Google Cloud

Professional Cloud Architect

Preparing for Professional Cloud Architect Journey for AWS Professionals

Plan:

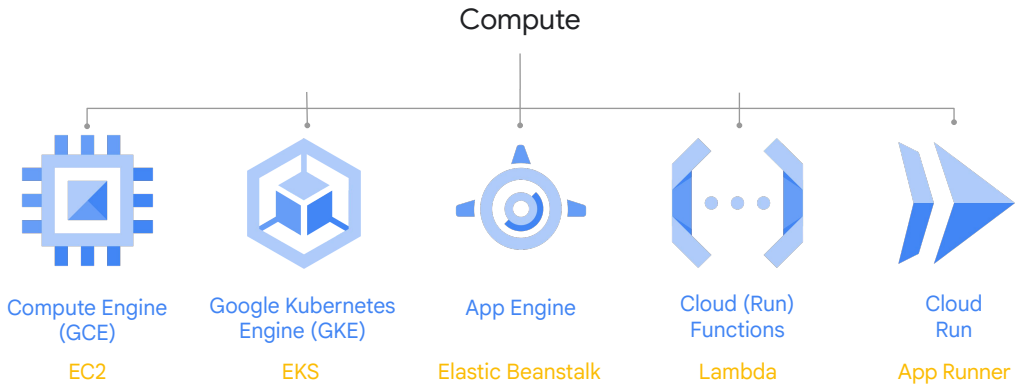
- 7 mins: Priyanka's slide
- 6 mins: VMs - talk, show shielded VMs etxc
- 5 mins: VM questions
- 5 mins: App Engine
- 5 mins: Cloud Run Functions (with "demo", show Eventarc)
- 10 mins: GKE (with Fleets and cluster creation, compare Autopilot and Standard)
- 5 mins: GKE questions
- 8 mins: Cloud Run with "demo" (show options)
- 3 mins: last diag question

Session 3 topics

Compute Options

1

Google Cloud offers a range of compute services



_____

GKE is best-in-class… (max cluster sizes, automation tools, startup times etc). Google also has a managed service for migrating containerized workloads to GKE.

_____

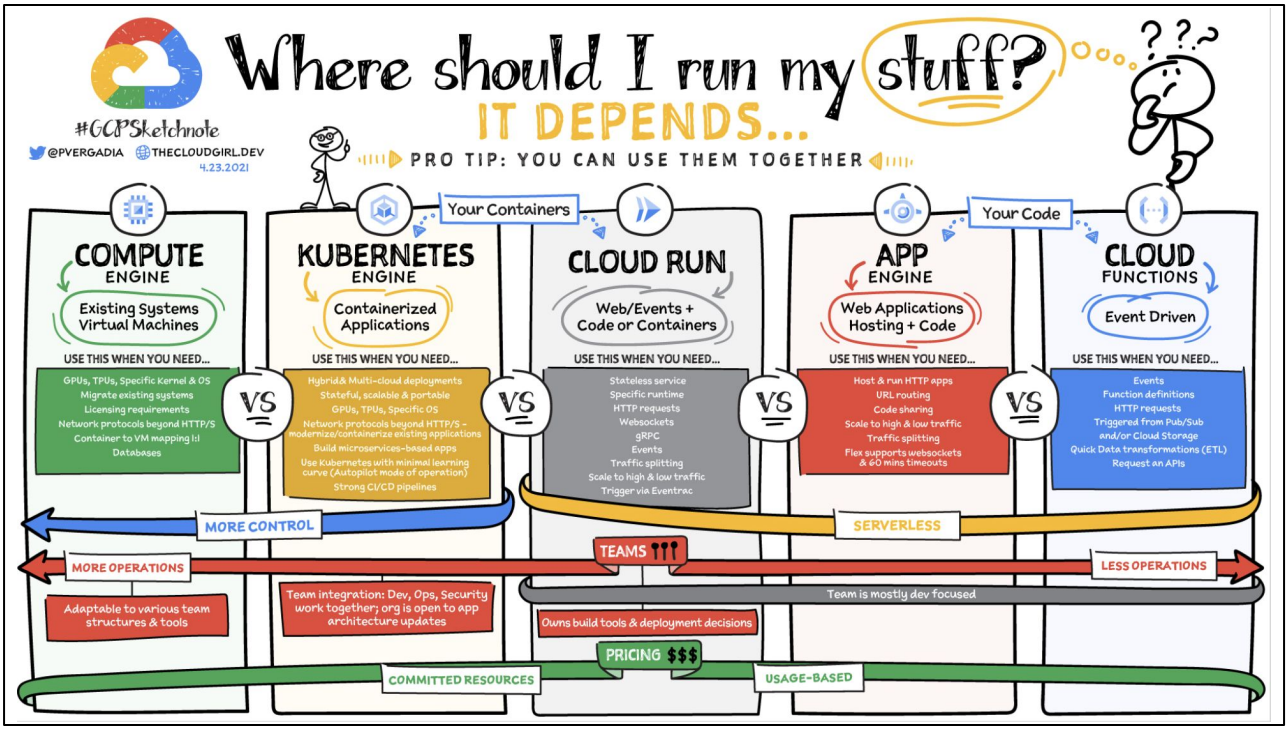
App Engine: Completely serverless compared to Elastic Beanstalk, and also very easy traffic splitting (important for deploying & testing new versions of applications)

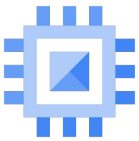
_____

AWS Fargate (vs Lambda vs App Runner):
https://nathanpeck.com/concurrency-compared-lambda-fargate-app-runner/
Key difference: concurrency. (Cloud Run concurrency: automatic, but we can define max number of concurrent requests per instance -> default 80, max 1000).
https://cloud.google.com/run/docs/about-concurrency

_____

Mention: "A major difference you'll notice with Google Cloud is that there is first-order support for dedicated *service accounts*. These are effectively users under which your (and Google Cloud's) programmatic APIs operate.". all environments in GCP will be working under a specific SA and hence would have different privileges.


#GCPSketchnote
@PVERGADIA
THECLOUDGIRL.DEV
4.23.2021
Where should I run my stuff?
IT DEPENDS...
PRO TIP: YOU CAN USE THEM TOGETHER
Your Containers
Your Code
COMPUTE ENGINE
Existing Systems Virtual Machines
USE THIS WHEN YOU NEED...
GPUs, TPUs, Specific Kernel & OS
Migrate existing systems
Licensing requirements
Network protocols beyond HTTP/S
Container to VM mapping !:!
Databases
VS
KUBERNETES ENGINE
Containerized Applications
USE THIS WHEN YOU NEED...
Hybrid& Multi-cloud deployments
Stateful, scalable & portable
GPUs, TPUs, Specific OS
Network protocols beyond HTTP/S - modernize/containerize existing applications
Build microservices-based apps
Use Kubernetes with minimal learning curve (Autopilot mode of operation)
Strong CI/CD pipelines
VS
CLOUD RUN
Web/Events + Code or Containers
USE THIS WHEN YOU NEED...
Stateless service
Specific runtime
HTTP requests
Websockets
gRPC
Events
Traffic splitting
Scale to high & low traffic
Trigger via Eventrac
VS
APP ENGINE
Web Applications Hosting + Code
USE THIS WHEN YOU NEED...
Host & run HTTP apps
URL routing
Code sharing
Scale to high & low traffic
Traffic splitting
Flex supports websockets & 60 mins timeouts
VS
CLOUD FUNCTIONS
Event Driven
USE THIS WHEN YOU NEED...
Events
Function definitions
HTTP requests
Triggered from Pub/Sub and/or Cloud Storage
Quick Data transformations (ETL)
Request an APIs
MORE CONTROL
SERVERLESS
MORE OPERATIONS
TEAMS
LESS OPERATIONS
Adaptable to various team structures & tools
Team integration: Dev, Ops, Security work together; org is open to app architecture updates
Owns build tools & deployment decisions
Team is mostly dev focused
PRICING $$$
COMMITTED RESOURCES
USAGE-BASED



Compute Engine (GCE)
EC2

Similarities between Amazon EC2 and Google Compute Engine instances

- Allow you to choose and configure RAM, CPU, and GPU
- Offer a wide range of operating systems
- Offer a high degree of flexibility and customization
- Offer the flexibility to add additional virtual disks
- Ephemeral and static (both public and private) IP addresses can be used
- Use metadata and scripts for bootstrapping
- Offer block storage options as part of their compute services
- Provide persistent disks
- Have several block storage types that cover a range of price and performance characteristics

Amazon EC2 instances and Google Compute Engine instances have a lot in common.

Both Google Compute Engine and Amazon EC2 instances allow you to choose and configure RAM, CPU, and GPU.

They offer a wide range of operating systems, a high degree of flexibility and customization and the flexibility to add additional virtual disks.

Ephemeral and static (both public and private) IP addresses can be used for both types of instances.

Both instance types can use metadata and scripts for bootstrapping.

Google Cloud and AWS both offer block storage options as part of their compute services. Both provide persistent disks. Each service has several block storage types that cover a range of price and performance characteristics.

In Google Cloud, regional persistent disks provide durable storage and replication of data between two zones in the same region. If you are designing robust systems on Compute Engine, consider using regional persistent disks to maintain high availability for resources across multiple zones.

Compute Engine (EC2)

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
- **OS Patch Management:** Patch all your machine at once
- **Rightsizing recommendations:** Optimize resource utilization with automatic recommendations
- **Per second billing**

Amazon EC2 instances and Compute Engine instances have a lot in common.

VMs: Both platforms enable you to create VMs by configuring your CPU, RAM, and attached storage parameters. Both platforms also offer opportunities to reduce costs with reserved and spot instances, and both offer VPCs to isolate services securely. Google offers semi-automatic ways to migrate VM-based workloads to GCE (similar to AWS): https://cloud.google.com/migration-center/docs. Both services let you migrate TO a specific cloud, but not in 2 directions.

bullet
Both Compute Engine and Amazon EC2 instances allow you to choose and configure RAM, CPU, and GPU.

bullet
Both offer a wide range of operating systems.

bullet
Both offer a high degree of flexibility and customization.

bullet

Both offer the flexibility to add additional virtual disks.

bullet
Ephemeral and static (both public and private) IP addresses can be used for both types of instances.

bullet
Both instance types can use metadata and scripts for bootstrapping.

bullet
Google Cloud and AWS both offer block storage options as part of their compute services. Both provide persistent disks. Each service has several block storage types that cover a range of price and performance characteristics.

Compute Engine (EC2)

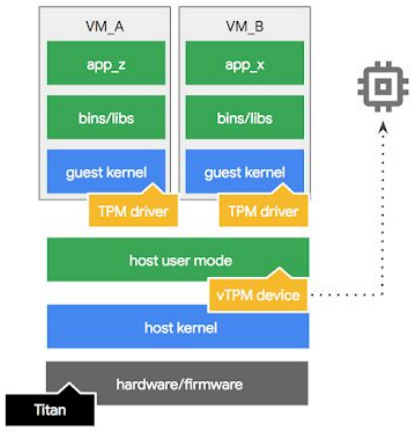
Virtual machines on Google's infrastructure

- **Shielded VMs:** VMs include a set of security controls that ensures your instances haven't been compromised by boot- or kernel-level malware or rootkits.
- **Sole Tenant Nodes:** A physical Compute Engine server that is dedicated to hosting only your project's VMs
- **Bare Metal Solutions:** Provides hardware to run specialized workloads with low latency
- **Confidential computing:** Encrypt your most sensitive data while it's being processed
- **VMware Engine:** Provides fully managed VMware foundation stack

— Shielded, Confidential and Sole-tenants covered on next slides

Shielded VM (AWS Nitro System)

- Instances firmware signed and verified using Google's Certificate Authority
- Ensures firmware is unmodified
- Establishes the root of trust for Secure Boot
- Unified Extensible Firmware Interface (UEFI) firmware securely stores the keys used by the software manufacturers to sign the system firmware, the system boot loader, and any binaries they load.
- Requirements:
  - Must use OS images with shielded VM features
    - Container-Optimized OS, Ubuntu, and Windows Server
    - CLI option: --image-project=gce-uefi-images
  - Custom images can be built from shielded VM images



Google Cloud

– “AWS Shield” is equivalent to Cloud Armor, NOT “shielded VMs)
– also, AWS (Nitro System / Nitro Enclaves) are rather an equivalent of Confidential VMs

https://sites.google.com/corp/google.com/shielded/vm

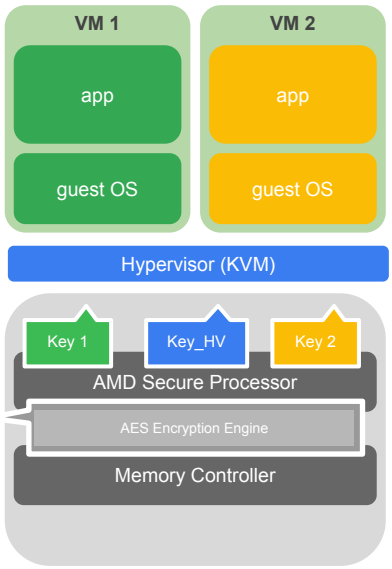
Shielded VM is the first offering in the Shielded Cloud initiative. The Shielded Cloud initiative is meant to provide an even more secure foundation for all of Google Cloud Platform (Google Cloud) by providing verifiable integrity and offering features, like vTPM shielding or sealing, that help prevent data exfiltration.

Protection:

- Malicious guest system firmware, including malicious drivers
- Malicious guest OS, including guest kernel
- Integrity notifications to protect against tampering

Confidential VMs (AWS Nitro Enclaves)

- Same as a regular Compute Engine VM
  - Anything that runs on VM runs on confidential VM
- Data encrypted while in use
  - Memory encrypted, decrypted only on CPU chip
  - A key per VM
    - Random, ephemeral, generated by HW
    - Not extractable from HW
- Scale up to 224 vCPUs and 896 GiB memory



Google Cloud

- encryption in use!

GCP confidential VMs leverage the Secure Encrypted Virtualization (SEV) feature of 2nd Gen AMD EPYC.

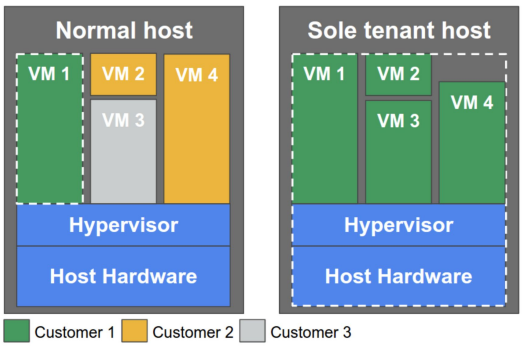
AWS took a different approach:

The AWS Nitro is made from Nitro Cards (to provision and manage compute, memory, and storage), Nitro Security Chip (the link between the CPU and the place where customer workloads run), and the Nitro Hypervisor

Sole-Tenant Nodes (Dedicated Instances)

Regular VMs on regular machines, dedicated specifically to your workloads.

- Dedicated hardware
- Mix-and-match VMs to consume host resources
- Full access to host resources for 10% premium*



Exam Tips: *More info on provisioning VMs on sole-tenant nodes can be found here.*

*10% Premium based on on-demand price

It essentially means you are renting physical servers entirely dedicated to your organization.
In GCP, Sole-Tenant Nodes provide physical Compute Engine server hardware that's dedicated to you. You have exclusive use of the physical server and can launch multiple VM instances on that dedicated hardware.

GCP vs AWS -> a bit different. In GCP, you can deploy single sole-tenant nodes, vs in AWS Dedicated Instances are physically isolated at the host hardware level from instances that belong to other AWS accounts. In GCP, you have more fine-grained control over placement, including the ability to specify which VMs go on which nodes within a node group = in GCP, you “reserve” whole machine, and then deploy VMs onto it. So AWS offers a simpler model with less configuration overhead, but also less control
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/dedicated-instance.html ->

[You probably already learned about Sole-Tenant Nodes. In short, they give you possibility to fulfill some compliance regulations or be aligned

with licencing models based on CPUs by making sure that you're the only customer using a dedicated hardware to deploy your VMs within GCP. Let me just mention I've seen much more detailed questions about Sole Tenant nodes on the exam, which require you to know quite a bit from technical perspective…

Compute Engine Machine Types

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

Google Cloud

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

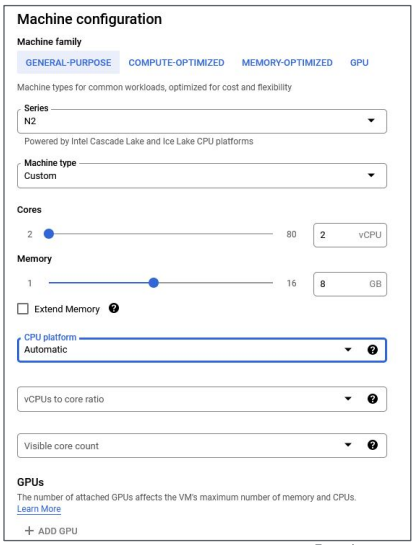
Creating custom machine types in Google Cloud

When to select custom:

- Requirements fit between the predefined types
- Need more or less memory or CPU than predefined options
- Using N or E machine families

Customize the amount of memory and vCPU for your machine:

- Either 1 vCPU or even number of vCPUs
- Can enable or disable hyperthreading
- Memory per vCPU varies by machine type (0.5 to 8 GB) - check Extend to use up to full RAM of host



Google Cloud

Source: Architecting with Compute Engine slides

If none of the predefined machine types match your needs, you can independently specify the number of vCPUs and the amount of memory for your instance. Custom machine types are ideal for the following scenarios:

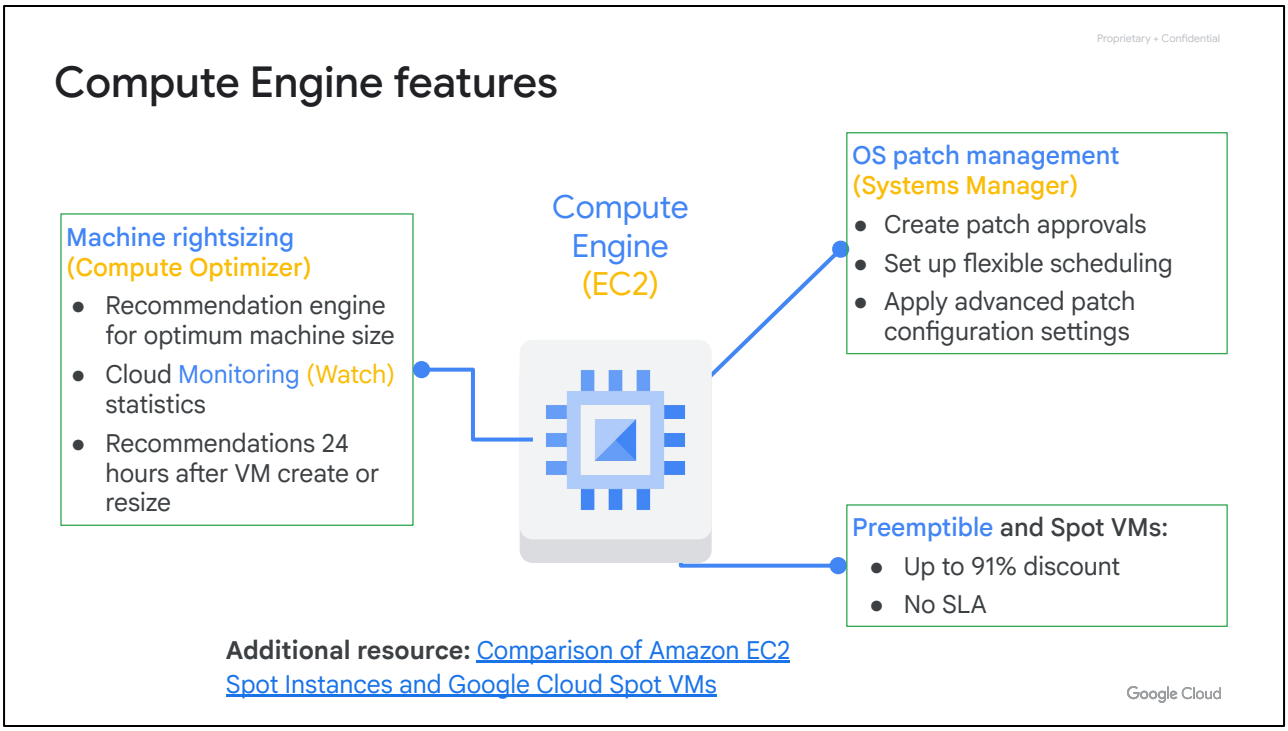
- When you have workloads that are not a good fit for the predefined machine types that are available to you.
- Or when you have workloads that require more processing power or more memory, but don't need all of the upgrades that are provided by the next larger predefined machine type.

