**Optimize Costs for Google Kubernetes Engine: Challenge Lab**

[[LTC](https://medium.com/@yakheng?source=post_page-----9176ee0c7821--------------------------------)](https://medium.com/@yakheng?source=post_page-----9176ee0c7821--------------------------------)

[LTC](https://medium.com/@yakheng?source=post_page-----9176ee0c7821--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F300379ee5967&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40yakheng%2Foptimize-costs-for-google-kubernetes-engine-challenge-lab-9176ee0c7821&user=LTC&userId=300379ee5967&source=post_page-300379ee5967----9176ee0c7821---------------------post_header-----------)

13 min read

·

Nov 29, 2023

This is the solution to the [Challenge Lab](https://www.cloudskillsboost.google/focuses/16327?catalog_rank=%7B%22rank%22%3A1%2C%22num_filters%22%3A0%2C%22has_search%22%3Atrue%7D&parent=catalog&search_id=28680336) for the [Optimize Costs for Google Kubernetes Engine](https://www.cloudskillsboost.google/course_templates/655) skill badge quest.

The solution below was applied on 29 December 2023.

**Challenge scenario**

You are the lead Google Kubernetes Engine admin on a team that manages the online shop for **OnlineBoutique**.

You are ready to deploy your team’s site to Google Kubernetes Engine but you are still looking for ways to make sure that you’re able to keep costs down and performance up.

You will be responsible for deploying the **OnlineBoutique** app to GKE and making some configuration changes that have been recommended for cost optimization.

Here are some guidelines you’ve been requested to follow when deploying:

Create the cluster in the zone. The naming scheme is team-resource-number, e.g. a cluster could be named Cluster Name. For your initial cluster, start with machine size e2-standard-2 (2 vCPU, 8G memory). Set your cluster to use the **rapid** release-channel.

**Task 1. Create our cluster and deploy our app**

1. Before you can deploy the application, you’ll need to create a cluster and name it as Cluster Name.
2. Start small and make a zonal cluster with only two (2) nodes.
3. Before you deploy the shop, make sure to set up some namespaces to separate resources on your cluster in accordance with the 2 environments — dev and prod.
4. After that, deploy the application to the dev namespace with the following command:

git clone https://github.com/GoogleCloudPlatform/microservices-demo.git &&  
cd microservices-demo && kubectl apply -f ./release/kubernetes-manifests.yaml --namespace dev

Note: You can check that your **OnlineBoutique** store is up and running by navigating to the IP address for your **frontend-external** service.

👇 👇 👇 Solution:

1. Activate Cloud Shell to execute the following gcloud command-line commands.
2. Set default compute zone.

gcloud config set compute/zone [Zone specified in "Challenge scenario" at start of lab]

To verify, gcloud config list compute/zone

3. Create GKE zonal cluster.

gcloud container clusters create [Cluster Name] --num-nodes=2 --machine-type=e2-standard-2 --release-channel=rapid --zone=[Zone]

To verify, in **Cloud Console > Navigation Menu > Kubernetes Engine > Clusters** window, check for [Cluster Name] with a green tick mark for its status. If you click on [Cluster Name] and select the **NODES** tab, there should be 2 nodes listed.

Reference: [gcloud container clusters create command](https://cloud.google.com/sdk/gcloud/reference/container/clusters/create)

4. Create dev and prod namespaces.

kubectl create namespace dev &&   
kubectl create namespace prod

To verify, kubectl get namespace. Output should show these 2 new namespaces, in addition to cluster's system namespaces.

5. Deploy application to dev namespace.

git clone https://github.com/GoogleCloudPlatform/microservices-demo.git &&  
cd microservices-demo && kubectl apply -f ./release/kubernetes-manifests.yaml --namespace dev

To verify, **Cloud Console > Navigation Menu > Kubernetes Engine > Workloads** to check for the deployed micro-services.  
To check that **OnlineBoutique** store is up and running, get the external IP address for **frontend-external** service using kubectl get services --namespace=dev.  
Alternatively, in **Cloud Console** for **Kubernetes Engine**, click on **Services & Ingress**. Frontend-external service has an **External load balancer endpoint**. Copy and paste the external IP address in a new browser tab to access the online boutique.

