks that could suddenly increase usage.
    - **Types of Quotas**:
        - **Rate Quotas**: Reset after a fixed time. (Example: **GKE allows 3,000 API calls per project every 100 seconds**).

        - **Allocation Quotas**: Limit how many resources you can create (Example: **Each project can have up to 15 Virtual Private Cloud networks by default**).
    - You can **request an increase** in quotas if needed.

Google Cloud gives you **complete control over your spending** with **smart pricing, built-in discounts, budgeting tools, and protective quotas**. You won't wake up to unexpected bills—**you decide how much to spend!**

Key Learning from this module

Cloud computing is built on five essential traits: **on-demand access**, **internet availability**, **pooled resources**, **elasticity**, and **pay-as-you-go pricing**, which have evolved from **physical colocation** to **virtualized data centers**, and now to **container-based**, **highly automated cloud environments** pioneered by **Google**. **Google Cloud** empowers businesses to operate like **data-driven tech companies** by offering a **modern third-wave architecture** that emphasizes **automation**, **scalability**, and **security**.

The cloud operates on service models like **IaaS (Infrastructure as a Service)**, **PaaS (Platform as a Service)**, **SaaS (Software as a Service)**, and **Serverless**—each offering different levels of **abstraction** and **control**. **Serverless**, in particular, **eliminates infrastructure management altogether**, letting developers **focus purely on code**.

Google Cloud leverages its **powerful global network infrastructure**—with over **100 caching nodes** and **121 zones across 40 regions**—to ensure **high performance**, **low latency**, and **maximum uptime**.

Their **commitment to sustainability** is unparalleled, aiming for **24/7 carbon-free operations by 2030**, and has already achieved milestones like **carbon neutrality since 2007** and **100% renewable energy usage since 2017**.

Google Cloud also stands out with its **multi-layered security approach** that spans from **physical hardware to app-level operations**—featuring **encryption**, **access controls**, **intrusion detection**, and a **zero-trust model**.

They reduce **vendor lock-in risk** by embracing **open-source tech** and **multi-cloud compatibility**, allowing **seamless migration and integration**.

Lastly, **Google Cloud's flexible pricing**, **per-second billing**, **automatic discounts**, and **budgeting tools** offer full **cost control** and **transparency**, protecting users from **unexpected bills** while ensuring **efficient cloud resource usage**.

Resources and Access in the cloud

Google Cloud's Functional Structure: Simplified Overview

Google Cloud organizes its resources in a structured hierarchy with four levels:

1. **Resources (Bottom Level)** – These include virtual machines, Cloud Storage buckets, and BigQuery tables.
2. **Projects** – These group related resources together and serve as the foundation for enabling services, managing APIs, and handling billing.
3. **Folders** – Used to organize multiple projects, allowing for better policy and permission management.
4. **Organization Node (Top Level)** – This is the highest level and contains all projects, folders, and resources of a company.

How Policies Work

Policies can be applied at any level (organization, folder, project) and automatically apply to everything below them. For example, a policy set at the folder level will apply to all projects within that folder.

Projects in Detail

- Each project is a separate entity with its own billing, management, and access control.
- Every project has three unique identifiers:
    - **Project ID (permanent and unique)**
    - **Project Name (user-defined and changeable)**
    - **Project Number (used internally by Google Cloud)**
- The **Resource Manager tool** helps in creating, updating, and deleting projects programmatically.

Using Folders for Better Management

- Folders help structure projects into departments or teams within an organization.
- Permissions and policies set on a folder apply to all resources within it.
- This simplifies management and prevents errors from manually duplicating policies across projects.

The Role of the Organization Node

- It acts as the central hub that holds all folders, projects, and resources.
- It allows organizations to assign special roles such as:
    - **Policy Administrators** (control who can modify policies)
    - **Project Creators** (control who can create projects and spend money)
- Companies using **Google Workspace** automatically have an organization node. Others can create one using **Cloud Identity**.

This structured approach ensures better resource management, security, and cost control within Google Cloud.

Understanding IAM in Google Cloud

When an organization has many projects, folders, and resources, it's essential to control who has access to what. Google Cloud uses **Identity and Access Management (IAM)** to help administrators manage this.

How IAM Works

IAM policies define:

1. **Who** (the principal) can access resources – This can be a Google account, group, service account, or Cloud Identity domain.
2. **What they can do** (their role) – A role is a set of permissions that control what actions a principal can perform.

IAM follows a hierarchy, meaning when a role is assigned to a resource, it also applies to everything below it in the structure.

Types of Roles in IAM

IAM has three types of roles:

1. **Basic Roles** – These are broad and apply to an entire project:
    - **Viewer** – Can only view resources.
    - **Editor** – Can view and modify resources.
    - **Owner** – Can modify resources, manage permissions, and handle billing.
    - **Billing Administrator** – Can manage billing without modifying resources.
    - **Caution:** Basic roles might be too broad for projects with sensitive data.
2. **Predefined Roles** – Google Cloud provides service-specific roles.
    - Example: In **Compute Engine**, the "instanceAdmin" role allows managing virtual machines at different levels (project, folder, or organization).
    - These roles provide **better control than basic roles** since they are designed for specific services.
3. **Custom Roles** – Created by organizations for fine-grained control.
    - Example: A company could create an **"instanceOperator"** role to allow stopping and starting virtual machines but not reconfiguring them.
    - **Limitations:**
        - Requires manual permission management.
        - Can only be assigned at the **project or organization level** (not at the folder level).

Deny Rules in IAM

IAM policies also allow **deny rules**, which block specific principals from using certain permissions, even if they have a role that would normally grant them access. Deny policies take priority over allowed policies and apply down the hierarchy.

Conclusion

IAM in Google Cloud ensures secure and organized access control. While **basic roles** are broad, **predefined roles** help tailor access based on services, and **custom roles** allow precise control. Organizations often follow the **least-privilege model**, giving users only the permissions they need to perform their jobs.

Understanding Service Accounts in Google Cloud

When a program running on a **Compute Engine virtual machine (VM)** needs to access other cloud services, it shouldn't require manual permission every time. Instead, we use **service accounts** to automate access securely.

What is a Service Account?

A **service account** is a special kind of account that a virtual machine or application can use to authenticate itself to other Google Cloud services **without human intervention**.

Why Use Service Accounts?

- Suppose you have an **application running on a VM** that needs to **store data in Cloud Storage**.
- You don't want to make this data public, but you need the application to access it.
- Instead of using a human account, you create a **service account** that lets the VM securely interact with Cloud Storage.

How Do Service Accounts Work?

- **Identified by an email address** (but they don't use passwords).
- **Use cryptographic keys** to securely authenticate and access resources.
- **Can have specific roles** assigned to them.
    - Example: If a service account has the **Instance Admin role**, then any application running on a VM using this account can create, modify, and delete other VMs.

