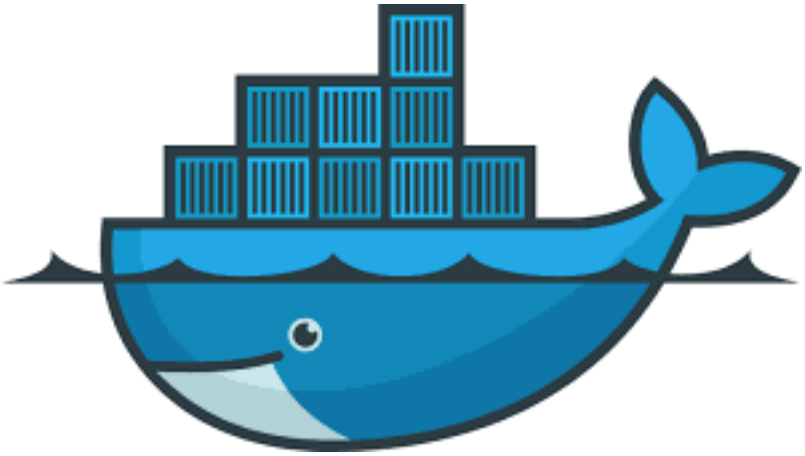
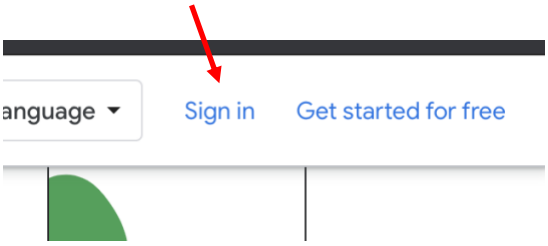


Intro to DevOps, Containers & Orchestration

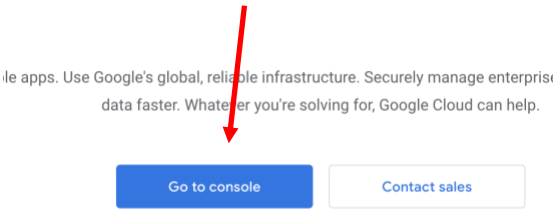
Ken Lowenstein
klowenst@cisco.com
November 2020

Lab Section A – Access Google Cloud Platform

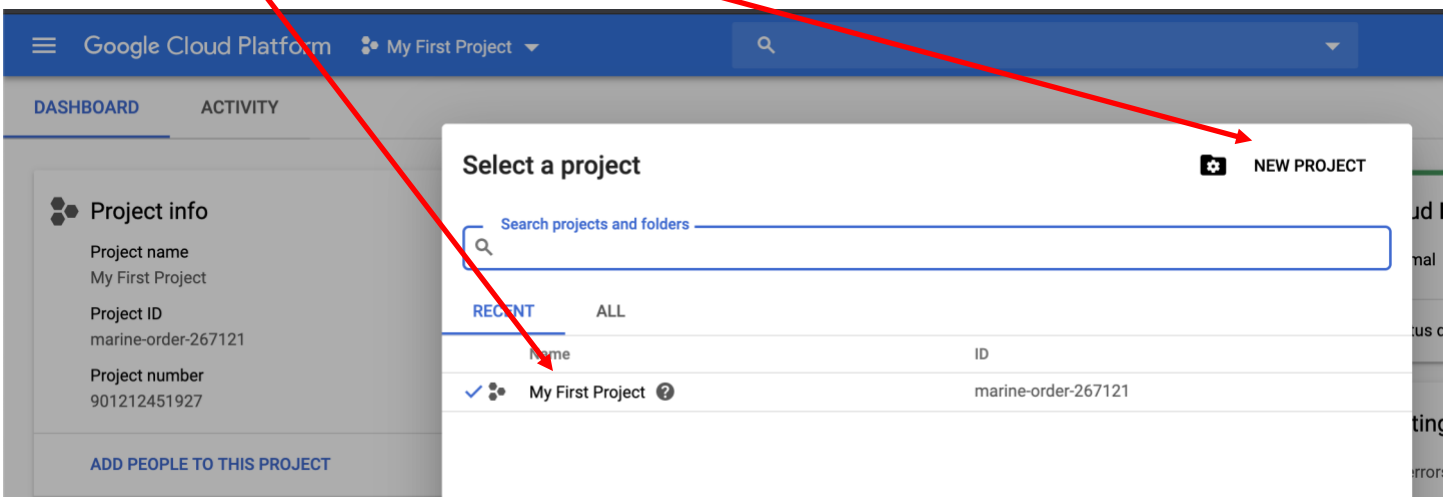
Login to GCP - https://cloud.google.com
Click “Sign In” button

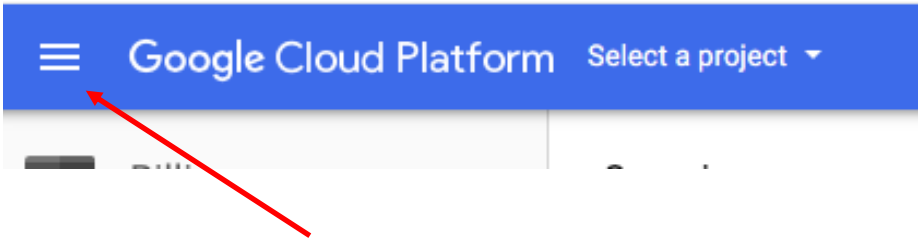


Click Go to console



Select an existing project or create a new one:



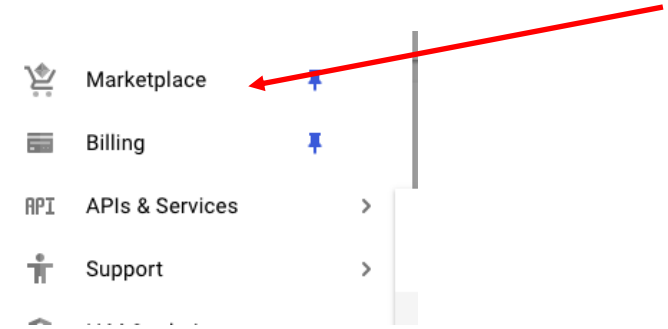


Remember this ‘hamburger’ icon – it brings up the main menu in GCP

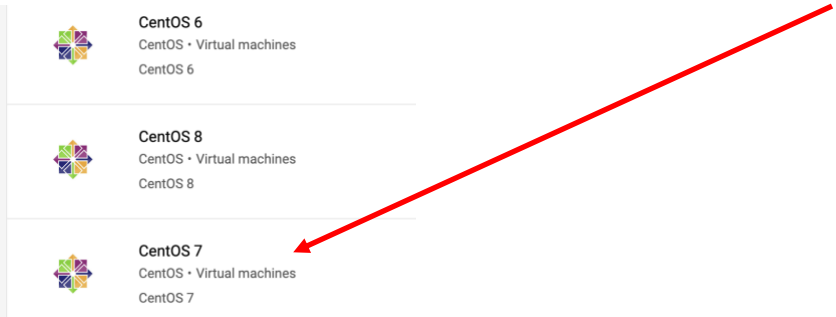
IMPORTANT NOTE: If you quit lab early be sure to delete all compute instances, Kubernetes Clusters and Network Load Balancers – the clock is running…

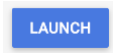
Lab Section B – Installing Docker

Click the 3 lines (hamburger) on menu and choose “Marketplace”

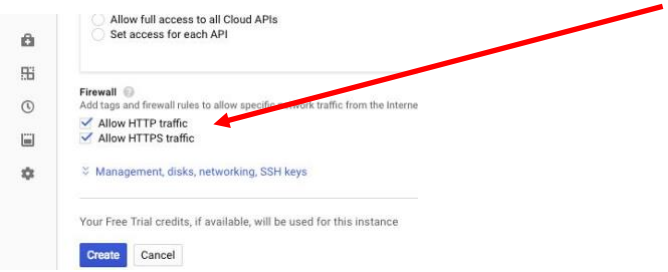


type “Centos” in the search bar and double click “Centos 7”