**Task 2. Migrate to an optimized node pool**

1. After successfully deploying the app to the dev namespace, take a look at the node details:

You come to the conclusion that you should make changes to the cluster’s node pool:

* There’s plenty of left over RAM from the current deployments so you should be able to use a node pool with machines that offer **less RAM**.
* Most of the deployments that you might consider increasing the replica count of will require only 100mcpu per additional pod. You could potentially use a node pool with **less total CPU** if you configure it to use smaller machines. However, you also need to consider how many deployments will need to scale, and how much they need to scale by.

1. Create a new node pool named Pool Name with **custom-2-3584** as the machine type.
2. Set the **number of nodes** to **2**.
3. Once the new node pool is set up, migrate your application’s deployments to the new nodepool by **cordoning off and draining** default-pool.
4. Delete the default-pool once the deployments have safely migrated.

**👇 👇 👇 Solution:**

1. Set kubectl context to be in the dev namespace so that there is no need to use the --namespace=dev flag for every command.

kubectl config set-context --current --namespace=dev

Without this step, commands will be executed in the default namespace.

2. Create new Pool Name node pool.

gcloud container node-pools create `Pool Name` \  
--cluster=[Cluster Name] \  
--machine-type=custom-2-3584 \  
--num-nodes=2 \  
--zone=[Zone]

To verify, click on the [Cluster Name] in the Cloud Console for Kubernetes, then refresh the **NODES** tab. Check for the newly created node pool, in addition to the existing default-pool.

3. Cordon off the existing default-pool node pool.

for node in $(kubectl get nodes -l cloud.google.com/gke-nodepool=default-pool -o=name); do  
 kubectl cordon "$node";  
done

*Expected output:*

*node/gke-onlineboutique-clust-default-pool-dbxxxxxx-gwxt cordoned  
node/gke-onlineboutique-clust-default-pool-dbxxxxxx-pc9w cordoned*

If you want to verify, kubectl get nodes will show default-pool's nodes having "*Ready,SchedulingDisabled*" status. Alternatively, click on the [Cluster Name] in the Cloud Console for Kubernetes, then refresh the **NODES** tab. Status for default-pool's nodes will be reflected as "*Cordoned*".

4. Drain default-pool.

for node in $(kubectl get nodes -l cloud.google.com/gke-nodepool=default-pool -o=name); do  
 kubectl drain --force --ignore-daemonsets --delete-local-data --grace-period=10 "$node";  
done

Code explanation:

* these are Linux **for** commands
* syntax: **for** NAME in [WORDS …]; do COMMANDS; done
* the NAME is set to each element present in the [WORDS …] list. for each element in WORDS, the set of COMMANDS will be executed.
* ; semi-colon is a command seperator, allows putting 2 or more commands on the same line.
* the $(…) expression is known as command substitution and it invokes a subshell.
* the command within the braces of $( ) is executed in the subshell environment, and the stdout output is returned to the parent command in the $( ) position.
* [“kubectl get” command](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#get)
* -l: label to select/query/filter with. Requires key1=value1,key2=value2
* -o=name flag specifies that the output is in the format of names
* [kubectl cordon](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#cordon)
* [kubectl drain](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#drain)
* In “$node”, the $node is syntax for variable substitution. You substitute the node variable with its value.
* The double quotes “” means to treat the content within the quotes as a single word.
* [special characters in Bash scripts](https://tldp.org/LDP/abs/html/special-chars.html)

5. Check whether deployments safely migrated to new Pool Name node pool.

kubectl get pods -o=wide

Check if the pods are listed in the new `gke-onlineboutique-c-optimized-pool-x-xxxxxxxx-xxxx`` node.

