**Authentication is done by cloud identity  
Authorization is done by cloud IAM  
Auditing is done by cloud operations(audit logging & reports api)  
BGP:-border gateway protocol**enables the internet to exchange routing information between autonomous systems (AS). As networks interact with each other, they need a way to communicate. This is accomplished through peering. BGP makes peering possible. **ICMP:- internet control message protocol**it is used for reporting errors and performing network diagnostics. In error reporting process ICMP sends messages from the receiver to sender when data not comes through as it should. **VPC networks**

A Virtual Private Cloud (VPC) network is a virtual version of a physical network that is implemented inside of Google's production network by using [Andromeda](https://www.usenix.org/system/files/conference/nsdi18/nsdi18-dalton.pdf).

A VPC network does the following:

* Provides connectivity for your [Compute Engine virtual machine (VM) instances](https://cloud.google.com/vpc/docs/vpc#vm-instances).
* Offers native internal passthrough Network Load Balancers and proxy systems for internal Application Load Balancers.
* Connects to on-premises networks by using Cloud VPN tunnels and VLAN attachments for Cloud Interconnect.
* Distributes traffic from Google Cloud external load balancers to backends.

# Subnets

[Virtual Private Cloud (VPC) networks](https://cloud.google.com/vpc/docs/vpc) are global resources. Each VPC network consists of one or more IP address ranges called subnets. Subnets are regional resources, and have IP address ranges associated with them.

## **Types of subnets**

VPC networks support the following subnet types:

* IPv4 only (single-stack) subnets, with only IPv4 subnet ranges
* IPv4 and IPv6 (dual-stack) subnets, with both IPv4 and IPv6 subnet ranges

### VPC firewall rules . VPC firewall rules let you allow or deny traffic to or from [virtual machine (VM) instances](https://cloud.google.com/vpc/docs/vpc#vm-instances) in a VPC network based on port number, tag, or protocol. ingress egress **Routes**

Routes tell VM instances and the VPC network how to send traffic from an instance to a destination, either inside the network or outside of Google Cloud. Each VPC network comes with some [system-generated routes](https://cloud.google.com/vpc/docs/vpc#system-generated-routes) to route traffic among its subnets and send traffic from [eligible instances](https://cloud.google.com/vpc/docs/vpc#internet_access_reqs) to the internet.

You can create custom static routes to direct some packets to specific destinations.

### **Subnet routes**

Subnet routes define paths to resources like VMs and internal load balancers in a VPC network.

Each subnet has at least one subnet route whose destination matches the primary IPv4 range of the subnet. If the subnet has secondary IP ranges, there's a corresponding subnet route for each of its secondary IP address ranges. If the subnet has an IPv6 range, there's a corresponding subnet route for the IPv6 address range. For more information about subnet IP ranges, see [Subnets](https://cloud.google.com/vpc/docs/subnets).

### **Forwarding rules**

While routes govern traffic leaving an instance, forwarding rules direct traffic to a Google Cloud resource in a VPC network based on IP address, protocol, and port.

Some forwarding rules direct traffic from outside of Google Cloud to a destination in the network; others direct traffic from inside the network. Destinations for forwarding rules are target instances, load balancer targets (target proxies, target pools, and backend services), and Cloud VPN gateways.

# IP addresses

Resources such as VM instances and load balancers have IP addresses in Google Cloud. These IP addresses let Google Cloud resources communicate with other resources in Google Cloud, in on-premises networks, or on the public internet. This page describes the IP address categorization used by Google Cloud.

Google Cloud uses the following labels to describe different IP address types. For example, an internal IP address is not publicly routed. An external IP address is a publicly routed IP address. You can assign an external IP address to the network interface of a Google Cloud VM.

**External IP address**

External IP addresses are publicly advertised, meaning they are reachable by any host on the internet. External IP addresses must be [publicly routable IP addresses](https://wikipedia.org/wiki/IP_address#Public_address). Resources with external IP addresses can communicate with the public internet  
 **Internal IP address**

Internal IP addresses cannot be reached from the internet, and are not publicly routable.

Internal IPv4 addresses can be private IPv4 addresses or they can be privately used public IPv4 addresses. For a list of valid internal IPv4 addresses, see [Valid IPv4 ranges](https://cloud.google.com/vpc/docs/subnets#valid-ranges).

Internal IPv6 addresses are unique within Google Cloud. For more information,  
 **Private IP address**

Private IP addresses are addresses that [cannot be routed on the internet](https://wikipedia.org/wiki/IP_address#Private_addresses)

**Public IP address**

Public IP addresses are internet routable. In Google Cloud, external IPv4 and IPv6 addresses are always public IP addresses

## **Regional and global IP addresses**

When you list or describe IP addresses in your project, Google Cloud labels addresses as global or regional, which indicates how a particular address is being used. When you associate an address with a regional resource, such as a VM, Google Cloud labels the address as regional. Regions are Google Cloud regions, such as us-east4 or europe-west2.

## **VPC sharing and peering**

### **Shared VPC**

You can share a VPC network from one project (called a host project) to other projects in your Google Cloud organization. You can grant access to entire Shared VPC networks or select subnets therein by using [specific IAM permissions](https://cloud.google.com/vpc/docs/shared-vpc#iam_in_shared_vpc). This lets you provide centralized control over a common network while maintaining organizational flexibility. Shared VPC is especially useful in large organizations.  
VPC Network Peering lets you build [software as a service (SaaS)](https://wikipedia.org/wiki/Software_as_a_service) ecosystems in Google Cloud, making services available privately across different VPC networks, whether the networks are in the same project, different projects, or projects in different organizations.

With VPC Network Peering, all communication happens by using internal IP addresses. Subject to firewall rules, VM instances in each peered network can communicate with one another without using external IP addresses.

## **Hybrid cloud**

### **Cloud VPN**

Cloud VPN is one of the fastest and easiest ways to connect to your on-premises network VPN connection traffic traveling between the two networks is encrypted by one VPN Gateway  
  
VPN supports High availability and provides a maximum bandwidth of 1.5 to 3 gbps per tunnel it supports both static and bgp based routing   
**classic VPN** and **havpn classic**  
VPN gateways have a single interface a single external IP address and support tunnels that use static routing you can also configure Dynamic routing for classic VPN but only for tunnels that connect to third-party VPN Gateway software running on Google Cloud VM instances  
**Hivpn** provides an SLA of 99.99 service availability hence it is a recommended choice for new deployments when you create a ha VPN

Cloud VPN lets you connect your VPC network to your physical, on-premises network or another cloud provider by using a secure [virtual private network](https://wikipedia.org/wiki/Virtual_private_network).

### **Cloud Interconnect**

Cloud Interconnect lets you connect your VPC network to your on-premises network by using a high speed physical connection.  
**dedicated and partner interconnect**   
**dedicated interconnect** provides a physical connection between your on-premises network and Google's Network dedicated interconnect enables you to transfer large amount of data between your networks which can be more cost effective than purchasing additional bandwidth over the public internet your network  
  
**partner interconnect** connection is useful if your data center is in a physical location that can't reach a dedicated interconnect co-location facility or your data needs don't warrant a

## **Cloud Load Balancing**

Google Cloud offers several global and regional load balancing configurations to distribute traffic and workloads across many backend types.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Load balancer** | **Deployment mode** | **Traffic type** | **Network service tier** | **Load-balancing scheme**[**†**](https://cloud.google.com/load-balancing/docs/load-balancing-overview#load-balancing-scheme) |
| [Application Load Balancers](https://cloud.google.com/load-balancing/docs/application-load-balancer) | Global external | HTTP or HTTPS | Premium | EXTERNAL\_MANAGED |
| Regional external | HTTP or HTTPS | Standard | EXTERNAL\_MANAGED |
| Classic | HTTP or HTTPS | Global in Premium Tier  Regional in Standard Tier | EXTERNAL |
| Regional internal | HTTP or HTTPS | Premium | INTERNAL\_MANAGED |
| Cross-region internal | HTTP or HTTPS | Premium | INTERNAL\_MANAGED |
| [Proxy Network Load Balancers](https://cloud.google.com/load-balancing/docs/proxy-network-load-balancer) | Global external | TCP with optional SSL offload | Premium Tier | EXTERNAL\_MANAGED |
| Classic | TCP with optional SSL offload | Global in Premium Tier  Regional in Standard Tier | EXTERNAL |
| Regional external | TCP | Standard only | EXTERNAL\_MANAGED |
| Regional internal  Always regional | TCP without SSL offload | Premium only | INTERNAL\_MANAGED |
| Cross-region internal  multi-regional | TCP without SSL offload | Premium only | INTERNAL\_MANAGED |
| [Passthrough Network Load Balancers](https://cloud.google.com/load-balancing/docs/passthrough-network-load-balancer) | External  Always regional | TCP, UDP, ESP, GRE, ICMP, and ICMPv6 | Premium or Standard | EXTERNAL |
| Internal  Always regional | TCP, UDP, ICMP, ICMPv6, SCTP, ESP, AH, and GRE | Premium only | INTERNAL |

## Cloud Load Balancing A load balancer distributes user traffic across multiple instances of your applications. By spreading the load, load balancing reduces the risk that your applications experience performance issues. **Types of load balancers**

Cloud Load Balancing offers two types of load balancers: Application Load Balancers and Network Load Balancers. You'd choose an Application Load Balancer when you need a Layer 7 load balancer for your applications with HTTP(S) traffic. You'd choose a Network Load Balancer when you need a Layer 4 load balancer that supports TLS offloading (with a proxy load balancer) or you need support for IP protocols such as UDP, ESP, ICMP, and so on (with a passthrough load balancer).

**CIDR**  
  
Classless Inter-Domain Routing (CIDR) is a method of IP address allocation and IP routing that allows for more efficient use of IP addresses. CIDR is based on the idea that IP addresses can be allocated and routed based on their network prefix rather than their class, which was the traditional way of IP address allocation.

CIDR addresses are represented using a slash notation, which specifies the number of bits in the network prefix. For example, an IP address of 192.168.1.0 with a prefix length of 24 would be represented as 192.168.1.0/24. This notation indicates that the first 24 bits of the IP address are the network prefix and the remaining 8 bits are the host identifier.

**machine images**: we can store copy of VM using machine type that can used to restore the images whenever the data got deleted or removed( we can recreate the VMs by using images).

**images:** best for infrastruture re-use VM.. it contains the OS and necessary to boot a VM.

**snapshot:** best for data backups( persistent desk like it is used to store all the files and configurations).  
**Snapshots** are backups of persistent disks.  
 They’re commonly used to recover, transfer, or make data accessible to other resources in your project.

**instance group:**  to create number of VMs with same configurations by using instance template

**instance template:** to create a instance group we r using instance template.

**Health checking** mechanisms determine whether VM instances respond properly to traffic.

**Virtual Private Cloud (VPC)**

VPC is a Logical representation of an on-prem network. This is a global construct in GCP

**VPC modes**

There are two modes in GCP. Auto mode and custom mode

**VPC subnets**

In GCP these are regional and assigned to an IP address range

**CIDRload address**

A unique address used to identity host on network. Made up of network and host portions

**Subnet mask**

This segments and IP address into network and host portions. It determines how must host are available on the network. This can be manipulated to form CIDR blocks

**IPV4**

This is a 32 bit, 4 octet address. Written in binary or dotted decimal format. E.g. [192.168.10.20](http://192.168.10.20/) or

**IPV6**

This is a 128 bit, hexadecimal address. 2001:DB8:7654:3210:FEDC:BA98:764:3203

**Private IP (RFC1918)**

A special range that can be used internally by anyone. These are non internet routable

**Public IP**

IP address that is routable on the internet

**DHCP**

Dynamic Host Control protocol. A method to automatically assign an IP address to a client

S**tatic IP**

An IP that does not change after being assigned

**Bring Your Own IP (BYOIP)**

Use external IP addresses that you own in Google Cloud

**Alias IP**

Additional addresses that can be assigned to your VM, these can be taken from the primary or secondary address range

**Secondary IP**

Secondary range of IP address that can be assigned to your VM

R**estricted.googleapi** [s.com](http://s.com/) **IP**

Access external GCP APIs via google private network. [199.36.153.4/30](http://199.36.153.4/30). Used when VPC service controls are enabled and you need to access only VPC service control supported APIs

**Private.googleapis.c om IP**

Access external GCP APIs via google private network. [199.36.153.8/30](http://199.36.153.8/30)

**Network Time Protocol (NTP)**

Is used to synchronize systems timer across a network. This is used on both internal and external networks.

**What is the OSI Model**

A 7 layer conceptual model that provides interoperability of the TCP stack

Application Layer (Layer 7)

User interface and application. Protocols examples HTTP, HTML

Presentation Layer (Layer 6)

Formats data to be presented. Protocols examples JPEG, ASCII, GIF

Session Layer (Layer 5)

Creates, tracks, ends the sessions between different systems

Transport layer (Layer 4)

Handles message delivery using connection and connectionless protocols. Protocol examples TCP, UDP

Network layer (Layer 3)

Focuses on subnets, route path selection. Protocols examples IP, ICMP,. Router work here

Data layer (Layer 2)

Focuses of transferring data frames over physical layer. Protocol, ARP, PPP, VLANS. Switches work here

Physical layer (Layer 1)

Transmission of raw bits over physical mediums. Examples network cables, wireless

**What is the Internet Model**

A 4 layer model conceptual model of the TCP/IP stack

**Application Layer**

User interface and application.

**Transport layer**

Responsible for end to end data handling of data streams

**Internet layer**

Link layer

Responsible for routing packets through networks

From a device it interacts with physical network

G**CP Services operating at different OSI layers**

Layer 7

HTTPS Load balances, Cloud Armor

Layer 4

Load balancers

Layer 3

Interconnect

Layer 2

Interconnect VLANs

**Transmission Control Protocol (TCP)**

This is a connection oriented protocol that handles reliability, flow and congestion control of packets. It establishes a connection before sending a packet

**Transmission Control Block (TCB)**

Contains all the information about the connection and implements the sliding window

**Sliding window**

Determines the amount of bytes that one system can send to the other. Once the agreed bytes are received and processed, the sender sends another set of bytes to the receiver until all data is sent

**Three-way handshake**

This is the sequence to form a TCP connection. It involve the SYN, SYN/ACK, ACK flag exchange between client/server

**Flag**

These indicate the state of the connection

**SYN**

The SYN or synchronize flag is sent to start the TCP connection process

**ACK**

The ACK or the acknowledgement flag. This confirms that data was received

**FIN**

A flag sent to request termination of connection

**User Datagram Protocol (UDP)**

This is a best effort delivery protocol

**Quick UDP Internet Connections(QUIC)**

A Google made transport layer protocol. This is built on top of UDP

**Transport Layer Security (TLS)**

A protocol that provides cryptography by using certificates

**Data messages types**

These are frames, packets, datagrams. They may exist at different layers of the OSI model

**Maximum transfer unit (MTU)**

The size of the largest unit of data that can be transmitted over the network

**Time to Live (TTL)**

This indicates the life of the packet usually has a max of 255 hops. This ensures packets don't exist forever in a network

**Unicast message** These are sent on a 1 to 1 basis on a network

**Multicast message** These are sent to subscribed groups on a network

**Broadcast messages** are sent to every device on a network.

**Domain Name Service (DNS)** Resolves names to IP addresses

**Cloud DNS** Google Cloud DNS offering

**Internal DNS** used internally within a private network

**DNS Security Extensions (DNSSEC)**  
Uses digital signature to secure DNS information

**Hybrid DNS**

Used internally within a private network

DNS configured between cloud and on-prem or external networks

Address resolution Protocol (ARP)

Protocol used to resolve IP address to a MAC/link layer address. Maintained in the ARP table.

Reverse ARP (RARP)

Media Access Control address(MAC)

Network Address Translation (NAT)

This is the inverse of ARP. Used to resolve MAC to IP addresses.

Unique hexadecimal identifier assigned to a network interface controller (NIC) card. Usually a 12 digit hexadecimal number

**Cloud NAT**

Google Cloud managed NAT service Allows private IP ranges to communicate with the internet. Maintains a NAT table of private to public address & port mappings for communications

**Routing**

Selecting a path for traffic to flow within internal networks or between different networks

**Router**

Allows communication between different networks

**Cloud Router**

Google Cloud router that allows you to dynamically exchange routes between your VPC and on-prem using BGP

**Routing table**

A repository of all the routing information within a network

**Routing modes**

These are static or dynamic

**Static routing**

These routes are fixed an don't update. They usually have to be manually adjusted

**Dynamic routing**

These routes update to reflect current state

**Route summarization**

Used to reduce the number of route advertised to neighbours. See example

**next-hop**

The address of the next router in the transit route of a packet

**Software Defined Networking (SDN)**

A software based networking approach that uses application programming interfaces (API) to communicate with underlying infrastructure to control the network traffic.

**Border Gateway Protocol (BGP)**

Is the path vector protocol of the internet. Made up of Autonomous systems (AS) and uses TCP port 179

**Autonomous System (AS)**

Is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators

**Autonomous System Number (ASN)**

The number used to identify an AS. This can be 16 bit or 32 bit

**External BGP (eBGP)**

BGP connection formed between different AS's

**Internal BGP (IBGP)**

Connection formed within the same AS

**Multiple Exit Discriminator (MED)**

This a one of several BGP attributes used to influence path selection. This is non transitive and the lower metric wins

**AS-path-prepend**

This is one of several BGP attributes used to influence path selection. This is a mandatory attribute. The shorter path should be preferred

**Multiprotocol label switching (MPLS)**

This is a switching method that uses labels Instead of IP information to transmit packets across the backbone core at high speed

**Bidirectional Forwarding Detection (BFD**)

This is a protocol that detects failure quickly on links when enabled. In GCP you can use this feature with Cloud router

**optical circuit switching**

Maps optical input to output ports to form a connection

**wave division multiplexing**

WDM technology allows you to combine multiple optical signal onto a single optical fiber

**Clos topology**

A non blocking, multistage switching network, used in data center switching fabrics

**Merchant switch silicon**

Chip made by 3Ps that are sold to any consumers to design a product based on it

**Data Center Fabric**

This is a Data Center design comprised of leaf and spine switches that allows low latency and scalable data center operations.

**Top-of-Rack switches**

These switches are place in the same rack as other equipment to connect all equipment in the rack and to connect to other TOR switches in the DC

**OpenFlow**

OpenFlow is a communications protocol that allows network controllers to directly program the network forwarding plane

**Leaf and Spine**

A two layer full mesh topology. Has leaf switches and spine switches

East West traffic

North South Traffic

**Communication traffic flow between devices in a Data center**

In and out communication traffic flow between Data center and outside networks.