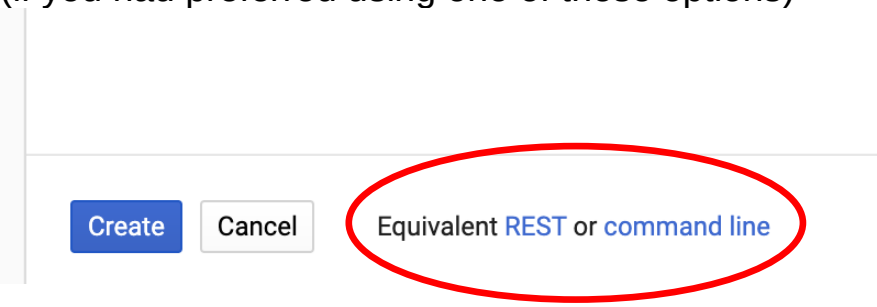


Click “Launch” 

Accept all defaults (or you can choose region local to you if you like)
AND click Allow http and https traffic under firewall section

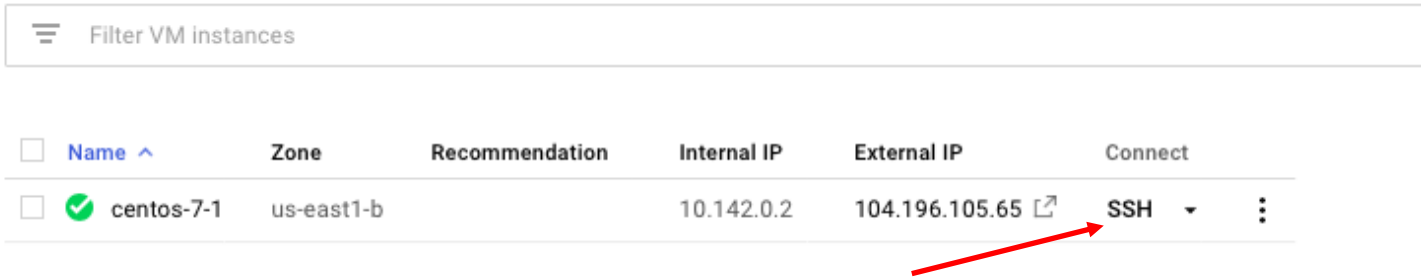


Before clicking “Create” View REST and/or command line to view REST or CLI output (if you had preferred using one of these options)



Now click create

Once started (icon will turn green) click on the SSH tab to open a remote console



NOTE: typed commands are shown in italics and highlighted. Some commands are in a smaller font (on one single line) as a work-around with PDF line break issues

Install the docker engine in the Centos VM

sudo yum check-update

Update the package database

curl -fsSL https://get.docker.com | sh

- Add official Docker repository, download and install latest Docker version

sudo systemctl start docker

- Start the Docker daemon

sudo systemctl status docker

- Verify Docker is running

You should see message similar to the following –

```
[klowenst@centos-7-1 ~]$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendo
   Active: active (running) since Thu 2020-10-29 13:41:32 UTC; 11s ago
     Docs: https://docs.docker.com
 Main PID: 1636 (dockerd)
    Tasks: 10
```

sudo systemctl enable docker.service

- Start the Docker daemon on start-up

sudo usermod -a -G docker $USER

- Run Docker daemon as “non-sudo” user

Exit the console by typing “exit”
Reopen the console to enable docker commands to run as a non-sudo user

type
docker info

- Verify docker daemon is running as regular user

Running command as non-sudo user should return status similar to below

```
[klowenst@centos-7-1 ~]$ docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 1
 Server Version: 19.03.13
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-f
```

Extra credit: If you would like to have the ability to see docker command options after typing ‘docker’ then tab key – follow these instructions.

sudo yum install -y bash-completion bash-completion-extras
sudo curl -L https://raw.githubusercontent.com/docker/compose/1.22.0/contrib/completion/bash/docker-compose -o /etc/bash_completion.d/docker-compose
source ~/.bash_profile
exec $SHELL -l

Working with Images

From the SSH window enter the following commands

docker images

- Note that there are currently no images on the system

docker ps

- There are no running containers

docker pull ubuntu

- download an image

```
[klowenst@centos-7-docker-installed ~]$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
6a5697faee43: Pull complete
ba13d3bc422b: Pull complete
a254829d9e55: Pull complete
Digest: sha256:fff16eea1a8ae92867721d90c59a75652ea66d29c05294e6e2f898704bdb8cf1
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[klowenst@centos-7-docker-installed ~]$
```

Docker is pulling the layered image

type *docker images* command again

```
[klowenst@centos-7-1 ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              d70eaf7277ea        5 days ago          72.9MB
[klowenst@centos-7-1 ~]$
```

We have the “latest” Ubuntu image stored locally, - we could have specified a version.

type *docker inspect <image ID>* - copy and paste <image ID> from previous “docker images” output

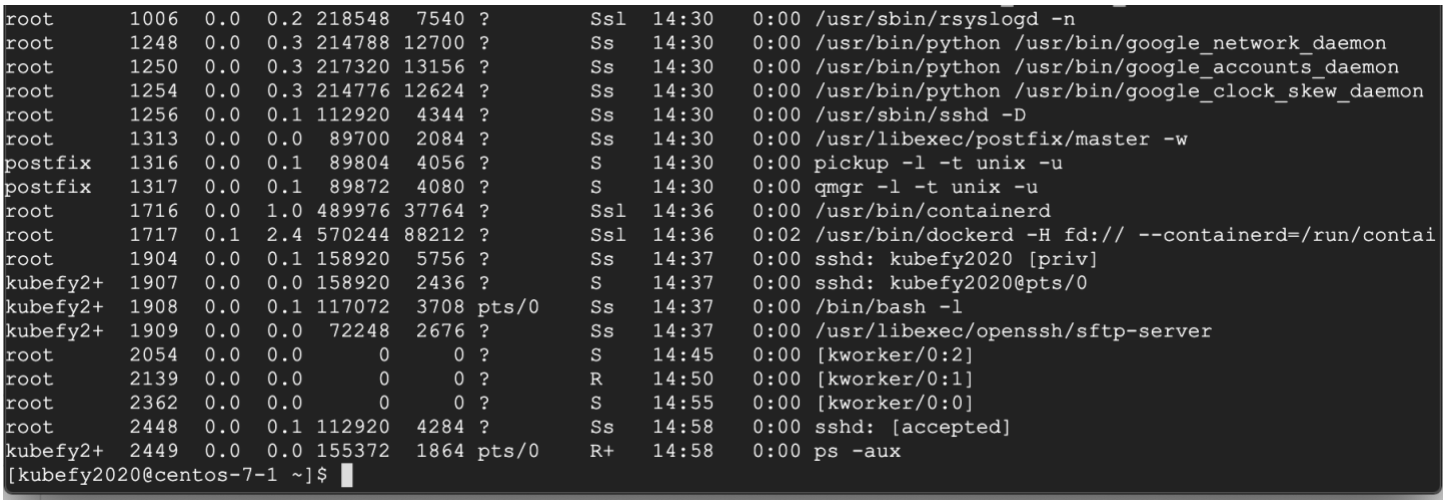
- i.e. (from above screen shot) – docker inspect d70eaf7277ea

