Module 1 – Intro and fundamentals

## Slide 3:-

Google offers multiple certifications, staring from fundamentals to professionals. Also offers role specific certifications

Open the link and go through course content for this course

## Slide 5 :-

* Trust and Security :- by default cannot communicate between different regions, segregation of resources of different organization.
* Open cloud Platform :- Provides integrations with multiple tools and architectures like Kubernetes environment. Allows onboarding multiple google open source tools instead of cloud native solutions
* Global Network Infrastructure :- Very big network connected directly via high speed optical fibre (Open the link and show connectivity diagram)
* AI Driven Cloud :- Multiple AI services, data science, machine learning modules provided, easy integration with different services

## Slide 8:-

* **Distance**: Choose zones based on the location of your customers and where the data is supposed to live. Store resources in zones that are closer to your point of service in order to keep network latency low.
* **Communication:** It’s important to be mindful that communication across and within regions will incur different costs and happen at different speeds. Typically, communication within a region will be cheaper than communication across different regions.
* **Redundant Systems:** As we mentioned above, Google is big on the fact that you should deploy fault-tolerant systems with high availability in case of unexpected failures. Therefore, you should design any important systems with redundancy across multiple regions and zones. This is to mitigate any possible effects if your instances were to experience an unexpected failure.
* **Resource Distribution:** Zones are designed to be independent of one another so if one zone fails or becomes unavailable, you can transfer traffic to another zone in the same region to keep your services running.
* **Cost/Pricing:** You should always check the pricing to compare the cost between regions.

Module 2 :- Compute VM Instance

**Slide 1 :- Compute Engine**

**Google Compute Engine** is a virtual machine (VM) service that allows users to create and run VMS on Google’s infrastructure.

 Compute Engine provides a variety of VM types, including standard, high-memory, high-CPU, and custom machine types.

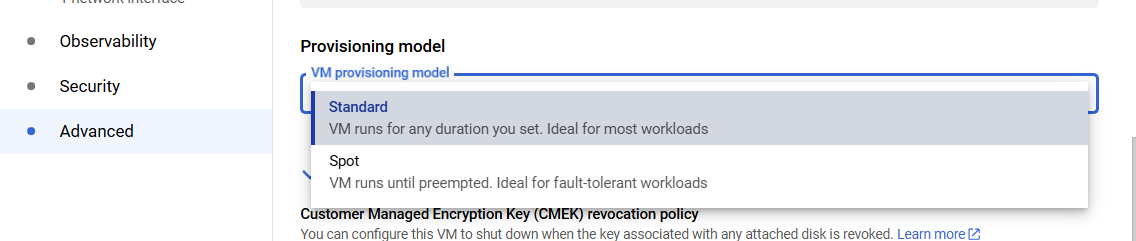
**Slide 2 :- Virtual machine**

In simple words, virtual machine is a digital version of physical computer.

Each virtual machine consists of a complete operating system and a set of virtualized hardware resources, including CPU, memory, storage, and network interfaces