**Colocation**

3P Data Center facilities where multiple tenants can house their data center equipment

**Dedicated Interconnect**

Dedicated connection between Google and your private network. Available from 10 GBit/s to 100 GBit/s. Has high availability configurations and you can use multiple links

**Partner Interconnect**

Highly available connection between Google and your network provisioned through a Service provider. Available from 50 MBit/s to 10 GBit/s. Has high availability configuration and you can use multiple links

**Virtual private network (VPN)**

This offers a secure connection between two locations over a secure IPSEC tunnel

**Carrier Peering**

Google Cloud service that enables you to access Google Workspace and other Google apps via service provider connection

**Direct Peering**

Google Cloud service that enables you to access google Workspace and other Google apps via direct connection to Google edge

**Shared VPC**

GCP service that allow you to provision and connect host projects, and service projects

**VPC Network Peering**

GCP service that allow you to connect between different VPC's in the same or separate project and organizations. 1-to-1 peering that is not transitive. Max peering per VPC is 25 connections

**Traffic Director**

Google Cloud service that offers a fully managed traffic control plane for service mesh

**Firewalls**

Allow, deny & filter traffic based on rules. Affect ingress and egress traffic

**Firewalls rules**

Criteria used to deny, allow access in Google Cloud. e.g. IP, source, tag, service account

**Distributed denial of service (DDoS)**

This is a type of attack that affect availability of service by overloading the systems

**Cloud Armor**

Google Cloud service that provides filtering at OSI layer 7 to 4

**VPC service controls**

Google Cloud service that allows you the ability to create perimeters that protect resources and data

**Cloud Identity-Aware Proxy (IAP)**

Google Cloud service that controls access to your application and restricts it to only authorized users

**Security Command Center**

Google Cloud service that has asset discovery, threat detection, and threat prevention components

**Beyond Corp**

Google Cloud zero trust model

**Cloud IDS**

Google Cloud's Intrusion Detection System. Detect and logs potential threats

**HTTP(S) LB**

Global load balancer for HTTP(S) traffic

SSL proxy

Global load balancer for SSL traffic

TCP proxy

Global for TCP traffic

**Network LB**

Regional LB used to load balance TCP traffic (available internally and externally)

Internal LB

Regional LB used with a VPC

**NEG**

Network Endpoint Group are used to attach a backend pool to a load balancing service in Google Kubernetes Engine

I**ngress**

Allows HTTP(S) traffic connections to a kubernetes cluster

Content Delivery Network (CDN)

Caches content at a distribution endpoint closest to customer.

**Cloud CDN**

Google Cloud's standard web acceleration CDN offering.

Media CDN

Google Cloud's media delivery solution. Can handle high throughput media like streaming.

**Hypertext Transfer Protocol (HTTP)**

Protocol used for transmitting hypermedia documents. This is a standard on the internet, more commonly in it secure form HTTP(S)

**HTTPS**

*Secure version of HTTP enabled by using TLS on the connection*

**ping**

This tool checks the availability of host by using Internet Control Message Protocol

Traceroute or tracert

S**hows the hops between source and destination**

nslookup

Allows you to resolve IP from host name

Domain information groper (dig)

Performs DNS lookup and displays the answers of the query

ipconfig or ifconfig

Show the IP address, subnet and gateway information of a system

**Flow logs**

This GCP service tells you about the traffic flow in your VPC

**Network Intelligence Center**

GCP service that provides you with a few tools to gain visibility into your network

**Cloud Audit Logs**

Google Cloud logs that provide information on activities in your cloud. A few are; Admin Activity. Data Access, system events and Policy denied, audit logs

**Cloud Operations**

Google Cloud tool that allows you to monitor, log and trace application and systems in your environments

**Packet Mirroring**

Packet Mirroring clones the traffic on the network and forwards it for examination. See more her

**My Traceroute (MTR)**

Is an application that combines the functions of the traceroute and ping programs in one network diagnostic tool

**Terraform is an open source infrastructure as code tool created by HashiCorp that lets you provision Google Cloud resources with declarative configuration files**   
  
**IaC configuration workflow**

**Scope**

Confirm the

resources required

for a project.

**Author**

Author the

configuration files

based on the scope.

**Initialize**

Download the

provider plugins and

initialize directory

**Plan**

View execution plan

for resources

created, modified,

or destroyed.

**Apply**

Create actual

infrastructure

resources.

What is infrastructure as code (IaC)?  
Infrastructure as code, as the name indicates, involves managing and provisioning the cloud infrastructure as code through a configuration language. In other words, instead of clicking around a web UI or using SSH to connect to a server and manually executing commands, with IaC you can write code in files to define, provision, and manage your infrastructure. Gone are the days where a system administrator has to manually configure hundreds of servers, networks, and firewall rules by interacting with a UI. With IaC, you only need to declare the desired end state of the infrastructure, and IaC tools manage provisioning the infrastructure.  
**terraform init : command** initialize required plugins and downloads   
**terraform Plan** The "terraform plan" command generates an execution plan that shows what actions Terraform will take to achieve the desired state.   
**terraform Apply**: The "terraform apply" command then applies those changes to the infrastructure.  
  
  
**Terraform features**Multi-cloud and multi-API

Open core with enterprise support

Large community

Infrastructure provisioning

advantages   
Declarative   
Specify the desired state of infrastructure, not updates.

Code management  
Commit, version, trace, and collaborate, just like source code.

Auditable  
Compare infrastructure between desired state and current state.

Portable  
Build reusable modules across an organization.  
  
Problems that IaC can solve  
● Inability to scale rapidly: High business demand requires the rapid scaling of

IT infrastructure across industries.

● Operational bottlenecks: Due to the rapid scaling of IT infrastructure, Ops

teams need to overcome new organizational and technical bottlenecks, such

as managing infrastructure consistently in scale.

● Disconnected feedback loops: Whenever the infrastructure is changed, the

dev and Ops teams struggle to collaborate and audit changes. Both closing

the communication gap and auditing changes between the software and IT

teams are imperative for successful deployments.

● Manual errors: Increased quantity and scale has led to greater human error

with the potential for significant impacts.  
Provisioning and configuring are two terms that are sometimes misinterpreted

**Infrastructure as code**

● Used for provisioning and managing cloud

resources.   
● Example: Creating and provisioning a VM

instance.  
 ● Referring to frameworks that manipulate

Google Cloud APIs to deploy the

infrastructure.

**Configuration Management**

● Used for virtual machine OS-level

configuration.   
● Example: Configuring the internals of the

VMs.   
● Referring to package configurations and

software maintenance.

.

**What is Compute Engine, and what is its primary purpose of cloud computing?**

 Google Compute Engine is an Infrastructure as a Service (IaaS). It allows users to run virtual machines (VMs) in the cloud, providing scalable and flexible computing resources**.**

**How can you create a VM instance in Compute Engine, and what parameters can you customize during the creation process?**

 You can create a VM instance through the Google Cloud Console or by using the gcloud command-line tool. During creation, you can customize parameters such as machine type, boot disk image, network settings, and metadata.

**Explain the concept of Virtual Private Cloud (VPC) in Google Compute Engine.**

 Virtual Private Cloud (VPC) is a virtual network that provides isolation for resources within Google Cloud. It allows you to define and control IP addresses, firewall rules, and route tables for your virtual machine instances.

**What role does a firewall play in Google Compute Engine, and how can you configure it?**

It control incoming and outgoing traffic to VM instances. You can configure firewall rules to specify which traffic is allowed or denied based on source IP, destination IP, and port.

**Describe the auto-scaling features available in Google Compute Engine and when it is beneficial to use them.**

 Google Compute Engine offers managed instance groups with auto-scaling capabilities**.** Auto-scaling allows you to automatically adjust the number of VM instances in response to changes in demand, ensuring optimal resource utilization*.*

**How can you configure networking options for a virtual machine instance in Google Compute Engine?**

 You can configure networking options during instance creation, including specifying the VPC, subnetwork, external IP address, and defining custom network tags for firewall rules. Additional configurations can be made post-creation through the Google Cloud Console or using gcloud commands.

* **What types of storage options are available for VM instances in Google Compute Engine?**

*Google Compute Engine offers persistent disks and local SSDs.* Persistent disks are network-attached storage devices, while local SSDs provide high-performance, temporary storage that is physically attached to the host machine.

**Explain how Identity and Access Management (IAM) works**.

 IAM allows you to manage access control for resources. You can grant specific permissions to users or groups, defining who can perform actions on resources like virtual machine instances.

**How does Google Compute Engine integrate with other Google Cloud services?**

 Google Compute Engine can integrate with various Google Cloud services through APIs and SDKs. For example, it can connect with services like Google Cloud Storage, Google Cloud SQL, and Google Kubernetes Engine for a comprehensive cloud computing solution.

**What factors contribute to the cost of running virtual machine instances on Google Compute Engine?**

 The cost of running virtual machine instances is influenced by factors such as the *selected machine type, usage duration, disk storage, network usage, and any additional features or services utilized*.

**How can you SSH into a virtual machine instance on Google Compute Engine.?**

You can SSH into a virtual machine instance using the gcloud compute ssh command or through the Google Cloud Console  
 **what are some best practices for securing SSH access?**

Best practices for securing SSH access include using SSH keys, disabling password authentication, and restricting access through firewall rules.

**Explain the purpose of custom metadata for virtual machine instances in Google Compute Engine.**

Custom metadata allows you to associate key-value pairs with your virtual machine instances. This metadata can be used to pass configuration information to instances or for scripting purposes.

**How can you create a snapshot of a persistent disk in Google Compute Engine, and what are snapshots used for?**

 You can create a snapshot using the Google Cloud Console or the gcloud command-line tool. Snapshots are used for backup and recovery purposes, allowing you to capture a point-in-time image of a persistent disk.

**Explain the difference between preemptible and regular (non-preemptible) virtual machine instances in Google Compute Engine.**

 Preemptible instances are short-lived, cost-effective instances that can be terminated by Google Cloud with short notice. Regular instances are persistent and remain running until manually stopped or terminated.

**What is the significance of labels in Google Compute Engine, and how can they be used to organize resources?**

 Labels are key-value pairs that can be attached to resources, providing a way to categorize and organize them. They can be used for resource identification, billing, and filtering when managing multiple resources.

**How can you monitor the performance of virtual machine instances in Google Compute Engine, and what tools are available for this purpose?**

 Monitoring tools such as Google Cloud Monitoring and Stackdriver provide insights into the performance and health of virtual machine instances. You can set up alerts, view metrics, and analyze logs to monitor resource usage.

**How can you troubleshoot connectivity issues between virtual machine instances in the same network on Google Compute Engine?**

 Troubleshooting steps may include checking firewall rules, verifying network configurations, and using tools like ping and traceroute to diagnose and resolve connectivity issues.

* **Explain the role of Cloud Storage in relation to virtual machine instances on Google Compute Engine**.

 Cloud Storage can be used to store and manage data separately from virtual machine instances. It provides scalable and durable object storage, and you can access data stored in Cloud Storage from your virtual machine instances.

**What is the purpose of a custom image in Google Compute Engine, and how can you create and use one?**

 A custom image is a boot disk image that you create and use as a template for creating new virtual machine instances. You can capture the state of an instance and use it to create identical instances in the future.

**create a bucket**

1. In the Google Cloud console, on the **Navigation menu** (Navigation menu icon), click **Cloud overview > Dashboard** .
2. In the **Dashboard** tab of the resulting screen, the **Project info** section shows your Google Cloud project ID.
3. Select and copy the project ID.
4. In the Google Cloud console, on the **Navigation menu** (Navigation menu icon), click **Cloud Storage** > **Buckets**.
5. Click **Create bucket**.
6. For **Name**, paste in the Google Cloud project ID string you copied in an earlier step. These lab instructions will later refer to the name that you typed as [BUCKET\_NAME].
7. Click **Continue**.
8. For **Location Type**, select **Region** and select REGION from the dropdown.
9. Click **Continue**.
10. Click on **Choose how to control access to objects** and uncheck **Enforce public access prevention on this bucket**, then select **Fine-grained**.
11. Click **Continue**.
12. Leave all other values as their defaults.
13. Click **Create**.

### With out json key we can use this gcloud auth print-access-token :---- to get token Create a virtual machine (VM) instance

1. On the **Navigation menu** (Navigation menu icon), click **Compute Engine** > **VM instances**.
2. Click **Create instance**.
3. For **Name**, type **first-vm** as the name of your instance.
4. For **Region**, select REGION.
5. For **Zone**, select ZONE.
6. For **Machine type**, examine the options.
7. To see the breakdown of estimated costs, click **Details** to the right of the **Machine type** list underneath the estimated costs.
8. For **Machine type**, click **2 vCPUs (e2-standard-2)**.

How did the cost change?

1. For **Machine type,** click **e2-micro (2 shared vCPU)**.

The micro type is a shared-core VM that is inexpensive.

1. For **Firewall**, click **Allow HTTP traffic**.
2. Leave the remaining settings as their defaults, and click **Create**.

**Explore the VM details**

1. On the **VM instances** page, click the name of your VM: **first-vm**.
2. In **Machine configuration**, notice the value of **CPU platform**, notice the value, and click **Edit**.
3. Click on the pencil icon on the top to edit the first-vm instance.

**Note:** You can't change the machine type, the CPU platform, or the zone of a running Google Cloud VM. You can add network tags and allow specific network traffic from the internet through firewalls.

1. Scroll down to the **Management** section and examine **Availability policies**.
2. Click **Cancel**.

### Create an IAM service account

An IAM service account is a special type of Google account that belongs to an application or a virtual machine, instead of to an individual end user.

1. On the **Navigation menu**, click **IAM & admin** > **Service accounts**.
2. Click **+ Create service account**.
3. On the **Service account details** page, specify the **Service account name** as test-service-account.
4. Click **Create and Continue**.
5. On the **Grant this service account access to project** page, specify the role as **Basic** > **Editor**.
6. Click **Continue**.
7. Click **Done**.
8. On the **Service accounts** page, move to the extreme right of the test-service-account and click on the three dots.
9. Click **Manage keys**.
10. Click **ADD KEY**
11. Select **Create new key**
12. Select **JSON** as the key type.
13. Click **Create**.

A JSON key file is downloaded. In a later step, you find this key file and upload it to the VM.

1. Click **Close**.

**Cloud Shell, use the gsutil command to create a bucket:**

gcloud storage buckets create gs://$MY\_BUCKET\_NAME\_2 --location=REGION

In the Google Cloud console, on the **Navigation menu** (Navigation menu icon), click **Cloud Storage** > **Buckets**, or click **Refresh** if you are already in the Cloud Storage page.

The second bucket should appear in the **Buckets** list.

1. **Create a VM in the default zone that you set earlier in this task using the new environment variable to assign the VM name:**

gcloud compute instances create $MY\_VMNAME \

--machine-type "e2-standard-2" \

--image-project "debian-cloud" \

--image-family "debian-11" \

--subnet "default"

### Use the gcloud command line to create a second service account

1. In Cloud Shell, execute the following command to create a new service account:

gcloud iam service-accounts create test-service-account2 --display-name "test-service-account2"

1. **Note:** If you see the following output, type Y and press ENTER:  
     
   **In Cloud Shell, execute the following command to grant the second service account the viewer role:**
2. gcloud projects add-iam-policy-binding $GOOGLE\_CLOUD\_PROJECT --member serviceAccount:test-service-account2@${GOOGLE\_CLOUD\_PROJECT}.iam.gserviceaccount.com --role roles/viewer  
     
     
     
   **Then, to change the object to have private access, execute the following command:**

gsutil acl set private gs://$MY\_BUCKET\_NAME\_1/cat.jpg

git clone https://github.com/googlecodelabs/orchestrate-with-kubernetes.git

In Cloud Shell, execute the following command to create a test directory:

mkdir test

## Building containers with DockerFile and Cloud Build

1. On the Google Cloud console title bar, click **Activate Cloud Shell**.
2. When prompted, click **Continue**.

Cloud Shell opens at the bottom of the Google Cloud console window.

1. Create an empty quickstart.sh file using the nano text editor:

nano quickstart.sh

1. Add the following lines in to the quickstart.sh file:

#!/bin/sh

echo "Hello, world! The time is $(date)."

1. Save the file and close nano by pressing the CTRL+X keys, then press Y and ENTER.
2. Create an empty Dockerfile file using the nano text editor:

nano Dockerfile

1. Add the following Dockerfile command:

FROM alpine

This instructs the build to use the Alpine Linux base image.

1. Add the following Dockerfile command to the end of the Dockerfile:

COPY quickstart.sh /

This adds the quickstart.sh script to the / directory in the image.

1. Add the following Dockerfile command to the end of the Dockerfile:

CMD ["/quickstart.sh"]

This configures the image to execute the /quickstart.sh script when the associated container is created and run.

The Dockerfile should now look like this:

FROM alpine

COPY quickstart.sh /

CMD ["/quickstart.sh"]

1. Save the file and close nano by pressing the CTRL+X keys, then press Y and ENTER.
2. In Cloud Shell, run the following command to make the quickstart.sh script executable:

chmod +x quickstart.sh

1. Create a new Docker repository named quickstart-docker-repo in the location with the description "Docker repository"

gcloud artifacts repositories create quickstart-docker-repo --repository-format=docker \

--location="REGION" --description="Docker repository"

1. In Cloud Shell, run the following command to build the Docker container image in Cloud Build:

gcloud builds submit --region=us-central1 --tag us-central1-docker.pkg.dev/${DEVSHELL\_PROJECT\_ID}/quickstart-docker-repo/quickstart-image:tag1

When the build completes, your Docker image is built and pushed to the Artifact Registry.

1. In the Google Cloud console, in the **Search Bar** (Located at the top of the console window), Search for **Artifact Registry**.
2. Click the repository named quickstart-docker-repo.

The quickstart-image Docker image appears in the list.

## Building containers with a build configuration file and Cloud Build

1. In Cloud Shell enter the following command to clone the repository to the lab Cloud Shell:

git clone <https://github.com/GoogleCloudPlatform/training-data-analyst>

1. Create a soft link as a shortcut to the working directory:

ln -s ~/training-data-analyst/courses/ak8s/v1.1 ~/ak8s

1. Change to the directory that contains the sample files for this lab:

cd ~/ak8s/Cloud\_Build/a

A sample custom cloud build configuration file called cloudbuild.yaml has been provided for you in this directory as well as copies of the Dockerfile and the quickstart.sh script you created in the first task.

1. Run the below command.

export REGION="REGION"

sed -i "s/us-central1/$REGION/g" cloudbuild.yaml

1. In Cloud Shell, execute the following command to view the contents of cloudbuild.yaml:

cat cloudbuild.yaml

1. n Cloud Shell, execute the following command to start a Cloud Build using cloudbuild.yaml as the build configuration file:

gcloud builds submit --config cloudbuild.yaml

1. In the Google Cloud console, in the **Search Bar** (Located at the top of the console window), Search for **Artifact Registry**.
2. Click the repository named quickstart-docker-repo.

Two versions of quickstart-image are now in the list.

1. In the Google Cloud Console, on the **Navigation menu** (Navigation menu icon), click **Cloud Build** > **History**. From the dropdown select the **Region** as **us-central1**.

Two builds appear in the list.

1. Click the build ID for the build at the top of the list.

The details of the build, including the build log, are displayed.