Examine the output

Type

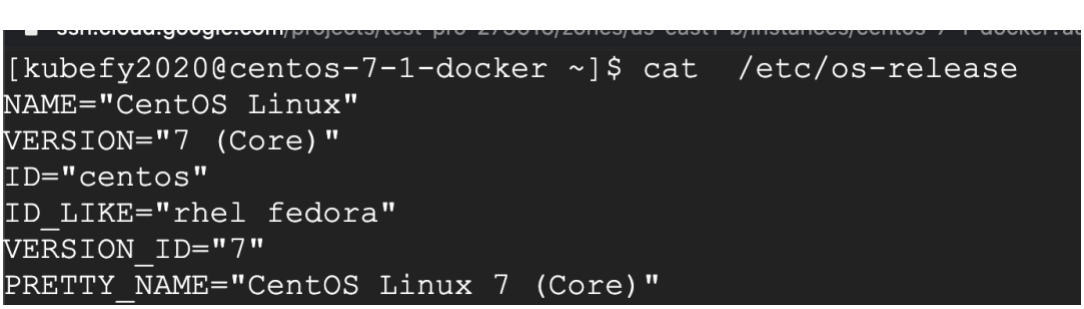
ps -aux

Notice all the processes currently running on your Centos 7 VM OS



Now type

cat /etc/os-release



Notice you are running “CentOS Linux 7”

Creating the first container

type
docker run -it ubuntu (-I = interactive –t=terminal)

you are now “in” the container – notice the prompt change

Type
ps -aux

```
[klowenst@centos-7-1 ~]$ docker run -it ubuntu
root@67ace5d20aec:/# ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.0   4100  2164 pts/0    Ss   13:49   0:00 /bin/bash
root           8  0.0  0.0   5880  1408 pts/0    R+   13:49   0:00 ps -aux
```

Only 2 processes are running in this container – “/bin/bash” & the command you just entered, “ps aux” (notice you are running as root user inside container)

Now type
cat /etc/os-release again

```
root@67ace5d20aec:/# cat  /etc/os-release
NAME="Ubuntu"
VERSION="20.04.1 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.1 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
```

We are now running “Ubuntu 20.04” inside the container

Let’s exit the container but leave it running

Press and hold “CTRL” key then “p-q”

You should now be back at your linux prompt

type:

docker ps

```
root@67ace5d20aec:/# [klowenst@centos-7-1 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
          NAMES
67ace5d20aec        ubuntu              "/bin/bash"         4 minutes ago       Up 4 minutes
          cranky_napier
[klowenst@centos-7-1 ~]$
```

Notice that the container is running, has a unique ID, the command is the “/bin/bash/” shell and a random short name has been assigned

Reattach to the running container

docker attach <image ID>

- where <image ID> is container ID from previous docker ps command
- we are now back in the container (notice prompt)

While back in the container type
traceroute yahoo.com

the command will fail as traceroute package is not installed

Let’s add traceroute to this container
apt-get update
apt-get install traceroute
now let’s try the command again
traceroute yahoo.com

Exit the container by typing “exit”

type
docker ps
notice the container is no longer running

```
[klowenst@centos-7-1 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
          NAMES
[klowenst@centos-7-1 ~]$
```

Now we can create a new image, adding the new “layer” (with traceroute installed)

Let’s get the container ID (the appended container still exists but not running)
docker ps -a (‘-a’ signifies all, not just running containers)

```
[klowenst@centos-7-1 ~]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
 PORTS               NAMES
67ace5d20aec        ubuntu              "/bin/bash"         7 minutes ago       Exited
                    cranky_napier
[klowenst@centos-7-1 ~]$
```

copy the container ID and paste after commit
docker commit <container ID> ubuntu_traceroute
create a new image

type *docker images* command and we will now see 2 images

```
[klowenst@centos-7-1 ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu_traceroute   latest              2c5fca0d2e83        20 seconds ago      99.7MB
ubuntu              latest              d70eaf7277ea        5 days ago          72.9MB
[klowenst@centos-7-1 ~]$
```

Note: You can restart either container by using a “docker run -it <container ID OR image name>“

Type *docker inspect ubuntu*
We see the original 3 layers

```
    },
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:47dde53750b4a8ed24acebe52cf31ad131e73a9611048fc2f92c9b6274ab4bf3",
            "sha256:0c2689e3f9206b1c4adfb16a1976d25bd270755e734588409b31ef29e3e756d6",
            "sha256:cc9d18e90faad04bc3893cfaa50b7846ee75f48f5b8377a213fa52af2189095c"
        ]
```

Type *docker inspect ubuntu_traceroute*

We now see the original 3 layers plus the layer we added

```
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:47dde53750b4a8ed24acebe52cf31ad131e73a9611048fc2f92c9b6274ab4bf3",
            "sha256:0c2689e3f9206b1c4adfb16a1976d25bd270755e734588409b31ef29e3e756d6",
            "sha256:cc9d18e90faad04bc3893cfaa50b7846ee75f48f5b8377a213fa52af2189095c",
            "sha256:f5d8b04f08578a87284c7e04c3958c25702bff99ed2218e7d676b28d291f0421"
        ]
```

Running a Web Server

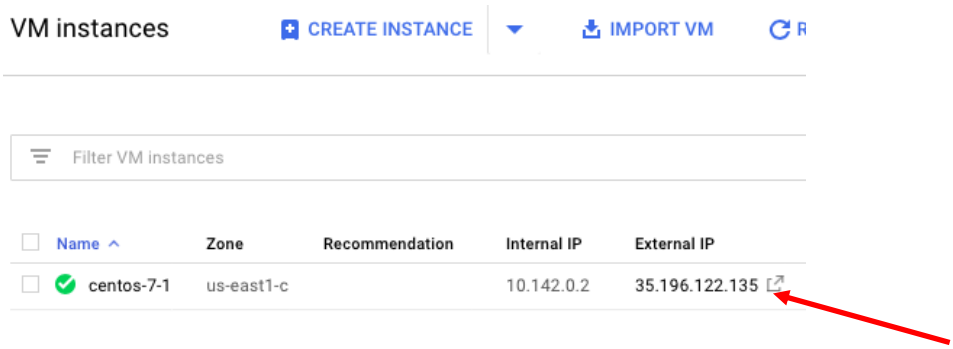
type
docker run -d -p 80:80 nginx

- docker will pull and run the image (-d = in detached mode / -p = publish container's port(s) to the host).
- You can verify by typing "*docker ps"*

Go to the Compute Engine VM Instances page

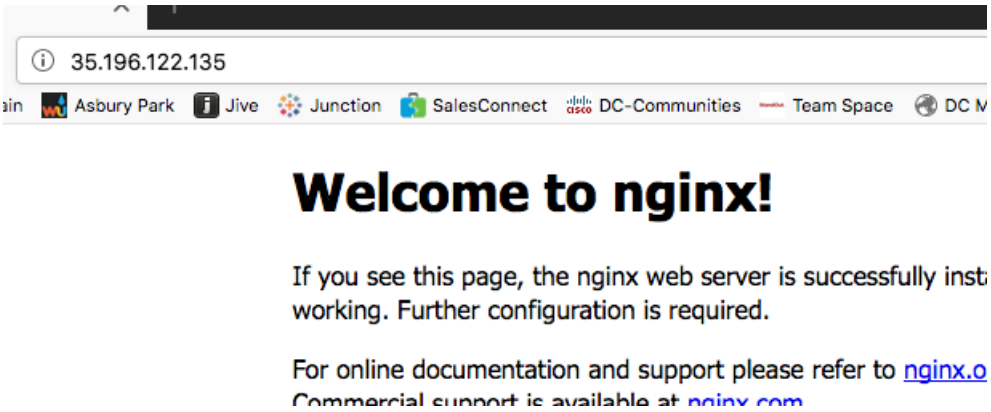
- (Click on 3 bars on top of page  > then click "Compute Engine"

Get the external IP address of the server.



