Google Cloud

Partner Certification Academy

Professional Cloud Developer

The information in this presentation is classified:

Google confidential & proprietary

⚠ This presentation is shared with you under NDA.

- Do **not** record or take screenshots of this presentation.
- Do **not** share or otherwise distribute the information in this presentation with anyone **inside** or **outside** of your organization.

Thank you!

Google Cloud

UPDATE

Source Materials

Some of this program's content has been sourced from the following resources:

- Google Cloud certification site
- Google Cloud documentation
- Google Cloud console
- Google Cloud courses and workshops
- Google Cloud white papers
- Google Cloud Blog
- Google Cloud YouTube channel
- Google Cloud samples
- Google codelabs
- Google Cloud partner-exclusive resources

Google Cloud Skills Boost for Partners

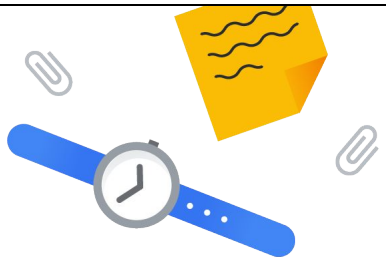
- Google Cloud Fundamentals: Core Infrastructure
- Logging, Monitoring and Observability in Google Cloud

Google Cloud Partner Advantage

- Identity Management Technical Deep Dive
- Access Management Technical Deep Dive
- Cloud Foundations: Cost Control Technical Deep Dive [PSO Y22]

Session logistics

- When you have a question, please:
  - Click the Raise hand button in Google Meet.
  - Or add your question to the Q&A section of Google Meet.
  - Please note that answers may be deferred until the end of the session.
- These slides are available in the Student Lecture section of your Qwiklabs classroom.
- The session is **not recorded.**
- Google Meet does not have persistent chat.
  - If you get disconnected, you will lose the chat history.
  - Please copy any important URLs to a local text file as they appear in the chat.



Module Agenda

01 Compute Engine

02 Managed Instance Groups

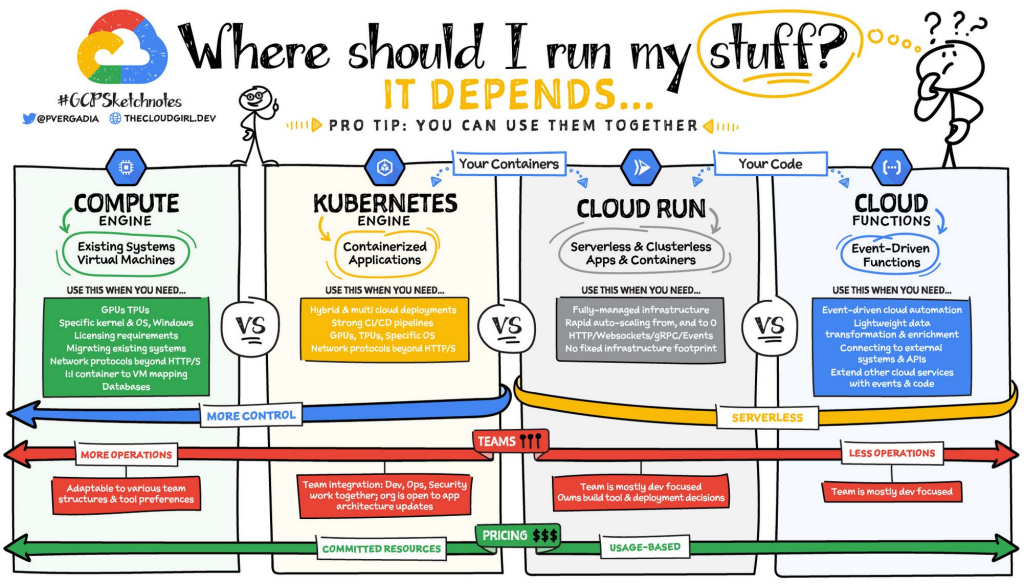
03 GKE

04 Cloud Run

05 Cloud Functions

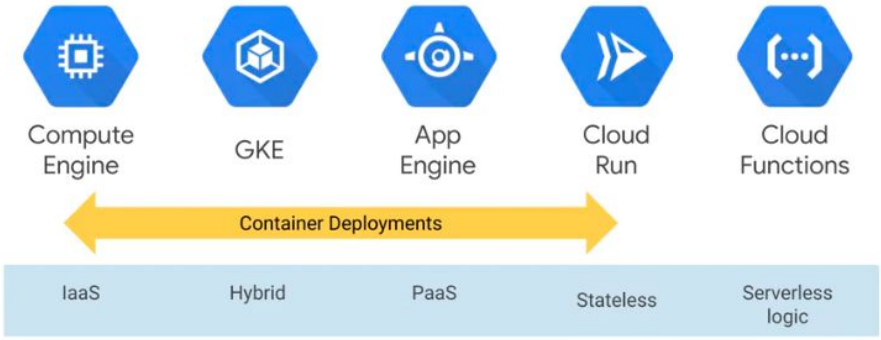
06 App Engine

07 Secret Manager



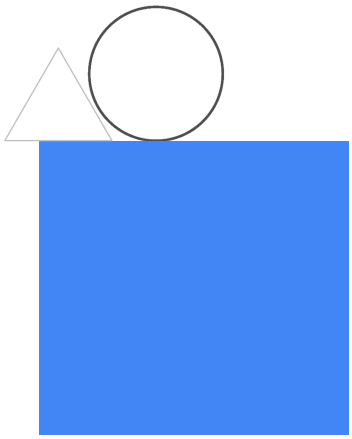
https://cloud.google.com/blog/topics/developers-practitioners/where-should-i-run-my-stuff-choosing-google-cloud-compute-option

Compute options



https://cloud.google.com/blog/topics/developers-practitioners/where-should-i-run-my-stuff-choosing-google-cloud-compute-option

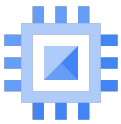
Compute Engine



Compute Engine

Virtual machines on Google's infrastructure

- **Predefined machine types**: Pre-built and ready-to-go configurations
- **Custom machine types**: Create VMs with optimal amounts of vCPU (cores) and memory (RAM), while balancing cost
- **Spot machines and preemptible virtual machines**: Reduce computing costs
- **Confidential computing**: Encrypt your most sensitive data while it's being processed
- **Rightsizing recommendations**: Optimize resource utilization with automatic recommendations
- **Per second billing**



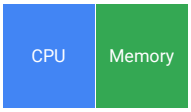
When to use Compute Engine?

- If your application cannot be packaged as a **container**.
- If your application is not stateless, or must respond to requests or events delivered using protocols other than HTTP.
- If you want maximum flexibility.

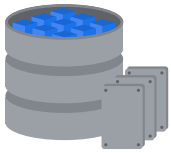
Cloud Run requires that your application be packaged and deployed as a container image. If your application is not stateless, or must respond to requests or events delivered using protocols other than HTTP, then Cloud Run will not be suitable.

If you need access to VPC resources, or need to deploy your containers to custom machine types, consider Cloud Run on GKE.

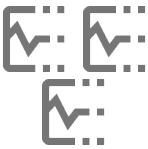
Run your application on high-performance, scalable VMs with Compute Engine



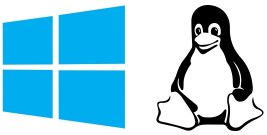
Predefined and custom machine types



Persistent Disks and Local SSDs



Spot/Pre-emptible VMs

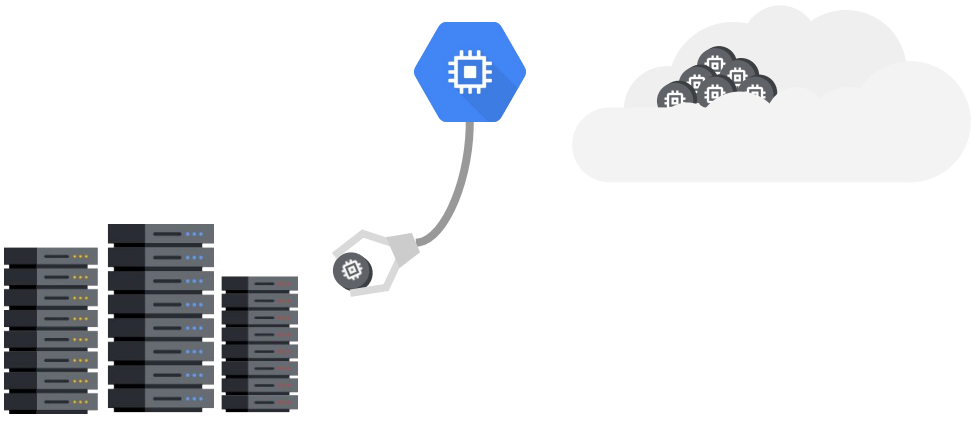


Windows, Linux OS, or Bring Your Own

Compute Engine supports predefined machine types. You can also create custom machine types to create VMs with the optimal amount of CPU and memory for your workloads. Compute Engine supports persistent disks and local SSDs. You can also launch preemptible VMs for large compute and batch jobs.

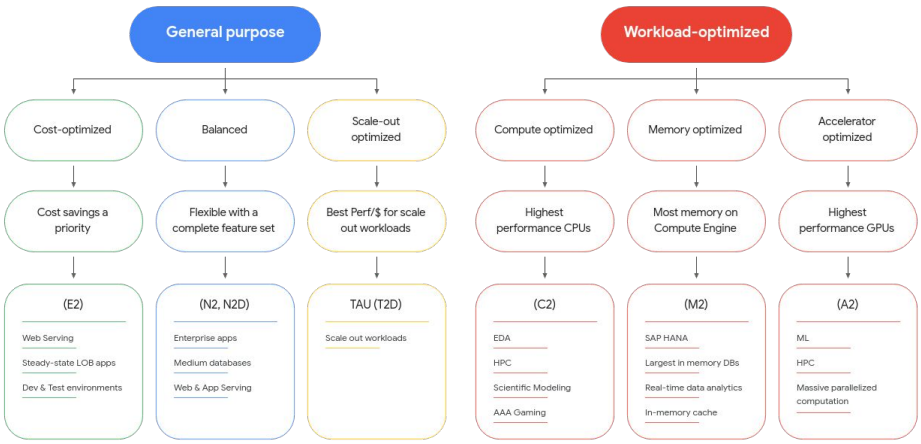
You can run your choice of operating system including Debian, CentOS, and various other flavors of Linux and Windows. You can also use a shared image from the Google Cloud community or bring your own operating system.

Use Compute Engine for lift-and-shift migration



Compute engine is ideal for lift-and-shift migration. You can move virtual machines from your on-premises data center or another cloud provider to Google Cloud without changing your application.

Compute Engine Machine Types



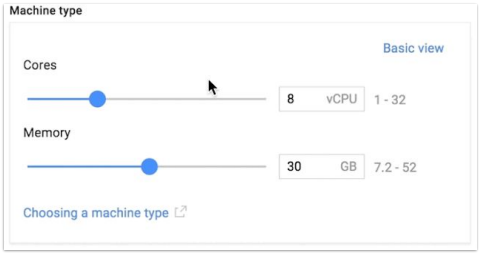
Tip: More detail and use cases found here: https://cloud.google.com/compute#section-6

Choosing the right VM type
https://cloud.google.com/compute#section-6

pls-academy-pca-student-slides-1-2301

Custom Machine Types

- Specify number of vCPU cores the memory
- Optimize resources
- Manage costs



A machine type specifies a particular collection of virtualized hardware resources available to a VM instance, including the system memory size, vCPU count, and maximum persistent disk capability.

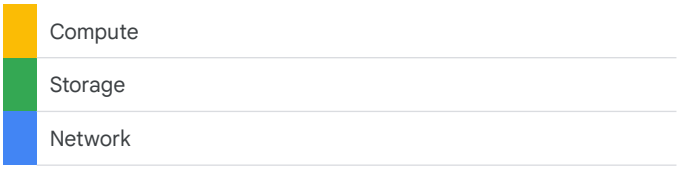
Predefined machine types:

- Have a fixed collection of resources, are managed by Compute Engine and are available in multiple different classes.
- Each class has a predefined ratio of GB of memory per vCPU

Custom machine types:

- These let you specify the number of vCPUs and the amount of memory for your instance.

Compute Engine



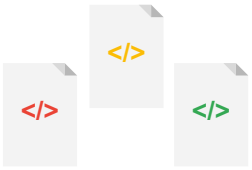
Compute Engine

Compute

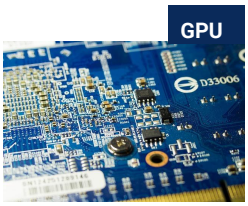
Storage

Network

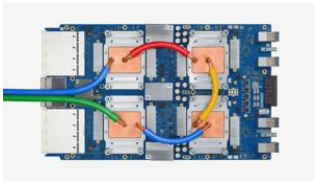
Use Compute Engine for maximum flexibility



Third-party software



Graphics Processing Unit



TensorFlow Processing Unit (TPU)

Compute engine offers you the most flexibility to configure your resources for the specific type of application that you need to run. You can run any third party license software on Compute Engine.

You can attach GPUs to Compute Engine VMs to speed up machine learning and data processing workloads.

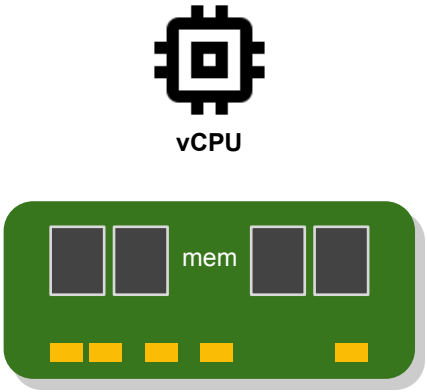
You can use Compute Engine for applications that require network protocols other than HTTP or HTTPS.

Compute

Several machine types

- Network throughput scales 2 Gbps per vCPU (small exceptions)
- Theoretical max of 32 Gbps with 16 vCPU or 100 Gbps with T4 or V100 GPUs

A vCPU is equal to 1 hardware hyper-thread



Let's start by looking at the compute options.

Compute Engine provides several different machine types that we'll discuss later in this module. If those machines don't meet your needs, you can also customize your own machine.
Your choice of CPU will affect your network throughput. Specifically, your network will scale at 2 Gbits per second for each CPU core, except for instances with 2 or 4 CPUs which receive up to 10 Gbits per second of bandwidth.

As of this recording there is a theoretical maximum throughput of 32 Gbits per second for an instance with 16 or more CPUs and a 100 Gbits per second maximum throughput for specific instances that have T4 or V100 GPUs attached.

When you're migrating from an on-premises setup, you're used to physical cores, which have hyperthreading. On Compute Engine, each virtual CPU (or vCPU) is implemented as a single hardware hyper-thread on one of the available CPU Platforms.

For an up-to-date list of all the available CPU platforms, please refer to the documentation page.

Compute Engine

Compute

Storage

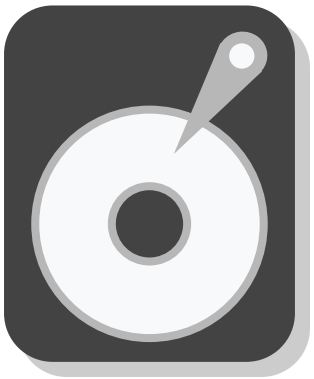
Network

Storage

Disks

- Standard, SSD, or Local SSD
- Standard and SSD PDs scale in performance for each GB of space allocated

Resize disks or migrate instances with no downtime



After you pick your compute options, you want to choose your disk.

You have three options: Standard, SSD, or local SSD. So basically, do you want the standard spinning hard disk drives (HDDs), or flash memory solid-state drives (SSDs)? Both of these options provide the same amount of capacity in terms of disk size when choosing a persistent disk. Therefore, the question really is about performance versus cost, because there's a different pricing structure.

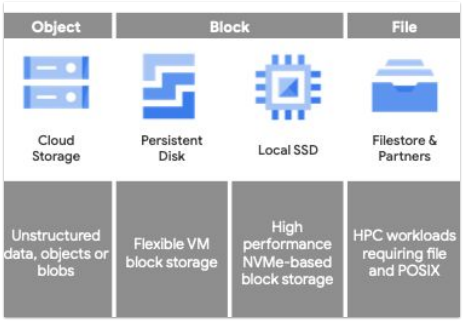
Basically, SSDs are designed to give you a higher number of IOPS per dollar versus standard disks, which will give you a higher amount of capacity for your dollar.

Local SSDs have even higher throughput and lower latency than SSD persistent disks, because they are attached to the physical hardware. However, the data that you store on local SSDs persists only until you stop or delete the instance. Typically, a local SSD is used as a swap disk, just like you would do if you want to create a ramdisk, but if you need more capacity, you can store those on a local SSD. You can create instances with up to eight separate 375-GB local SSD partitions for a total of 3 TB of local SSD space for each instance.

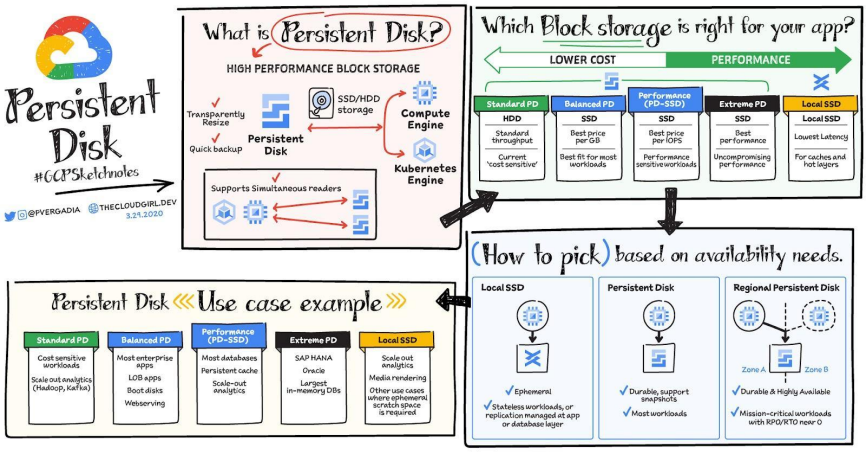
Standard and non-local SSD disks can be sized up to 257 TB for each instance. The performance of these disks scales with each GB of space allocated.

