

Multitier web application on cloud

Cloud Computing and Network Structure



April 27, 2017

Northeastern University

Contents

[Multi-Tier Web Application on Cloud 2](#_Toc481178534)

[Problem Statement 2](#_Toc481178535)

[Objective 2](#_Toc481178536)

[Design and Deployment Model 2](#_Toc481178537)

[Implementation 6](#_Toc481178538)

[Demonstration Results 10](#_Toc481178539)

[Conclusion 10](#_Toc481178540)

[Appendix 12](#_Toc481178541)

# Multi-Tier Web Application on Cloud

## Problem Statement

The document presents an overview of web application and its deployment over cloud using Kubernetes. The web application represents structural information as diagrams of abstract graphs and networks and provides diagrams in the form of SVG (Scalable Vector Graphics). To enable the web application support automatic deployment, balancing heavy loads efficiently, resource monitoring, auto-scaling and allow rolling updates with no downtime and making application highly available all the time.

## Objective

To design and deploy web application which takes textual information from the user and converts it into graphical representation using jetty web Server. The web application will be deployed over cloud using Google Container Engine and Kubernetes serving customer demand efficiently.

Using Kubernetes, the web application will be deployed quickly and predictably

* The web application will able to scale on fly
* It will able to seamlessly roll out its new features without going down
* Optimized used of resources
* Load balancing effectively

## Design and Deployment Model

#### Web Application Model

The web application provides a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.

The Application model take descriptions of graphs in a simple text language, and make diagrams in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser.

#### Web Server

Web application-server is a lightweight Java based HTTP server that invokes the Graphviz dot binary installed locally. An HTTP POST is submitted by the user with the dot graph as the request body and the server returns back a graph in SVG format. The web server uses the Graphviz Java API, a Java wrapper that invokes the dot binary using Runtime.exec.

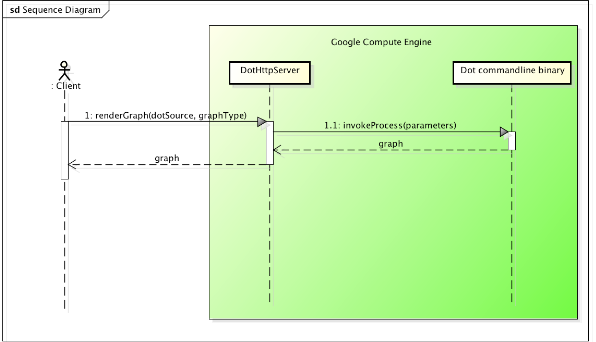


Figure 1: Sequence Diagram

#### Kubernetes Model

Kubernetes (k8s) is an open source platform for automating container operations such as deployment, scheduling and scalability across a cluster of nodes.

In this model Kubernetes have been implemented to serve below functionalities to the web application:

* Automate the deployment and replication of containers
* Scale in or out containers on the fly
* Organize containers in groups and provide load balancing between them,
* Easily roll out new versions of application containers,
* Provide container resilience, if a container dies it gets replaced

Kubernetes also can deploy a full cluster of multi-tiered containers (frontend, backend) with a just single configuration file.

The deployment of web application has been implemented by incorporating various components which are available in kubernetes. These are:

Cluster: A cluster is a group of nodes, they can be physical servers or virtual machines that has the Kubernetes platform installed.



Figure 2: Cluster in Kubernetes

Pods: are scheduled to Nodes and contain a group of co-located Containers and Volumes. Containers in the same Pod share the same network namespace and can communicate with each other using localhost. Pods are ephemeral rather than durable entities.

Labels: A Label is a key/value pair attached to Pods and convey user-defined attributes. For example, you might create a ‘tier’ and an ‘app’ tags to tag your containers by applying the Labels (tier=frontend, app=myapp) to your frontend Pods and Labels (tier=backend, app=myapp) to backend Pods. You can then use Selectors to select Pods with particular Labels and apply Services or Replication Controllers to them.

Replication Controllers: Replication Controllers ensure the specified number of Pod “replicas” are running at any one time. If you created a Replication Controller for a Pod and specified 3 replicas, it will create 3 Pods and will continuously monitor them. If one Pod dies then the Replication Controller will replace it to maintain a total count of 3.

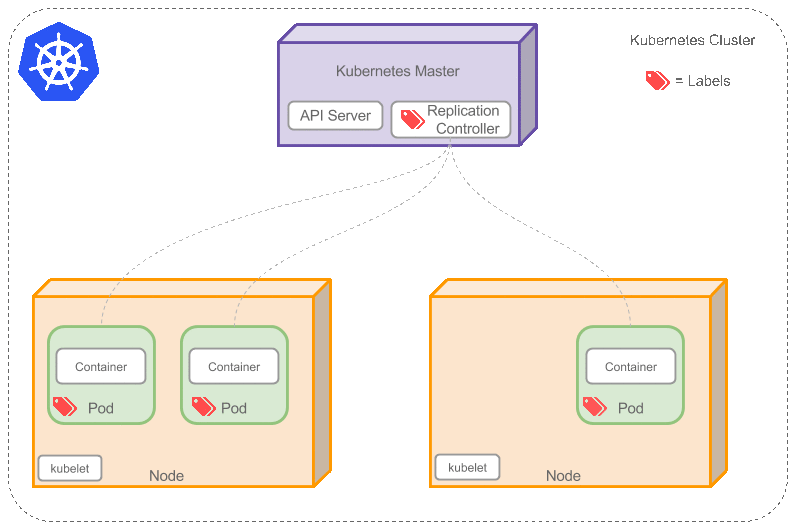


Figure 3: Replication Controllers

Services: Service is an abstraction that defines a set of Pods and a policy to access them. Services find their group of Pods using Labels.

There is a special type of Kubernetes Services called ‘Load Balancer’, which is used as an external load balancer to balance traffic between several Pods. Handy for load balancing Web traffic for example. The Load Balancer has been implemented in this design.

Nodes: It is a physical or virtual machine that acts as a Kubernetes worker, used to be called Minion. Each node runs the following key Kubernetes components:

* Kubelet: is the primary node agent.
* kube-proxy: used by Services to proxy connections to Pods as explained above.
* Docker or Rocket. The container technology that Kubernetes uses to create containers.

Kubernetes Master provides a unified view into the cluster and has several components such the Kubernetes API Server. The API Server provides a REST endpoint that can be used to interact with the cluster. The master also includes the Replication Controllers used to create and replicate Pods.

Container: Kubernetes provide container centric infrastructure. It offers agile application creation and deployment, provides frequent container image build with quick and easy rollbacks. It also provides resources utilization and isolation with predictable application performance.