In a new browser type in http://<external IP address>

The NGINX default web page should show.



Let's stop this nginx container and delete it.

Back in the Centos VM – type
docker stop <container id>
Now enter *docker ps* to verify it is no longer running

Creating Dockerfiles

Will a create simple Dockerfile that uses nginx and copies content into the container

First let's install a text editor (so we don't have to struggle using 'vi')
sudo yum install nano -y

Let's get some content
type
curl http://www.gutenberg.org/files/2600/2600-h/2600-h.htm >index.html

(gutenberg.org may have occasional outages as improvements are made. If you run into a 'time-out try one of these links instead)
curl http://dev.gutenberg.org/files/2600/2600-h/2600-h.htm >index.html
curl http://www.authorama.com/alice-in-wonderland-1.html >index.html

type
cat index.html

- to verify contents are there (this is a large file, hit CTRL-C to stop)

Let's create a docker file

type
nano Dockerfile ('Dockerfile' is case sensitive)

Paste the following 2 lines into the file

from nginx
copy ./index.html /usr/share/nginx/html

```
from nginx
copy ./index.html /usr/share/nginx/html
```

To save file
hit ‘control – o’ and ‘enter” to write file
hit ‘control – x’ to exit nano

type
cat Dockerfile

- to verify file was written

we will now build the image

type
docker build -t testweb .

- (be sure to include “.” (. = this directory)

Confirm that the image was built
type
docker images

- to verify image (testweb) is there

```
[klowenst@centos-7-1 ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
testweb             latest              ecced2a3e8b3        6 seconds ago       137MB
ubuntu_traceroute   latest              2c5fca0d2e83        3 minutes ago       99.7MB
ubuntu              latest              d70eaf7277ea        5 days ago          72.9MB
nginx               latest              f35646e83998        2 weeks ago         133MB
[klowenst@centos-7-1 ~]$
```

type
docker history testweb

- to see the layer we added to image several seconds ago

```
[klowenst@centos-7-1 ~]$ docker history testweb
IMAGE               CREATED             CREATED BY                                      SIZE
OMMENT
ecced2a3e8b3        45 seconds ago      /bin/sh -c #(nop) COPY file:ea63ff61269ed737…   4.07MB
f35646e83998        2 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:1d0a4127e78a26c1…   1.96kB
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:e7e183879c35719c…   1.2kB
<missing>           2 weeks ago         /bin/sh -c set -x     && addgroup --system -…   63.6MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.4        0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.3     0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:0dc53e7886c35bc21…   69.2MB
```

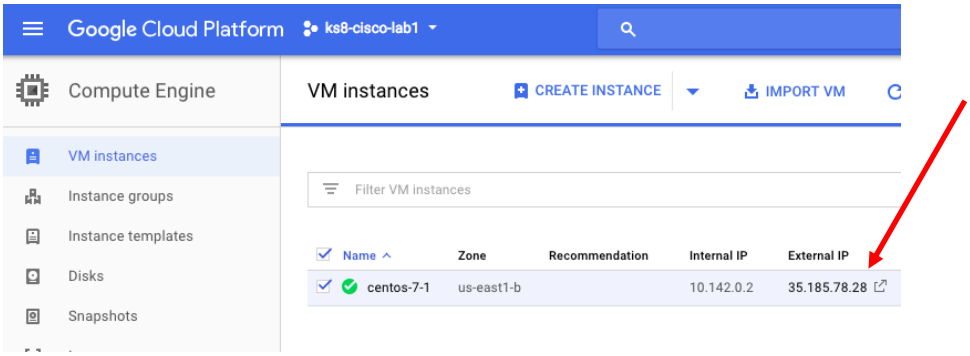
Type *docker history nginx* to view original image we copied from
– notice there is one less “layer”

```
[klowenst@centos-7-1 ~]$ docker history nginx
IMAGE               CREATED             CREATED BY                                      SIZE
OMMENT
f35646e83998        2 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL SIGTERM           0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:0fd5fca330dcd6a7…   1.04kB
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:1d0a4127e78a26c1…   1.96kB
<missing>           2 weeks ago         /bin/sh -c #(nop) COPY file:e7e183879c35719c…   1.2kB
<missing>           2 weeks ago         /bin/sh -c set -x     && addgroup --system -…   63.6MB
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV PKG_RELEASE=1~buster     0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.4        0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.3     0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>           2 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:0dc53e7886c35bc21…   69.2MB
[klowenst@centos-7-1 ~]$
```

Now run the container

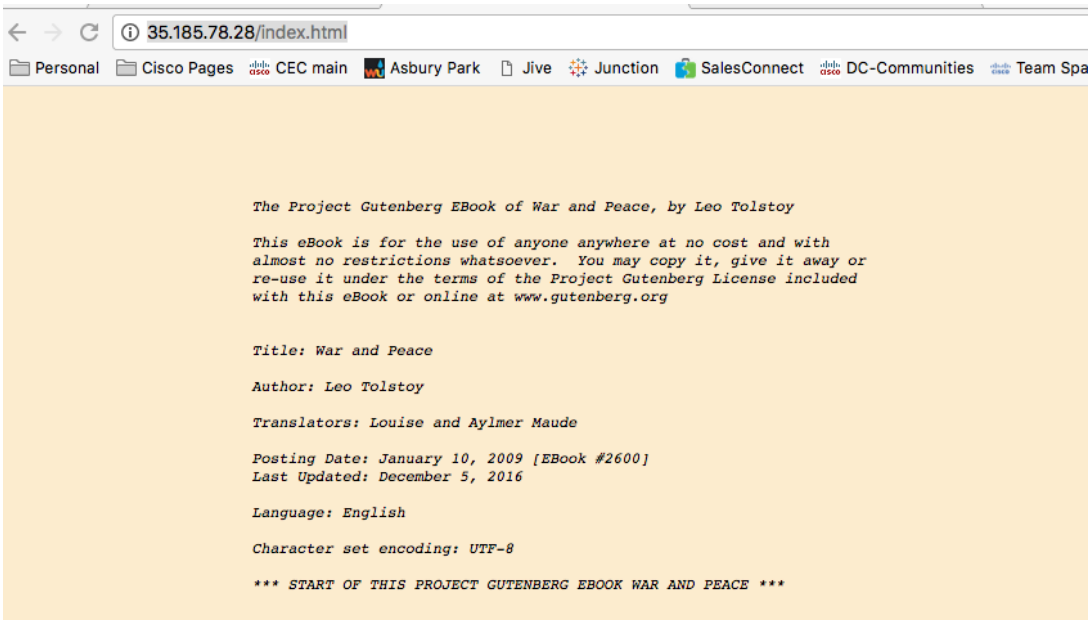
docker run -d -p 80:80 testweb

Retrieve the IP address of VM from GCP console (VM instances)



Verify web server is running by opening browser with this address - adding "/index.html" http://<your_external_ip>/index.html

NOTE: If you have issues viewing link – try another browser or clear the cache and try again.