Compute Engine Data Storage

- Each VM has a boot persistent disk (PD) that contains the operating system
  - Not best practice to place non-system data on the boot disk
- When apps require additional storage space add one or more additional storage options
  - **Cloud Storage buckets**: Affordable object storage
  - **Additional persistent disk(s)**: Efficient, reliable block storage.
  - **Local SSD:** High performance, transient, local block storage.
  - **Filestore**: High performance file storage for Google Cloud users.

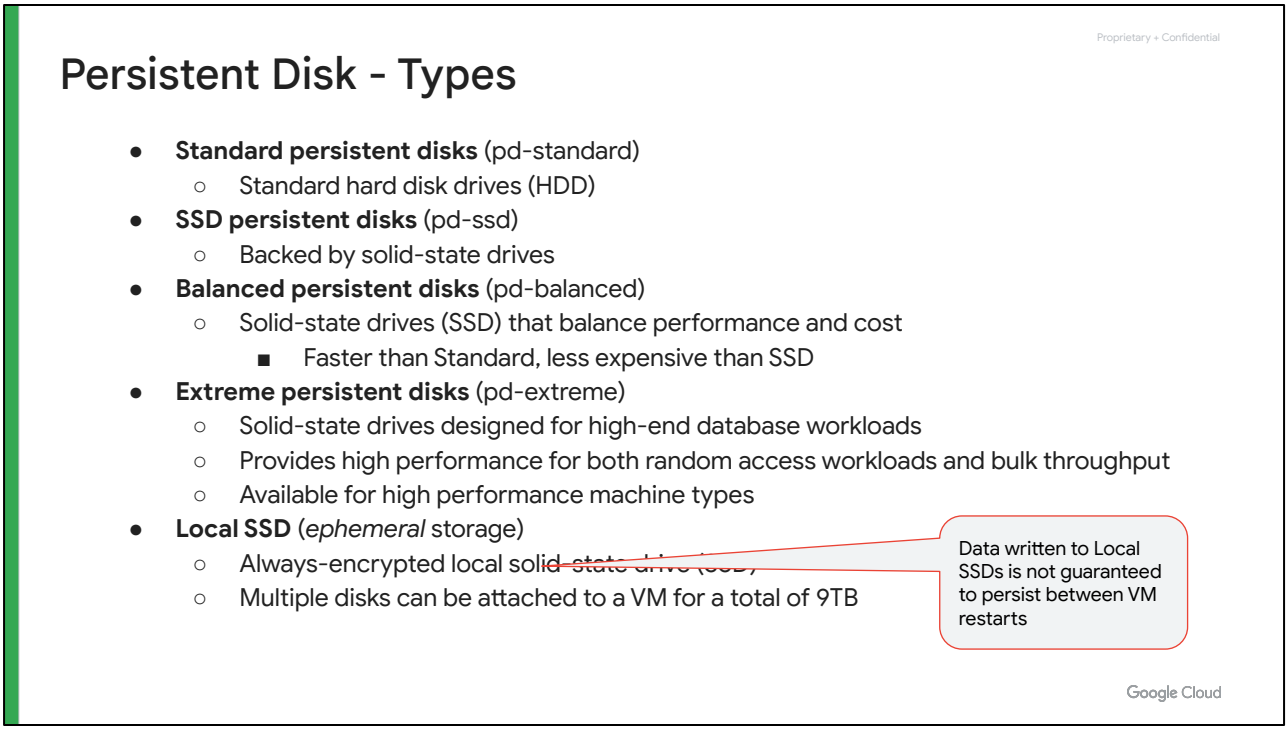


Persistent Disk



A Google Cloud block storage options cheat sheet

A Google Cloud block storage options cheat sheet
https://cloud.google.com/blog/topics/developers-practitioners/google-cloud-block-storage-options-cheat-sheet



Link to disk types: https://cloud.google.com/compute/docs/disks#disk-types
Link to extreme disk: https://cloud.google.com/compute/docs/disks/extreme-persistent-disk
Local ssd: https://cloud.google.com/compute/docs/disks/local-ssd

The first disk that we create is what we call a persistent disk. That means it's going to be attached to the VM through the network interface. Even though it's persistent, it's not physically attached to the machine. This separation of disk and compute allows the disk to survive if the VM terminates. You can also perform snapshots of these disks, which are incremental backups that we'll discuss later.

The choice between HDD and SSD disks comes down to cost and performance. To learn more about disk performance and how it scales with disk size, please refer to the documentation page.

Another cool feature of persistent disks is that you can dynamically resize them, even while they are running and attached to a VM.

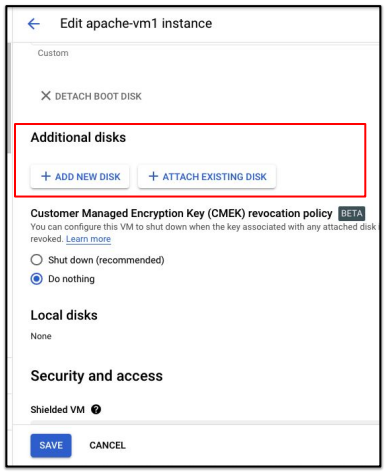
You can also attach a disk in read-only mode to multiple VMs. This allows you to share static data between multiple instances, which is cheaper than replicating your data to unique disks for individual instances.

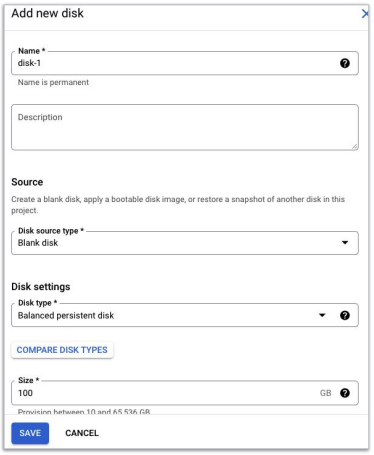
Zonal persistent disks offer efficient, reliable block storage. Regional persistent disks provide active-active disk replication across two zones in the same region. Regional persistent disks deliver durable storage that is synchronously replicated across zones and are a great option for high-performance databases and enterprise applications that also require high availability. When you configure a zonal or regional persistent disk, you can select one of the following disk types.

- Standard persistent disks (`pd-standard`). These types of disks are back by standard hard disk drives (HDD).
- Balanced persistent disks (`pd-balanced`). These types of disks are backed by solid state drives (SSD). They are an alternative to SSD persistent disks that balance performance and cost.
- SSD persistent disks (`pd-ssd`). These types of disks are backed by by solid state drives (SSD).

By default, Compute Engine encrypts all data at rest. Google Cloud handles and manages this encryption for you without any additional actions on your part. However, if you wanted to control and manage this encryption yourself, you can either use Cloud Key Management Service to create and manage key encryption keys (which is known as customer-managed encryption keys) or create and manage your own key encryption keys (known as customer-supplied encryption keys).

Adding disks to VMs with the Console





Must format a blank disk after creation

Formatting and mounting a non-boot disk on a Linux VM
https://cloud.google.com/compute/docs/disks/add-persistent-disk#format_and_mount_linux

Formatting and mounting a non-boot disk on a Windows VM
https://cloud.google.com/compute/docs/disks/add-persistent-disk#format_and_mount_windows

Regional disks provide high availability

- Synchronously replicate of data between two zones in a region
- In the event of a zonal outage where VM instance becomes unavailable
  - Spin up a VM in the secondary zone and force attach the disk
    - Time to recover = time to create VM (several minutes) + time to force attach disk (~1 minute)

```
$ gcloud compute instances attach-disk myvm2 \
  --disk data-disk --disk-scope=regional \
  --force-attach
```


High availability options using regional PDs
https://cloud.google.com/compute/docs/disks/high-availability-regional-persistent-disk

What is Filestore?

- Cloud-based managed file storage service for the Unix file system (POSIX)
- Provides native experience for standing up Network Attached Storage (NAS) for Compute Engine and Kubernetes Engine
- High-performance, fully managed network attached storage
  - Mount as file shares on Compute Engine instances
  - Used to store and serve files such as documents, images, videos, audio files, and other data
- Pay for what you use
- Capacity scales automatically scale based on demand
- Use cases:
  - Enterprise application migrations (SAP)
  - Media rendering where file shares are needed
  - Web content management

YouTube video: https://www.youtube.com/watch?v=CUwpXqEitA0

Filestore
https://cloud.google.com/filestore

Cloud Storage

Object

A piece of data. Identified by an **object name.**

Bucket

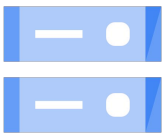
A container for **objects**. Identified by a **bucket name.**

Location

A region or multi-region where a **bucket** is located and its **objects** are stored.

Storage Class

Allows **objects** to be stored and served with different access frequencies, availability levels, and pricing profiles



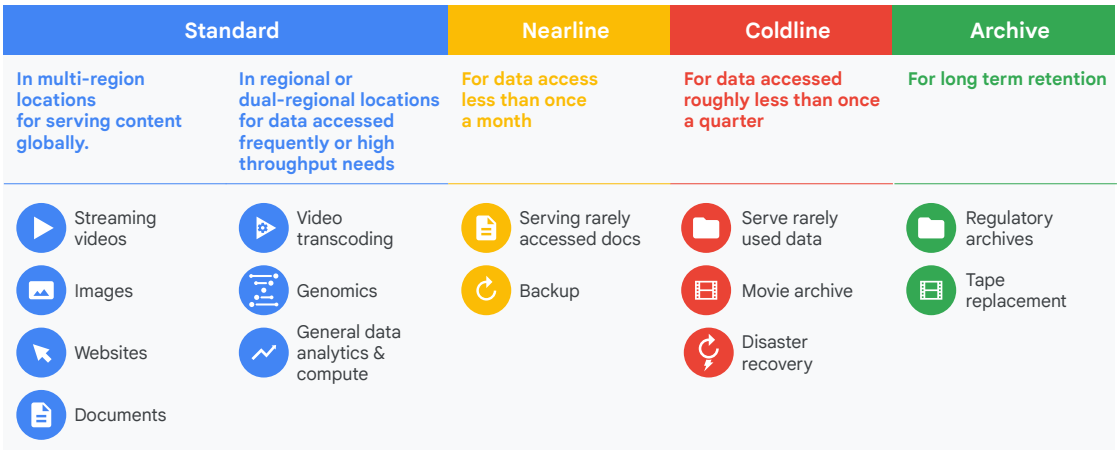
Object
A piece of data. Identified by an object name.

Bucket
A container for objects. Identified by a bucket name.

Location
A region or multi-region where a bucket is located and its objects are stored.

Storage Class
Allows objects to be stored and served with different access frequencies, availability levels, and pricing profiles

Options for any use case



Cloud Storage Bucket Locations

- **Multiregional**
  - Choose from U.S., Europe, or Asia
  - Data is replicated in multiple data centers in the chosen area
  - Use for delivering web content with the least latency
- **Dual Regional**
  - Preconfigured to store data in two regions in a geographic location
- **Regional**
  - Choose a specific data center to store data in
  - Keep your data close to the applications that use it

Source: 796 Google Cloud Advanced Skills & Certification: Professional Data Engineer

When creating a bucket, you first must choose a location.

Multiregional buckets store data in more than one region in either the USA, Europe, or Asia. Use this storage class when you want to maximize availability, like when you're delivering web content from a bucket. This storage class also gives you disaster recovery. If an entire region is down, your data is still available.

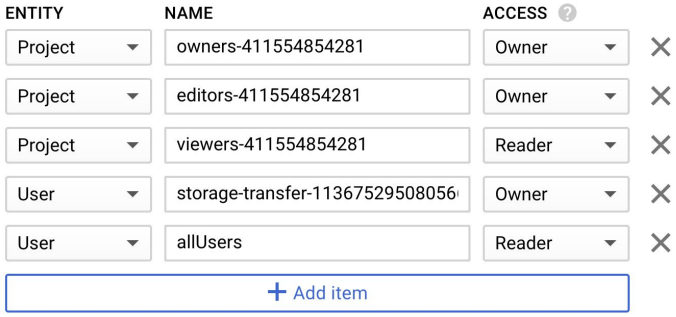
Dual-region buckets store data to two regions within a geographic location. For example, the location NAM4 will replicate data to Iowa and South Carolina. The locations EUR4 will replicate data to the Netherlands and Finland.

Choose regional storage when you want all your data to be in a single region. Let's say you have a lot of data that you want to analyze with a Hadoop cluster. Keep the data in the same region as

your Hadoop cluster. Single region storage is the least expensive solution.

Storage Object ACLs

Access Control Lists (ACLs) can be used to grant access to objects in buckets



Source: 796 Google Cloud Advanced Skills & Certification: Professional Data Engineer

You can also grant read or read/write access to individual objects within a bucket using Access Control Lists.

Signed URLs

- Provide temporary access to buckets
  - Create a service account with rights to storage
  - Create a service account key
  - Use signurl command to create a URL that allows access to the resource
    - -d parameter is used to specify duration

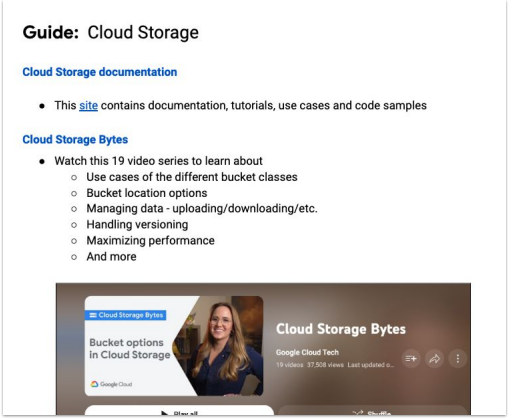
```
$ gcloud iam service-accounts keys create ~/key.json --iam-account
  storage-admin-sa@doug-demo-project.iam.gserviceaccount.com

$ gsutil signurl -d 10m ~/key.json gs://super-secure-bucket/noir.jpg
```


Source: 796 Google Cloud Advanced Skills & Certification: Professional Data Engineer

Sometimes, you want to programatically grant temporary access to an object in a bucket. You can do this with what's called a signed URL. Use the gsutil signurl command as shown on the screen. The -d parameter determines how long the signed URL works.

Guide: Cloud Storage

- Bucket location options
- Storage classes
- Managing data
- Handling versioning
- Maximizing performance
- Cloud Storage IAM
- Project level vs bucket level
- IAM roles for Cloud Storage
- Cloud Storage data encryption options
- Object Lifecycle Management
- More



Cloud Storage

Compute Engine

Compute

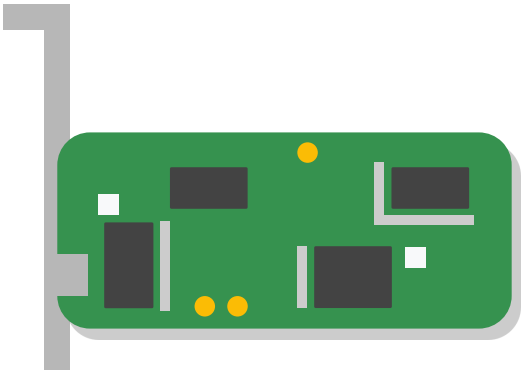
Storage

Network

Networking

Robust networking features

- Default, custom networks
- Inbound/outbound firewall rules
  - IP based
  - Instance/group tags
- Regional HTTPS load balancing
- Network load balancing
  - Does not require pre-warming
- Global and multi-regional subnetworks



As for networking, we have already seen networking features applied to Compute Engine in the previous module's lab. We looked at the different types of networks and created firewall rules using IP addresses and network tags.

You'll also notice that you can do regional HTTPS load balancing and network load balancing. This doesn't require any pre-warming because a load balancer isn't a hardware device that needs to analyze your traffic. A load balancer is essentially a set of traffic engineering rules that are coming into the Google network, and VPC is applying your rules destined for your IP address subnet range. We'll learn more about load balancers later in the course.

For a quick walk through of the VM instance creation process, refer to this demo.

