VM with 2 nics:

gcloud compute instances create vm1 --machine-type=n1-standard-4 \

--network-interface '' \

--network-interface network=net1,subnet=subnet-a,private-network-ip=10.10.10.2,address=34.174.57.131 \

--network-interface network=net2,subnet=subnet-b,private-network-ip=10.10.20.2,no-address

gcloud compute instances create vm1 \

--zone europe-west1-a \

--network-interface \

network=test-vpc-peer-vm1,subnet=test-vpc-peer-vm1-subnet, \

stack-type=IPV4\_ONLY, \

address=34.174.57.131 | no-address, \

private-network-ip=INTERNAL\_IPV4\_ADDRESS \

internal-ipv6-address=INTERNAL\_IPV6\_ADDRESS \

...

--network-interface \

network=NETWORK,subnet=SUBNET, \

stack-type=STACK\_TYPE, \

external-ipv6-address=EXTERNAL\_IPV6\_ADDRESS, \

external-ipv6-prefix-length=96 \

ipv6-network-tier=PREMIUM \

Query HC:

You can utilize these queries in two ways: either by employing them directly in order to display all the logs, or by incorporating the SEARCH("") function, which enables you to utilize keywords for faster retrieval of specific logs.

1/ HTTP response error: Unknown [Code: 421]

Query - 421:

resource.type="gce\_instance\_group"

resource.labels.instance\_group\_name=""

log\_name="projects/………./logs/compute.googleapis.com%2Fhealthchecks"

resource.labels.project\_id=".........."

SEARCH("421")

2/ HTTP response: , Error: Timeout waiting for connect

Query - timeout waiting:

resource.type="gce\_instance\_group"

resource.labels.instance\_group\_name=".............."

log\_name="projects/…………../logs/compute.googleapis.com%2Fhealthchecks"

resource.labels.project\_id=".............."

SEARCH("timeout waiting")

3/ HTTP response: OK [Code: 200]

Query - 200:

resource.type="gce\_instance\_group"

resource.labels.instance\_group\_name="................"

log\_name="projects/…………../logs/compute.googleapis.com%2Fhealthchecks"

resource.labels.project\_id="............."

SEARCH("200")

Kindly note that to view the logs, you will need to extend the period beyond 7 days, ideally in this case to 14 or 30 days. You could further customize these queries by changing the resource.type. For example, instead of specifying the resource.type="gce\_instance\_group", you could opt for resource.type="gce\_health\_check".

[1]: Create health checks - Required firewall rules

<https://cloud.google.com/load-balancing/docs/health-checks#firewall_rules>

==============

**A. Creating backend VMs and instance groups**

1. Create the VM in the ¨default¨ VPC

gcloud compute instances create vm-a2 \

--zone=us-west1-a \

--image-family=debian-10 \

--image-project=debian-cloud \

--tags=network-lb \

--network=default \

2. Create the unmanaged instance group:

gcloud compute instance-groups unmanaged create ig-a \

--zone us-west1-a

3. Add the VMs to the instance groups:

gcloud compute instance-groups unmanaged add-instances ig-a \

--zone us-west1-a \

--instances vm-a1

4. Create a firewall rule that allows external traffic to reach the backend instance.

gcloud compute firewall-rules create allow-network-lb \

--network=default \

--action=ALLOW \

--direction=INGRESS \

--source-ranges=35.191.0.0/16,209.85.152.0/22,209.85.204.0/22 \

--target-tags network-lb \

--rules=tcp

**B. Configuring load balancer components**

1. Create a static external IP address for your load balancer.

gcloud compute addresses create network-lb-ip \

--region us-west1

2. Create a new HTTP health check to test TCP connectivity to the VMs on 80 (I have tested with both HTTP and TCP and it has worked on both).

gcloud compute health-checks create http hc-http-80 \

--region us-west1 \

--port 80

3. Create the backend service for HTTP traffic:

gcloud compute backend-services create network-lb-backend-service \

--protocol tcp \

--region us-west1 \

--health-checks hc-http-80 \

--health-checks-region us-west1 \

4. Add the primary backend to the backend service:

gcloud compute backend-services add-backend network-lb-backend-service \

--region us-west1 \

--instance-group ig-a \

--instance-group-zone us-west1-a

5. Created a forwarding rule for the backend service. Use the IP address reserved in step 1 as the static external IP address of the load balancer.

