**10.1P: MONITORING AND VISIBILITY**

Instructions

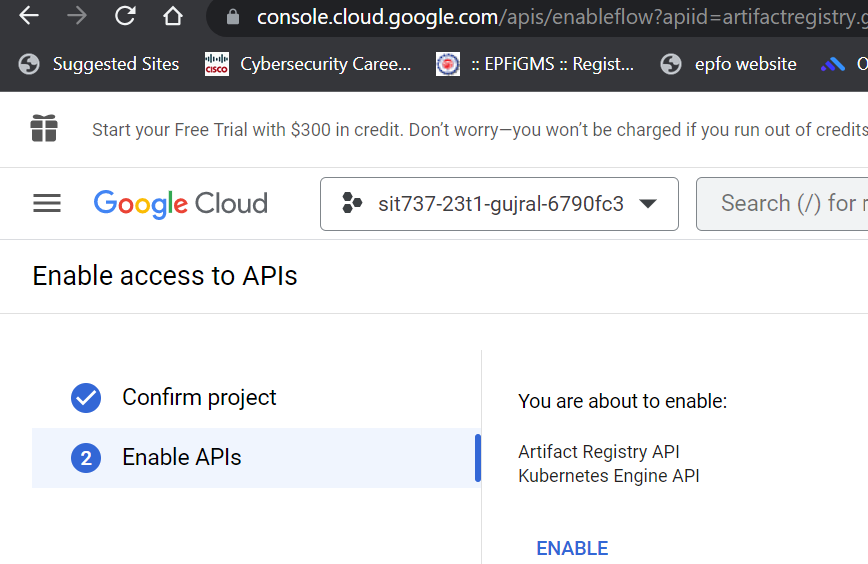
• Containerize a simple application using Node.js, Docker, and Kubernetes.

• Deploy the containerized application to a GCP Kubernetes cluster.

• Ensure proper monitoring and visibility of your cloud-native application.

Solution

1. Logged in to GCP and navigated under the project sit737-23t1-gujral-6790fc3.
2. Enabled Artifact Registry and Google Kubernetes Engine APIs.



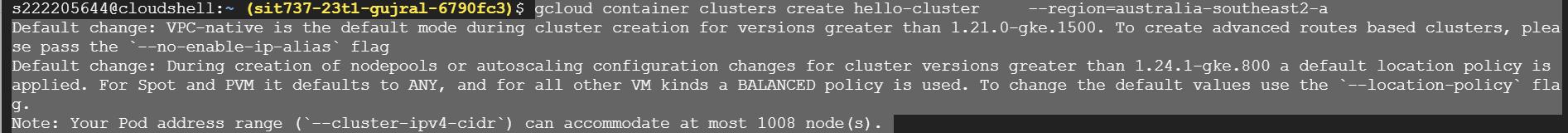
1. Created a GKE cluster

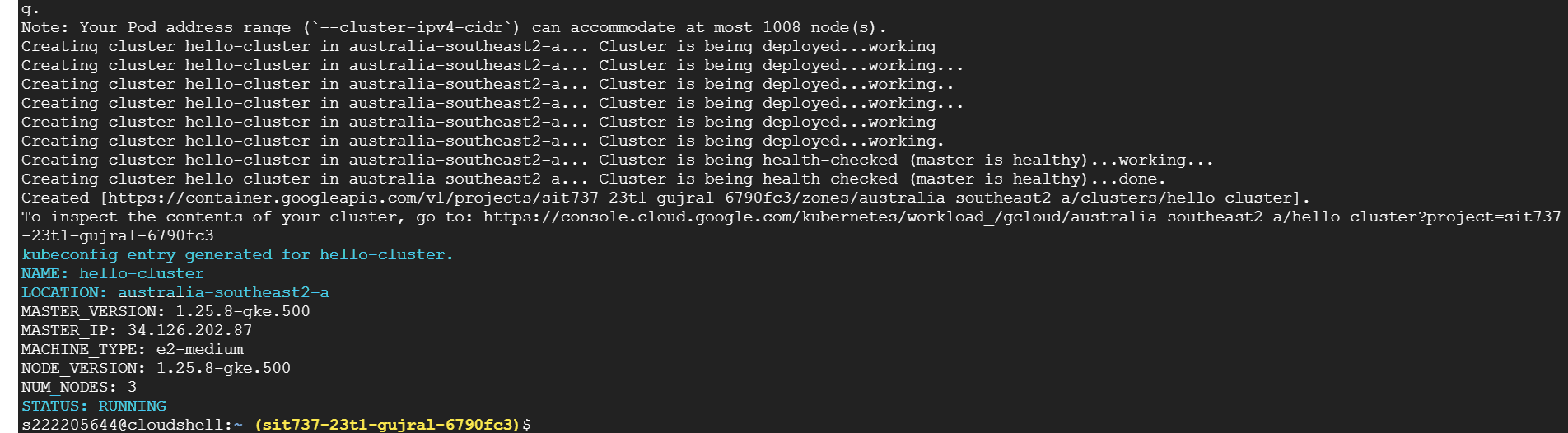
The cluster includes many worker nodes which are VM instances, 3 in this case as seen in this screenshot below. Applications are deployed on clusters, and run on nodes.