VMs *must* have internal IP and *can* have external IP addresses

Internal IP

- Allocated from subnet range to VMs by DHCP
- Alternatively, can reserve (static) internal IP address

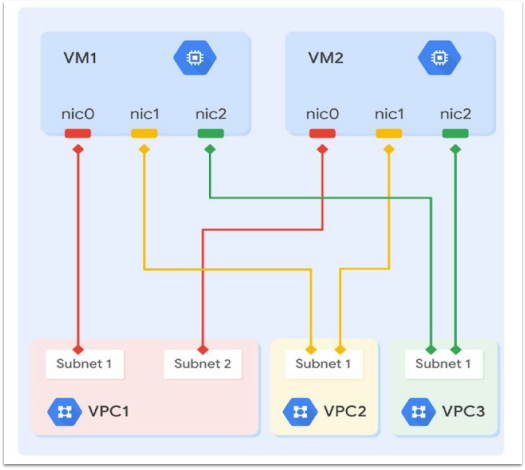
External IP

- Assigned from pool (ephemeral)
- Alternatively, can reserve (static) external IP address
- Bring Your Own IP address (BYOIP)

IP addresses
https://cloud.google.com/compute/docs/ip-addresses

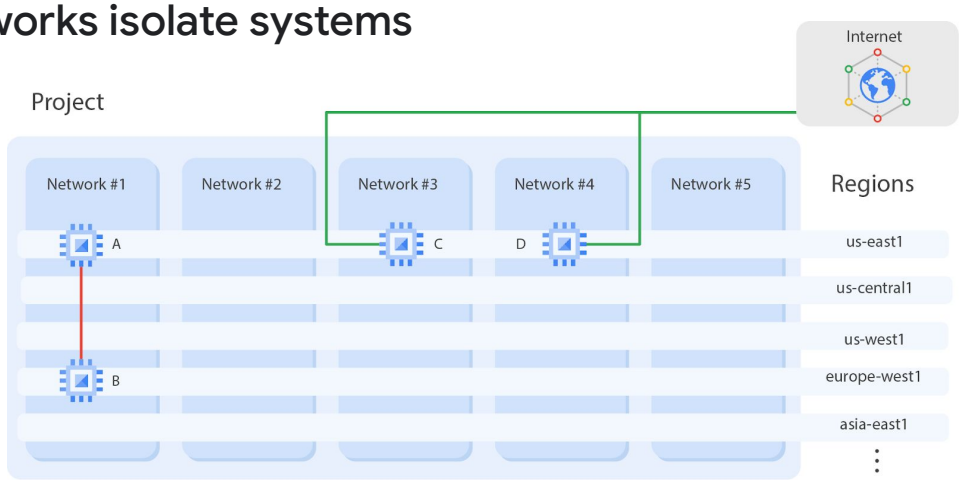
VMs can connect to multiple VPCs

- VMs have Multi-NIC support (8 max)
  - Each NIC must connect to a different VPC network
  - Allows communication between VPCs using private IPs
- Are other ways to accomplish private IP communication between VPCs, such as
  - VPC Peering
  - VPN
  - These will be discussed later



Creating instances with multiple network interfaces
https://cloud.google.com/vpc/docs/create-use-multiple-interfaces

Networks isolate systems



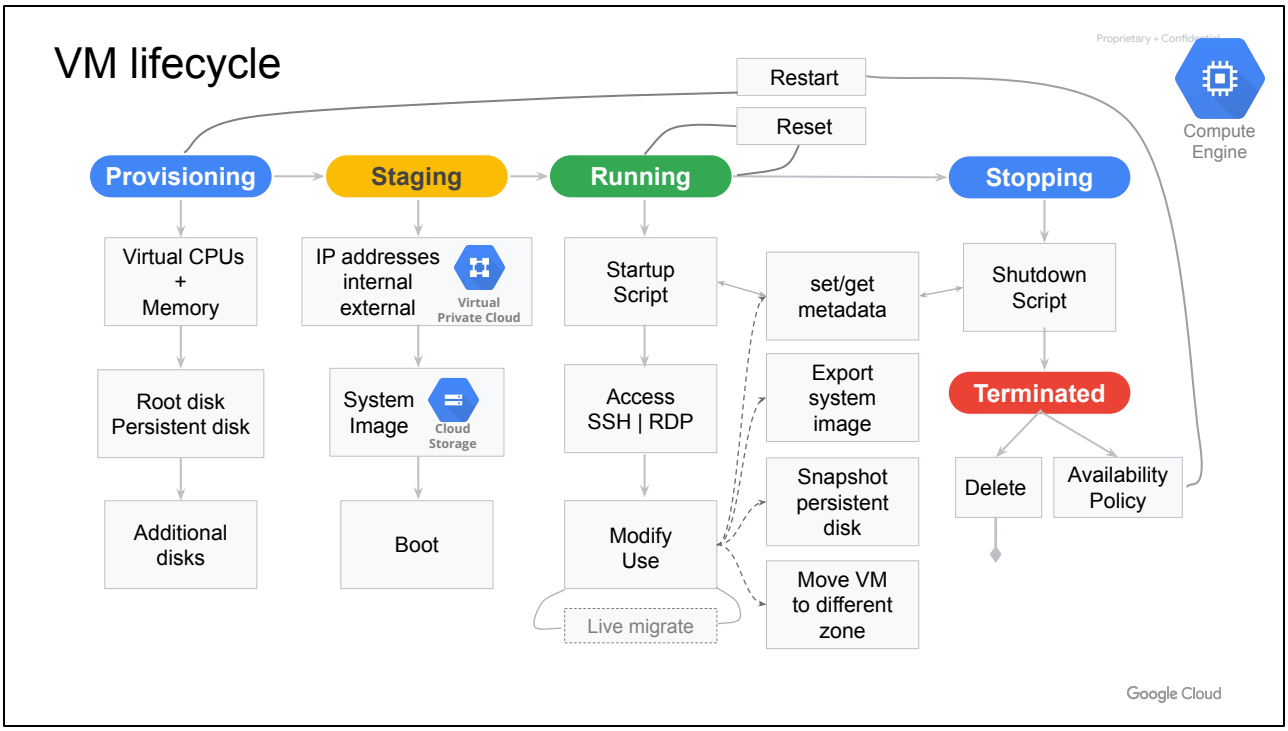
- **A** and **B** can communicate over internal IPs *even though they are in different regions.*
- **C** and **D** must communicate over external IPs *even though they are in the same region.*

On this slide, we have an example of a project that contains 5 networks. All of these networks span multiple regions across the world, as you can see on the right.

Each network contains separate virtual machines: A, B, C, and D. Because VMs A and B are in the same network, network 1, they can communicate using their internal IP addresses, even though they are in different regions. Essentially, your virtual machines, even if they exist in different locations across the world, take advantage of Google's global fiber network. Those virtual machines appear as though they're sitting in the same rack when it comes to a network configuration protocol.

VMs C and D, however, are not in the same network. Therefore, by default, these VMs must communicate using their external IP addresses, even though they are in the same region. The traffic between VMs C and D isn’t actually touching the public internet, but is going through the Google Edge routers. This has different billing and security ramifications that we will explore later.

Compute Engine, more info...



The lifecycle of a VM is represented by different statuses. We will cover this lifecycle on a high level, but we recommend returning to this diagram as a reference.

When you define all the properties of an instance and click Create, the instance enters the provisioning state. Here the resources such as CPU, memory, and disks are being reserved for the instance, but the instance itself isn't running yet. Next, the instance moves to the staging state where resources have been acquired and the instance is prepared for launch. Specifically, in this state, Compute Engine is adding IP addresses, booting up the system image, and booting up the system.

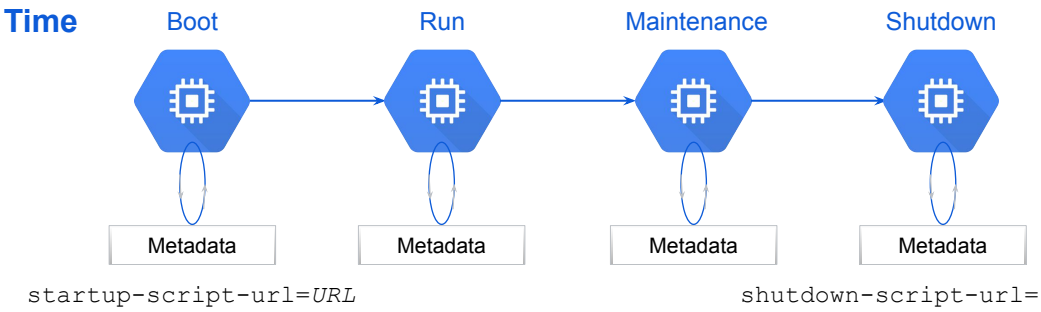
After the instance starts running, it will go through pre-configured startup scripts and enable SSH or RDP access. Now, you can do several things while your instance is running. For example, you can live migrate your virtual machine to another host in the same zone instead of requiring your instance to be rebooted. This allows Google Cloud to perform maintenance that is integral to keeping the infrastructure protected and reliable, without interrupting any of your VMs. While your instance is running, you can also move your VM to a different zone, take a snapshot of the VM's persistent disk, export the system image, or reconfigure metadata. We will explore some of these tasks in later labs.

Some actions require you to stop your virtual machine; for example, if you want to

upgrade your machine by adding more CPU. When the instance enters this state, it will go through pre-configured shutdown scripts and end in the terminated state. From this state, you can choose to either restart the instance, which would bring it back to its provisioning state, or delete it.

You also have the option to reset a VM, which is similar to pressing the reset button on your computer. This actions wipes the memory contents of the machine and resets the virtual machine to its initial state. The instance remains in the running state through the reset.

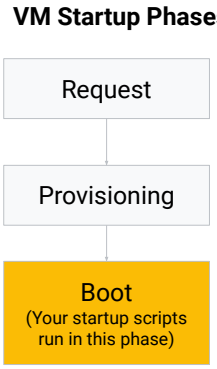
Metadata and scripts



Every VM instance stores its metadata on a metadata server. The metadata server is particularly useful in combination with startup and shutdown scripts, because you can use the metadata server to programmatically get unique information about an instance, without additional authorization. For example, you can write a startup script that gets the metadata key/value pair for an instance's external IP address and use that IP address in your script to set up a database. Because the default metadata keys are the same on every instance, you can reuse your script without having to update it for each instance. This helps you create less brittle code for your applications.

Storing and retrieving instance metadata is a very common Compute Engine action. We recommend storing the startup and shutdown scripts in Cloud Storage, as you will explore in the upcoming lab of this module.

Consider startup time



- Profile startup scripts
- Consider custom image
- Set appropriate target usage levels in autoscaling policy

Consider startup time when you use Compute Engine instances. A virtual machine can take about 60 seconds to spin up and become available. You can, however, launch hundreds of VMs within a few minutes.

To ensure that VM instances launch quickly, profile your startup scripts to identify and correct steps that take a long time to complete. If you're downloading and installing a lot of software, consider creating a custom image with all the software pre-installed. Plan ahead for bursts of traffic by setting appropriate target usage levels in your auto-scaling policy.

For more information, see https://cloudplatform.googleblog.com/2017/07/three-steps-to-Compute-Engine-startup-time-bliss-Google-Cloud-Performance-Atlas.html.

VM access

- **Linux: SSH**
  - SSH from Cloud Console or CloudShell via Cloud SDK
  - SSH from computer or third-party client and generate key pair
  - Requires firewall rule to allow tcp:22
- **Windows: RDP**
  - RDP clients
  - Powershell terminal
  - Requires setting the Windows password
  - Requires firewall rule to allow tcp:3389

For accessing a VM, the creator of an instance has full root privileges on that instance.

On a Linux instance, the creator has SSH capability and can use the Cloud Console to grant SSH capability to other users.

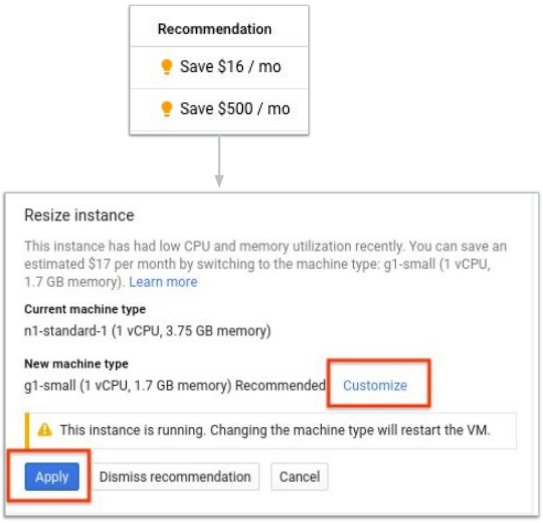
On a Windows instance, the creator can use the Cloud Console to generate a username and password. After that, anyone who knows the username and password can connect to the instance using a Remote Desktop Protocol, or RDP, client.

We listed the required firewall rules for both SSH and RDP here, but you don't need to define these if you are using the default network that we covered in the previous module.

Refer to these links, for more information on SSH key management and creating passwords for Windows instances.

Rightsizing recommendations

- Help optimize the resource utilization of VMs
- Generated automatically based on system metrics gathered by the Cloud Monitoring service over the previous 8 days.
- Suggests resizing instance's machine type to more efficiently use the instance's resources.
  - Avoid paying for idle and oversized resources
- Recommendations are free of charge.



Applying Machine Type Recommendations
https://cloud.google.com/compute/docs/instances/apply-machine-type-recommendations-for-instances

- Compute Engine provides machine type recommendations to help you optimize the resource utilization of your virtual machine (VM) instances.
- These recommendations are generated automatically based on system metrics gathered by the Cloud Monitoring service over the previous 8 days.
- Use these recommendations to resize your instance's machine type to more efficiently use the instance's resources. This feature is also known as rightsizing recommendations.

Note: You might have valid reasons for running a particular instance at very low or very high utilization. Machine type recommendations are suggestions to help you more efficiently use your instances, but they might not be appropriate for every situation.

Preemptible and Spot VMs (1/2)

- A highly discounted VM compared to the price of standard VMs
  - Discount of **60-91%** discount
  - Availability depends on having excess compute capacity in a zone
    - May or may not have availability in a given zone at a given time
    - Will have to try another zone or wait for the resource to be available
- Compute Engine might stop preemptible/spot instances at any time due to system events
  - **Preemptible VMs** - always stopped after they run for 24 hours.
    - May be stopped before the 24 hour time period
    - When restarted, the 24 hour clock resets
  - **Spot VMs** - stopped/deleted when Google needs the resource elsewhere
    - Spot VMs are the latest version of preemptible VMs
    - Can specify termination or deletion when creating the VM

Preemptible VM instances
https://cloud.google.com/compute/docs/instances/preemptible

Spot VMs
https://cloud.google.com/spot-vms
https://cloud.google.com/compute/docs/instances/spot

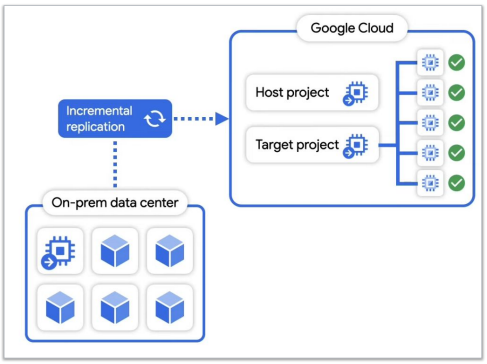
Preemptible and Spot VMs (2/2)

- Offer the same machine types, options, and performance as regular compute instances
- Use cases
  - Stateless and scalable workloads that can be stopped and checkpointed in less than 30 seconds, or is location and hardware flexible
- Provides no live migration or automatic restart during maintenance events
- Not covered by Service Level Agreement due to the preceding limitations,
- No free tier

Tip: Look through all the links in Top 5 use cases for Google Cloud Spot VMs explained + best practices

Migrate to Compute Engine Virtual Machines

- Migrate VMs to Google Cloud Compute Engine directly from on-premise including support for customizations to networking, disks, and more
  - Includes VMs running Microsoft Windows applications such as SQL Server
- Some of the benefits include
  - Built-in testing makes it fast and easy to validate before migration
  - Can replicate data from the source workload to the destination without manual steps or interruptions to the running workload
  - Provides usage-driven analytics to help you rightsize destination instances and avoid cloud over-provisioning



Migrate to Virtual Machines
https://cloud.google.com/migrate/virtual-machines

Migrating VMs with Migrate to Virtual Machines: Getting started
https://cloud.google.com/architecture/migrating-vms-migrate-for-compute-engine-getting-started

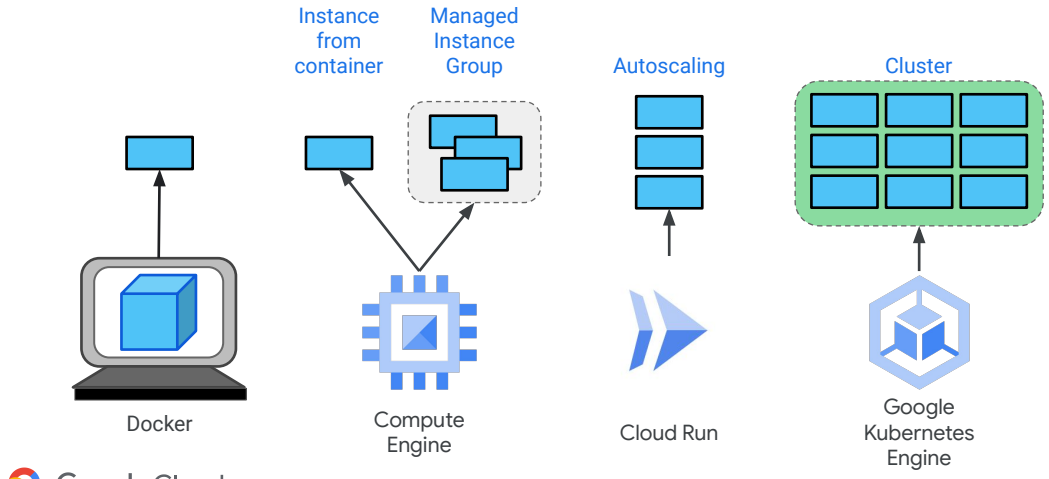
Migrate to Containers

- Automated approach to migrate *applications* running in VMs to
  - Google Kubernetes Engine
  - Cloud Run
  - Anthos clusters
- Just some of the benefits
  - Higher utilization and density of nodes, leveraging automatic bin-packing and auto-scaling capabilities of GKE
  - Reduced downtime by leveraging Kubernetes features like self healing and dynamic scaling
  - Environment parity with improved visibility and monitoring, makes finding and fixing problems less toilsome

https://cloud.google.com/migrate/containers

Blog: Migrating apps to containers? Why Migrate to Containers is your best bet
https://cloud.google.com/blog/products/containers-kubernetes/how-migrate-for-anthos-improves-vm-to-container-migration

Where can you run containers?