It costs slightly more to use a custom machine type than an equivalent predefined machine type, and there are still some limitations in the amount of memory and vCPUs you can select:

- Only machine types with 1 vCPU or an even number of vCPUs can be created.
- Memory must be between 0.9 GB and 6.5 GB per vCPU (by default).
- The total memory of the instance must be a multiple of 256 MB.

By default, a custom machine can have up to 6.5 GB of memory per vCPU. However, this might not be enough memory for your workload. At an additional cost, you can get

more memory per vCPU beyond the 6.5 GB limit. This is referred to as *extended memory,* and you can learn more about this on this documentation page.



Spot (Preemptible was v1): Due to the short-lived nature of spot instances, they were traditionally used for applications that are fault tolerant or insensitive to disruption (for example, batch jobs). However, with automation and advanced cost optimization technology, it is possible to use spot instances even for mission critical applications.

AWS Spot Instances allow you to bid on spare Amazon EC2 computing capacity. Since the pricing is based on supply and demand, the costs can vary.
Azure: Instead of bidding, you simply set a maximum price (in USD) you're willing to pay per hour for the VM. If the spot price goes above your maximum price, your VM will be de-allocated.
GCP: discount of up to 91%. You don't bid; pricing adjustments are made no more than once a month, and guaranteeing a minimum discount of 60%. Spot VMs integrate with Google Kubernetes Engine (GKE), making them well-suited for containerized workloads.

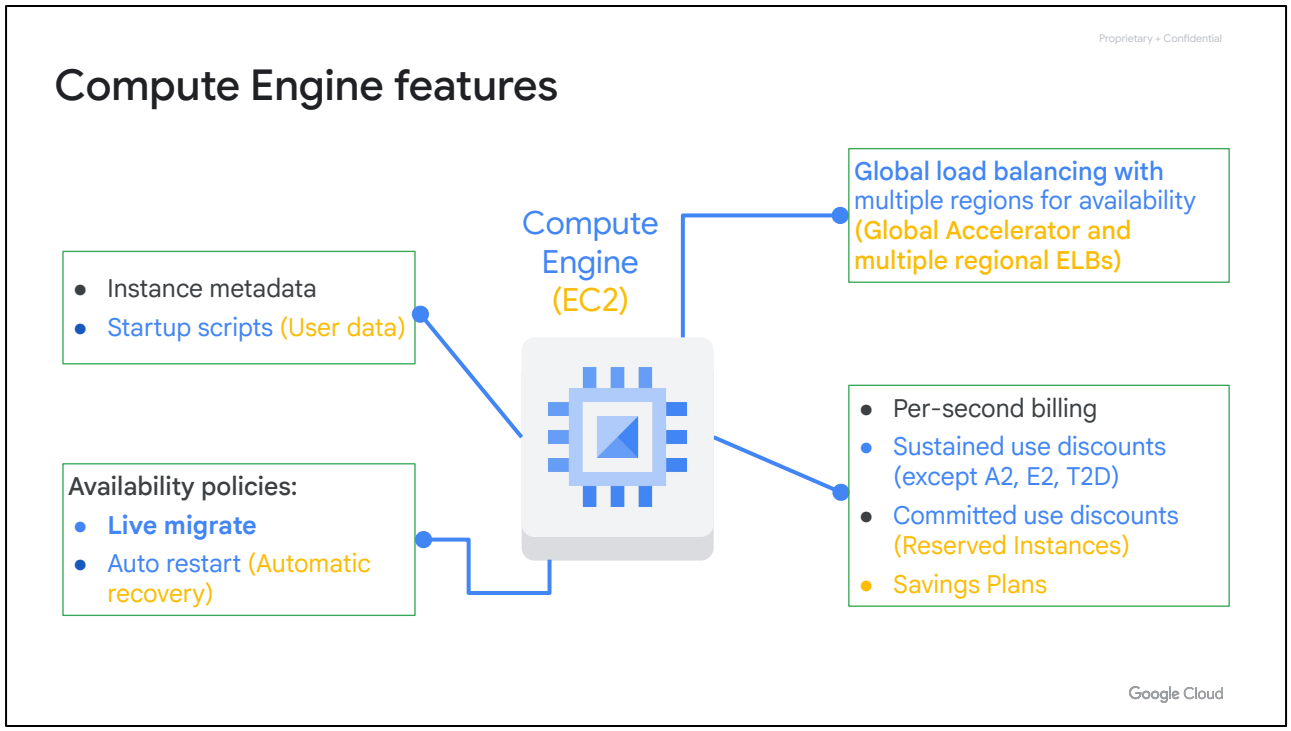
Source: Architecting with Compute Engine slides.

Speaker Notes:

OS Patch Management can be compared with AWS System Manager

Similar concept than on AWS - One important difference: In GCP you have the Hierarchical concept of Project. You can apply metadata at the project level so every instances in the project apply the metadata. For example: "enable-osconfig" set to "true" at the project level allows the usage of patch management for all instances in a project but it could also be applied to only specific instances.

Several different features will be covered throughout this module, such as machine rightsizing, startup scripts, metadata, availability policies, OS patch management, and pricing and usage discounts.



GCP handles 'startup scripts' differently than AWS, by running the script each reboot (as opposed to once: userdata).

+ Metadata Server: metadata implementation in GCP is quite different to EC2, and also more powerful. The most obvious difference is that GCE allows the definition of any arbitrary number of metadata fields, with any field names and values of your choosing. One very common use of multiple metadata values on is to configure the bootstrapping process of configuration management tools such as Ansible or Puppet. If you are using these tools to auto-configure a new instance, then they need instructions on where to fetch the configuration, and what type of instance should be figured.

AWS Comparison to “live migration”: AWS EC2 instances often require a reboot (or a stop and start, which is a form of downtime) for certain planned maintenance events. While AWS provides notification and tools to manage this, it still fundamentally requires an interruption to the VM's uptime.

Similar pricing options for VMs

Managing Compute Engine instances: Availability policy

Automatic restart

Crash or other maintenance event - your instance automatically restarts by default, but this can be changed.

Automatic restart does not apply to preemption or a user-initiated termination.

On host maintenance

This option determines whether the host is live-migrated or terminated due to a maintenance event. Live migration is the default behavior.

Live migration

During a maintenance event, a VM is migrated to different hardware without interruption.

Metadata indicates the occurrence of live migration.

Live Migration: AWS has historically taken a different philosophical approach, encouraging users to build resilient applications that can survive individual VM failures ("cattle, not pets"), rather than trying to keep every single VM alive forever.

- **Standard EC2:** For standard instances, AWS uses **Scheduled Events**. They notify you ahead of time that a niche needs maintenance. You must manually stop and start the instance (which moves it to healthy hardware) during a maintenance window, or AWS will eventually forcibly reboot it for you.
- **New Exception (Dedicated Hosts):** As of late 2024, AWS introduced live migration specifically for **EC2 Dedicated Hosts**. This brings them closer to parity with GCP/Azure features, but only for customers paying for dedicated physical hardware, not generic multi-tenant EC2 instances.

Availability policies

–

Availability policies can be configured using the Google Cloud console, gcloud CLI, or the API. The availability policy can be configured to live migrate if supported by the instance type. Live migration manages the move of a running VM to a new node without terminating the VM. Alternatively, the availability policy can be configured to terminate the VM and optionally automatically restarted.

A VM’s availability policy determines how the instance behaves during a maintenance event to prevent your applications from experiencing disruptions. These are known as "scheduling options" in SDK/API.

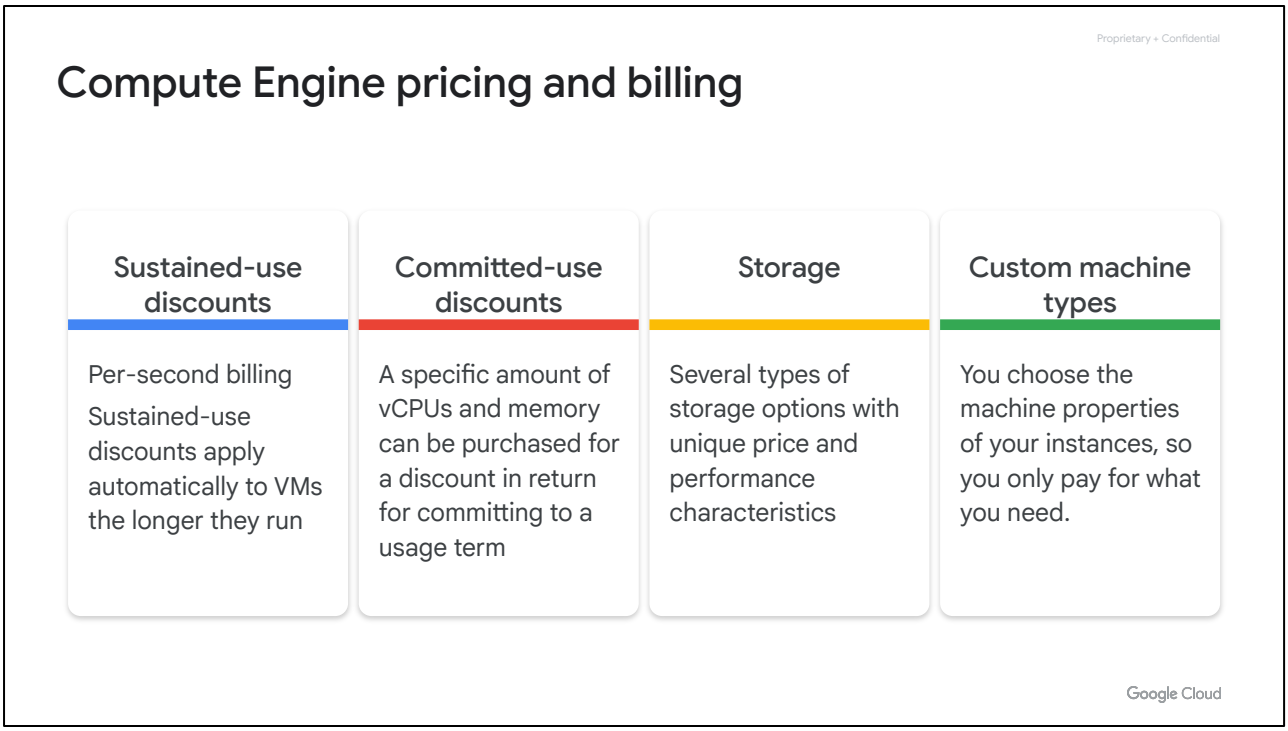
These availability policies can be configured both during the instance creation and while an instance is running by configuring the Automatic restart and On host maintenance options.

Let's explore the options.

The first option is automatic restart. If your VM is terminated due to a crash or other maintenance event, your instance automatically restarts by default, but this can be changed. This automatic restart does not apply to preemption or a user-initiated termination.

The second option is on host maintenance. This option determines whether the host is live-migrated or terminated due to a maintenance event. Live migration is the default behavior.

Finally, with live migration, during a maintenance event a VM is migrated to different hardware without interruption. Metadata indicates the occurrence of live migration.



Compute Engine offers a high degree of flexibility, and its pricing is designed so that you pay only for what you need in your virtual machines.

Let's examine some key aspects of Compute Engine pricing.

The first aspect is sustained-use discounts. For the use of virtual machines, Compute Engine bills by the second with a one-minute minimum.

Sustained-use discounts start to apply automatically to virtual machines the longer they run.

So, for each VM that runs for more than 25% of a month, Compute Engine automatically applies a discount for every additional minute.

Next is committed-use discounts.

For stable and predictable workloads, a specific amount of vCPUs and memory can be purchased for up to a 57% discount off of normal prices in return for committing to a usage term of one year or three years.

Another key aspect is storage.

Compute Engine doesn’t require a particular option or machine type to get high throughput between processing and persistent disks. That’s the default, and it comes

to you at no extra cost.

Compute Engine offers several types of storage options for your instances.

Each of the following storage options has unique price and performance characteristics:

- Zonal persistent disk offers efficient, reliable block storage.
- Regional persistent disk offers Regional block storage replicated in two zones.
- With Local SSD you get High performance, transient, local block storage.
- Cloud Storage buckets are for Affordable object storage.
- And Filestore is high performance file storage for Google Cloud users.

If you are not sure which option to use, the most common solution is to add a persistent disk to your instance.

And finally, there's also custom machine types. You only pay for what you need with custom machine types.

Compute Engine lets you choose the machine properties of your instances (like the number of virtual CPUs and the amount of memory) by using a set of predefined machine types or by creating your own custom machine types.

OS Images (AMI)

- Public base images*
  - Google, third-party vendors, and community; Premium images (p)
  - Linux
    - CentOS, CoreOS, Debian, RHEL(p), SUSE(p), Ubuntu, openSUSE, and FreeBSD
  - Windows
    - Windows Server 2022(p), 2019(p), 2016(p), 2012-r2(p) and more
    - SQL Server pre-installed on Windows(p)
- Custom images
  - Create new image from VM: pre-configured and installed SW
  - Import from on-prem, workstation, or another cloud
  - Management features: image sharing, image family, deprecation

*Check the website for an updated list of public images

AMIs can be imported to GCP:
https://cloud.google.com/compute/docs/import/import-aws-image

---

Images
https://cloud.google.com/compute/docs/images

You can select either a public or custom image.

You can choose from both Linux and Windows images. Some of these images are premium images, as indicated in parentheses with a p. These images will have per-second charges after a 1-minute minimum, with the exception of SQL Server images, which are charged per minute after a 10-minute minimum. Premium image prices vary with the machine type. However, these prices are global and do not vary by region or zone.

You can also use custom images. For example, you can create and use a custom image by pre-installing software that's been authorized for your particular organization.

You also have the option of importing images from your on-premises or workstation,

or from another cloud provider. This is a no-cost service that is as simple as installing an agent, and we highly recommend that you look at it. You can also share custom images with anybody in your project or among other projects, too.

Differences between AMIs and Google Images

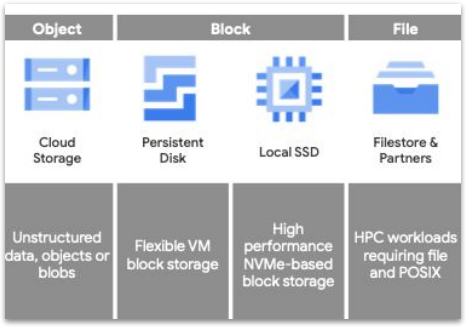
Feature	AMIs	Google images
Custom images	Custom AMI	Custom Images
Templates	Launch templates	Instance templates
Automated scaling up or down	Amazon EC2 Auto Scaling	Compute Engine Autoscaler
Automated scale out	Auto Scaling groups	Managed Instance Groups (MIGs)
VM imported file format	OVA, VMKD, VHD/VHDX, RAW	RAW **Additional resource:** Other support file formats

This table summarizes the differences between Google Images and what you find in AWS.

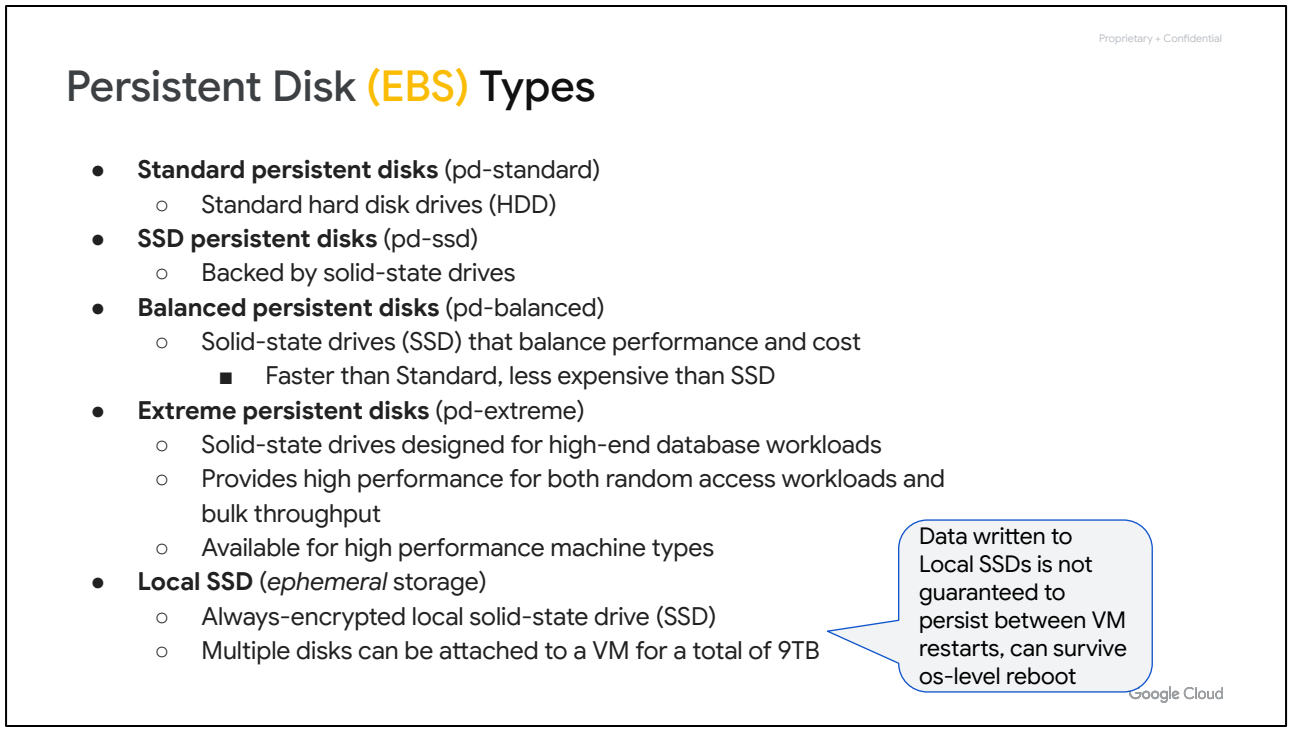
Refer to the additional resource “Other support file formats” to learn more about what VM imported file formats are supported.