6. Delete old default-pool node pool.

gcloud container node-pools delete default-pool --cluster=[Cluster Name] --zone=[Zone]

When asked to continue, type y and enter.  
To verify, click on the [Cluster Name] in the Cloud Console for Kubernetes, then refresh the **NODES** tab. Check for default-pool removal.

**Task 3. Apply a frontend update**

You just got it all deployed, and now the dev team wants you to push a last-minute update before the upcoming release! That’s ok. You know this can be done without the need to cause down time.

1. Set a pod disruption budget for your **frontend** deployment.
2. Name it **onlineboutique-frontend-pdb**.
3. Set the **min-availability** of your deployment to **1**.

Now, you can apply your team’s update. They’ve changed the file used for the home page’s banner and provided you an updated docker image:

gcr.io/qwiklabs-resources/onlineboutique-frontend:v2.1

4. **Edit** your **frontend** deployment and change its image to the updated one.

While editing your deployment, change the **ImagePullPolicy** to **Always**.

👇 👇 👇 Solution:

1. Create PodDisruptionBudget

kubectl create poddisruptionbudget onlineboutique-frontend-pdb --selector app=frontend --min-available 1

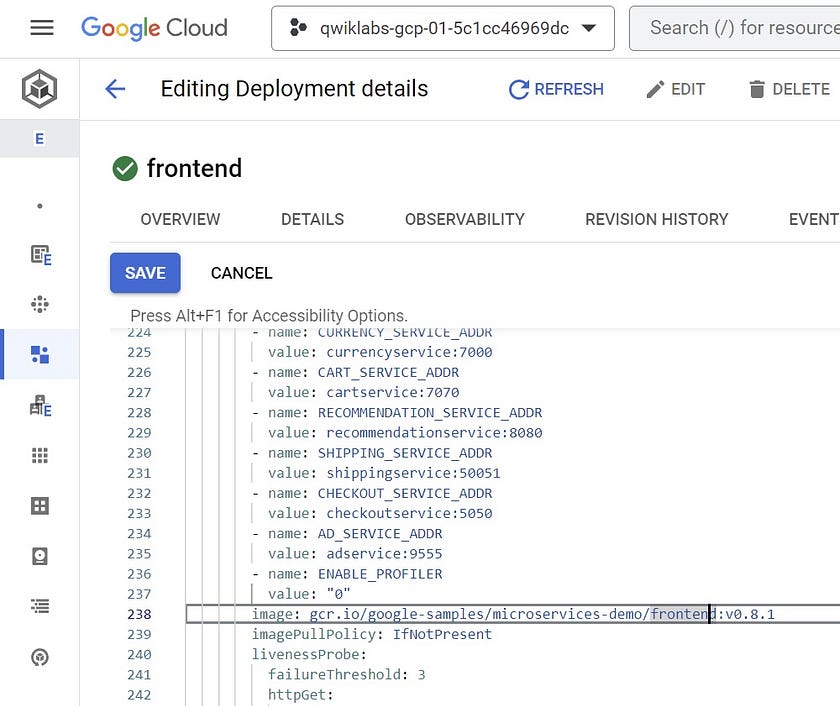
To target the **frontend** deployment, use the the selector for the frontend deployment. Get to know the selector from the [kubernetes-manifests.yaml file](https://github.com/GoogleCloudPlatform/microservices-demo/blob/main/release/kubernetes-manifests.yaml)  
To verify, kubectl get poddisruptionbudgets