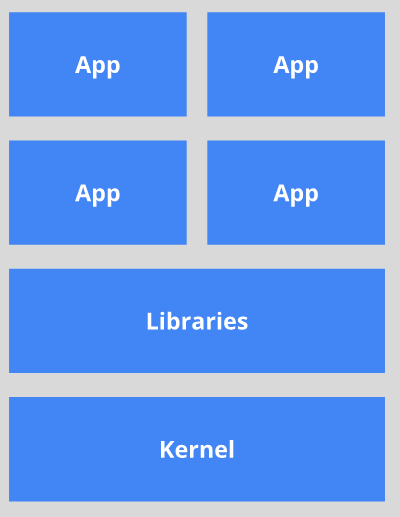
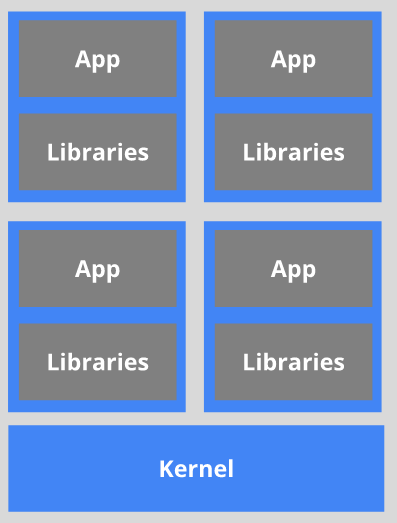
 

Figure 3: Application on host Figure 4: Deploy application using container

## Implementation

Once the design and deployment of model has been framed, the document further mentions about the implementation of multi-tier web application on Google container engine.

#### Creating Kubernetes Cluster

We need to create a Kubernetes cluster by creating a project inside Google cloud console. We need to enable the API to create compute engine Services.

Steps to create the cluster in GCE are:

1. list all the zones using the following Google Cloud SDK command:

gcloud compute zones list

1. Then we choose the us-central1-a

gcloud config set compute/zone us-central1-a

1. Next step we create a Kubernetes cluster:

gcloud container clusters create graphviz-app

By default if we do not specify the number of nodes and their types, then Container Engine will use three n1-standard-1 (1 vCPU, 3.75 GB memory) Compute Engine VMs for the cluster.

If we want to explicitly specify the number of nodes and their types then we would use the following command:

gcloud container clusters create graphviz-app \

--num-nodes 1 \

--machine-type g1-small

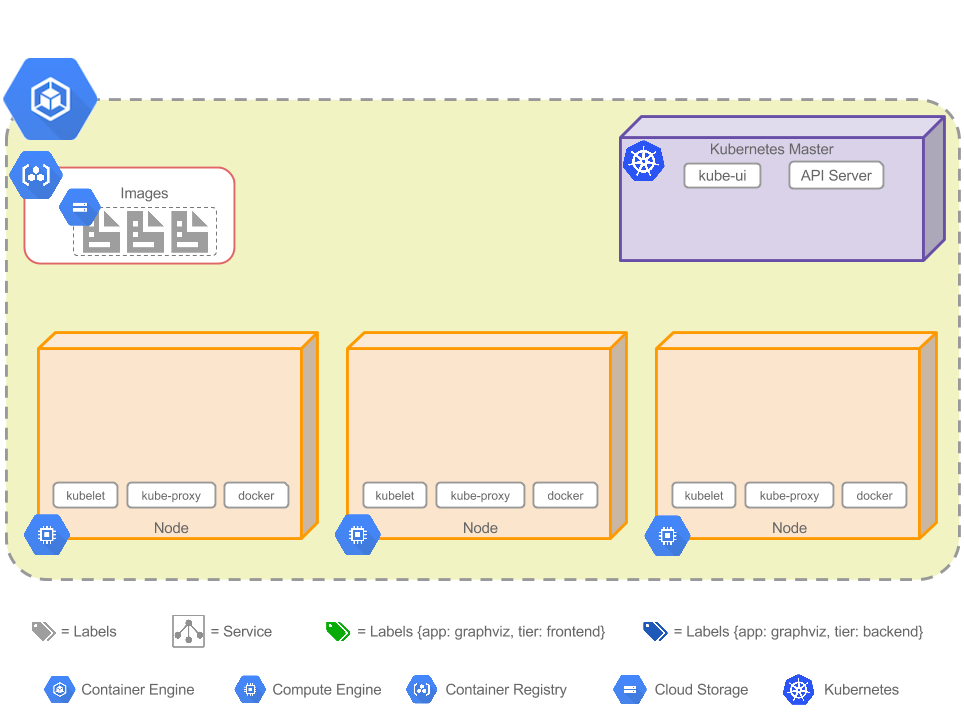


Figure 4: Kubernetes Cluster

#### Configuration files for Cluster

The configuration files for our cluster are in GitHub and contain the following four YAML files. Note Kubernetes configurations can also be written in JSON:

* **backend-controller.yaml** – Replication Controller “backend-contr” for the backend Pods, this will deploy two Pods.
* **frontend-controller.yaml** – Replication Controller “frontend-contr” for the frontend Pods, this will deploy three Pods.
* **backend-service.yaml** – Service “backend-service” to load balance the backend Pods.
* **frontend-service.yaml** – Service “frontend-service” an external load balancer for the frontend Pods that allows Web traffic.

#### Deploying the pods

1. We deploy the Replication Controller and backend Pods by running the following command:

kubectl create -f backend-controller.yaml

1. And verify that this Replication Controller is created

kubectl get rc

1. We can list all the Pods in the cluster using the following commands:

kubectl get pods -o wide

1. To confirm that the backend Pods are created by running the following command using the hostname of one of the nodes:

gcloud compute ssh gke-graphviz-app-e92dffb2-node-4hnk

1. Once we login to the node, we can view all the running Docker containers by running the following command:

sudo docker ps

1. We deploy the Replication Controller and Frontend Pods by running the following command:

kubectl create -f frontend-controller.yaml

1. And we can also verify from the UI and by running the kubectl command that three extra frontend Pods are created

kubectl get rc

kubectl get pods -o wide

After deploying the frontend and backend Replication Controller, we have a total of 5 Pods and the cluster as mentioned in the below figure.