Compute Engine Data Storage

- Each VM has a boot **Persistent Disk (Elastic Block Storage, EBS)** 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 (S3) buckets**: Affordable object storage
  - **Additional persistent disk(s)**: Efficient, reliable block storage.
  - **Local SSD (Ephemeral Storage)**: High performance, transient, local block storage.
  - **Filestore (EFS)**: High performance file storage for Google Cloud users.



– Data is ALWAYS encrypted in GCP!!!



Regional PDs: AWS does not currently offer a raw EBS block volume that is synchronously replicated across multiple Availability Zones (AZs) for you.

- **How it works:** Standard EBS volumes are strictly tied to one AZ. If that AZ fails, the data on that volume is inaccessible until the AZ recovers (unless you have snapshots).

In Google Cloud, regional persistent disks (no AWS equivalent) provide durable storage and replication of data between two zones in the same region. If you are designing robust systems on Compute Engine, consider using regional persistent disks to maintain high availability for resources across multiple zones.

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).

Filestore (EFS)

- 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 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

Snapshot storage

- Stored in Cloud Storage and have a choice of
  - Multi-regional location, such as Asia
    - Provides higher availability (99.95% SLA vs 99.9% for regional)
    - Disaster Recovery
    - Potentially slower snapshot restoration performance
  - Regional location, such as asia-south1
    - Use for compliance, e.g., GDPR
    - Use when all resources created from the snapshot will be in the same region - provides fastest restoration performance
- Network costs may be incurred when creating a disk from a snapshot
  - Multi-regional location storage
    - No network costs as long as the new persistent disk is created in one of the regions of the multi-regional group
  - Regional storage
    - *Will* incur network costs if the new disk is created in another region

PD snapshots => equivalent EBS snapshots. Both incremental forever, both stored in binary storage (GCS vs S3).
Difference: AWS snapshots are regional; GCP can be multi-regional / dual-region

Snapshot use cases

- Snapshots have many use cases
  - Can be used as source for a new disk
  - Can play a part in a disaster recovery plan
  - Can backup data
  - Can be used to move a VM to another zone/region
  - Can be used to migrate data from one disk type to another

Key differences: Compute Engine vs EC2

Part 1

Predefined instance configurations

Both offer a variety of predefined instance configurations with specific amounts of virtual CPU, RAM, and network.

- Amazon EC2: Instance types
- Compute Engine: Machine types

Machine images

Both use machine images to create new instances.

- AWS: AMIs
- Compute Engine: Images

Storage options

Both offer block storage options as part of their compute services.

- AWS: AWS EBS
- Compute Engine: Persistent disks

The first difference relates to Predefined instance configurations.

Compute Engine and Amazon EC2 both offer a variety of predefined instance configurations with specific amounts of virtual CPU, RAM, and network.

- Amazon EC2 refers to these configurations as instance types
- Compute Engine refers to them as machine types

When it come to machine images, Compute Engine and Amazon EC2 both use machine images to create new instances.

- AWS calls these images Amazon Machine Images (or AMIs)
- Compute Engine calls them images

For storage options, both offer block storage options as part of their compute services.

- AWS provides AWS Elastic Block Storage (or EBS)
- Compute Engine provides persistent disks

Key differences: Compute Engine vs EC2

Part 2

Local disks

- **AWS:** Instance stores, which are not adjustable
- **Compute Engine:** Local SSD, which can be attached to almost any machine type
  - Up to 24 can be attached to a single instance

Discount pricing

- **Amazon EC2:** Offers discounts for reserved instances, savings plans, and spot instances
- **Compute Engine:** Offers discounts for Preemptible or Spot VMs and Sustained- and Committed-use instances

Start time

- **Amazon EC2:** Instances have a wide range of start times depending upon the instance type and AMI
- **Compute Engine:** An instance typically takes about 30 seconds to start

The next difference relates to local disks.

- On AWS, local disks are called instance stores. The number and size of these disks depend on the specific instance type and are not adjustable.
- On Compute Engine, local disks are referred to as local SSD and can be attached to almost any machine type, and a maximum of 24 can be attached to a single instance as partitions for 9TB per instance. Refer to the additional resource “About local SSD disks” to learn more about this.

Compute Engine and Amazon EC2 approach discount pricing differently.

- Amazon EC2 offers discounts for reserved instances, savings plans, and spot instances.
- Compute Engine offers discounts for Preemptible or Spot VMs and Sustained- and Committed-use instances.

Finally, let look at start time.

- Amazon EC2 instances have a wide range of start times depending upon the instance type and AMI.
- A Compute Engine instance typically takes about 30 seconds to start.

Diagnostic Question Discussion

You want to create a number of spot Linux virtual machine instances using Google Compute Engine. You want to properly shut down your application before the virtual machines are preempted.

What should you do?

A. Create a shutdown script named k99.shutdown in the /etc/rc.6.d/ directory

B. Create a shutdown script registered as a xinetd service in Linux and configure a Cloud Monitoring endpoint check to call the service

C. Create a shutdown script and use it as the value for a new metadata entry with the key shutdown-script in the Cloud Platform Console when you create the new virtual machine instance

D. Create a shutdown script, registered as a xinetd service in Linux, and use the gcloud compute instances add-metadata command to specify the service URL as the value for a new metadata entry with the key shutdown-script-url

C

Diagnostic Question Discussion

You want to create a number of spot Linux virtual machine instances using Google Compute Engine. You want to properly shut down your application before the virtual machines are preempted.

What should you do?

A. Create a shutdown script named k99.shutdown in the /etc/rc.6.d/ directory

B. Create a shutdown script registered as a xinetd service in Linux and configure a Cloud Monitoring endpoint check to call the service

C. Create a shutdown script and use it as the value for a new metadata entry with the key shutdown-script in the Cloud Platform Console when you create the new virtual machine instance

D. Create a shutdown script, registered as a xinetd service in Linux, and use the gcloud compute instances add-metadata command to specify the service URL as the value for a new metadata entry with the key shutdown-script-url

C

Diagnostic Question Discussion

You need to deploy an application on Google Cloud that must run on a Debian Linux environment. The application requires extensive configuration in order to operate correctly. You want to ensure that you can install Debian distribution updates with minimal manual intervention whenever they become available.

What should you do?

A. Create a Compute Engine instance template using the most recent Debian image. Create an instance from this template, and install and configure the application as part of the startup script. Repeat this process whenever a new Google-managed Debian image becomes available.

B. Create a Debian-based Compute Engine instance, install and configure the application, and use OS patch management to install available updates.

C. Create an instance with the latest available Debian image. Connect to the instance via SSH, and install and configure the application on the instance. Repeat this process whenever a new Google-managed Debian image becomes available.

D. Create a Docker container with Debian as the base image. Install and configure the application as part of the Docker image creation process. Host the container on Google Kubernetes Engine and restart the container whenever a new update is available.

Google Cloud

B

Effectively, it's about OS patching. Automation is the optimal solution!

Eliminate D -> Docker container…
Eliminate A & C => too much work, not needed!

Diagnostic Question Discussion

You need to deploy an application on Google Cloud that must run on a Debian Linux environment. The application requires extensive configuration in order to operate correctly. You want to ensure that you can install Debian distribution updates with minimal manual intervention whenever they become available.

What should you do?

More about patch management here.

A. Create a Compute Engine instance template using the most recent Debian image. Create an instance from this template, and install and configure the application as part of the startup script. Repeat this process whenever a new Google-managed Debian image becomes available.

B. Create a Debian-based Compute Engine instance, install and configure the application, and use OS patch management to install available updates.

C. Create an instance with the latest available Debian image. Connect to the instance via SSH, and install and configure the application on the instance. Repeat this process whenever a new Google-managed Debian image becomes available.

D. Create a Docker container with Debian as the base image. Install and configure the application as part of the Docker image creation process. Host the container on Google Kubernetes Engine and restart the container whenever a new update is available.

Google Cloud

B

Effectively, it's about OS patching. Automation is the optimal solution!

Eliminate D -> Docker container…
Eliminate A & C => too much work, not needed!

Diagnostic Question Discussion

To reduce costs, the Director of Engineering has required all developers to move their development infrastructure resources from on-premises virtual machines
(VMs) to Google Cloud Platform. These resources go through multiple start/stop events during the day and require state to persist. You have been asked to design the process of running a development environment in Google Cloud while providing cost visibility to the finance department.

Which two steps should you take? (Choose two.)

A. Use persistent disks to store the state. Start and stop the VM as needed

B. Use the --auto-delete flag on all persistent disks and terminate the VM

C. Apply VM CPU utilization label and include it in the BigQuery billing export

D. Use Google BigQuery billing export and labels to associate cost to groups

E. Store all state into local SSD, snapshot the persistent disks, and terminate the VM

F. Store all state in Google Cloud Storage, snapshot the persistent disks, and terminate the VM

Google Cloud

Question with easy answers, but also suggests some very complex solution to distract you.

A + D

B -> does not make sense
C -> ???
E -> local SSDs -> data is lost when a VM goes down. We can snapshot, but it's very complex solution
F -> too complex.

Diagnostic Question Discussion

To reduce costs, the Director of Engineering has required all developers to move their development infrastructure resources from on-premises virtual machines
(VMs) to Google Cloud Platform. These resources go through multiple start/stop events during the day and require state to persist. You have been asked to design the process of running a development environment in Google Cloud while providing cost visibility to the finance department.

Which two steps should you take? (Choose two.)

A. Use persistent disks to store the state. Start and stop the VM as needed

B. Use the --auto-delete flag on all persistent disks and terminate the VM

C. Apply VM CPU utilization label and include it in the BigQuery billing export

D. Use Google BigQuery billing export and labels to associate cost to groups

E. Store all state into local SSD, snapshot the persistent disks, and terminate the VM

F. Store all state in Google Cloud Storage, snapshot the persistent disks, and terminate the VM

Google Cloud

Question with easy answers, but also suggests some very complex solution to distract you.

A + D

B -> does not make sense
C -> ???
E -> local SSDs -> data is lost when a VM goes down. We can snapshot, but it's very complex solution
F -> too complex.

Diagnostic Question Discussion

You need to deploy a stateful workload on Google Cloud. The workload can scale horizontally, but each instance needs to read and write to the same POSIX filesystem. At high load, the stateful workload needs to support up to 100 MB/s of writes.

What should you do?

A. Use a persistent disk for each instance.

B. Use a regional persistent disk for each instance.

C. Create a Cloud Filestore instance and mount it in each instance.

D. Create a Cloud Storage bucket and mount it in each instance using gcsfuse.

Google Cloud

Eliminate:
A -> PD cannot be mounted on many instances in R/W mode simultaneously
B -> only on 2 VMs, only 1 active
C -> OK
D -> not performant enough

Diagnostic Question Discussion

You need to deploy a stateful workload on Google Cloud. The workload can scale horizontally, but each instance needs to read and write to the same POSIX filesystem. At high load, the stateful workload needs to support up to 100 MB/s of writes.

What should you do?

A. Use a persistent disk for each instance.

B. Use a regional persistent disk for each instance.

C. Create a Cloud Filestore instance and mount it in each instance.

D. Create a Cloud Storage bucket and mount it in each instance using gcsfuse.

Google Cloud

Eliminate:
A -> PD cannot be mounted on many instances in R/W mode simultaneously
B -> only on 2 VMs, only 1 active
C -> OK
D -> not performant enough

Diagnostic Question Discussion

You need to host an application on a Compute Engine instance in a project shared with other teams. You want to prevent the other teams from accidentally causing downtime on that application.

What should you do?

A. Use a Shielded VM.

B. Use a Preemptible VM.

C. Use a sole-tenant node.

D. Enable deletion protection on the instance.

Google Cloud

– question which expects you to know all those different options

D

https://cloud.google.com/compute/docs/instances/preventing-accidental-vm-deletion

Diagnostic Question Discussion

You need to host an application on a Compute Engine instance in a project shared with other teams. You want to prevent the other teams from accidentally causing downtime on that application.

What should you do?

A. Use a Shielded VM.

B. Use a Preemptible VM.

C. Use a sole-tenant node.

D. Enable deletion protection on the instance.

https://cloud.google.com/compute/docs/instances/preventing-accidental-vm-deletion

Google Cloud

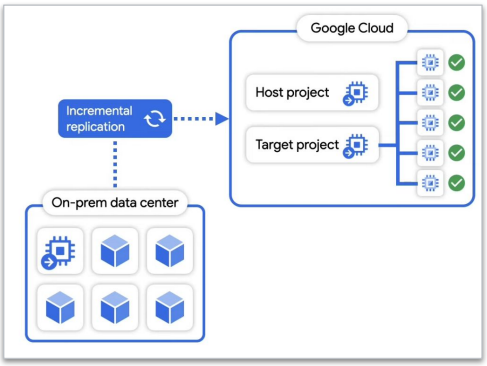
– question which expects you to know all those different options

D

https://cloud.google.com/compute/docs/instances/preventing-accidental-vm-deletion

Migrate to Virtual Machines (AWS Application Migration Service)

- Migrate VMs to Google Cloud Compute Engine directly from on-premise or AWS/Azure including support for customizations to networking, disks, and more
- 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



AWS:
https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/migrate-an-on-premises-vm-to-amazon-ec2-by-using-aws-application-migration-service.html

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 (AWS App2Container)

- 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

AWS:
https://aws.amazon.com/blogs/architecture/migrate-your-applications-to-containers-at-scale/
“AWS App2Container is a command line interface (CLI) tool for modernizing .NET and Java applications into containerized applications.” -> NOT as powerful / user-friendly as “Migrate to Containers”.

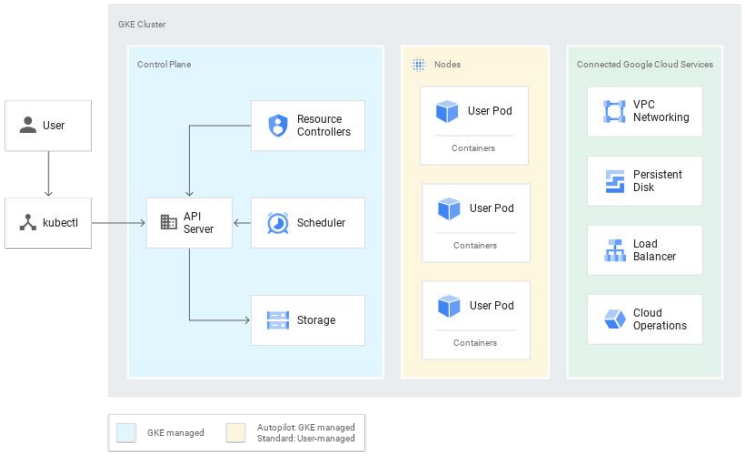
https://cloud.google.com/migrate/containers

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

Google Kubernetes Engine (GKE)

EKS

GKE cluster architecture



– intro to containers and Kubernetes skipped -> assuming everyone knows

Let’s take a look at the architecture of the Google Kubernetes Engine (GKE) clusters that run your containerized workloads

This diagram shows the architecture of a GKE cluster. This diagram specifically illustrates where resources exist and who manages those resources, rather than a depiction of the flow of traffic.

A GKE cluster consists of a control plane and worker machines called nodes.

The control plane and nodes make up the Kubernetes cluster orchestration system.

GKE Autopilot manages the entire underlying infrastructure of clusters, including the control plane, nodes, and all system components. If you use GKE Standard mode, GKE manages the control plane and system components, and you manage the nodes.

GKE and EKS: Similarities

GKE and EKS are both based on Kubernetes.

- Use a cluster of VMs to run and manage containers
- Use a cluster of VMs to run and manage containers
- Node agents are Kubelets
- Group containers by Pods
- Can run on a variety of clouds as well as physical hardware and VMs
- Support containerd container runtimes on Kubernetes 1.24 or later
- Support Docker container runtimes if running a Kubernetes version older than 1.24

GKE and Amazon Elastic Kubernetes Service (or EKS) are both based on Kubernetes, which was originally developed by Google to help manage its own infrastructure.

After Kubernetes was released as an open source project in 2014, the open source community adopted the technology and other cloud vendors, such as AWS, started to offer managed Kubernetes services in their own platforms.

Based on that history, it is no surprise that Google Cloud offers an extensive Kubernetes solution, including GKE Enterprise. If you are already using Kubernetes in EKS, the concepts and commands will be familiar to you in GKE.

Let's explore the similarities between the two services.

- Both EKS and GKE use a cluster of VMs to run and manage containers.
- Node agents for GKE and EKS are Kubelets, which is an open source agent used with Kubernetes.
- GKE and EKS both group containers by Pods, which is how Kubernetes runs applications.
- Both can run on a variety of clouds as well as physical hardware and VMs.
- Both EKS and GKE support only containerd container runtimes on Kubernetes 1.24 or later.
- Both EKS and GKE support Docker container runtimes if running a Kubernetes version older than 1.24.