2. Get the frontend deployment’s yaml manifest file.

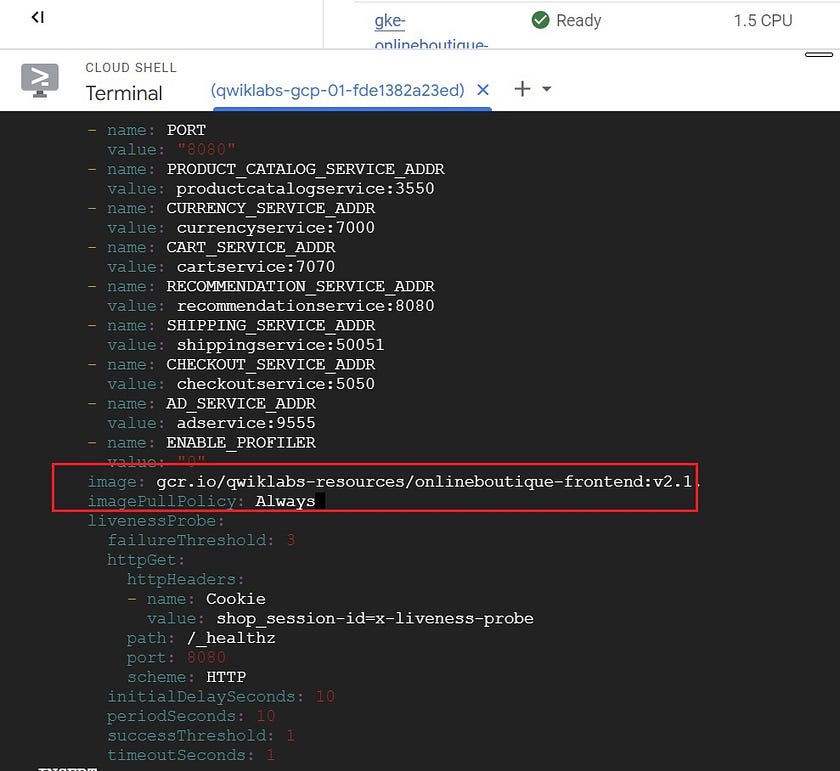
In the Cloud console, from Kubernetes Engine > Workloads > click on **frontend**. Select the **YAML** tab. Click on **EDIT** on the horizontal menu at the top to edit. Alternatively,

kubectl edit deployment frontend

3. In the editor, under .spec.containers, replace **image: gcr.io/google-samples/microservices-demo/frontend:v0.8.1** to **image: gcr.io/qwiklabs-resources/onlineboutique-frontend:v2.1**  
One line down, replace **imagePullPolicy: IfNotPresent** to **imagePullPolicy: Always**  
In the Cloud console, click **SAVE** after editing. For the CLI vi editor, to save changes made, press ESC + :wq.