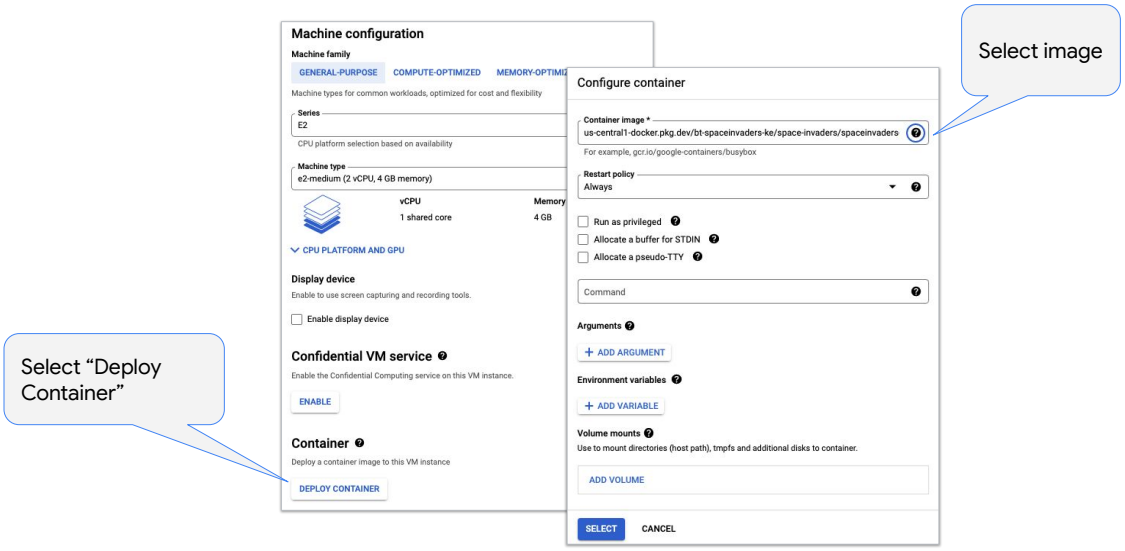
Google Cloud

Run containers directly on Compute Engine using a Container Optimized OS

- Simplifies app deployment for your VM infrastructure
  - Use familiar tools such as the Google Cloud CLI or the Compute Engine API to manage VMs
  - Create scalable services using managed instance groups (MIGs) running containers
    - Provides autoscaling, autohealing, rolling updates, multi-zone deployments, and backends for load balancing
- Example use case
  - Lift and shift from on-premise VMs running a containerized application
- Limitations
  - Each VM instance is limited to one container
  - Consider Google Kubernetes Engine if need to deploy multiple containers per VM instance

Deploying containers on VMs and MIGs
https://cloud.google.com/compute/docs/containers/deploying-containers

Compute Engine - Deploying a container



Containers on Compute Engine

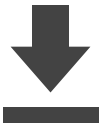
Containers on Compute Engine
https://cloud.google.com/compute/docs/containers

Limitation: only 1 container per VM:
https://cloud.google.com/compute/docs/containers/deploying-containers#limitations

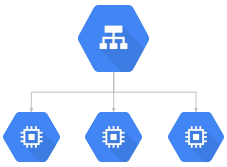
To run more than one container per VM, use Google Kubernetes Engine

Use Compute Engine for full control of infrastructure

Machine type and OS



Software



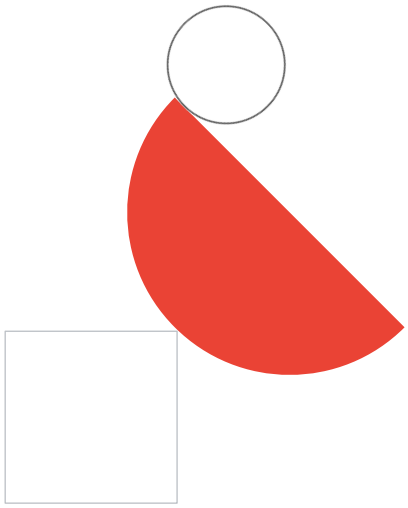
Instance groups with global load balancing

Use Compute Engine for full control of your infrastructure. Compute Engine enables you to create highly customized VMs for specialized applications that have unique compute or operating system requirements.

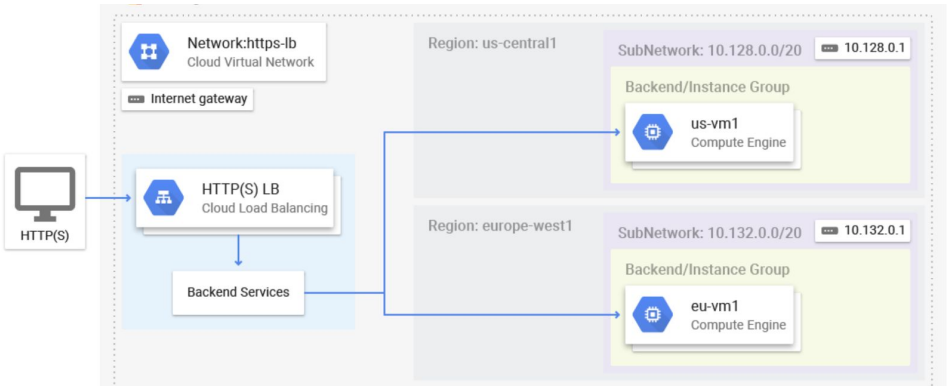
You can install and patch software running on the VM.

You can create managed instance groups based on an instance template. You can configure global load balancing and auto scaling of the managed instance groups. Compute Engine can perform health checks and replace unhealthy instances in an instance group. It auto-scales instances based on the traffic volume in specific regions.

Manage Instance Groups



Scalable, Fault Tolerant Architecture Illustrated



This illustration shows a fault tolerant architecture using load balancing and autoscaling.

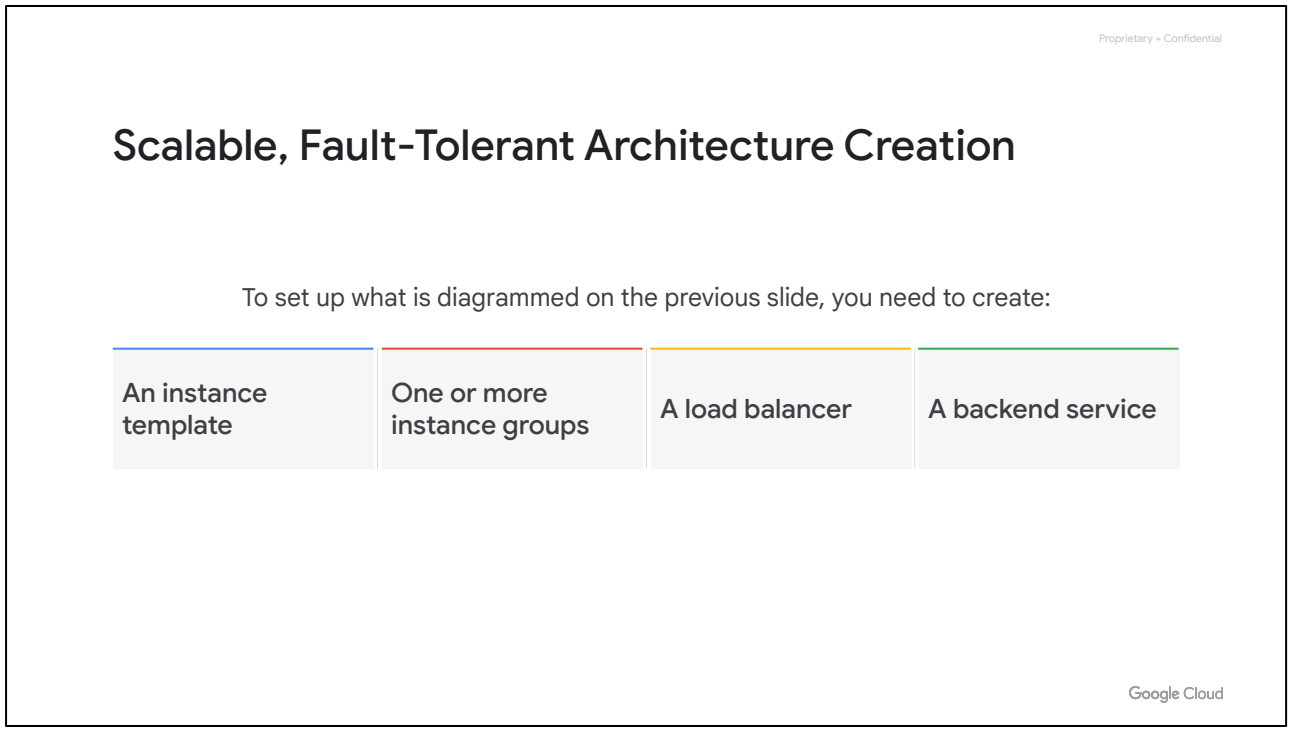
In this example, HTTP(S) traffic is being delivered to an Global Load Balancer. The load balancer is a managed service, so performance and high availability is built into the service.

The backend service is responsible for distributing traffic to the appropriate region based on latency.

Each region, in this example, has a managed instance group of instances that will be increased or decreased based on a predefined metric you choose based on the application.

The infrastructure shown can survive a region outage and still function.

pls-academy-pca-student-slides-3-2301



In order to build the infrastructure in the previous slide, you would need to:

- Build an instance template
- Create one or more managed instance groups(MIG) at the regions required
- Build a global load balancer with the MIG as the backend service

pls-academy-pca-student-slides-3-2301

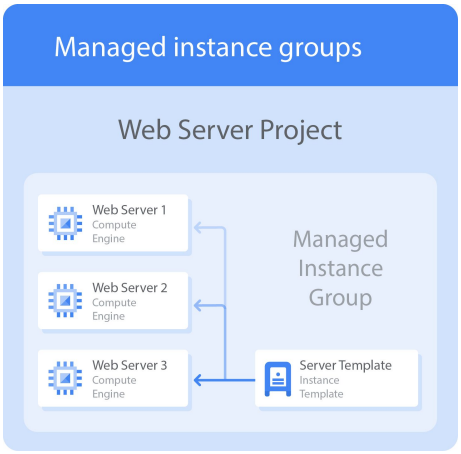
Compute Engine elasticity - Instance Groups

- Two types
  - **Managed instance groups**
    - Group of **identical** machines created from a template
    - Use case: When need VM elasticity based on demand
  - **Unmanaged instance groups**
    - Group of VMs with **different configurations**
    - Use case: Lift and shift of on-premise workloads that need a load balancer to serve traffic
      - No automatic elasticity or automatic healing

https://cloud.google.com/compute/docs/instance-groups

Managed instance groups create VMs based on templates

- Instance templates define the VMs: image, machine type, etc.
  - Test to find the smallest machine type that will run your program
- Instance group manager creates the machines.
- Optimize cost and meet varying user workloads via autoscaling
- Enable auto healing via health checks
- Can be single zone or regional
  - Use multiple zones for high availability



Managed instance groups create VMs based on instance templates. Instance templates are just a resource used to define VMs and managed instance groups. The templates define the boot disk image or container image to be used, the machine type, labels, and other instance properties like a startup script to install software from a Git repository.

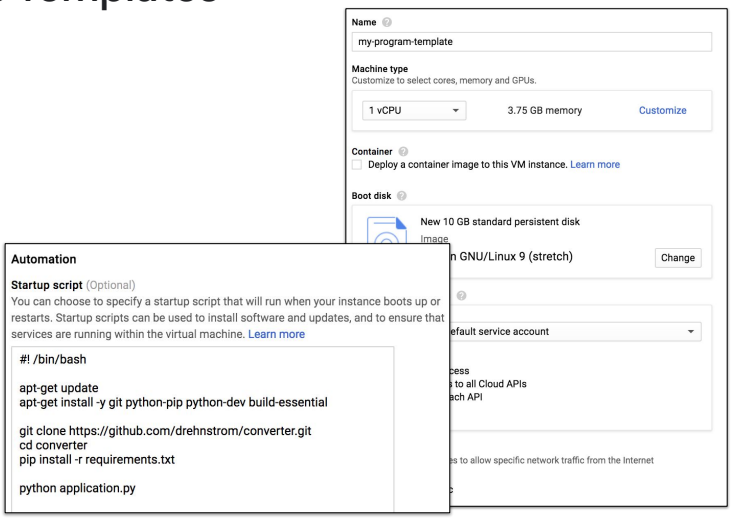
The virtual machines in a managed instance group are created by an instance group manager. Using a managed instance group offers many advantages, such as autohealing to re-create instances that don't respond and creating instances in multiple zones for high availability.

pls-academy-pca-student-slides-3-2301

Creating an Instance Templates

Contains the information required to build a VM for a deployment

- Like building a VM, but don't create the instance, create the template
- Include a startup script to automate code deployment
- Can also use a custom image



https://cloud.google.com/compute/docs/instance-templates

Google Cloud

Creating an instance template is similar to creating a virtual machine.

The template will include all information required to build the instance.

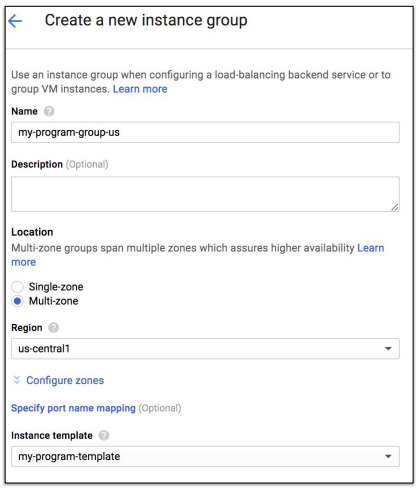
The information provided to build the template can include startup scripts to automate code deployment if needed.

pls-academy-pca-student-slides-3-2301

Creating an Instance Group

Instance groups create machines based on instance templates

- Specify how many machines to create
- In which region
- Based on what template
- Autoscaler adds and removes machines based on demand
- Health check ensures machines are working



An instance group enables you to manage a group of virtual machines as a single entity.

All virtual machines in a managed instance group are built from the same template.

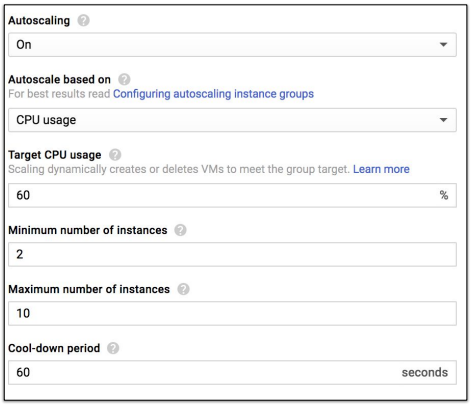
When building the group, you specify:

- Number and location of instances
- Template to build from
- Metric or metrics to use to scale instances
- Health check to verify machines are working properly

pls-academy-pca-student-slides-3-2301

Specifying an Autoscaling Configuration

- Defines when to create or destroy machines
- Pick a metric and a threshold
  - CPU utilization
  - Load balancing capacity
  - Monitoring metrics
  - Predictions based on history
- Specify minimum and maximum number of machines



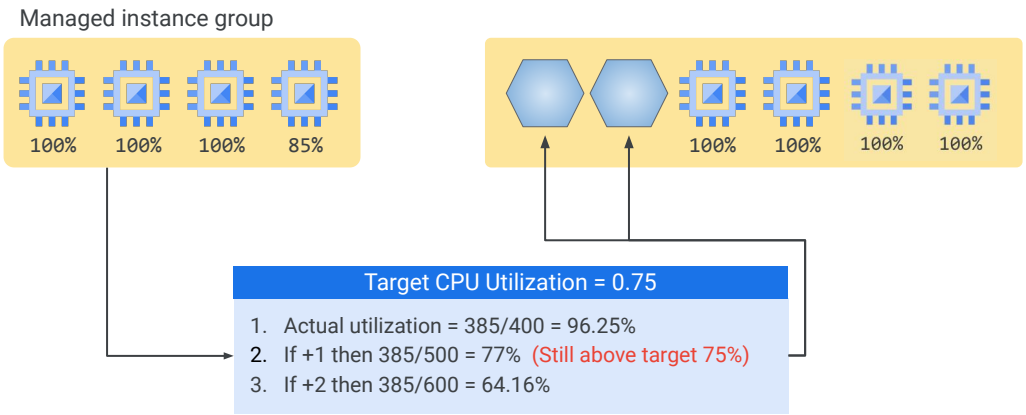
https://cloud.google.com/compute/docs/autoscaler

Part of the configuration of the managed instance group includes how to configure autoscaling. This configuration defines when to create and destroy instances.

Pick the metrics and thresholds to use.

Specify a minimum and maximum number of instances to run.

Example Scale-out policy decision



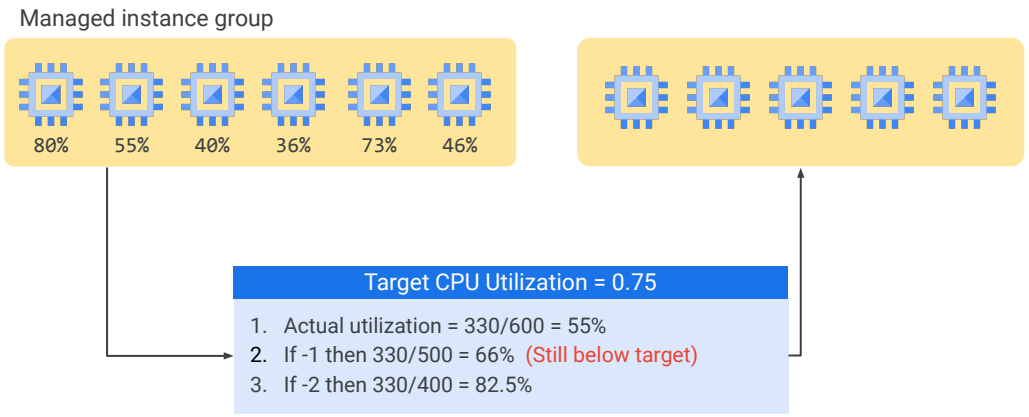
The percentage utilization that an additional VM contributes depends on the size of the group. The 4th VM added to a group offers 25% increase in capacity to the group. The 10th VM added to a group only offers 10% more capacity, even though the VMs are the same size.