type
docker ps
docker stop <container_ID>

STOP HERE

Lab Section C – Running Kubernetes

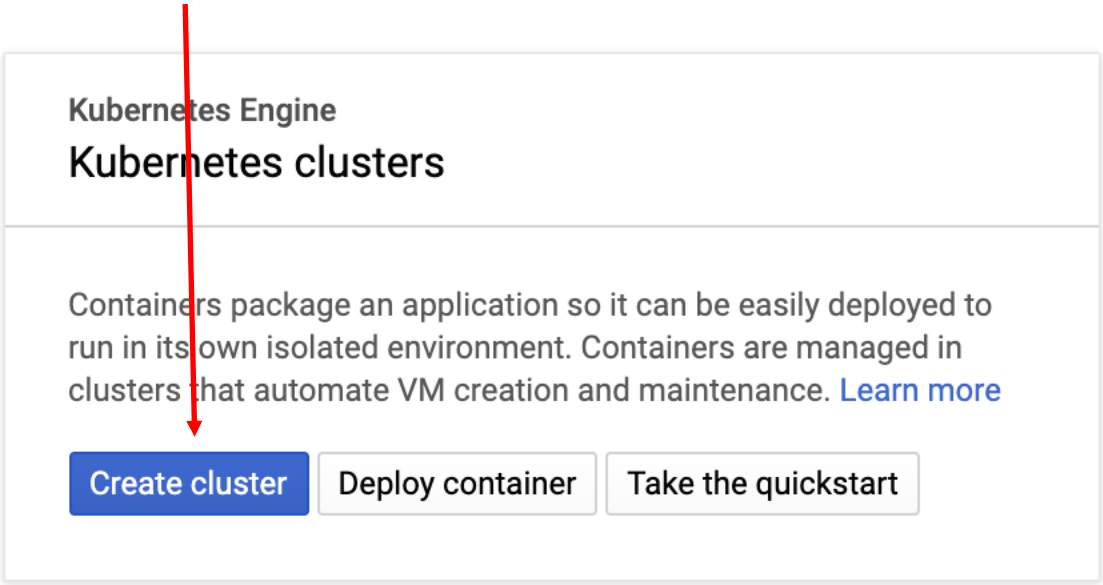
Building your Cluster

Within your project

> click on the 3 horizontal bars to the left of Google Could Platform 

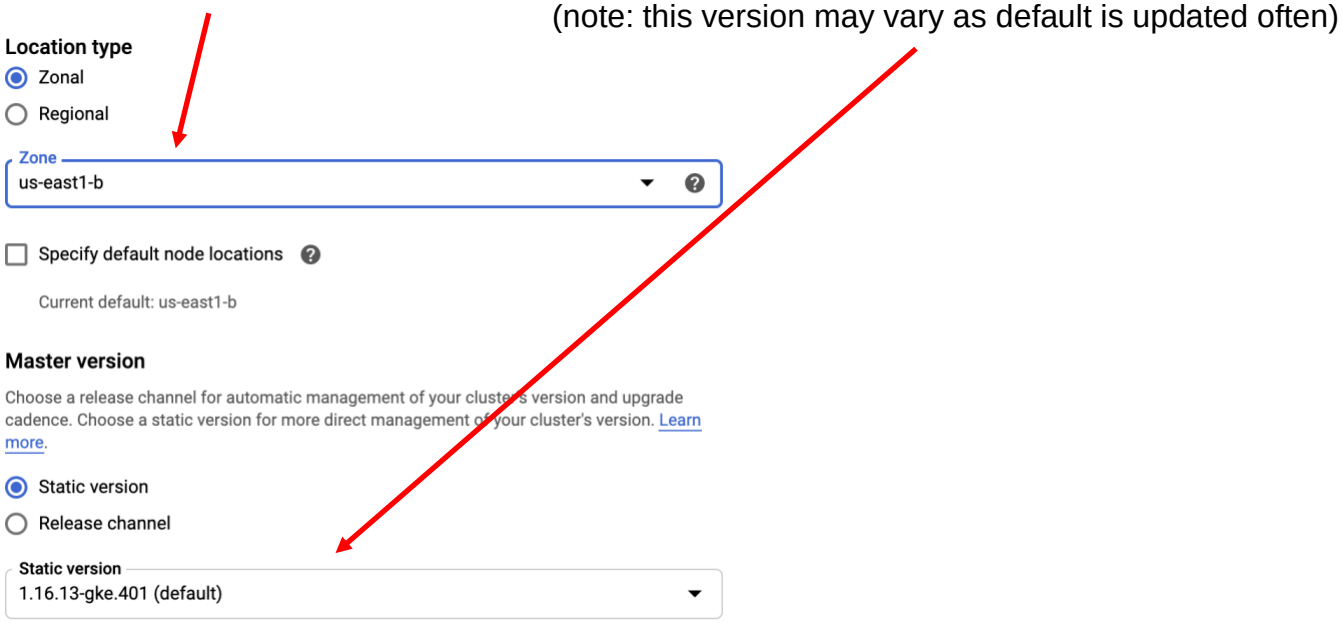
> click on Kubernetes Engine (wait for it to initialize – you may have to refresh screen)

> click “Create cluster”

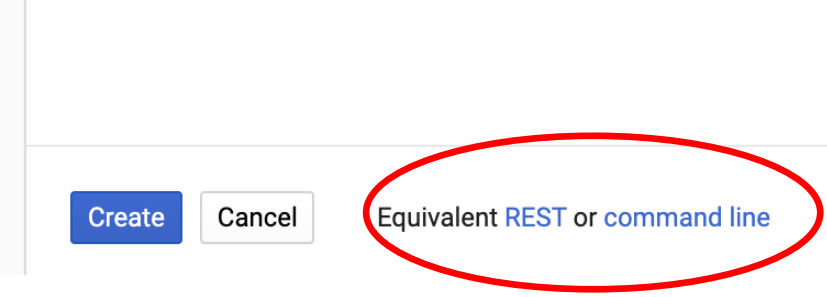


Change to **your** zone

Choose **1.16.13-gke.401 (default)** cluster version
(note: this version may vary as default is updated often)

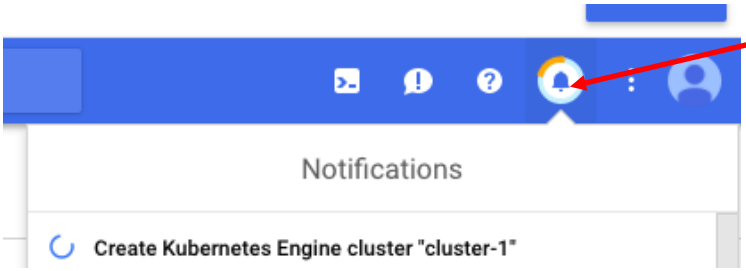


View REST and/or command line to view REST or CLI output

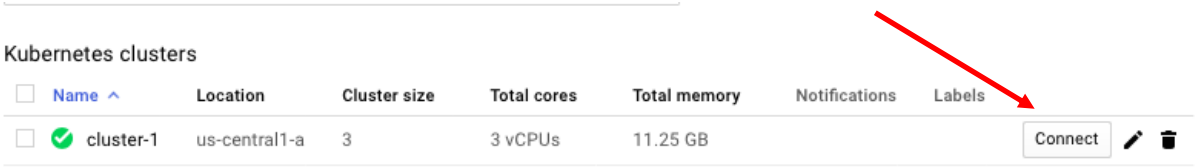


Click “Create”

You can always check the status (of anything) by clicking the Notifications icon

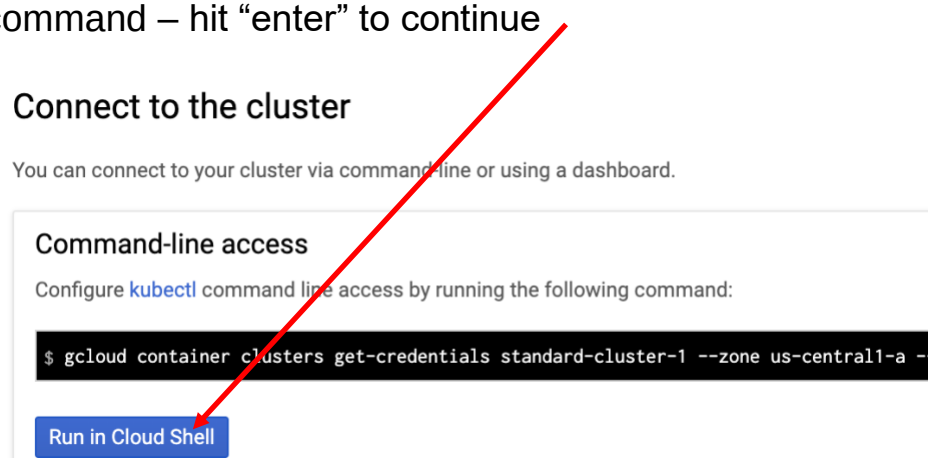


When cluster icon turns green - click connect



A dialog box will open –

Click “Run in Cloud Shell” – The cloud shell console will open and populate the command – hit “enter” to continue