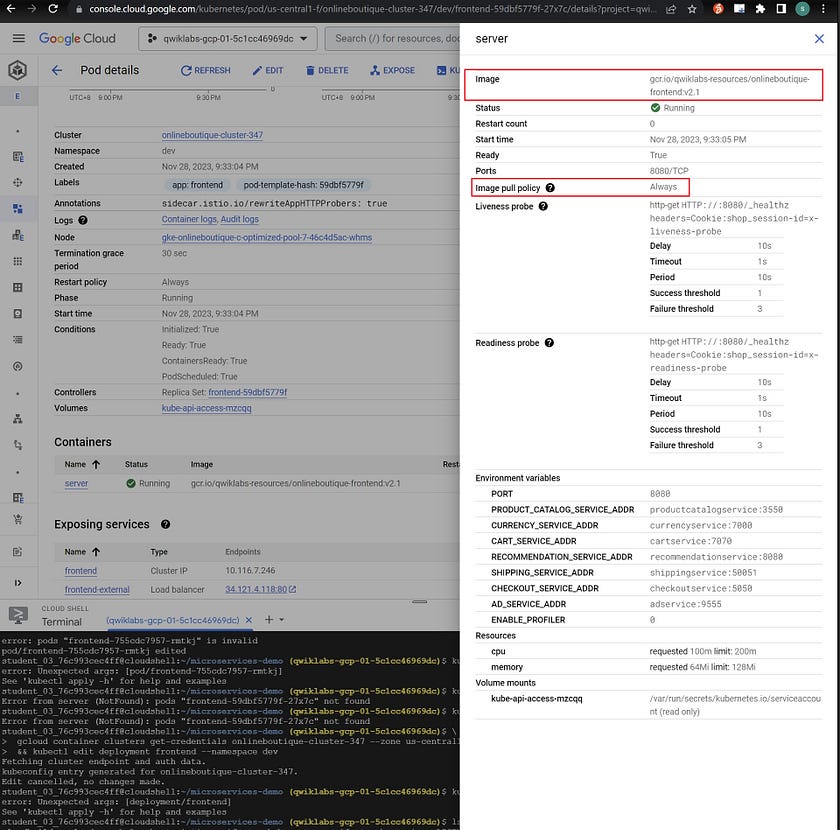


Editing frontend deployment YAML. YAML editor in Google Cloud Console.

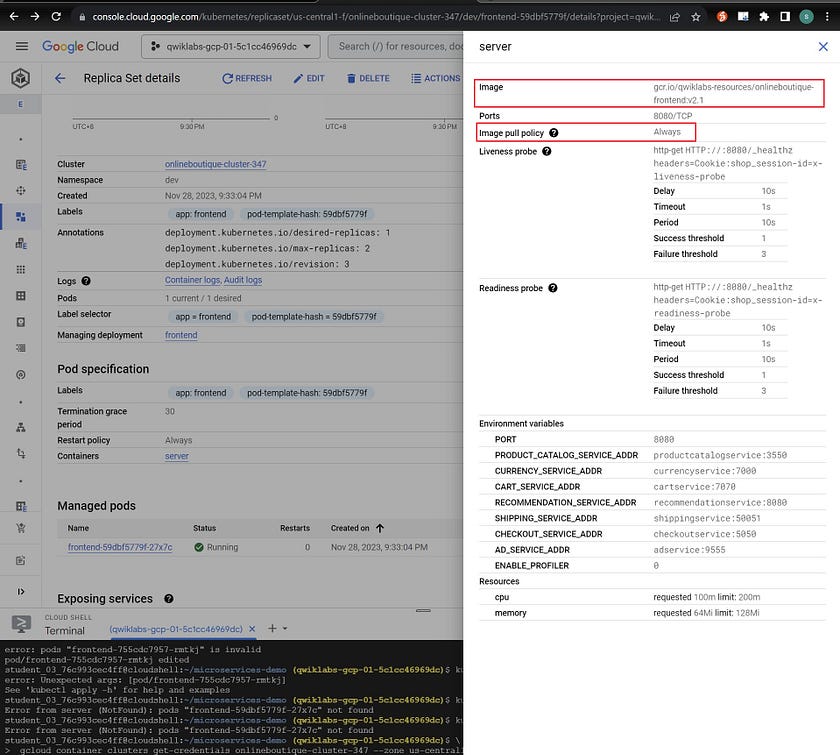


*“kubectl edit deployment frontend”. vi editor in Cloud Shell.*

To verify, in Cloud console > Kubernetes Engine > Workload > frontend > frontend-xxxxxx under Managed pods > click on server of Containers: **server** under Pod specification.



Kubernetes Engine > Workloads > frontend > pod details > server under Containers.



Same info when you go from Cluster > Nodes > frontend services.

References:

* [kubectl edit](https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/#kubectl-edit).
* [Images and ImagePullPolicy](https://kubernetes.io/docs/concepts/containers/images/)
* [kubectl patch](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#patch) is possible, but the data model for this frontend deployment is not simple. It is time consuming to create the CLI command.
* [Updating API Objects in place using kubectl patch](https://kubernetes.io/docs/tasks/manage-kubernetes-objects/update-api-object-kubectl-patch/)
* [kubectl set image](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#-em-image-em-) can be used to update to the new image. However, this command does not have an option (flag) to update imagePullPolicy.

kubectl set image deployment/frontend server=gcr.io/qwiklabs-resources/onlineboutique-frontend:v2.1

* syntax: kubectl set image deployment/[name of deployment] [container name]=[image name]:[image tag]

To verify that the app is working, kubectl get service --namespace=dev, copy the **EXTERNAL\_IP** for **frontend-external** service and run it in a new browser window. You should see a new different frontpage.