Managing Service Accounts

Service accounts **themselves** are also **Google Cloud resources**, meaning you can control **who can manage them** using IAM policies.

- Example:
    - **Alice** needs to manage service accounts → She gets the **editor role**.
    - **Bob** only needs to see a list of service accounts → He gets the **viewer role**.

Key Takeaway

Service accounts **allow secure, automated access** between cloud services **without human involvement**, while still being **managed through IAM roles** like any other Google Cloud resource.

Managing User Access in Google Cloud with Cloud Identity

When new users start using **Google Cloud**, they often log in with a **Gmail account** and use **Google Groups** to collaborate. While this approach is simple, it can cause problems later because user access **isn't centrally managed**.

Challenges of Using Individual Gmail Accounts

- If a team member **leaves** the company, there's **no quick way** to remove their access to cloud resources.
- Managing users across multiple cloud services can become difficult over time.

Solution: Cloud Identity

Google offers **Cloud Identity**, a tool that allows organizations to **centrally manage** users and groups using the **Google Admin Console**.

How Cloud Identity Helps:

- **Centralized User Management** – Admins can control user access from one place.
- **Seamless Integration** – Works with **Active Directory (AD)** or **LDAP** systems, so employees can use their existing usernames and passwords.
- **Quick User Removal** – When someone **leaves** the company, their account can be **immediately disabled** to prevent unauthorized access.
- **Two Editions Available:**
    1. **Free Edition** – Basic user and group management.
    2. **Premium Edition** – Includes mobile device management.

Google Workspace Customers

If your company already uses **Google Workspace**, Cloud Identity features are **built-in** and can be accessed via the **Google Admin Console**.

Key Takeaway

Using **Cloud Identity** ensures that access to **Google Cloud resources** is **secure, organized, and easy to manage**, reducing security risks when employees leave the company.

Four Ways to Access and Use Google Cloud

There are **four main ways** to interact with **Google Cloud**:

1. **Google Cloud Console**
2. **Cloud SDK & Cloud Shell**
3. **Google Cloud APIs**
4. **Google Cloud Mobile App**

1. Google Cloud Console (GUI – Web Interface)

- A **web-based interface** to manage Google Cloud resources.
- Lets you **deploy, scale, and monitor** cloud services.
- Helps **track costs** and **set budgets** to control spending.
- Provides a **search tool** to quickly find resources.
- Allows **direct SSH connection** to virtual machines (VMs) from the browser.

2. Cloud SDK & Cloud Shell (Command-Line Tools)

Cloud SDK (Software Development Kit):

- A set of **command-line tools** for managing Google Cloud.
- Includes:
    - **Google Cloud CLI (`gcloud`)** – The main tool for managing cloud services.
    - **`bq` (BigQuery tool)** – Used for managing large datasets.

Cloud Shell (Browser-based Command Line):

- A **Debian-based virtual machine** that runs in the browser.
- Comes with **5GB of storage** and pre-installed Google Cloud tools.
- Always **up-to-date and authenticated**, making it secure and ready for use.

3. Google Cloud APIs (For Developers & Automation)

- Every Google Cloud service has an **API (Application Programming Interface)** that allows apps to interact with it.
- **Google APIs Explorer** helps users see available APIs and test them.
- **Google provides client libraries** in popular programming languages to simplify API usage:
    - **Java, Python, PHP, C#, Go, Node.js, Ruby, and C++**

4. Google Cloud App (Mobile App)

What can you do with the app?

- **Start/Stop** virtual machines (**Compute Engine instances**).
- **Monitor logs and metrics** (CPU usage, network activity, requests per second).
- **Manage Cloud SQL databases** (start/stop instances).
- **Handle App Engine deployments** (rollback, split traffic).
- **View billing information** and receive alerts if projects go over budget.
- **Incident management** – get notified about issues.

Where to get it?
The app is available at **cloud.google.com/app**

Conclusion

Google Cloud provides multiple ways to interact with its services:

- **Cloud Console** for an easy-to-use web interface.
- **Command-line tools (SDK & Shell)** for scripting and automation.
- **APIs** for integrating with custom applications.
- **Mobile App** for quick monitoring and management.

Each method serves different needs based on how you want to manage your cloud resources.

Key Learning from this module

Google Cloud follows a structured hierarchy for resource management, ensuring efficient organization, security, and cost control. At the top level, the Organization Node holds all folders, projects, and resources, while projects act as individual entities with separate billing and access controls. IAM (Identity and Access Management) plays a crucial role in defining who can access resources and what actions they can perform, offering basic, predefined, and custom roles to suit different needs. Service accounts enable secure, automated access between cloud services without human intervention, ensuring smooth operations for applications. To streamline user management, Cloud Identity helps organizations centrally control access, reducing security risks. Google Cloud offers multiple ways to interact with its services, including the Cloud Console for a graphical interface, Cloud SDK & Shell for command-line operations, APIs for automation, and a mobile app for quick management on the go. This comprehensive structure allows businesses to efficiently manage their cloud infrastructure while maintaining security and operational efficiency.

Virtual Machines and networks in the cloud

Google Cloud Virtual Private Cloud (VPC) & Networking

A **Virtual Private Cloud (VPC)** in Google Cloud is a **private cloud environment** that runs within a **public cloud infrastructure**. It allows users to securely run applications, store data, and manage resources while benefiting from the **scalability** and **flexibility** of public cloud services.

Key Features of Google Cloud VPC

1. **Private Yet Scalable**
    - A **VPC** provides data **isolation** while still leveraging the **public cloud's** computing power. It can host **websites, store data, and run applications** securely.
2. **Connectivity**
    - VPC networks connect Google Cloud resources to each other and the internet, with firewall rules controlling access and static routes directing traffic to specific destinations.
3. **Global Scope**
    - Unlike traditional networks, **Google Cloud VPC networks are global,** having **multiple subnets across different Google Cloud regions** spanning into **multiple zones** within a region.
4. **Subnet Expansion**
    - A **subnet's size** can be increased by expanding its **IP address range** without **disrupting** existing virtual machines (VMs).
5. **Example Scenario**
    - Imagine a **VPC network** named **vpc1** with two **subnets** in:
        - **Asia-East1 region**
        - **US-East1 region**
    - If three **Compute Engine VMs** are attached to this VPC, they can communicate as if they were on the same **local network**, even if they are in **different zones** allowing **high availability and resilience** while keeping the **network structure simple**.

Why Use Google Cloud VPC?

- **Security**: Keeps your data private while allowing flexible access controls.
- **Scalability**: Expands as needed without affecting existing resources.
- **Global Connectivity**: Connects resources across different regions seamlessly.
- **Simplified Networking**: Easier management of cloud infrastructure across multiple locations.

What is a Subnet?

A **subnet (subnetwork)** is a **smaller, segmented part of a larger VPC network**. It defines a range of **IP addresses** for resources in a specific **region** within a **Google Cloud VPC**.