GKE and EKS: Differences

Topic	EKS	GKE
Running commands	Done through a combination of kubectl, AWS CLI, and the Amazon EKS command line tool eksctl	Done through a combination of the open source kubectl command line and the gcloud CLI
Auto-repair nodes	EKS Auto Mode helps keep the nodes in your EKS cluster in a healthy, running state	Node auto-repair helps keep the nodes in your GKE cluster in a healthy, running state
Control plane upgrades	Node auto-upgrades keep the nodes in your cluster up-to-date with the cluster control plane version when your control plane is updated on your behalf	Node auto-upgrades keep the nodes in your cluster up-to-date with the cluster control plane version when your control plane is updated on your behalf

Additional resource: Comparison of EKS and GKE

Now let's examine how certain GKS and EKS features differ.

The first difference relates to running commands.

- Running commands in EKS can be done through a combination of kubectl, AWS CLI, and the Amazon EKS command line tool eksctl.
- In GKE, running commands can be done through a combination of the open source kubectl command line and the gcloud CLI.

The second difference relates to auto-repair nodes.

- EKS Auto Mode helps keep the nodes in your EKS cluster in a healthy, running state.
- In GKE, node auto-repair helps keep the nodes in your cluster in a healthy, running state.

The third and final difference is control plan upgrades.

- In both EKS and GKE, node auto-upgrades keep the nodes in your cluster up-to-date with the cluster control plane version when your control plane is updated on your behalf.

Refer to the additional resource "Comparison of EKS and GKE" to learn more about the similarities and differences between these two services.

GKE: Autopilot Mode (EKS on Fargate)

GKE manages underlying infrastructure of the cluster, including the nodes

High availability

Regional cluster; Regular Release Channel; Auto-Update; Auto-Repair; Surge Upgrade;

Network

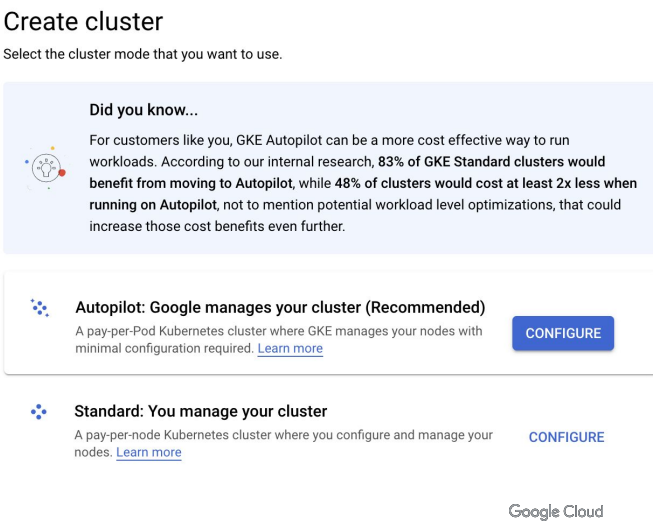
VPC Native (alias IP); IP-friendly (limit cluster size/ pods per node); full network flexibility

Highly Scalable

Node Auto Provision; Horizontal Pod Autoscaler; Vertical Pod Autoscaler

Secured by default

Workload Identity; Shielded Nodes; Secure-boot-disk; COS and Containerd, block known unsecure features.



Google Cloud

Autopilot is all about least operational overhead
AWS: EKS on Fargate. But: This is not just "serverless nodes" (like Fargate); it is a fully managed cluster. Google manages the control plane, the worker nodes, security patching, and scaling. You just submit standard Kubernetes YAMLs, and Google charges you for the CPU/RAM your pods request. It supports almost the entire K8s API

[First of all, the recommended and default type of GKE cluster is the Autopilot one now. But at the same time, the exam was last updated when Autopilot wasn't even there, which means you should know quite a bit about options in Standard GKE clusters, where you would still make most of the decisions about the settings and the infrastructure itself.]

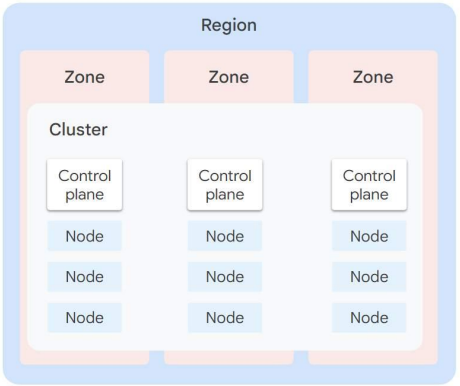
Regional GKE clusters

Available for GKE Standard and Autopilot modes only

Regional clusters have a single API endpoint for the cluster.

A cluster's control planes and nodes are spread across multiple Compute Engine zones within a region.

This maintains the availability of the application and control plane across multiple zones in a region.



You can address this concern by using a GKE regional cluster. This is available for both GKE Standard and Autopilot modes.

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.

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 three zones, each containing one control plane and three 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.

Note that once you build a zonal cluster, you can't convert it into a regional cluster, or vice versa.

Pod Disruption Budget, Readiness and Liveness Probes

A **PDB (Pod Disruption Budget)** limits the number of pods of a replicated application that can be taken down **simultaneously** from **voluntary** disruptions.

An Application Owner can create a **PodDisruptionBudget** object (PDB) for each application.

Readiness probes: designed to know when your app is ready to serve traffic.

Liveness probes: designed to let Kubernetes know if your app is alive or dead.

```
kind: PodDisruptionBudget
metadata:
 name: km-pdb
spec:
 minAvailable: 2
 selector:
   matchLabels:
     app: kobimysql
```

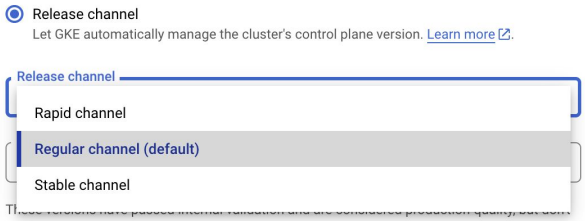
```
maxUnavailable: 1
```

Exam Tip:
- *See how to ensure stateful workloads are disruption-ready*
- *Great explanation of Readiness and Liveness probes here.*

[You might see a question or two about the concepts you see here. Basically, they allow you to keep your application healthy and available even if you're updating it to a newer release. For example when you want to quickly replace one version of you microservice with a new one. Being quick might be important, but it's also very import to still have your application running while it's being updated, and this is why you should be familiar with those concepts. In the exam tips on the bottom of the slide, you will find links to great explanations, so make sure to see those]

Best practices for GKE upgrades

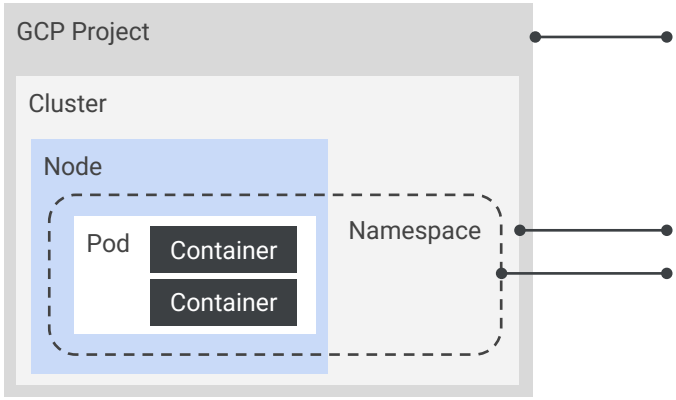
1. **Setup multiple environments**: at a minimum pre-production and production clusters
2. **Enroll Clusters in Release Channels**: Stable or Regular release channels for production clust



1. **Create continuous upgrade strategy**: Receive updates about new GKE versions through cluster upgrade notifications through Pub/Sub
2. **Schedule maintenance windows and exclusions**: to increase upgrade predictability
3. **Set tolerance for disruption**: To ensure that pods sufficient number of replicas, use Pod Disruption Budget

[Here we see a couple of best practices regarding GKE upgrades, mostly at infrastructure layer; they are mostly self explanatory, so let's move forward]

GKE: Using IAM and RBAC

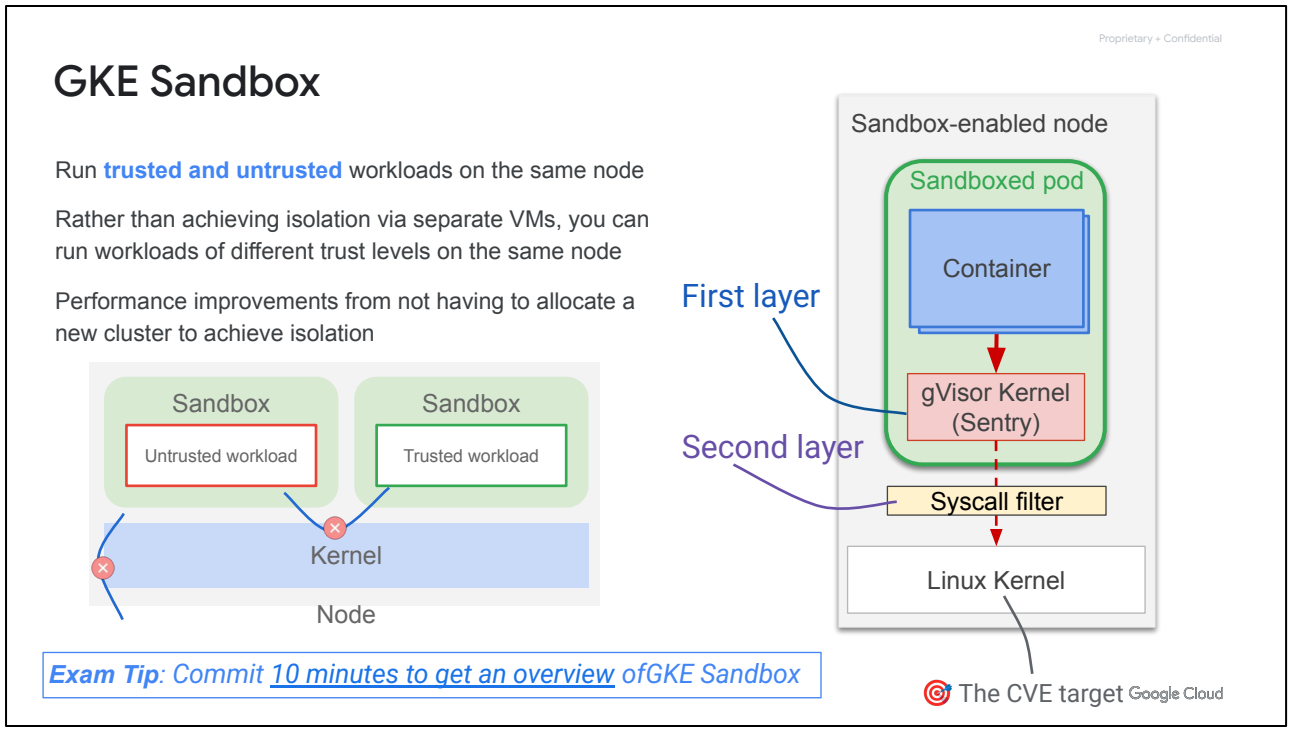


Use IAM at the project level
Set roles for
- Cluster Admin: manage clusters
- Container Developer: API access within clusters

Use RBAC at the cluster and namespace level
Set permissions on individual clusters and namespaces

Exam Tip: *IAM and Kubernetes RBAC work together to help manage access to your cluster. RBAC controls access on a cluster and namespace level, while IAM works on the project level*

[When it comes to controlling access to your clusters, you can use just IAM, or - and that's actually a best practice - you might use IAM at a project level, and if you have different groups of developers using different namespaces or clusters, that's where you might supplement IAM with RBAC, which is Kubernetes-native way of managing permissions]



[GKE Sandbox is something you need to understand on high level, mostly to identify its use-case. This feature basically prevents untrusted code from affecting your kernel on the VM level. So if you need to use a Docker image that you don't trust, for example you're using one from a 3rd party provider, it's best if you use this additional prevention mechanism.]

GKE Sandbox

https://cloud.google.com/kubernetes-engine/docs/concepts/sandbox-pods

GKE Sandbox uses gVisor, an open source project.

GKE Sandbox provides an extra layer of security to prevent untrusted code from affecting the host kernel on your cluster nodes. A container runtime such as docker or containerd provides some degree of isolation between the container's processes and the kernel running on the node. However, the container runtime often runs as a privileged user on the node and has access to most system calls into the host kernel.

Multi-tenant clusters and clusters whose containers run untrusted workloads are more exposed to security vulnerabilities than other clusters. Examples include SaaS providers, web-hosting providers, or other organizations that

allow their users to upload and run code. A flaw in the container runtime or in the host kernel could allow a process running within a container to "escape" the container and affect the node's kernel, potentially bringing down the node.

GKE Sandbox uses gVisor, an open source project. Direct access to the host kernel is limited.

When you enable GKE Sandbox on a node pool, a sandbox is created for each Pod running on a node in that node pool. In addition, nodes running sandboxed Pods are prevented from accessing other Google Cloud services or cluster metadata.

GKE networking: Subnet sizes

Subnet size for nodes	Maximum nodes	Maximum Pod IP addresses needed	Recommended Pod address range
/29	4	1,024	/21
/28	12	3,072	/20
/27	28	7,168	/19
/26	60	15,360	/18
/25	124	31,744	/17
/24	252	64,512	/16
/23	508	130,048	/15
/22	1,020	261,120	/14
/21	2,044	523,264	/13
/20	4,092	1,047,552	/12
/19	8,188	2,096,128	/11 (maximum Pod address range)

***Exam Tip**: make sure to watch this video to understand GKE networking well!*

[When it comes to GKE networking, it's pretty complex topic, since you may waste tens or even hundreds of thousands or IP addresses if you don't explicitly plan the size of your clusters and amount of Pods you'll have. So the exam expects you are able to optimize IP usage for GKE and you might need to calculate subnet sizes for your GKE nodes, Pods and services based on business assumptions you'll get. If that's something you don't feel confident with, make sure to have a look at the video linked on the bottom of the slide]

https://www.youtube.com/watch?v=aVBV4O3h4AY&t=1s

GKE networking: Example

Subnet size for nodes	Maximum nodes	Maximum Pod IP addresses needed	Recommended Pod address range
/29	4	1,024	/21

$2^{(32-29)} = 8$ (4 of these are reserved for GCP)

110 pods running in each node -> **4 * 110 = 440**

Twice number of IPs per pod **440 * 2 = 880**

2^{10} number of IPs for pods

*Assuming the default maximum of 110 pods per node

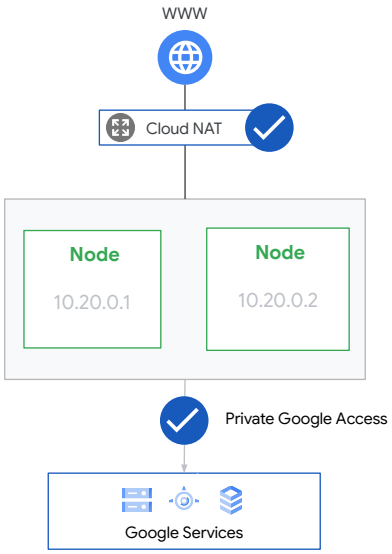
[And that's an example of what you might expect. The question might ask you to choose an optimal subnet size for a GKE cluster with up to 4 nodes and up to 110 pods on each of them. You need to know the rules and then make the calculations in your head, since you cannot use any materials at the exam]

GKE best practices: Private Clusters

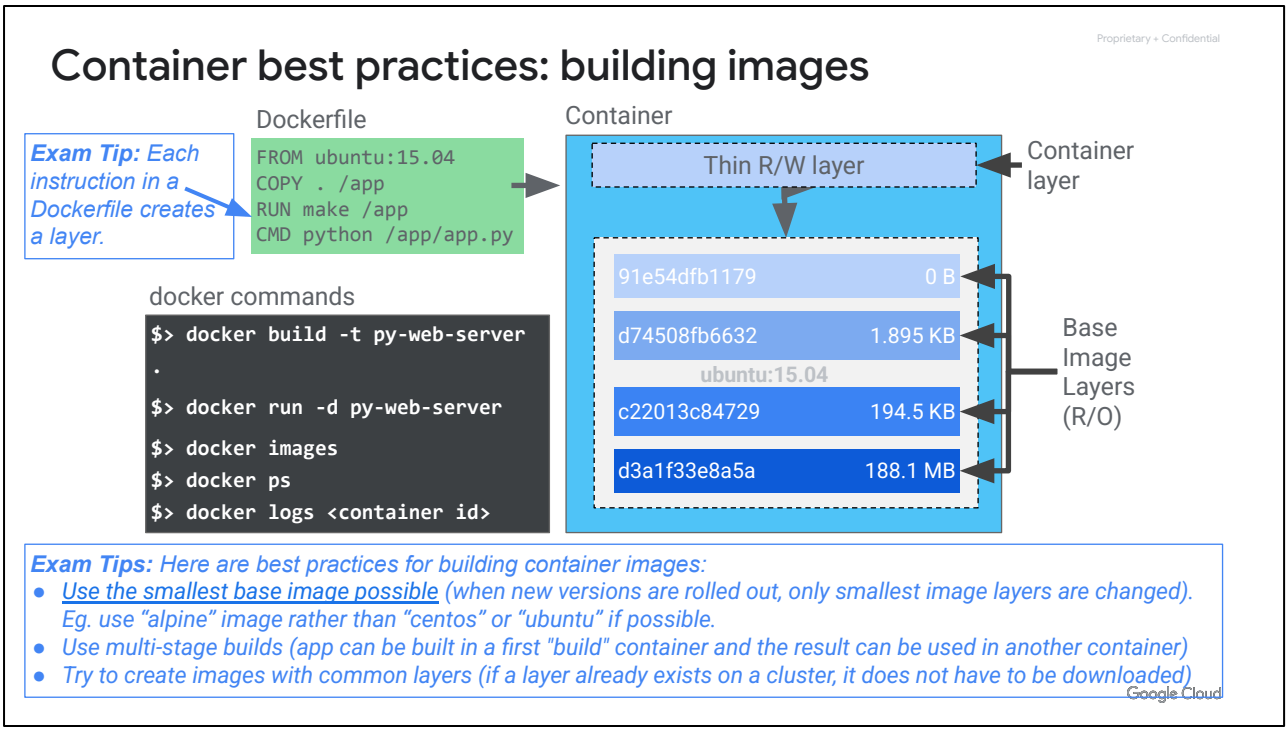
Private clusters **isolate nodes** from having inbound and outbound connectivity to the public internet

- Nodes have only **private IP addresses**
- Nodes use **Private Google Access** to communicate with Google APIs
- Nodes can use Cloud NAT to reach the internet
- **Control Plane** gets an additional private endpoint **for the cluster nodes to talk to the control plane.**

Exam Tip:
- *Private Clusters are definitely a best practice with GKE*
- *Having a Private Cluster does NOT mean you can't expose workloads via Services to the outside world!*



[Private Clusters are definitely a best practice in GKE. You can think about them in a similar way to MIGs without external IPs. Once you deploy a Private GKE cluster, you still need to decide how to access the control plane, and if your cluster nodes need to access some GCP services (like GCS or anything else), you would enable Private Google Access, just like for a normal VM.]



