





GCP Compute Options:

Compute Engine	IAAS
Kubernetes Engine	IAAS/PAAS
App Engine	PAAS
Standard	
Flexible	
Cloud Functions	FAAS

GCP compute and processing options

					
	Compute Engine	Kubernetes Engine	App Engine Standard	App Engine Flexible	Cloud Functions
Language support	Any	Any	Python Node.js Go Java PHP	Python Node.js Go Java PHP Ruby .NET Custom Runtimes	Python Node.js Go
Usage model	IaaS	IaaS PaaS	PaaS	PaaS	Microservices Architecture
Scaling	Server Autoscaling	Cluster	Autoscaling managed servers		Serverless
Primary use case	General Workloads	Container Workloads	Scalable web applications Mobile backend applications		Lightweight Event Actions

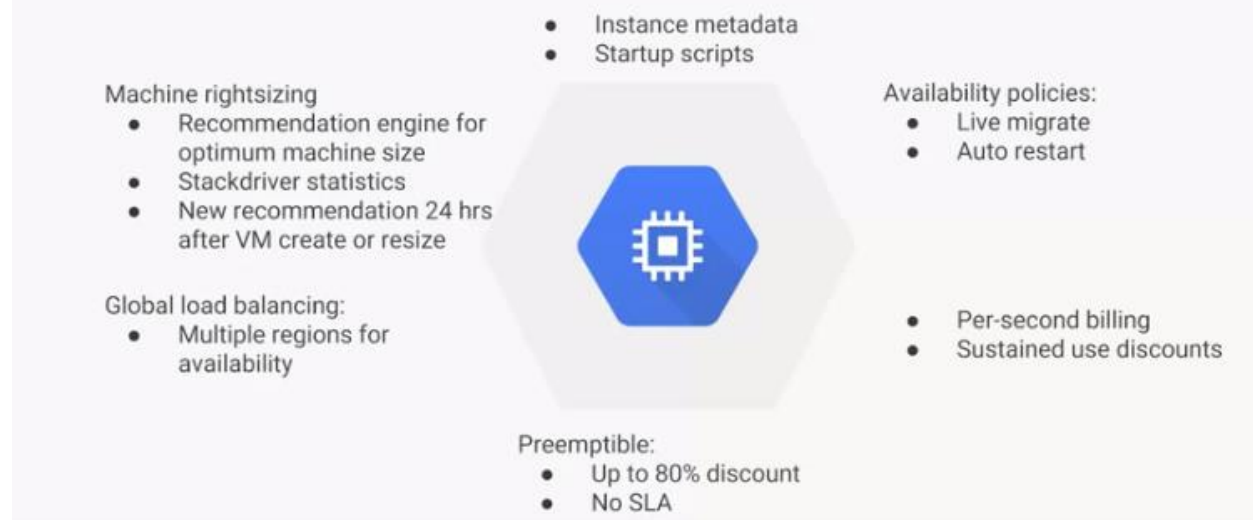
Compute Engine: IAAS (VCPU's, RAM, Disk, Nw, OS)

Predefined or custom machine types:

- vCPUs (cores) and Memory (RAM)
- Persistent disks: HDD, SSD, and Local SSD
- Networking
- Linux or Windows

Features of a Compute Engine:

Compute Engine features



Virtual Machine types:

Ondemand –		cost more ()	100 \$
Purchased /Committed usage instances		- 1 year/3 year/5 year	40\$
Preemptible	Instances	Unused Capacity from GCP /region/Zone	20\$
<ul style="list-style-type: none">• High Cpu• High Memory• Standard• Shared core Machine Types• Customize the VM based on Business requirement			

Important Note from **On-premises to cloud Work Load Mapping**

1 Physical Core in On-premises 2 Vcpu

1 Vcpu → 1 Hyperthreaded Core

Network front 1 VCPU corresponds to 2GBps

Max thro;put 16 Gbps or 8 Vcpu

Storage:

Persistent Disk (Standard, SSD, local SSD)

Networking:

Robust networking features

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

Compute engine Pricing

Per-second billing, sustained use discounts

- 1 minute minimum

Preemptible instances

- Live at most 24 hours
- Can be pre-empted with a 30-second notification via API
- Up to 80% discount

Custom machine types

- Customize amount of memory and CPU

Recommendation Engine

- Notifies you of underutilized instances

How to access a VM – Linux:

SSH can be from Console , Cloud Shell, through SDK

SSH from a computer, 3rd party Client and generate Key-value pair

You should allow port 22 –ssh in firewall rule ..