In this case shown in the diagram Autoscaler is conservative and rounds up. In other words, it would prefer to start an extra VM that isn't really needed than to possibly run out of capacity.

https://cloud.google.com/compute/docs/autoscaler/understanding-autoscaler-decisions

https://cloud.google.com/compute/docs/autoscaler/multiple-policies

Example Scale-in policy decision



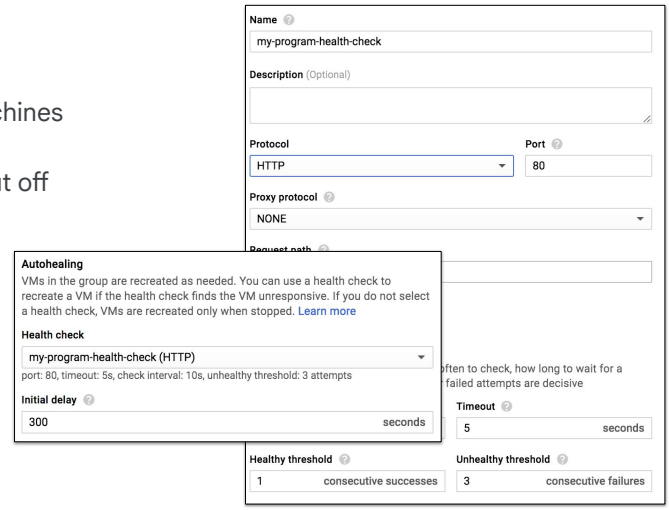
In this example, removing one VM doesn't get close enough to the target of 75%. Removing a second VM would exceed the target. Autoscaler behaves conservatively. So it will shut down one VM rather than two VMs. It would prefer underutilization over running out of resource when it is needed.

TIP: When would you use Cloud Monitoring metrics for autoscaling? A couple of examples: When the customer has a custom metric (e.g., 'x' number of people playing a game) that they want to use for scaling. Or if the application running on the VMs is reading data from a Pub/Sub queue, maybe scale when the queue reaches a certain size.

pls-academy-pca-student-slides-3-2301

Health Checks

- Simply makes requests to the machines in the instance groups, and if the machines don't work, they are shut off and new ones created
- Parameters control:
  - Where to make request
  - Via what port
  - How often



Set up an application health check and autohealing

Google Cloud

Health checks ensure that only fully operable instances are being used from the group. With health checks, instances that fail the health check can be excluded and new instances can take their place.

Parameters required to configure the health check include:

- Where to make the request
- What port to use
- And how often to run the check

Health checks need a firewall rule to allow incoming probes from certain ports. See https://cloud.google.com/compute/docs/instance-groups/autohealing-instances-in-migs

pls-academy-pca-student-slides-3-2301

Stateful Managed Instance Groups

- MIGs support both stateful and stateless workloads
  - Stateful workloads preserve individual VM state (for example, a database shard, or app configuration) on the VM's disks
    - Not easy to scale horizontally (add more nodes)
      - Could require data replication, creation or deletion of data shards, or changing the overall application configuration
    - If VM is reported as “unhealthy”
      - Need to retain VM identity (name), IP address, metadata, and data to be used when a new one is created
  - Stateless workloads, like a web frontend, do not retain any state on the individual VMs
    - Easy to scale up/down as needed

Stateful managed instance groups

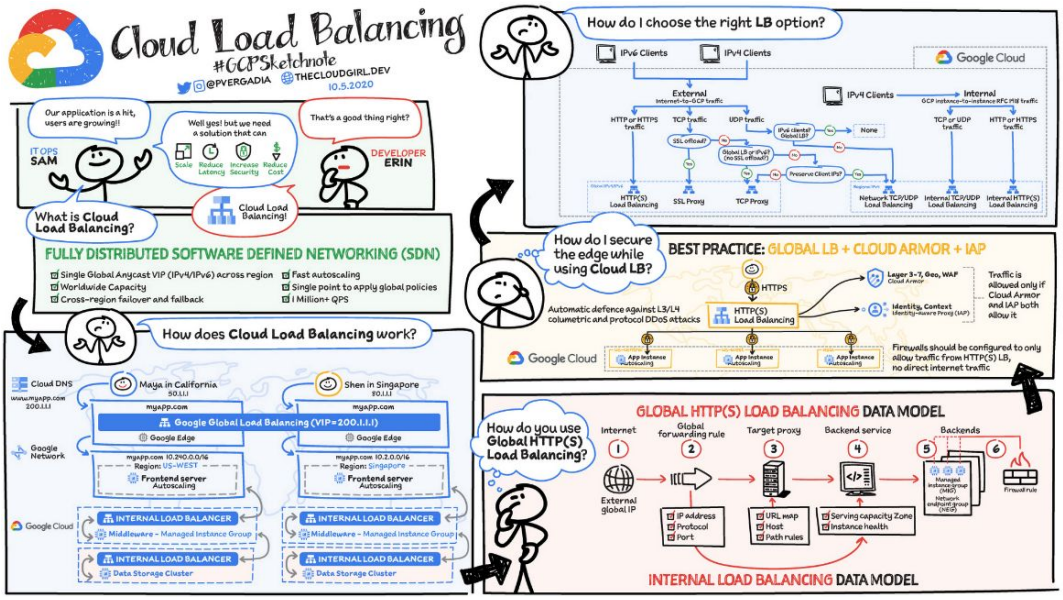
When to use Stateful Managed Instance Groups

- Databases such as Cassandra, ElasticSearch, MongoDB, and ZooKeeper.
  - You are responsible for replicating data if the database doesn't do it for you
- Data processing applications such as Kafka (Pub/Sub) and Flink (Dataflow)
- Other stateful applications such as TeamCity, Jenkins, Bamboo, or custom stateful workloads
- Legacy monolith applications that store application state on a boot disk or additional persistent disks
- Batch workloads with checkpointing.
  - Can preserve checkpointed results of long-running computation on disk in anticipation of workload or VM failure or instance preemption.
  - Stateful MIGs can recreate a failed machine, while preserving its data disk, so that your computation can continue from the last checkpoint.

Comparison of Stateless and Stateful MIGs

	Stateless	Stateful
Autoscaling	YES	NO
Disk preservation	NO	YES
Auto-healing	YES	YES
Auto-updating	YES	YES
Load balancing	YES	YES
Multi-zone deployment	YES	YES

Health checks help ensure a Stateful VM can be recreated if a failure occurred

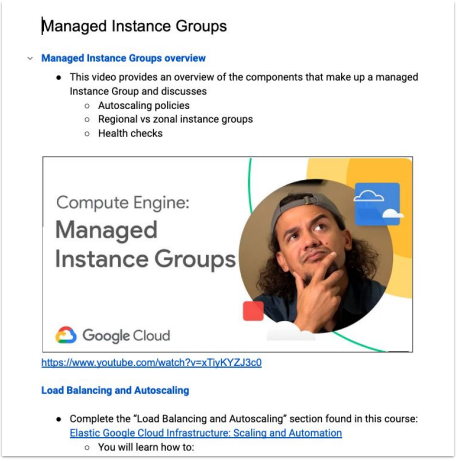


Video: https://www.youtube.com/watch?v=h8EqM6Xt3MA

Corresponding blog
https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-load-balancing

Guide: Manage Instance Groups

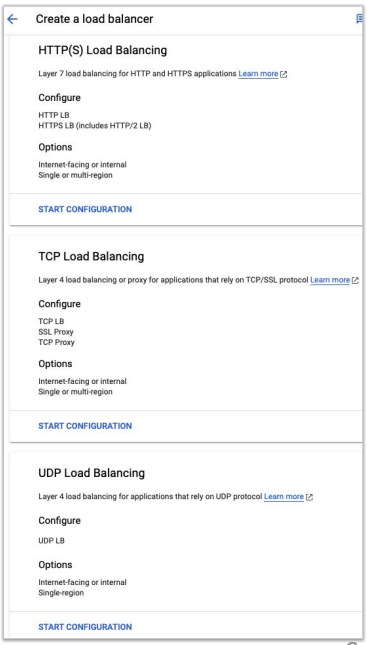
- Autoscaling policies
- Regional vs zonal instance groups
- Health checks
- Configurations
- Auto-healing
- Best practices



Manage Instance Groups

Load balancing Features

- High performance
- Fully distributed and software defined
- Anycast IP
- Regional/zonal spillover and failover
- Intelligent backend autoscaling and health checks



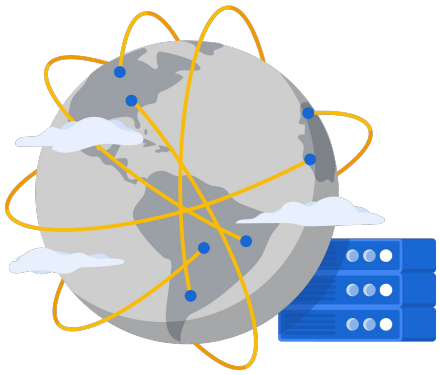
Cloud Load Balancing
https://cloud.google.com/load-balancing

- Load balancers are **software defined and distributed**, which means they are **highly performant** and not bottlenecked by a single appliance
- They are **tightly integrated with Compute Engine and GKE** to allow for **intelligent autoscaling** based on various metrics, and **health checks**, so we can route traffic to the **healthy instances and locations.**
- Load balancers are available based on **geo scope** (regional / global), **network tier** (premium / standard) and **proxy or pass-through**
- **Pass-through** proxies means client IP is preserved. **Proxy** means the client IP is not preserved, and a different connection is established between the LB and the backend instances.
- **High-level review**
  - **Internal**
    - **Regional** and require **premium network tier**
    - **L4 (TCP/UDP)** and **L7 (HTTP/s)**
  - **External**
    - Supports both **network tiers**
      - **Proxy** load balancers (TCP/SSL/HTTPs) are available as **global resources** by **directing traffic** to **healthy**

        - **backends** closest to the end-user. For that, they make use of **Google global network infrastructure**, and accordingly require **Premium Tier.**
        - They are also supported with **standard tier**, in which case they effectively function as **regional load balancers**
- **HTTPs** load balancer easily integrates with **Cloud CDN** and **Cloud Armor (WAF)**
- The **Network Load Balancer** provides a L4 (TCP/UDP) regional load balancer that is pass-through

HTTP(S) load balancing

- Global or regional (internal or external) load balancing
- Anycast IP address
- HTTP on port 80 or 8080
- HTTPs on port 443
- IPv4 or IPv6
- Autoscaling
- URL maps



HTTP(S) Load Balancing

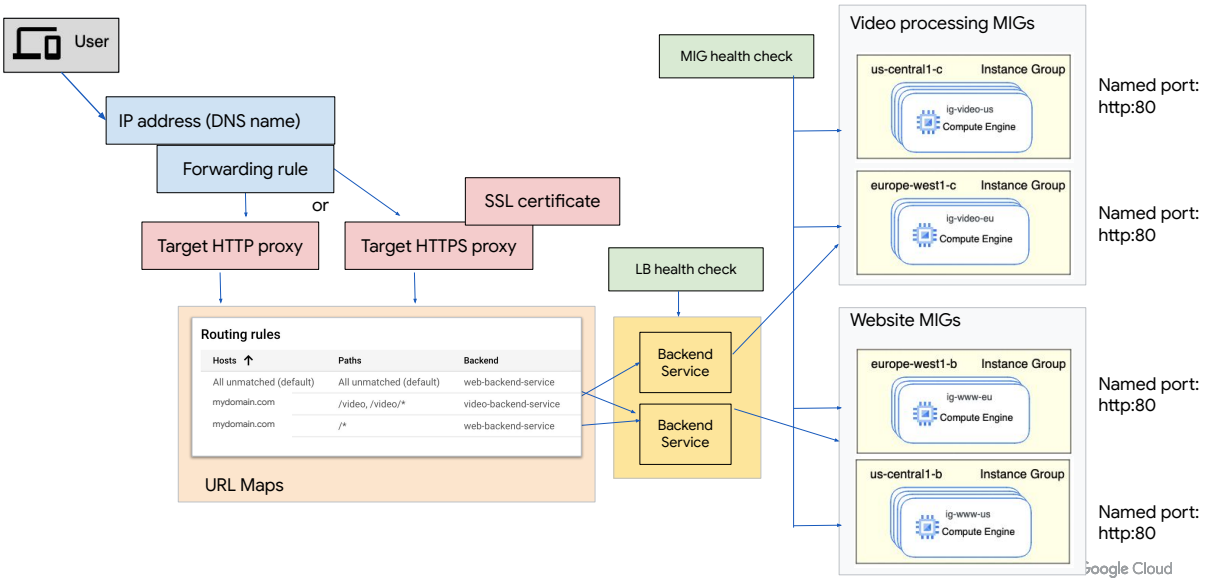
Google Cloud's HTTP(S) load balancing provides global load balancing for HTTP(S) requests destined for your instances. This means that your applications are available to your customers at a single anycast IP address, which simplifies your DNS setup. HTTP(S) load balancing balances HTTP and HTTPS traffic across multiple backend instances and across multiple regions.

HTTP requests are load balanced on port 80 or 8080, and HTTPS requests are load balanced on port 443.

This load balancer supports both IPv4 and IPv6 clients, is scalable, requires no pre-warming, and enables content-based and cross-region load balancing.

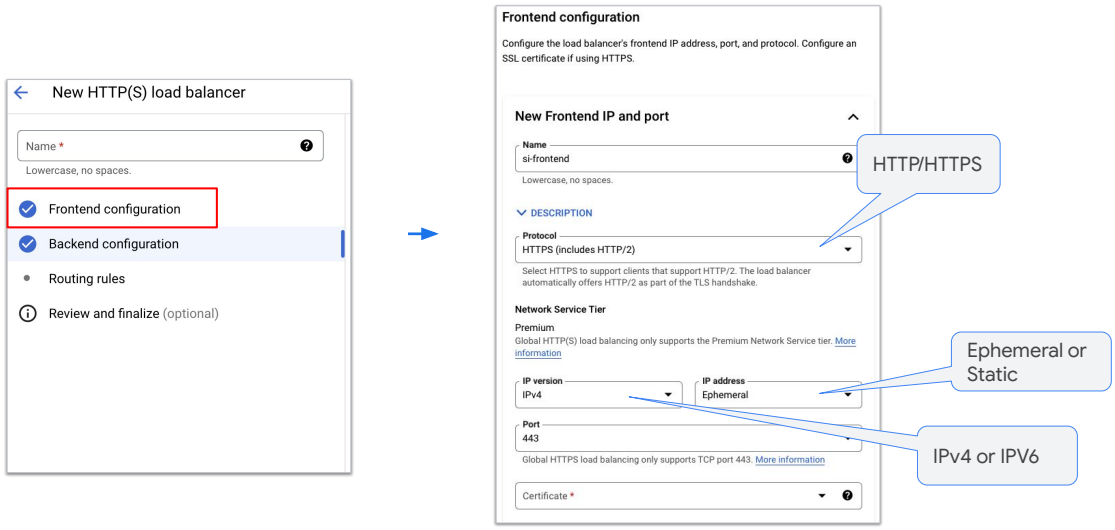
You can configure URL maps that route some URLs to one set of instances and route other URLs to other instances. Requests are generally routed to the instance group that is closest to the user. If the closest instance group does not have sufficient capacity, the request is sent to the next closest instance group that does have capacity.

Example: Load balancing MIGs across regions and backends



Tutorial: Request routing to a multi-region external HTTPS load balancer
https://cloud.google.com/load-balancing/docs/https/setting-up-https#gcloud_3

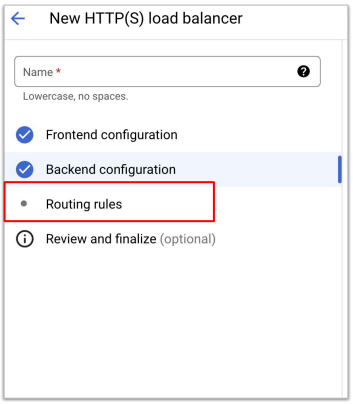
Setting up the Frontend configuration



URL maps overview
https://cloud.google.com/load-balancing/docs/url-map-concepts

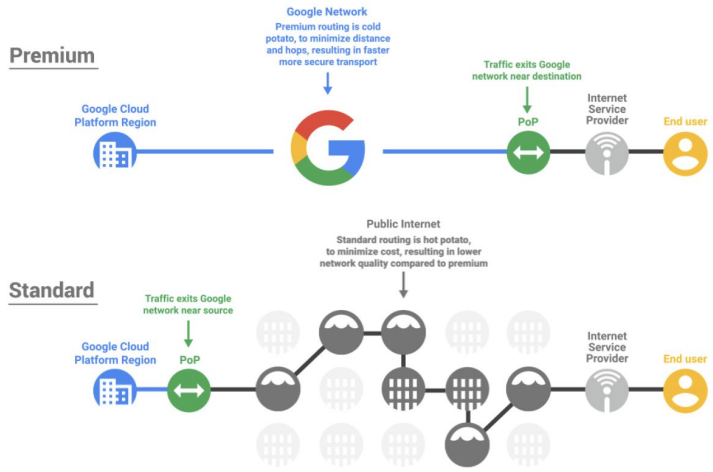
Setting up Routing rules

- Paths determines which backend service receives the traffic



URL maps overview
https://cloud.google.com/load-balancing/docs/url-map-concepts

Network Service Tiers



Premium Tier
https://cloud.google.com/network-tiers/docs/overview#premium_tier

Premium Tier delivers traffic from external systems to Google Cloud resources by using Google's low latency, highly reliable global network. This network consists of an extensive private fiber network with over 100 points of presence (PoPs) around the globe. This network is designed to tolerate multiple failures and disruptions while still delivering traffic.

Premium Tier supports both regional external IP addresses and global external IP addresses for VM instances and load balancers. All global external IP addresses must use Premium Tier. Applications that require high performance and availability, such as those that use HTTP(S), TCP proxy, and SSL proxy load balancers with backends in more than one region, require Premium Tier. Premium Tier is ideal for customers with users in multiple locations worldwide who need the best network performance and reliability.