**Task 4. Autoscale from estimated traffic**

A marketing campaign is coming up that will cause a traffic surge on the OnlineBoutique shop. Normally, you would spin up extra resources in advance to handle the estimated traffic spike. However, if the traffic spike is larger than anticipated, you may get woken up in the middle of the night to spin up more resources to handle the load.

You also want to avoid running extra resources for any longer than necessary. To both lower costs and save yourself a potential headache, you can configure the Kubernetes deployments to scale automatically when the load begins to spike.

1. Apply **horizontal pod autoscaling** to your **frontend deployment** in order to handle the traffic surge.
2. Scale based on a target cpu percentage of 50.
3. Set the pod scaling between 1 minimum and Max Replicas maximum.

Of course, you want to make sure that users won’t experience downtime while the deployment is scaling.

4. To make sure the scaling action occurs without downtime, set the deployment to scale with a target cpu percentage of 50%. This should allow plenty of space to handle the load as the autoscaling occurs.

5. Set the deployment to scale between 1 minimum and Max Replicas maximum pods.

But what if the spike exceeds the compute resources you currently have provisioned? You may need to add additional compute nodes.

6. Next, ensure that your cluster is able to automatically spin up additional compute nodes if necessary. However, handling scaling up isn’t the only case you can handle with autoscaling.

7. Thinking ahead, you configure both a minimum number of nodes, and a maximum number of nodes. This way, the cluster can add nodes when traffic is high, and reduce the number of nodes when traffic is low.

8. Update your **cluster autoscaler** to scale between **1 node minimum** and **6 nodes maximum**.

9. Lastly, run a load test to simulate the traffic surge.

Fortunately, **OnlineBoutique** was designed with built-in load generation. Currently, your dev instance is simulating traffic on the store with ~10 concurrent users.

10. In order to better replicate the traffic expected for this event, run the load generation from your loadgenerator pod with a higher number of concurrent users with this command. Replace **YOUR\_FRONTEND\_EXTERNAL\_IP** with the IP of the frontend-external service:

kubectl exec $(kubectl get pod --namespace=dev | grep 'loadgenerator' | cut -f1 -d ' ') -it --namespace=dev -- bash -c 'export USERS=8000; locust --host="http://YOUR\_FRONTEND\_EXTERNAL\_IP" --headless -u "8000" 2>&1'

Now, observe your **Workloads** and monitor how your cluster handles the traffic spike. You should see your recommendationservice crashing or, at least, heavily struggling from the increased demand.

Apply **horizontal pod autoscaling** to your **recommendationservice** deployment. Scale based off a target cpu percentage of 50 and set the pod scaling between 1 minimum and 5 maximum.

**Note:** The process of scaling from the load test and having to provision any new nodes will take a couple of minutes altogether.

👇 👇 👇 Solution:

1. Applying horizontal pod autoscaling to frontend deployment.

kubectl autoscale deployment/frontend --cpu-percent=50 --min=1 --max=[Max Replicas]

To verify, kubectl get hpa

2. Enable node autoscaling for GKE cluster.

gcloud container clusters update [Cluster Name] --enable-autoscaling --min-nodes=1 --max-nodes=6

Reference and things to note:

* [gcloud container clusters update](https://cloud.google.com/sdk/gcloud/reference/container/clusters/update)
* node autoscaling is enabled for the default node pool if node pool is not specified with the --node-pool flag
* --min-nodes and --max-nodes are on a **per zone basis** for the active node pool.
* --total-min-nodes and --total-max-nodes apply to **all nodes** in the node pool.

