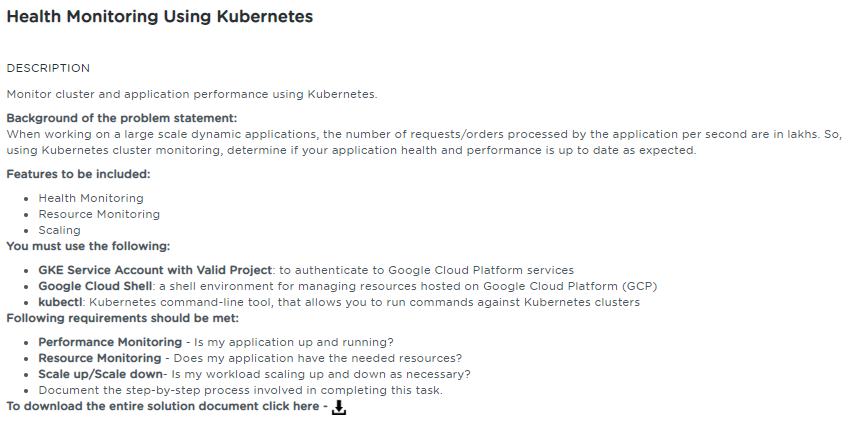
**Health Monitoring Using Kubernetes - Assessment**



**Table of Contents**

[**1.** **Project Abstract** 1](#_Toc44798422)

[**2.** **Work Environment** 2](#_Toc44798423)

[**2.1.** **Prerequisites** 2](#_Toc44798424)

[**2.2.** **Setting up Kubernetes Cluster** 2](#_Toc44798425)

[**3.** **Health/Resource Monitoring** 3](#_Toc44798426)

[**3.1.** **Kubernetes Metrics Server** 3](#_Toc44798427)

[**3.1.1.** **Deploy Metrics-Server** 3](#_Toc44798428)

[**3.1.2.** **Metrics-Server Service Details** 4](#_Toc44798429)

[**3.1.3.** **View the Metrics of Pods/Nodes** 5](#_Toc44798430)

[**4.** **Scale up/Scale down application cluster size/number of nodes** 6](#_Toc44798431)

[**4.1.** **Create Horizontal Pod Auto scale (HPA) definition file** 6](#_Toc44798432)

[**4.2.** **Auto scale (Scale up & Scale down event)** 6](#_Toc44798433)

[**5.** **References** 8](#_Toc44798434)

# **Project Abstract**

The intent of this project is to perform the following activities:

* Monitor the performance of your application and identify if it is up and running as expected.
* Monitor resources in terms of CPU/Memory/disk so as to ensure that your application has enough resources required for its proper functioning.
* Scale up/scale down the application cluster size/no. of nodes.

# **Work Environment**

## **Prerequisites**

Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster.

**Note:**

Not using the GKE Cluster as mentioned in the problem statement and requirement specification as am facing issue in creating the Cluster in my Trial account

Am using the SimpliLearn Practice lab session to implement this assessment.

## **Setting up Kubernetes Cluster**

Using the SimpliLearn Practice Lab session, Kubernetes Cluster is being setup with one master and two worker nodes

Run the below command on the master

|  |
| --- |
| $ sudo –i  $ kubeadm reset |

**kubeadm reset** is responsible for cleaning up a node local file system from files that were created using the kubeadm init or kubeadm join commands

Run the below command in order to set up the Kubernetes control plane

|  |
| --- |
| $ kubeadm init |

After executing the init command, Run the below commands to initialize the configuration and set the correct permissions

|  |
| --- |
| $ mkdir -p $HOME/.kube  $ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  $ sudo chown $(id -u):$(id -g) $HOME/.kube/config |

On Master, Run the below command to verify the master has all the required components running successfully

|  |
| --- |
| $ kubectl get pods –n kube-system |

Run the below command on the master to create the join-token

|  |
| --- |
| $ kubeadm token create --print-join-command |

On the slave nodes(kslave1 and kslave2), Run the command

|  |
| --- |
| $ kubeadm join 172.31.17.255:6443 --token wrfgen.elfhgxgoeqd7bnxn --discovery-token-ca-cert-hash sha256:655827d00fec5fa80d71caf58fa67b9efbc8824f1224952a4ad9fb5574fc6eaa |

On Master node, Run the following command to verify the worker nodes are joined the cluster

|  |
| --- |
| $ kubectl get nodes |

For the command output, please refer the screenshots document (Section: References)

# **Health/Resource Monitoring**

In addition to application metrics that can be monitored using kubectl logs of running pods, Stackdriver custom metrics can also use measurements of your cluster's infrastructure not included in system metrics, such as container Disk I/O. You can deploy your own infrastructure monitoring agents to collect and push these metrics to Stackdriver.