Run the following commands to verify cluster is up and running

kubectl cluster-info

```
0 upgraded, 0 newly installed, 0 to remove and 17 not upgraded.
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl cluster-info
Kubernetes master is running at https://35.231.41.175
GLBCDefaultBackend is running at https://35.231.41.175/api/v1/namespaces/kube-system/services/default-http-backend:http/proxy
Heapster is running at https://35.231.41.175/api/v1/namespaces/kube-system/services/heapster/proxy
KubeDNS is running at https://35.231.41.175/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://35.231.41.175/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy
```

kubectl get nodes

```
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get nodes
NAME                                                STATUS   ROLES    AGE   VERSION
gke-standard-cluster-1-default-pool-9cb2ae4a-1j6z   Ready    <none>   23m   v1.13.11-gke.23
gke-standard-cluster-1-default-pool-9cb2ae4a-wsr6   Ready    <none>   23m   v1.13.11-gke.23
gke-standard-cluster-1-default-pool-9cb2ae4a-xngg   Ready    <none>   23m   v1.13.11-gke.23
kubefy2020@cloudshell:~ (marine-order-267121)$
```

kubectl get pods --all-namespaces

```
kube-system   fluentd-gcp-v3.2.0-m85hz                                    2/2     Running   0          24m
kube-system   heapster-7f47454698-dtgct                                   3/3     Running   0          24m
kube-system   kube-dns-79868f54c5-49nzm                                   4/4     Running   0          25m
kube-system   kube-dns-79868f54c5-zrpvq                                   4/4     Running   0          24m
kube-system   kube-dns-autoscaler-bb58c6784-xt2tj                         1/1     Running   0          24m
kube-system   kube-proxy-gke-standard-cluster-1-default-pool-9cb2ae4a-1j6z 1/1     Running   0          24m
kube-system   kube-proxy-gke-standard-cluster-1-default-pool-9cb2ae4a-wsr6 1/1     Running   0          24m
kube-system   kube-proxy-gke-standard-cluster-1-default-pool-9cb2ae4a-xngg 1/1     Running   0          24m
kube-system   l7-default-backend-fd59995cd-cjzms                          1/1     Running   0          25m
kube-system   metrics-server-v0.3.1-57c75779f-ljx2n                       2/2     Running   0          24m
kube-system   prometheus-to-sd-67fdw                                      2/2     Running   0          24m
kube-system   prometheus-to-sd-6j4gj                                      2/2     Running   0          24m
kube-system   prometheus-to-sd-vcvqg                                      2/2     Running   0          24m
kubefy2020@cloudshell:~ (marine-order-267121)$
```

Exposing an External IP Address to Access an Application in a Cluster

kubectl apply -f https://k8s.io/examples/service/load-balancer-example.yaml

- Create a Hello World application in your cluster

load-balancer-example.yaml (from hyperlink above)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/name: load-balancer-example
  name: hello-world
spec:
  replicas: 5
  selector:
    matchLabels:
      app.kubernetes.io/name: load-balancer-example
  template:
    metadata:
      labels:
        app.kubernetes.io/name: load-balancer-example
    spec:
      containers:
      - image: gcr.io/google-samples/node-hello:1.0
        name: hello-world
        ports:
        - containerPort: 8080
```

kubectl get deployments hello-world
kubectl describe deployments hello-world

- Display information about the Deployment

kubectl get replicasets
kubectl describe replicasets

- Display information about your ReplicaSet objects

kubectl expose deployment hello-world --type=LoadBalancer --name=my-service

- Create a Service object that exposes the deployment

kubectl get services my-service

- Display information about the Service

Output: (may take a minute or so to change from <pending> to external IP address)
You also have the option to append “watch” to above command. This will show you status of command every 2 seconds (CTRL-C to exit)

watch kubectl get services my-service

```
kubefy2020@cloudshell:~ (marine-order-267121)$
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get services my-service
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
my-service    LoadBalancer   10.12.2.215    35.202.225.104   8080:31935/TCP   60s
kubefy2020@cloudshell:~ (marine-order-267121)$
```

kubectl describe services my-service

- Display detailed information about the Service

Output:

```
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl describe services my-service
Name:                     my-service
Namespace:                default
Labels:                   app.kubernetes.io/name=load-balancer-example
Annotations:              <none>
Selector:                 app.kubernetes.io/name=load-balancer-example
Type:                     LoadBalancer
IP:                       10.12.2.215
LoadBalancer Ingress:     35.202.225.104
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  31935/TCP
Endpoints:                10.8.0.6:8080,10.8.1.4:8080,10.8.1.5:8080 + 2 more...
Session Affinity:         None
```

Make note of the external IP address exposed by your service (LoadBalancer Ingress). Also note the port value (8080)

You can see that the service has several endpoints: 10.8.0.6:8080,10.8.1.4:8080,10.8.1.5:8080 + 2 more. These are internal addresses of the pods that are running the Hello World application. To verify these are pod addresses, enter this command:

kubectl get pods --output=wide

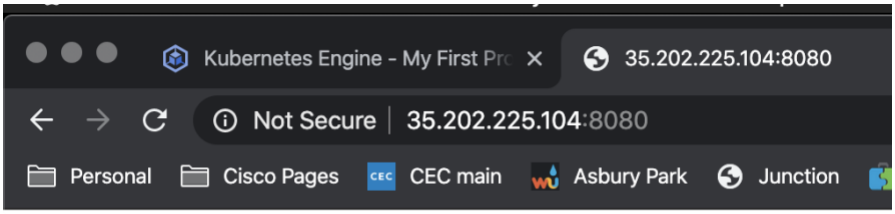
Output:

```
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get pods --output=wide
NAME                     READY   STATUS    RESTARTS   AGE    IP         NODE
hello-world-5885cc9fd8-54j6d   1/1     Running   0          6m9s   10.8.1.4   gke-standard-cluster-1-default-pool-ffa4414f-xj0s
hello-world-5885cc9fd8-97m4c   1/1     Running   0          6m9s   10.8.1.5   gke-standard-cluster-1-default-pool-ffa4414f-xj0s
hello-world-5885cc9fd8-cxn85   1/1     Running   0          6m9s   10.8.0.6   gke-standard-cluster-1-default-pool-ffa4414f-53fr
hello-world-5885cc9fd8-m8tr9   1/1     Running   0          6m9s   10.8.2.6   gke-standard-cluster-1-default-pool-ffa4414f-qnfz
hello-world-5885cc9fd8-w8g9n   1/1     Running   0          6m9s   10.8.2.7   gke-standard-cluster-1-default-pool-ffa4414f-qnfz
kubefy2020@cloudshell:~ (marine-order-267121)$
```

Open a browser and connect to: http://<external-ip>:<port>

- from kubectl get services my-service or kubectl describe services my-service

output should return “Hello Kubernetes !”

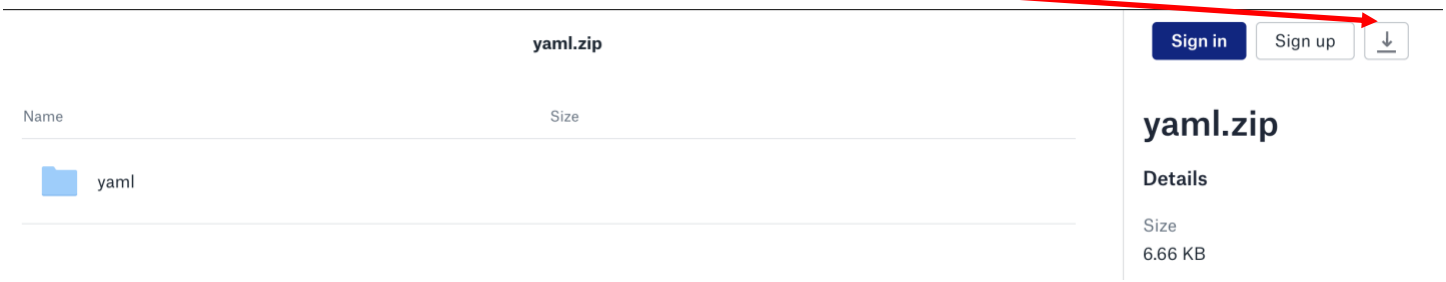


Create a Guestbook Kubernetes Engine cluster

We will demonstrate how to build a simple multi-tier web application (Guestbook with Redis and PHP) using Kubernetes Engine. The tutorial application is a guestbook that allows visitors to enter text in a log and to see the last few logged entries.

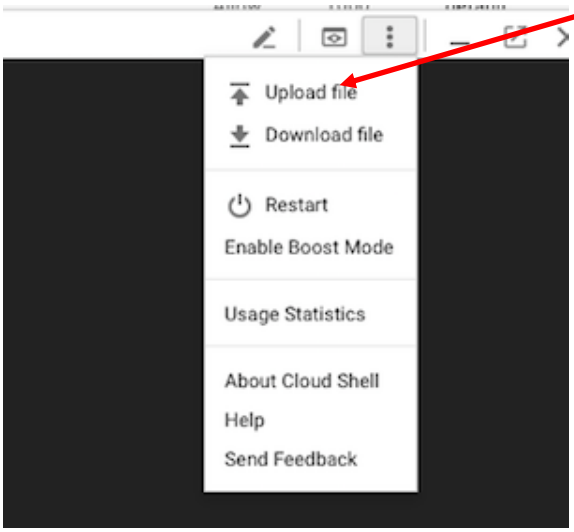
go to tiny.cc/ourlab

Download yaml.zip



Unzip the folder to your desktop

Upload yaml files to cloud shell (via cloud shell tool)



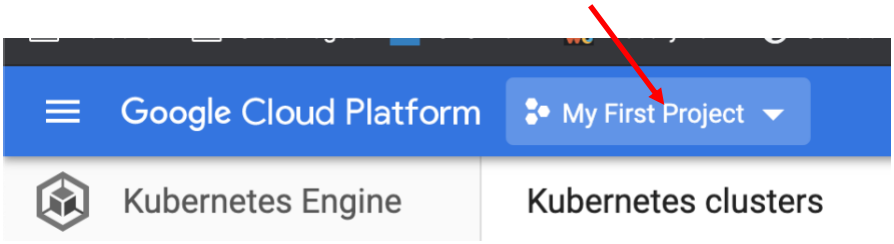
In Cloud Shell type
ls -l

- in console to verify yaml files - then continue with lab

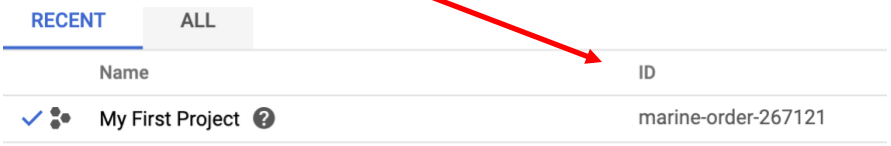
In your Cloud Shell console type:
gcloud config set project xxxxxxx

- where “xxxxxxx” is your project ID.

from GCP console click on Project Name

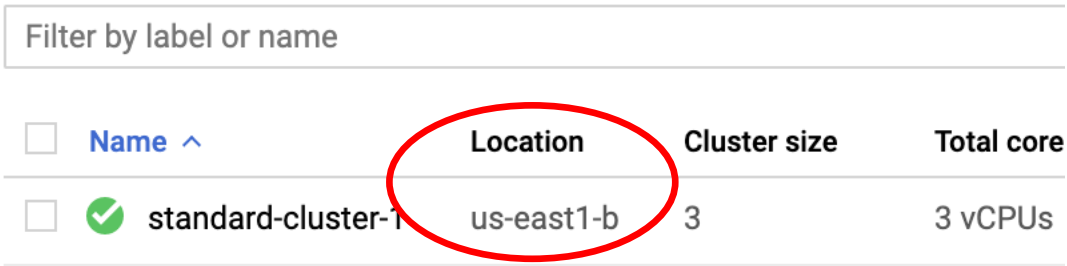


to view project ID



gcloud config set compute/zone xxxxxx

- where xxxxxx is your location (locate in GCP – Main Menu > Kubernetes Engine > location



Create the Kubernetes Engine via CLI

gcloud container clusters create guestbook --num-nodes=3

(disregard any warnings)
Note: You should now notice 2 clusters in the GCP dashboard

Name	Location	Cluster size	Total cores	Total memory
guestbook	us-east1-b	3	3 vCPUs	11.25 GB
standard-cluster-1	us-east1-b	3	3 vCPUs	11.25 GB

Get a list of clusters in your project or details for a single cluster (via CLI)

gcloud container clusters list
- should see both clusters

gcloud container clusters describe guestbook
- get information on the guestbook cluster

Step 1 – Set up a Redis Master

The guestbook application uses Redis to store its data. It writes its data to a Redis master instance and reads data from multiple Redis worker (slave) instances. The first step is to deploy a Redis master.

We will use the manifest yaml file named redis-master-deployment to deploy the Redis master. This manifest file specifies a Deployment controller that runs a single replica Redis master Pod:

redis-master-deployment.yaml

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: master
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: k8s.gcr.io/redis:e2e  # or just image: redis
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

kubectl create -f redis-master-deployment.yaml

- deploy the Redis Master

kubectl get pods

- get POD-NAME

```
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
redis-master-77bb4946d6-w27dz   1/1     Running   0          2m36s
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$
```

kubectl logs -f POD-NAME (copy POD-NAME from previous command)

- verify Redis master POD is running

```
[1] 28 Oct 15:03:28.214 # Server started, Redis version 2.8.19
[1] 28 Oct 15:03:28.214 # WARNING you have Transparent Huge Pages (THP) support enabled
eate latency and memory usage issues with Redis. To fix this issue run the command 'ech
sparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain
edis must be restarted after THP is disabled.
[1] 28 Oct 15:03:28.215 * The server is now ready to accept connections on port 6379
```

hit “CTRL-C” to break

Create redis-master service

The guestbook application needs to communicate to Redis master to write it's data. You need to create a Kubernetes Service to proxy the traffic to the Redis master Pod.

A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them. It is effectively a named load balancer that proxies traffic to one or more Pods. The Service will tell the Pods to proxy based on Pod labels.

The redis-master-service.yaml manifest file describing a Service resource for the Redis master

redis-master-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: redis
    role: master
    tier: backend
spec:
  ports:
  - name: redis
    port: 6379
    targetPort: 6379
  selector:
    app: redis
    role: master
    tier: backend
```

This manifest file creates a Service named redis-master with a set of label selectors. These labels match the set of labels that are deployed in the previous step. Therefore, this Service routes the network traffic to the Redis master Pod created in Step 1.