gcloud compute forwarding-rules create network-lb-forwarding-rule \

--region us-west1 \

--load-balancing-scheme external \

--address network-lb-ip\

--ports 80 \

--backend-service network-lb-backend-service

Despite completing these steps, the health check remained unsuccessful, thus, I went ahead and verified the issue from the backend perspective, specifically if the VM was listening on port 80, by using the following command:

sudo netstat -tulpn | grep :80

Upon investigation, I found that the VM was not listening on port 80 by default, therefore I installed Nginx and started it and the outcome was that the VM now was listening on the 80 port:

sudo apt update

sudo apt install nginx

sudo systemctl status nginx

sudo netstat -tulpn | grep :80

#Output:

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 665/nginx: master p

tcp6 0 0 :::80 :::\* LISTEN 665/nginx: master p

Upon rechecking the load balancer, it confirmed that the backend was marked as healthy, indicating that everything was functioning correctly. As an additional test, you can use the curl command to make a web request to the load balancer by specifying its IP address and port (curl <ip:port>).

===========

> Created Dual stack VPC [1]

gcloud compute networks create test-unmig \

--subnet-mode=custom \

--enable-ula-internal-ipv6 \

--bgp-routing-mode=regional \

--mtu=1460

> Added subnet

gcloud compute networks subnets create test-unmig-subnet \

--network=test-unmig \

--range=10.0.0.0/24 \

--stack-type=IPV4\_IPV6 \

--ipv6-access-type=INTERNAL \

--region=europe-west1

> Created VM instance from GUI with dual stack network - using the same VPC and subnet as used in before

> Create instance group [3]

gcloud compute instance-groups unmanaged create test-unmig-group \

--zone=europe-west1-b

> Adding instance to UNMIG

gcloud compute instance-groups unmanaged add-instances test-unmig-group \

--zone=europe-west1-b \

--instances=instance-unmig

Upon completion, the outcome remained consistent, as the Unmanaged Instance Group panel continued to display only the IPv4 IP address for the internal IP of the VM contained within it.

Regarding your second inquiry, as per documentation [5], it would be feasible to set up an internal passthrough Network Load Balancer with VM instance group backends using dual-stack subnets.

Moreover, addressing your third inquiry, based on documentation [6], it may be possible to handle both IPv4 and IPv6 traffic by creating separate forwarding rules for each. The forwarding rules can reference the same backend service, provided that the backend service points to dual-stack backends.

[1]: Create a custom mode VPC network with a dual-stack subnet

<https://cloud.google.com/vpc/docs/create-modify-vpc-networks#create-network-dual>

[2]: Add a dual-stack subnet

<https://cloud.google.com/vpc/docs/create-modify-vpc-networks#add-subnet-ipv6>

[3]: Create unmanaged group

<https://cloud.google.com/compute/docs/instance-groups/creating-groups-of-unmanaged-instances#create_unmanaged_group>

[4]: Add instances

<https://cloud.google.com/compute/docs/instance-groups/creating-groups-of-unmanaged-instances#addinstances>

[5]: Set up an internal passthrough Network Load Balancer with dual-stack subnets (IPv4 and IPv6) with VM instance group backends

<https://cloud.google.com/load-balancing/docs/internal/setting-up-internal#configure-test-environment-ipv6>

[6]: Internal passthrough Network Load Balancer overview - Architecture

<https://cloud.google.com/load-balancing/docs/internal#architecture>

=================

[1]: Set up Cloud NAT with GKE

<https://cloud.google.com/nat/docs/gke-example>

[2]: Yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: “my-ingress”

annotations:

kubernetes.io/ingress.global-static-ip-name: "troon-test-staticip"

kubernetes.io/ingress.class: "gce"

spec:

rules:

- http:

paths:

- backend:

service:

name: matrixtest-np

port:

number: 80

path: /\*

pathType: ImplementationSpecific