1. **In Cloud Shell, change to the directory that contains the sample files for this lab:**

cd ~/ak8s/Cloud\_Build/b

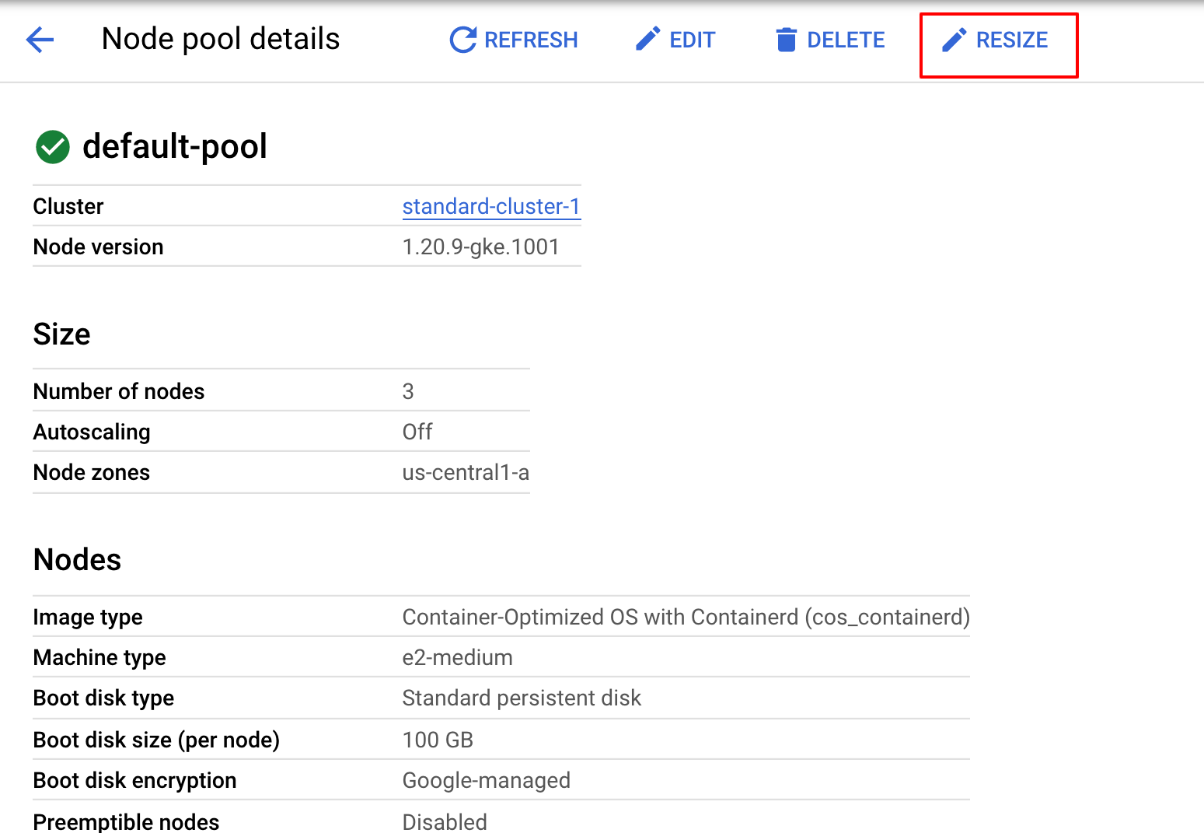
1. **tart a Cloud Build using cloudbuild.yaml as the build configuration file:**

gcloud builds submit --config cloudbuild.yaml

## Modify GKE clusters

It is easy to modify many of the parameters of existing clusters using either the Google Cloud Console or Cloud Shell. In this task, you use the Google Cloud Console to modify the size of GKE clusters.

1. In the Google Cloud Console, on the **Navigation menu** (Navigation menu icon), click **Kubernetes Engine** > **Clusters** > **standard-cluster-1**, click **NODES** at the top of the details page.
2. In **Node Pools** section, click **default-pool**.
3. In the Google Cloud Console, click **RESIZE** at the top of the **Node Pool Details** page.
4. Change the number of nodes from 3 to 4 and click **RESIZE**.



1. In the Google Cloud Console, on the **Navigation menu** (Navigation menu icon), click **Kubernetes Engine** > **Clusters**.

When the operation completes, the **Kubernetes Engine > Clusters** page should show that standard-cluster-1 now has four nodes.

## Deploy a sample workload

In this task, using the Google Cloud console you will deploy a Pod running the nginx web server as a sample workload.

1. In the Google Cloud Console, on the **Navigation menu**(Navigation menu icon), click **Kubernetes Engine** > **Workloads**.
2. Click **Create Deployment** to show the Create a deployment wizard.
3. Click **Continue** to accept the default container image, nginx:latest, which deploys 3 Pods each with a single container running the latest version of nginx.
4. Scroll to the bottom of the window and click the **Deploy** button leaving the **Configuration** details at the defaults.
5. When the deployment completes your screen will refresh to show the details of your new nginx deploymen

## View details about workloads in the Google Cloud Console.

1. In the Google Cloud Console, on the **Navigation menu** (Navigation menu icon), click **Kubernetes Engine** > **Workloads**.
2. In the Google Cloud Console, on the **Kubernetes Engine > Workloads** page, click **nginx-1**.

This displays the overview information for the workload showing details like resource utilization charts, links to logs, and details of the Pods associated with this workload.

1. In the Google Cloud Console, click the **Details** tab for the **nginx-1** workload. The Details tab shows more details about the workload including the Pod specification, number and status of Pod replicas and details about the horizontal Pod autoscaler.
2. Click the **Revision History** tab. This displays a list of the revisions that have been made to this workload.
3. Click the **Events** tab. This tab lists events associated with this workload.
4. And then the **YAML** tab. This tab provides the complete YAML file that defines these components and full configuration of this sample workload.
5. Still in the Google Cloud Console's **Details** tab for the **nginx-1** workload, click the **Overview** tab, scroll down to the **Managed Pods** section and click the name of one of the Pods to view the details page for that Pod.
6. The Pod details page provides information on the Pod configuration and resource utilization and the node where the Pod is running.
7. In the **Pod details** page, you can click the Events and Logs tabs to view event details and links to container logs in Cloud Operations.
8. Click the **YAML** tab to view the detailed YAML file for the Pod configuration.

**kubectl get pods**, which list of all of the pods in the cluster and tells you their status  
For more information on a specific pod,   
**kubectl describe my-pod-name**, which gives you detailed information about the pod named my-pod-name.  
  
**kubectl exec my-pod-name** You can even test and debug within your pod using the command to execute commands and applications.

The logs command provides a powerful tool to see what's happening inside a pod.

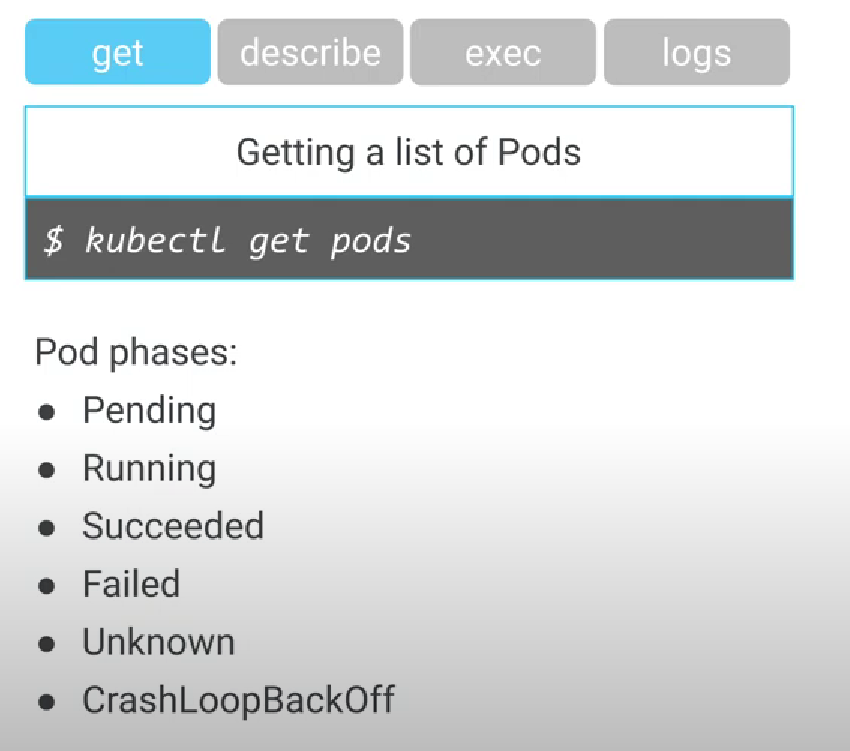
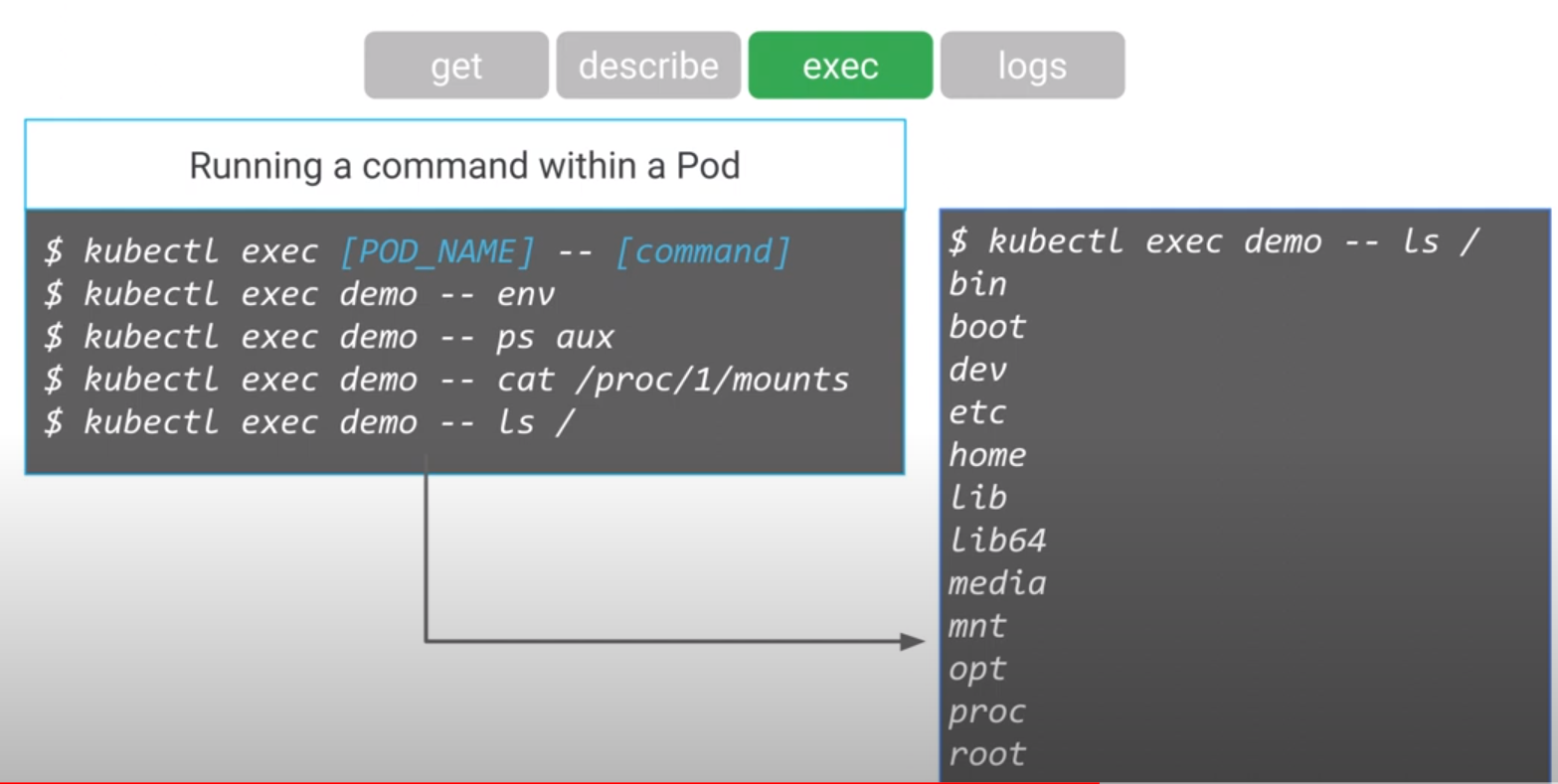
Logs are always useful in troubleshooting.

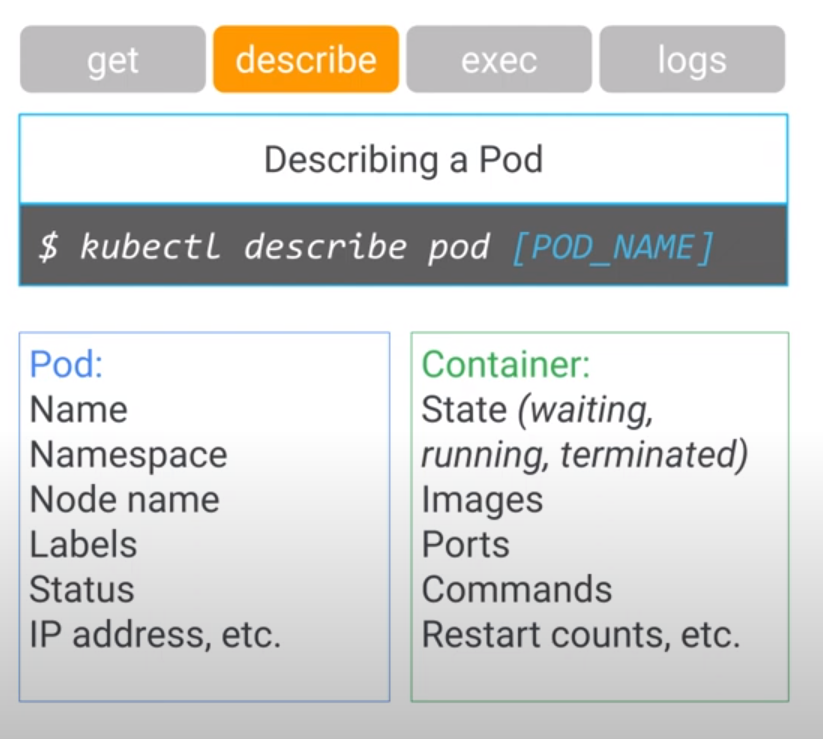
The log command allows you to quickly and easily view errors or debugging messages written out by the applications running inside pods.

The logs command is particularly useful when you need to find out more information about containers that are failing to run successfully.

CrashLoopBackOff means that one of the containers in the pod exited unexpectedly even after it was restarted at least once.

This is a common error.

Usually CrashLoopBackOff means the pod isn't configured correctly.  
  
  


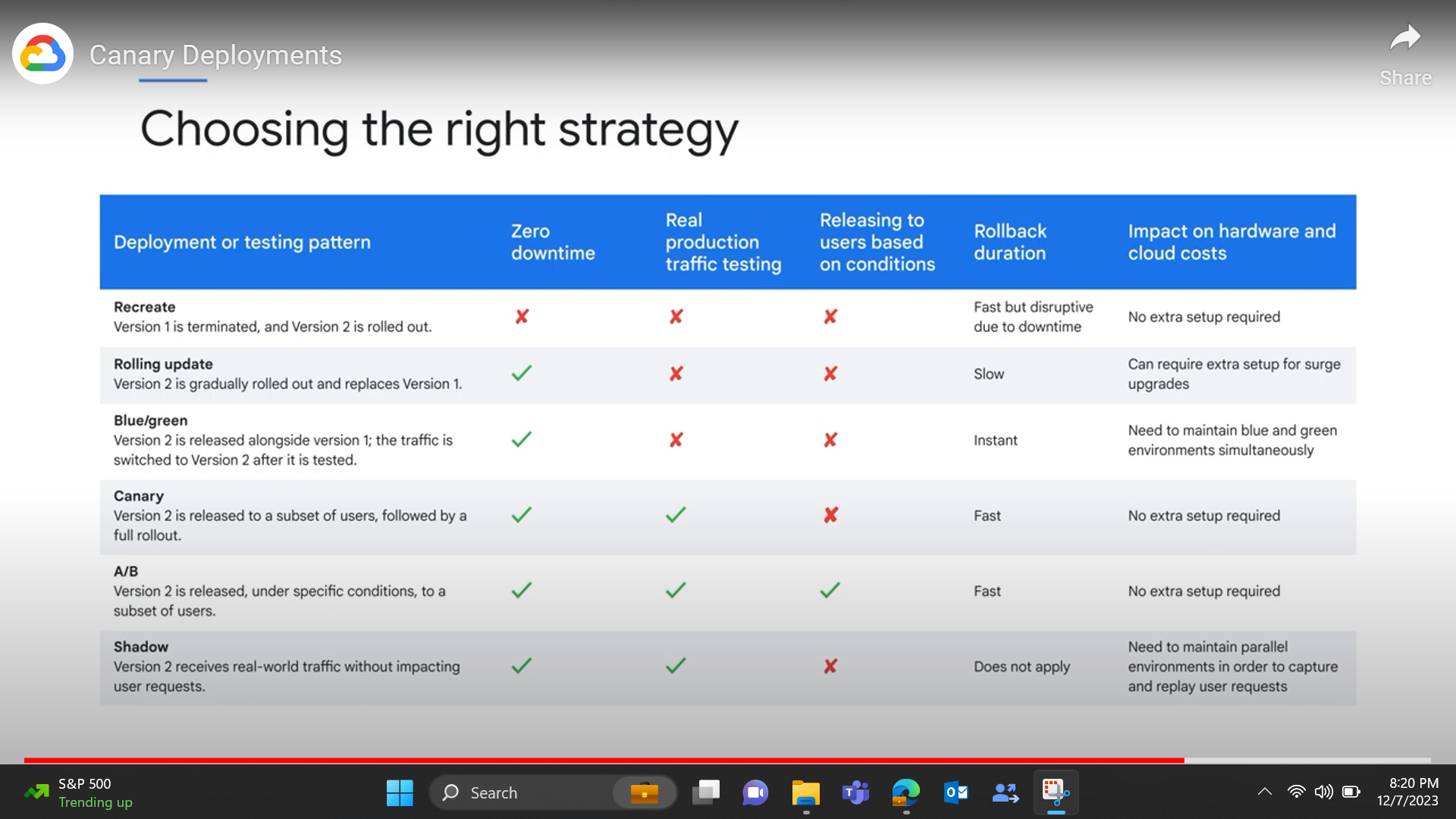
  
  
  
resize the nodes:  
  
  
  
gcloud container clusters resize $my\_cluster --zone $my\_zone --num-nodes=4

**Open the kubeconfig file with the nano text editor:**

nano ~/.kube/config

**You can now examine all of the authentication and endpoint configuration data stored in the file. Information for the cluster should appear.. The information was populated during cluster creation.**

1. **Press CTRL+X to exit the nano editor.**

*   
    
    
  You can list the active account name with this command:

gcloud auth list

* You can list the project ID with this command:

gcloud config list project

In Cloud Shell, type the following command to set the environment variable for the zone and cluster name:

export my\_zone=ZONE

export my\_cluster=standard-cluster-1

to view a list of Pods in the deployments, execute the following command:

kubectl get deployments

To scale the Pod back up to three replicas, execute the following command:

kubectl scale --replicas=3 deployment nginx-deployment

### Trigger a deployment rollout

To update the version of nginx in the deployment, execute the following command:

kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1 --record

To view the rollout status, execute the following command:

kubectl rollout status deployment.v1.apps/nginx-deployment

View the rollout history of the deployment:

kubectl rollout history deployment nginx-deployment

To roll back to the previous version of the nginx deployment, execute the following command:

kubectl rollout undo deployments nginx-deployment

View the updated rollout history of the deployment:

kubectl rollout history deployment nginx-deployment

‘  
  
  
  
yaml file deployment  
apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-canary

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

track: canary

Version: 1.9.1

spec:

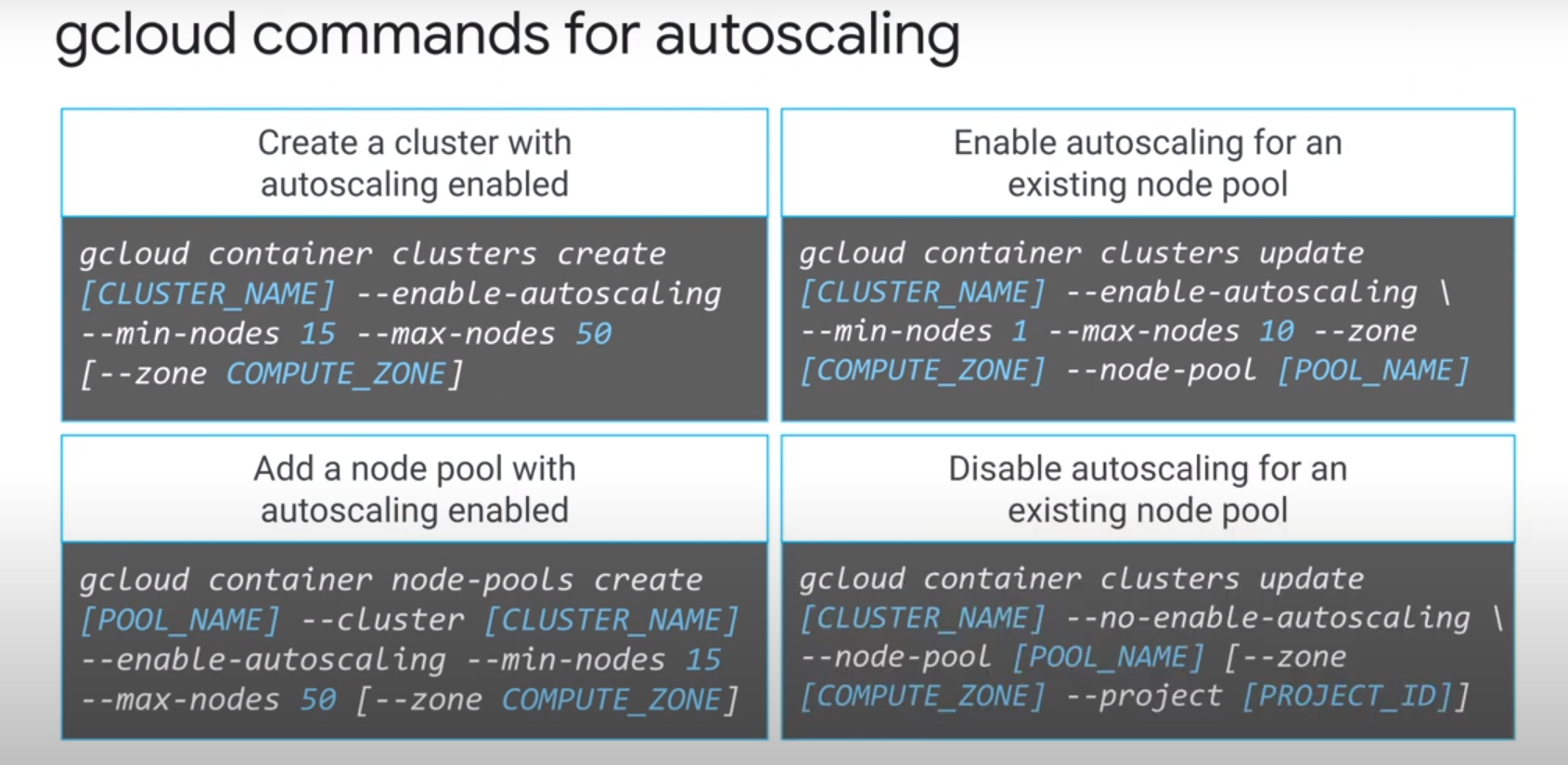
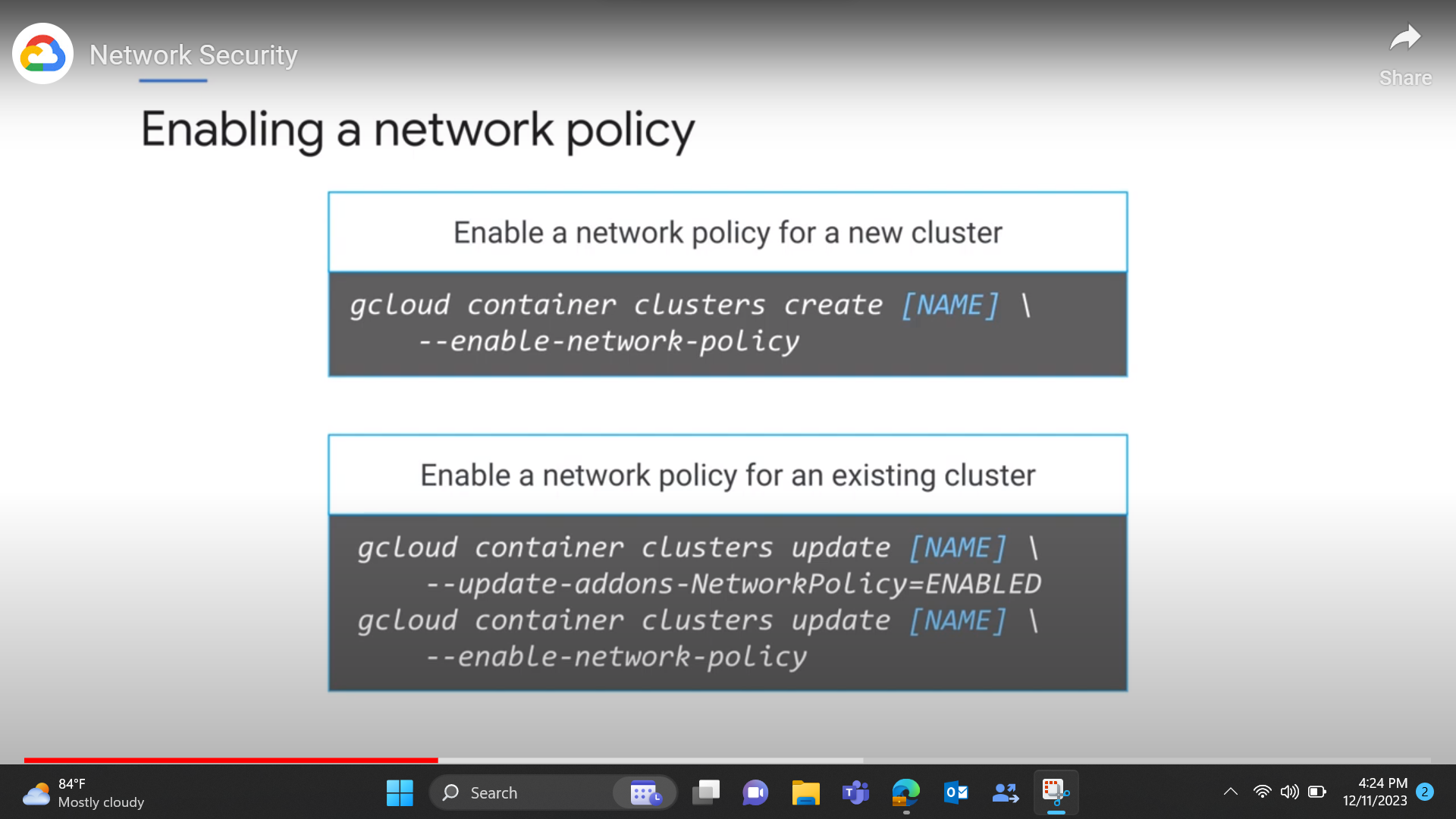
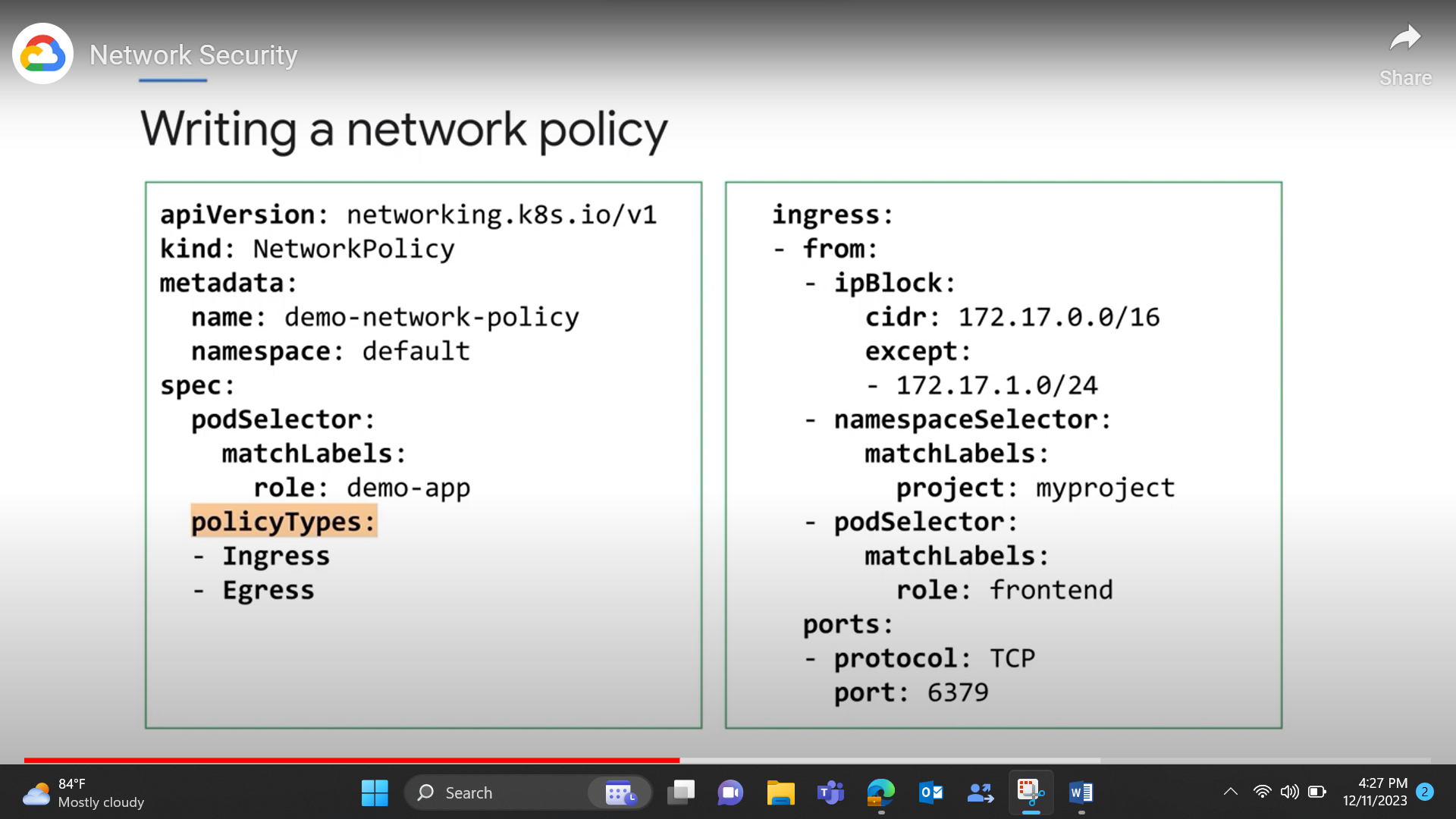
containers:

- name: nginx

image: nginx:1.9.1

ports:

- containerPort: 80  
  
  
  
Gcloud commands for autoscaling

  
  
  
  
Enabling a network policy  
  
  
  
  
  
  
**kubectl get namespaces -** list of all namespace

**kubectl get pods** - list of all pods

**kubectl get pods -o wide - l**ist of all pods

**kubectl get pods --field-selector=spec. nodeName=[server-name]**

- list of all pods running on a particular node server

**kubectl get replicationcontroller [replication-controllername] -** List a specific replication controller in plain text

**kubectl get replicationcontroller, services**- list of all replication controllers and services

**kubectl get deamonset-** list of all daemon set

**kubectl create namespace [namespace-name]**- Create a new namespace

**kubectl create –f [filename]**- Create a resource from a JSON or YAML file

**kubectl describe nodes [node-name]**- See details about a particular node

**kubectl describe pods [pod-name]**- See details about a particular pod

**Kubectl describe –f pod.json**- See details about a pod whose name and type are listed in pod.json

**kubectl describe pods**- See details about all pods

|  |  |
| --- | --- |
| what is storage? | In gcp storage refers to the varoius services and solutions offered to store and manage data in the cloud.there are three types of storage in GCP: block storage,object storage, filestore |
| what is database? | It is managed service used to store ,manage and retrieve structured data. GCP offers two types of databases like relational and non relational |
| what is difference between storage and database? |  |
| what are storage options? | there are three types of storage in GCP: cloud storage,filestore,persistent disk |
| what is object storage? | object storage is also called as cloud storage.it is a computer data storage that manages data as objects not in files and folders hierarchy. it manages data in an unstructured format called objects. unstructured data means videos, email,web pages ,s ensor data. |
| what is filestore? | file storage stores data as files and presents it to its final users as a hierarchial directories structure. |
| what is difference between relational and non relational database? | relational:stored data in structured format like in tables with rows and columns. examples of relational databases are cloud sql,spanner. non relational:stores data in various formats like key-value pairs,documents,graphs, videos and images. examples are cloud firestore,cloud bigtable, cloud datastore. |
| what are cloud storages classes? | storage classes enable users to optimize their storage costs based on how requently they access their data and the level of redundancy required. there are three- types of storage classes : standard storage class,nearline storage class,cold line storage class, archive storage class |
| what is standard storage class? | 1.suitable for frequently accessed data with high availability and low latency. 2.higher storage costs compared to other classes. 3.Best for short-term storage and frequently accessed data |
| what is Nearline storage class? | 1. this is best for storing infrequently accessed data like reading and modifying data on average once a month or less example:data backups,long-tail multimedia content. 2.nearline storage keeps its data stores its data within one region and is cheap compared to standard storage.Best for backups and data accessed less than once a month |
| what is cold-line storage? | Best for disaster recovery and data accessed less than once a quarter.used to store data that needs access a few times a year .coldline has a minimum storage duration of 90 days. it is cost-effective. |
| what is Archive storage? | Best for long-term digital preservation of data accessed less than once a year.it is designed for data that you rarely access and can tolerate higher retrivel times. it offers lower storage costs but comes with the higher retrivel fees and longer retrievel times. it's the best coice for data that you plan to access less than once a year. |
| what is the difference between cloud sql and cloud spanner? | cloud spanner can handle large datasets and can distribute data across multiple regions,making it an ideal choice for businesses that operate globally. it provides low-latency and high throughput.it can distribute data across multiple regions.cloud sql on the other hand, is suitable for businesses that require a fully-managed,relational database solution. it is ideal for small to medium-sized businesses that need to store and manage structured data in a familier sql environment. cloud sql provides automatic backups and can be easily integrated with other google cloud platform services. |
| what is the difference between cloud bigtable and bigquery? | bigtable is a NOSQL database optimized for storing and retrieving large amounts of structured and semi-structured data, whereas bigquery is a data warehouse optimized for running complex SQL queries on large datasets. bigtable is a fully managed service that charges based on usage,while bigquery charges based on the data processed by queries. |
| what is the difference sql and nosql data? | sql means data stored in structured format like tables with rows and coloumns. Nosql databases means graphs and key-value pairs. they are designed to handle structured and un structured data . |
| what is the difference between bucket and storage? | bucket: A bucket in cloud storage is a logical container used to store objects. It's similar to a folder in a file system. but it doesn't have a hierarchical structure.each object stored in a cloud storage belongs to a specific bucket and has a unique object identifier witin that bucket. storage:It is scalable and durable object storage service provided by GCP. cloud storage allows you to store and retrive objects(files) in a highly available,secure and globally distributed manner. it is used for storing unstructured data.within cloud storage data is organized into conatiners |
| what is the difference between datastore and storage? | datastore is a NOSQL document -oriented database service provided by google cloud. it is suitable for handling structured data,allowing users to store and query data using a hierarchical entity model.it's commonly used for web and mobile applications. storage in GCP refers to the various cloud based storages. used for storing large files,backups,media ,logs. |
| what is persistant disk? | it is a block storage device that provides durable and high-performance storage for VM instances.it is used to store data independently from the VM and persists even if the instance got stopped or deleted. |
| what is firestore? | cloud firestore is flexible ,scalable database for mobile,web and server development from firebase and google cloud |
| what is cloud sql? | It is fully managed database service that helps in setting-up , maintain , manage relational database on the GCP.It is a fully-managed relational database service provided by GCP. It allows you to set up, manage, and scale databases in the cloud without worrying about infrastucture management.cloud sql supports popular database engines like MYSQL,postgreSQL, and SQL server. |
| what is cloud spanner? | cloud spanner is a fully managed, globally distributed and stongly consistent relational databse service provided by GCP. It offer horizontal scalability across multiple regions while maintaining strong consistency. It handles maintennace,updates ,backups and scaling. |
| what is block storage? | block storage involves dividing data into fixed-sized blocks and storing them on block devices such as ard drives and SSDs. these blocks are accesses using low-level protocols.it is typically used to create and manage persistent disks for virtual machine instances. |
| what is bigtable? | It is google NOSQL bigdata databse service. It is designed to handle massive workloads at consistent low letency and high throughout. IT is commonly used for various purposes such as IOT data storage,time-series data analysis, financial data processing. |
| Can I control access to my data in Google Cloud Storage? | Yes, Google Cloud Storage offers robust access control mechanisms. You can manage access permissions at the bucket and object levels using Identity and Access Management (IAM) policies and access control lists (ACLs). |
| Is there a way to migrate my on-premises data to Google Cloud Storage? | Yes, Google Cloud provides various tools and methods for migrating data, such as using the Transfer Appliance, Storage Transfer Service, gsutil (a command-line tool), or third-party solutions. |
| What is the difference between regional and multi-regional storage in Google Cloud Storage? | Regional storage keeps data in a specific geographic location (like a single region), while multi-regional storage distributes data across multiple regions for higher availability and global accessibility. |
| What is the maximum size of an object that I can store in Google Cloud Storage? | The maximum size of an object stored in Google Cloud Storage is 5 terabytes (TB). Larger objects can be stored by using a method called object composition or by splitting them into multiple parts. |
| what is bigquery? | BigQuery is a serverless, highly scalable, and fully managed data warehouse that's designed for running SQL-like queries on large datasets. It's ideal for data analysis, business intelligence, and ad-hoc querying.BigQuery organizes data in tables with a schema that can be defined, and it supports semi-structured data types like JSON, allowing flexibility in data ingestion. |
| How can you control access to your objects in Google Cloud Storage, and what is the role of Identity and Access Management (IAM)? | Access to objects is controlled through Cloud Storage's fine-grained access control and IAM. IAM allows you to grant specific permissions to users or groups, determining who can perform actions on your buckets and objects. |
| What is a bucket in Google Cloud Storage, and how do you create one? | A bucket is a container for storing objects in Google Cloud Storage. You can create a bucket using the Cloud Console, the gsutil command-line tool, or through the Cloud Storage API. |