[It's good to know a couple of things about best practices for building Docker images. I've seen questions about how to use smallest images possible or use multi-stage builds, so have a look at the Exam Tips at the bottom of the slide]

A docker container is an image built in layers. Each layer is created by an instruction in the dockerfile.

All the layers except for the top one are locked. The thin read/write layer at the top is where you can make changes to a running container.

For example, if you needed to change a file, those changes would be written here.

The layered design inside of a container isolates functions. This is what makes the container stable and portable.

Here are a few of the common docker commands.

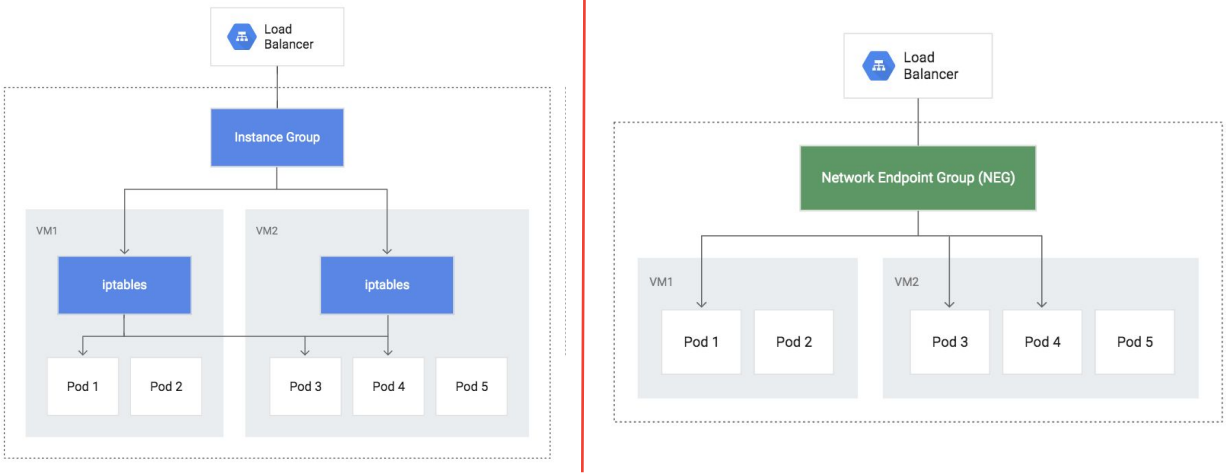
The docker build command creates the container image.

The docker run command runs the container.

There are other docker commands that can help you list images, check the status of a

running container, work with logs, or stop a running container.

Ingress service: standard (non-NEG) vs NEG



Exam Tip: *NEG is often preferred as a container-native load balancing type.*

[When exposing your GKE workloads, you will use Services, but technically they are the same Load Balancers as you're using for any other purposes in GCP. Just make sure to understand the Network Endpoint Group for your Load Balancers, since it can optimize the latency and it's really a best practice if you're exposing Kubernetes-based workloads]

How to bootstrap / change a GKE cluster

Exam Tips:

- *When creating / modifying / deleting cluster (or its node pools), you should use gcloud command (since you're interacting with GCP to manage INFRASTRUCTURE for GKE). For example:*
  - *To create a cluster: 'gcloud container clusters create… '*
  - *To resize a cluster (change number of nodes): 'gcloud container clusters **resize** CLUSTER_NAME --node-pool POOL_NAME --num-nodes NUM_NODES'*
  - *To enable autoscaling on a node pool of existing cluster: 'gcloud container clusters **update** CLUSTER_NAME **--enable-autoscaling** --node-pool=POOL_NAME --min-nodes=MIN_NODES --max-nodes=MAX_NODES --region=COMPUTE_REGION'*
  - *To disable autoscaling on a node pool of existing cluster: 'gcloud container clusters update CLUSTER_NAME **--no-enable-autoscaling** --node-pool=POOL_NAME --region=COMPUTE_REGION'*
- *When interacting with Kubernetes objects (eg. you'd like to deploy some Pods), you should use 'kubectl' command, eg:*
  - *To scale a deployment: 'kubectl autoscale deployment hello-app --cpu-percent=80 --min=1 --max=5'*

Google Cloud

[I think you're already aware that you might see some commands on the exam and you need to differentiate between them a bit. So here are some GKE-specific commands which might be useful. Specifically, make sure to differentiate between 'gcloud' command which you would use at infrastructure level (like: creating or resizing a GKE cluster) and 'kubectl' commands which you would use at workload level - when deploying or changing your microservices on existing cluster]

So the first thing I need to do before I really do anything else in Google Kubernetes engine is create a cluster. And to do that, I can't use that Kubernetes command because we don't have anything with Kubernetes in it yet. So there's a special Command G Cloud container clusters create, we're going to name our cluster k1. And when I do that command, all of this gets created for me. Now in real life, I can say, what size virtual machine do I want this to be? Don't want two nodes or 10. nodes are, what do I want? What kind of network do I want them to be on, there lots of configurations. But just for this class, all we really need to know is hey, there's a command creates a cluster. Yay. And it really is that simple

Now that you've built a container, you'll want to deploy one into a **cluster**.

Kubernetes can be configured with many options and add-ons, but can be time consuming to bootstrap from the ground up. Instead, you can bootstrap Kubernetes using **Google Kubernetes Engine** or (GKE).

GKE is a hosted Kubernetes by Google. GKE clusters can be customized and they support different machine types, number of nodes, and network settings.

To start up Kubernetes on a **cluster** in GKE, all you do is run this command: **`$> gcloud container clusters create k1`**

At this point, you should have a cluster called 'k1' configured and ready to go.

You can check its status in admin console.

Update strategies: Rolling update

When a Deployment is updated, it launches a new ReplicaSet and creates a new set of Pods in a controlled fashion.

The example here explains a rolling update strategy, which is also known as a ramped strategy.

Let's examine this process.

First, new Pods are launched in a new ReplicaSet. Kubernetes creates a new ReplicaSet with the updated configuration. This new ReplicaSet starts creating new pods.

Deployment monitors the new pods' health to ensure they are healthy and ready to serve traffic.

Then, as new pods become available, the service directing traffic to the pods starts to gradually send traffic to the new pods.

The old ReplicaSet starts to scale down, reducing the number of old pods. This happens gradually.

This process continues until all the pods in the Deployment are running the new version.

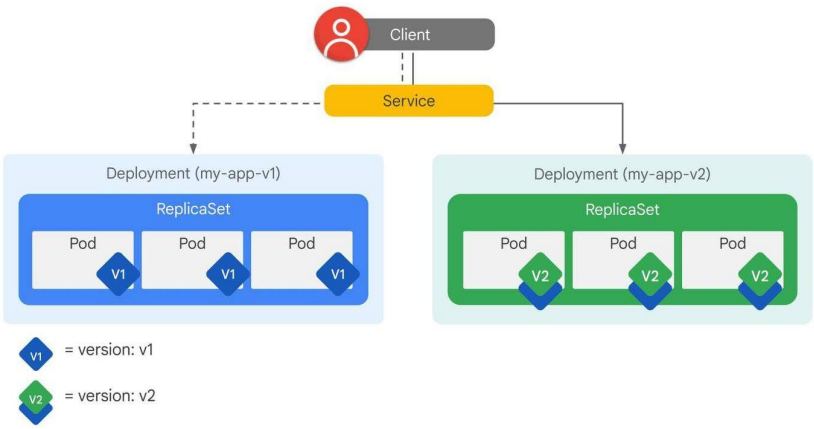
Finally, once all the old pods are terminated, the old ReplicaSet is deleted, and the new ReplicaSet manages all the pods in the Deployment.

The advantage of this process is that updates are slowly released, which ensures the availability of the application.

However, this process can take time, and there's no control over how the traffic is directed to the old and new Pods.

Now let's look at another strategy.

Update strategies: Blue/green



A blue/green deployment strategy is useful when you want to deploy a new version of an application and also ensure that application services remain available while the Deployment is updated.

With a blue/green update strategy, a completely new Deployment is created with a newer version of the application. In this example, it's my-app-v2.

When the Pods in the new Deployment are ready, the traffic can be switched from the old blue version to the new green version.

A Kubernetes service lets you manage the network traffic flows to a selection of Pods. This set of Pods is selected using a label selector.

Here, in the Service definition, Pods are selected based on the label selector, where Pods in this example belong to my-app and to version v1.

When a new Deployment (labeled v2 in this example), is created and is ready, the version label in the Service is changed to the newer version, labeled v2 in this example.

Now, the traffic will be directed to the newer set of Pods, the green deployment with the v2 version label, instead of to the old blue deployment Pods that have the v1 version label.

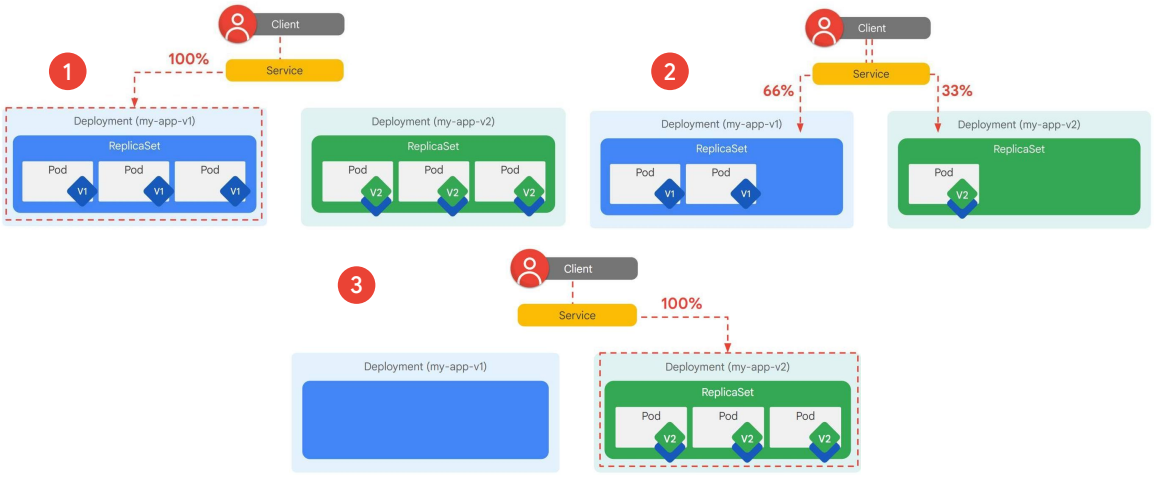
The blue Deployment with the older version can then be deleted.

The advantage of this update strategy is that the rollouts can be instantaneous, and the newer versions can be tested internally before releasing them to the entire user base, for example by using a separate service definition for test user access.

The disadvantage is that resource usage is doubled during the Deployment process.

Now let’s look at a third strategy.

Update strategies: Canary (1/2)



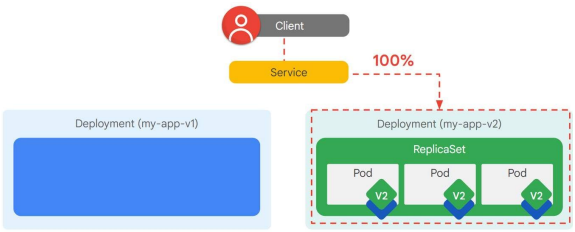
The canary method is another update strategy based on the blue/green method, but traffic is gradually shifted to the new version.

The main advantages of using canary deployments are that you can minimize excess resource usage during the update, and because the rollout is gradual, issues can be identified before they affect all instances of the application.

1. In this example, initially 100% of the application traffic is directed to my-app-v1.
2. When the canary deployment starts, a subset of the traffic, 33% in the example, or a single Pod, is redirected to the new version, my-app-v2, while 66%, or two Pods, from the older version, my-app-v1, remain running.
3. When the stability of the new version is confirmed, 100% of the traffic can be routed to this new version.

Update strategies: Canary (2/2)

- The Service selector is based only on the application label and does not specify the version.
- This setting allows the Service to select and direct the traffic to the Pods from both Deployments.
- A subset of users will be directed to the new version.
- May require tools such as Istio to accurately shift the traffic.



In the blue/green update strategy covered previously, both the app and version labels were selected by the Service, so traffic would only be sent to the Pods that are running the version defined in the Service.

Let's take a look at how this works in a Canary update strategy.

In a Canary update strategy, the Service selector is based only on the application label and does not specify the version.

The selector in this example covers all Pods with the app:my-app label. This means that with this Canary update strategy version of the Service, traffic is sent to all Pods, regardless of the version label.

This setting allows the Service to select and direct the traffic to the Pods from both Deployments. Initially, the new version of the Deployment will start with zero replicas running. Over time, as the new version is scaled up, the old version of the Deployment can be scaled down and eventually deleted.

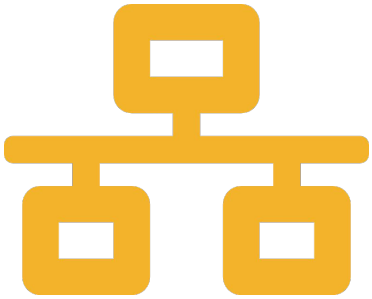
With the canary update strategy, a subset of users will be directed to the new version. This allows you to monitor for errors and performance issues as these users use the new version, and you can roll back quickly, minimizing the impact on your overall user base, if any issues arise.

However, the complete rollout of a Deployment using the canary strategy can be a

slow process and may require tools such as Istio to accurately shift the traffic.

Update strategies: Session affinity

- A Service configuration does not normally ensure that all requests from a single client will always connect to the same Pod.
- To prevent this, set the sessionAffinity field to ClientIP in the specification of the service if you need a client's first request to determine which Pod will be used for all subsequent connections.



A Service configuration does not normally ensure that all requests from a single client will always connect to the same Pod.

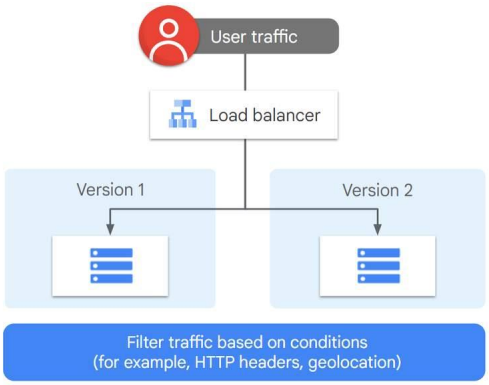
Each request is treated separately and can connect to any Pod deployment.

This potential can cause issues if there are significant changes in functionality between Pods as may happen with a canary deployment.

To prevent this, you can set the sessionAffinity field to ClientIP in the specification of the service if you need a client's first request to determine which Pod will be used for all subsequent connections.

Update strategies: A/B testing

- Test a hypothesis by using variant implementations.
- To perform an A/B test, you route a subset of users to new functionality based on routing rules.
- A/B testing is best used to measure the effectiveness of functionality in an application.



With A/B testing, you test a hypothesis by using variant implementations. A/B testing is used to make business decisions (not only predictions) based on the results derived from data.

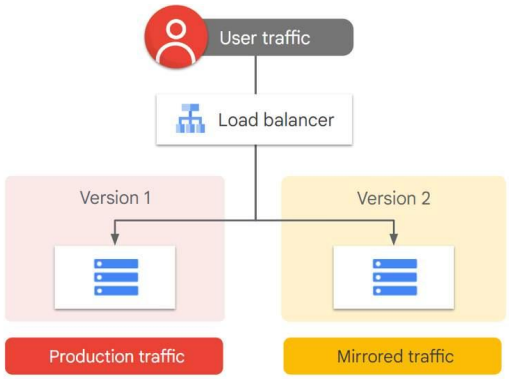
When you perform an A/B test, you route a subset of users to new functionality based on routing rules as shown in the example here.

Routing rules often include factors such as browser version, user agent, geolocation, and operating system. After you measure and compare the versions, you update the production environment with the version that yielded better results.

A/B testing is best used to measure the effectiveness of functionality in an application. Use cases for the deployment patterns discussed earlier focus on releasing new software safely and rolling back predictably. In A/B testing, you control your target audience for the new features and monitor any statistically significant differences in user behavior.

Update strategies: Shadow testing

- Deploy and run a new version alongside the current version.
- An incoming request is mirrored and replayed in a test environment.
- Ensure the shadow tests do not trigger side effects that can alter the existing production environment or the user state.
- It's typically combined with other approaches like canary testing.
- Shadow testing offers many benefits.



With shadow testing, you deploy and run a new version alongside the current version, but in such a way that the new version is hidden from the users, as the diagram here illustrates.

An incoming request is mirrored and replayed in a test environment. This process can happen either in real time or asynchronously after a copy of the previously captured production traffic is replayed against the newly deployed service.

You need to ensure that the shadow tests do not trigger side effects that can alter the existing production environment or the user state.

Traffic shadowing is typically combined with other approaches like canary testing. After testing a new feature by using traffic shadowing, you then test the user experience by gradually releasing the feature to an increasing number of users over time. No full rollout occurs until the application meets stability and performance requirements.

Shadow testing has many key benefits:

- Because traffic is duplicated, any bugs in services that are processing shadow data have no impact on production.
- When used with tools such as Diffy, traffic shadowing lets you measure the behavior of your service against live production traffic. This ability lets you test for errors, exceptions, performance, and result parity between application versions.

- Finally, there is a reduced deployment risk.

Choosing the right strategy

Most critical considerations

What are your most critical considerations? For example, is downtime acceptable?

Constraints and processes

Are you constrained by costs, skills, testing controls and other processes and elements?

Microservices in your environment

Are your microservices fully autonomous, or are they hybrid?

Does the new release involve schema changes?

Additional resource: Summary of deployment and testing patterns

You can deploy and release your application in several ways. Each approach has advantages and disadvantages.

The best choice comes down to the needs and constraints of your business.

Let's look at what you should consider when choosing the right approach.

First, what are your most critical considerations? For example, is downtime acceptable?

Consider constraints and processes:

- Do costs constrain you?
- Does your team have the right skills to undertake complex rollout and rollback setups?
- Do you have tight testing controls in place, or do you want to test the new releases against production traffic to ensure the stability of the release and limit any negative impact?
- Do you want to test features among a pool of users to cross-verify certain business hypotheses?
- Can you control whether targeted users accept the update? For example, updates on mobile devices require explicit user action and might require extra permissions.