## **Kubernetes Metrics Server**

* Metrics Server is a scalable, efficient source of container resource metrics for Kubernetes built-in auto scaling pipelines.
* Collects resource metrics from Kubelets and exposes them in Kubernetes apiserver through [Metrics API](https://github.com/kubernetes/metrics) for use by [Horizontal Pod Autoscaler](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/).
* Metrics API can also be accessed by **kubectl top**, making it easier to debug auto scaling pipelines.

You can use Metrics Server for:

* CPU/Memory based horizontal auto scaling (learn more about [Horizontal Pod Autoscaler](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/))

## **Deploy Metrics-Server**

Run the below command to deploy the Metrics server

|  |
| --- |
| $ kubectl apply -f <https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.6/components.yaml> |

The command output creates the metrics server as shown below

clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created

clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created

rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created

apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created

serviceaccount/metrics-server created

deployment.apps/metrics-server created

service/metrics-server created

clusterrole.rbac.authorization.k8s.io/system:metrics-server created

clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created

Deploy the Metrics server patch for the correct execution

|  |
| --- |
| $ kubectl patch deploy metrics-server -p "$(cat k8s-metrics-server.patch.yaml)" -n kube-system |

**Note**:

Get the **k8s-metrics-server.patch.yaml** file from the link given below

wget-c https://gist.githubusercontent.com/initcron/1a2bd25353e1faa22a0ad41ad1c01b62/raw/008e23f9fbf4d7e2cf79df1dd008de2f1db62a10/k8s-metrics-server.patch.yaml

Run the below command to view the status of the Metrics server

|  |
| --- |
| $ kubectl get pods -n kube-system |

The command output shows the status of the metrics server as shown below

NAME READY STATUS RESTARTS AGE

coredns-5c98db65d4-6tjhg 1/1 Running 2 25h

coredns-5c98db65d4-pdf7h 1/1 Running 2 25h

etcd-kmaster 1/1 Running 2 25h

kube-apiserver-kmaster 1/1 Running 2 25h

kube-controller-manager-kmaster 1/1 Running 2 25h

kube-proxy-chz6s 1/1 Running 2 25h

kube-proxy-qgxs8 1/1 Running 2 25h

kube-proxy-tnw9c 1/1 Running 2 25h

kube-scheduler-kmaster 1/1 Running 2 25h

metrics-server-8787d6d45-hwhzv 1/1 Running 0 23s

weave-net-cd8dx 2/2 Running 5 25h

weave-net-s7z8b 2/2 Running 7 25h

weave-net-v98dt 2/2 Running 6 25h

Run the below command to view the status of the Metrics server

|  |
| --- |
| $ kubectl logs metrics-server-8787d6d45-hwhzv -n kube-system |

The command output shows the logs of the metric-server as shown below:

I0630 14:23:19.023359 1 serving.go:312] Generated self-signed cert (/tmp/apiserver.crt, /tmp/apiserver.key)

I0630 14:23:19.513254 1 secure\_serving.go:116] Serving securely on [::]:4443

## **Metrics-Server Service Details**

Run the below command to view the details of the metrics-server service

|  |
| --- |
| $ **kubectl get svc –n kube-system**  NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP,9153/TCP 27h  metrics-server ClusterIP 10.98.232.58 <none> 443/TCP 154m  $ **kubectl describe svc metrics-server -n kube-system**  Name: metrics-server  Namespace: kube-system  Labels: kubernetes.io/cluster-service=true  kubernetes.io/name=Metrics-server  Annotations: kubectl.kubernetes.io/last-applied-configuration:  {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"kubernetes.io/cluster-service":"true","kubernetes.io/name":"Me...  Selector: k8s-app=metrics-server  Type: ClusterIP  IP: 10.98.232.58  Port: <unset> 443/TCP  TargetPort: main-port/TCP  Endpoints: 10.36.0.1:4443  Session Affinity: None  Events: <none> |