|  |  |
| --- | --- |
| what is storage? | In gcp storage refers to the varoius services and solutions offered to store and manage data in the cloud.there are three types of storage in GCP: block storage,object storage, filestore |
| what is database? | It is managed service used to store ,manage and retrieve structured data. GCP offers two types of databases like relational and non relational |
| what is difference between storage and database? |  |
| what are storage options? | there are three types of storage in GCP: cloud storage,filestore,persistent disk |
| what is object storage? | object storage is also called as cloud storage.it is a computer data storage that manages data as objects not in files and folders hierarchy. it manages data in an unstructured format called objects. unstructured data means videos, email,web pages ,s ensor data. |
| what is filestore? | file storage stores data as files and presents it to its final users as a hierarchial directories structure. |
| what is difference between relational and non relational database? | relational:stored data in structured format like in tables with rows and columns. examples of relational databases are cloud sql,spanner. non relational:stores data in various formats like key-value pairs,documents,graphs, videos and images. examples are cloud firestore,cloud bigtable, cloud datastore. |
| what are cloud storages classes? | storage classes enable users to optimize their storage costs based on how requently they access their data and the level of redundancy required. there are three- types of storage classes : standard storage class,nearline storage class,cold line storage class, archive storage class |
| what is standard storage class? | 1.suitable for frequently accessed data with high availability and low latency. 2.higher storage costs compared to other classes. 3.Best for short-term storage and frequently accessed data |
| what is Nearline storage class? | 1. this is best for storing infrequently accessed data like reading and modifying data on average once a month or less example:data backups,long-tail multimedia content. 2.nearline storage keeps its data stores its data within one region and is cheap compared to standard storage.Best for backups and data accessed less than once a month |
| what is cold-line storage? | Best for disaster recovery and data accessed less than once a quarter.used to store data that needs access a few times a year .coldline has a minimum storage duration of 90 days. it is cost-effective. |
| what is Archive storage? | Best for long-term digital preservation of data accessed less than once a year.it is designed for data that you rarely access and can tolerate higher retrivel times. it offers lower storage costs but comes with the higher retrivel fees and longer retrievel times. it's the best coice for data that you plan to access less than once a year. |
| what is the difference between cloud sql and cloud spanner? | cloud spanner can handle large datasets and can distribute data across multiple regions,making it an ideal choice for businesses that operate globally. it provides low-latency and high throughput.it can distribute data across multiple regions.cloud sql on the other hand, is suitable for businesses that require a fully-managed,relational database solution. it is ideal for small to medium-sized businesses that need to store and manage structured data in a familier sql environment. cloud sql provides automatic backups and can be easily integrated with other google cloud platform services. |
| what is the difference between cloud bigtable and bigquery? | bigtable is a NOSQL database optimized for storing and retrieving large amounts of structured and semi-structured data, whereas bigquery is a data warehouse optimized for running complex SQL queries on large datasets. bigtable is a fully managed service that charges based on usage,while bigquery charges based on the data processed by queries. |
| what is the difference sql and nosql data? | sql means data stored in structured format like tables with rows and coloumns. Nosql databases means graphs and key-value pairs. they are designed to handle structured and un structured data . |
| what is the difference between bucket and storage? | bucket: A bucket in cloud storage is a logical container used to store objects. It's similar to a folder in a file system. but it doesn't have a hierarchical structure.each object stored in a cloud storage belongs to a specific bucket and has a unique object identifier witin that bucket. storage:It is scalable and durable object storage service provided by GCP. cloud storage allows you to store and retrive objects(files) in a highly available,secure and globally distributed manner. it is used for storing unstructured data.within cloud storage data is organized into conatiners |
| what is the difference between datastore and storage? | datastore is a NOSQL document -oriented database service provided by google cloud. it is suitable for handling structured data,allowing users to store and query data using a hierarchical entity model.it's commonly used for web and mobile applications. storage in GCP refers to the various cloud based storages. used for storing large files,backups,media ,logs. |
| what is persistant disk? | it is a block storage device that provides durable and high-performance storage for VM instances.it is used to store data independently from the VM and persists even if the instance got stopped or deleted. |
| what is firestore? | cloud firestore is flexible ,scalable database for mobile,web and server development from firebase and google cloud |
| what is cloud sql? | It is fully managed database service that helps in setting-up , maintain , manage relational database on the GCP.It is a fully-managed relational database service provided by GCP. It allows you to set up, manage, and scale databases in the cloud without worrying about infrastucture management.cloud sql supports popular database engines like MYSQL,postgreSQL, and SQL server. |
| what is cloud spanner? | cloud spanner is a fully managed, globally distributed and stongly consistent relational databse service provided by GCP. It offer horizontal scalability across multiple regions while maintaining strong consistency. It handles maintennace,updates ,backups and scaling. |
| what is block storage? | block storage involves dividing data into fixed-sized blocks and storing them on block devices such as ard drives and SSDs. these blocks are accesses using low-level protocols.it is typically used to create and manage persistent disks for virtual machine instances. |
| what is bigtable? | It is google NOSQL bigdata databse service. It is designed to handle massive workloads at consistent low letency and high throughout. IT is commonly used for various purposes such as IOT data storage,time-series data analysis, financial data processing. |
| Can I control access to my data in Google Cloud Storage? | Yes, Google Cloud Storage offers robust access control mechanisms. You can manage access permissions at the bucket and object levels using Identity and Access Management (IAM) policies and access control lists (ACLs). |
| Is there a way to migrate my on-premises data to Google Cloud Storage? | Yes, Google Cloud provides various tools and methods for migrating data, such as using the Transfer Appliance, Storage Transfer Service, gsutil (a command-line tool), or third-party solutions. |
| What is the difference between regional and multi-regional storage in Google Cloud Storage? | Regional storage keeps data in a specific geographic location (like a single region), while multi-regional storage distributes data across multiple regions for higher availability and global accessibility. |
| What is the maximum size of an object that I can store in Google Cloud Storage? | The maximum size of an object stored in Google Cloud Storage is 5 terabytes (TB). Larger objects can be stored by using a method called object composition or by splitting them into multiple parts. |
| what is bigquery? | BigQuery is a serverless, highly scalable, and fully managed data warehouse that's designed for running SQL-like queries on large datasets. It's ideal for data analysis, business intelligence, and ad-hoc querying.BigQuery organizes data in tables with a schema that can be defined, and it supports semi-structured data types like JSON, allowing flexibility in data ingestion. |
| How can you control access to your objects in Google Cloud Storage, and what is the role of Identity and Access Management (IAM)? | Access to objects is controlled through Cloud Storage's fine-grained access control and IAM. IAM allows you to grant specific permissions to users or groups, determining who can perform actions on your buckets and objects. |
| What is a bucket in Google Cloud Storage, and how do you create one? | A bucket is a container for storing objects in Google Cloud Storage. You can create a bucket using the Cloud Console, the gsutil command-line tool, or through the Cloud Storage API. |

|  |  |  |  |
| --- | --- | --- | --- |
| Questions | | **Answers** | |
| [Virtual Private Cloud (VPC)](https://cloud.google.com/vpc/docs/vpc) | | VPC is a Logical representation of an on-prem network. This is a global construct in GCP. | |
| [VPC modes](https://cloud.google.com/vpc/docs/vpc) | | There are two modes in GCP. Auto\_mode and Custom\_Mode | |
| VPC subnets | | In GCP these are regional and assigned to an IP address range | |
| [IP ADDRESS](https://cloud.google.com/vpc/docs/subnets) | | A unique address used to identity host on network. Made up of network and host portions | |
| [CIDR](https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html) | | This segments and IP address into network and host portions. It determines how must host are available on the network. This can be manipulated to form CIDR blocks. | |
| IPV4 | | This is a 32 bit, 4 octet address. Written in binary or dotted decimal format. E.g. **192.168.10.20** or **11000000.10101000.00001010.00010100** | |
| IPV6 | | This is a 128 bit, hexadecimal address. 2001:DB8:7654:3210:FEDC:BA98:764:3203 | |
| Internal IP | | A special range that can be used internally by anyone. These are non internet routable | |
| Public IP | | IP address that is routable on the internet | |
| DHCP | | Dynamic Host Control protocol. A method to automatically assign an IP address to a client | |
| Static IP | | An IP that does not change after being assigned | |
| Ephemeral IP | | Temporary IP that is not reserved | |
| [What is the OSI Model​](https://learningnetwork.cisco.com/s/article/osi-model-reference-chart) | | 7 layer conceptual model that provides interoperability of the TCP stack​.Application Layer (Layer 7)​,Presentation Layer (Layer 6),Session Layer (Layer 5)​,Transport layer (Layer 4)​,Network layer (Layer 3)​,Data layer (Layer 2)​,Physical layer (Layer 1)​. | |
| Application Layer (Layer 7)​ | | User interface and application. Protocols examples HTTP, HTML​ | |
| Presentation Layer (Layer 6)​ | | Formats data to be presented. Protocols examples JPEG, ASCII, GIF​ | |
| Session Layer (Layer 5)​ | | Creates, tracks, ends the sessions between different systems​ | |
| Transport layer (Layer 4)​ | | Handles message delivery using connection and connectionless protocols. Protocol examples TCP, UDP​ | |
| Network layer (Layer 3)​ | | Focuses on subnets, route path selection. Protocols examples IP, ICMP,. Router work here​ | |
| Data layer (Layer 2)​ | | Focuses of transferring data frames over physical layer. Protocol, ARP, PPP, VLANS. Switches work here​ | |
| Physical layer (Layer 1)​ | | Transmission of raw bits over physical mediums. Examples network cables, wireless​ | |
| What is interoperability?​ | | The ability to communicate between different communication devices in a standard way.​ | |
| [Transmission Control Protocol (TCP)​](https://datatracker.ietf.org/doc/html/rfc793) | | This is a connection oriented protocol that handles reliability, flow and congestion control of packets. It establishes a connection before​ sending a packet. | |
| User Datagram Protocol (UDP) | | UDP is a very common protocol for voice and video traffic.User Datagram Protocol, or UDP, is a communication protocol used across the Internet for especially time-sensitive transmissions such as video playback or DNS lookups. | |
| Maximum transfer unit (MTU)​ | | The size of the largest unit of data that can be transmitted over the network​ | |
| Time to Live (TTL)​ | | this indicates the life of the packet usually has a max of 255 hops. This ensures packets don’t exist forever in a network​ | |
| Domain Name Service (DNS)​ | | Resolves names to IP addresses​. we have two types of DNS:1.Internal DNS-Used internally within a private network​​,2.Hybrid DNS-​DNS configured between cloud and on-prem or external networks. | |
| [Media Access Control address(MAC)​](https://en.wikipedia.org/wiki/MAC_address) | | Unique hexadecimal identifier assigned to a network interface controller (NIC) card. Usually a 12 digit hexadecimal number.​ | |
| Network Address Translation (NAT) | | allows private IP ranges to communicate with the internet. Maintains a NAT table of private to public address & port mappings for communications.​ | |

**diff b/w dynamic block & data source in terraform  
  
Dynamic Blocks:**Purpose: Dynamic blocks allow you to dynamically generate and repeat nested blocks within a resource or module based on a collection of values.

Use Case: Used when you want to create multiple instances of a particular block within a resource, where the number of instances is determined dynamically or based on a variable.

Example:

resource "aws\_instance" "example" {

count = 3

dynamic "ebs\_block\_device" {

for\_each = var.ebs\_volumes

content {

device\_name = ebs\_block\_device.key

volume\_size = ebs\_block\_device.value

}

}

}

Location: Dynamic blocks are used within resource or module blocks.

**Data Sources:**

Purpose: Data sources are used to fetch and retrieve data from external sources or existing infrastructure. They allow you to import information into your Terraform configuration.

Use Case: Useful when you need to query information about existing resources in a cloud provider, fetch data from an API, or obtain information from external systems.

Example:

data "aws\_ami" "latest\_amazon\_linux" {

most\_recent = true

owners = ["amazon"]

filter {

name = "name"

values = ["amzn2-ami-hvm-\*-x86\_64-gp2"]

}

}

Location: Data sources are used outside of resource blocks and are typically declared at the root level of your Terraform configuration

**what is API**

API stands for "Application Programming Interface." An API is a set of rules and protocols that allows one piece of software or application to interact with another. It defines the methods and data formats that applications can use to communicate with each other. APIs are used to enable the integration of different software systems, allowing them to work together and share data or functionality.  
Here are some key points about APIs:

1. **Communication Protocol:** APIs define how different software components should interact. This includes specifying the methods (functions or procedures) that can be called, the data formats that should be used, and the rules for exchanging information.
2. **Abstraction:** APIs provide a level of abstraction, allowing developers to use a service or functionality without needing to understand the internal implementation details. This abstraction simplifies development and promotes modularity.
3. **Interoperability:** APIs enable different software systems, including applications, libraries, or services, to work together seamlessly. They facilitate interoperability by defining a common interface that developers can implement in their software.
4. **Web APIs:** Many modern APIs are web APIs, which means they are accessed over the internet using standard web protocols such as HTTP. Web APIs are commonly used for integrating web services, cloud services, and various online platforms.
5. **REST and GraphQL:** REST (Representational State Transfer) and GraphQL are two common architectural styles for designing APIs. RESTful APIs use standard HTTP methods (GET, POST, PUT, DELETE) to perform operations, while GraphQL allows clients to request only the data they need.
6. **API Documentation:** API documentation is crucial for developers to understand how to use an API. It provides information on available endpoints, request and response formats, authentication methods, and other relevant details.
7. **Authentication:** APIs often require authentication to ensure that only authorized users or applications can access their functionality. Common authentication methods include API keys, OAuth, and token-based authentication.  
     
     
     
    **what is Anthos  
   Anthos provides platform administrators a single place to monitor and manage their landscape including policy control and marketplace access   
   this reduces person hours in management enforcement discovery and communication**   
      
   Anthos is a hybrid and multi-cloud platform developed by Google Cloud (GCP). It allows organizations to build, manage, and deploy applications across on-premises data centers, Google Cloud Platform, and other cloud providers. Anthos provides a consistent platform and set of tools for managing applications in diverse environments, promoting flexibility and avoiding vendor lock-in.  
   Key features and components of Anthos include:
8. **Hybrid and Multi-Cloud Management**
   * Anthos allows organizations to manage their applications consistently across different environments, including on-premises data centers, Google Cloud, and other public clouds.
   * It provides a unified control plane for deploying and managing applications, regardless of where they are running.
9. **Kubernetes-based Container Orchestration:**
   * Anthos is built on Kubernetes, an open-source container orchestration platform. This allows organizations to containerize their applications and use Kubernetes to orchestrate, deploy, and scale them.
10. **Anthos Config Management:**
    * Anthos Config Management is a part of Anthos that enables the management of Kubernetes configurations across multiple clusters. It allows organizations to define policies and configurations in a centralized manner.
11. **Service Mesh with Anthos Service Mesh:**
    * Anthos Service Mesh is based on the open-source Istio project and provides a way to connect, manage, and secure microservices in a Kubernetes environment. It helps organizations implement features such as traffic management, security, and observability.
12. **Migrate for Anthos:**
    * Migrate for Anthos is a tool that facilitates the migration of virtual machines (VMs) and applications from on-premises environments or other cloud providers to Google Cloud and Anthos.
13. **Security and Policy Management:**
    * Anthos includes security features such as identity and access management, encryption, and automated security updates. It also provides tools for policy enforcement and compliance across hybrid and multi-cloud environments.
14. **Application Modernization:**
    * Anthos supports modern application development practices by providing tools for containerization, continuous integration and delivery (CI/CD), and other DevOps processes.

**HA and classic interconnect and bandwidth**

In the context of Google Cloud Platform (GCP), "HA" typically refers to High Availability, and "classic interconnect" refers to the earlier version of Dedicated Interconnect. Let's explore both terms and their relationship to bandwidth:

**High Availability (HA):**

**Definition:** High Availability (HA) is a design approach and set of practices that aim to ensure a system or service remains operational and accessible for a high percentage of time.

**Importance:** High Availability is crucial for mission-critical applications and services to minimize downtime, maintain service continuity, and provide a reliable experience to users.

**Strategies:** Achieving High Availability often involves redundant systems, load balancing, failover mechanisms, and other strategies to mitigate the impact of potential failures.

**Classic Interconnect:**

**Definition:** Classic Interconnect, in the context of Google Cloud, refers to the earlier version of Dedicated Interconnect, which is a type of network connection that provides a direct physical link between an on-premises network and Google's network.

**Purpose:** Classic Interconnect allows organizations to establish dedicated and private connections to Google Cloud resources, offering a more consistent and potentially higher-performance network connection compared to internet-based connections.

**Features:** Classic Interconnect provides dedicated bandwidth options, such as 1 Gbps and 10 Gbps, and supports traffic between on-premises networks and resources in a Google Cloud Virtual Private Cloud (VPC).

**Bandwidth:**

**Definition:** Bandwidth refers to the maximum rate of data transfer across a network, usually measured in bits per second (bps), kilobits per second (Kbps), megabits per second (Mbps), or gigabits per second (Gbps).

**Importance:** The available bandwidth is a crucial factor in determining the capacity and performance of a network connection. It affects how quickly data can be transmitted between different points in a network.

**Connection Types and Bandwidth in GCP:**

**Dedicated Interconnect:** In addition to Classic Interconnect, Google Cloud offers Dedicated Interconnect, which is the newer version of the service. Dedicated Interconnect provides higher flexibility and features, including higher bandwidth options (up to 100 Gbps), VLAN attachments for segmentation, and global routing.

**Partner Interconnect:** Partner Interconnect is another option that allows organizations to connect to Google Cloud through supported service providers, offering additional flexibility and connectivity options.

In summary, High Availability is a design principle for ensuring system resilience, Classic Interconnect (or Dedicated Interconnect) provides dedicated and private network connections to Google Cloud, and bandwidth is a critical factor that influences the performance and capacity of network connections in the cloud.

**centralized monitoring**

The concept of centralized monitoring in GCP typically refers to the practice of using a centralized tool or service to monitor and analyze the performance, health, and logs of various resources and services across the entire Google Cloud environment.

Key components and features related to centralized monitoring in GCP include:

1. **Cloud Monitoring (formerly known as Stackdriver Monitoring):** This is a GCP service that provides monitoring, logging, and diagnostics for applications that run on Google Cloud. It allows you to collect and view metrics, set up alerting policies, and create dashboards to visualize the performance of your resources.
2. **Cloud Logging (formerly known as Stackdriver Logging):** This service allows you to store, search, analyze, and monitor logs generated by your applications and infrastructure on Google Cloud. Centralized logging helps in troubleshooting, auditing, and gaining insights into the behavior of your applications.
3. **Alerting Policies:** You can set up alerting policies in Cloud Monitoring to be notified when certain conditions are met. This allows you to proactively address issues before they impact your applications or services.
4. **Dashboards:** You can create custom dashboards in Cloud Monitoring to visualize key metrics and performance indicators for your resources. Dashboards provide a centralized view of your system's health and performance.
5. **Integration with Other GCP Services:** Cloud Monitoring integrates with various GCP services, allowing you to monitor not only virtual machine instances but also services like Google Kubernetes Engine (GKE), Cloud Functions, and more.
6. **Monitoring API:** The Monitoring API allows you to programmatically interact with Cloud Monitoring, enabling automation and integration with other tools or systems.

**how to migrate AWS to GCP and explain workflow of that**

Migrating from Amazon Web Services (AWS) to Google Cloud Platform (GCP) involves careful planning, assessment, and execution to ensure a smooth transition. Below is a general workflow for migrating from AWS to GCP. Keep in mind that the actual process may vary based on the specific applications, services, and resources you are migrating.