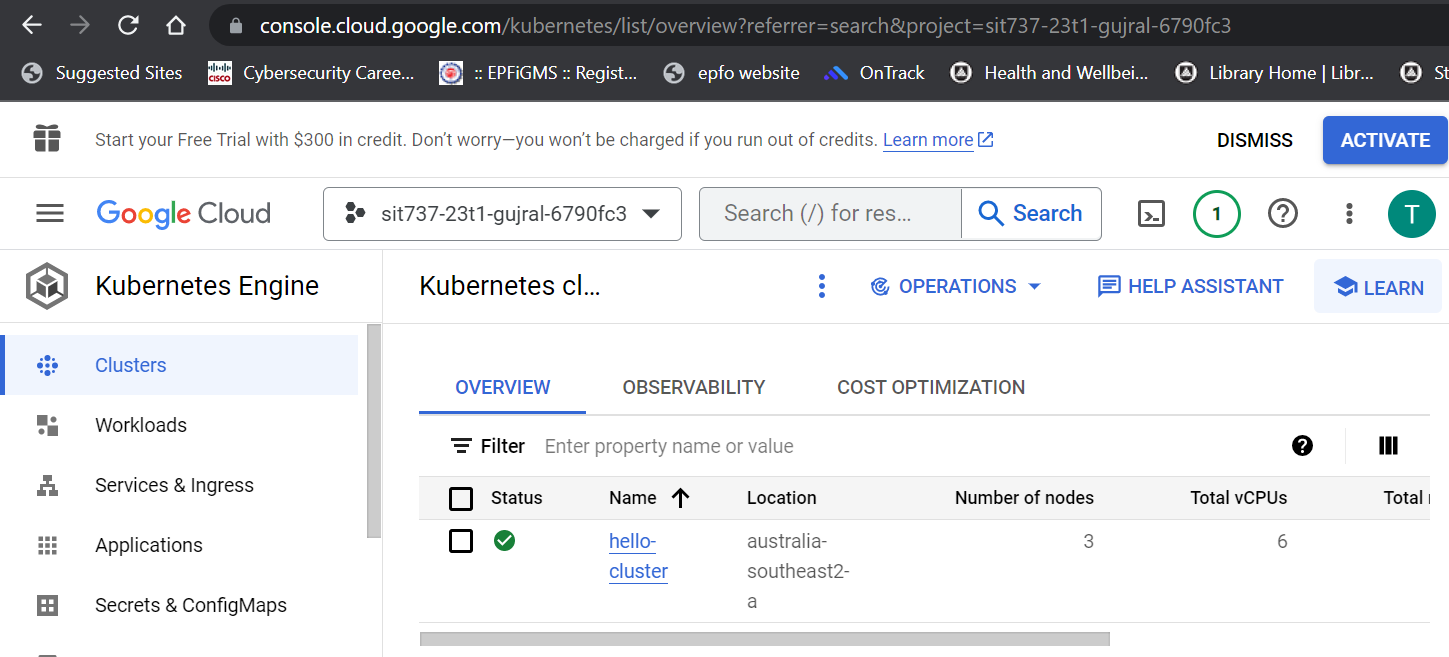
1. Created a standard cluster named hello-cluster, using command 🡪

gcloud container clusters create hello-cluster \  
    --region=*COMPUTE\_REGION*

here*,* compute\_region for Melbourne could be chosen as australia-southeast2-a, australia-southeast2-b, australia-southeast2-c. The one used below is australia-southeast2-a





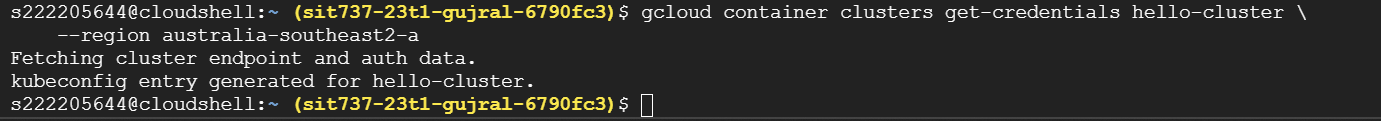


1. Got authentication credentials for the cluster

Authentication credentials allow for communication interact with the cluster.

