Preparing for the Associate Cloud Engineer (ACE) Exam: Ensuring successful operation of a cloud solution

1. Section 4.1 - Managing Compute Engine resources
2. Section 4.2 - Managing Kubernetes Engine resources
3. Section 4.3 - Managing App Engine resources
4. Section 4.4 - Managing data solutions

5 Section 4.5 - Managing networking resources

6 Section 4.6 - Monitoring and logging

1 Section 4.1 - **Managing Compute Engine resources**

2 Section 4.2 - Managing Kubernetes Engine resources

3 Section 4.3 - Managing App Engine resources

4 Section 4.4 - Managing data solutions

Exam Guide: Section 4.1

4.1 Managing Compute Engine resources. Tasks include:

- Managing a single VM instance (e.g., start, stop, edit configuration, or delete an instance).
- SSH/RDP to the instance.
- Attaching a GPU to a new instance and installing CUDA libraries.
- Viewing current running VM Inventory (instance IDs, details).
- Working with snapshots (e.g., create a snapshot from a VM, view snapshots, delete a snapshot).
- Working with Images (e.g., create an image from a VM or a snapshot, view images, delete an image).
- Working with Instance Groups (e.g., set auto scaling parameters, assign instance template, create an instance template, remove instance group).
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, GCloud SDK).

Exam Guide: Section 4.1

4.1 Managing Compute Engine resources. Tasks include:

- Managing a single VM instance (e.g., start, stop, edit configuration, or delete an instance).
- SSH/RDP to the instance.
- Attaching a GPU to a new instance and installing CUDA libraries.
- Viewing current running VM Inventory (instance IDs, details).
- **Working with snapshots (e.g., create a snapshot from a VM, view snapshots, delete a snapshot).**
- **Working with Images (e.g., create an image from a VM or a snapshot, view images, delete an image).**
- Working with Instance Groups (e.g., set auto scaling parameters, assign instance template, create an instance template, remove instance group).
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, GCloud SDK).

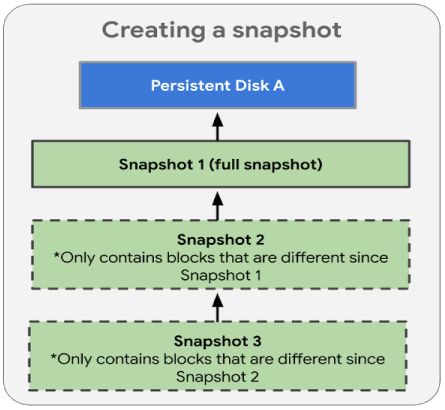
VM Images Overview

Use operating system images to create boot disks for your instances. You can use one of the following image types:

- **Public images** are provided and maintained by Google, open-source communities, and third-party vendors. By default, all projects have access to these images and can use them to create instances.
- **Custom images** are available only to your project. You can create a custom image from boot disks and other images. Then, use the custom image to create an instance.

You can use most public images at no additional cost, but there are some premium images that do add additional cost to your instances.
Custom images that you import to Compute Engine add no cost to your instances, but do incur an image storage charge while you keep your custom image in your project.
Some images are capable of running containers on Compute Engine.
Support for Compute Engine provided public OS images are subject to the lifecycle of the respective OS.

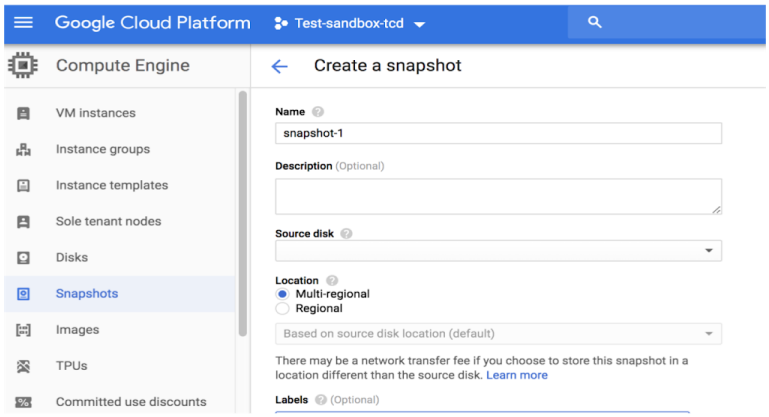
Creating Snapshots



Snapshots are incremental and automatically compressed, so you can create regular snapshots on a persistent disk faster and at a much lower cost than if you regularly created a full image of