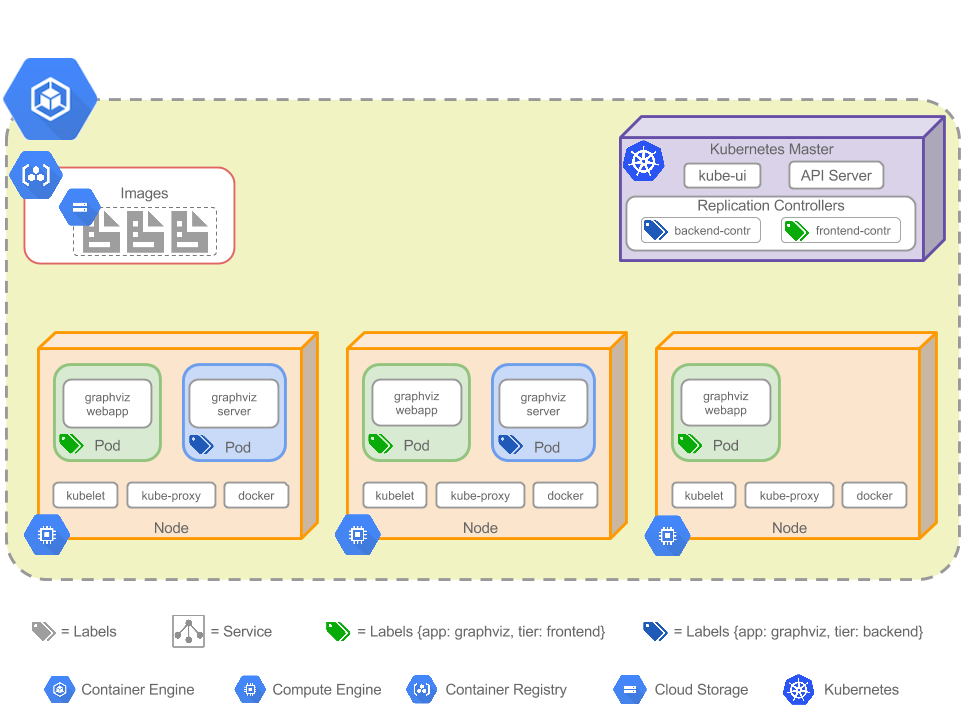


Figure 5: Cluster Status after deploying Pods

## Demonstration Results

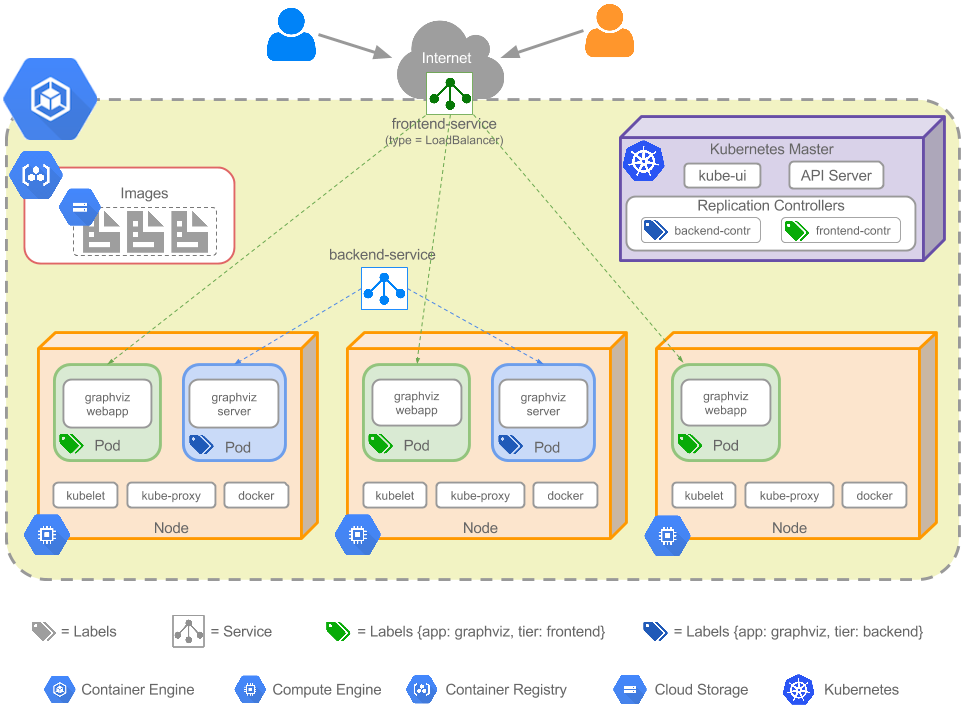


Figure 6: Multi-tier Web Application Implementation model on cloud

The cluster has three frontend containers running the Jetty server with the web application. The cluster also has two backend containers that run a simple HTTP server and have the web application installed. The model serves the load balancing feature offered by replication controller. Replication controller always ensure that the number of replicas mentioned for the container should be maintained as configured. The model also provides scaling feature by scaling up the number of replicas. It also offers high availability of web application. Using Service, we can provide transparent load balancing between the controllers as well as dynamically resolving the IP addresses to access any specific pods in the network.

The user pastes their DOT text on the frontend and clicks ‘Render Graph’ button, this fires an AJAX call to the Jetty server which sends a request to one of the backend servers to render the graph. The final graph in SVG is then rendered to the user.

## Conclusion

The web application has been successfully deployed over Kubernetes providing scaling functionality. The web application is also highly available due to its deployment over cloud using Kubernetes. It is also able to handle load balancing feature and run the application successfully.

The web application is successfully running on the IP address: <http://35.184.41.135/> and it is rendering the textual information provided into the graphical format.

We can test the log files which generates on pods to identify the processing of requests on frontend and backend controller.