Command 🡪

gcloud container clusters get-credentials hello-cluster \  
    --region australia-southeast2-a



1. Deployed application to the cluster
2. Node.js application used is a basic one that prints Welcome to microservices, similar to a Hello World application

Index.js -

const { json } = require('express');

const express = require('express');

const app = express();

//Constants

const PORT = 3000;

const HOST = '0.0.0.0';

// Home page request

app.get('/', (req, res) => {

  res.send("Welcome to the microservice");

});

app.listen(PORT, HOST, () => {

  console.log('Running on http://${HOST}:${PORT}');

});

Dockerfile –

#denotes base image

FROM node:14

#setting working directory

WORKDIR /usr/src/app

COPY package\*.json ./

#to install the package listed in package.json file

RUN npm install

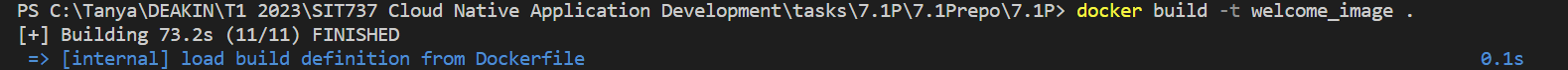
COPY index.js index.js

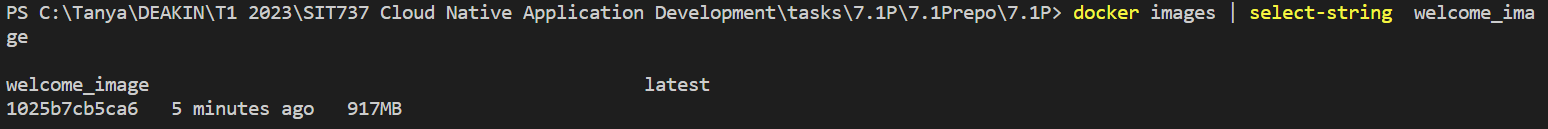
#exposing port outside

EXPOSE 3000

CMD ["node", "index.js"]

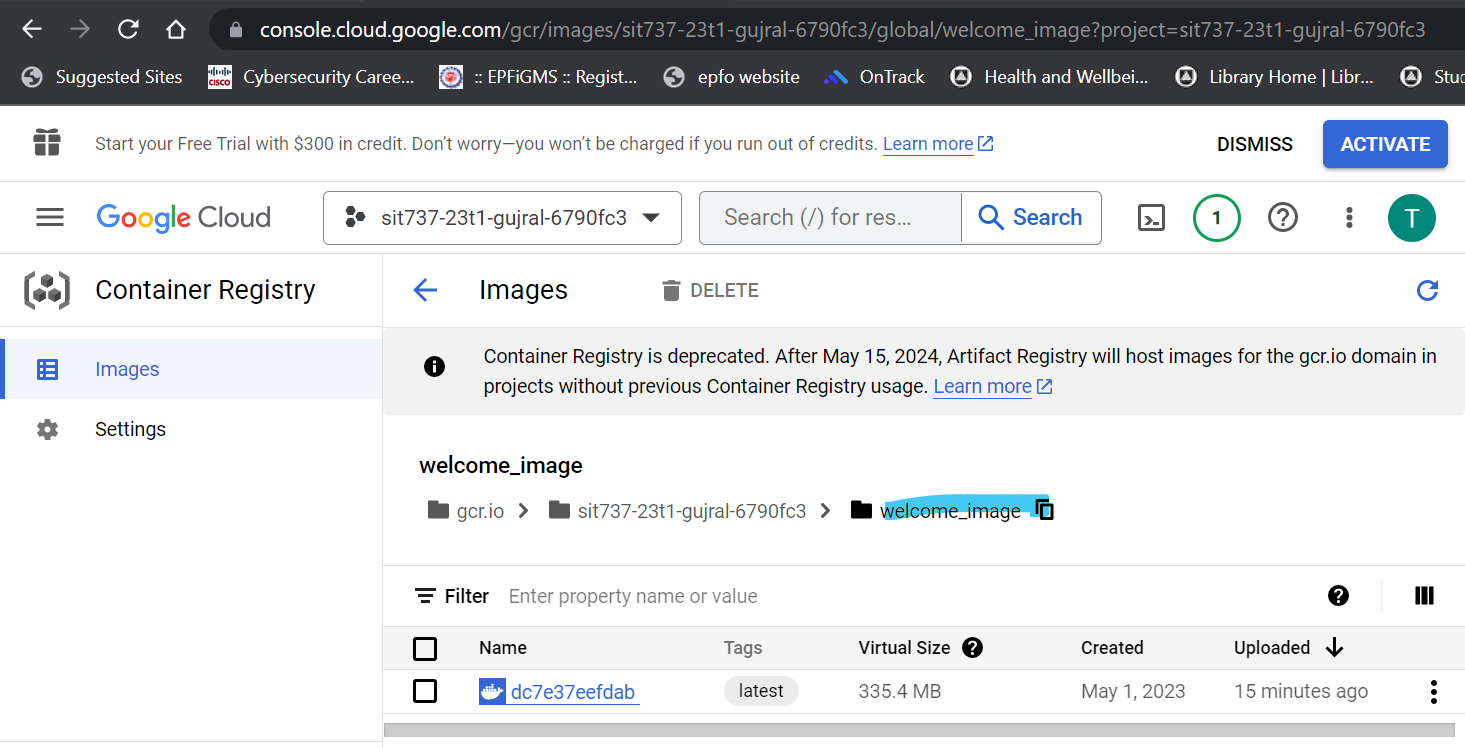
1. Built the image using Docker build command 🡪 docker build -t welcome\_image .