Google Cloud VPC provides a **powerful, flexible, and secure** networking environment to build and scale applications globally.

Google Cloud Compute Engine

What is a Compute Engine?

Google Cloud's Compute Engine is an **Infrastructure as a Service (IaaS)** solution that lets users create and run **virtual machines (VMs)** on Google's infrastructure.

Key Features of Compute Engine:

- **No Upfront Investment** – Pay only for what you use.
- **Scalability** – Run thousands of virtual CPUs with high speed and consistent performance.

- **Customizable VMs** – Choose CPU, memory, storage, and operating system (Linux, Windows Server, or custom OS).
- **Multiple Ways to Create VMs** – Use Google Cloud Console, Cloud CLI, or Compute Engine API.
- **Cloud Marketplace** – Get pre-configured solutions from Google and third-party vendors.

Pricing & Discounts:

1. **Billing Per Second** – Minimum charge of 1 minute.
2. **Sustained-Use Discounts** – Automatically applied if a VM runs for **more than 25% of a month**.
3. **Committed-Use Discounts** – Up to **57% off** for committing to 1 or 3 years of usage.

4. **Preemptible & Spot VMs** – Cheaper options (up to **90% savings**) for jobs that can be stopped and restarted if Google needs resources.
    - **Preemptible VMs** – Max runtime of **24 hours**.
    - **Spot VMs** – No max runtime, but same pricing as Preemptible VMs.

Storage & Custom Machine Types:

- **High-speed storage** is included by default at no extra cost.
- **Custom Machine Types** – Choose exact CPU and memory configurations to avoid paying for unused resources.

Autoscaling and Load Balancing in Compute Engine

1. Choosing Machine Properties:

- With **Compute Engine**, you can select the number of virtual CPUs (vCPUs) and memory by using **predefined machine types** or by creating **custom machine types.**

2. Autoscaling Feature:

- Automatically **adds or removes VMs** based on workload demand optimizing **performance and cost efficiency**.

3. Load Balancing:

- Distributes incoming traffic across multiple VMs. VPC supports different types of load balancing.

4. Scaling Strategies:

- **Scaling Up:** Using **large VMs** for tasks like **in-memory databases and analytics**.
- **Scaling Out:** Most users **add more smaller VMs** instead of increasing individual VM size.

5. VM Limitations:

- The **maximum CPUs per VM** depend on its **machine family** and **user quota**, which varies by region.

Virtual Private Cloud (VPC) Compatibility Features

1. Routing in VPCs:

- Like physical networks, **VPCs have built-in routing tables** eliminating the need to manage a **router,** traffic moves within the network, across subnetworks, or between Google Cloud zones **without requiring an external IP**.

2. Firewall Protection:

- **A global distributed firewall** is automatically provided controlling **both incoming and outgoing traffic**.
- Firewall rules can be **easily managed** using **network tags**.
    - Example: Tag web servers as **"WEB"** and allow only **ports 80 and 443** for them.

3. Connecting Multiple VPCs:

- **VPC Peering:** Allows **direct traffic exchange** between VPCs in different projects.

- **IAM Integration:** Provides **granular access control** over interactions between projects and VPCs.

Cloud Load Balancing in Google Cloud

1. Why Load Balancing?

- When your application scales from **4 VMs to 40 VMs**, a **load balancer** ensures users are directed to available instances spreading **user traffic** across multiple VMs, reducing performance issues.

2. Google Cloud Load Balancing:

- **Fully managed, software-defined, and distributed**—no need to manage or scale it yourself.

- Supports **HTTP, HTTPS, TCP, SSL, and UDP traffic**.
- **Cross-region balancing** with **automatic failover**—if a backend becomes unhealthy, traffic is gradually shifted elsewhere.
- **No "pre-warming" required**—Google Cloud automatically handles sudden spikes in traffic.

3. Types of Load Balancers:

Google Cloud offers different load balancers based on the **OSI model layer** they operate at:

Type	**Layer**	**Best For**	**Key Features**
Application Load Balancer	Application (Layer 7)	Web apps (HTTP/HTTPS)	Supports **content-based routing**, **SSL/TLS termination**, acts as a **reverse proxy**.
Proxy Network Load Balancer	Transport (Layer 4)	Managing TCP/UDP/IP traffic	**Acts as a reverse proxy**, supports on-prem and cloud backends.
Passthrough Network Load Balancer	Transport (Layer 4)	Direct traffic forwarding	**Preserves original source IP**, no modification of connections, ideal for **direct server return** apps.

These load balancers provide **scalability, reliability, and high availability** for applications in Google Cloud.

Google Cloud DNS & Content Delivery Network (CDN)

1. Google's Public DNS (8.8.8.8)

- **8.8.8.8** is a free **public DNS service** by Google that translates **internet hostnames to IP addresses** using Google's advanced DNS infrastructure.

2. Cloud DNS – Managed DNS for Google Cloud Applications

- **Cloud DNS** helps users find applications hosted in **Google Cloud**.

- **Key Features:**
    - **Managed service** running on Google's infrastructure.
    - **Low latency & high availability**.
    - **Cost-effective** way to make applications accessible.
    - **Globally redundant**—DNS information is stored across multiple locations worldwide.
    - **Fully programmable**—manage millions of **DNS zones & records** via:
        - **Cloud Console**
        - **Command-line interface (CLI)**
        - **API**

3. Cloud CDN – Content Delivery Network

- Uses **Google's global system of edge caches** to store content closer to users.

- **Benefits:**

    - **Faster content delivery** with **lower latency**.
    - **Reduces server load** by offloading requests to cache servers.
    - **Cost savings** by reducing bandwidth consumption.

- **Enabling Cloud CDN:**
    - After setting up an **Application Load Balancer**, enable **Cloud CDN** with **a single checkbox**.
- **CDN Interconnect:**
    - If you're using a **third-party CDN**, it may be part of **Google Cloud's CDN Interconnect program**, allowing continued use with optimizations.

Connecting Google VPC to Other Networks

Google Cloud offers multiple ways to connect **Google Virtual Private Cloud (VPC)** to **on-premises networks** or **other cloud networks**:

1. Cloud VPN – Secure Internet-Based Connection

- Creates a secure "tunnel" connection over the internet.
- Can be made **dynamic** using **Cloud Router**, which exchanges route information via **Border Gateway Protocol (BGP)**.
- If a **new subnet** is added to Google VPC, the **on-premises network automatically receives routes** to it.

Limitations:

- Relies on the public internet, so security and bandwidth reliability may be concerns.

2. Direct Peering – Connecting to Google's Network

- Places a **router** in the **same public data center** as a **Google Point of Presence (PoP)** to exchange traffic.
- Google has **100+ PoPs worldwide**.
- If the customer is **not in a PoP**, they can use a **Carrier Peering partner** to connect.

Limitations:

- No Google SLA (Service Level Agreement) coverage.