How to access a VM – Windows:

You need Windows Username and Password to be set up ..

should allow tcp 3389 –TCP in firewall rule ..(to allow RDP connection)

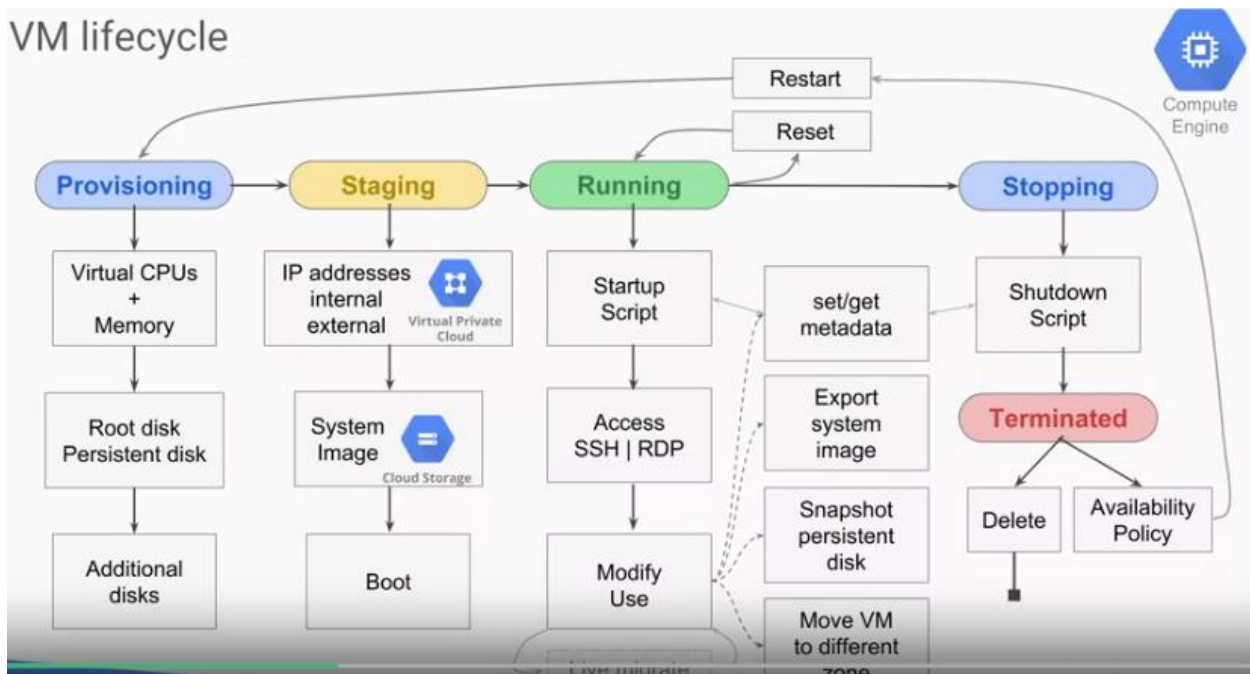
RDP Clients: Chrome Ext, 3rd party apps, MS Win RDP Client

Linux FreeRDP, remmina

Powershell terminal

Vm Life Cycle :

VM lifecycle



A Quick Recap :

- 1) Google Cloud Platform Resource Hierarchy**
- 2) IAM (Identity Access Management)**
- 3) Google cloud Storage**
- 4) Google cloud shell SDK (gsutil)**

Day 2 -Deep Dive

1) Compute Offerings from GCp

a) Compute Engine - IAAS

Create 3 virtual machines and get them added to Unmanaged groups and access and infer that Auto-scaling and Auto-healing is not possible

Instructions:

```
#!/bin/bash
sudo su -
apt-get update
apt-get install apache2 -y
service apache2 start
service apache2 status
cat >/var/www/html/index.html <<EOF
<html>
<body bgcolor="ffeedd">
<h1> Emea instance group </h1>
</body> </html>
EOF
```

Load Balancing:

Global	HTTP(S) load balancing	Distributes HTTP(S) traffic among groups of instances based on: <ul style="list-style-type: none">• Proximity to the user• Requested URL• Both	External
	SSL Proxy load balancing	Distributes SSL traffic among groups of instances based on proximity to the user.	
	TCP Proxy load balancing	Distributes TCP traffic among groups of instances based on proximity to the user.	
Regional	Network load balancing	<ul style="list-style-type: none">• Distributes traffic among a pool of instances within a region.• Can balance any kind of TCP/UDP traffic.	Internal
	Internal load balancing	Distributes traffic from GCP virtual machine instances to a group of instances in the same region.	