(iv) Pushed the docker image to GCR

Command used 🡪 docker push gcr.io/sit737-23t1-gujral-6790fc3/welcome\_image

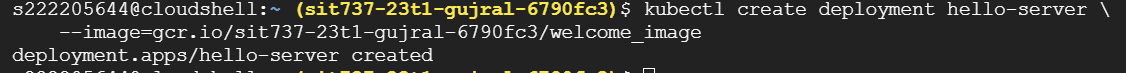


1. Created the Deployment

Used the command below to create a deployment named hello-server. The pod of the deployment runs the welcome\_image container image.

Command 🡪

kubectl create deployment hello-server \  
    --image=gcr.io/sit737-23t1-gujral-6790fc3/welcome\_image



1. Exposed the Deployment

This is to expose the deployed application outside the cluster, to the internet for any user to be able to access it. It uses a service to expose the hello-server to external traffic.

Command 🡪

kubectl expose deployment hello-server --type LoadBalancer --port 80 --target-port 3000

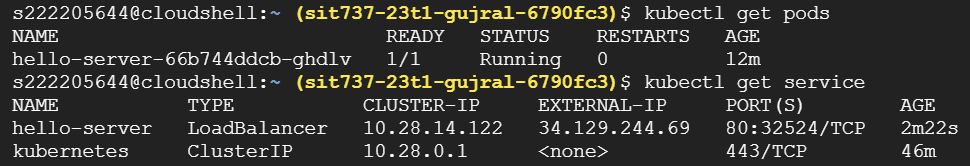
The --type LoadBalancer flag creates a Compute Engine load balancer for the container. The port 3000 of the application is connected to port 80 i.e., the internet here.



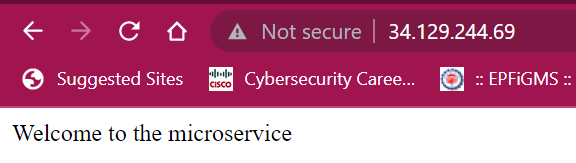
1. Inspecting and viewing the application

kubectl get pods

kubectl get service hello-server



1. Accessing the application from web browser by using the external IP address -



1. Monitoring

Metrics Scope in GCP provides a wide-range of dashboards showcasing the metrics related to the cloud assets and provides with a detailed real-time monitoring.

Observing logs and metrics help sin not only troubleshooting and maintenance but also provides analytics. The most common use of GCP’s monitoring dashboards is to monitor the availability, health of APIs, performance, scalability etc.

* **Custom dashboard** “10.1P Dashboard” link - <https://console.cloud.google.com/monitoring/dashboards/builder/7e539051-fe0d-45b0-87e7-b73894538a32;duration=PT1H>

Created a custom dashboard adding 6 of the metrics below –

1. Kubernetes container – CPU limit utilization
2. VM Instance – CPU utilization
3. Kubernetes Node – CPU Usage Time
4. Kubernetes Pod – Bytes Received
5. Kubernetes Container – Memory limit utilization
6. Log entries MEAN

The JSON file for the custom dashboard “10.1P Dashboard” is uploaded in the Github repository for the project