3. Dedicated Interconnect – Private & Reliable

- Establishes **one or more direct, private connections** to Google.
- SLA of up to **99.99%** uptime if Google's **topology requirements** are met.
- Can be **backed up by a VPN** for even higher reliability.

Best for: Businesses needing **high uptime and reliable connectivity**.

4. Partner Interconnect – Flexible Cloud Connectivity

- Provides **connectivity through a Google-approved service provider**.
- Useful when:
    - **The data center is far from a Google colocation facility**.
    - **A full 10 Gbps Dedicated Interconnect** isn't needed.
- Can be configured for **mission-critical or less sensitive workloads**.
- SLA **up to 99.99%**, **but Google does not cover third-party provider issues**.

Best for: Locations **unable to reach a Dedicated Interconnect facility**.

5. Cross-Cloud Interconnect – Connecting Google Cloud to Other Clouds

- **Establishes dedicated high-bandwidth connections** between **Google Cloud and other cloud providers**.
- Reduces **complexity** and **supports multicloud strategies**.
- Available in **10 Gbps or 100 Gbps** connections.
- **Supports encryption** and **site-to-site data transfer**.

Best for: Businesses using **multiple cloud providers (AWS, Azure, etc.)**.

Choosing the Right Option

Option	Connection Type	Security & Reliability	Best Use Case
Cloud VPN	Secure tunnel over the internet	Secure, but dependent on internet bandwidth	Small-to-medium workloads
Direct Peering	Public router at a Google PoP	No SLA, but better than VPN	General-purpose connections
Dedicated Interconnect	Private direct link to Google	SLA up to **99.99%**, highest reliability	Critical applications needing **high uptime**
Partner Interconnect	Private link via 3rd party	SLA up to **99.99%**, depends on provider	**Flexible alternative** if Dedicated Interconnect isn't an option
Cross-Cloud Interconnect	Private link to another cloud provider	High security & encryption	**Multicloud strategy**

Key Learning from this Module

The key learning from this information is that **Google Cloud Virtual Private Cloud (VPC) and networking solutions provide scalable, secure, and globally connected cloud infrastructure**. A **VPC** enables users to isolate their resources while benefiting from cloud

scalability, allowing seamless connectivity between applications, storage, and workloads across multiple regions. **Compute Engine** offers customizable virtual machines (VMs) with autoscaling and load balancing to optimize performance and cost. Networking features like **firewall rules, routing, and VPC peering** ensure secure communication across cloud environments.

For connectivity, Google Cloud offers **various options to integrate VPCs with on-premises or other cloud networks**. **Cloud VPN** provides encrypted internet-based tunnels, while **Direct Peering** allows traffic exchange through Google's global network. **Dedicated Interconnect and Partner Interconnect** offer private, high-reliability connections, with **Cross-Cloud Interconnect** facilitating seamless multicloud strategies. Load balancing and **Cloud CDN** help optimize traffic distribution and content delivery for faster access and cost efficiency.

Overall, **Google Cloud's networking solutions enhance security, performance, and scalability** for businesses operating in the cloud, enabling them to choose the best connectivity method based on their needs for reliability, bandwidth, and security.

Cloud Storage

Introduction to Cloud Storage

Google **Cloud Storage** is a highly **durable, available, and scalable object storage service** designed for **developers and IT organizations**.

What is Object Storage?

Unlike traditional **file storage** (which organizes data in folders) or **block storage** (which divides data into chunks), object storage:

- Manages **data as objects** by storing **binary data**, associated **metadata** (e.g., creation date, author, resource type, and permissions), and a **globally unique identifier**. It Uses **URLs for unique keys**, making it ideal for **web applications**. **Use Cases:** storing **videos, images, and audio files**.

Cloud Storage: Features and Use Cases

Google **Cloud Storage** allows customers to store and retrieve **any amount of data** as needed.

Use Cases:

- Hosting **website content**.
- Storing **backup and archived data**.
- Distributing large objects through **direct download**.
- Storing **intermediate processing results** in workflows.

Key Characteristics:

- **Scalable & Fully Managed**: No need to provision storage manually.
- **Global Accessibility**: Data can be stored in **single-region, multi-region, or dual-region locations**.

Buckets and Object Management

1. Buckets:

- All objects in Cloud Storage are stored in **buckets**.
- Each bucket has:
    - A **globally unique name** and **specific geographic storage location** (e.g., choosing **Europe** if most users are there).

2. Object Immutability:

- Objects **cannot be modified** directly. Instead, a **new version** is created when changes are made.
- Options for handling new versions:
    - **Default**: New versions **overwrite** old ones.
    - **Versioning Enabled**: Maintains a **detailed history** of modifications.

With **object versioning**, you can:

- List **archived versions** of an object.
- **Restore** an object to an earlier version.
- **Permanently delete** an old version if needed.

Security & Access Control

Controlling access to Cloud Storage is essential, especially for **sensitive or personally identifiable information (PII)**.

1. IAM (Identity and Access Management):

- Recommended for most cases.
- **Permissions are inherited** from the **project → bucket → object**.

2. Access Control Lists (ACLs):

- Used for **finer-grained control** when IAM is not enough.
- An ACL consists of:
    - **Scope**: Defines **who** can access (specific users or groups).

- **Permission**: Defines **what actions** (e.g., read/write) are allowed.

Storage Lifecycle Management

Since storing and retrieving large amounts of data can be costly, Cloud Storage supports **lifecycle management policies** to automatically manage objects over time.

Examples:

- **Delete objects older than 365 days**.
- **Remove objects created before a specific date (e.g., January 1, 2013)**.
- **Keep only the latest 3 versions** of each object in a versioned bucket.

This helps **optimize storage costs** by ensuring **unused data is not retained unnecessarily**.

Cloud Storage - Storage Classes & Data Transfer

1. Storage Classes

Storage Class	Best For	Access Frequency	Minimum Storage Duration	Cost Considerations
Standard Storage	Frequently accessed ("hot") data	High	No minimum	Higher cost, suitable for short-term storage
Nearline Storage	Infrequently accessed data	Once a month or less	30 days	Lower cost, good for backups and archives
Coldline Storage	Rarely accessed data	Once every 90 days	90 days	Very low cost, suitable for long-term storage
Archive Storage	Long-term archival and disaster recovery	Less than once a year	365 days	Lowest cost, highest access fees

- **Common Features Across Storage Classes:**
  - Unlimited storage, no minimum object size.
  - Worldwide accessibility, low latency, and high durability.
  - Uniform security, tools, and APIs.

    - **Geo-redundancy** (if stored in multi-region or dual-region) for disaster protection and performance.
    - **Autoclass**: Automatically transitions data between storage classes based on access patterns to optimize cost.

2. Data Security

- **Encryption**:
    - Data is encrypted **before being written to disk** at no extra cost.
    - Data transfer is encrypted using **HTTPS/TLS (Transport Layer Security)**.