the disk. Incremental snapshots work in the following manner:
The first successful snapshot of a persistent disk is a full snapshot that contains all the data on the persistent disk.
The second snapshot only contains any new data or modified data since the first snapshot. Data that hasn't changed since snapshot 1 isn't included. Instead, snapshot 2 contains references to snapshot 1 for any unchanged data.
Snapshot 3 contains any new or changed data since snapshot 2 but won't contain any unchanged data from snapshot 1 or 2. Instead, snapshot 3 contains references to blocks in snapshot 1 and snapshot 2 for any unchanged data.
This repeats for all subsequent snapshots of the persistent disk. Snapshots are always created based on the last successful snapshot taken.

Creating Snapshots



Compute Engine stores multiple copies of each snapshot across multiple locations with automatic checksums to ensure the integrity of your data. Use IAM roles to share snapshots across projects.
To see a list of snapshots available to a project, use the gcloud compute snapshots list command:
gcloud compute snapshots list
To list information about a particular snapshot, such as the creation time, size, and source disk, use the gcloud compute snapshots describe command:
gcloud compute snapshots describe [SNAPSHOT_NAME]
where [SNAPSHOT_NAME] is the name of the snapshot for which you want to see the snapshot information.

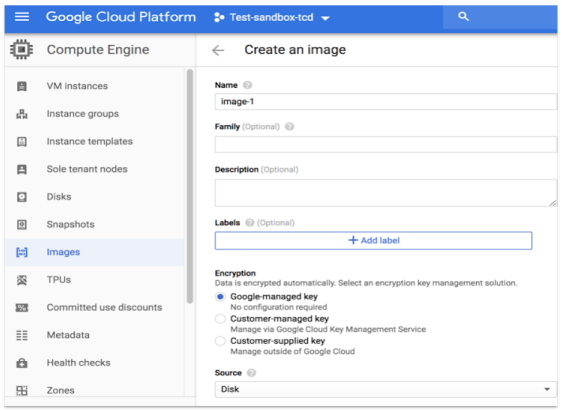
Creating Custom Images

You can create disk images from the following sources:

- A persistent disk, even while that disk is attached to an instance.
- A snapshot of a persistent disk
- Another image in your project
- An image that is shared from another project
- A compressed RAW image in Google Cloud Storage

A custom image is a boot disk image that you own and control access to.
If you regularly update your custom images with newer configurations and software, you can group those images into an image family.
The image family always points to the most recent image in that family so your instance templates and scripts can use that image without having to update references to a specific image version.

Creating Custom Images



In the Google Cloud Platform Console, go to the Create an image page
Specify the source from which you want to create an image. This can be a persistent disk, a snapshot, another image, or a disk.raw file in Google Cloud Storage.
Specify the properties for your image. For example, you can specify an image family name for your image to organize this image as part of an image family.
If you are creating an image from a disk attached to a running image, check "Force creation from running instance" to confirm that you want to create the image while the instance is running.
Click Create to create the image.

1 Section 4.1 - Managing Compute Engine resources

2 **Section 4.2 - Managing Kubernetes Engine resources**

3 Section 4.3 - Managing App Engine resources

4 Section 4.4 - Managing data solutions

Exam Guide: Section 4.2

4.2 Managing Kubernetes Engine resources. Tasks include:

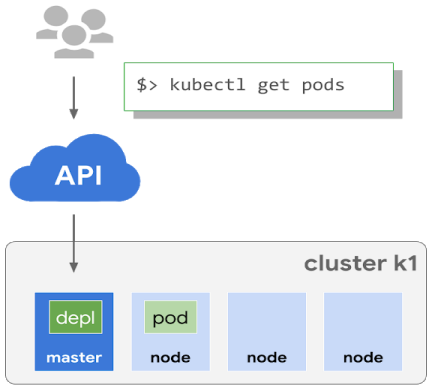
- Viewing current running cluster inventory (nodes, pods, services).
- Browsing the container image repository and viewing container image details.
- Working with nodes (e.g., add, edit, or remove a node).
- Working with pods (e.g., add, edit, or remove pods).
- Working with services (e.g., add, edit, or remove a service).
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Exam Guide: Section 4.2