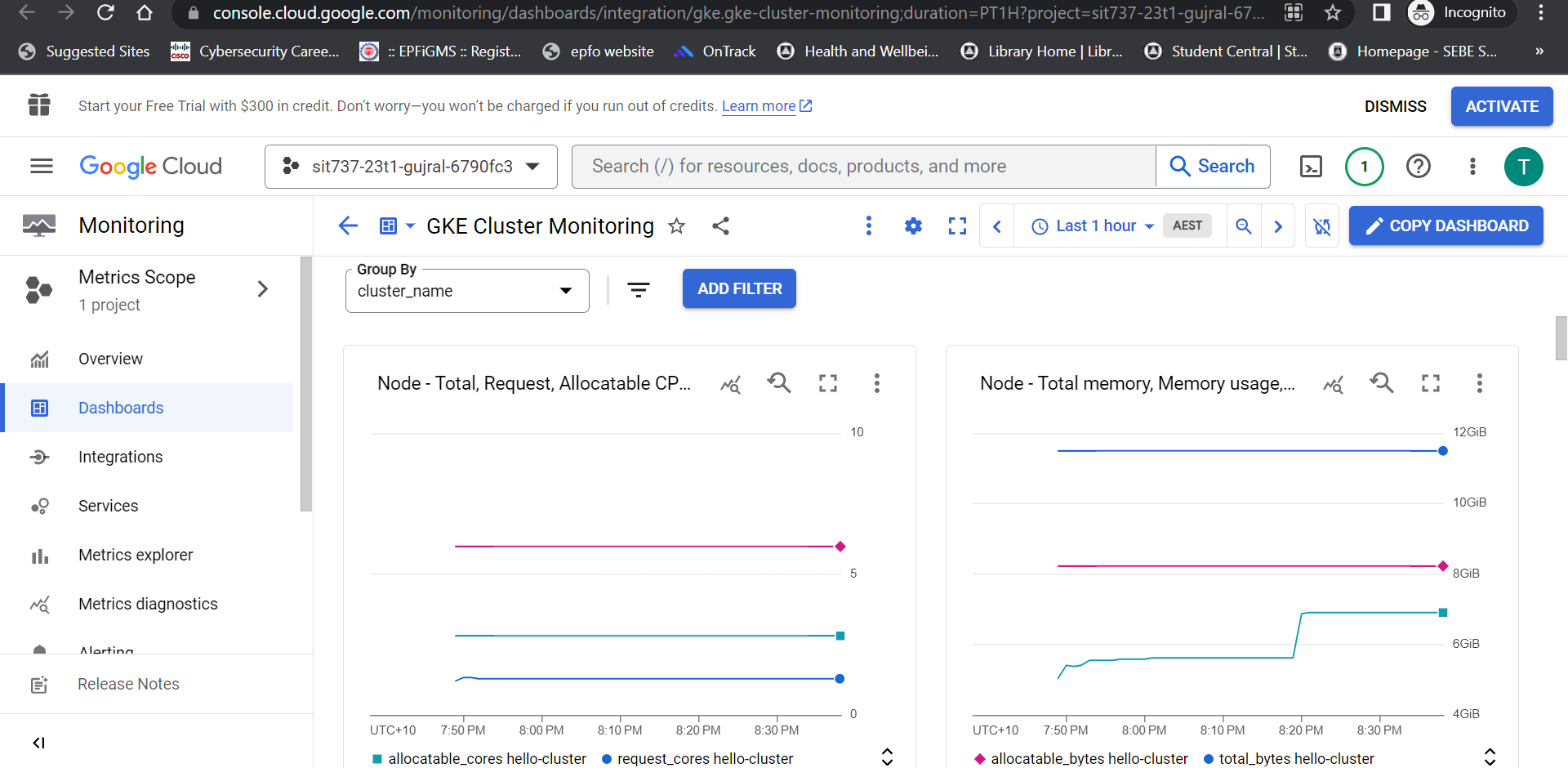
* There are a variety of **default dashboards** covering the metrics exhaustively -

The link to the default dashboards is - <https://console.cloud.google.com/monitoring/dashboards?project=sit737-23t1-gujral-6790fc3&pageState=(%22dashboards%22:(%22t%22:%22All%22))>