3. Get IP of frontend-external service.

kubectl get services

4. Run load generation.

kubectl exec $(kubectl get pod --namespace=dev | grep 'loadgenerator' | cut -f1 -d ' ') -it --namespace=dev -- bash -c 'export USERS=8000; locust --host="http://YOUR\_FRONTEND\_EXTERNAL\_IP" --headless -u "8000" 2>&1'

Code explanation:

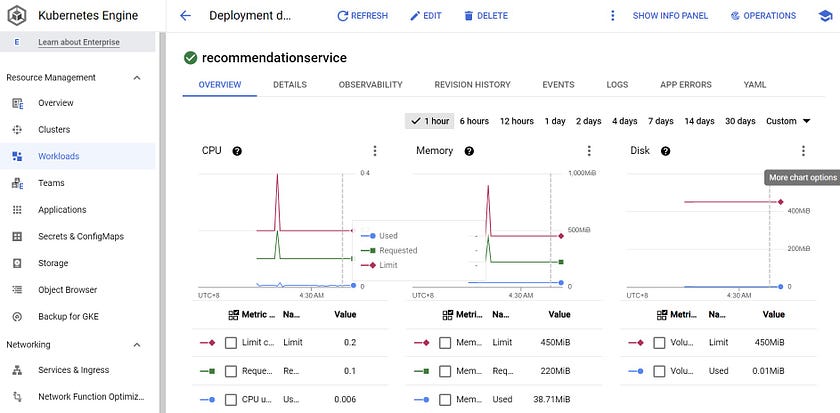
* the $ ( ) expression is for command substitution and it invokes a subshell environment to run the command within the braces of $ ( ). The stdout is returned to the parent command in the $ ( ) position.
* kubectl get pod --namespace=dev will return all the pods within the dev namespace
* the pod names are then filtered for 'loadgenerator' using grep and the line with 'loadgenerator' is output to the next code block.
* cut is a Linux/Unix command-line utility. It is used in this case, to cut the data piped in from the grep code block.
* the -f1 flag (--fields=LIST) tells cut to select the 1st field
* -d option tells cut utility to use a single character space ' ' as the delimiter between fields.
* aim of codes within $() is to get the name of the pod running the locust load generator.
* kubectl exec is to execute a command within the pod.
* -it is the flag to force open an interactive terminal shell within the pod (first container by default). Allows you to pass command/data via stdin (the keyboard) to application thread, and receive stdout(your screen).
* What comes after -- are the commands for the app within container.
* bash specifies invocation of a bash shell
* -c tells bash to read and execute the *command\_string* within 'export ...2>&1', then exit.
* *export USERS=8000* is to set the “USERS” variable to 8000, i.e. 8000 concurrent users, simulating a traffic spike
* locust is the load testing tool installed on the pod
* --headless is to run locust without the web UI
* -u "8000" specifies 8000 users
* For 2>&1, 2 refers to the second file descriptor of the process, i.e. stderr. > means redirection. &1 means the target of the redirection should be the same location as the first file descriptor, i.e. stdout . So, 2>&1 first directs output to stdout and then redirects stderr there as well.

References:

* [cut command syntax](https://linuxize.com/post/linux-cut-command/)
* [Bash shell commands — invoking bash](https://www.gnu.org/software/bash/manual/bash.html#Invoking-Bash)
* [locust load testing tool](https://locust.io/)
* [running locust without the web UI](https://docs.locust.io/en/stable/running-without-web-ui.html)
* [Bash I/O redirection](https://tldp.org/LDP/abs/html/io-redirection.html)

5. In the Cloud console, go to Navigation menu > Kubernetes Engine > Workloads > recommendationservice overview. Refresh to observe effect of load test. On each CPU/Memory/Disk chart, click on the 3 dots at the top, and select “Expand chart legend” for clarity. You may see CPU usage time increasing past requested towards limit. You may not ..

6. Apply horizontal pod autoscaler to recommendationservice.



For my lab, recommendationservice was not stressed ..

**Task 5. (Optional) Optimize other services**

While applying horizontal pod autoscaling to your frontend service keeps your application available during the load test, if you monitor your other workloads, you’ll notice that some of them are being pushed heavily for certain resources.