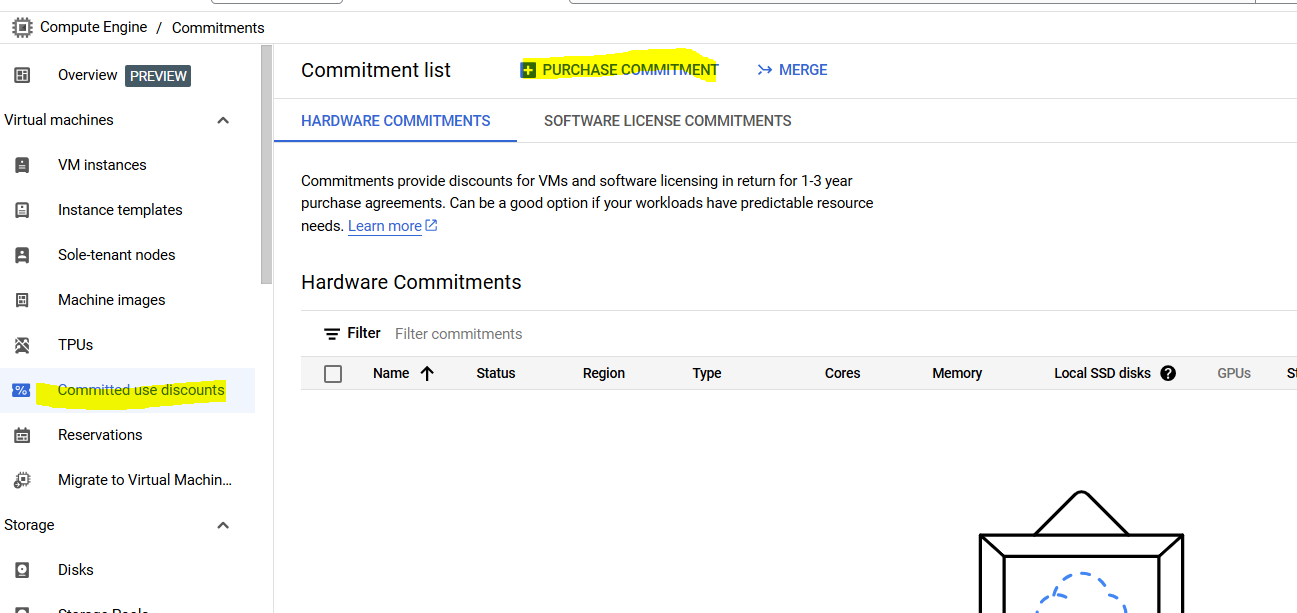
**Slide 13 Spot Instance:-**

Show below option at gcp while creating instance, but cannot create it in free account



Slide 14 :- **Committed use discounts**

Go through options in below



Moduel-4 Containers:-

Slide 8 and 9 (Docker Theory)

Docker is the most popular container environment in the world by far.

It's the de facto standard for containers.

It's so popular that today when people say container, they usually mean Docker.

It was released in 2013 and immediately gained a huge popularity, which just increased since then.

So how exactly does Docker works?

What is its architecture?

You can see here the main three parts of the Docker environment and let's go through them together.

First the Docker server or daemon.

This is a service that runs on the server and responsible for managing the Docker containers.

It turns them on or off.

It keeps tab of their activity.

It builds containers based on images, as we will see in a minute, and it exposes API for management

tools.

This is the heart of the Docker environment and without it nothing will work.

Next are the images.

Image is the set of definitions and software for a container to run.

If you're familiar with virtual machine images, that is the same idea.

So for example, you can have an image for the front end of the air application and an image of the

database of the e-commerce system.

Images are the building blocks of the containers.

Now note that images are static files and they don't run.

They are the basis for the containers, but they themselves just lie on the disk and waiting to be called.

Now where do we get the images from?

For that we have the container registries.

Container registry is basically a collection of images from where you can pull the image you want,

and either use it as is or customize it to your needs.

For example, you can pull an image for a baseline dotnet core application and then customize it by

copying into it your specific code components.

You then get a new custom image you can use as a basis for the container.

Note that container registries can be public, for example, Docker Hub, which is used by many of Docker

users for creating baseline images or private.

And then it's accessible only for you or the organization.

And then we have the containers themselves.

A container is an image that is built and run.

The container runs in a sandbox created and managed by the Docker daemon, and is a living, breathing

instance of the image.

The container is a full blown process and can do whatever it wants, assuming it has permissions.

Again, similar to VM, it's a piece of software that runs on the machine isolated from the host, but

can use its resources.

And then we have the client.

The client, which is a CLI command line interface, is where you can send instructions to the daemon.

This is your gateway to the whole functionality of Docker.

And using it you can pull images, customize them, run the containers, list them, shut them off,

check the status, etc. this is where the IT guys or the DevOps guys will spend most of their time when

working with Docker.

So that is the basic architecture of Docker and these are the main parts in it.

Now we said that one of the core capabilities of Docker is customizing images.

The ability to pull an image from a repository and then add resources to it is one of the major success

factors of Docker.

And this customization is done using Docker file.

So Docker file is a file that contains instructions for building custom images.

It can instruct the Docker daemon to copy files, run commands, change working directory, and more.

Docker files are usually quite small because there are so many baseline images to begin from, and the

customization required is not that big.

Here is an example of such file.

As you can see, it's quite easy to understand what's going on.

The file instructs Docker to copy some files, then run npm install set environment variable and copy

some more files.

This is a classic example of a docker file, and most files look quite similar to this one.