4.2 Managing Kubernetes Engine resources. Tasks include:

- Viewing current running cluster inventory (nodes, pods, services).
- Browsing the container image repository and viewing container image details.
- Working with nodes (e.g., add, edit, or remove a node).
- **Working with pods (e.g., add, edit, or remove pods).**
- Working with services (e.g., add, edit, or remove a service).
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

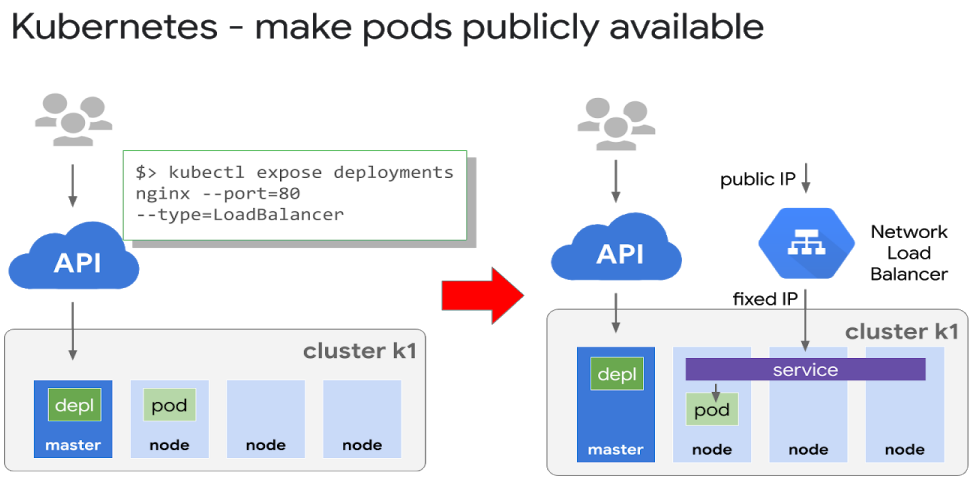
Kubernetes - show running pods



A Deployment represents a group of replicas of the same Pod and keeps your Pods running even when nodes they run on fail. It could represent a component of an application or an entire app. In this case, it's the nginx web server.

To see the running Pods, run the command:

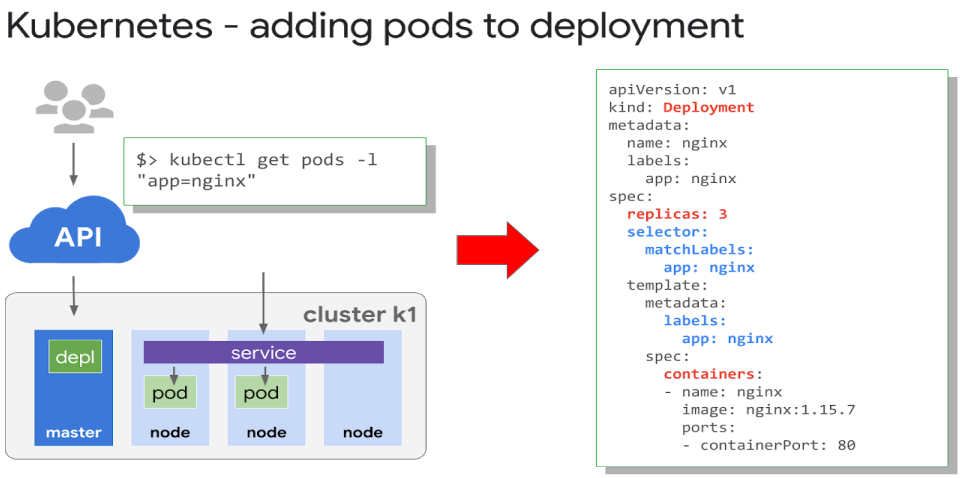
$ kubectl get pods



By default, Pods in a Deployment are only accessible inside your GKE cluster. To make them publicly available, you can connect a load balancer to your Deployment by running the kubectl expose command

Kubernetes then creates a Service with a fixed IP for your Pods, and a controller says "I need to attach an external load balancer with a public IP address to that Service so others outside the cluster can access it".

In GKE, the load balancer is created as a Network Load Balancer.



Instead of issuing commands, you can provide a configuration file that tells Kubernetes what you want your desired state to look like, and Kubernetes figures out how to do it.

To get the file, you can run a kubectl get pods command as shown here to get a .yaml file.