If you still have time left in the lab, inspect some of your other workloads and try to optimize them by applying autoscaling towards the proper resource metric.

You can also see if it would be possible to further optimize your resource utilization with Node Auto Provisioning.

👇 👇 👇 Solution:

1. Try to enable node auto provisioning. This allows cluster autoscaler to create new node pools with different node specifications.

gcloud container clusters update [Cluster Name] --enable-autoprovisioning --max-cpu= --max-memory=

Reference:

* [gcloud container clusters update — enable-autoprovisioning](https://cloud.google.com/sdk/gcloud/reference/container/clusters/update#--enable-autoprovisioning)
* --max-cpu refers to the max number of cores (e.g. 4) to which the cluster can scale,
* --max-memory refers to the max number of gigabytes of memory (e.g. 5) to which the cluster can scale.

To verify, **Navigation menu > Kubernetes Engine > Clusters** to observe whether new node pool created. [None observed for me]

2. “optimize-utilization” for autoscaler profile is likely NOT an option for a production e-commerce website which may encounter spiky traffic. It is more suitable for batch workloads.

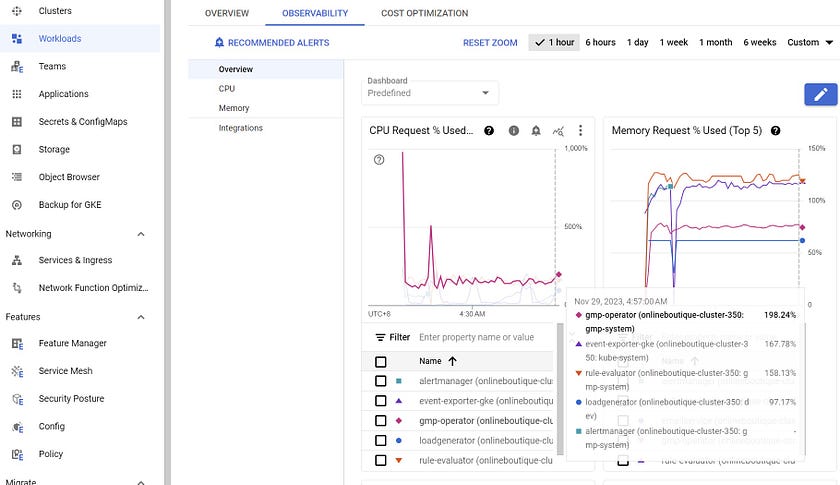
gcloud container clusters update [Cluster Name] --autoscaling-profile=optimize-utilization

3. Vertical Pod Autoscaling (VPA) will not give accurate results for this lab as VPA requires time (at least 24 hours recommended) to collect usage data. Relevant command:

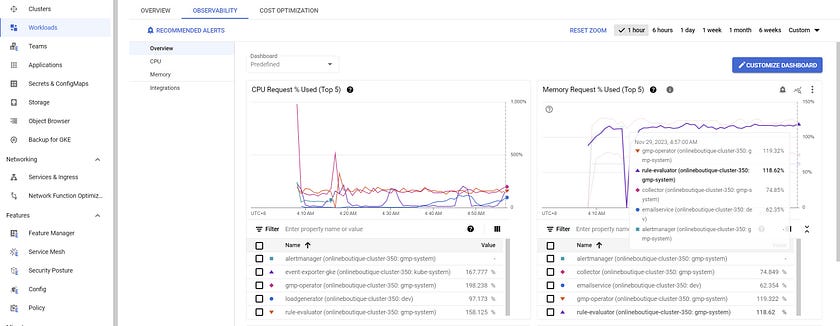
gcloud container clusters update [Cluster Name] --enable-vertical-pod-autoscaling

Need to add VPA.yaml referring to target deployment and apply manifest file.

For my lab, only Kube system resources were stressed:



Observability for Workloads CPU Requests



Observability for W