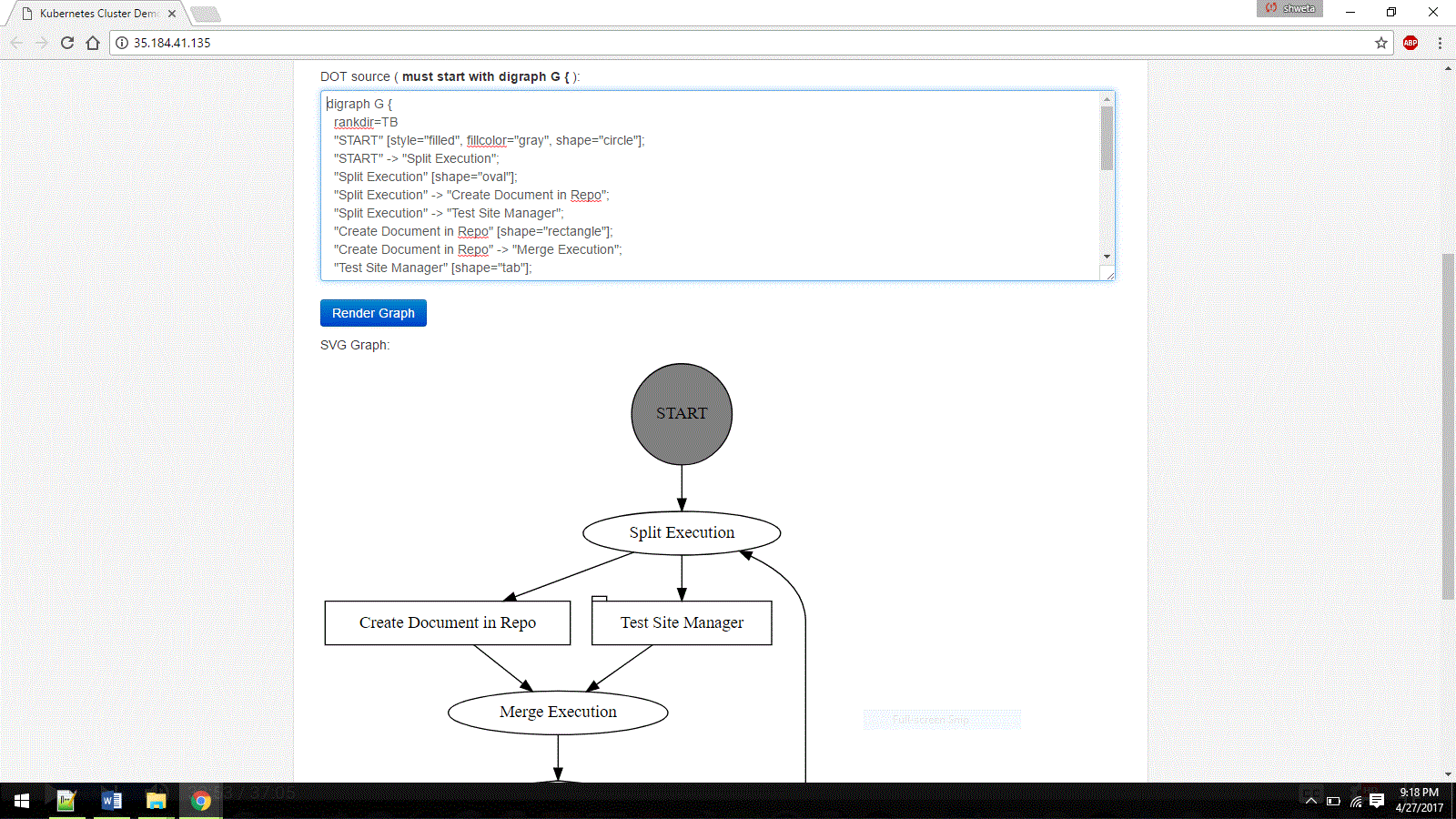


Figure 7: Web Application running on kubernetes cluster

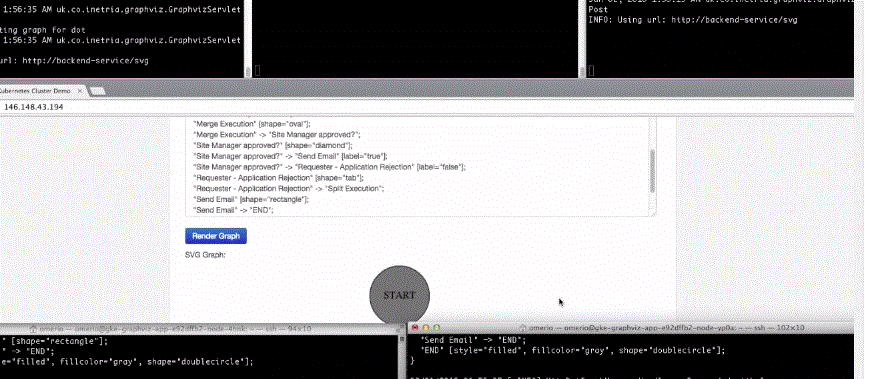


Figure 8:Load Testing on Web Application

## Appendix

This section mentions the code implementation of run multi-tier web application in cloud using Kubernetes and Google container engine. The steps are presented followed by the shell commands along with the console output that generates after each successful execution of every command.

There is a separate logfile.txt is attached in the final Project folder.

Also, the link of the git repository to access all the code files, logfile and respective snapshots taken during development is: <https://github.com/kapgateshweta/cloud_computing>

1. Check the present working directory using pwd command

***Command:***

$pwd

*Console Output:*

/home/kapgateshweta

1. Create a project in google cloud environment and enable the API to setup zones and instances for the project.Then compute the list of zones.

***Command :***

$gcloud compute zones list

*Console Output :*

kapgateshweta@cloudproject-165913:~$ gcloud compute zones list

NAME REGION STATUS NEXT\_MAINTENANCE TURNDOWN\_DATE

asia-east1-a asia-east1 UP

asia-east1-c asia-east1 UP

asia-east1-b asia-east1 UP

asia-northeast1-c asia-northeast1 UP

asia-northeast1-a asia-northeast1 UP

asia-northeast1-b asia-northeast1 UP

asia-southeast1-a asia-southeast1 UP

asia-southeast1-b asia-southeast1 UP

europe-west1-b europe-west1 UP

europe-west1-c europe-west1 UP

europe-west1-d europe-west1 UP

us-central1-b us-central1 UP

us-central1-c us-central1 UP

us-central1-f us-central1 UP

us-central1-a us-central1 UP

us-east1-d us-east1 UP

us-east1-b us-east1 UP

us-east1-c us-east1 UP

us-west1-b us-west1 UP

us-west1-a us-west1 UP

1. Configure the zone from the above list

***Command:***

$gcloud config set compute/zone us-central-a

*Console Output:*

Updated property [compute/zone].

1. Create a cluster named grahviz-app(any other name can also be provided)

***Command:***

$gcloud container clusters create graphviz-app

Console Output:

Creating cluster graphviz-app...done.

Created [https://container.googleapis.com/v1/projects/cloudproject-165913/zones/us-central1-a/clusters/graphviz-app].

kubeconfig entry generated for graphviz-app.

NAME ZONE MASTER\_VERSION MASTER\_IP MACHINE\_TYPE NODE\_VERSION NUM\_NODES STATUS

graphviz-app us-central1-a 1.5.6 104.198.133.51 n1-standard-1 1.5.6 3 RUNNING

1. See the description of the created cluster to login into the dashboard which enables us to see the CPU, memory consumption of the clusters. If not specified then it creates default specifications as: three nodes(3VMs) with 1cpu core and 3.75 GB of ram

***Command:***

$ gcloud container clusters describe graphviz-app

*Console Output:*

clusterIpv4Cidr: 10.88.0.0/14

createTime: '2017-04-27T15:09:06+00:00'

currentMasterVersion: 1.5.6

currentNodeCount: 3

currentNodeVersion: 1.5.6

endpoint: 104.198.133.51

initialClusterVersion: 1.5.6

instanceGroupUrls:

- https://www.googleapis.com/compute/v1/projects/cloudproject-165913/zones/us-central1-a/instanceGroupManagers/gke-graphviz-app-default-pool-9e4ea79c-grp

legacyAbac:

enabled: true

locations:

- us-central1-a

loggingService: logging.googleapis.com

masterAuth:

clientCertificate: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMyekNDQWNPZ0F3SUJBZ0lSQUlzT002UDhLMS9tdFd6S2xEK29QakF3RFFZSktvWklodmNOQVFFTEJRQXcKTHpFdE1Dc0dBMVVFQXhNa1lUTXpOMkUyWXpVdE5qZGlOeTAwTmpka

kxUazBOREF0TURSa09EQTBOR0poTmpBeQpNQjRYRFRFM01EUXlOekUxTURrd04xb1hEVEl5TURReU5qRTFNRGt3TjFvd0VURVBNQTBHQTFVRUF4TUdZMnhwClpXNTBNSUlCSWpBTkJna3Foa2lHOXcwQkFRRUZBQU9DQVE4QU1JSUJDZ0tDQVFFQXgraUtjUmFWUVc

rcVhIbE0KeldiUTBnQ1pCZDJyOVF0SlJNRzE1bjJIYkZFZDJvaGdWZCtLSlFSZUUyTjhPV0xhZ1hGRWNDc2pEOFgwRDQvSgpuRnI1c2lqNXh0dlpTRlZjNFJQMEVtaFRPQVFYcnA2Tjl1bWNwRkp0OTVxWUhKcE4wQTBBaTFtRHFoY0ZDU29JCmJJMUhCU2J6YjFLV

yt0VGpudmxnTnJIcEQrV0gvQW1tWTBHVkJJeFI4R0hmcnJHWGhIUEgwckdrd0ZFOGs0WlQKRE9QdDNOR0xUbzR6Zll2b0RvRnVQVUFlaW5nUXN5K3JRVzM4QnJoUFVwNjN1b0FOakV3cVh1TWFlUGJ5TEZ3Qwozck95WnE1VlAwUDJleTZERkNCOXBHUlBFMHk2c00