### Workflow for AWS to GCP Migration:

#### **1. Assessment and Planning:**

a. **Inventory Analysis:** - Identify all resources in your AWS environment, including virtual machines, databases, storage, networking, and other services.

b. **Dependency Mapping:** - Understand dependencies between different components to plan for a phased migration.

c. **Performance and Cost Analysis:** - Evaluate performance metrics and analyze costs to ensure optimal resource allocation in GCP.

d. **Security and Compliance Review:** - Review security and compliance requirements in both AWS and GCP, ensuring a smooth transition without compromising security.

#### **2. Setting Up GCP Environment:**

a. **Create a GCP Project:** - Set up a new GCP project to host your migrated resources.

b. **Configure Networking:** - Establish network connectivity in GCP, including VPCs, subnets, and VPNs if needed.

c. **Identity and Access Management (IAM):** - Configure IAM policies to manage access control in GCP.

#### **3. Data Migration:**

a. **Compute Resources:** - Replicate virtual machines and configurations from AWS to GCP, ensuring compatibility.

b. **Data Transfer:** - Migrate data from AWS S3 to Google Cloud Storage, using tools like **gsutil** or third-party solutions.

c. **Database Migration:** - Migrate databases using services like Cloud SQL (for relational databases) or other GCP-compatible databases.

#### **4. Application Migration:**

a. **Containerization (Optional):** - If applicable, containerize applications using Docker and migrate them to Google Kubernetes Engine (GKE).

b. **App Engine or Compute Engine:** - Choose the appropriate compute service in GCP (e.g., Google Compute Engine or App Engine) for hosting applications.

#### **5. Testing:**

a. **Validation and Testing:** - Conduct thorough testing to ensure that applications and services work as expected in the GCP environment.

b. **User Acceptance Testing (UAT):** - Involve end-users in UAT to verify that the migrated applications meet their requirements.

#### **6. Optimization and Monitoring:**

a. **Performance Optimization:** - Optimize resource configurations and performance based on monitoring data.

b. **Set Up Monitoring and Alerting:** - Configure monitoring and alerting in GCP using Cloud Monitoring to track the health and performance of your resources.

#### **7. Final Data Sync and Cutover:**

a. **Incremental Data Sync:** - Perform a final data synchronization to ensure that the latest changes are migrated.

b. **Cutover:** - Plan a cutover window to switch traffic and operations from AWS to GCP.

#### **8. Post-Migration Validation:**

a. **Validation Checks:** - Validate that all resources and services are running smoothly in the GCP environment.

b. **Documentation:** - Update documentation, including configurations, networking details, and operational procedures.

#### **9. Decommissioning:**

a. **AWS Resource Cleanup:** - Decommission and de-provision resources in AWS that are no longer needed.

b. **Billing Analysis:** - Review AWS billing to ensure all unnecessary resources are terminated.

#### **10. Continuous Improvement:**

a. **Monitoring and Optimization:** - Continuously monitor and optimize the GCP environment based on performance and cost metrics.

b. **Feedback and Documentation:** - Gather feedback from users and operations teams, and update documentation accordingly.

### Considerations and Tips:

* **Downtime Mitigation:** Plan for minimal downtime during the migration by using strategies like blue-green deployments or phased migrations.
* **Data Transfer Tools:** Utilize GCP's data transfer tools or third-party solutions for efficient and secure data migration.
* **Compliance and Security:** Ensure compliance with regulatory requirements and implement security best practices during migration.
* **Backup and Rollback Plan:** Have a backup plan and rollback strategy in case any issues arise during the migration process.
* **Communication:** Keep stakeholders informed throughout the migration process to manage expectations and address concerns.

**how to unlock terraform State file**

Terraform state files are used to store the state of your infrastructure managed by Terraform. These files contain information about the resources that Terraform manages, and they are critical to maintaining and updating your infrastructure. By default, state files are stored locally or remotely, depending on your configuration.

If you find yourself in a situation where you need to "unlock" the Terraform state, it usually means that Terraform is in a locked state, preventing certain operations due to potential conflicts. This often occurs if a previous Terraform operation was interrupted or if the state is being accessed by another process.

Here are steps to potentially resolve or unlock the Terraform state:

**1. Check State Lock:**

Before attempting any actions, you should understand why the state is locked. Run the following command to check the state lock:

terraform state list

If the state is locked, Terraform will display a message indicating that it's locked by a particular process.

**2. Determine the Cause:**

* **Check for Running Operations:** Use the **terraform plan** or **terraform apply** commands to check if there are ongoing operations. If there are, wait for them to complete.
* **Check for Other Locks:** Ensure that no other processes or Terraform executions are holding locks on the state file.

**3. Unlock the State:**

If you are sure that no other process is holding the lock and there are no ongoing operations, you can attempt to manually unlock the state:

terraform force-unlock <LOCK\_ID>

Replace **<LOCK\_ID>** with the lock ID displayed when you ran **terraform state list**.

**Example:** terraform force-unlock my-lock-id

Be cautious with the **force-unlock** command, as it should only be used when you are certain that no other processes are actively using the state.

**4. Retry the Operation:**

After unlocking the state, retry the Terraform operation that you were attempting (e.g., **terraform apply**).

**Important Considerations:**

* **Concurrency:** Avoid running multiple Terraform commands simultaneously, especially on the same state file.
* **Shared State:** If using remote state (e.g., with Terraform Cloud, AWS S3, or another backend), ensure that multiple users or processes are not concurrently trying to modify the state.
* **Backup:** Before using **force-unlock**, consider creating a backup of your Terraform state file to avoid accidental data loss.
* **Team Communication:** If working in a team, communicate with your team to ensure that no one else is actively modifying the state.

**connectivity in GCP**

GCP provides a range of networking features and services that allow you to establish and manage connectivity within and between different components of your infrastructure. Here are some key aspects of connectivity in GCP:

**1. VPC (Virtual Private Cloud):**

* **Definition:** VPC is a virtual network dedicated to your GCP project. It provides an isolated and secure environment where you can deploy your resources.
* **Connectivity within VPC:**
  + Resources within the same VPC can communicate with each other directly.
  + Subnets within a VPC allow you to organize resources and control access.

**2. Interconnecting Networks:**

* **VPC Peering:**
  + VPC peering enables connectivity between different VPCs within the same organization.
  + Peered VPCs can communicate with each other as if they were part of the same network.
* **Shared VPC:**
  + Shared VPC allows resources to be shared across multiple projects, enabling a common networking infrastructure.

**3. VPN (Virtual Private Network) and Interconnect:**

* **Cloud VPN:**
  + Cloud VPN allows you to securely connect your on-premises network to your GCP VPC over the public internet.
* **Cloud Interconnect:**
  + Cloud Interconnect provides dedicated, private connections between your on-premises network and Google's network.

**4. Load Balancing:**

* **HTTP(S) Load Balancing:**
  + Distributes incoming HTTP and HTTPS traffic across multiple backend instances.
  + Provides high availability and scalability for web applications.
* **Network Load Balancing:**
  + Distributes TCP/UDP traffic among multiple backend instances.
  + Suitable for non-HTTP(S) applications.

**5. Firewall Rules:**

* **Firewall Rules:**
  + GCP allows you to define firewall rules to control incoming and outgoing traffic to and from your resources.
  + Helps you define network access policies based on source and destination IP addresses, ports, and protocols.

**6. Cloud DNS:**

* **Cloud DNS:**
  + Google Cloud DNS is a scalable, reliable, and managed Domain Name System (DNS) service.
  + Helps you manage the DNS records for your domains.

**7. Network Monitoring and Logging:**

* **VPC Flow Logs:**
  + VPC Flow Logs capture information about network flows within a VPC, helping you analyze and troubleshoot network traffic.
* **Stackdriver Logging and Monitoring:**
  + Stackdriver provides logging and monitoring services for your resources, allowing you to gain insights into the performance and health of your infrastructure.

**8. Anthos Service Mesh:**

* **Anthos Service Mesh:**
  + Anthos Service Mesh (based on Istio) helps manage and secure microservices-based applications.
  + Provides features like traffic management, security, and observability for services.

**shared VPC and transitive peering and VPC peering**

Shared VPC, VPC (Virtual Private Cloud) peering, and transitive peering are networking concepts in Google Cloud Platform (GCP) that allow you to establish and manage network connectivity between different Virtual Private Clouds. Let's explore each concept:

**1. VPC Peering:**

* **Definition:**
  + VPC peering allows two VPCs to be connected directly, enabling VM instances in one VPC to communicate with VM instances in another VPC.
* **Key Points:**
  + VPC peering is non-transitive, meaning if VPC A is peered with VPC B and VPC B is peered with VPC C, then VPC A and VPC C are not automatically connected.
  + Each VPC maintains its own separate routing table.
* **Use Cases:**
  + Connecting resources in different VPCs within the same organization.
  + Collaboration between different teams or projects.

**2. Shared VPC:**

* **Definition:**
  + Shared VPC (Shared Virtual Private Cloud) allows a host project to share its VPC network resources with other service projects.
* **Key Points:**
  + The host project owns the VPC and defines the network structure.
  + Service projects can use subnets from the shared VPC.
  + IAM (Identity and Access Management) controls access to the shared VPC.
* **Use Cases:**
  + Centralized management of networking resources by an organization.
  + Enabling multiple projects to use a common networking infrastructure.

**3. Transitive Peering:**

* **Definition:**
  + Transitive peering refers to the ability to extend VPC peering relationships beyond a direct connection, creating a "chain" of VPCs that can communicate with each other.
* **Key Points:**
  + Transitive peering is not natively supported in GCP VPC peering.
  + Achieved through custom solutions or third-party tools.
  + Allows communication between VM instances in different VPCs through a chain of peered connections.
* **Use Cases:**
  + Enabling communication between multiple VPCs in a more complex, interconnected architecture.
  + Facilitating communication in scenarios where direct VPC peering is not practical.

**Summary:**

* **VPC Peering:** Direct connection between two VPCs, non-transitive, each VPC maintains separate routing tables.
* **Shared VPC:** Centralized VPC management in a host project, allows service projects to use subnets, IAM controls access.
* **Transitive Peering:** Not natively supported in GCP VPC peering, achieved through custom solutions, allows chaining VPC peering connections.

**how to use one folder resource in another folder in terraform**

Step 1: Organize Your Project Structure

Assume you have the following directory structure:

# project/

|-- main.tf

|-- variables.tf

|-- outputs.tf

|-- modules/

| |-- module1/

| |-- main.tf

| |-- variables.tf

| |-- outputs.tf

|-- resources/

|-- resource1.tf

|-- resource2.tf

project/: The main project directory.

project/main.tf: Main configuration file.

project/variables.tf: Variable definitions.

project/outputs.tf: Output definitions.

project/modules/: Directory to store modules.

project/modules/module1/: Module directory.

project/modules/module1/main.tf: Module configuration file.

project/modules/module1/variables.tf: Module variable definitions.

project/modules/module1/outputs.tf: Module output definitions.

project/resources/: Directory to store resources.

project/resources/resource1.tf: Resource configuration file.

project/resources/resource2.tf: Another resource configuration file.

Step 2: Create a Module

Create a module in the modules directory (project/modules/module1). This module can encapsulate the resources you want to reuse. For example:

// modules/module1/main.tf

variable "param1" {

type = string

default = "default\_value"

}

resource "example\_resource" "example" {

param1 = var.param1

# ... other resource configurations ...

}

output "example\_output" {

value = example\_resource.example.output\_value

}

**Step 3: Use the Module in the Main Configuration**

Reference the module in your main.tf file in the main project directory:

// main.tf

provider "aws" {

region = "us-east-1"

}

module "my\_module" {

source = "./modules/module1"

param1 = "custom\_value"

}

resource "another\_resource" "another" {

# ... configuration for another resource ...

}

output "module\_output" {

value = module.my\_module.example\_output

}

Step 4: Run Terraform Commands

Now, you can navigate to the main project directory (project/) and run Terraform commands:

terraform init

terraform apply  
  
  
  
**Kubernetes features**

Kubernetes is a powerful and widely used container orchestration platform that automates the deployment, scaling, and management of containerized applications. It offers a variety of features that contribute to its popularity and effectiveness in orchestrating container workloads. Here are some key features of Kubernetes:

1. **Container Orchestration:**
   * **Pods:** The smallest deployable units in Kubernetes, representing one or more containers that share storage and network resources.
   * **Deployments:** Enable declarative updates to applications, managing the deployment and scaling of replicas.
   * **ReplicaSets:** Maintain a stable set of replica pods running at any given time.
2. **Automatic Load Balancing:**
   * Kubernetes provides built-in load balancing for services, distributing traffic across multiple pods.
3. **Scaling:**
   * **Horizontal Pod Autoscaling (HPA):** Automatically adjusts the number of pod replicas in response to changes in CPU utilization or other custom metrics.
   * **Cluster Autoscaler:** Automatically adjusts the number of nodes in a cluster based on resource requirements.
4. **Service Discovery and Load Balancing:**
   * Kubernetes services provide a stable endpoint for accessing a set of pods, allowing easy discovery and load balancing.
5. **Rolling Updates and Rollbacks:**
   * Kubernetes supports rolling updates, allowing you to update your application without downtime, and rollbacks in case of issues.
6. **Self-healing:**
   * Kubernetes restarts containers that fail, replaces and reschedules containers that don't respond to health checks, and kills containers that violate pod constraints.
7. **Secrets and Configuration Management:**
   * Kubernetes allows the management of sensitive information and configuration parameters using Secrets and ConfigMaps.
8. **Storage Orchestration:**
   * Kubernetes supports a variety of storage solutions and provides mechanisms for exposing storage to pods, such as Persistent Volumes (PVs) and Persistent Volume Claims (PVCs).
9. **Multi-Environment Deployment:**
   * Kubernetes is designed to be cloud-agnostic, allowing deployment on a variety of cloud providers or on-premises infrastructure.
10. **Resource Management:**
    * Kubernetes allows you to specify resource requirements and limits for pods, ensuring fair resource distribution among applications.
11. **Role-Based Access Control (RBAC):**
    * Kubernetes provides RBAC for controlling access to the Kubernetes API and resources based on roles and role bindings.
12. **Network Policies:**
    * Define how groups of pods are allowed to communicate with each other and other network endpoints.
13. **Logging and Monitoring:**
    * Kubernetes integrates with various logging and monitoring solutions, and it exposes metrics for performance monitoring.
14. **Extensibility:**
    * Kubernetes is highly extensible through the use of custom resources and controllers, allowing the addition of new functionalities.
15. **Community and Ecosystem:**
    * Kubernetes has a large and active open-source community, contributing to a rich ecosystem of tools, plugins, and extensions.

These features collectively make Kubernetes a robust platform for

**difference between normal firewall rule and GKE firewall rules**

### 1. **Compute Engine Firewall Rules:**

* **Scope:** Compute Engine firewall rules are network-level rules that apply to the entire Google Cloud project.
* **Target:** They are typically used to control traffic to and from Compute Engine instances, load balancers, and other resources within the Google Cloud project.
* **Application:** Compute Engine firewall rules are not specific to Kubernetes and are used at the infrastructure level.
* **Implementation:** Configured using Google Cloud Console, gcloud command-line tool, or the Compute Engine API.
* **Use Case:** These rules are well-suited for traditional VM-based applications and services running in the Google Cloud project.

### 2. **GKE Network Policies:**

* **Scope:** GKE Network Policies are specific to Kubernetes clusters and are used to control communication between pods within the cluster.
* **Target:** They allow fine-grained control over the traffic between pods based on labels, namespaces, and other Kubernetes-specific attributes.
* **Application:** Network Policies are designed to secure communication between pods within a GKE cluster, offering a layer of security at the application level.
* **Implementation:** Network Policies are implemented and configured at the Kubernetes layer and require support from the underlying network plugin (e.g., Calico, Cilium).
* **Use Case:** Network Policies are ideal for microservices architectures where you want to control communication between different parts of your application running in the same cluster.

### Summary:

* **Compute Engine Firewall Rules:** These are broader network-level rules that apply to the entire Google Cloud project. They are not specific to Kubernetes and are used to control traffic between various resources in the project.
* **GKE Network Policies:** These are Kubernetes-specific and provide granular control over communication between pods within a GKE cluster. They are more focused on securing communication within the microservices architecture.

**dedicated connect in GCP**

For connectivity between on-premises environments and Google Cloud, some relevant services and features include:

1. **Cloud Interconnect:**
   * Google Cloud Interconnect provides connectivity between your on-premises network and Google Cloud. It includes Dedicated Interconnect and Partner Interconnect options.
2. **Cloud VPN:**
   * Cloud VPN allows you to securely connect your on-premises network to your Google Cloud Virtual Private Cloud (VPC) network through an IPsec VPN connection.
3. **Direct Peering:**
   * Google's Direct Peering allows you to establish direct connectivity between your on-premises network and Google's network at various peering locations.
4. **Carrier Peering:**
   * Carrier Peering enables service providers to offer connectivity services to their customers with dedicated connections to Google Cloud.
5. **Network Service Tiers:**
   * Google Cloud offers Premium and Standard Network Service Tiers, allowing you to choose the level of network service quality for your traffic between on-premises and the cloud.

When implementing connectivity between your on-premises environment and Google Cloud, it's essential to review the documentation, understand the requirements, and choose the appropriate networking solution based on your specific use case and performance requirements.

 **what is VPN and what kinds of VPN in GCP and difference**A Virtual Private Network (VPN) is a technology that allows the creation of a secure and encrypted connection between two or more networks over an insecure network, such as the internet. VPNs are commonly used to establish secure communication channels between remote sites, allowing secure data transmission over public networks.

In Google Cloud Platform (GCP), there are two main types of VPNs: Cloud VPN and HA VPN. Both types provide secure connections between your on-premises network and your GCP Virtual Private Cloud (VPC) network.

### 1. **Cloud VPN:**

* **Description:**
  + Cloud VPN is a simple and cost-effective solution for creating secure connections over the internet.
* It uses industry-standard IPsec protocols to create encrypted connections between your on-premises network and your GCP VPC network.
* **Features:**
  + Supports both policy-based and route-based VPNs.
  + Suitable for low to moderate bandwidth requirements.
  + Easier to set up and configure.

### 2. **HA VPN (High Availability VPN):**

* **Description:**
  + HA VPN is designed for higher availability and reliability.
  + It utilizes two tunnels (each with a different external IP address) to ensure redundancy and failover in case one tunnel becomes unavailable.
* **Features:**
  + Provides higher availability and reliability compared to Cloud VPN.
  + Suitable for applications with higher availability requirements.
  + Utilizes Border Gateway Protocol (BGP) for dynamic routing.

### **Differences:**

Here are some key differences between Cloud VPN and HA VPN in GCP:

1. **Redundancy:**
   * **Cloud VPN:** Uses a single tunnel for connectivity.
   * **HA VPN:** Utilizes two tunnels for redundancy and failover.
2. **Availability:**
   * **Cloud VPN:** Suitable for standard use cases with lower availability requirements.
   * **HA VPN:** Provides higher availability and is recommended for applications with more stringent availability needs.
3. **Configuration Complexity:**
   * **Cloud VPN:** Generally easier to set up and configure.
   * **HA VPN:** Involves additional configuration for the redundant tunnel and BGP setup.
4. **Cost:**
   * **Cloud VPN:** Generally more cost-effective for lower bandwidth requirements.
   * **HA VPN:** May incur higher costs due to the redundancy and increased availability.