In this case, the .yaml file declares you want three replicas of your nginx Pod.

It also defines a selector field so your Deployment knows how to group specific Pods as replicas, and you add a label to the Pod template so they get selected.

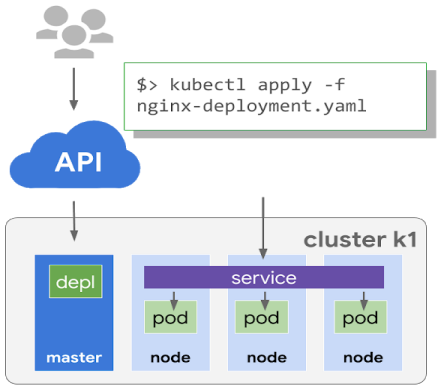
Kubernetes - adding pods to deployment

```yaml
apiVersion: v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.7
        ports:
        - containerPort: 80
```

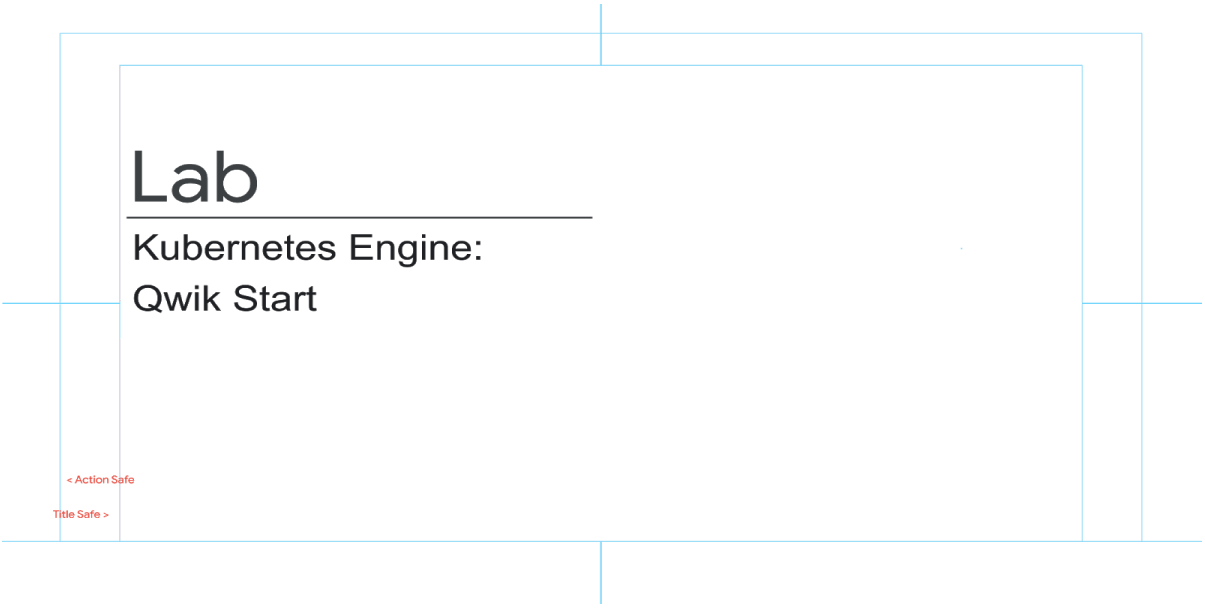
```yaml
apiVersion: v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.10.0
        ports:
        - containerPort: 80
```

To run five replicas instead of three, all you do is update the Deployment config file.

Kubernetes - adding pods to deployment



And then run the kubectl apply command to use the updated config file.



Google Kubernetes Engine (GKE) provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure.
The Kubernetes Engine environment consists of multiple machines (specifically Google Compute Engine instances) grouped together to form a container cluster.
In this lab, you will get hands on practice with container creation and application deployment with GKE.

1 Section 4.1 - Managing Compute Engine resources

2 Section 4.2 - Managing Kubernetes Engine resources

3 **Section 4.3 - Managing App Engine resources**

4 Section 4.4 - Managing data solutions

Exam Guide: Section 4.3

4.3 Managing App Engine resources. Tasks include:

- Adjusting application traffic splitting parameters.
- Setting scaling parameters for autoscaling instances.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Exam Guide: Section 4.3

4.3 Managing App Engine resources. Tasks include:

- **Adjusting application traffic splitting parameters.**
- Setting scaling parameters for autoscaling instances.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

App Engine instances overview

App Engine Instances:

- Are the basic building blocks of App Engine
- Provide all the resources necessary to host your application
- Are the means by which App Engine scales your application to meet demand
- Can be resident or dynamic
  - Manual scaling uses resident instances
  - Basic or automatic scaling uses dynamic instances

Instances are the basic building blocks of App Engine, providing all the resources needed to successfully host your application. This includes the language runtime, the App Engine APIs, and your application's code and memory. Each instance includes a security layer to ensure that instances cannot inadvertently affect each other.