## **View the Metrics of Pods/Nodes**

The content below shows the deployment yml file for the php-apache image

|  |
| --- |
| $ **apiVersion: apps/v1**  **kind: Deployment**  **metadata:**  **creationTimestamp: null**  **labels:**  **run: php-apache**  **name: php-apache**  **spec:**  **replicas: 1**  **selector:**  **matchLabels:**  **run: php-apache**  **strategy: {}**  **template:**  **metadata:**  **creationTimestamp: null**  **labels:**  **run: php-apache**  **spec:**  **containers:**  **- image: k8s.gcr.io/hpa-example**  **name: php-apache**  **ports:**  **- containerPort: 80**  **resources:**  **requests:**  **cpu: 200m** |

Create the deployment object and view the deployment object (php-apache) running status

|  |
| --- |
| $ **kubectl create –f hpa\_deployment.yml**  **deployment.apps/php-apache created**  **$ kubectl get pods**  **NAME READY STATUS RESTARTS AGE**  **php-apache-d89b5f47b-zqscp 1/1 Running 0 17s** |

Below is the content for the Service definition file to enable the communication for the Deployment object (php-apache)

|  |
| --- |
| $ **apiVersion: v1**  **kind: Service**  **metadata:**  **creationTimestamp: null**  **name: php-apache**  **spec:**  **ports:**  **- port: 80**  **protocol: TCP**  **targetPort: 80**  **selector:**  **run: php-apache**  **status:**  **loadBalancer: {}** |

To test the pod communication through the Service, Run the curl command as shown below:

|  |
| --- |
| $ kubectl get svc  NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 28h  php-apache ClusterIP 10.107.200.76 <none> 80/TCP 2m50s  $ curl 10.107.200.76  root@kmaster:~/k8s\_health\_monitoring# curl 10.107.200.76 |

Run the below command to view the CPU and Memory metrics of the running pod php-apache-d89b5f47b-zqscp

|  |
| --- |
| $ kubectl top pods  NAME CPU(cores) MEMORY(bytes)  php-apache-d89b5f47b-zqscp 1m 13Mi |

Run the below command to view the CPU and Memory metrics of the nodes

|  |
| --- |
| $ kubectl top nodes  NAME CPU(cores) CPU% MEMORY(bytes) MEMORY%  kmaster 107m 5% 1131Mi 7%  kslave1 31m 1% 446Mi 11%  kslave2 39m 1% 422Mi 11% |

# **Scale up/Scale down application cluster size/number of nodes**

This will allow you to auto scale your application at runtime based upon the number of nodes a cluster requires to execute a particular number of requests/orders to handle large scale data applications.

## **Create Horizontal Pod Auto scale (HPA) definition file**

Below content shows the definition of the Horizontal Pod Auto scale definition file which has the minimum and maximum replicas definition based on the targetCPUUtilizationPercentage of the php-apache Deployment object

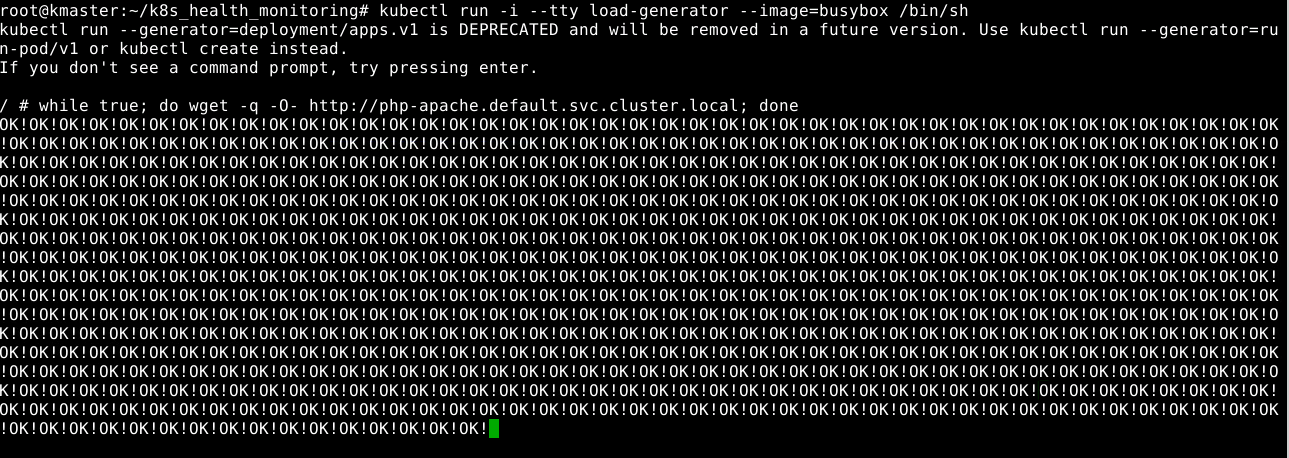
|  |
| --- |
| **apiVersion: autoscaling/v1**  **kind: HorizontalPodAutoscaler**  **metadata:**  **creationTimestamp: null**  **name: php-apache**  **spec:**  **maxReplicas: 10**  **minReplicas: 1**  **scaleTargetRef:**  **apiVersion: apps/v1**  **kind: Deployment**  **name: php-apache**  **targetCPUUtilizationPercentage: 50** |