HTTPS Load Balancing:

HTTP(S) load balancing

- HTTP(S) load balancing:
 - Provides global load balancing for HTTP(S) requests destined for your instances.
 - Supports both IPv4 and IPv6 addresses for client traffic.
 - Uses instance groups to organize instances.
- HTTP requests can be load balanced based on ports:
 - 80
 - 8080
- HTTPS requests can be load balanced on port:
 - 443

HTTPS Load Balancing

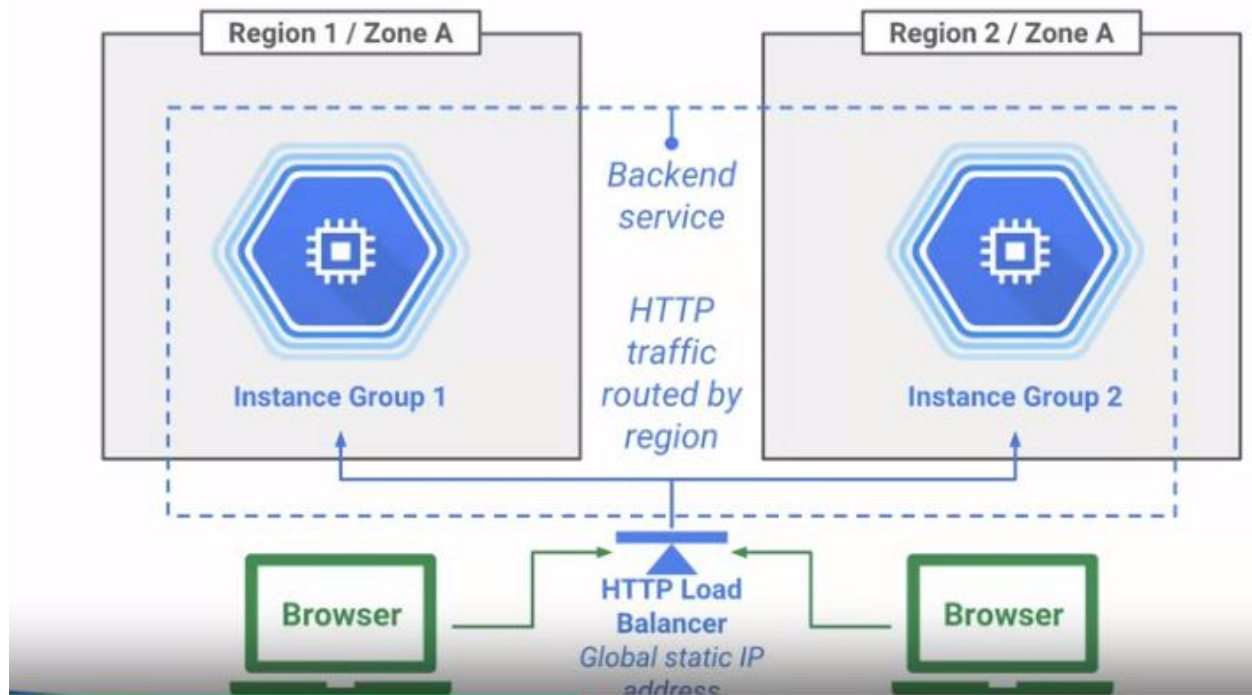
- HTTPS uses SSL (Secure Sockets Layer) to establish encryption for a secure link
- In HTTP(S) Load Balancing the Target Proxy terminates the client-originated SSL session, therefore, HTTP(S) requires the additional configuration of the SSL certificate
 - Provides a single place to maintain and update client certificates
- You can then load balance between the Target Proxy and the VMs using either TCP or a separate internally-originated SSL session

Cross Region Load Balancing:

Cross-Region Load Balancing

- HTTP/HTTPS only
- Cross-region using a single global IP address
- Requests routed to the closest region
- Automatically reroutes to next closest once capacity is reached
- Eliminates need for DNS-based load balancing