The final consideration is the microservices in your environment:

- Are microservices in your environment fully autonomous?
- Or, do you have a hybrid of microservice-style applications working alongside traditional, difficult-to-change applications? For more information, refer to deployment patterns on hybrid and multi-cloud environments.
- Does the new release involve any schema changes? If yes, are the schema changes too complex to decouple from the code changes?

Refer to the additional resource ‘Summary of deployment and testing patterns’ for a summary of the salient characteristics of the deployment and testing patterns.

Rolling back Deployments

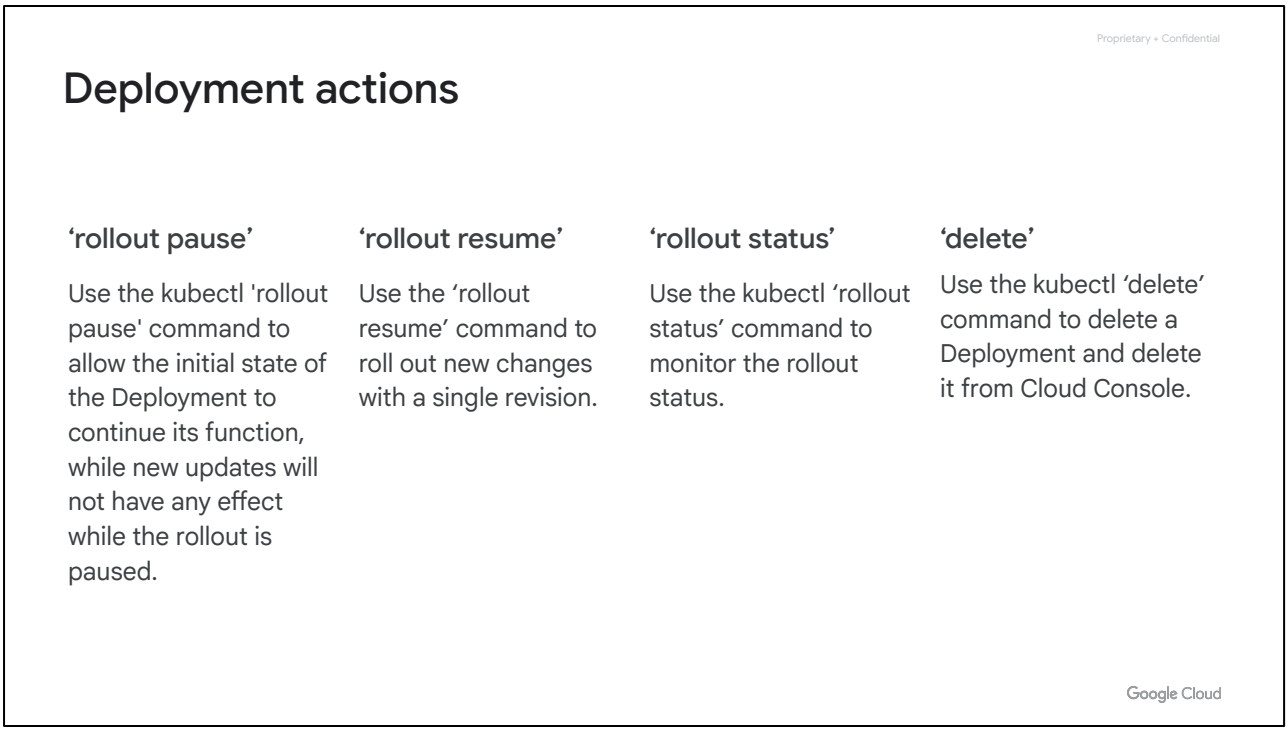
- Use the kubectl 'rollout undo' command to return to a previous version, ensuring that you specify the revision number.
- Inspect the rollout history using the kubectl 'rollout history' command.
- Start Cloud Shell from your Console to input these commands.
- By default, the details of 10 previous ReplicaSets are retained, but you can change this limit under the Deployment specification.

You roll back using a kubectl 'rollout undo' command. A simple 'rollout undo' command will revert the Deployment to its previous revision.

You roll back to a specific version by specifying the revision number.

If you're not sure of the changes, you can inspect the rollout history using the kubectl 'rollout history' command. The Cloud Console doesn't have a direct rollback feature; however, you can start Cloud Shell from your Console and use these commands. The Cloud Console also shows you the revision list with summaries and creation dates.

By default, the details of 10 previous ReplicaSets are retained, so that you can roll back to them. You can change this default by specifying a revision history limit under the Deployment specification.



When you edit a deployment, your action normally triggers an automatic rollout.

But if you have an environment where small fixes are released frequently, you'll have a large number of rollouts. In a situation like that, you'll find it more difficult to link issues with specific rollouts.

To help, you can use these commands.

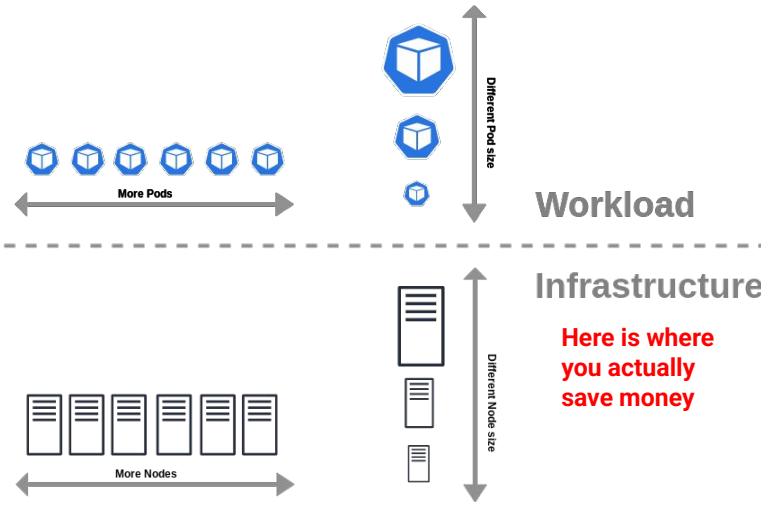
You can temporarily pause a rollout by using the kubectl 'rollout pause' command. The initial state of the Deployment prior to pausing it will continue its function, but new updates to the Deployment will not have any effect while the rollout is paused. The changes will only be implemented once the rollout is resumed.

When you resume the rollout with the 'rollout resume' command, all these new changes will be rolled out with a single revision.

You can also monitor the rollout status by using the kubectl 'rollout status' command.

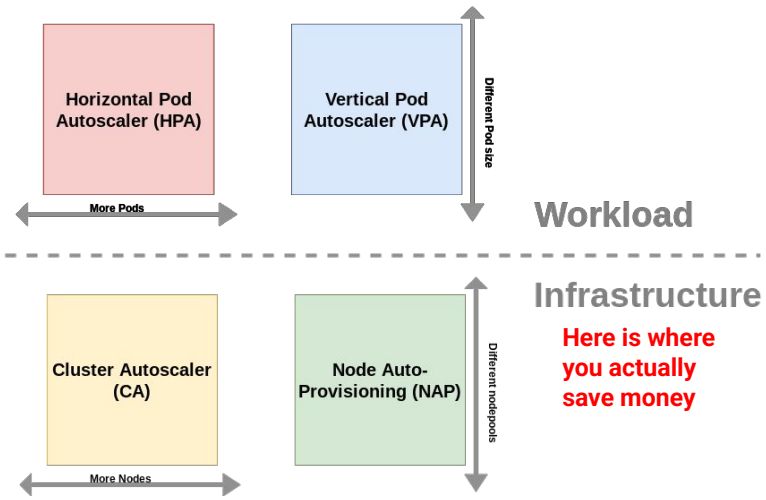
You can delete a deployment by using the kubectl 'delete' command, and you can also delete it from the Cloud Console. Either way, Kubernetes will delete all resources managed by the Deployment, especially running Pods.

Kubernetes: The 4 scalability dimensions



[As you know, public cloud allows you to scale your workloads and only pay for the resources you're using. And that's why it makes sense to have some automation which would scale your workloads AND the infrastructure at the same time. With Autopilot clusters you would expect Google to make decisions about infrastructure for you, but with Standard clusters, you are still responsible for this part.]

GKE supports all 4 scalability dimensions



[So you should be aware of different concepts like HPA, VPA on Workload level, but also the ones on Infrastructure level. Google allows you to set it all up and you should be more or less familiar with how to do it as an architect]

First, Horizontal Pod autoscaling.

Horizontal Pod autoscaling automatically increases or decreases the number of Pods on demand.

It uses a k8s controller object called **Horizontal Pod Autoscaler** or HPA to regularly adjust the number of Pod replicas in a replication controller or deployment to match the observed metrics such as average CPU and memory utilization.

So you will see the number of Pods keep increasing or decreasing dynamically.

Vertical Pod scaling on the other hand, **automatic-ally** increases and decreases resources assigned to individual Pods based on observed usage.

changing.

Vertical Pod Autoscaler (VPA) automatically recommends or applies values for CPU and memory requests and limits for Pods.

It's also suitable for stateful applications or when horizontal scaling is not applicable

Third is the Multidimensional Pod autoscaling.

As the name suggests, it does both vertical and horizontal at the same time.

It is currently in Beta, it supports only horizontal scaling based on CPU and vertical scaling based on memory.

And lastly, Cluster Autoscaler.

Cluster Autoscaler or CAS automatically increases or decreases a number of nodes in the cluster in response to workload demands.

Cluster Autoscaler (CAS) controls the number of nodes and it is disabled by default.

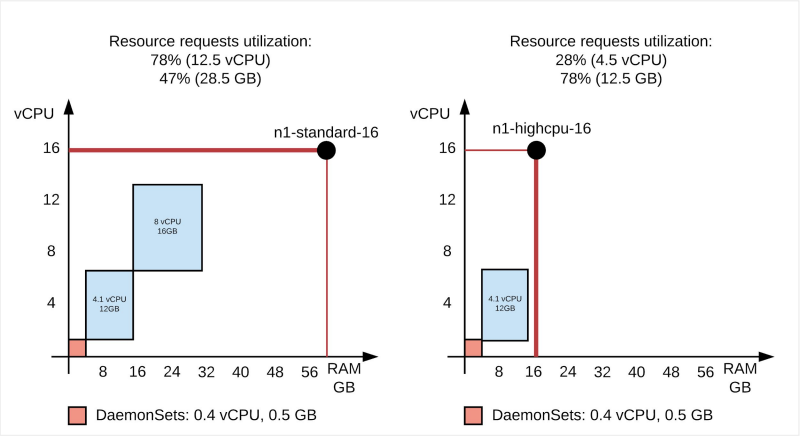
It automatically adjusts the size of the k8s cluster when there are Pods that failed to schedule due to insufficient resources or there are nodes that have been underutilized for a period of time.

For GKE, the Cluster Autoscaler works on a per-node pool basis, it increases or decreases the size of node pool automatically by adding and removing VM instances in and out of the pool based on the resource requests of Pods.

For GKE, the CAS works on a per-node pool basis, it increases or decreases the size of node pool automatically by adding and removing VM instances in the Managed Instance Group (MIG) for the node pool. CAS makes the decisions based on the resource requests of Pods.

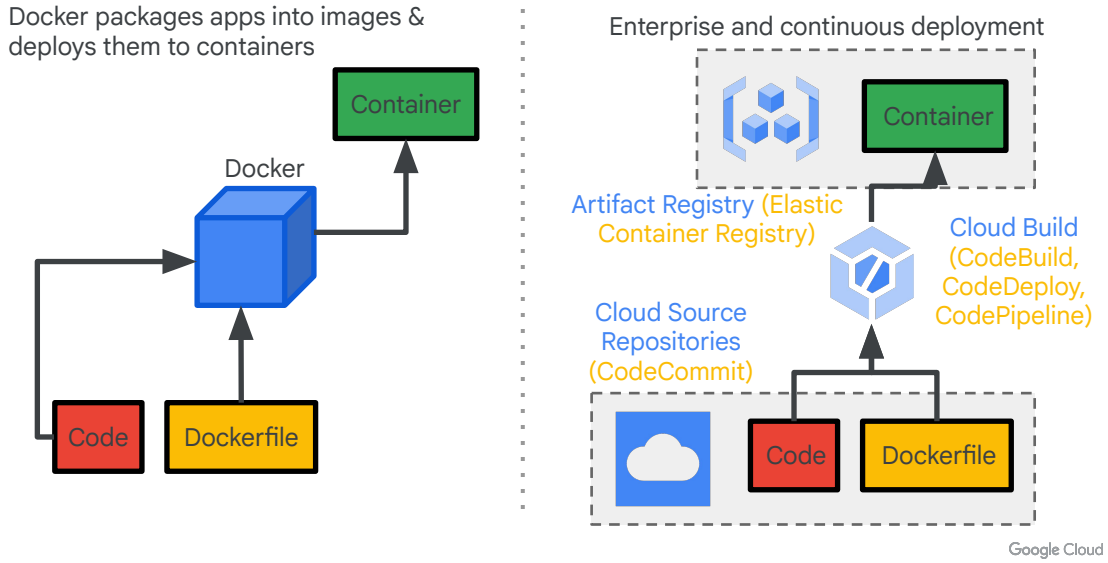
GKE: Binpacking

- Make sure your workload fit well inside the machine size
- You can create multiple node pools and use either nodeSelector or Node Affinity to select which node your pod must run.
- Another simpler option is to configure Node auto-provisioning



[And of course, since you're paying for resources with GKE Standard, you should use your resources effectively. That's where you would make decisions about additional Node Pools, different node sizes, using Spot nodes or nodes with GPUs attached. And since you may end up with a pretty wide types and sizes of worker nodes even in a single GKE cluster, you need to know how to schedule your Pods exactly where you want]

Container and code storage

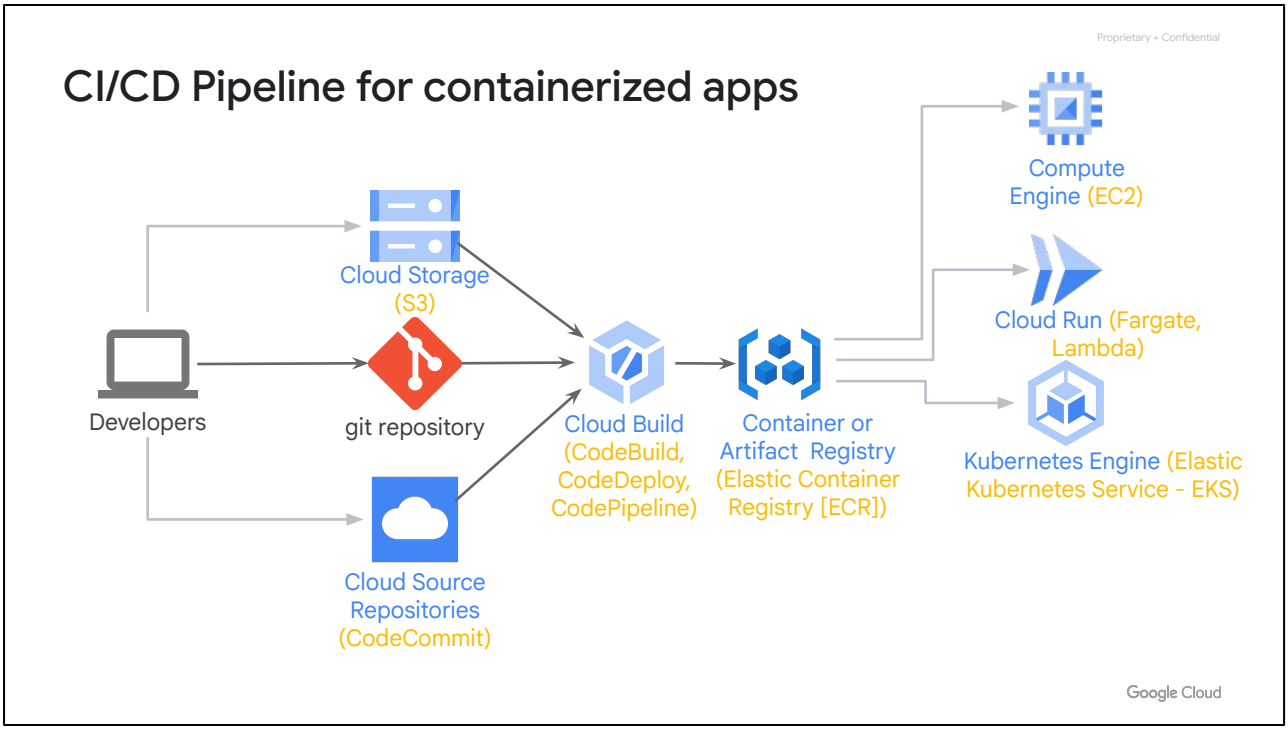


Docker is software that builds containers. You supply application code, and instructions, called a Dockerfile, and Docker follows the instructions and assembles the code and dependencies into the container. Containers can be "run", much as an application can run. However, it is a self-contained environment that can run on many platforms.

Google Cloud offers a service called Cloud Build which functions similarly to Docker. It accepts code and configuration and builds containers. Cloud Build offers many features and services that are geared towards professional development. It is designed to fit into a continuous development / continuous deployment workflow. And it is designed to scale to handle many application developers working on and continuously updating a live global service.

If you had a hundred developers sharing source files, you would need a system for managing them, for tracking them, versioning them, and enforcing a check-in, review, and approval process. Cloud Source Repositories is a cloud-based solution.

If you were deploying hundreds of containers you would not being keeping it to yourself. One of the reasons to use containers is to share them with others. So you need a way to manage and share them. And this is the purpose of Artifact Registry. Artifact Registry has various integrations with Continuous Integration / Continuous Deployment services.



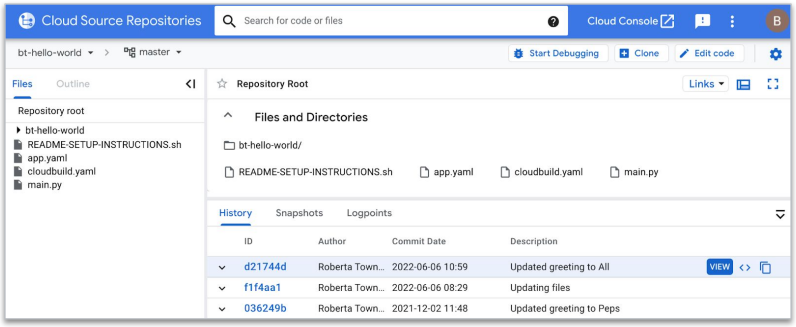
Quick focus on the Registry being the single staging point for various execution platforms (aka portability)
Cloud Build can retrieve the source code for your builds from a variety of storage locations: Cloud Source Repositories, Cloud Storage (which is Google Cloud's object-storage service), or git-compatible repositories like GitHub and Bitbucket.