Instances are the computing units that App Engine uses to automatically scale your application. At any given time, your application can be running on one instance or many instances, with requests being spread across all of them.

Instances are resident or dynamic. A dynamic instance starts up and shuts down automatically based on the current needs. A resident instance runs all the time, which can improve your application's performance. Both dynamic and resident instances instantiate the code included in an App Engine service version.

If you use manual scaling for an app, the instances it runs on are resident instances. If you use either basic or automatic scaling, your app runs on dynamic instances.

App Engine instance scaling

- App Engine monitors the incoming traffic for each instance
- If using an automated scaling method, keep in mind new instances can be added very quickly
- You can control the rate in the App Engine task queue
- Scaling also happens in reverse when traffic load decreases

When you upload a version of a service, the app.yaml specifies a scaling type and instance class that apply to every instance of that version. The scaling type controls how instances are created. The instance class determines compute resources (memory size and CPU speed) and pricing. There are three scaling types: manual, basic, and automatic. The available instance classes depend on the scaling type.

Manual scaling

A service with manual scaling uses resident instances that continuously run the specified number of instances irrespective of the load level. This allows tasks such as complex initializations and applications that rely on the state of the memory over time.

Automatic scaling

Auto scaling services use dynamic instances that get created based on request rate, response latencies, and other application metrics. However, if you specify a number of minimum idle instances, that specified number of instances run as resident instances while any additional instances are dynamic.

Basic Scaling

A service with basic scaling uses dynamic instances. Each instance is created when the application receives a request. The instance will be turned down when the app becomes idle. Basic scaling is ideal for work that is intermittent or driven by user activity.

1 Section 4.1 - Managing Compute Engine resources

2 Section 4.2 - Managing Kubernetes Engine resources

3 Section 4.3 - Managing App Engine resources

4 **Section 4.4 - Managing data solutions**

Exam Guide: Section 4.4

4.4 Managing data solutions. Tasks include:

- Executing queries to retrieve data from data instances (e.g., Cloud SQL, BigQuery, Cloud Spanner, Cloud Datastore, Cloud Bigtable, Cloud Dataproc).
- Estimating costs of a BigQuery query.
- Backing up and restoring data instances (e.g., Cloud SQL, Cloud Datastore, Cloud Dataproc).
- Reviewing job status in Cloud Dataproc or BigQuery
- Moving objects between Cloud Storage buckets.
- Converting Cloud Storage buckets between storage classes.
- Setting object lifecycle management policies for Cloud Storage buckets.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Exam Guide: Section 4.4

4.4 Managing data solutions. Tasks include:

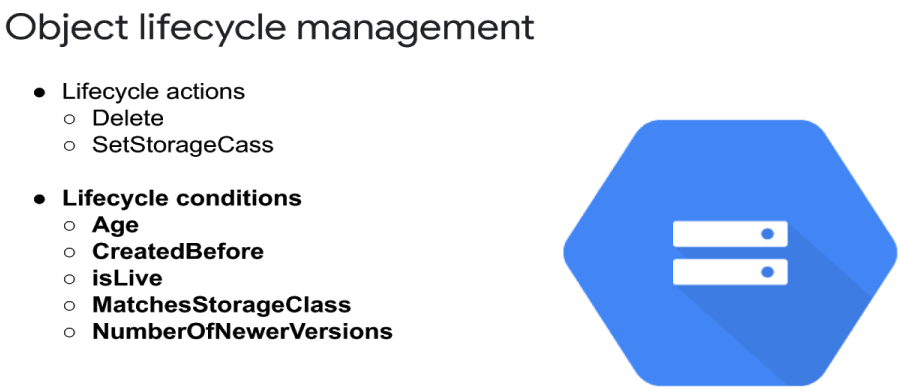
- Executing queries to retrieve data from data instances (e.g., Cloud SQL, BigQuery, Cloud Spanner, Cloud Datastore, Cloud Bigtable, Cloud Dataproc).
- Estimating costs of a BigQuery query.
- Backing up and restoring data instances (e.g., Cloud SQL, Cloud Datastore, Cloud Dataproc).
- Reviewing job status in Cloud Dataproc or BigQuery
- Moving objects between Cloud Storage buckets.
- Converting Cloud Storage buckets between storage classes.
- **Setting object lifecycle management policies for Cloud Storage buckets.**
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Object lifecycle management