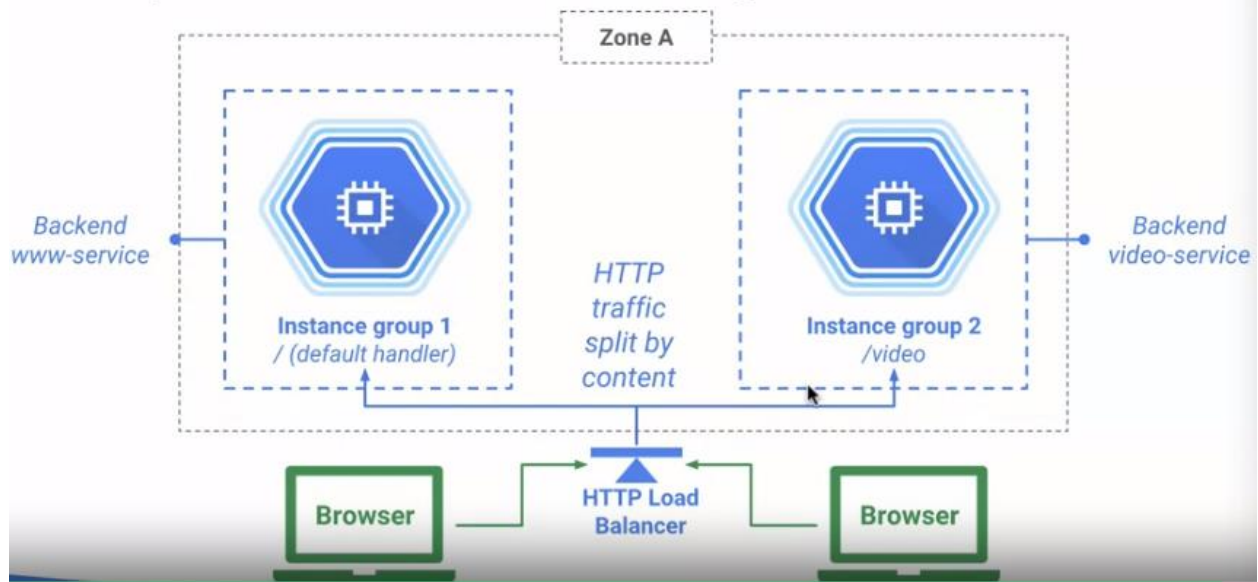
Example: Cross-Region Load Balancing



Content-Based Load Balancing

- HTTP/HTTPS only
- Create multiple backend services to handle content types
- Add path rules to backend services
 - /video for video services
 - /static for static content
- Configure different instance types for different content types

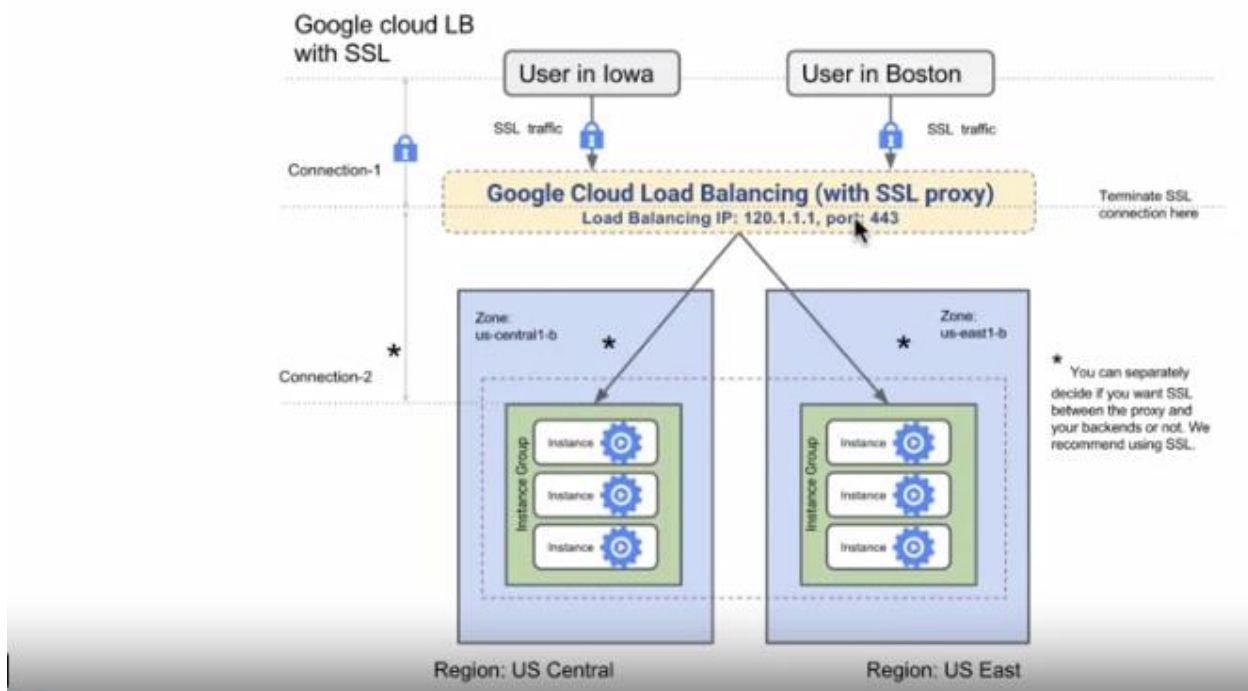
Example: Content-Based Load Balancing



SSL proxy

- Another kind of load balancing
 - Intended for non-HTTP(S) traffic using SSL
 - For HTTP(S) traffic, use HTTP(S) Load Balancing
 - Supports both IPv4 and IPv6 addresses for client traffic
- Benefits:
 - Performs global load balancing of SSL traffic, routing clients to the closest instance with capacity, similar to what HTTP(S) Load Balancing does for HTTP(S) traffic
 - Client IPv6 requests are terminated at the global load balancing layer, then proxied over IPv4 to your backends