To generate a build with Cloud Build, you define a series of steps. For example, you can configure build steps to fetch dependencies, compile source code, run integration tests, or use tools such as Docker, Gradle, and Maven. Each build step in Cloud Build runs in a Docker container.

Then Cloud Build can deliver your newly built images to various execution environments: not only GKE, but also Cloud Run and Cloud Functions.

Cloud Source Repositories (CodeCommit)

- Fully featured, private Git repositories hosted on Google Cloud
- Easily perform Git operations required by your workflow
- Sync with existing GitHub or Bitbucket repo



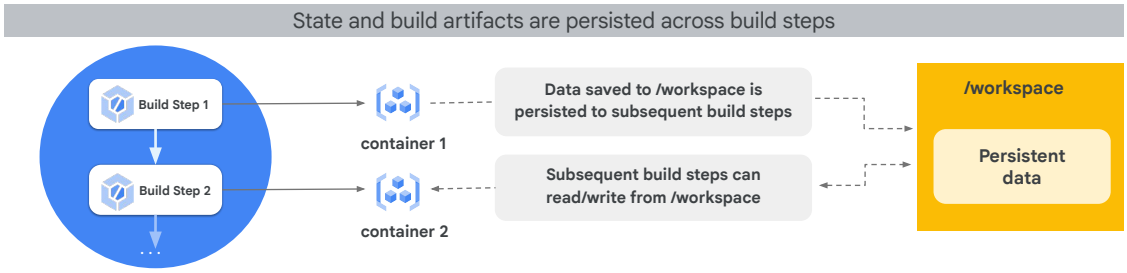
Google Cloud

Cloud Source Repositories
https://cloud.google.com/source-repositories
A single place for your team to store, manage, and track code.

More than just a private Git repository
Design, develop, and securely manage your code. Collaborate easily on a fully featured, scalable, and private Git repository. Extend your Git workflow by connecting to other Google Cloud tools, including Cloud Build, App Engine, Pub/Sub, and operations products such as Cloud Monitoring and Cloud Logging.

Cloud Build (CodeBuild, CodeDeploy, CodePipeline)

- Fully managed service
- Create triggers to automatically build, test, and/or deploy source code in Cloud Storage (S3), Cloud Source Repositories (CodeCommit), GitHub, or Bitbucket
- Produces artifacts such as Docker containers or Java archives
  - Build software quickly across multiple programming languages, including Java, Go, Node.js, and more



Cloud Build
https://cloud.google.com/cloud-build
Build, test, and deploy on our serverless CI/CD platform.

Cloud Build is a service that executes your builds on Google Cloud Platform infrastructure. Cloud Build can import source code from Google Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

Documentation
https://cloud.google.com/build/docs

Cloud Build is a service that executes your builds on Google Cloud Platform infrastructure. Cloud Build can import source code from Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

Cloud Build executes your build as a series of *build steps*, where each build step is run in a Docker container. A build step can do anything that can be done from a container irrespective of the environment. To perform your tasks, you can either use the supported build steps provided by Cloud Build or write your own build steps.

- Supported Build Steps → https://cloud.google.com/build/docs/deploying-builds/deploy-cloud-run

- Write your own build steps → https://cloud.google.com/build/docs/configuring-builds/use-community-and-custom-builders

Overview of Cloud Build
https://cloud.google.com/build/docs/overview

- Cloud Build is a service that executes your builds on Google Cloud Platform's infrastructure.
- Cloud Build can import source code from a variety of repositories or cloud storage spaces, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives.

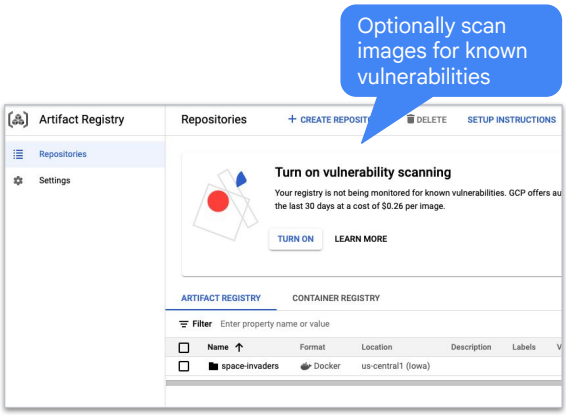
Quickstarts - Docker
https://cloud.google.com/build/docs/quickstart-build

Quickstarts - Go
https://cloud.google.com/build/docs/building/build-go

Artifact Registry (Elastic Container Registry)

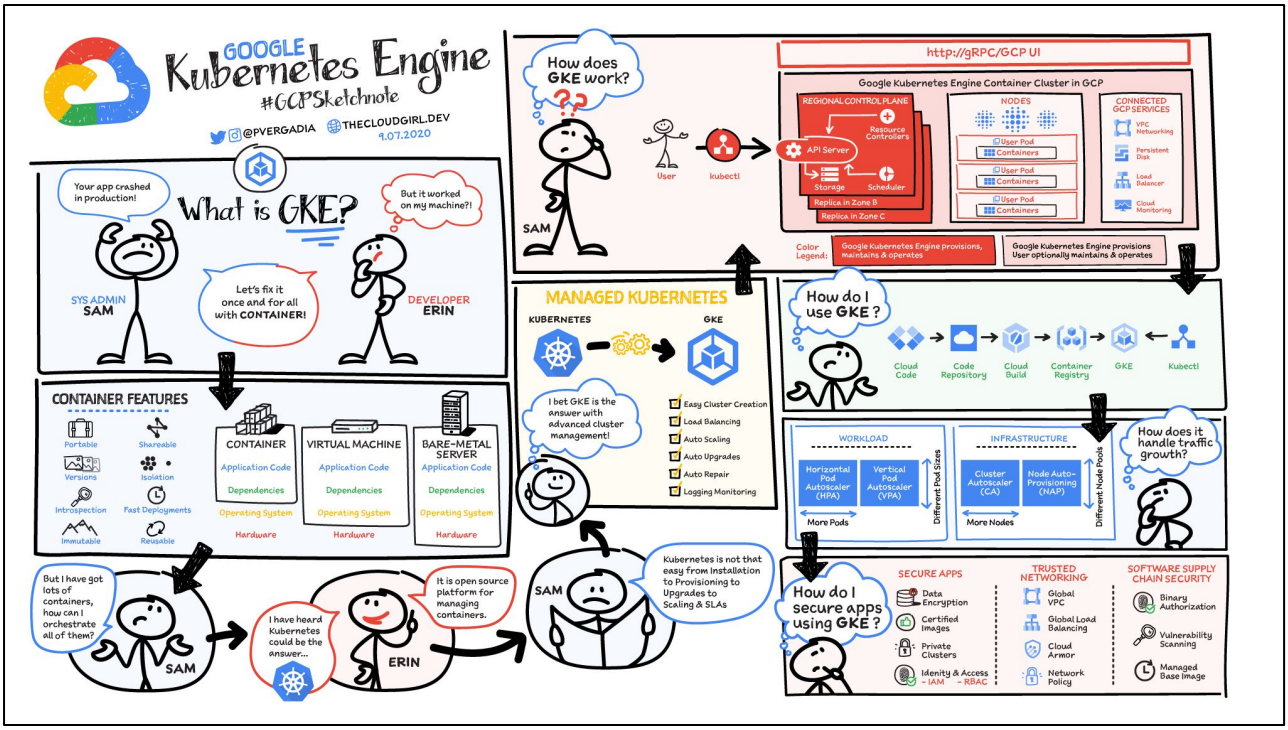
Provides the same management features as Container Registry plus more:

- Offers
  - Built-in vulnerability scanning of container images (paid / free or paid)
  - Controlled access with fine-grained control (IAM)
  - Regional and multi-regional (can automate copying to other regions and/or accounts) repositories
    - Can have multiple registries (repositories per registry) per project (account)
- Store multiple artifact formats
  - Docker images
  - OS packages for Linux distributions
  - Language packages for Python, Java, and Node



Container registry was regional only
Container registry - used Cloud Storage IAM.
Artifact Registry has its own IAM roles and can be deployed multiregional.

Optional demo - show how can deploy a image into CE, GKE or Cloud Run from within the Artifact Registry



[OK, I know that was super quick and high level, but do you have any questions regarding GKE?]

https://thecloudgirl.dev/GKE.html

Diagnostic Question Discussion

Your company has an application running as a Deployment in a Google Kubernetes Engine (GKE) cluster. When releasing new versions of the application via a rolling deployment, the team has been causing outages. The root cause of the outages is misconfigurations with parameters that are only used in production. You want to put preventive measures for this in the platform to prevent outages.

What should you do?

A. Configure liveness and readiness probes in the Pod specification.

B. Configure health checks on the managed instance group.

C. Create a Scheduled Task to check whether the application is available.

D. Configure an uptime alert in Cloud Monitoring.

A

A: Configuring the right liveness and readiness probes prevents outages when rolling out a new ReplicaSet of a Deployment, because Pods are only getting traffic when they are considered ready.
B: With GKE, you do not deal with MIGs.
C: Does not use GKE tools and is therefore not the best option.
D: Does alert you but does not prevent the outage.

Diagnostic Question Discussion

Your company has an application running as a Deployment in a Google Kubernetes Engine (GKE) cluster. When releasing new versions of the application via a rolling deployment, the team has been causing outages. The root cause of the outages is misconfigurations with parameters that are only used in production. You want to put preventive measures for this in the platform to prevent outages.

What should you do?

A. **Configure liveness and readiness probes in the Pod specification.**

B. Configure health checks on the managed instance group.

C. Create a Scheduled Task to check whether the application is available.

D. Configure an uptime alert in Cloud Monitoring.

A

A: Configuring the right liveness and readiness probes prevents outages when rolling out a new ReplicaSet of a Deployment, because Pods are only getting traffic when they are considered ready.
B: With GKE, you do not deal with MIGs.
C: Does not use GKE tools and is therefore not the best option.
D: Does alert you but does not prevent the outage.

Diagnostic Question Discussion

Your team needs to create a Google Kubernetes Engine (GKE) cluster to host a newly built application that requires access to third-party services on the internet.
Your company does not allow any Compute Engine instance to have a public IP address on Google Cloud. You need to create a deployment strategy that adheres to these guidelines.

What should you do?

A. Configure the GKE cluster as a private cluster. Configure Private Google Access on the Virtual Private Cloud (VPC).

B. Configure the GKE cluster as a public cluster and then disable external IPs on each of the cluster nodes.

C. Configure the GKE cluster as a private cluster, and configure Cloud NAT Gateway for the cluster subnet.

D. Create a Compute Engine instance, and install a NAT Proxy on the instance. Configure all workloads on GKE to pass through this proxy to access third-party services on the Internet.

C. Simple?

Diagnostic Question Discussion

Your team needs to create a Google Kubernetes Engine (GKE) cluster to host a newly built application that requires access to third-party services on the internet.
Your company does not allow any Compute Engine instance to have a public IP address on Google Cloud. You need to create a deployment strategy that adheres to these guidelines.

What should you do?

A. Configure the GKE cluster as a private cluster. Configure Private Google Access on the Virtual Private Cloud (VPC).

B. Configure the GKE cluster as a public cluster and then disable external IPs on each of the cluster nodes.

C. Configure the GKE cluster as a private cluster, and configure Cloud NAT Gateway for the cluster subnet.

D. Create a Compute Engine instance, and install a NAT Proxy on the instance. Configure all workloads on GKE to pass through this proxy to access third-party services on the Internet.

C. Simple?

Diagnostic Question Discussion

Your company uses Google Kubernetes Engine (GKE) as a platform for all workloads. Your company has a single large GKE cluster that contains batch, stateful, and stateless workloads. The GKE cluster is configured with a single node pool with 200 nodes. Your company needs to reduce the cost of this cluster but does not want to compromise availability.

What should you do?

A. Create a second GKE cluster for the batch workloads only. Allocate the 200 original nodes across both clusters.

B. Configure CPU and memory limits on the namespaces in the cluster. Configure all Pods to have a CPU and memory limits.

C. Configure a HorizontalPodAutoscaler for all stateless workloads and for all compatible stateful workloads. Configure the cluster to use node auto scaling.

D. Change the node pool to use preemptible VMs.

C

A: Is not necessary because you can have multiple node pools with different configurations.
B: Optimizes resource usage of CPU/memory in your existing node pool but does not necessarily improve cost - still an option that should be considered.
C: This looks really good. Autoscaling workloads and the node pools makes your whole infrastructure more elastic and gives you the option to rely on the same node pool.
D: This might not be a good option for every type of workload. Batch and stateless workloads can often handle this quite well, but stateful workloads are not well-suited for operation on preemptible VMs.

Diagnostic Question Discussion

Your company uses Google Kubernetes Engine (GKE) as a platform for all workloads. Your company has a single large GKE cluster that contains batch, stateful, and stateless workloads. The GKE cluster is configured with a single node pool with 200 nodes. Your company needs to reduce the cost of this cluster but does not want to compromise availability.

What should you do?

A. Create a second GKE cluster for the batch workloads only. Allocate the 200 original nodes across both clusters.

B. Configure CPU and memory limits on the namespaces in the cluster. Configure all Pods to have a CPU and memory limits.

C. Configure a HorizontalPodAutoscaler for all stateless workloads and for all compatible stateful workloads. Configure the cluster to use node auto scaling.

D. Change the node pool to use preemptible VMs.

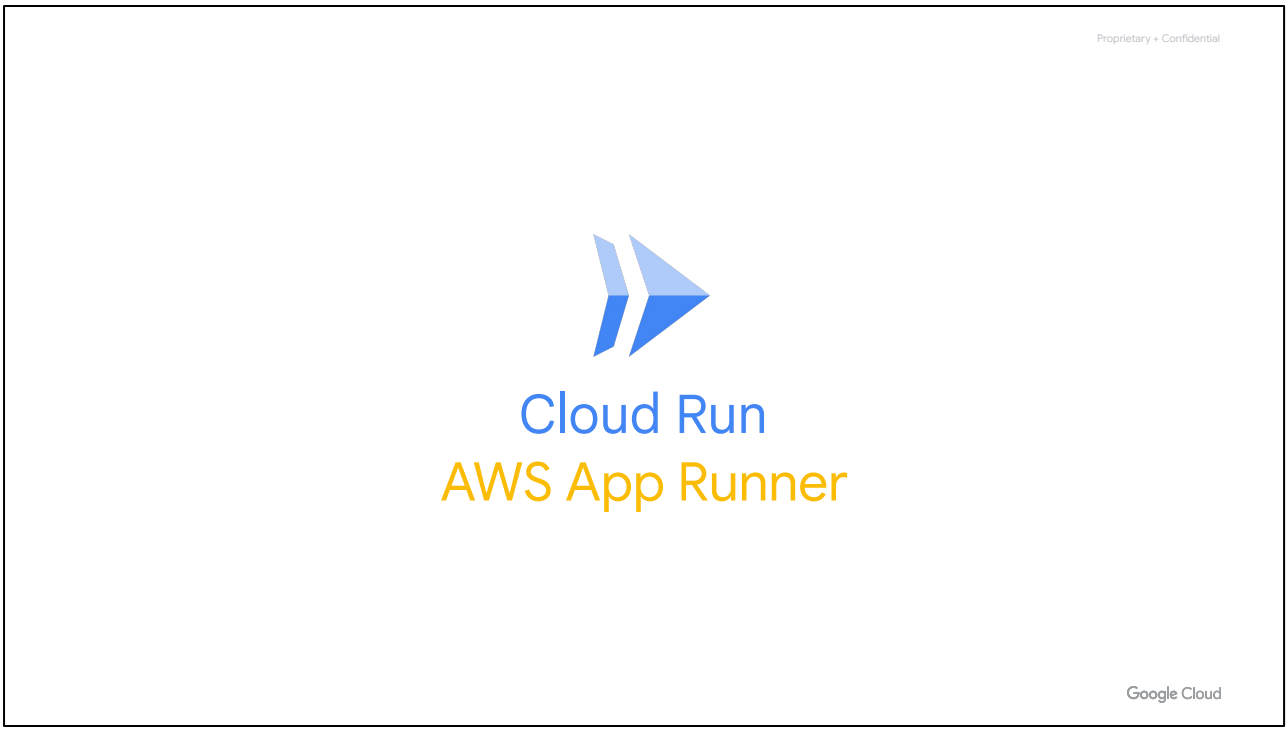
C

A: Is not necessary because you can have multiple node pools with different configurations.
B: Optimizes resource usage of CPU/memory in your existing node pool but does not necessarily improve cost - still an option that should be considered.
C: This looks really good. Autoscaling workloads and the node pools makes your whole infrastructure more elastic and gives you the option to rely on the same node pool.
D: This might not be a good option for every type of workload. Batch and stateless workloads can often handle this quite well, but stateful workloads are not well-suited for operation on preemptible VMs.

Compute: What if i need a fully managed solution?

Fully managed

A fully managed environment lets you focus on code while it manages infrastructure concerns. It offloads the administration tasks like provisioning, installation, configuration, patching, upgrades/updates, backups, etc.



Cloud Run may be used for almost any standard stateless web API or event-handler. It is vary easy t to use and often the cost-effective due to its concurrency model.

Cloud Run (AWS App Runner)

- Enables **stateless** containers.
- Fully managed serverless platform that runs individual containers.
- Based on the open-source knative project (https://knative.dev/)
- Abstracts away infrastructure management.
- Automatically scales up and down.

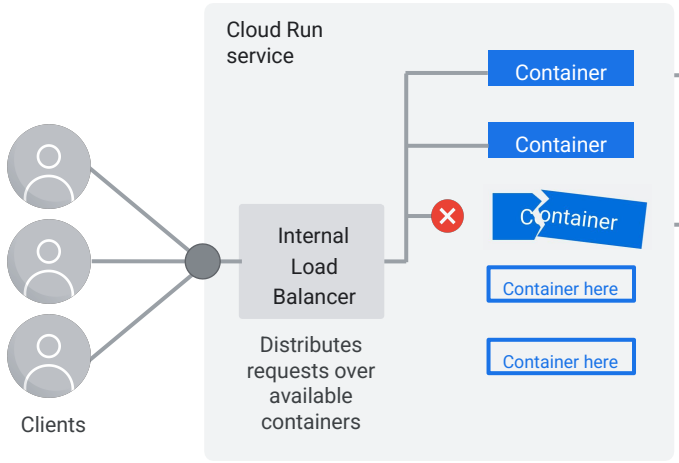
Exam Tip: *"Stateless" is the key here. Cloud Run is MUCH newer than App Engine (2019 vs 2008) and uses Kubernetes (App Engine uses pre-K8s and pre-Docker containers). Otherwise, use-cases for App Engine and Cloud Run are similar.*

Killer Feature: Concurrency. A single Cloud Run instance can handle up to 1000 concurrent requests (default 80). This is massively more efficient and cheaper for high I/O web APIs than standard functions that only handle one request at a time.
Speed: It has incredibly fast cold starts and scales aggressively to handle massive traffic spikes almost instantly.
Portability: Because it is based on the open-source **Knative** standard, you can theoretically take your Cloud Run YAML and deploy it to any Kubernetes cluster running Knative.