0ejhXNVFOUlN6c3IwVzNUOUNjQXVoZXN0MTlkSWpxUVJlCjdnZDZmd0lEQVFBQm94QXdEakFNQmdOVkhSTUJBZjhFQWpBQU1BMEdDU3FHU0liM0RRRUJDd1VBQTRJQkFRQUEKTGVsc1d1YzRkdlRGRFo3cHFwV2FiQVF1OVMvNFRIbGhMQW95UThtVWdCc2tVeTNib

lYxYUJEYjVsd1hKN2VFQQpzZ1NyS0VzRTZiR1pkQ0hsSEM0bHpNRW91SS9HT3JyZHRLYlJURENldE8yeGNhYVBQbWlnWjBBc05yZTRMektrCmpaY1hKY1IwdU1QOTFGQjBxLzZjR2txeEZyK2hkOURiZnYxbGF0dTJFZUVaRGMxQ0NReXpFdkIxQUM0TEduNXAKUjF

XeE5Xc2Uyd0dLWHRWb1BEMVBkZHk0WUtocFJuUEVuSVplN1pud1lieDFJSWZoaHd5T3gxZ3JvWGpqTDY3TAp3WkN6MWYvanNXOXF3eGU5V3d4MGhzdVZWaURZVmN6TFpnbGkwdkhqb29WRForUisvdXM1bmJldW1zTzlQZmwvCjUyTHlSY2RKY1RLai8yVEpiNWFtC

i0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K

clientKey: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBUUVBeCtpS2NSYVZRVytxWEhsTXpXYlEwZ0NaQmQycjlRdEpSTUcxNW4ySGJGRWQyb2hnClZkK0tKUVJlRTJOOE9XTGFnWEZFY0NzakQ4WDBENC9KbkZyNXNpajV4d

HZaU0ZWYzRSUDBFbWhUT0FRWHJwNk4KOXVtY3BGSnQ5NXFZSEpwTjBBMEFpMW1EcWhjRkNTb0liSTFIQlNiemIxS1crdFRqbnZsZ05ySHBEK1dIL0FtbQpZMEdWQkl4UjhHSGZyckdYaEhQSDByR2t3RkU4azRaVERPUHQzTkdMVG80emZZdm9Eb0Z1UFVBZWluZ1F

zeStyClFXMzhCcmhQVXA2M3VvQU5qRXdxWHVNYWVQYnlMRndDM3JPeVpxNVZQMFAyZXk2REZDQjlwR1JQRTB5NnNNNHoKOFc1UU5SU3pzcjBXM1Q5Q2NBdWhlc3QxOWRJanFRUmU3Z2Q2ZndJREFRQUJBb0lCQVFDZlY1MnlFbndaMG5ZSwppNjFLTTkvUzVDTVU1K

0kreWlZMnBlamx0YjBSKzM0K1NHSHhVT0wrS29NS0ZVeFpPWkJOaVFreUYxQTFMR1lqCldVc3NVOStmd01XSFdvblEvQ215Um9OUGdOU0VXODY4b3lrcFhJQTlBbThnNDVySGdiN3dUTXpZUmh1S29ocWoKZVZ1Sm12cTNBcHNiYXN0cFYvT3ZMYnNoSGFKVnhnZm9

MWFJNdjRMditCWDd6enVLcHB2UGZVNkJ1TW5oam43UgpzQzZUcGFWZXFSNGRiUHRwd01zYTR6UXA3LzM5Ym55dnd4RFdUcm01V01nVDVhR0RiYzdmSjc5bVpISVpncFdmClBzdEpWYlNUeGxsNVNsQzQyZnl0bE5FaU5ZbnRLZjRNbUFVVytkNFR2Z29JbWR4eEdXV

FJ2Rzh0bzdSNFpPYUsKNFlNM1h5cUJBb0dCQU8rYit2OFlXL2xtVVFLcmFoUnEwMVFiS2tia0czN0JTWlVvYStyd3dGK00rQTlYeHZTQgpqUkNJeThuVXRUUTB0RHM5aCtSaENuUldCcHBzakY3Nk1ESVYrSGtKNitzczJTSm8zam83VThRZHBaYnBXVUF4CldDMkl

kNGJZWmV6V2dCeVZrTEZHamd4THp6czRkRTdhWUhncnp4VDJvSlgwcUZ4clFnMjVlS0doQW9HQkFOV1YKVWhNQi91MFpTcTJsaGpyR1pxOXVUTWI4a1EyL0FZay94RWdLZWRKbzZMaXdySkFUb0JrdXg4VDdkcGovTlJqVgpISVJqT25qbVV3bnI3eXczLzdYckpVQ

mZEVVpERVFadm82Uyt4amV3a2dzd2Q1WUVVQzBsai9vZno2R0MzbisvCnVhbzYvVlNzWUJlUE1PUHpNdVU1Y0d3emVCYk5sNFRQSkRGRXN1Z2ZBb0dBWE5mQnIrcU9hTnJlZFB4YWhFTEkKVkFTYW94RnNIZmRqUGFxRjlta0hBWDhyOXpYNTMyLzdhdmEya0NRZGV

uY1pod0xXandXUytYVUQ1ZjQ5ZmVmUAptUGl1YUtnUmcvVEN4ZlNBZDJ2WmoxbzFlWDQ5bVFRRnVNN3lFWXYwSW9zUzVRQ0hLa20rYk1GTE1SVTA3QUR0CjlHSjFRUmczTFNPNXdPWEdtRzdQY1VFQ2dZQkg3Qm1RUndyR1lzN3YvSjVmNm5HbnY3ZmhCaW4rZkFTV

GFFSzcKazdPRU5FbXg3K1NjTzgrY3kxRUFOMEd2c2JqNm5NRVduNmVRRU5lRHJzeUNrblRzbUx1Wm1xaUlzWXJwSk96eApURURKZmc4ZjczbEZyandsaDlHZEE5dFNZeFJ6NW1jcU9PVVgvMWY0ZHdLYXNFbkNRUDVub1VsZVBHczZrTnA2Ck1NMGlOd0tCZ0grYSt