A few of the dashboards are captured and showcased below –

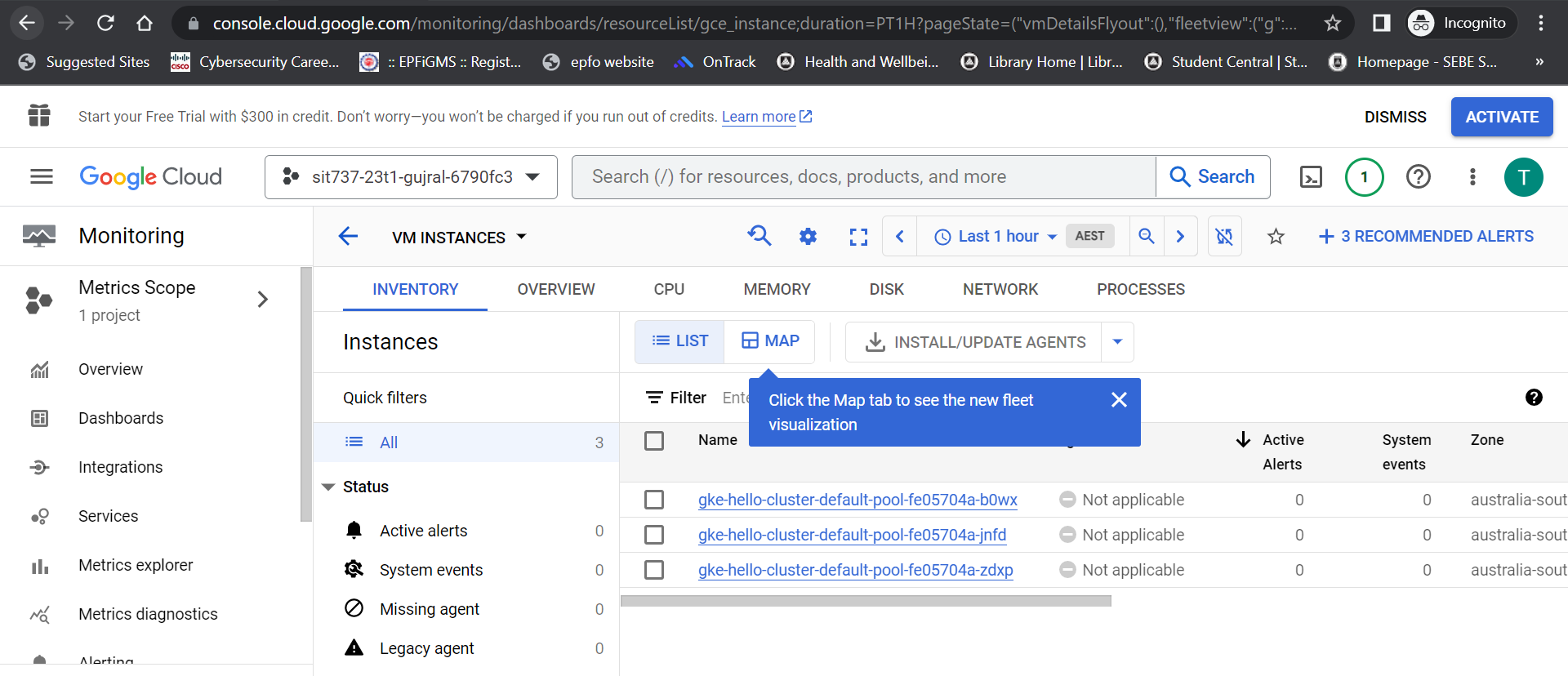
1. GKE cluster monitoring

<https://console.cloud.google.com/monitoring/dashboards/integration/gke.gke-cluster-monitoring?project=sit737-23t1-gujral-6790fc3>