- App Runner adds another layer of abstraction and lifts the developer experience to a new level -> like Cloud Run!
- App Runner focuses on a single scenario: a web application answering incoming HTTP requests synchronously.
- App Runner is powered by AWS Fargate
- App Runner is newer (2021)
- App Runner is one of the easiest ways to run containers on AWS.
-
-
- *[you can think about Cloud Run like Serverless Kubernetes in GCP. It's where you still have your Docker image, but you don't even want to think of a GKE cluster, but instead - you'd like to*

- *have a service based on this image of yours, exposed to the outside world, automatically scalable and so on.]*
-
- So what's actually Cloud Run and how it differs from App Engine? Well, Cloud run is serverless Kubernetes. Since App Engine was released in 2008, it's obviously not Kubernetes. But the idea is the same: Cloud Run allows you to deploy an app and have it consumed, without any sizing or defining the resources underneath.
- So Cloud Run differentiates with App Engine in being Kubernetes-based and it really does not have such restrictions for programming languages or libraries you can use like App Engine does. And since it's Kubernetes-based, which are super-popular nowadays, it can be used in many more use-cases, also in those created with microservices approach.
- What's super-important is that with Cloud Run your workload need to be stateless. If you don't know what that means, don't worry, it will be explained during one of the courses
- So App Engine and Cloud Run can be used in similar scenarios, but when we see at the history, we'll notice that Cloud Run is a MUCH newer and more modern service that might even become a successor of App Engine in the future.
-
-
- Cloud Run is a managed compute platform that enables you to run stateless containers via web requests or Pub/Sub events. Cloud Run is serverless: it abstracts away all infrastructure management so you can focus on developing applications. It is built on Knative, an open-source, Kubernetes-based platform. It builds, deploys, and manages modern serverless workloads. Cloud Run gives you the choice of running your containers either fully-managed or in your own GKE cluster.
-
- Cloud Run automatically scales up and down from zero depending on traffic almost instantaneously, so you never have to worry about scale configuration. Cloud Run also charges you only for the resources you use (calculated down to the nearest 100 milliseconds), so you will never have to pay for your over-provisioned resources.
-
- With Cloud Run you can deploy your stateless containers with a consistent developer experience to a fully managed environment or to your own GKE cluster. This common experience is enabled by Knative, an open API and runtime environment built on Kubernetes that gives you freedom to move your workloads across different environments and platforms: fully managed on Google Cloud, on GKE, or anywhere Knative runs.
-

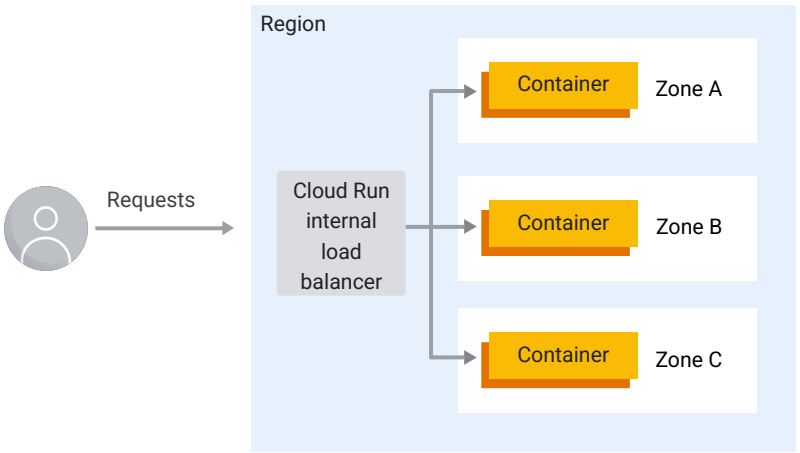
All incoming requests are handled with automatic scaling



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



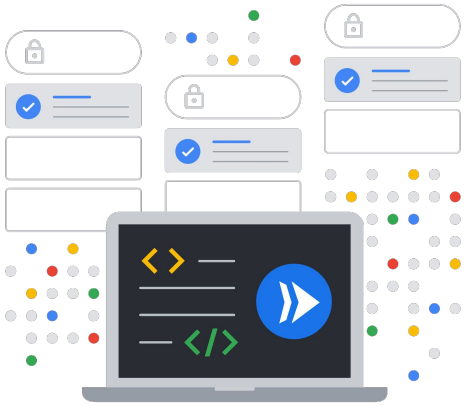
A region is a data center. For example: Council Bluffs (Iowa, North America)

Every region has three or more zones.

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

Cloud Run use cases

- Web applications, public APIs or websites
- Microservices-based applications that communicate using direct or asynchronous messages
- Event processing
- Scheduled tasks shorter than 60 minutes



Other example use cases found toward the bottom of this page:
https://cloud.google.com/run/

Containers in GCP = GKE or Cloud Run

OR

Exam Tip: *How to differentiate between GKE and Cloud Run?*

- *Cloud Run is fully serverless (GKE Standard was not… but Autopilot is…)*
- *Cloud Run are best when your biggest priority is time to market (fast development, deployment, scaling) and want to remove the ops and infra management from the process, or do not have a team to orchestrate and manage containers.*
- *98% of new Cloud Run users are able to code, build, and deploy an app within 5 minutes*

Google Cloud

So Cloud Run is an alternative to GKE in some cases, like if you have stateless workloads, if you'd like to deploy your image and expose this microservice in a couple of minutes; it you need to deploy your Proof Of Concept next day and you'd like to focus on your code instead of creating your cluster and so on.

Narration:

- At Google when we think Containers, we think GKE vs Cloud Run
- GKE is Google's Container Platform, it gives you complete control over every aspect of container orchestration, from networking, to storage, to how you set up observability.
- Serverless containers with Cloud Run are best when your biggest priority is time to market (fast development, deployment, scaling) and want to remove the ops and infra management from the process, or do not have a team to orchestrate and manage containers.
- Serverless is a higher abstraction level, therefore helps deploy faster to market, but also less control on infrastructure.
- We have many customers that start on GKE, then realize that they do not want to manage containers, and switch over to Cloud Run.
- Cloud Run is not based on kubernetes and does not require fancy kubtl commands. Many customers mention this allows their dev teams to start developing faster.

Containers are a method of packaging in which applications can be abstracted from the environment in which they actually run

Value Proposition of Containers: Containerization allows developers to focus on their applications, while IT operations teams can focus on deployment and management without bothering with application details such as specific software versions and configurations specific to the app.

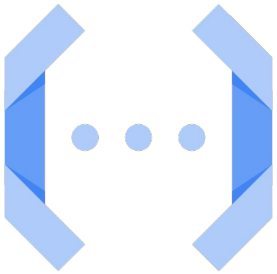
- **Less overhead:** containers require less system resources because they don't include OS images
- **Reduce IT management resources**
- **Portability:** Applications running in containers can be deployed easily to multiple different OS and platforms
- **Efficiency:** Containers allow applications to be more rapidly deployed, patched, or scaled
- **Better app development** experience

Cloud (Run) Functions
Lambda

Function-as-a-Service (FaaS)
They all allow you to run snippets of code without provisioning servers, scaling to zero when not in use.

Cloud (Run) Functions (Lambda)

- Create single-purpose functions that respond to events without a server.
  - Event examples: New instance created, file added to Cloud Storage (S3).
- Written in Javascript (Node.js), Python, Go, .NET, Java, PHP, PowerShell, C#, container (any language) custom runtime (any language), or Ruby.
  - Executes in managed environment on Google Cloud (AWS).



GCP: **Container-native**. Google has effectively merged Cloud Functions with Cloud Run. "Cloud Run functions" is now just a developer experience layer that takes your code, automatically containerizes it, and deploys it to Cloud Run.
biggest advantage: High concurrency (One instance can handle multiple simultaneous requests) vs Lambda: Typically 1 instance = 1 request at a time
High maximum runtime (60 minutes vs AWS 15 mins)

Source: Demo Template
Cloud Functions is a lightweight, event-based, asynchronous compute solution that allows you to create small, single-purpose functions that respond to cloud events without the need to manage a server or a runtime environment. You can use these functions to construct applications from bite-sized business logic. You can also use Cloud Functions to connect and extend cloud services.

You are billed, to the nearest 100 milliseconds, only while your code is running.

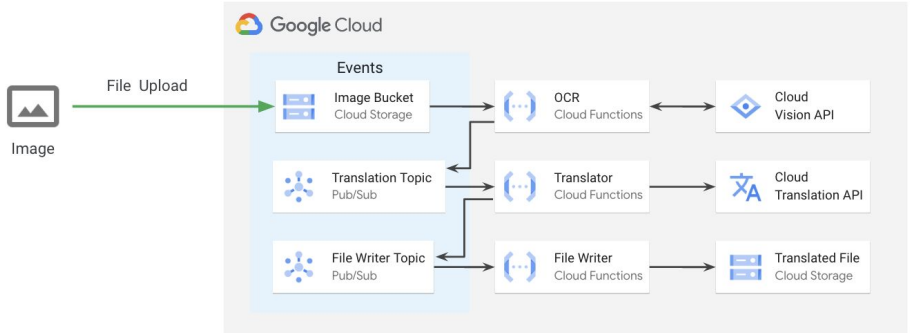
Cloud Functions are written in Javascript (Node.js), Python or Go and execute in a managed Node.js environment on Google Cloud. Events from Cloud Storage and Pub/Sub can trigger Cloud Functions asynchronously, or you can use HTTP invocation for synchronous execution.

Cloud Events are things that happen in your cloud environment. These might be things like changes to data in a database, files added to a storage system, or a new virtual machine instance being created.

Events occur whether or not you choose to respond to them. Creating a response to an event is done with a trigger. A trigger is a declaration that you are interested in a certain event or set of events. You create triggers to capture events and act on them.

Useful for event-driven, highly scalable microservices

- Can be triggered by changes in a storage bucket, Pub/Sub messages, web requests and other types of events
- Completely managed, scalable, and inexpensive



Tutorial: https://cloud.google.com/functions/docs/tutorials/ocr

AWS Lambda and Cloud Functions

AWS Lambda

A function can have multiple triggers.

Non-natively supported languages can still run using the Runtime API.

Cloud Functions

A function can only have one trigger, but multiple functions can have the same trigger.

Does not support additional runtimes; they are supported in Cloud Run.

Let's think about responding to cloud events. How do you perform automated actions in response to triggers in a distributed fashion?

You may have used AWS Lambda to do this. AWS Lambda contains features found in Cloud Functions and features found in Cloud Run, such as running compiled code from any language.

These are important points to remember:

- A function in AWS Lambda can have multiple triggers. On the other hand, a function in Cloud Functions can only have one trigger, but multiple functions can have the same trigger.
- Another key difference between AWS Lambda and Cloud Functions is language support. Non-natively supported languages can still run in AWS Lambda using the Runtime API. Cloud Functions does not support additional runtimes; they are supported in Cloud Run.

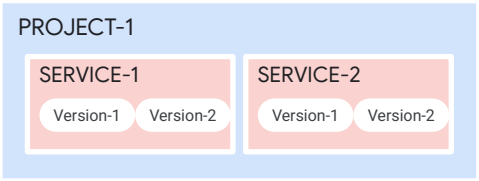
Refer to the additional resource ‘Comparison of features in AWS Lambda and Cloud Functions’ to learn more about some of the differences and similarities between Cloud Functions and AWS Lambda, and where some features of AWS Lambda are addressed in Google Cloud using Cloud Run.

App Engine
Elastic Beanstalk

Use case: They aim to abstract away the underlying infrastructure (servers, networking, OS patching) so developers can focus solely on deploying code.

App Engine: fully managed, serverless compute for low latency, highly scalable applications

- Developers focus on code, Google manages infrastructure
- Supports Node.js, Java, Ruby, C#, Go, Python, or PHP
- Autoscales automatically depending on load
- 2 Types
  - App Engine Standard
    - Limited language support
    - Free tier
  - App Engine Flexible
    - Google Cloud builds a Docker container
    - Runs on Compute Engine - no free tier



– Elastic Beanstalk is the equivalent of App Engine Flex! Will be mentioned in 1-2 slides, do not explain yet!

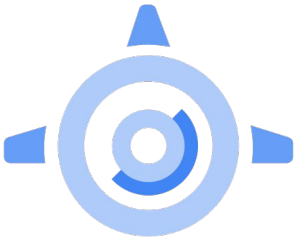
App Engine documentation
https://cloud.google.com/appengine/docs

App Engine is a fully managed, serverless application platform supporting the building and deploying of applications. Applications can be scaled seamlessly from zero upward without having to worry about managing the underlying infrastructure. App Engine was designed for microservices. For configuration, each Google Cloud project can contain one App Engine application, and an application has one or more services. Each service can have one or more versions, and each version has one or more instances. App Engine supports traffic splitting so it makes switching between versions and strategies such as canary testing or A/B testing simple. The diagram on the right shows the high-level organization of a Google Cloud project with two services, and each service has two versions. These services are independently deployable and versioned.

Proprietary + Confidential

App Engine Standard (no AWS equivalent) environment

- Easily deploy your applications.
- Autoscale workloads in seconds to meet demand.
- Economical
  - Free daily quota
  - Usage based pricing
- SDKs for development, testing and deployment.
  - Specific versions of Java, Python, PHP, Go, Node.js, and Ruby are supported.
  - Your application must conform to certain sandbox constraints dependant on runtime.
  - Can scale to 0 if no load



Google Cloud

AWS: no equivalent (can't scale to 0, moderate scaling speed - minutes)

Source: Demo Template

The App Engine standard environment is based on container instances running on Google's infrastructure. Containers are preconfigured with one of several available runtimes. Each runtime also includes libraries that support App Engine standard APIs. For many applications, the standard environment runtimes and libraries may be all you need.

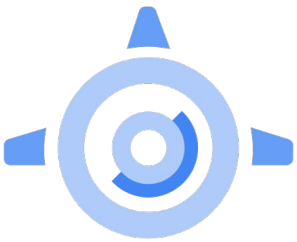
The App Engine standard environment makes it easy to build and deploy an application that runs reliably even under heavy load and with large amounts of data. It includes the following features:

- Persistent storage with queries, sorting, and transactions.
- Automatic scaling and load balancing.
- Asynchronous task queues for performing work outside the scope of a request.
- Scheduled tasks for triggering events at specified times or regular intervals.
- Integration with other Google Cloud services and APIs.

Proprietary + Confidential

App Engine Flexible (Elastic Beanstalk) environment

- Build and deploy containerized apps with a click.
- No sandbox constraints.
- Can access App Engine and VPC 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.



Google Cloud

GAE Flexible is functionally almost identical to AWS Elastic Beanstalk. They both act as "wrappers" that manage standard Virtual Machines (VMs) and Docker containers on your behalf.

GAE Standard's ability to **scale to zero** and its near-instant scaling for sudden traffic spikes. But it has a cost (not super flexible regarding supported languages and other constraints)

Source: Demo Template

If the restrictions of App Engine standard environment's sandbox model don't work for you, but you still want to take advantage of the benefits of App Engine (like automatic scaling up and down), consider App Engine flexible environment. Instead of the sandbox, App Engine flexible environment lets you specify the container your application runs in.

Your application runs inside Docker containers on Google Compute Engine virtual machines (VMs). App Engine manages these Compute Engine machines for you. They're health-checked, healed as necessary, and you get to choose what geographical region they run in. And critical, backward-compatible updates to their operating systems are automatically applied. All this so that you can just focus on your code.

Microservices, authorization, SQL and noSQL databases, traffic splitting, logging, search, versioning, security scanning, memcache, and content delivery networks are all supported natively. In addition, the App Engine flexible environment allows you to customize your runtime and even the operating system of your virtual machine using Dockerfiles.

- *Runtimes:* The flexible environment includes native support for Go, Java 8, PHP 5/7, Python 2.7/3.6, .NET, Node.js, and Ruby. Developers can customize these runtimes or provide their own runtime, such as Ruby or PHP, by supplying a custom Docker image or Dockerfile from the open source community.
- *Infrastructure customization:* Because VM instances in the flexible environment are Compute Engine virtual machines, you can use SSH to connect to every single VM and Docker container for debugging purposes and further customization.
- *Performance:* Take advantage of a wide array of CPU and memory configurations. You can specify how much CPU and memory each instance of your application needs, and the flexible environment will provision the necessary infrastructure for you.

App Engine manages your virtual machines, ensuring that:

- Instances are health-checked, healed as necessary, and co-located with other module instances within the project.
- Critical, backward-compatible updates are automatically applied to the underlying operating system.
- VM instances are automatically located by geographical region according to the settings in your project. Google's management services ensure that all of a project's VM instances are co-located for optimal performance.
- VM instances are restarted on a weekly basis. During restarts, Google's management services will apply any necessary operating system and security updates.

App Engine flexible environment apps that use standard runtimes can access App Engine services: Datastore, Memcache, task queues, logging, users, and so on.

Comparing the App Engine environments

	Standard environment	Flexible environment (Elastic Beanstalk)
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); can be used with EBS
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 (instance hours)	Pay for resource allocation based on GCE (EC2) usage; no automatic shutdown
Minimum instances	0	1

Source: Demo Template

Here's a side-by-side comparison of the standard and flexible environments. Notice that the standard environment starts up instances of your application faster, but that you get less access to the infrastructure in which your application runs. For example, the flexible environment lets you ssh into the virtual machines on which your application runs; it lets you use local disk for scratch space; it lets you install third-party software; and it lets your application make calls to the network without going through App Engine. On the other hand, the standard environment's billing can drop to zero for a completely idle application.

What if i need a trigger-based systems?

No server management

Deploy your code and let Google run and scale it for you.

Runs code in response to events

It allows you to trigger your code from Google Cloud, Firebase, Google Assistant, or many other services or call it directly from any web, mobile, or backend application via HTTP

Pay only for what you use

You are only billed for your function's execution time, metered to the nearest 100 milliseconds(rounded up). You pay nothing when your function is idle.

Make sure to...

Enjoy the journey as much as the destination!

Now that you know about the overall setup of this course and how to use the workbook, let's get started by exploring section 1 of the exam guide.