**what is BGP**

BGP, which stands for Border Gateway Protocol, is a standardized exterior gateway protocol used to exchange routing and reachability information among different autonomous systems (ASes) on the Internet. An autonomous system is a collection of IP networks and routers under the control of a single organization that presents a common routing policy to the internet.

Key features and characteristics of BGP include:

1. **Path Vector Protocol:**
   * BGP is a path vector protocol, meaning it makes decisions based on the path information along which data travels from source to destination. Each BGP router maintains a table of routes with associated attributes and selects the best path based on these attributes.
2. **Path Attributes:**
   * BGP routers exchange information about the reachability of networks along with various attributes that describe the path characteristics. Examples of path attributes include the Autonomous System Path (AS\_PATH), Next Hop, and various route policies.
3. **Inter-AS Routing:**
   * BGP is primarily used for routing between different autonomous systems. It is designed to provide a standardized way for different organizations or service providers to exchange routing information at the edge of their networks.
4. **Policy-Based Routing:**
   * BGP allows network administrators to implement policy-based routing decisions. This means that decisions about which routes to advertise or accept can be based on various policies and criteria, providing flexibility and control over routing decisions.
5. **Reliability and Stability:**
   * BGP is designed to be robust and provide stable routing in the face of changes. It uses mechanisms such as Route Refresh and graceful restart to minimize disruptions during updates.
6. **Multiprotocol Support:**
   * BGP supports routing for multiple network layer protocols, including IPv4 and IPv6. This makes it versatile in supporting different addressing schemes.
7. **External BGP (eBGP) and Internal BGP (iBGP):**
   * BGP can be used within an autonomous system (iBGP) for internal routing and between different autonomous systems (eBGP) for external routing. iBGP is used to distribute BGP information within an AS, and eBGP is used to exchange information between ASes.
8. **Prefix Aggregation:**
   * BGP supports the aggregation of IP prefixes, helping to reduce the size of the global routing table and improving scalability.

BGP is a critical component of the global Internet routing infrastructure.

**how can we achieve transitive peering**  
  
Transitive peering refers to the ability to route traffic between two endpoints through an intermediate device or network. In the context of networking, transitive peering is often associated with the BGP (Border Gateway Protocol) and the ability to route traffic through a third-party network.

Here are some general steps to achieve transitive peering using BGP:

### 1. **Understand the Basics of BGP:**

* Ensure a good understanding of BGP, including its operation, attributes (such as AS\_PATH), and how BGP routers exchange routing information.

### 2. **Establish BGP Peerings:**

* Establish BGP peerings between the relevant routers in each autonomous system (AS). This includes configuring eBGP peerings between routers in different ASes and iBGP peerings within the same AS.

### 3. **AS\_PATH Attribute:**

* Understand the AS\_PATH attribute in BGP. AS\_PATH is a BGP attribute that shows the AS numbers through which the route has passed. This attribute plays a crucial role in preventing routing loops.

### 4. **Configure Route Reflectors (Optional):**

* In larger BGP deployments, especially within the same AS, consider using route reflectors to simplify iBGP peering. Route reflectors help in avoiding the full mesh iBGP requirement.

### 5. **Manipulate AS\_PATH for Transitivity:**

* To achieve transitive peering, you may need to manipulate the AS\_PATH attribute to control the path that BGP routes take. This manipulation can involve prepending or removing AS numbers from the AS\_PATH.

### 6. **Use BGP Communities:**

* BGP communities are tags that can be attached to BGP routes to influence their behavior. Leveraging BGP communities can help in implementing policies for transitive peering.

### 7. **Implement Route Filtering and Policies:**

* Implement route filtering and policies on BGP routers to control which routes are advertised to or accepted from neighboring routers. This can help in ensuring that only specific routes are subject to transitive peering.

### 8. **Carefully Plan and Test:**

* Plan the transitive peering setup carefully, considering the implications on routing, traffic flow, and potential issues like routing loops. Test the configuration in a controlled environment before deploying in production.

### 9. **Monitor and Troubleshoot:**

* Implement monitoring tools to observe the behavior of BGP routes and peering relationships. Set up alerts for unexpected changes. Be prepared to troubleshoot and diagnose issues that may arise.

**load balancer and kinds of load balancer**

A load balancer is a device or software application that distributes incoming network traffic across multiple servers or resources to ensure that no single server bears too much load, thus improving the reliability, availability, and scalability of applications or services. Load balancing can be implemented at different layers of the networking stack, including application layer, transport layer, and network layer.

Here are common types of load balancers based on their deployment location and the OSI model layers at which they operate:

### 1. **Layer 4 (Transport Layer) Load Balancer:**

* Operates at the transport layer (TCP/UDP) of the OSI model.
* Distributes traffic based on network-level information (IP addresses and ports).
* Examples include software-based load balancers, like HAProxy, and hardware load balancers.

### 2. **Layer 7 (Application Layer) Load Balancer:**

* Operates at the application layer of the OSI model.
* Makes routing decisions based on content, such as HTTP headers, cookies, or application-level data.
* Ideal for handling application-specific tasks and optimizing traffic based on application requirements.
* Examples include NGINX, HAProxy, and cloud-based load balancers.

### 3. **Global Server Load Balancer (GSLB):**

* Distributes traffic across multiple data centers or geographical locations.
* Helps optimize performance and provide high availability on a global scale.
* Makes routing decisions based on factors such as proximity, server health, and load.
* Examples include F5 BIG-IP Global Traffic Manager (GTM) and cloud-based global load balancers.

### 4. **Hardware Load Balancer:**

* A physical device dedicated to load balancing.
* Often provides specialized hardware for high-performance and efficient traffic distribution.
* Examples include appliances from vendors like F5 Networks, Citrix, and others.

### 5. **Software Load Balancer:**

* A load balancer implemented as a software application.
* Can be deployed on general-purpose hardware or virtualized environments.
* Examples include NGINX, HAProxy, and software-based load balancers provided by cloud providers.

### 6. **Cloud Load Balancer:**

* Load balancing services provided by cloud providers.
* Optimized for scalability and ease of use in cloud environments.
* Examples include Google Cloud Load Balancer, AWS Elastic Load Balancing (ELB), and Azure Load Balancer.

### 7. **Reverse Proxy:**

* While not strictly a load balancer, a reverse proxy often includes load balancing capabilities.
* Handles incoming requests on behalf of servers and distributes the requests to the appropriate server.
* Examples include NGINX, Apache HTTP Server (with mod\_proxy), and others.

### 8. **DNS Load Balancing:**

* Distributes traffic by manipulating DNS responses to direct clients to different IP addresses.
* Examples include Global Server Load Balancers (GSLB) that operate at the DNS layer.

**how can we prevent Ip address in load balancer**

If you want to prevent specific IP addresses from accessing your application through a load balancer, you can typically achieve this through access control mechanisms provided by the load balancer itself or by implementing additional security measures in your application or infrastructure. The exact method may vary depending on the type of load balancer you are using.

Here are some general approaches:

### 1. **Load Balancer ACLs (Access Control Lists):**

* Many load balancers provide features to define Access Control Lists (ACLs) to restrict access based on IP addresses.
* In your load balancer configuration, you can set up rules to deny or allow traffic from specific IP addresses or ranges.
* Example (NGINX configuration):

nginxCopy code

location / { deny 192.168.1.1; allow 10.0.0.0/24; allow 192.168.1.0/24; deny all; }

### 2. **Cloud Load Balancers:**

* If you are using a cloud-based load balancer (e.g., AWS Elastic Load Balancing, Google Cloud Load Balancer, Azure Load Balancer), check the specific documentation for access control features.
* Cloud providers often provide security groups or network security policies to control incoming traffic.

### 3. **Web Application Firewall (WAF):**

* Consider implementing a Web Application Firewall (WAF) in front of your load balancer.
* WAFs can provide additional security features, including IP whitelisting or blacklisting.

### 4. **Network Security Groups (NSGs):**

* If you are using a virtualized or cloud-based load balancer, check if the underlying infrastructure supports Network Security Groups (NSGs).
* NSGs allow you to define rules at the network level to control traffic.

### 5. **Application Layer Filtering:**

* Implement application layer filtering within your application code.
* In the application itself, you can check the incoming requests and reject those coming from undesired IP addresses.

### 6. **Firewall Rules:**

* Implement firewall rules on the servers behind the load balancer.
* Configure firewall rules on each server to block traffic from specific IP addresses.

### 7. **DDoS Protection Services:**

* Consider using Distributed Denial of Service (DDoS) protection services, which often include IP filtering capabilities.
* These services can help mitigate malicious traffic and block access from unwanted IP addresses.

### Note:

* Ensure that you carefully test any access control measures to avoid unintentional blocking of legitimate traffic.
* Keep in mind that IP address blocking is a basic security measure, and for more advanced security, you may want to consider implementing additional authentication mechanisms and using security best practices.

**how many kinds of clusters in GKE and difference between them**

### 1. **Standard Cluster:**

* **Description:**
  + Standard clusters are the default and most commonly used type of GKE cluster.
  + They provide a balanced and managed Kubernetes environment with features like automatic upgrades, node auto-repair, and integrated monitoring.
* **Characteristics:**
  + Automatically provisioned and managed by GKE.
  + Supports both zonal and regional configurations.
  + Ideal for most general-purpose workloads.

### 2. **Autopilot Cluster:**

* **Description:**
  + Autopilot clusters are a newer and more streamlined type of GKE cluster introduced to simplify cluster management.
  + They abstract away many of the traditional cluster management tasks, allowing developers to focus more on application development.
* **Characteristics:**
  + Automatically scales nodes based on workload requirements.
  + Automatically manages resources, such as node pools, for optimal efficiency.
  + Simplifies management with a focus on automation.
  + Ideal for developers who want a hands-off approach to cluster management.

### Key Differences:

1. **Node Management:**
   * **Standard Cluster:**
     + Users need to manage node pools, including scaling and upgrades.
     + Node pools can be customized for specific configurations.
   * **Autopilot Cluster:**
     + Node management is abstracted away, and users do not need to create or manage individual node pools.
     + Resources are automatically provisioned and managed by GKE.
2. **Resource Efficiency:**
   * **Standard Cluster:**
     + Users have more control over node configurations, allowing for fine-tuning based on specific requirements.
   * **Autopilot Cluster:**
     + Designed for resource efficiency with automatic scaling and resource management.
     + Users relinquish some control in favor of simplified management.
3. **Cluster Configuration:**
   * **Standard Cluster:**
     + Users have greater flexibility in customizing node pools and configurations.
     + Suitable for more advanced configurations and use cases.
   * **Autopilot Cluster:**
     + Abstracts away many cluster configurations for simplicity.
     + Focuses on a more opinionated approach to cluster management.
4. **Use Cases:**
   * **Standard Cluster:**
     + Well-suited for a wide range of use cases, especially when users need fine-grained control over cluster configuration.
   * **Autopilot Cluster:**
     + Ideal for developers who prefer a more automated and simplified approach to cluster management.
     + Suitable for workloads with varying resource requirements.

**diff b/w stateless set and deployment**

In Kubernetes, both StatefulSets and Deployments are resource types used to manage applications, but they serve different purposes and are suitable for different types of workloads. Here are the key differences between StatefulSets and Deployments:

### StatefulSet:

1. **Stateful Operations:**
   * **Purpose:** StatefulSets are designed for stateful applications where each pod has a unique and stable network identity.
   * **Pod Naming:** Pods in a StatefulSet are given unique, ordered names, and they maintain the same hostname and network identity throughout their lifecycle.
   * **Persistent Storage:** StatefulSets are commonly used with Persistent Volumes (PVs) to provide persistent storage for stateful applications.
2. **Scaling:**
   * **Scaling:** StatefulSets provide ordered scaling, meaning that pods are scaled up or down in a predictable and ordered manner.
   * **Pod Lifecycle:** Pods are created and scaled in a predictable order, and they maintain a consistent identity even during rescheduling.
3. **Update and Rollback:**
   * **Update Strategy:** StatefulSets allow for rolling updates with a well-defined update strategy, ensuring that only one pod is updated at a time.
   * **Rollback:** Supports easy rollbacks to a previous version in case of issues during updates.
4. **Use Cases:**
   * **Examples:** StatefulSets are commonly used for applications like databases (e.g., MySQL, PostgreSQL), where each pod needs stable network identity and persistent storage.

### Deployment:

1. **Stateless Operations:**
   * **Purpose:** Deployments are designed for stateless applications where each pod is interchangeable and can be easily replaced.
   * **Pod Naming:** Pods in a Deployment do not have stable, predictable names, and they are replaced without maintaining a consistent identity.
2. **Scaling:**
   * **Scaling:** Deployments provide easy and fast scaling by creating or terminating pods based on demand.
   * **Pod Lifecycle:** Pods can be terminated and replaced at any time without any concern for maintaining a consistent identity.
3. **Update and Rollback:**
   * **Update Strategy:** Deployments support rolling updates but focus on quickly replacing pods rather than maintaining a specific order.
   * **Rollback:** Supports easy rollbacks to a previous version in case of issues during updates.
4. **Use Cases:**
   * **Examples:** Deployments are commonly used for stateless applications, such as web servers, APIs, or microservices, where each pod is interchangeable, and there is no reliance on stable network identity.

* **StatefulSets:** Best suited for stateful applications requiring stable network identity and persistent storage. Ordered scaling and updates are key features.
* **Deployments:** Ideal for stateless applications where pods are interchangeable, and scaling and updates can be done quickly. Stateless applications do not rely on maintaining a specific network identity.

**taint and Tolerent**

In Kubernetes, "taints" and "tolerations" are concepts used to control how Pods are scheduled onto nodes in a cluster. These mechanisms provide a way to influence the scheduling decisions based on node characteristics and pod requirements.

**Taints:**

* **Definition:** Taints are applied to nodes, indicating that the node has a certain characteristic or restriction that affects the types of Pods that can be scheduled onto it.
* **Usage:**
  + A node can have one or more taints applied.
  + Taints are represented by key-value pairs, where the key is the taint effect, and the value is an optional parameter.
* **Example:**
  + You can apply a taint to a node using the **kubectl taint** command. For example:

kubectl taint nodes node-1 key=value:NoSchedule

* **Effect:**
  + The taint effect can be one of the following:
    - **NoSchedule**: Pods that do not tolerate the taint will not be scheduled onto the node.
    - **PreferNoSchedule**: The system will try to avoid scheduling non-tolerant Pods, but it is not a strict restriction.
    - **NoExecute**: Existing Pods on the node that do not tolerate the taint will be evicted.

**Tolerations:**

* **Definition:** Tolerations are applied to Pods, indicating that the Pod can tolerate (or ignore) nodes with specific taints.
* **Usage:**
  + A Pod can have multiple tolerations specified in its configuration.
  + Tolerations are represented by key-value pairs, similar to taints.
* **Example:**
  + You can specify tolerations in the Pod YAML. For example:

tolerations: - key: "key" operator: "Equal" value: "value" effect: "NoSchedule"

* **Effect:**
  + The effect in tolerations should match the effect of the corresponding taint. Common effects are:
    - **NoSchedule**: Allows scheduling on nodes with the specified taint.
    - **PreferNoSchedule**: Preferentially allows scheduling on nodes without the specified taint.
    - **NoExecute**: Allows the Pod to remain running on a node even if the node gets tainted with **NoExecute**.

**Use Cases:**

* **Scenario 1: Node with Special Hardware:**
  + **Taint:** Taint nodes with special hardware to prevent non-compatible Pods from being scheduled.
  + **Toleration:** Configure Pods requiring that special hardware with tolerations to be scheduled on those nodes.
* **Scenario 2: Reserved Nodes:**
  + **Taint:** Taint nodes that should be reserved for specific workloads.
  + **Toleration:** Configure Pods of those specific workloads with tolerations to be scheduled on the reserved nodes.

Taints and tolerations provide a flexible mechanism for fine-tuning

**rebase in git**

Rebasing in Git is a way to incorporate changes from one branch into another by moving, combining, or modifying the commit history. It is an alternative to merging and can help create a cleaner and linear history. The **git rebase** command is used for this purpose. Here's a basic overview of how rebasing works:

**Basic Rebasing Workflow:**

1. **Create a Feature Branch:**

git checkout -b feature-branch

1. **Make Commits on the Feature Branch:**

# Make changes and commit git add . git commit -m "Feature A: Implement functionality"

1. **Fetch Changes from the Remote Repository (Optional):**

git fetch origin

1. **Rebase the Feature Branch onto the Latest Changes:**

git rebase origin/main

* + This command takes the commits on your feature branch, detaches them, applies the changes from the specified branch (e.g., **origin/main**), and then reattaches your commits.

1. **Resolve Conflicts (If Any):**
   * If there are conflicts during the rebase, Git will pause and allow you to resolve them before continuing.
   * After resolving conflicts, you can use **git rebase --continue** to continue the rebase.
2. **Push the Rebased Branch:**

git push origin feature-branch

* + Force push may be required if you've rewritten history: **git push --force-with-lease origin feature-branch**

**Common Options:**

* **Interactive Rebasing:**

git rebase -i origin/main

* + Allows you to interactively choose which commits to pick, edit, squash, or drop during the rebase.
* **Skip a Commit During Rebase:**

git rebase --skip

* + Skips the current commit if there are conflicts or issues.
* **Abort a Rebase:**

git rebase --abort

* + Aborts the rebase operation, restoring the branch to its original state.

**Caution:**

* **Avoid Rebasing Shared Branches:**
  + Rebasing can rewrite commit history, so it's generally not recommended on branches that are shared with others.
  + Use rebasing on local branches or branches that are not shared/public.
* **Force Push:**
  + If you've rebased and rewritten history, you may need to force push (**--force** or **--force-with-lease**) to update the remote branch.

**how dynamic lock work**

Dynamic locks, in the context of concurrent programming or multi-threading, typically refer to locks that are acquired and released dynamically at runtime. These locks are not associated with specific variables or resources at compile time but are instead acquired based on the program's runtime conditions. The concept is often tied to the idea of acquiring locks dynamically based on certain conditions or events.

Here's a general explanation of how dynamic locks might work:

1. Conditional Locking:

Dynamic locks are acquired based on conditions or events that occur during program execution.

The decision to acquire a lock is made dynamically, not predetermined at compile time.

2. Usage Scenario:

Consider a scenario where multiple threads are accessing a shared resource, and you want to protect access to that resource using locks.

The decision of which thread gets the lock may depend on dynamic conditions, such as the current state of the program or the resource.

3. Pseudo-Code Example:

python

Copy code

# Pseudo-code example in Python

lock = DynamicLock() # Initialize a dynamic lock

# Thread 1

if some\_condition:

lock.acquire() # Acquire the lock dynamically

# Perform operations on the shared resource

lock.release() # Release the lock

# Thread 2

if another\_condition:

lock.acquire() # Acquire the lock dynamically based on a different condition

# Perform operations on the shared resource

lock.release() # Release the lock

4. Use Cases:

Dynamic locks are useful in situations where the locking strategy needs to adapt based on changing conditions.

They might be used to avoid deadlocks or contention by dynamically adjusting which threads can access a resource at a given time.

5. Implementation:

The implementation of dynamic locks may vary based on the programming language and concurrency framework being used.

It could involve the use of condition variables, semaphores, or other synchronization primitives.