Run the below command to create the HorizontalPodAutoScale object

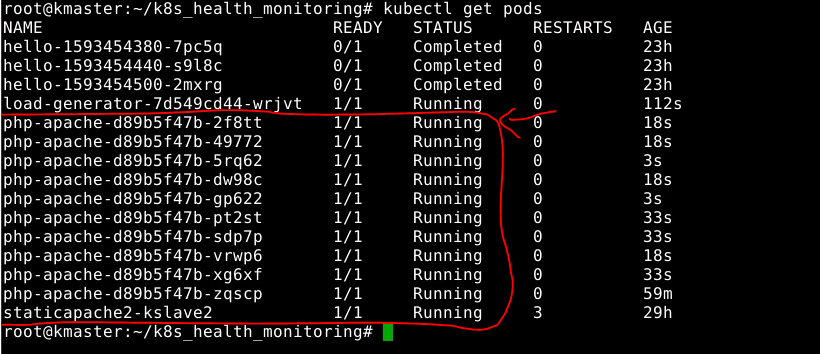
|  |
| --- |
| **$ kubectl create –f hpa\_scale.yml**  **horizontalpodautoscaler.autoscaling/php-apache created** |

## **Auto scale (Scale up & Scale down event)**

Below screenshot shows the deployment command generates the message to simulate the message generation to increase the CPU load so that we can prove the Scale up capability

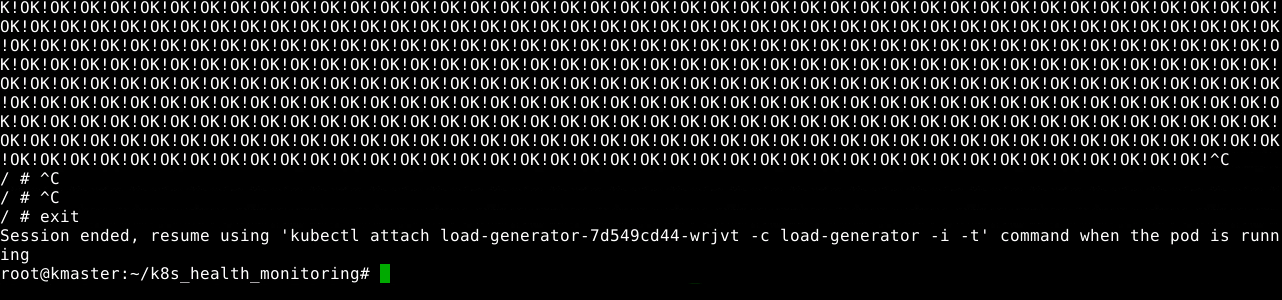


Below screenshot shows the Pod gets auto scaled according to the definition of the HorizontalPodAutoScale object i.e when the targetCPUUtilizationPercentage reaches 50%, the pod gets auto scaled.



Once it reaches the maximum replicas (defined as 10), we can stop generating the message just to prove the scale down capability i.e when the load decrease, the pod gets automatically scaled down to 1.

Below screenshot shows the message generation get stopped. This stops generating the message which in turn decreases the CPU load of the node



Below screenshot shows that only one pod (php-apache) is running as the pods get scaled down automatically when the CPU load gets decreased.



# **References**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Components** | **Reference** |
| 1 | Screenshots Document | 3.Health\_Monitoring\_using\_k8s\_screenshots.docx |
| 2 | Sources | 3.Health\_Monitoring\_using\_k8s\_sources.docx |