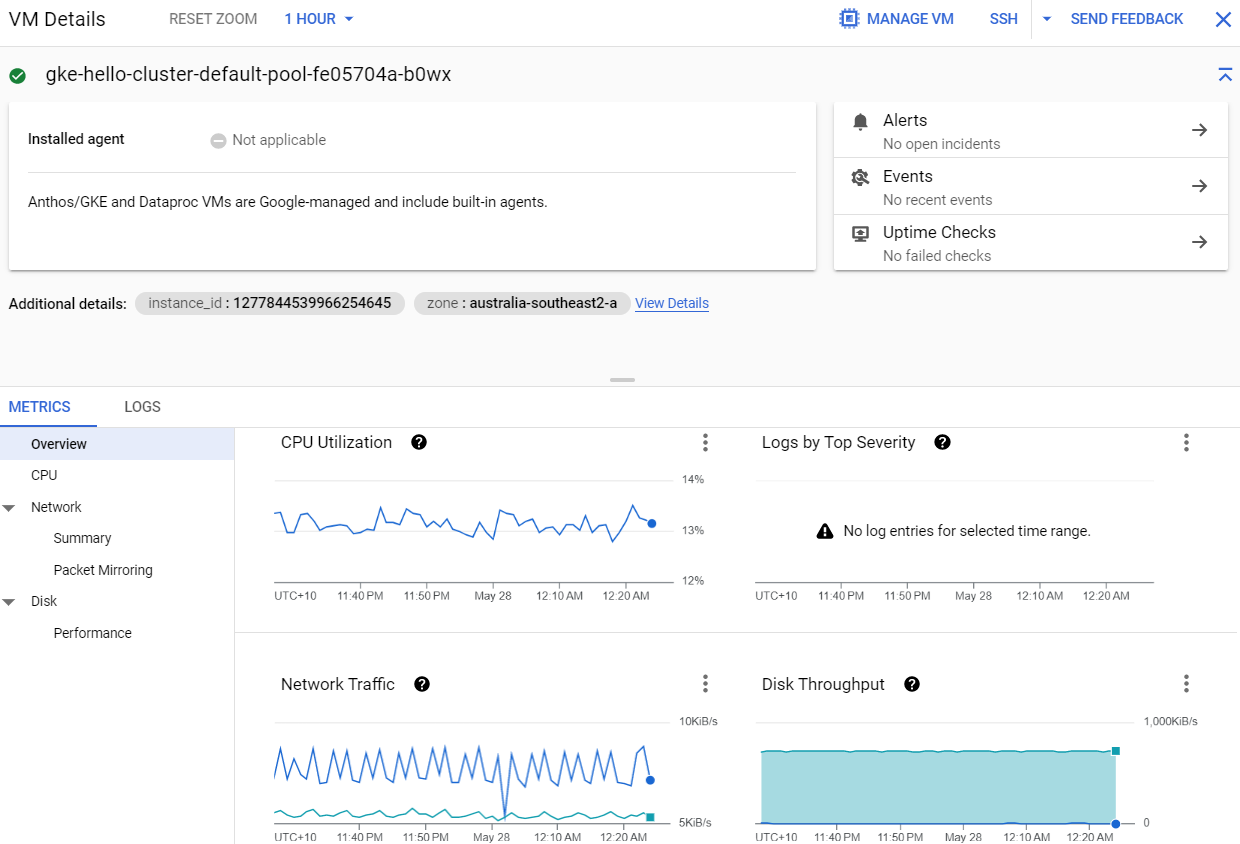


1. VM instances

<https://console.cloud.google.com/monitoring/dashboards/resourceList/gce_instance;duration=PT1H?pageState=(%22vmDetailsFlyout%22:(),%22fleetview%22:(%22g%22:%5B%22resource.labels.zone%22,%22metadata.systemLabels.name%22%5D,%22d%22:%5B%5D,%22s%22:%22COUNT%22,%22m%22:%22compute.googleapis.com%2Finstance%2Fcpu%2Futilization%22))&project=sit737-23t1-gujral-6790fc3>