Cloud Load Balancing with SSL proxy



\$ history

- 1 gcloud compute http-health-checks create webserver-health
- 2 gcloud config set project qwiklabs-gcp-61327791e4ad9ff0
- 3 gcloud compute http-health-checks create webserver-health
- 4 gcloud compute regions list
- 5 gcloud compute zones list
- 6 export MY_REGION=us-central1
- 7 export MY_ZONE1=us-central1-a
- 8 export MY_ZONE2=us-central1-f
- 9 gcloud compute target-pools create extloadbalancer --region \$MY_REGION --http-health-check webserver-health
- 10 gcloud compute target-pools add-instances extloadbalancer --instances webserver1,webserver2,webserver3 --instances-zone=\$MY_ZONE1
- 11 gcloud compute addresses list

```
12 export STATIC_EXTERNAL_IP=34.70.76.222
```

```
13 gcloud compute forwarding-rules create webserver-rule --region $MY_REGION --ports 80 --  
address $STATIC_EXTERNAL_IP --target-pool extloadbalancer
```

```
14 while true; do curl -m1 $STATIC_EXTERNAL_IP; done
```

```
15 history
```

```
gcloud compute instances create webserver4 \ --image-family debian-9 \ --  
image-project debian-cloud \ --tags int-lb \ --zone $MY_ZONE2 \ --subnet  
default \ --metadata startup-script-url="gs://cloud-  
training/archinfra/mystartupscript",my-server-id="WebServer-4"
```

```
export ILB_IP=<Enter the IP address of the internal load balancer>
```

```
for ((i=1;i<=10;i++)); do curl $ILB_IP; done
```

to simulate artificial workload .

Demos and Exercises:

1. Create unmanaged instance group and load balancer

2. Create managed instance group and load balancer

Keep autoscaling off.

Observe that autohealing is still on, if the health check is added.

Observe that the instance is replaced when the webserver is down, if health check is on.

3. Create a managed instance group with autoscaling on (and no load balancer health check) and with load balancer.

Stop the instance and observe that replacement instance is created.

4. Create a managed instance group with autoscaling on (with Load balancer healthcheck) and load balancer

Stop the webserver and observe that a replacement instance is created

5. Send overwhelming traffic to the instance and observe the scaling activity.

Visit monitoring tab and check the load on the instance groups.

Bench Marking commands

apt-get install apache2-utils

ab -n 100000 -c 1000 http://ip of the loadbalancer>/

6. Create a cloud function that gets triggered on upload of objects in a storage bucket and generate logs.

Step1: This function generates logs as and when data is overwritten or created in corresponding GCS bucket.

```
def hello_gcs_generic(data, context):
    """Background Cloud Function to be triggered by Cloud Storage.
    This generic function logs relevant data when a file is
    changed.

    Args:
        data (dict): The Cloud Functions event payload.
        context (google.cloud.functions.Context): Metadata of
    triggering event.
    Returns:
        None; the output is written to Stackdriver Logging
    """

    print('Event ID: {}'.format(context.event_id))
    print('Event type: {}'.format(context.event_type))
    print('Bucket: {}'.format(data['bucket']))
    print('File: {}'.format(data['name']))
    print('Metageneration: {}'.format(data['metageneration']))
    print('Created: {}'.format(data['timeCreated']))
    print('Updated: {}'.format(data['updated']))
```

Step 2: Switch to gcloud --> cd /home/krajeshkandikonda/python-docs-samples/functions/http

create/overwrite main.py with the above code.

Step 3: Deploy the function.

```
gcloud functions deploy hello_gcs_generic --runtime python37 --
trigger-resource rajeshbuck-1232 --trigger-event
google.storage.object.finalize
```

step 4: upload a newer version of the object in rajeshbuck-1232.

step 5: gcloud functions logs read hello_gcs_generic

Repeat step 4 and 5. You will understand that the function is getting invoked in the background as and when the objects inside the specified bucket are updated or new objects are created.

<https://cloud.google.com/functions/docs/calling/storage>