- **Lifecycle actions**
  - **Delete**
  - **SetStorageCass**

You can assign a lifecycle management configuration to a bucket. The configuration contains a set of rules which apply to current and future objects in the bucket. When an object meets the criteria of one of the rules, Cloud Storage automatically performs a specified action on the object. Here are some example use cases:
Downgrade the storage class of objects older than 365 days to Coldline Storage.
Delete objects created before January 1, 2013.
Keep only the 3 most recent versions of each object in a bucket with versioning enabled.

The following actions are supported for a lifecycle rule:
Delete: Delete live and/or archived objects. This action can be applied to both versioned and non-versioned objects. In a bucket with versioning enabled, deleting a live object archives the object, while deleting an archived object deletes the object permanently.
SetStorageClass: Change the storage class of live and/or archived objects. This action can be applied to both versioned and non-versioned objects.



The following conditions are supported for a lifecycle rule:
Age: This condition is satisfied when an object reaches the specified age (in days).
CreatedBefore: This condition is satisfied when an object is created before midnight of the specified date in UTC.
IsLive: If the value is true, this lifecycle condition matches only live objects; if the value is false, it matches only archived objects.
MatchesStorageClass: This condition is satisfied when an object in the bucket is stored as the specified storage class.
NumberOfNewerVersions: Relevant only for versioned objects. If the value of this condition is set to N, an object satisfies the condition when there are at least N versions (including the live version) newer than it. For live objects, the number of newer versions is considered to be 0.

5 **Section 4.5 - Managing networking resources**

6 Section 4.6 - Monitoring and logging

Exam Guide: Section 4.5

4.5 Managing networking resources. Tasks include:

- Adding a subnet to an existing VPC.
- Expanding a CIDR block subnet to have more IP addresses.
- Reserving static external or internal IP addresses.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Exam Guide: Section 4.5

4.5 Managing networking resources. Tasks include:

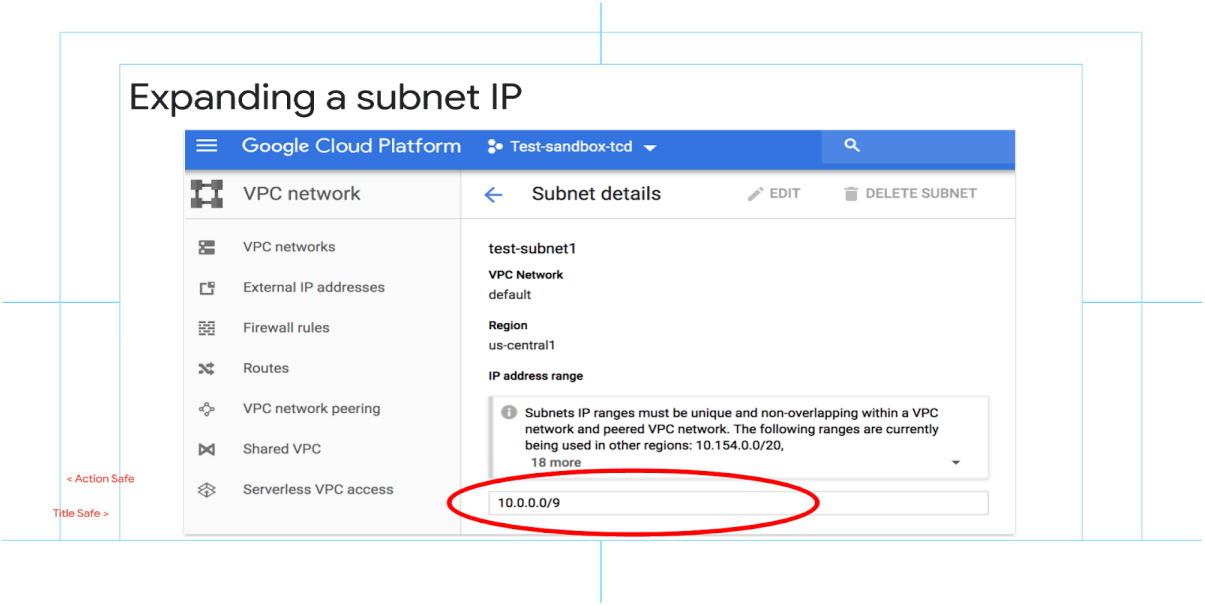
- Adding a subnet to an existing VPC.
- **Expanding a CIDR block subnet to have more IP addresses.**
- Reserving static external or internal IP addresses.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Subnet IP management overview