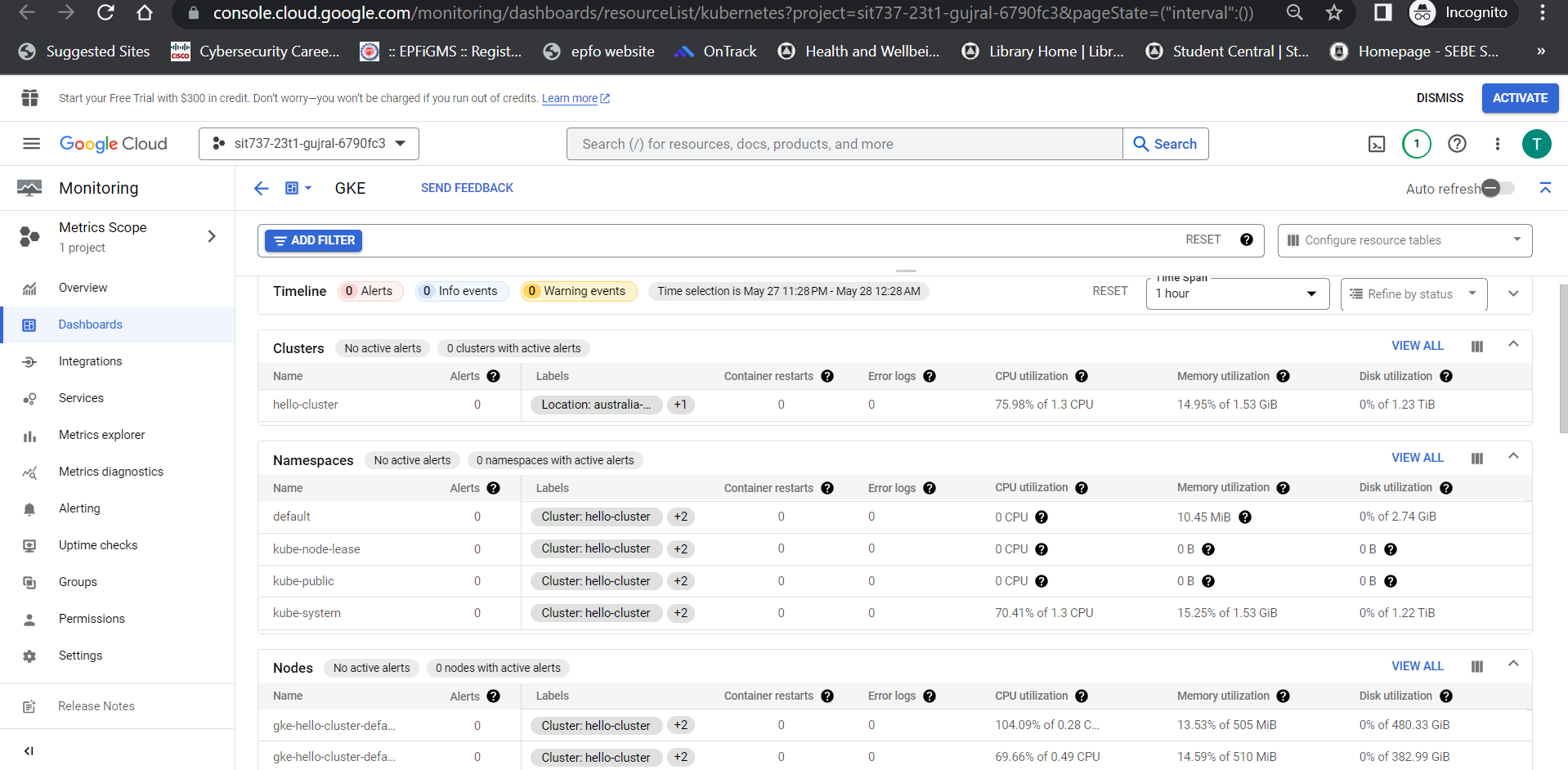


Further, for each VM metrics can be seen as –



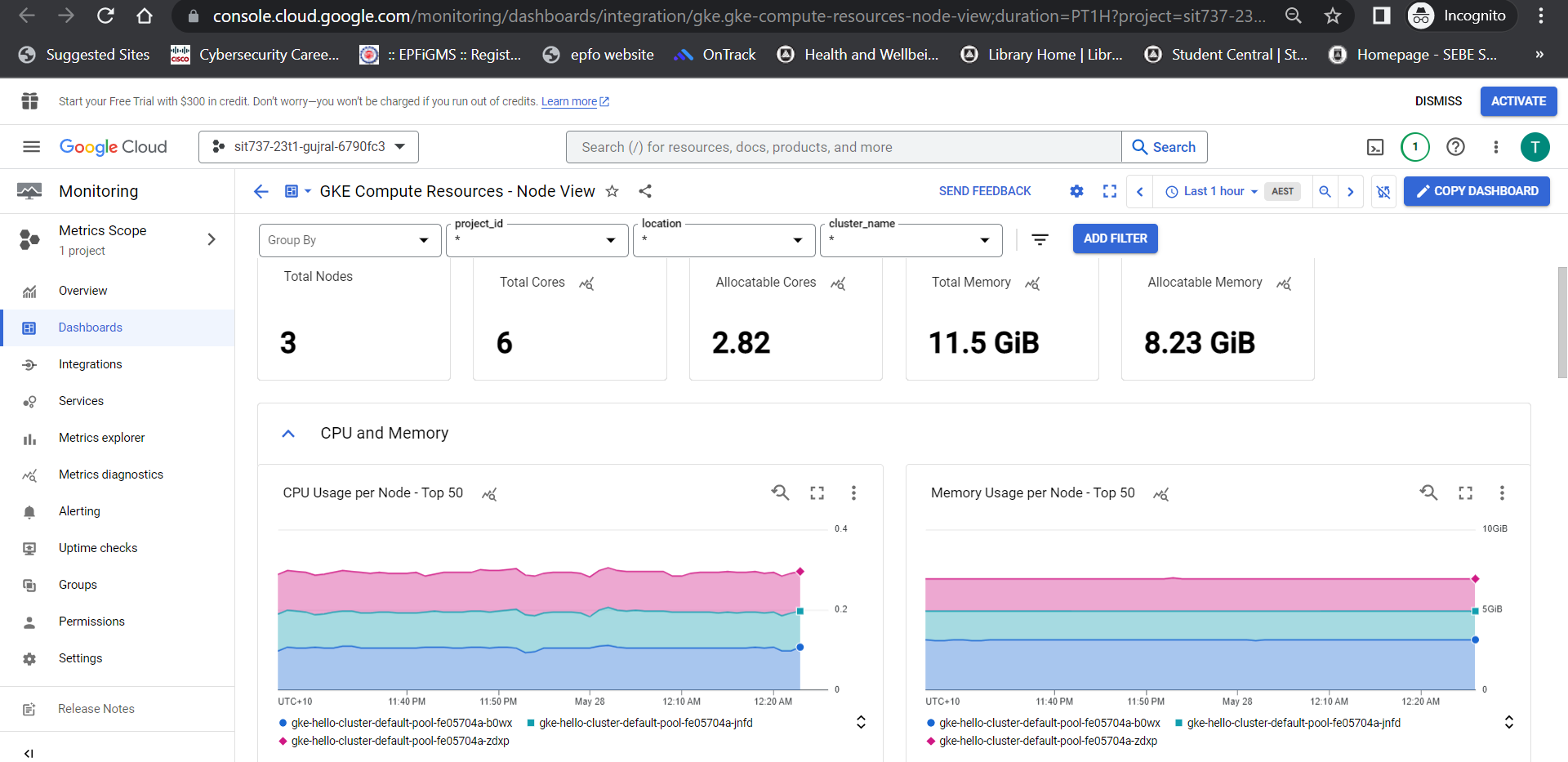
1. GKE metrics –

<https://console.cloud.google.com/monitoring/dashboards/resourceList/kubernetes?project=sit737-23t1-gujral-6790fc3&pageState=(%22interval%22:())>



1. GKE Compute Resources (Node View) –

<https://console.cloud.google.com/monitoring/dashboards/integration/gke.gke-compute-resources-node-view?project=sit737-23t1-gujral-6790fc3>



1. GKE Compute Resources (Cluster View) –

<https://console.cloud.google.com/monitoring/dashboards/integration/gke.gke-compute-resources-cluster-view;duration=PT1H?project=sit737-23t1-gujral-6790fc3>