3. Data Transfer Methods

- **Small-scale transfers**:
    - **gcloud storage** (Cloud SDK command-line tool).
    - **Cloud Console** (drag-and-drop in Chrome).
- **Large-scale transfers**:
    - **Storage Transfer Service**: Batch transfers from other cloud providers, different regions, or HTTP(S) endpoints.
    - **Transfer Appliance**: Physical device for transferring up to **1 petabyte** of data.
- **Google Cloud Integrations**:
    - **BigQuery & Cloud SQL** (import/export data).
    - **App Engine & Firestore** (stores logs, backups, images).
    - **Compute Engine** (stores startup scripts and images).

Cloud SQL - Managed Relational Database Service

1. Overview

- Fully managed relational databases: **MySQL, PostgreSQL, and SQL Server**.
- Automates maintenance tasks like **patching, updates, backups, and replication**.
- No software installation or maintenance required.

2. Scalability

Resource	**Maximum Capacity**

Processor Cores	128 cores
RAM	864 GB
Storage	64 TB

3. Replication & Backup

- **Replication Types**:
  - From a **Cloud SQL primary instance**.
  - From an **external primary instance**.
  - From **external MySQL instances**.
- **Managed Backups**:
  - **Securely stored** and accessible for restore.
  - Cost covers **seven backups** per instance.

4. Security

- **Encryption**:
  - Data encrypted **in transit** (Google's internal network).
  - Data encrypted **at rest** (database tables, temp files, and backups).
- **Firewall Protection**:
  - Controls **network access** to each database instance.

5. Integration & Accessibility

- **Google Cloud Services**:
  - Works with **App Engine** using standard drivers (**Connector/J for Java**, **MySQLdb for Python**).
  - **Compute Engine** instances can be authorized to access Cloud SQL.
- **External Tools**:
  - Compatible with **SQL Workbench, Toad, and MySQL drivers**.

Cloud Spanner - Globally Distributed Relational Database Service

1. Overview

- Fully managed **relational database service**. It is **horizontally scalable**, **strongly consistent Globally,** handles high throughput (tens of thousands of reads/writes per second), and supports **SQL** with joins and Secondary Indexes. It is used by **Google's mission-critical applications**.

Cloud Firestore - NoSQL Document Database

1. Overview

- **Horizontally scalable, NoSQL cloud database** for mobile, web, and server development. It stores data in documents which are organized into collections. It also supports complex nested objects and subcollections.

2. Data Structure & Querying

- Documents contain **key-value pairs** (e.g., `firstname: John`, `lastname: Doe`)
- Supports **NoSQL queries** to retrieve specific documents or filter data.
- Queries are **indexed by default** for **fast performance**, based on result set size rather than dataset size.

3. Real-Time Synchronization & Offline Support

- Uses **data synchronization** to update connected devices in real time.
- Enables **offline support** by caching actively used data.
- Syncs local changes **back to Firestore** when the device reconnects.

4. Google Cloud Integration

- **Multi-region data replication** for high availability.
- Supports **strong consistency**, **atomic batch operations**, and **real transactions**.

Google Cloud Bigtable

Overview:
Bigtable is Google's NoSQL big data database service, powering core Google services like Search, Analytics, Maps, and Gmail. It is optimized for massive workloads with **low latency and high throughput**, making it suitable for operational and analytical applications.

Use Cases:

Customers choose Bigtable when:

- They handle **more than 1TB of semi-structured or structured data**.
- Data requires **high throughput and rapid changes**.
- They work with **NoSQL data**, where strong relational semantics are unnecessary.
- Data has a **time-series format or natural semantic ordering**.
- They process **big data asynchronously or in real-time**.
- They run **machine learning algorithms** on the data.

Integration & Data Processing:

- Bigtable interacts with **Google Cloud services** and **third-party clients** via APIs.
- Data can be read and written through:
    - **Managed VMs**, **HBase REST Server**, or **Java Server using the HBase client**.
    - **Streaming frameworks** like Dataflow Streaming, Spark Streaming, and Storm.
    - **Batch processing** using Hadoop MapReduce, Dataflow, or Spark.
- Processed or summarized data can be written back to **Bigtable** or **downstream databases**.

Comparison of Google Cloud Storage Services

Storage Service	**Best For**	**Capacity**	**Key Features**	**Limitations**
Cloud Storage	Storing **immutable blobs** (e.g., large images, movies)	Petabytes (max **5TB per object**)	Highly scalable, supports object storage	No SQL queries, best for unstructured data
Cloud SQL	**OLTP workloads** with full SQL support (e.g., web frameworks, user credentials, customer orders)	Up to **64TB**, depending on machine type	Supports relational databases (MySQL, PostgreSQL, SQL Server)	Limited horizontal scalability (only read replicas)
Spanner	**Horizontally scalable** SQL database for large-scale applications	Petabytes	Distributed SQL, strong consistency, global transactions	More complex setup and higher cost compared to Cloud SQL

Firestore	**Massive scaling**, real-time queries, and **offline support**	Terabytes (max **1MB per entity**)	NoSQL document database, ideal for mobile and web apps	Not optimized for complex analytical queries
Bigtable	**Large-scale structured data** with high read/write throughput (e.g., AdTech, financial, IoT)	Petabytes (max **10MB per cell, 100MB per row**)	High-speed, low-latency NoSQL database	No SQL queries, no multi-row transactions
BigQuery	**Big data analysis and interactive queries**	Not purely a storage service	Serverless, supports SQL queries, optimized for analytics	Not designed for transactional or real-time applications

Key Learnings

Google Cloud Storage (GCS) is a highly durable, scalable, and globally available **object storage service** designed to manage data as objects with metadata and unique identifiers, making it ideal for storing unstructured data like videos, images, and backups. It differs from traditional file or block storage by using object-based architecture, which is perfect for web apps and large media handling. GCS allows users to store and retrieve any amount of data without worrying about provisioning storage, and supports data placement in single, dual, or multi-regions depending on latency and availability needs. Objects in GCS are immutable—if changes are required, new versions are created. You can either overwrite old objects by default or enable **versioning** to maintain historical records and recover older versions when needed.

Security in GCS is robust, using **IAM (Identity and Access Management)** for project-wide permission control and **ACLs (Access Control Lists)** for finer access granularity. All data is encrypted by default, both in transit using HTTPS/TLS and at rest on Google's infrastructure. To optimize storage costs, GCS supports **lifecycle management**, which allows you to automatically delete, archive, or retain specific versions of data based on custom rules—like deleting files older than a year or keeping only the latest three versions. GCS also offers four **storage classes** based on data access patterns: **Standard** (frequently accessed), **Nearline** (monthly access), **Coldline** (quarterly access), and **Archive** (annual or rare access), with an **Autoclass feature** to automatically shift data between classes and reduce costs.