XOFh3OVZhYzliSk8wZVpmdlByaG5NYTIrSEwxaHRrejRIRVUrdFVkelNhejdWYmx5K2EKaktSQ0o0bmpvZURBanVuQnB3MksrMkxTY081bHRwOVVINjQwSTFYVjg4WWg0M3lNeXczVlkxaXFQTmdielFXagppTktxZk56TUFIWGNVekNvNEJNRGdLdmlqN3BoelBoW

kRKZG5oS0VSU2tGM0tNaTJvQ016Ci0tLS0tRU5EIFJTQSBQUklWQVRFIEtFWS0tLS0tCg==

clusterCaCertificate: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURDekNDQWZPZ0F3SUJBZ0lRYS9lQXlqd0R1UmV3dENxa2NPTTFzVEFOQmdrcWhraUc5dzBCQVFzRkFEQXYKTVMwd0t3WURWUVFERXlSaE16TTNZVFpqTlMwMk4ySTNMVFEyTj

JNdE9UUTBNQzB3TkdRNE1EUTBZbUUyTURJdwpIaGNOTVRjd05ESTNNVFV3T1RBM1doY05Nakl3TkRJMk1UVXdPVEEzV2pBdk1TMHdLd1lEVlFRREV5UmhNek0zCllUWmpOUzAyTjJJM0xUUTJOMk10T1RRME1DMHdOR1E0TURRMFltRTJNREl3Z2dFaU1BMEdDU3FH

U0liM0RRRUIKQVFVQUE0SUJEd0F3Z2dFS0FvSUJBUURITE5RTk9QVHRZY3VCZmNwUmN0MlMxcXZUQUdGM2RYSVNaZUlRWVRWUAp0cTJBc2xyUDFpVStHaFBQNzBqMmZQYnBqOE1zRzJsN0trQUoxU3crOWpXdFByam9LRE51b3Vwc0FrTDZzK3krCmlLeFppU2pLUH

ZlcENSS0FVR3lBSGY1ajlEa1oxTTBYUDRoa0pBMmt2ZTQwYll5RS9XemtIOTZQZXRwMG1DTXUKSFN0UHZlWUZWUkxtTnJRcVVRV2o2MisxT0NwZy9oMmQ1d1pYOHA0aVZtQkVnd3crakZqb2NrMThGRW5SN2tRTwpsWEtuWlhDNkh4a2FEcGd5cWVTR2xDZTRFK09N

SnRubjZ1cElTV3M5aURSUUxHcVlQQ2ducTZ4VWp4QnVjTGNRCld1R1J4bkh5NjQ5RUtDUDZTSnZVci9EQ285d3hoYXhoMEdrUjdVMFN5a0tkQWdNQkFBR2pJekFoTUE0R0ExVWQKRHdFQi93UUVBd0lDQkRBUEJnTlZIUk1CQWY4RUJUQURBUUgvTUEwR0NTcUdTSW

IzRFFFQkN3VUFBNElCQVFCTQp2cWZoY2FlQzlBbHovOGY5S2UzNVJkbWFNM0k2V2tRblNFRkY3eWo2c1QrZlVNekNYVVY3amtpV0tLWTExOXNKCkJXK1Q5dEtMb0llU2YxVFNSTExXdm1aVXd1MU5hTmxDMU81RUlvTEZzK2RzZ05XU0tHRktOUlZNVC82d1lUTEUK

U0cxc3lkSWM0cWlQckttMWVPdjUxUTkyclR6NmxoRDVNRGVtWEpHMW1wWVhRa3oxWEY1Ujd3NGtMTnVIU0VJbQpya0RERVdqdnJrL0VkdmM4RkdSZFNST0E4UStzdzVJQTU4R3lrNFVGQ3dCQ2g4QXRVMkZHWjBhU1pUajRLTEZsCkd6c0lUUjBldHl6ZHpiRXlyaj

kxU1l1aE5wUXhVbnlFQlN6TVJ2cU92T0RzamRmc0FLeWJucGFSMkJ5THhkNXEKODdsRitGdVoxUmJ4RkplbWNKWFIKLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=

***password: auLuHSGIg0BGL8pC***

***username: admin (Note: This is username and password for the dashboard UI)***

monitoringService: monitoring.googleapis.com

name: graphviz-app

network: default

nodeConfig:

diskSizeGb: 100

imageType: COS

machineType: n1-standard-1

oauthScopes:

- https://www.googleapis.com/auth/compute

- https://www.googleapis.com/auth/devstorage.read\_only

- https://www.googleapis.com/auth/service.management.readonly

- https://www.googleapis.com/auth/servicecontrol

- https://www.googleapis.com/auth/logging.write

- https://www.googleapis.com/auth/monitoring

serviceAccount: default

nodeIpv4CidrSize: 24

nodePools:

- config:

diskSizeGb: 100

imageType: COS

machineType: n1-standard-1

oauthScopes:

- https://www.googleapis.com/auth/compute

- https://www.googleapis.com/auth/devstorage.read\_only

- https://www.googleapis.com/auth/service.management.readonly

- https://www.googleapis.com/auth/servicecontrol

- https://www.googleapis.com/auth/logging.write

- https://www.googleapis.com/auth/monitoring

serviceAccount: default

initialNodeCount: 3

instanceGroupUrls:

- https://www.googleapis.com/compute/v1/projects/cloudproject-165913/zones/us-central1-a/instanceGroupManagers/gke-graphviz-app-default-pool-9e4ea79c-grp

management: {}

name: default-pool

selfLink: https://container.googleapis.com/v1/projects/cloudproject-165913/zones/us-central1-a/clusters/graphviz-app/nodePools/default-pool

status: RUNNING

version: 1.5.6

selfLink: https://container.googleapis.com/v1/projects/cloudproject-165913/zones/us-central1-a/clusters/graphviz-app

servicesIpv4Cidr: 10.91.240.0/20

status: RUNNING

zone: us-central1-a

1. To get the information of cluster. This will showcase all the links through which we can get access to various units. Kubernetes dashboard provides user interface to see the pods, clusters, etc in graphical format.

***Command:***

$kubectl cluster-info

*Console Output*:

Kubernetes master is running at https://104.198.133.51

GLBCDefaultBackend is running at https://104.198.133.51/api/v1/proxy/namespaces/kube-system/services/default-http-backend

Heapster is running at https://104.198.133.51/api/v1/proxy/namespaces/kube-system/services/heapster

KubeDNS is running at https://104.198.133.51/api/v1/proxy/namespaces/kube-system/services/kube-dns

kubernetes-dashboard is running at https://104.198.133.51/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard

1. Create a controller called frontend which is designed to configure the frontend controller which will deploy three pods. We deploy the controller and Pods by the following command

***Command:***

$kubectl create -f backend-controller.yaml

*Output*:

replicationcontroller "backend-contr" created

1. To get the replication controller which in this case are two can be obtained by the following command.

***Command:***

$kubectl get rc

*Output:*

NAME DESIRED CURRENT READY AGE

backend-contr 2 2 0 29s

1. To look at the pods which are created using the controller. The command is as follows:

***Command:***

$kubectl get pods -o wide

*Output:*

NAME READY STATUS RESTARTS AGE IP NODE

backend-contr-m34j1 1/1 Running 0 4m 10.88.2.4 gke-graphviz-app-default-pool-9e4ea79c-64gz

backend-contr-prh3w 1/1 Running 0 4m 10.88.1.3 gke-graphviz-app-default-pool-9e4ea79c-z52m

1. We can check one of the backend controller by performing ssh command into it.

***Command:***

$ gcloud compute ssh gke-graphviz-app-default-pool-9e4ea79c-64gz

*Output:*

WARNING: The public SSH key file for gcloud does not exist.