- Each subnet has a primary range, which does not have to be contiguous with the secondary range(s)
- All primary and secondary ranges must be unique
- You can expand a subnet, but not shrink it, once it has been created
- The longest subnet mask you can use is /29 (eight IP addresses)

Each subnet must have a primary range, and, optionally, up to five secondary range for alias IP. Primary and secondary IP ranges must be RFC 1918 addresses.
Within a VPC network, all primary and secondary IP ranges must be unique, but they do not need to be contiguous. For example, the primary range of a subnet can be 10.0.0.0/24 while the primary range of another subnet in the same network can be 192.168.0.0/16.
The primary IP range for the subnet can be expanded, but not replaced or shrunk, after the subnet has been created.
You can remove and replace a subnet's secondary IP address range only if no instances are

using that range.
The minimum primary or secondary range size is eight IP addresses. In other words, the longest subnet mask you can use is /29.



You can expand the primary IP range of an existing subnet by modifying its subnet mask, setting the prefix length to a smaller number. The proposed new primary IP range of the subnet must follow the subnet rules.
When expanding the IP range of an automatically created subnet in an auto mode network (or in a custom mode network that was previously an auto mode network), the broadest prefix (subnet mask) you can use is /16. Any prefix broader than /16 would conflict with the primary IP ranges of the other automatically created subnets.
https://cloud.google.com/vpc/docs/using-vpc
Go to the VPC networks page in the Google Cloud Platform Console.
GO TO THE VPC NETWORKS PAGE All networks and subnets in your project are presented in a hierarchical view, where subnets are shown as entries within networks.
To focus on subnets for a particular network, click the name of a network. On its VPC network details page, click the name of a subnet in the Subnets tab to view its Subnet details page.
Click Edit.
Enter a new, broader CIDR block in the IP address range field.
Click Save.

5 Section 4.5 - Managing networking resources

6 **Section 4.6 - Monitoring and logging**

Exam Guide: Section 4.6

4.6 Monitoring and logging. Tasks include:

- Creating Stackdriver alerts based on resource metrics.
- Creating Stackdriver custom metrics.
- Configuring log sinks to export logs to external systems (e.g., on premises or BigQuery).
- Viewing and filtering logs in Stackdriver.
- Viewing specific log message details in Stackdriver.
- Using cloud diagnostics to research an application issue (e.g., viewing Cloud Trace data, using Cloud Debug to view an application point-in-time).
- Viewing Google Cloud Platform status.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Exam Guide: Section 4.6

4.6 Monitoring and logging. Tasks include:

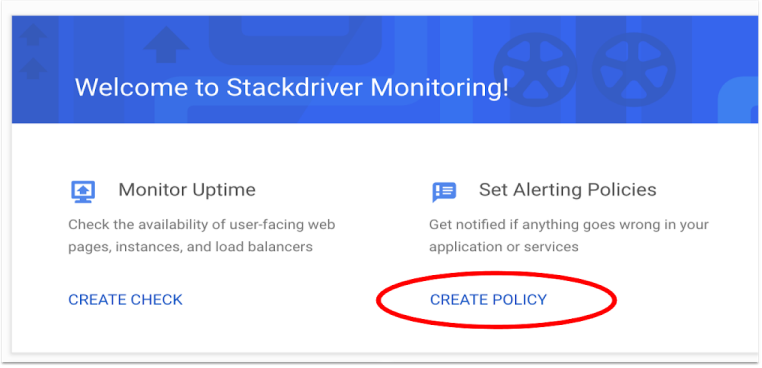
- **Creating Stackdriver alerts based on resource metrics.**
- Creating Stackdriver custom metrics.
- Configuring log sinks to export logs to external systems (e.g., on premises or BigQuery).
- Viewing and filtering logs in Stackdriver.
- Viewing specific log message details in Stackdriver.
- Using cloud diagnostics to research an application issue (e.g., viewing Cloud Trace data, using Cloud Debug to view an application point-in-time).
- Viewing Google Cloud Platform status.
- Working with management interfaces (e.g., Cloud Console, Cloud Shell, Cloud SDK).