Data transfer to GCS is flexible and supports both small and large-scale migrations. Small files can be uploaded via the **Cloud Console** or **gcloud CLI**, while enterprise-scale transfers use **Storage Transfer Service** or **Transfer Appliance**, a physical device for moving up to 1 petabyte of data. GCS also integrates seamlessly with other Google services like **BigQuery, Cloud SQL, Firestore, and Compute Engine**

.

Cloud Containers

Infrastructure as a Service (IaaS)

- IaaS allows developers to share compute resources using **virtual machines (VMs)** to virtualize hardware.
- Each developer can:
    - Deploy their own **Operating System (OS)**.
    - Access hardware components like **RAM, file systems, networking interfaces**, etc. and build applications in a **self-contained environment**.

What Are Containers?

- Containers combine:
    - **Independent scalability** of workloads from Platform as a Service (PaaS).
    - **Abstraction layer** of OS and hardware from IaaS.
- They allow for a **configurable system** where you can:
    - Install runtimes, web servers, databases, or middleware.
    - Configure system resources like **disk space, disk I/O, networking**.

Limitations of Virtual Machines

- In VMs, the smallest unit of compute is an app along with its **guest OS**.
- The **guest OS** can be **large (in gigabytes)** and can take **minutes to boot**.
- To scale applications, you must **copy the entire VM and boot the guest OS** each time, which is **slow and costly**.

Advantages of Containers

- A container is an **invisible box** around code and its dependencies.
- It has **limited access** to its own partition of the file system and hardware.
- Requires only a **few systems calls to create** and **starts as quickly as a process**.
- Only needs:
    - An **OS kernel** that supports containers.
    - A **container runtime**.
- The **OS is virtualized**, providing:

    - **Scalability similar to PaaS**.
    - **Flexibility similar to IaaS**.

Portability and Flexibility

- Code becomes **ultra-portable**.
- You can move from:
    - **Development** to **staging** to **production**.
    - From **local machines** to the **cloud**.
- No need to **change or rebuild** the application during this transition.

Example: Scaling a Web Server

- Containers allow you to **scale a web server in seconds**.
- You can deploy **dozens or hundreds** of container instances depending on workload, on a **single host**.

Building Applications with Multiple Containers

- Applications are often structured using **multiple containers**, each handling a specific function (microservices).
- These containers:
    - Communicate over **network connections**.
    - Are **modular**, **easily deployable**, and **independently scalable** across a group of hosts.

Dynamic Scaling and Fault Tolerance

- Hosts can:
    - **Scale up or down**.
    - **Start and stop containers** based on:
        - Changes in **application demand**.
        - **Host failures**.

Kubernetes and Its Role in Managing Containerized Applications

Introduction

- Kubernetes helps manage and scale containerized applications.
- Google Kubernetes Engine (GKE) can be used to bootstrap Kubernetes, saving time and effort when scaling applications and workloads.

What is Kubernetes?

- **Kubernetes** is an **open-source platform** for managing containerized workloads and services.
- It provides the ability to:
    - Orchestrate many containers across many hosts.
    - Scale containers as microservices.
    - Perform easy deployments, rollouts, and rollbacks.

Architecture Overview

- Kubernetes operates as a set of **APIs** used to deploy containers on **nodes** within a **cluster**.
- The system includes:
    - A **control plane** running primary components.
    - A set of **nodes** that run containers.
- A **node** represents a **computing instance**, like a machine (different from a Google Cloud node which is a VM in Compute Engine).

Pods: The Smallest Unit in Kubernetes

- A **Pod** is a wrapper around one or more containers.
- It represents the **smallest deployable unit** in Kubernetes.
- Each Pod:
    - Represents a **running process** in the cluster.
    - Can be a **component** or an **entire app**.
- Generally, a Pod contains **one container**, but can include multiple containers with **hard dependencies**, sharing **networking and storage resources**.
- Pods offer:
    - A **unique network IP**.
    - A set of **ports**.
    - Configurable **runtime options**.

Deployments and Replication

- To run a container inside a Pod: `kubectl run` command starts a **Deployment** with a container inside a Pod.
- A **Deployment**:
    - Represents a **group of Pod replicas**.
    - Ensures Pods continue running even if their nodes fail.
    - Can represent a **component** or **entire app**.
- To list running Pods: `kubectl get pods`

Services and Networking

- Kubernetes creates a **Service** with a **fixed IP address** for Pods.

- A **controller** can attach an **external load balancer** with a **public IP** to a Service for outside access.
- In GKE, this becomes a **network load balancer**.
- Any client accessing this IP will be routed to a Pod behind the Service.
- A **Service**:
    - Is an **abstraction** that defines a **logical set of Pods**.
    - Defines the **policy** for accessing those Pods.
- Since Pods have **dynamic IPs**, the Service provides a **stable endpoint**.
- Example: Two Pod sets – `frontend` and `backend`
    - Each is behind its own Service.
    - Frontend Pods access the backend through the **Service**, without needing to know Pod-level changes.

Scaling Deployments

- Use the command: `kubectl scale` to scale a Deployment.
- Example: Create **three Pods**, placed behind the Service, sharing a **single fixed IP address**.
- Kubernetes also supports **autoscaling** based on parameters like **CPU utilization thresholds**.

Declarative Configuration

- While **imperative commands** like `expose` and `scale` are useful for learning and testing,
- The **real power** of Kubernetes comes from using it **declaratively**.
- Instead of issuing commands, you define a **configuration file** to describe the **desired state**.
- Kubernetes automatically figures out **how to achieve that state**.
- Use a **Deployment configuration file** to define behavior.
- Check replica status with: `kubectl get deployments` or `kubectl describe deployments`
- To scale from three to five replicas:
    - Update the config file.
    - Apply the change with: `kubectl apply`
- Use:
  `kubectl get services` to retrieve the external IP of the Service.

Updating Applications Safely

- When updating to a new app version:
    - Update the container to deliver new code.
    - Avoid rolling out all changes at once to prevent risk.
- Use:

    - `kubectl rollout`
    - Or update the Deployment config file and apply it with: `kubectl apply`
- New Pods are created using the **update strategy**.
    - Example: New version Pods are created **one-by-one**.
    - Wait for each new Pod to become available **before terminating** an old one.

Google Kubernetes Engine (GKE)

What is GKE?

- **GKE (Google Kubernetes Engine)** is a **Google-hosted managed Kubernetes service** on the cloud.
- It simplifies Kubernetes deployment by **managing infrastructure and control plane** components.

GKE Cluster Composition

- GKE environment is composed of **multiple machines**, specifically **Compute Engine instances**.
- These instances are grouped together to form a **Kubernetes cluster**.

GKE vs. Kubernetes

- Though you can create Kubernetes clusters using GKE, **GKE simplifies the process** from the user's point of view.
- GKE **manages all control plane components**:
    - Still provides an **IP address** to send Kubernetes API requests.
    - But **Google handles the provisioning and management** of the control plane infrastructure.
    - It removes the need for a **separate control plane setup**.