WARNING: The private SSH key file for gcloud does not exist.

WARNING: You do not have an SSH key for gcloud.

WARNING: SSH keygen will be executed to generate a key.

This tool needs to create the directory [/home/kapgateshweta/.ssh]

before being able to generate SSH keys.

Do you want to continue (Y/n)? y

Generating public/private rsa key pair.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/kapgateshweta/.ssh/google\_compute\_engine.

Your public key has been saved in /home/kapgateshweta/.ssh/google\_compute\_engine.pub.

The key fingerprint is:

81:b6:66:c3:8e:92:d8:e9:e1:33:ad:b6:1b:fa:7c:92 kapgateshweta@cs-6967-devshell-vm-a5643406-aa99-4d1f-b9ed-20613dee0ae2-a6

The key's randomart image is:

+---[RSA 2048]----+

| |

| . |

| o . |

| o . . |

| \* S |

| o = . |

|. Bo. . |

| =E+o |

|.oOO |

+-----------------+

Updating project ssh metadata...|Updated [https://www.googleapis.com/compute/v1/projects/cloudproject-165913].

Updating project ssh metadata...done.

Warning: Permanently added 'compute.185747669857056584' (RSA) to the list of known hosts.

Connection to 35.184.4.175 closed.

1. The next step is to create front end controller which is running Jitty web server into it. It will deploy three pods. The pods can be created by mentioning the value of replica in the yaml file

***Command:***

$kubectl create -f frontend-controller.yaml

*Console Output:*

replicationcontroller "frontend-contr" created

1. To check the replication controllers as well as the pods. This step is optimal.

***Command:***

$kubectl get rc

*Console Output:*

NAME DESIRED CURRENT READY AGE

backend-contr 2 2 2 19m

frontend-contr 3 3 3 2m

1. The below command is to get the pods running. We can see based on the yaml file configuration, there are three front end controllers and two backend controllers.

***Command:***

$kubectl get pods -o wide

*Console Output:*

NAME READY STATUS RESTARTS AGE IP NODE

backend-contr-m34j1 1/1 Running 0 19m 10.88.2.4 gke-graphviz-app-default-pool-9e4ea79c-64gz

backend-contr-prh3w 1/1 Running 0 19m 10.88.1.3 gke-graphviz-app-default-pool-9e4ea79c-z52m

frontend-contr-2kjm3 1/1 Running 0 2m 10.88.1.5 gke-graphviz-app-default-pool-9e4ea79c-z52m

frontend-contr-c0kbc 1/1 Running 0 2m 10.88.2.5 gke-graphviz-app-default-pool-9e4ea79c-64gz

frontend-contr-d1j86 1/1 Running 0 2m 10.88.1.4 gke-graphviz-app-default-pool-9e4ea79c-z52m

1. This command is to ssh into any one of the node to see it is running properly. In this, I have chosen frontend controller as it has created after backend controller which have been already tested before.

***Command:***

$gcloud compute ssh gke-graphviz-app-default-pool-9e4ea79c-z52m

*Console Output:*

For the following instance:

- [gke-graphviz-app-default-pool-9e4ea79c-z52m]

choose a zone:

[1] asia-east1-a

[2] asia-east1-b

[3] asia-east1-c

[4] asia-northeast1-a

[5] asia-northeast1-b

[6] asia-northeast1-c

[7] asia-southeast1-a

[8] asia-southeast1-b

[9] europe-west1-b

[10] europe-west1-c

[11] europe-west1-d

[12] us-central1-a

[13] us-central1-b

[14] us-central1-c

[15] us-central1-f

[16] us-east1-b

[17] us-east1-c

[18] us-east1-d

[19] us-west1-a

[20] us-west1-b

Please enter your numeric choice: 12

Warning: Permanently added 'compute.8931511444951775048' (RSA) to the list of known hosts.

Enter passphrase for key '/home/kapgateshweta/.ssh/google\_compute\_engine':

Welcome to Kubernetes v1.5.6!

You can find documentation for Kubernetes at:

http://docs.kubernetes.io/

The source for this release can be found at:

/home/kubernetes/kubernetes-src.tar.gz

Or you can download it at:

https://storage.googleapis.com/kubernetes-release/release/v1.5.6/kubernetes-src.tar.gz

It is based on the Kubernetes source at:

https://github.com/kubernetes/kubernetes/tree/v1.5.6

For Kubernetes copyright and licensing information, see:

/home/kubernetes/LICENSES

1. Once we are inside the shell , we can check all the Dockers which are running on the kubernetes. The below command provides a list of all the dockers respective to the system and the web application.

***Command:***

$sudo docker ps

*Console Output:*

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

0134f438fc72 omerio/graphviz-webapp "/docker-entrypoint.b" 8 hours ago Up 8 hours k8s\_webapp.c26acaa1\_frontend-contr-d1j86\_default\_985602d5-2b61-11e7-b9bd-42010a800240\_bbfbe8d3

48ecb81ef1a8 omerio/graphviz-webapp "/docker-entrypoint.b" 8 hours ago Up 8 hours k8s\_webapp.c26acaa1\_frontend-contr-2kjm3\_default\_98552f5b-2b61-11e7-b9bd-42010a800240\_ac8d634d

a8f2e3b9c40f gcr.io/google\_containers/pause-amd64:3.0 "/pause" 8 hours ago Up 8 hours k8s\_POD.d8dbe16c\_frontend-contr-d1j86\_default\_985602d5-2b61-11e7-b9bd-42010a800240\_80251ee1

22f47d9843f6 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 8 hours ago Up 8 hours k8s\_POD.d8dbe16c\_frontend-contr-2kjm3\_default\_98552f5b-2b61-11e7-b9bd-42010a800240\_43a770bd

80e9dc09c525 omerio/graphviz-server "java -jar /opt/graph" 8 hours ago Up 8 hours k8s\_server.f795aed1\_backend-contr-prh3w\_default\_34e8c66a-2b5f-11e7-b9bd-42010a800240\_62e93008

64587b6c0623 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 8 hours ago Up 8 hours k8s\_POD.d8dbe16c\_backend-contr-prh3w\_default\_34e8c66a-2b5f-11e7-b9bd-42010a800240\_023cc4b7

349c19c270de gcr.io/google\_containers/fluentd-gcp:1.28.2 "/bin/sh -c 'rm /lib/" 9 hours ago Up 9 hours k8s\_fluentd-cloud-logging.6aa6c538\_fluentd-cloud-logging-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_51229922e92873f29e001ebdfb62368e\_09040be6

c6097603c6c1 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 9 hours ago Up 9 hours k8s\_POD.d8dbe16c\_fluentd-cloud-logging-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_51229922e92873f29e001ebdfb62368e\_adf65653

c3c80cdb8af8 gcr.io/google\_containers/kube-proxy:cf03177cc1a2711175fc70c089ff7474 "/bin/sh -c 'kube-pro" 9 hours ago Up 9 hours k8s\_kube-proxy.33bc2ebe\_kube-proxy-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_53a83392928a0c8ed723d2381448ae8d\_337ba988

f496c7c6ae69 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 9 hours ago Up 9 hours k8s\_POD.d8dbe16c\_kube-proxy-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_53a83392928a0c8ed723d2381448ae8d\_16a5314c

1. We can test the replication controller which is created. The purpose of replication controller is to always ensure that the number of replicas mentioned for the container should be maintained as configured. The below command will kill one of the controllers and replication controller in that case will create or replace the container.