6. Challenges:

Dynamic locking introduces complexities, and developers need to carefully manage the conditions under which locks are acquired and released.

Incorrect use of dynamic locks could lead to race conditions, deadlocks, or other concurrency issues.

Note:

Dynamic locks are a broad concept, and the actual implementation may depend on the specific context and programming environment.  
flow of terraform and git

**what is workload identity**

**anthos and anthos mess**

**prerequisites for migration**

**if we want to migrate 400 VMs to cloud how will u achieve it**

**propogation of GKE**

**how will to connect pods**

**in which language cloud logging store data**  
 **how many types of accounts in GCP**

1. **Google Cloud Account:**
   * This is the main account associated with billing and payments for GCP services. It is used to manage access to resources and services.
2. **Google Cloud Identity Account:**
   * Google Cloud Identity is an identity management service that allows organizations to manage users and groups. Google Cloud Identity accounts can be associated with GCP for user authentication and access control.
3. **Service Account:**
   * Service accounts are used for authenticating applications and services running on GCP. They are associated with a project and can be granted specific roles and permissions.
4. **User Account:**
   * User accounts are associated with individual users who can access and manage GCP resources. These accounts are often linked to a corporate email address.
5. **Organization Account:**
   * An organization account represents an organization on GCP. It is associated with a domain and is used for organizing and managing projects and resources.
6. **Google Workspace Account:**
   * If you are using Google Workspace (formerly G Suite), there might be integration with GCP. Google Workspace accounts can be used for accessing GCP resources and services.
7. **Google Cloud Support Account:**
   * Customers with Google Cloud Support have a separate support account that they can use to manage support-related activities.
8. **Billing Account:**
   * A billing account is associated with the financial aspects of using GCP. It is used to pay for the services consumed by the associated projects.
9. **API Console Project:**
   * Each project in the Google Cloud Console is associated with an API Console Project. This is where you manage APIs, credentials, and other project-specific configurations.

**COMPUTE ENGINE**

1. Google Cloud Compute Engine is an Infrastructure as a Service (IaaS) offering provided by Google Cloud Platform (GCP). It allows users to run virtual machines (VMs) on Google's infrastructure. Compute Engine provides scalable and flexible computing resources, enabling users to deploy and manage virtual machines in the cloud.  
     
   **Virtual Machines (VMs):** Users can create and manage virtual machines with various configurations, such as the choice of operating system, CPU, memory, and storage.
2. **Scalability:** Compute Engine allows you to scale your infrastructure up or down based on your requirements. You can add or remove virtual machine instances to match the demand of your applications.
3. **Customization:** Users have control over the virtual machine instances, including the ability to customize the machine types, add GPUs, and configure networking options.
4. **Preemptible VMs:** These are lower-cost, short-lived instances suitable for fault-tolerant and batch processing workloads. However, they can be preempted by Google with very little notice.
5. **Persistent Disks:** Compute Engine provides durable and high-performance block storage options for the virtual machines. Persistent disks can be attached to VM instances to store data.
6. **Load Balancing:** Google Compute Engine includes load balancing services to distribute incoming network traffic across multiple instances to ensure high availability and reliability.
7. **Networking:** Compute Engine offers a variety of networking features, including virtual private cloud (VPC), firewalls, and the ability to create custom network topologies.
8. **Integration with Other Google Cloud Services:** Compute Engine can be integrated with other Google Cloud services, such as Google Cloud Storage, BigQuery, and others, to build comprehensive cloud-based applications.

**what is GKE**  
It is a managed Kubernetes service provided by Google Cloud Platform (GCP) for orchestrating and deploying containerized applications. Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Key features of Google Kubernetes Engine (GKE) include:

1. **Managed Kubernetes:** GKE abstracts the complexity of setting up and managing a Kubernetes cluster. Google takes care of the underlying infrastructure, including the master nodes, and provides a simplified interface for users to deploy and manage containerized applications.
2. **Containerized Applications:** GKE is designed to work seamlessly with Docker containers. Users can package their applications and dependencies into containers, which can then be deployed and managed by Kubernetes.
3. **Scalability:** GKE allows for easy scaling of applications by adding or removing container instances based on demand. Kubernetes can automatically manage the scaling process, ensuring optimal resource utilization.
4. **Automated Updates:** GKE automatically handles Kubernetes version upgrades, ensuring that your clusters are running the latest stable releases without manual intervention. This helps in maintaining security and reliability.
5. **Integration with Google Cloud Services:** GKE integrates with other Google Cloud services, such as Cloud Storage, BigQuery, and Pub/Sub. This allows for seamless integration of containerized applications with various cloud services.
6. **Multi-Region and Multi-Zone Deployments:** GKE supports deployment across multiple regions and availability zones, providing high availability and reliability for applications.
7. **Security Features:** GKE includes security features such as node auto-repair, which automatically replaces failed nodes, and Identity and Access Management (IAM) integration for fine-grained access control.
8. **Monitoring and Logging:** GKE integrates with Google Cloud Monitoring and Logging, allowing users to monitor the health and performance of their applications, as well as access logs for troubleshooting.

**Cloud Functions** is a serverless compute service provided by Google Cloud Platform (GCP). It allows developers to write, deploy, and run single-purpose functions without the need to provision or manage servers. Cloud Functions automatically scales based on demand, and users are billed only for the actual compute resources consumed during the execution of functions.

Key features of Cloud Functions include:

1. **Event-Driven:** Cloud Functions are typically triggered by events that occur in other GCP services, such as changes in Cloud Storage, new messages in Pub/Sub, HTTP requests, or changes in Firestore. This event-driven model enables developers to build serverless applications that respond to specific events.
2. **Supported Runtimes:** Cloud Functions supports various programming languages, including Node.js, Python, Go, Java, .NET, and more. This flexibility allows developers to choose the language that best suits their needs.
3. **Automatic Scaling:** Cloud Functions automatically scales based on the number of incoming requests. Each function instance is independent, allowing for efficient scaling up or down to handle varying workloads.
4. **Zero Server Management:** With serverless computing, developers don't need to worry about server provisioning, maintenance, or scaling. Google Cloud takes care of the underlying infrastructure, allowing developers to focus solely on writing code.
5. **Stateless Execution:** Cloud Functions are designed to be stateless, meaning they don't retain information between invocations. However, data can be stored in external services like databases or object storage.
6. **Built-in Monitoring and Logging:** Cloud Functions integrates with Google Cloud's monitoring and logging services, providing insights into function performance, errors, and execution logs.
7. **Billing based on Execution:** Users are billed based on the actual compute resources consumed during the execution of functions. This pay-as-you-go model is cost-effective, especially for sporadically or infrequently triggered functions.
8. **Integration with GCP Services:** Cloud Functions easily integrates with other Google Cloud services, allowing developers to build comprehensive serverless applications that leverage the capabilities of various GCP services

**App Engine**  
Google App Engine is a fully managed Platform as a Service (PaaS) offering provided by Google Cloud Platform (GCP). It allows developers to build and deploy applications without managing the underlying infrastructure. App Engine supports multiple programming languages and provides an environment for developing scalable and reliable web applications.

Key features of Google App Engine include:

1. **Fully Managed:** App Engine abstracts away the infrastructure management, allowing developers to focus on building and deploying applications. Google takes care of tasks such as scaling, load balancing, and patching.
2. **Auto-Scaling:** App Engine automatically scales your application based on demand. It can handle varying levels of traffic and adjust resources dynamically to ensure optimal performance and cost efficiency.
3. **Multi-Language Support:** App Engine supports multiple programming languages, including Python, Java, Node.js, Go, Ruby, and PHP. This flexibility allows developers to use the language that best fits their application requirements.
4. **Built-in Services:** App Engine provides several built-in services, such as NoSQL data storage (Datastore), a fully managed relational database (Cloud SQL), caching (Memorystore), task queues, and more. These services simplify the development process by offering commonly used functionalities without requiring additional configuration.
5. **Versioning and Rollback:** App Engine allows you to deploy multiple versions of your application simultaneously. This facilitates easy rollbacks in case of issues with a new deployment. Each version can be assigned a unique URL, making it easy to test and switch between versions.
6. **Integrated Security:** App Engine applications run within a secure and isolated environment. Google Cloud provides security features, such as encryption in transit and at rest, Identity and Access Management (IAM) controls, and regular security updates.
7. **Traffic Splitting:** App Engine allows you to split traffic between different versions of your application. This can be useful for A/B testing, gradual deployments, or canary releases.
8. **Monitoring and Logging:** App Engine integrates with Google Cloud's monitoring and logging services, providing insights into application performance, error tracking, and usage metrics.

APIs (Application Programming Interfaces) and services are fundamental concepts in software development, especially in the context of building and integrating applications.

**APIs (Application Programming Interfaces**):

An API is a set of rules and protocols that allows one software application to interact with another. It defines the methods and data formats that applications can use to communicate with each other. APIs can be used for various purposes, including accessing web-based services, databases, or integrating with third-party software.

**Google Cloud Storage.**   
It offers scalable and durable object storage suitable for various use cases, ranging from simple data storage to serving static content for web applications. Here are some key features and aspects of Google Cloud Storage:

1. **Object Storage:** Google Cloud Storage is designed for storing and retrieving any amount of data in the form of objects. Objects can be files, images, videos, and more. Each object is stored in a bucket, which is a container for objects within Google Cloud Storage.
2. **Scalability:** Google Cloud Storage is highly scalable, allowing you to store and retrieve large amounts of data. It automatically scales to accommodate the size and volume of your data, and you can easily increase your storage capacity as needed.
3. **Durability and Redundancy:** Google Cloud Storage is designed for high durability. It automatically replicates your data across multiple locations, providing redundancy to ensure that your data is resilient to hardware failures or other issues.
4. **Access Control:** Google Cloud Storage provides fine-grained access control, allowing you to manage who can access your data and what level of permissions they have. This is achieved through Identity and Access Management (IAM) roles and permissions.
5. **Security:** Data in Google Cloud Storage can be encrypted both in transit and at rest. It supports customer-managed encryption keys (CMEK) for added control over the encryption keys used to protect your data.
6. **Storage Classes:** Google Cloud Storage offers different storage classes to accommodate various access patterns and performance requirements. These include Standard, Nearline, Coldline, and Archive storage classes.  
   **Nearline Storage Class:**
   * **Use Case:** Nearline is designed for data that is accessed less frequently, typically accessed once a month or less.
   * **Performance:** It offers lower storage costs compared to the Standard storage class but with slightly higher latency when retrieving data.
   * **Access Cost:** Retrieval of data from Nearline incurs additional costs compared to Standard storage.
   * **Data Retention:** Nearline has a minimum storage duration of 30 days. If an object is deleted or overwritten within the first 30 days, you are still charged for the minimum storage duration.
7. **Coldline Storage Class:**
   * **Use Case:** Coldline is suitable for data that is accessed less than once a quarter.
   * **Performance:** It provides even lower storage costs compared to Nearline but with a longer access time.
   * **Access Cost:** Retrieval of data from Coldline incurs higher costs than both Standard and Nearline storage classes.
   * **Data Retention:** Coldline has a minimum storage duration of 90 days, and similar to Nearline, charges still apply if an object is deleted or overwritten within this period.
8. **Archive Storage Class:**
   * **Use Case:** Archive is intended for data that is rarely accessed, with the expectation of retrieval once or twice a year.
   * **Performance:** It offers the lowest storage costs among the storage classes but with the highest access latency.
   * **Access Cost:** Retrieval of data from Archive incurs the highest costs, and there is a retrieval period (around 4 hours) before the data is available.
   * **Data Retention:** Archive has a minimum storage duration of 365 days. Early deletion or overwrite within this period still incurs charges.
9. **Integration with Other GCP Services:** Google Cloud Storage integrates seamlessly with other Google Cloud services, enabling you to build comprehensive cloud-based applications. For example, it can be used in conjunction with Google Cloud Functions, Google Cloud Pub/Sub, or Google Cloud BigQuery.
10. **Global Reach:** Google Cloud Storage has a global infrastructure, allowing you to store and access your data from various locations around the world. This ensures low-latency access and improved performance for users in different regions.

To use Google Cloud Storage, you create a storage bucket and upload your objects to it. You can interact with Google Cloud Storage using the Cloud Console, the gsutil command-line tool, or client libraries for various programming languages.

**BigQuery** is a fully-managed, serverless data warehouse provided by Google Cloud Platform (GCP). It is designed to analyze large datasets in real-time using SQL-like queries. BigQuery is part of Google Cloud's suite of data and analytics services and is widely used for business intelligence, data exploration, and machine learning applications.

Key features and aspects of BigQuery include:

1. **Serverless Architecture:** BigQuery is a serverless data warehouse, meaning users do not need to manage any infrastructure. Google Cloud handles the underlying infrastructure, ensuring automatic scaling and high availability.
2. **Scalability:** BigQuery is built for massive scalability. It can handle large datasets, ranging from gigabytes to petabytes, and automatically scales resources based on query complexity and data volume.
3. **Standard SQL:** BigQuery supports ANSI SQL, making it easy for users familiar with SQL to write queries for data analysis. It also includes extensions for handling nested and repeated data structures.
4. **Real-time Analytics:** BigQuery provides low-latency querying, enabling users to run SQL-like queries on large datasets in real-time. This makes it suitable for interactive and exploratory data analysis.
5. **Integrations:** BigQuery integrates seamlessly with other GCP services, such as Cloud Storage, Cloud Dataflow, and Cloud Machine Learning Engine. It can also ingest data from external sources, making it versatile for various data workflows.
6. **Multi-Regional Availability:** BigQuery offers multi-regional availability, allowing users to run queries on data stored in different geographic regions for improved performance.
7. **Security:** BigQuery includes robust security features, including encryption at rest and in transit, identity and access management (IAM) controls, and audit logging. It also supports fine-grained access controls at the dataset and project levels.
8. **Cost Model:** BigQuery follows a serverless pricing model, where users are billed based on the amount of data processed by their queries. This allows for cost-effective usage, especially when dealing with large datasets.
9. **Data Partitioning and Clustering:** BigQuery supports table partitioning and clustering, enabling users to organize and optimize their data for improved query performance.
10. **Data Federated Queries:** BigQuery supports federated queries, allowing users to analyze data across multiple locations, including external databases, Cloud SQL, and Cloud Storage.

**Cloud Run** is a fully managed compute platform provided by Google Cloud Platform (GCP) that allows developers to deploy containerized applications in a serverless environment. It abstracts away the infrastructure management, allowing you to focus on building and deploying your applications without dealing with the underlying servers.

Key features and aspects of Cloud Run include:

1. **Containerized Deployments:** Cloud Run allows you to deploy containerized applications. You can use Docker containers to package your applications and dependencies, making it easy to build and deploy consistent and portable applications.
2. **Serverless Model:** Cloud Run follows a serverless model, where you only pay for the compute resources consumed during the execution of your application. It automatically scales your application in response to incoming requests, handling the scaling and load balancing for you.
3. **Automatic Scaling:** Cloud Run can scale your application from zero to handle incoming requests and scale down to zero when there is no traffic. This helps in optimizing resource utilization and reducing costs when the application is not in use.
4. **Support for Stateless Applications:** Cloud Run is designed for stateless applications, meaning that each request is treated independently, and there is no reliance on the state stored on the server. Stateful data should be stored in external services like databases or object storage.
5. **HTTP(S) and gRPC Endpoints:** Cloud Run supports both HTTP(S) and gRPC protocols, making it suitable for a variety of web applications and microservices.
6. **Integration with Container Registry:** You can store your container images in Google Container Registry or other container registries and deploy them directly to Cloud Run.
7. **Environment Variables and Secrets:** Cloud Run allows you to configure environment variables and secrets for your applications, providing a way to manage configuration settings securely.
8. **Managed SSL Certificates:** Cloud Run provides automatic provisioning and renewal of SSL/TLS certificates for custom domains, ensuring secure communication with your applications.
9. **Integration with Google Cloud Logging and Monitoring:** Cloud Run seamlessly integrates with Google Cloud's monitoring and logging services, providing insights into your application's performance, errors, and usage.
10. **Custom Domains:** You can assign custom domains to your Cloud Run services, making it easy to expose your applications with branded URLs.

To deploy an application on Cloud Run, you need to containerize your application, push the container image to a container registry, and then deploy it on Cloud Run using the Cloud Console, the **gcloud** command-line tool, or other deployment methods.

Cloud Run is suitable for a wide range of use cases, including web applications, APIs, and microservices, providing a flexible and scalable platform for deploying containerized applications without managing the underlying infrastructure

Apigee is a platform provided by Google Cloud for developing and managing APIs (Application Programming Interfaces). It offers a comprehensive set of tools for API lifecycle management, including API design, deployment, security, analytics, and monetization.

Here's an overview of some key features and components of Apigee:

API Design and Development: Apigee provides tools for designing APIs using standards like OpenAPI (formerly known as Swagger) or GraphQL. Developers can define API specifications, including endpoints, request/response formats, authentication mechanisms, and documentation.

**API Gateway:** The API gateway component of Apigee acts as a front door for managing external and internal API traffic. It handles tasks such as routing, request/response transformation, rate limiting, caching, and security enforcement (e.g., authentication, authorization, and threat protection).

**Security:** Apigee offers various security features to protect APIs and data. These include OAuth 2.0 and JWT (JSON Web Tokens) support for authentication, OAuth scopes for fine-grained access control, TLS/SSL encryption for data in transit, and protection against common web threats like OWASP Top 10 vulnerabilities.

**Analytics and Monitoring:** Apigee provides analytics dashboards and reports to monitor API usage, performance, and behavior in real-time. It captures metrics such as traffic volume, response times, error rates, and geographic distribution, allowing API administrators to gain insights and make informed decisions.

**Monetization:** Organizations can use Apigee to monetize their APIs by implementing various business models, such as subscription-based pricing, pay-per-call, or revenue sharing. Apigee facilitates billing, invoicing, and revenue tracking for API products, enabling organizations to generate revenue from their digital assets.

**Developer Portal:** Apigee includes a developer portal where API consumers can discover, explore, and consume APIs. Developers can access API documentation, code samples, SDKs, and interactive API consoles. The portal also supports developer registration, key management, and community forums.

Integration: Apigee integrates with other Google Cloud services and third-party systems for seamless development and operation of APIs. It supports integration with cloud-native technologies like Kubernetes and Istio for containerized deployments and microservices architectures.

Overall, Apigee aims to streamline the process of building, deploying, and managing APIs, enabling organizations to accelerate digital transformation initiatives, drive innovation, and deliver superior digital experiences to their customers and   
  
**Apigee is a platform provided by Google Cloud for developing and managing APIs (Application Programming Interfaces).**  
**Apigee is an API management platform It enables organizations to design, secure, deploy, monitor, and scale APIs (Application Programming Interfaces) for their systems and services.  
  
API allows different software applications to communicate with each other  
It is a set of rules, protocols, and tools that allows different software applications to communicate with each other.  
  
git is a softwere use to track the changes in a file**

**github is use to host the repository in a server**

**Git is a distributed version control system (DVCS) used for tracking changes in source code during software development.  
  
GitHub is a web-based platform and hosting service for version control using Git**  
**terraform init : command** initialize required plugins and downloads   
**terraform Plan** command generates an execution plan that shows what actions Terraform will take to achieve the desired state.   
**terraform Apply**: command then applies those changes to the infrastructure.  
**terraform destroy :** command is use to delete the infrastructure  
**module** a module is a container for multiple resources that are used together.