GKE Modes

There are **two modes** in GKE for node configuration and management:

Mode	**Managed By**	**Key Characteristics**
Autopilot	GKE (recommended)	GKE manages all infrastructure, including: - Node config - Autoscaling - Auto-upgrades - Baseline security & networking

Standard	User	You manage everything: - Node configuration - Management - Optimization

Autopilot Mode Details

- **Optimized for production**
- Provides a **strong security posture** Enhances **operational efficiency**
- Recommended unless specific infrastructure control is needed (as offered in Standard mode)

Creating GKE Clusters

- You can create a cluster via:
  - **Google Cloud Console**
  - **`gcloud` command-line tool** from Cloud SDK,

  Example: `gcloud container clusters create k1`

Cluster Customization

- GKE clusters support customization of:
  - **Machine types**
  - **Number of nodes**
  - **Network settings**

Kubernetes in GKE

- Kubernetes is the interface to **interact with and manage the cluster**.
- Through Kubernetes, you can:
  - Deploy and manage applications
  - Perform administrative tasks
  - Set policies
  - Monitor the health of workloads

GKE Advanced Features from Google Cloud

GKE provides several advanced management features:

Feature	**Description**

Load Balancing	Google Cloud's built-in load balancing for Compute Engine instances
Node Pools	Designate subsets of nodes within a cluster for added flexibility
Automatic Node Scaling	Adjusts the number of node instances automatically based on demand
Automatic Node Upgrades	Keeps node software up to date automatically
Node Auto-Repair	Maintains node health and availability
Logging and Monitoring	Powered by **Google Cloud Observability** for full visibility into the cluster

Key Learnings

Infrastructure as a Service (IaaS) provides developers with virtual machines to share compute resources, allowing them to run custom operating systems and access hardware-level components like RAM, file systems, and networking interfaces. However, virtual machines come with limitations such as large size, slow startup, and high costs due to guest OS overhead. This is where **containers** come in. Containers are lightweight, start almost instantly, and wrap application code with its dependencies in an isolated environment. They combine the scalability of PaaS and the flexibility of IaaS, making applications ultra-portable and consistent across environments—whether moving from local development to production or across clouds. Containers are ideal for microservice-based architecture where each service can run in its own container and communicate with others over the network.

Kubernetes is the open-source platform that helps manage and orchestrate these containerized applications efficiently. It operates with a control plane and a set of worker nodes and uses **Pods** (the smallest deployable unit) to run containers. Kubernetes provides automated deployment, scaling, load balancing, and self-healing features. Through commands like `kubectl run`, `kubectl get pods`, and `kubectl scale`, developers can deploy and manage applications

with ease. **Services** in Kubernetes abstract away dynamic IPs of Pods and offer stable networking endpoints, essential for maintaining communication between frontend and backend components.

For managing Kubernetes in the cloud, **Google Kubernetes Engine (GKE)** is a fully managed service that simplifies cluster creation and scaling. GKE supports two modes: **Autopilot**, where Google handles node management, scaling, security, and updates (ideal for production), and **Standard**, where users control the infrastructure. GKE clusters run on Google Compute Engine instances and come with features like automatic node upgrades, auto-scaling, node pools, logging, and monitoring—ensuring high availability, security, and observability. GKE can be managed via the Cloud Console or the `gcloud` CLI, offering flexibility in cluster setup, machine types, and network configurations. Declarative configuration in Kubernetes lets you define your desired state in YAML files, which Kubernetes ensures is maintained—making updates, rollouts, and scaling safer and more reliable.

Applications in the cloud

What is Cloud Run?

- **Cloud Run** is a **managed compute platform** that runs **stateless containers** triggered via:
    - **Web requests**
    - **Pub/Sub events**
- It is **serverless**, meaning:
    - No infrastructure management required
    - Developers can focus solely on application development
- Built on **Knative**:
    - An open API and runtime environment based on **Kubernetes**
- Can run:
    - Fully managed on **Google Cloud**
    - On **Google Kubernetes Engine (GKE)**
    - Anywhere **Knative** is supported

Key Features

Feature	**Description**
Serverless	Handles all infrastructure tasks
Fast scaling	Instantly scales from zero based on demand
Pay-per-use	Charged only for actual resource usage, calculated to the nearest **100ms**

HTTPS URL	Each deployment receives a unique **HTTPS endpoint**
Request handling	Dynamically starts/stops containers to handle incoming traffic
Encryption	HTTPS and encryption managed by Cloud Run

Developer Workflow (3 Steps)

1. **Write** your app using your preferred language
    - Ensure it starts a server to handle web requests
2. **Build and package** the app into a **container image**
3. **Push** the image to **Artifact Registry**
    - Cloud Run then **deploys** it

Deployment Options

- **Container-based workflow**:
    - Provides **transparency** and **flexibility**
- **Source-based workflow**:
    - You deploy **source code** instead of a container image
    - Cloud Run builds and packages it into a container using **Buildpacks** (open-source project)

Pricing Model

Cost Element	**Details**
Resource Usage	Charged only while containers handle web requests
Granularity	Charges calculated in **100ms** intervals
Idle Time	**No charge** when containers are not handling requests
Requests	Small fee per **1 million requests**
Resource-Based Pricing	Cost increases with more **vCPU and memory** used

Supported Applications

- Cloud Run can run **any Linux 64-bit compiled binary**
- Popular supported languages include:
    - **Java**, **Python**, **Node.js**, **PHP**, **Go**, **C++**
- Also supports less common languages like:
    - **Cobol**, **Haskell**, **Perl**
- Requirement: The application must be able to **handle web requests**

Cloud Run Functions Summary

Event-Driven Applications: The Need

- Many modern applications include event-driven components.
- **Example use case:** A user uploads an image.
    - **Tasks triggered by this event:**
        - Convert the image to a standard format
        - Generate multiple thumbnail sizes
        - Store the processed files in a repository
- **If integrated directly into the application:**
    - Requires always-on compute resources
    - Wasteful if the event occurs only occasionally

What Are Cloud Run Functions?

Feature	**Description**
Lightweight compute	Event-based and asynchronous
Single-purpose functions	Each function handles a specific business logic task
No infrastructure to manage	No need for server or runtime environment management
Automatic execution	Triggered by events (e.g., image upload)

Used in workflows	Functions can be composed to build complex workflows from small logic units
Extend cloud services	Functions can interact with and extend other services on Google Cloud

Billing and Cost Model

Parameter	**Description**
Granularity	Charged to the nearest 100 milliseconds
Runtime-based	Charged only while code is running

Supported Programming Languages

Cloud Run functions support writing source code in the following languages:

- Node.js
- Python
- Go
- Java
- .NET Core
- Ruby
- PHP

Trigger Mechanisms

Cloud Run functions can be triggered via:

Trigger Type	**Execution Mode**
Cloud Storage events	Asynchronous
Pub/Sub events	Asynchronous

HTTP invocation	Synchronous

Basic Linux Commands

Below you will find a reference list of a few very basic Linux commands which may be included in the instructions or code blocks for this lab.