Built-in monitoring with Stackdriver

- Many GCP services have stackdriver monitoring integration built in:
  - Compute Engine, App Engine, Kubernetes Engine, Cloud SQL, Datastore, BigQuery, Networking, Pub/Sub, and several more

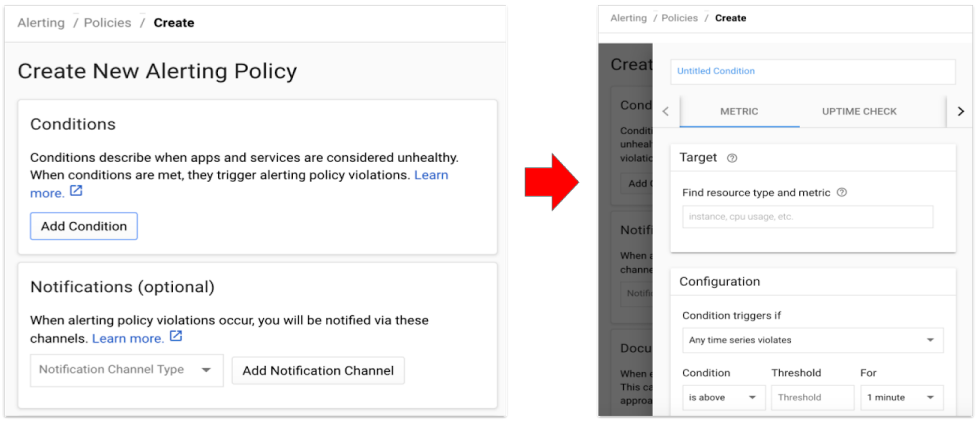
Several services, including App Engine flexible, App Engine standard, and Kubernetes engine have stackdriver monitoring built in.
For other services without stackdriver monitoring built in, such as compute engine, there are monitoring agents that can be installed.
Stackdriver monitoring agents even exist for Amazon EC2 and on-prem services.
This allows stackdriver to provide a multicloud/hybrid monitoring solution.

Creating custom alerts on Stackdriver



You can create and manage alerting policies with the Stackdriver Monitoring console, the Stackdriver Monitoring API, and Cloud SDK.
Each policy specifies the following:
Conditions that identify an unhealthy state for a resource or a group of resources.
Optional notifications sent through email, SMS, or other channels to let your support team know a resource is unhealthy.
Optional documentation that can be included in some types of notifications to help your support team resolve the issue.
When events trigger conditions in one of your alerting policies, Stackdriver Monitoring creates and displays an incident in the Stackdriver Monitoring console. If you set up notifications, Stackdriver Monitoring also sends notifications to people or third-party notification services. Responders can acknowledge receipt of the notification, but the incident remains open until resources are no longer in an unhealthy state.

Creating custom alerts on Stackdriver



In the Google Cloud Platform Console, choose your project from the drop-down list.
Go to Stackdriver > Monitoring. (If this is the first time you've used the project in Stackdriver, you must first create a Workspace.)
Go to Alerting > Create a Policy.
The following sections under Configuration help you describe and customize your alerting policy:
Conditions
Notifications
Documentation
Name this policy
Fill in the conditions under which you wish to have an alert sent and what kind of notification you wish to receive.
Click Save.

Suggested study resources for this section

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

Creating, deleting, deprecating custom images: https://cloud.google.com/compute/docs/images/create-delete-deprecate-private-images

Creating snapshots: https://cloud.google.com/compute/docs/disks/create-snapshots

How App Engine instances are managed: https://cloud.google.com/appengine/docs/standard/python/how-instances-are-managed

Object lifecycle management: https://cloud.google.com/storage/docs/lifecycle

Expanding subnets: https://cloud.google.com/vpc/docs/using-vpc#expand-subnet

Introduction to Alerting: https://cloud.google.com/monitoring/alerts/

Managing alerting policies: https://cloud.google.com/monitoring/alerts/using-alerting-ui