Standard Tier
https://cloud.google.com/network-tiers/docs/overview#standard_tier

Standard Tier delivers traffic from external systems to Google Cloud resources by routing it over the internet. It leverages the double redundancy of Google's network only up to the point where Google's data center connects to a peering PoP. Packets that leave Google's network are delivered using the public internet and are subject to the reliability of intervening transit providers and ISPs. Standard Tier provides network

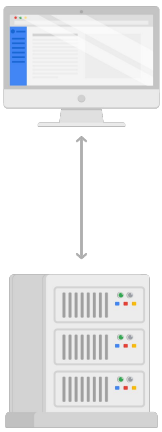
quality and reliability comparable to that of other cloud providers.

Standard Tier is priced lower than Premium Tier because traffic from systems on the internet is routed over transit (ISP) networks before being sent to VMs in your VPC network or regional Cloud Storage buckets. Standard Tier outbound traffic normally exits Google's network from the same region used by the sending VM or Cloud Storage bucket, regardless of its destination. In rare cases, such as during a network event, traffic might not be able to travel out the closest exit and might be sent out another exit, perhaps in another region.

Standard Tier offers a lower-cost alternative for the following use cases:

- You have applications that are not latency or performance sensitive.
- You're deploying VM instances or using Cloud Storage that can all be within a single region.

DNS resolution for internal addresses



Each instance has a hostname that can be resolved to an internal IP address:

- The hostname is the same as the instance name
- FQDN is [hostname].c.[project-id].internal

Example: guestbook-test.c.guestbook-151617.internal

Name resolution is handled by internal DNS resolver:

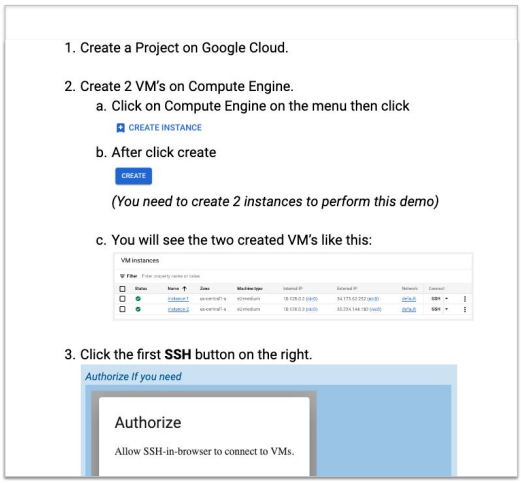
- Provided as part of Compute Engine (169.254.169.254)
- Configured for use on instance via DHCP
- Provides answer for internal and external addresses

Let's start with internal addresses.
Each instance has a hostname that can be resolved to an internal IP address. This hostname is the same as the instance name. There is also an internal fully qualified domain name (or FQDN) for an instance that uses the [hostname] .c format. [Project Id].internal as shown in the slide.
If you delete and recreate an instance, the internal IP address might change. This change can break connections to other Compute Engine resources, which must obtain the new IP address before connecting again. However, the DNS name always points to a specific instance, no matter what the internal IP address is.
Each instance has a metadata server that also acts as a DNS resolver for that instance. The metadata server handles all DNS queries for local network resources and routes all other queries to Google's public DNS servers for public name resolution. We mentioned earlier that an instance does not recognize any external IP addresses assigned to it. Instead, the network stores a lookup table that matches the external IP addresses with the internal IP addresses of the relevant instances.

For more information, including how to configure your own resolver on instances, see: https://cloud.google.com/compute/docs/vpc/internal-dns

Demo: MIG, Load Balancer and Routing

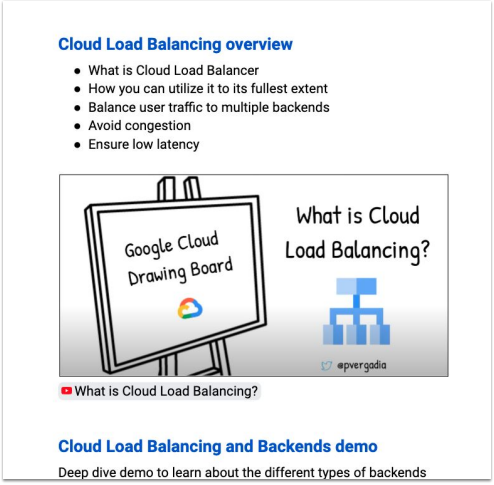
- In this demo, you will create a Managed Instance Groups and Load Balancers to point to a specific routes on your web application.



Mig, Load Balancer and Routing

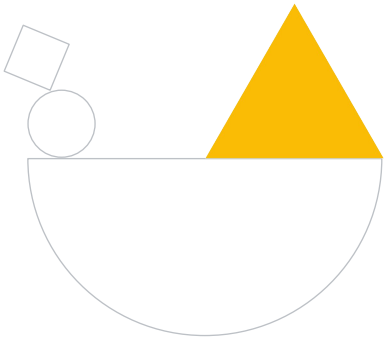
Guide: Load Balancers

- What is Cloud Load Balancer
- How you can utilize it to its fullest extent
- Balance user traffic to multiple backends
- Avoid congestion
- Ensure low latency
- Demos
- Labs



Cloud Load Balancing

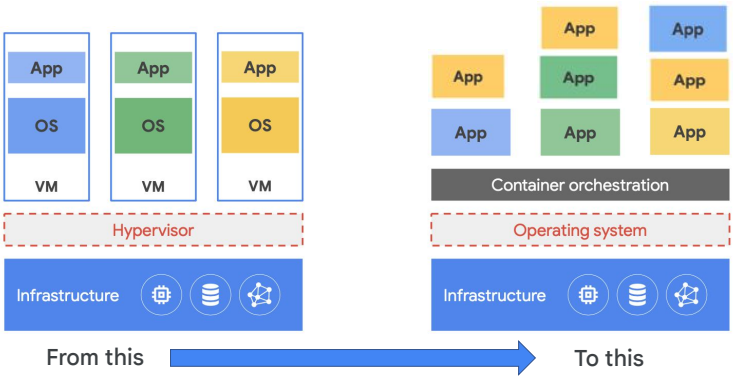
Cloud Run



Containers are an approach to app isolation

Containers typically deployed on VMs

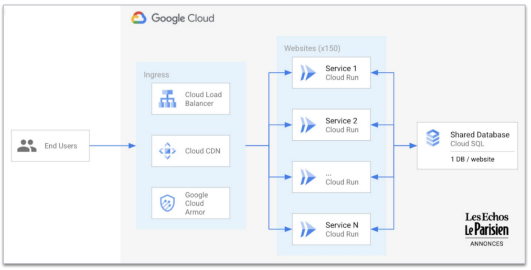
- Allows better utilization of VMs resources
- A VM that ran one application before can now run many containers



Google Cloud

Cloud Run provides Containers-as-a-Service

- Fully managed serverless platform that runs individual containers.
- Based on the open-source knative project (https://knative.dev/)
- Two deployment methods
  - Fully managed
    - Autoscaling, connectivity managed by Google
  - Cloud Run for Anthos
    - Knative running on a GKE cluster
      - Allows you to:
        - Access VPC network
        - Tune size of compute engine instance, use GPUs, etc.
        - Run service in all GKE regions

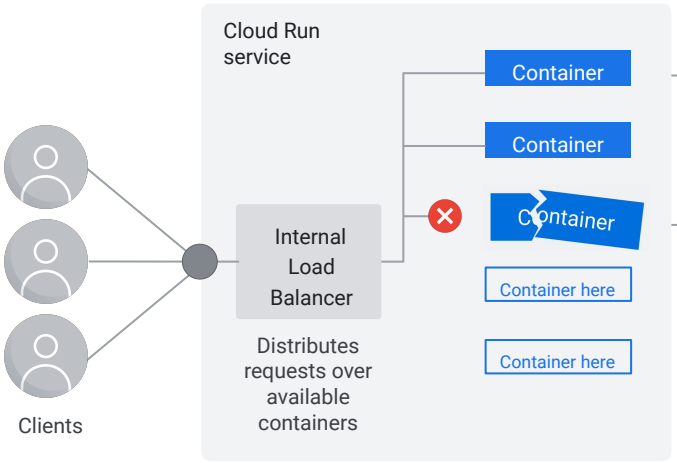


Scaling quickly to new markets with Cloud Run—a web modernization story

Cloud Run
https://cloud.google.com/run

Knative
https://knative.dev/

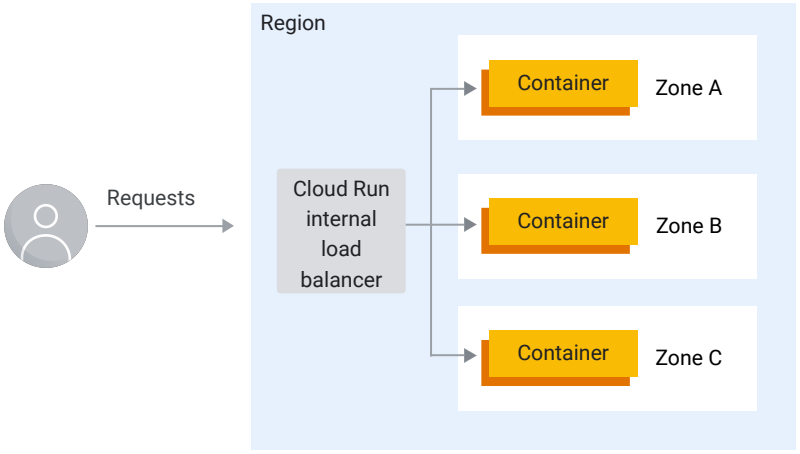
All incoming requests are handled with automatic scaling



Auto scaling: Add containers to handle requests and remove them if they are unused

Auto healing: Monitor and replace unhealthy containers

Cloud Run balances containers across zones in a region

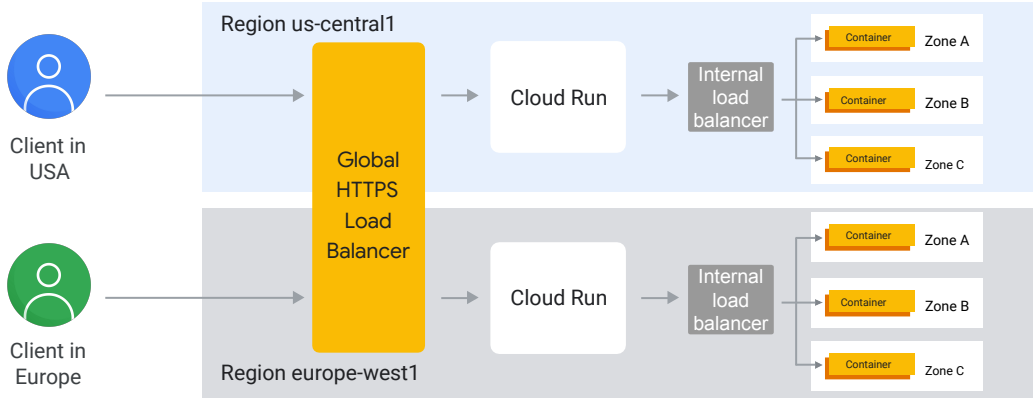


A region is a data center. For example: Council Bluffs (Iowa, North America)

Every region has three or more zones.

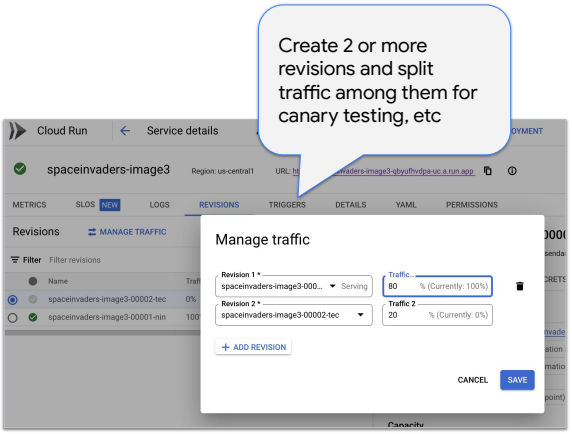
It's unlikely for a zone to go down, and a multi-zone failure is highly unlikely.

Global load balancer delivers lowest end-user latency



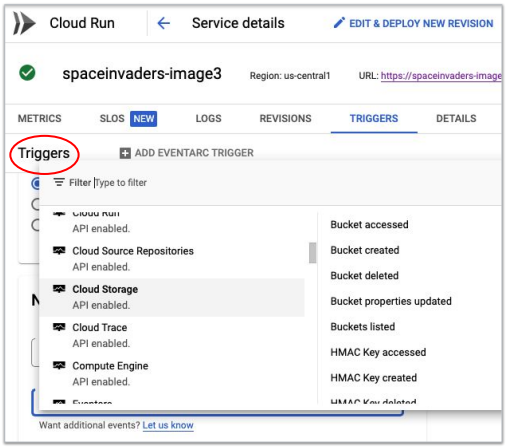
Cloud Run traffic splitting

- Split traffic across two or more revision
  - Great for canary deployments
  - Gradual rollouts for revisions
- Easily roll back to a previous revision



Invoking Cloud Run

- Can be invoked by
  - HTTPS endpoint
  - gRPC
  - Websockets
  - Pub/Sub
  - Eventarc triggers
    - Cloud Storage
    - BigQuery
    - And more

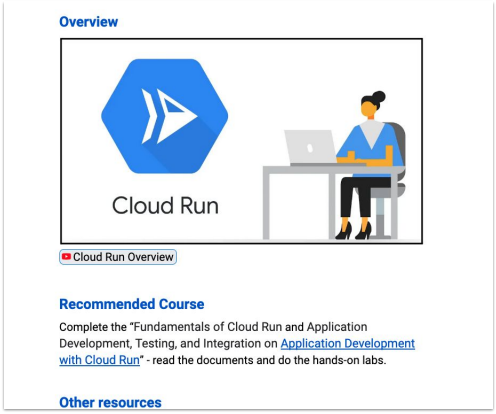


Eventarc overview
https://cloud.google.com/eventarc/docs/overview

Guide: Cloud Run

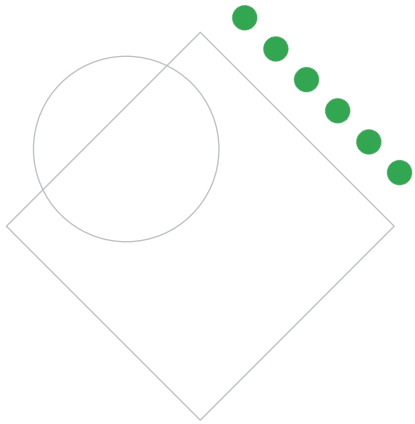
Topics covered

- Cloud Run
- Containers
- Provisioning and Scaling
- Event-driven http invocations
- Serveless containers



Cloud Run

GKE



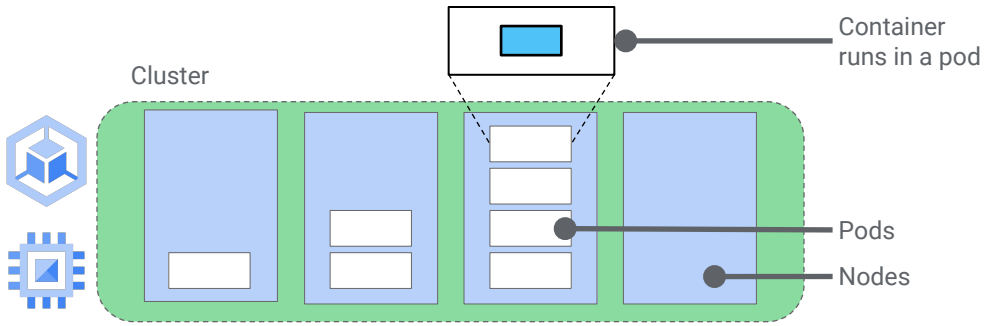
The word Kubernetes comes from the Greek word for helmsman or pilot

- Just like a cargo ship, a VM (aka “node” in Kubernetes) can host multiple containers
- Each container is independent of the others
  - May have multiples of the same container running depending on the use case
    - Number can be scaled up/down as needed

Photo by Ian Taylor on Unsplash

Google Cloud

Kubernetes cluster has nodes, pods, and containers



Cluster nodes Auto-upgrading
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-upgrades

Cluster nodes Auto-repair
https://cloud.google.com/kubernetes-engine/docs/how-to/node-auto-repair

A Kubernetes cluster is composed of nodes, which are a unit of hardware resources. Nodes in GKE are implemented as VMs in Compute Engine. Each node has pods. Pods are resource management units. A pod is how Kubernetes controls and manages resources needed by applications and how it executes code. Pods also give the system fine-grain control over scaling.

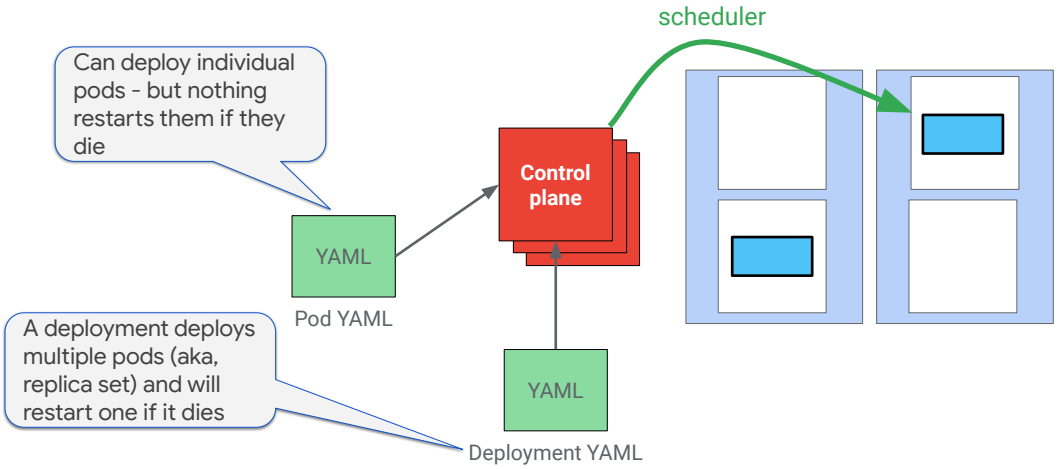
Each pod hosts, manages, and runs one or more containers. The containers in a pod share networking and storage.