Command -->	Action	.	Command -->	Action
mkdir (*make directory*)	**create a new folder**	**.**	**cd (*change directory*)**	**change location to another folder**
ls (*list*)	**list files and folders in the directory**	**.**	**cat (*concatenate*)**	**read contents of a file without using an editor**
apt-get update	**update package manager library**	**.**	**ping**	**signal to test reachability of a host**
mv (*move*)	**moves a file**	**.**	**cp (*copy*)**	**makes a file copy**
pwd (*present working directory*)	**returns your current location**	**.**	**sudo (*super user do*)**	**gives higher administration privileges**

Key Learnings

Cloud Run is Google Cloud's fully managed **serverless compute platform** that runs **stateless containers**, automatically triggered by web requests or Pub/Sub events. It eliminates the need for infrastructure management, letting developers focus entirely on building applications. Cloud Run is built on **Knative**, an open-source Kubernetes-based platform, which means you can run your services fully managed on Google Cloud, on GKE, or anywhere else Knative is supported. It scales applications from **zero to high traffic** instantly and only charges you **for the resources used** during request handling—measured in 100ms intervals. Every deployed service gets a **secure HTTPS URL** and handles incoming requests by dynamically starting or stopping containers.

The **developer workflow** is simple: write your app in any language (Java, Python, Node.js, Go, etc.), make sure it listens to HTTP requests, containerize it, and deploy it either by pushing the image to Artifact Registry or directly from source using **Buildpacks**. Cloud Run supports **any 64-bit Linux binary**, meaning you can even run apps in Cobol or Haskell as long as they respond to HTTP requests.

A powerful feature of Cloud Run is its **event-driven functions**. These are small, single-purpose units of code (like image resizing, sending notifications, or transforming data) that automatically trigger on events such as file uploads or Pub/Sub messages. You don't need to manage infrastructure, and you pay only while the function runs. This makes Cloud Run ideal for building **lightweight, scalable, and cost-efficient microservices** or integrating **asynchronous workflows** in modern applications.

Cloud Run's pricing is **resource-based**—more vCPU and memory usage will cost more—but it saves cost during **idle times** as you're not charged when containers are inactive. It also allows HTTP-based synchronous triggers or asynchronous execution through **Cloud Storage and Pub/Sub events**. Together, Cloud Run and its functions offer a **flexible, modern, and scalable approach to cloud-native app development**, especially when combined with containers and event-driven architectures.

Google Cloud – Prompt Engineer Guide

1. Key Concepts

a. Generative AI (Gen AI)

- A subset of AI that creates content (text, images, etc.) using generative models in response to prompts.
- Gained popularity post-2021; AI itself dates back to the 1950s.
- Works like a conversation tool that generates data with patterns similar to its training data.
- Widely used across various industries.

b. Prompt

- An instruction, question, or cue given to an AI to trigger a response.
- Central to how Gen AI works.

2. Large Language Models (LLMs)

Definition

- Sophisticated models trained on massive datasets (text, images, code).
- "Large" refers to training data size and the number of parameters (can be in billions/trillions).
- General-purpose but can be fine-tuned for specific tasks.

Training Process

- **Pre-training**: Learn patterns and structures from large datasets.
- **Fine-tuning**: Refine model behavior using smaller, task-specific datasets.

Functionality

- Predicts likely next words or sentences—similar to autocomplete.
- Can generate **hallucinations** (nonsensical or incorrect outputs).

Causes of Hallucination

- Insufficient or noisy data.
- Lack of context or constraints in the prompt.

3. Prompt Engineering

Definition

- The method of crafting prompts to get accurate responses from LLMs.

Types of Prompts

Type	Description
Zero-shot	No example or context provided.
One-shot	One example included for clarity.
Few-shot	Two or more examples provided to guide the model.
Role-based	Assigns a specific role or persona to the model for focused responses.

Prompt Components

- **Preamble**: Introductory context and instructions.
- **Input**: The specific request or task for the model.

4. Practical Scenario: Sasha (Cloud Architect)

- Tasked with designing a prototype Google Cloud VPC network.
- Uses Gemini (Gen AI tool in Google Cloud Console) to aid the process.
- Refines prompts by:
    - Adding technical details (e.g., dual stack subnet).
    - Using role prompts for more targeted output.
    - Structuring questions clearly and contextually.

5. Prompt Engineering Best Practices

Best Practice	**Explanation**
Write detailed & explicit instructions	Avoid vague prompts. Be clear and specific.
Define boundaries	Instruct what *to do* rather than what *not to do*.
Provide fallback outputs	Give standard responses for uncertain outputs.
Use personas	Add role context to help the model focus on the domain.
Keep sentences concise	Split complex ideas into smaller, simpler sentences.

6. Final Application Example

- Sasha updates her prompt with clear requirements for a centrally managed VPC network that connects across regions and minimizes firewall policies.
- Gemini recommends a **hub-and-spoke architecture**, effectively matching her needs.

Key Learning

This module introduces the foundational concepts of **Generative AI (Gen AI)** and how it interacts with users through **prompts**, especially using **Large Language Models (LLMs)**. Gen AI refers to a subset of AI that can create original content like text or images based on patterns it learned during training. At the heart of Gen AI is the **prompt**, which acts like an instruction that triggers a meaningful response from the model. LLMs, which power Gen AI tools, are trained on huge datasets and consist of billions (sometimes trillions) of parameters. They operate on prediction—generating the most likely next word or phrase, much like a sophisticated autocomplete. These models undergo **pre-training** on broad data and **fine-tuning** for specialized tasks. However, they

may generate **hallucinations** (inaccurate or irrelevant outputs), usually due to poor-quality data or unclear prompts.

This is where **Prompt Engineering** becomes important. It's the art of designing effective prompts to get accurate responses from LLMs. Different types of prompting exist: **zero-shot** (no example), **one-shot**, **few-shot**, and **role-based** prompts where the model is asked to "act as" a specific expert. A well-structured prompt usually has two parts: a **preamble** (context-setting) and the **input** (the actual request). For example, in a practical case, **Sasha**, a Cloud Architect, uses Google's **Gemini** AI tool to design a VPC network. By adding more context and refining her prompt with technical details like "dual stack subnet" or using role-based prompting, she receives accurate recommendations, such as a **hub-and-spoke architecture**.

To build effective prompts, you should follow best practices like writing clear, detailed instructions, defining what *to do* (instead of what *not* to do), using fallback responses for uncertain situations, assigning specific personas to guide model behavior, and breaking down complex questions into short, simple sentences. Altogether, this module teaches that crafting the right prompt is just as critical as knowing the technology—it's what turns an average AI response into a precise and useful solution.