There are two web application controllers and if we delete one of them, replication controller should either replace it or recreate it to maintain the replicas.

***Command:***

$ sudo docker kill 0134f438fc72

*Output:*

0134f438fc72

1. After the container has been killed we can run the below command to see the status of containers running in Docker.

The output of the command shows that, we still can see two replicas of the web application. The container has been replaced by replication container.

***Command:***

$sudo docker ps

*Console Output:*

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

3a3747504287 omerio/graphviz-webapp "/docker-entrypoint.b" 25 seconds ago Up 25 seconds k8s\_webapp.c26acaa1\_frontend-contr-d1j86\_default\_985602d5-2b61-11e7-b9bd-42010a800240\_18481a5a

48ecb81ef1a8 omerio/graphviz-webapp "/docker-entrypoint.b" 8 hours ago Up 8 hours k8s\_webapp.c26acaa1\_frontend-contr-2kjm3\_default\_98552f5b-2b61-11e7-b9bd-42010a800240\_ac8d634d

a8f2e3b9c40f gcr.io/google\_containers/pause-amd64:3.0 "/pause" 8 hours ago Up 8 hours k8s\_POD.d8dbe16c\_frontend-contr-d1j86\_default\_985602d5-2b61-11e7-b9bd-42010a800240\_80251ee1

22f47d9843f6 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 8 hours ago Up 8 hours k8s\_POD.d8dbe16c\_frontend-contr-2kjm3\_default\_98552f5b-2b61-11e7-b9bd-42010a800240\_43a770bd

80e9dc09c525 omerio/graphviz-server "java -jar /opt/graph" 8 hours ago Up 8 hours k8s\_server.f795aed1\_backend-contr-prh3w\_default\_34e8c66a-2b5f-11e7-b9bd-42010a800240\_62e93008

64587b6c0623 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 8 hours ago Up 8 hours k8s\_POD.d8dbe16c\_backend-contr-prh3w\_default\_34e8c66a-2b5f-11e7-b9bd-42010a800240\_023cc4b7

349c19c270de gcr.io/google\_containers/fluentd-gcp:1.28.2 "/bin/sh -c 'rm /lib/" 9 hours ago Up 9 hours k8s\_fluentd-cloud-logging.6aa6c538\_fluentd-cloud-logging-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_51229922e92873f29e001ebdfb62368e\_09040be6

c6097603c6c1 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 9 hours ago Up 9 hours k8s\_POD.d8dbe16c\_fluentd-cloud-logging-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_51229922e92873f29e001ebdfb62368e\_adf65653

c3c80cdb8af8 gcr.io/google\_containers/kube-proxy:cf03177cc1a2711175fc70c089ff7474 "/bin/sh -c 'kube-pro" 9 hours ago Up 9 hours k8s\_kube-proxy.33bc2ebe\_kube-proxy-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_53a83392928a0c8ed723d2381448ae8d\_337ba988

f496c7c6ae69 gcr.io/google\_containers/pause-amd64:3.0 "/pause" 9 hours ago Up 9 hours k8s\_POD.d8dbe16c\_kube-proxy-gke-graphviz-app-default-pool-9e4ea79c-z52m\_kube-system\_53a83392928a0c8ed723d2381448ae8d\_16a5314c

1. We can also test by checking all the pods that are running using the below command.

The Restart status shows that the frontend controller has been restarted by replication controller after we killed it.

*Command:*

$kubectl get pods

*Console Output:*

NAME READY STATUS RESTARTS AGE

backend-contr-m34j1 1/1 Running 0 9h

backend-contr-prh3w 1/1 Running 0 9h

frontend-contr-2kjm3 1/1 Running 0 8h

frontend-contr-c0kbc 1/1 Running 0 8h

frontend-contr-d1j86 1/1 Running 1 8h

1. We can scale up /down the replication controller using kubectl and we can mention the number of replicas. The below command can scale down the replicas to 2.

**Command:**

$kubectl scale rc frontend-contr --replicas=2

*Console Output*:

replicationcontroller "frontend-contr" scaled

root@frontend-contr-2kjm3:/var/lib/jetty# nslookup backend-service

Server: 10.91.240.10

Address: 10.91.240.10#53

Name: backend-service.default.svc.cluster.local

Address: 10.91.245.94

1. To provide connection to frontend and backend controller we need a way to dynamically resolve to access specific pod using Service. We provide transparent load balancing between the controllers. Below command is to deploy the frontend service.In similar fashion we also create service for backend controller.

***Command:***

$kubectl create -f frontend-service.yaml

*Console Output:*

service "frontend-service" created

1. To access the services created in the above commands for backend and frontend controllers, we can check using the below command:

We can see in the output front end and backend service along with the ports on which they are running.

***Command:***

$kubectl get services

*Console Output:*

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

backend-service 10.91.245.94 <none> 80/TCP 25m

frontend-service 10.91.244.56 <pending> 80:31638/TCP 43s

kubernetes 10.91.240.1 <none> 443/TCP 9h

1. When we want to run the web application, using the describe command we can get the IP address on which it is running after deploying it to kubernetes. The IP address in this case is 35.184.41.135 which is loadbalance Ingress value.

***Command:***

$ kubectl describe service frontend-service

*Console Output:*

Name: frontend-service

Namespace: default

Labels: app=graphviz

tier=frontend

Annotations: <none>

Selector: app=graphviz,tier=frontend

Type: LoadBalancer

IP: 10.91.244.56

LoadBalancer Ingress: 35.184.41.135

Port: <unset> 80/TCP

NodePort: <unset> 31638/TCP

Endpoints: 10.88.0.8:8080,10.88.1.5:8080,10.88.2.5:8080

Session Affinity: None

Events:

FirstSeen LastSeen Count From SubObjectPath Type Reason Message

--------- -------- ----- ---- ------------- -------- ------ -------

1m 1m 1 service-controller Normal CreatingLoadBalancer Creating load balancer

30s 30s 1 service-controller Normal CreatedLoadBalancer Created load balancer

22. We can place the above IP address in the web browser and can see the web application is running successfully. We can test the load balancing capability of kubernetes by logging into SSH of all the nodes and when we hit the render graph button , we can see the request is being processed in various nodes thereby balancing the load successfully.