So typically, there is one container per pod, unless the containers hold closely related applications. For example, a second container might contain the logging system for the application in the first container.

A pod can be moved from one node to another without reconfiguring or rebuilding anything.

This design enables advanced controls and operations that gives systems built on Kubernetes unique qualities.

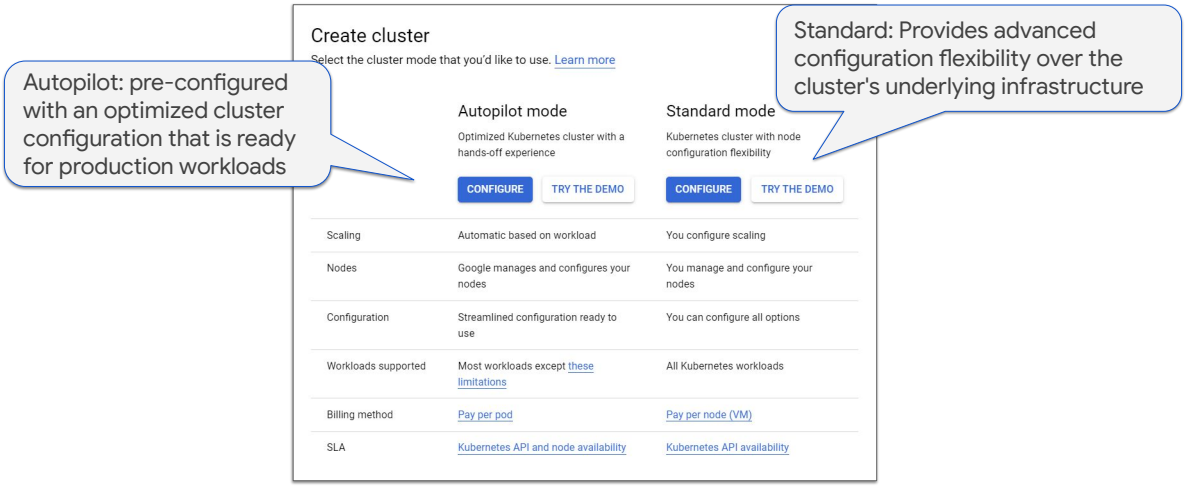
Kubernetes jobs run containers on nodes



Each cluster has a control plane node that determines what happens on the cluster. There are usually at least three of them for availability. And they can be located across zones. A Kubernetes job makes changes to the cluster.

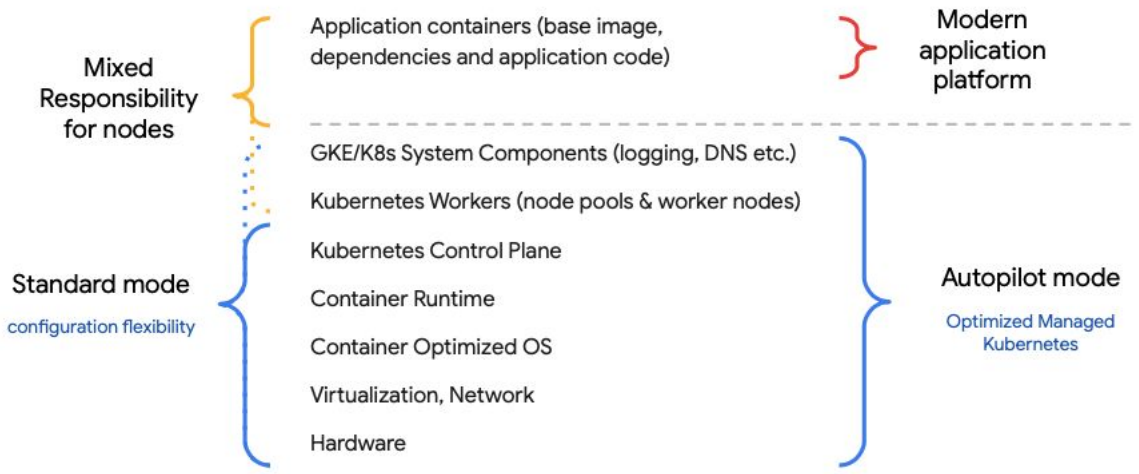
For example a pod YAML file provides the information to start up and run a pod on a node. If for some reason a pod stops running or a node is lost, the pod will not automatically be replaced. The Deployment YAML tells Kubernetes how many pods you want running. So the Kubernetes deployment is what keeps a number of pods running. The Deployment YAML also defines a Replica Set, which is how many copies of a container you want running. The Kubernetes scheduler determines on which node and in which pod the replica containers are to be run.

Creating Kubernetes Engine Clusters



Types of clusters (Standard vs Autopilot, Regional vs Zonal)
https://cloud.google.com/kubernetes-engine/docs/concepts/types-of-cluster

GKE Autopilot mode - Shared Responsibility model



AutoPilot
https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview

Overview of deploying workloads
https://cloud.google.com/kubernetes-engine/docs/how-to/deploying-workloads-overview

The illustration is an example of a deployment configuration.

The top portion of the bolded text depicts setting file as a deployment.

The middle bolded text section shows replicas set to 3. So this deployment will always have 3 pods running.

The bottom portion of the text sets the container image used for the pod.

Deploying to Kubernetes via command line

Create cluster

```
$ gcloud container clusters create si-cluster \
  --zone us-central1-a --machine-type=e2-micro \
  --num-nodes 2
```

Connect and apply yaml file

```
$ gcloud container clusters get-credentials si-cluster --zone us-central1-a
$ kubectl apply -f devops-deployment.yaml
```

Show the running pods

```
$ kubectl get pods
```

Show all the deployments

```
$ kubectl get deployments
```


gcloud container syntax
https://cloud.google.com/sdk/gcloud/reference/container

kubectl Cheat Sheet
https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Services allow communication to/from pods

- Pods in a deployment are regularly created and destroyed, causing their IP addresses to change constantly
  - Makes it difficult for frontend applications to identify which pods to connect to
- Services consist of
  - A set of pods
  - A policy to access them
- The most common types of Kubernetes services are
  - **ClusterIP**
  - **NodePort**
  - **LoadBalancer**
  - **Ingress (not really a service)**

Services
https://cloud.google.com/kubernetes-engine/docs/concepts/service

Another overview:
https://kubernetes.io/docs/concepts/services-networking/service/

Exposing applications using services (includes add, edit and remove):
https://cloud.google.com/kubernetes-engine/docs/how-to/exposing-apps

Service details

- **ClusterIP**
  - Default service
  - Internal to the cluster and allows applications within the pods to communicate with each other
- **NodePort**
  - Opens a specific port on each nodes in the cluster
  - Traffic sent to that port is forwarded to the pods
- **LoadBalancer**
  - Standard way to expose a Kubernetes service externally so it can be accessed over the internet
  - In GKE this creates a Network Load Balancer with one IP address accessible to external users
- **Ingress**
  - In GKE, the ingress controller creates an HTTP(S) Load Balancer, which can route traffic to services in the Kubernetes cluster based on path or subdomain

Summary of Kubernetes Terms

- **Pods** are the smallest unit of deployment
  - Usually pods represent a single container
- **Clusters** are collections of machines that will run containers
  - Each machine is a node in the cluster
- **Replica sets** are used to create multiple instances of a pod
  - Guarantee pods are healthy and the right number exist
- **Load balancers** route requests to pods
- **Autoscalers** monitor load and create or delete pods
- **Deployments** are configurations that define service resources

A few terms to understand when working with Kubernetes.

Pods are the smallest unit of deployment. A pod can be comprised of one or more containers, but typically it is one pod per container.

Clusters are a collection of instances the containers can run on. Each instance is a node in a cluster.

Replica sets are used to create multiple instances of a pod. The replica sets can guarantee the desired number is always running and only healthy pods are used.

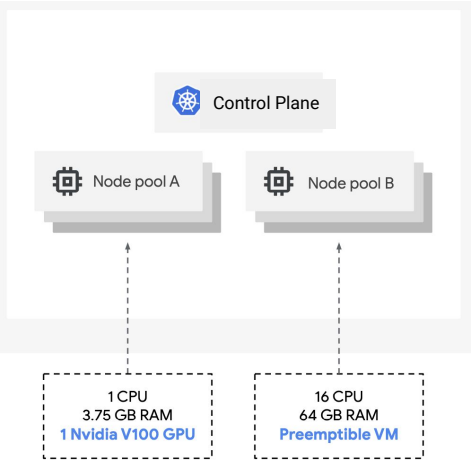
Kubernetes can leverage cloud load balancers to distribute traffic to multiple pods. The load balancer is run as a network service.

To ensure the proper performance of your application, you can configure autoscalers to add and remove pods.

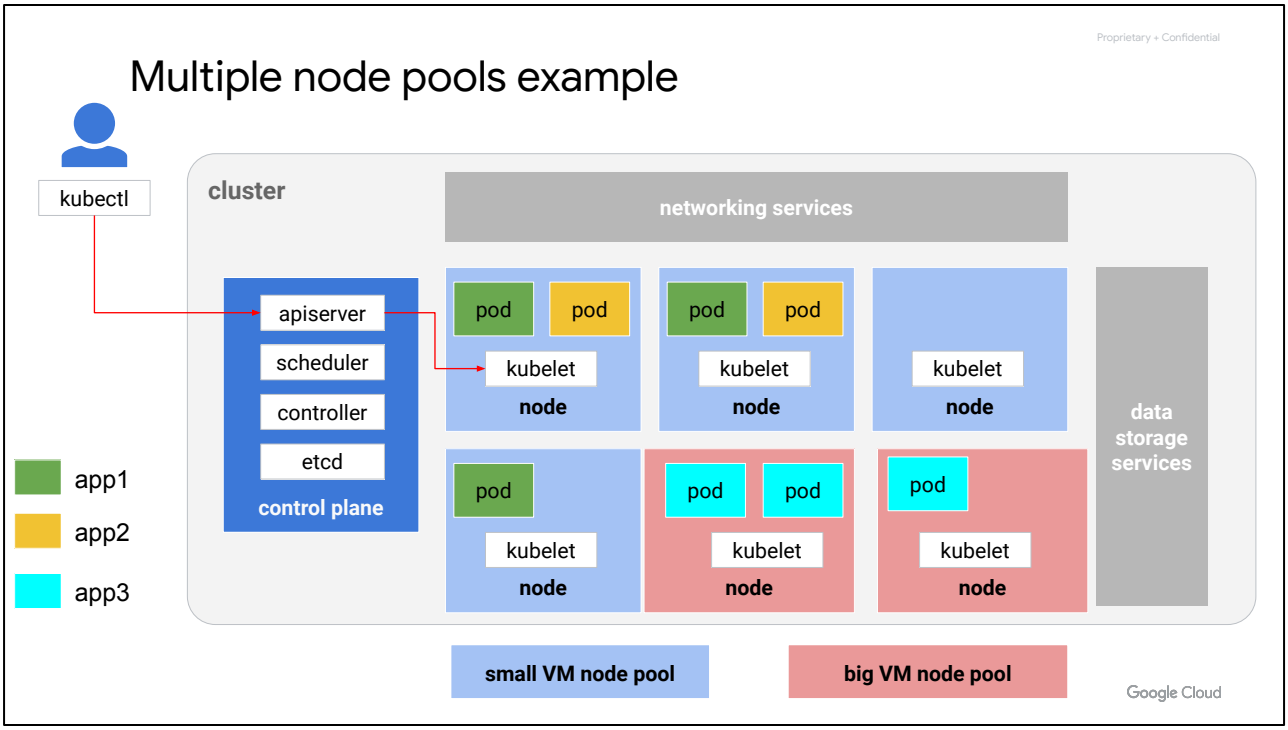
Deployments are configurations that define service resources.

Node pools are cluster subsets with identical machine configurations

- Share the same hardware, OS, and GKE version
  - Sizing, scaling, and upgrades, operate per pool
  - Can cover a full region – or just select zones
- Additional pools of different sizes and types can be added after cluster creation
  - For example, a pool with local SSDs, or Spot VMs, or a specific image or a different machine type
- Common configurations:
  - Each stateful application gets their own node pool
  - Batch jobs in a node pool with Preemptible VMs



About node pools
https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools



Here we have 2 node pools. The blue pool is running two apps and the red pool has one app deployed. Both pools are controlled by the same control plane.

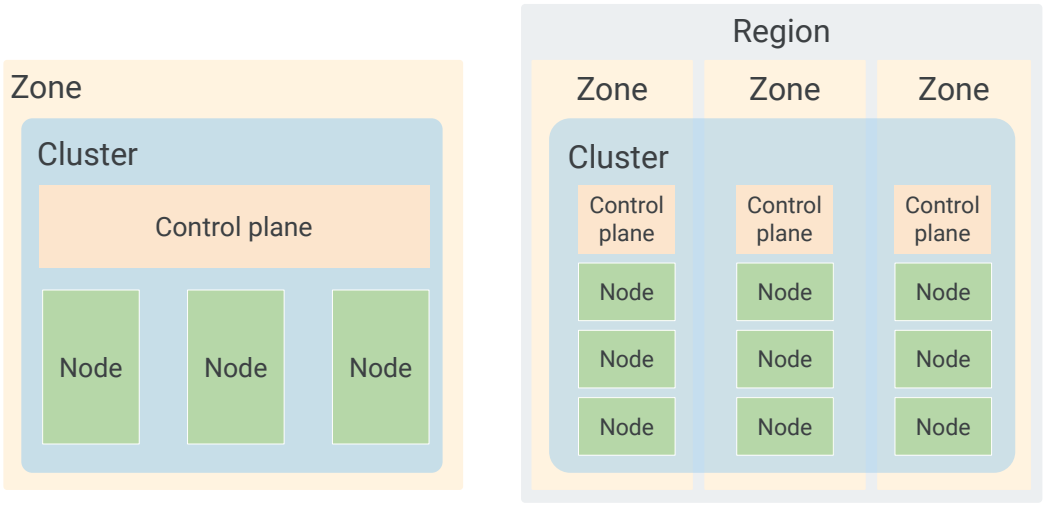
Cluster administrators configure the cluster by sending requests to **apiservers** on the Control Plane using a command-line tool called **kubectl**. Kubectl can be installed and run anywhere.

From there, the apiserver communicates with the cluster in two primary ways:

- To the **kubelet** process that runs on each node
- To any node, pod, or service through the apiserver's proxy functionality (not shown).

Then pods are started on various nodes. In this example, there are three types of pods running (shown in yellow, green and teal).

Zonal vs regional clusters



By default, a cluster launches in a single Google Cloud compute zone with three identical nodes, all in one node pool. The number of nodes can be changed during or after the creation of the cluster. Adding more nodes and deploying multiple replicas of an application will improve an application's availability. But only up to a point. What happens if the entire compute zone goes down?

You can address this concern by using a GKE regional cluster. Regional clusters have a single API endpoint for the cluster. However, its control planes and nodes are spread across multiple Compute Engine zones within a region.

Regional clusters ensure that the availability of the application is maintained across multiple zones in a single region. In addition, the availability of the control plane is also maintained so that both the application and management functionality can withstand the loss of one or more, but not all, zones. By default, a regional cluster is spread across 3 zones, each containing 1 control plane and 3 nodes. These numbers can be increased or decreased. For example, if you have five nodes in Zone 1, you will have exactly the same number of nodes in each of the other zones, for a total of 15 nodes. Once you build a zonal cluster, you can't convert it into a regional cluster, or vice versa.

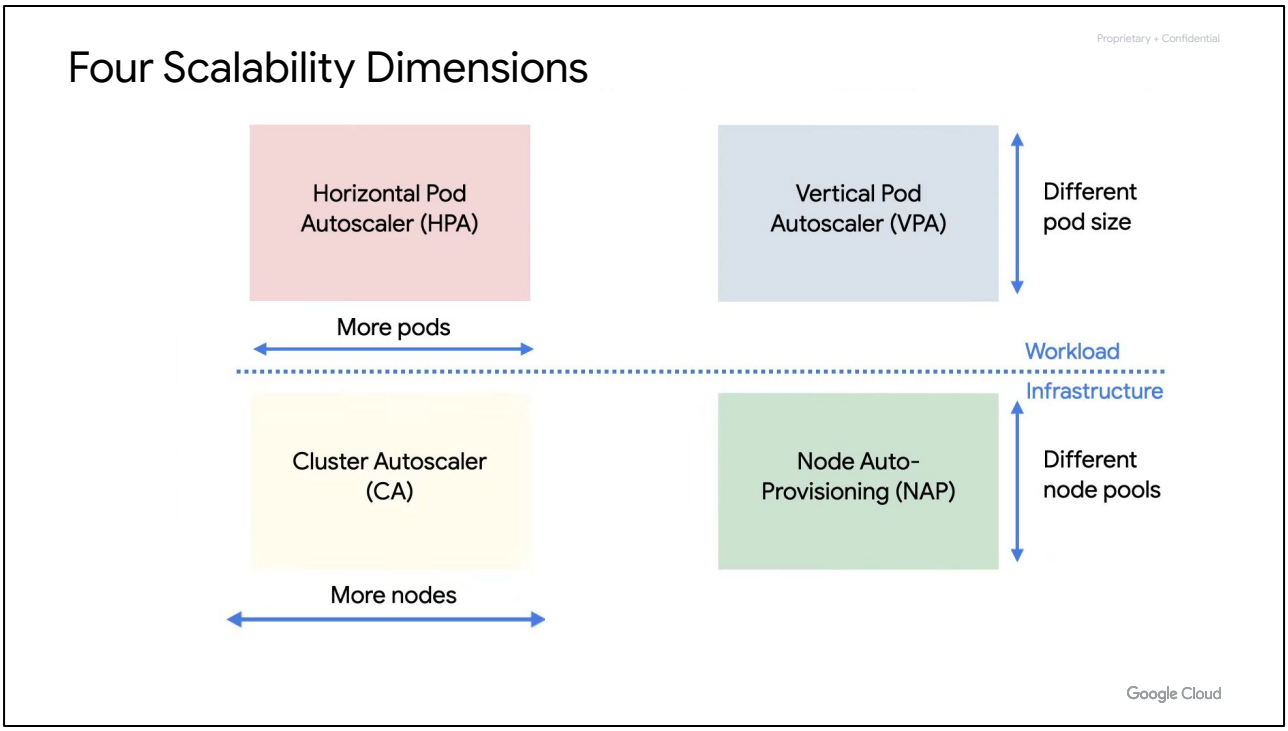
Autoscaling Strategies

Horizontal Pod Autoscaler (HPA)

Vertical Pod Autoscaler (VPA)

Cluster Autoscaler (CA)

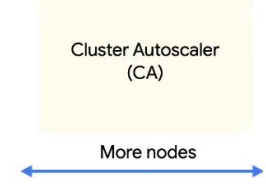
Node Auto Provisioning (NAP)



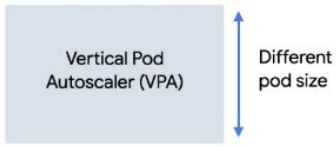
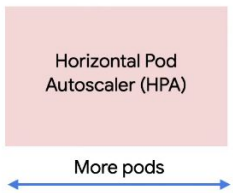
Configuring multidimensional Pod autoscaling
https://cloud.google.com/kubernetes-engine/docs/how-to/multidimensional-pod-autoscaling

Recommendations for planning, architecting, deploying, scaling, and operating large workloads on Google Kubernetes Engine (GKE) clusters
https://cloud.google.com/kubernetes-engine/docs/best-practices/scalability

Summary: Cluster Autoscaling vs Node Auto-provisioning

	Automatically resizes node pools based on the workload demands High demand: adds nodes to the node pool. Low demand: scales back down to a minimum size	Requires active monitoring of node resource usage to ensure haven't over/under provisioned node resources ● Under provisioned: Extra overhead of adding additional nodes when needed (+ node cost) ● Over provisioned: Paying too much per node
	Google optimizes the compute resources in each node pool to match workload requirements	● Not best for sudden spikes in traffic ● Will take longer to create a new node pool vs adding a node to an existing pool

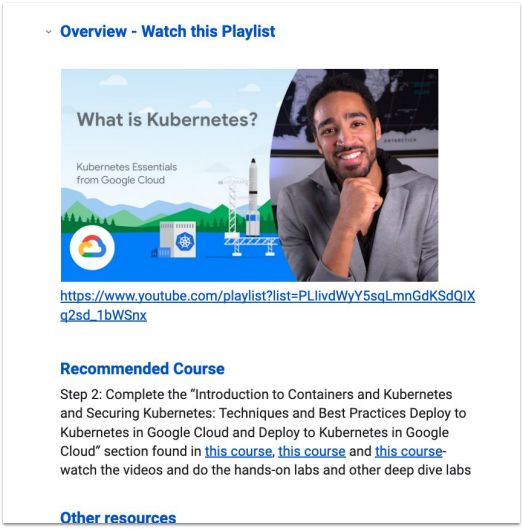
Summary: Vertical vs Horizontal Pod Autoscaling

	Use when are unsure of the optimal resource requests the container application needs Can use once or on a ongoing basis	If you do it manually: • Over-estimating memory/cpu results in over-provisioning the # of nodes needed when pods scale up, increasing costs • Under-estimating resources means that pods will be killed when the max limit is reached
	Use when have sudden spikes in traffic	Can use in conjunction with Vertical Pod Autoscaler to ensure pod resource requests are optimized

Guide: GKE

Topics covered

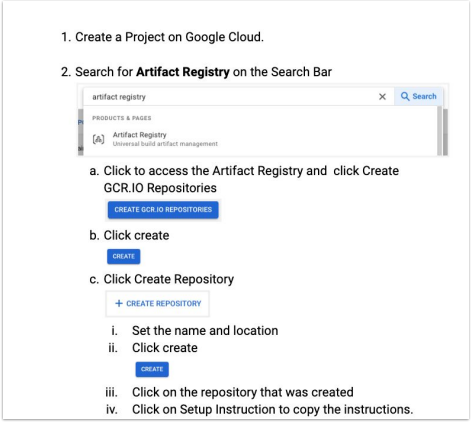
- Containers
- Kubernetes
- Orchestration
- Migrating a Monolithic to Microservices
- Deploy Apps on Google Kubernetes Engine
- Scale Apps on Google Kubernetes Engine



GKE

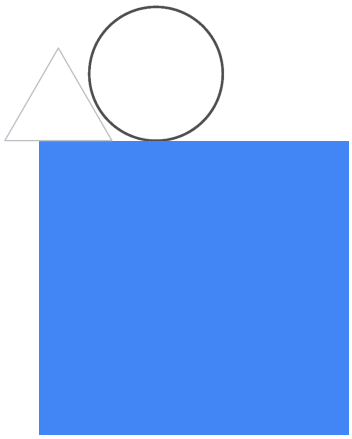
Demo: Container - Cloud Run, GCE and GKE

- In this demo you will learn to create container using Cloud Build and store this container image on Artifact Registry, then deploy on Cloud Run, Compute Engine and Kubernetes Engine.

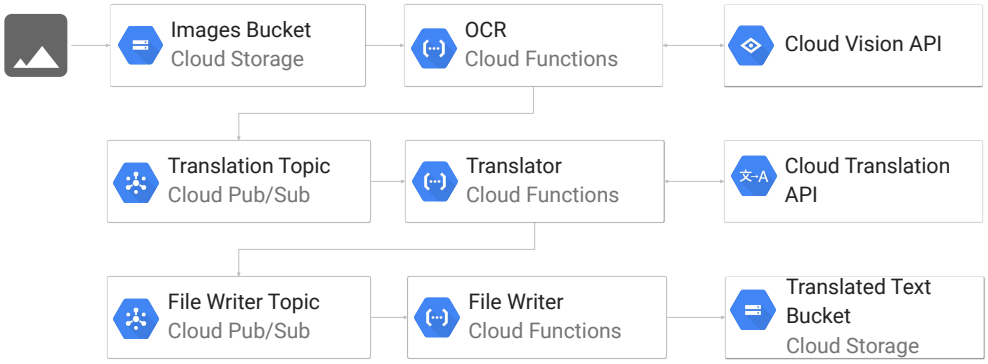


GKE

Cloud Functions



Develop event-driven, serverless, highly scalable microservices with Cloud Functions



With Cloud Functions, you can develop a highly scalable, serverless, event-driven application. Each role is a lightweight microservice that lets you integrate application components and data sources. Cloud Functions is ideal for microservices that require a small piece of code to quickly process data related to an event. Cloud Functions are billed based on how long the function runs, the number of times it is called, and the resources you provision for the function.

Consider an application that allows users to upload images that contain text to a bucket on Google Cloud Storage. The OCR Cloud Function is triggered when a new image is uploaded to the image bucket. This function uses the Cloud Vision API to extract text from images and then queues the text to Cloud Pub/Sub for translation. The Translator Cloud function, triggered by the Cloud Pub/Sub topic, invokes the Google Cloud Translation API to translate the text. It queues the translated text in the File Writer topic in Cloud Pub/Sub. File Writer Cloud Function writes the translated text for each image as separate files in Cloud Storage.

In this example, all application components use fully managed services and APIs such as Cloud Storage, Cloud Pub/Sub, Cloud Functions, Cloud Vision API, and Cloud Translation API. These services automatically scale depending on the amount of data received and the required computing resources. This scalability and reliability allows you to focus on your application code.

For more information about Cloud Functions, see https://cloud.google.com/functions/ .

Focus on code: Node.js and Google Cloud Client Libraries

index.js

```
/**
 * Triggered from a message on a Cloud Pub/Sub topic.
 *
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} The callback function.
 */
exports.subscribe = function subscribe(event, callback) {
  // The Cloud Pub/Sub Message object.
  const pubsubMessage = event.data;

  // We're just going to log the message to prove that
  // it worked.
  console.log(Buffer.from(pubsubMessage.data, 'base64').toString());

  // Don't forget to call the callback.
  callback();
};
```

package.json

```
{
  "name": "sample-pubsub",
  "version": "0.0.1"
}
```

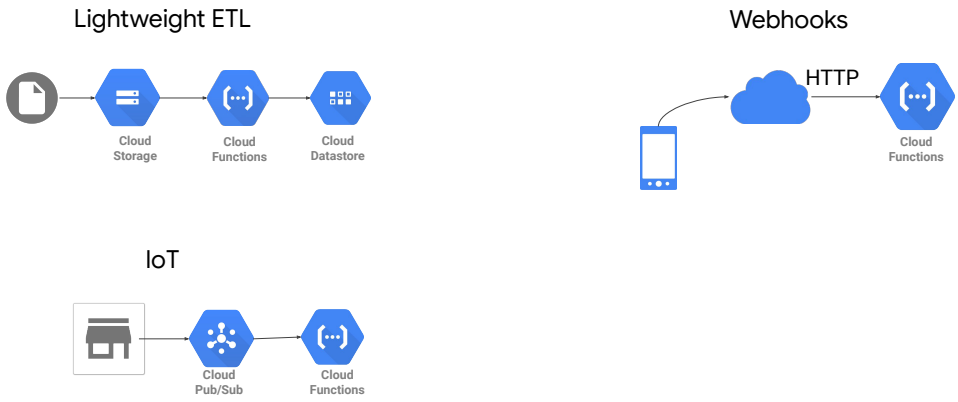

When using the Node.js runtime, your function's source code must be exported in a Node.js module.

You don't need to upload zip files with packaged dependencies. You can specify any dependencies for your Node.js cloud function in a package.json file. The Cloud Functions service automatically installs all dependencies before running your code.

You can use Google Cloud Client Libraries to programmatically interact with other Google Cloud Platform services.

Cloud Functions now supports several different Node.js runtime versions, as well as Python 3 and Go.

Use Cloud Functions to enable event-driven processing or to develop lightweight microservices



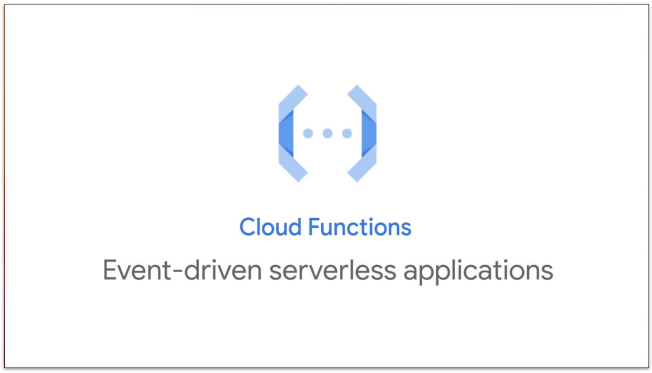
Cloud Functions can be used in a variety of use cases that require event-driven processing or lightweight microservices.

You can use Cloud Functions for lightweight extract-transform-load or ETL operations. For example, when a file is uploaded to Cloud Storage, a Cloud Function can be called to transform and upload the content to a database. You can also use Cloud Functions to process IoT streaming data or other application messages that are published to a Cloud Pub/Sub topic.

Cloud Functions can serve as a webhook. For example, you can set up a webhook that is automatically invoked after every commit to a Git repository. External and internal clients can make direct HTTP calls to invoke microservices that are deployed as Cloud Functions.

For more information about Cloud Functions, see https://cloud.google.com/functions/.

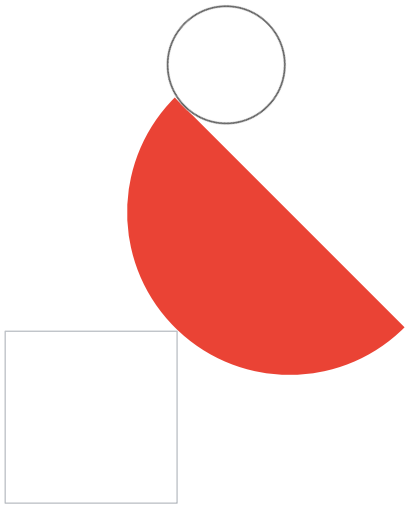
Recommended demo



In this video, we'll give you a quick tutorial of Cloud Functions, and show you how to deploy a Cloud Function from a Google Cloud project.

https://www.youtube.com/watch?v=vM-2O-uKBNQ

App Engine



App Engine is a PaaS for building scalable applications

- App Engine makes deployment, maintenance, and scalability easy so you can focus on innovation
- Especially suited for building scalable web applications and mobile backends

App Engine is a platform for building scalable web apps and mobile backends. It lets you focus on innovating your applications by managing the application infrastructure for you. For example, App Engine manages the hardware and network infrastructure needed to run your code.

App Engine provides built-in services and APIs such as NoSQL datastores, memcache, load balancing, health checking, application registration, and a user authentication API common to most applications.

App Engine will automatically scale your application in response to the amount of traffic it receives, so you only pay for the resources you use. Simply upload your code and Google will manage your app's availability. There are no servers for you to provision or maintain.

Security Scanner automatically scans and detects common web application vulnerabilities. It allows for early identification of threats and offers very low false positive rates. You can easily configure, run, schedule and manage security scans from the Google Cloud Platform Console.

App Engine works with popular development tools like Eclipse, IntelliJ, Maven, Git, Jenkins and PyCharm. You can build your apps with the tools you love, without changing your workflow.

App Engine standard environment

- Easily deploy your applications
- Autoscale workloads to meet demand
- Economical
  - Free daily quota
  - Usage based pricing
- SDKs for development, testing and deployment

App Engine's default environment is based on container instances running on Google's infrastructure. Containers are pre-configured with one of several available runtimes (Java 7, Python 2.7, Go, PHP and others). Each runtime also includes libraries that support standard App Engine APIs. For many applications, the standard environment runtimes and libraries may be all you need.

The App Engine standard environment makes it easy to build and deploy an application that runs reliably even under heavy load and with large amounts of data. Includes the following features:

- Persistent storage with queries, sorting, and transactions
- Autoscaling and load balancing
- Asynchronous task queues to perform work outside the scope of a request
- Scheduled tasks to trigger events at specific times or regular intervals
- Integration with other cloud services and Google APIs

App Engine standard environment: Requirements

- Specific versions of Java, Python, PHP, Go, Node.js, and Ruby are supported.
- Your application must conform to certain sandbox constraints dependant on runtime

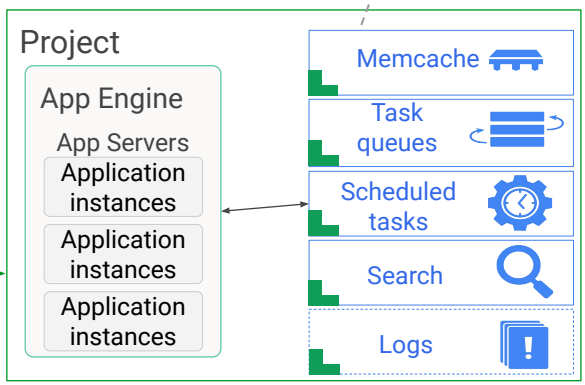
Example App Engine standard workflow: Web applications

1. Develop & test the web application locally

2. Use the SDK to deploy to App Engine

3. App Engine automatically scales & reliably serves your web application

App Engine can access a variety of services using dedicated APIs



App Engine flexible environment

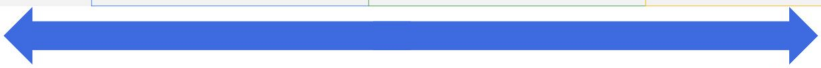
- Build and deploy containerized apps with a click.
- No sandbox constraints.
- Can access App Engine resources.
- Standard runtimes: Python, Java, Go, Node.js, PHP, .NET, and Ruby.
- Custom runtime support: Any language that supports HTTP requests.
- Package your runtime as a Dockerfile.

Comparing the App Engine environments

	Standard environment	Flexible environment
Instance startup	Seconds	Minutes
SSH access	No	Yes (although not by default)
Write to local disk	No (some runtimes have read and write access to the /tmp directory)	Yes, ephemeral (disk initialized on each VM startup)
Support for 3rd-party binaries	For certain languages	Yes
Network access	Via App Engine services	Yes
Pricing model	After free daily use, pay per instance class, with automatic shutdown	Pay for resource allocation per hour; no automatic shutdown

Deploying Apps: GKE vs App Engine

	Google Kubernetes Engine	App Engine flexible environment	App Engine standard environment
Language support	Any	Any	Specific versions of Python, Java, Node.js, Go, PHP, and Ruby
Service model	Hybrid	PaaS	PaaS
Primary use case	Container-based workloads	Web and mobile applications, container-based workloads	Web and mobile applications

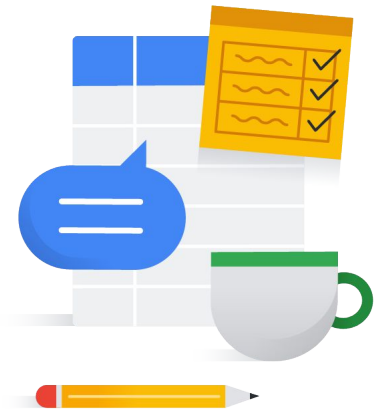


Toward managed infrastructure

Toward dynamic infrastructure

Program issues or concerns?

- Problems with **accessing** Cloud Skills Boost for Partners
  - partner-training@google.com
- Problems with **a lab** (locked out, etc.)
  - support@qwiklabs.com
- Problems with accessing Partner Advantage
  - https://support.google.com/googlecloud/topic/9198654




Google Cloud