The ports section of the manifest declares a single port mapping. In this case, the Service will route the traffic on port: 6379 to the targetPort: 6379 of the containers that match the specified selector labels. Note that the containerPort used in the Deployment must match the targetPort to route traffic to the Deployment.

kubectl apply -f redis-master-service.yaml

- Start the Redis master's Service

kubectl get service

- Verify the service is created

```
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$ kubectl get service
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.79.240.1    <none>        443/TCP    10m
redis-master   ClusterIP   10.79.243.39   <none>        6379/TCP   73s
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$
```

Step 2 – Set up Redis workers

Although the Redis master is a single pod, you can make it highly available and meet traffic demands by adding a few Redis worker replicas.

Take a look at the redis-slave-deployment.yaml manifest file describing a Deployment for the Redis worker pods

examples/guestbook/redis-slave-deployment.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-slave
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access an environment variable to find the master
          # service's host, comment out the 'value: dns' line above, and
          # uncomment the line below:
          # value: env
        ports:
        - containerPort: 6379
```

Deployment is responsible for achieving the configuration declared in the manifest file. For example, this manifest file defines two replicas for the Redis workers. If there are not any replicas running, Deployment would start the two replicas on your container cluster. If there are more than two replicas are running, it would kill some to meet the specified configuration.

In this case, the Deployment object specifies two replicas. To create the Redis worker Deployment, run:

kubectl apply -f redis-slave-deployment.yaml

Verify that the two Redis worker replicas are running by querying the list of Pods:

kubectl get pods

```
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
redis-master-77bb4946d6-w27dz   1/1     Running   0          9m11s
redis-slave-84548fdbc-9ffq9     1/1     Running   0          98s
redis-slave-84548fdbc-pq6fq     1/1     Running   0          98s
```

Create redis-slave service

The guestbook application needs to communicate to Redis workers to read data. To make the Redis workers discoverable, you need to set up a Service. A Service provides transparent load balancing to a set of Pods.

The redis-slave-service.yaml defines the Service configuration for the Redis workers

redis-slave-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
  labels:
    app: redis
    role: slave
    tier: backend
spec:
  ports:
  - port: 6379
  selector:
    app: redis
    role: slave
    tier: backend
```

This file defines a Service named redis-slave running on port 6379. Note that the selector field of the Service matches the Redis worker Pods created in the previous step.

Create the redis-slave Service by running:

kubectl create -f redis-slave-service.yaml

kubectl get service

- verify service is created

```
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$ kubectl get service
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.79.240.1    <none>        443/TCP    12m
redis-master   ClusterIP   10.79.243.39   <none>        6379/TCP   2m58s
redis-slave    ClusterIP   10.79.240.54   <none>        6379/TCP   8s
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$
```

Step 3: Set up the guestbook web frontend

Now that the Redis storage of your guestbook up, start the guestbook web servers. Like the Redis workers, this is a replicated application managed by a Deployment. This tutorial uses a simple PHP frontend. It is configured to talk to either the Redis worker or master Services, depending on whether the request is a read or a write. It exposes a simple JSON interface, and serves a jQuery-Ajax-based UX.
Take a look at the frontend-deployment.yaml manifest file describing the Deployment for the guestbook web server

frontend-deployment.yaml

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: frontend
  labels:
    app: guestbook
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

kubectl apply -f frontend-deployment.yaml

- create guestbook web frontend Deployment

kubectl get pods -l app=guestbook -l tier=frontend

- verify the three replicas are running by querying the list of the labels that identify the web frontend (1 of 3 shown running – 2 still being created)

```
NAME                        READY   STATUS              RESTARTS   AGE
frontend-678d98b8f7-5t8gm   0/1     ContainerCreating   0          27s
frontend-678d98b8f7-7zmxf   1/1     Running             0          27s
frontend-678d98b8f7-nx984   0/1     ContainerCreating   0          27s
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$
```

Expose frontend on an external IP address

The redis-slave and redis-master Services you created in the previous steps are only accessible within the container cluster, because the default type for a Service is ClusterIP.
ClusterIP provides a single IP address for the set of Pods the Service is pointing to. This IP address is accessible only within the cluster.
However, you need the guestbook web frontend Service to be externally visible. That is, you want a client to be able to request the Service from outside the container cluster. To accomplish this, you need to specify type: LoadBalancer in the Service configuration. The frontend-service.yaml manifest file specifying this configuration looks like this

frontend-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

kubectl create -f frontend-service.yaml

- create the frontend service

Step 4: Visit the guestbook website

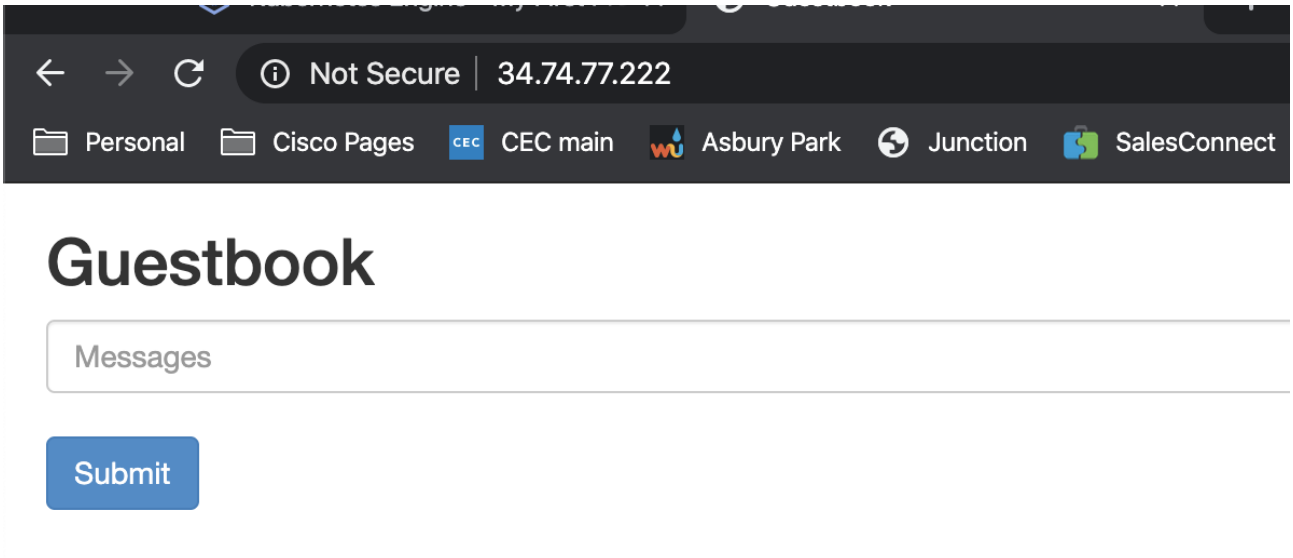
To access the guestbook Service, you need to find the external IP of the Service you just set up by running the command (this may take a minute or so to populate external IP address)

kubectl get service frontend

- find the external IP of the Service you just set up (will take a little while)

```
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$ kubectl get service frontend
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
frontend   LoadBalancer   10.79.242.49   35.185.22.184    80:30343/TCP   47s
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$
```

Copy the IP address in EXTERNAL-IP column, and load the page in your browser:



type:
kubectl get pods

- you should see the 3 frontend PODs we created earlier

```
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
frontend-74b4665db5-25w9v         1/1     Running   0          3m28s
frontend-74b4665db5-ptmpn         1/1     Running   0          3m28s
frontend-74b4665db5-qcvkc         1/1     Running   0          3m28s
redis-master-74474c9cd-v66xl      1/1     Running   0          8m22s
redis-slave-74ccb764fc-4xcsv      1/1     Running   0          5m5s
redis-slave-74ccb764fc-cxbp7      1/1     Running   0          5m5s
kubefy2020@cloudshell:~ (marine-order-267121)$
```

Let's scale this up

A ReplicaSet ensures that a specified number of pod replicas are running at any given time. However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to pods along with a lot of other useful features.
If you look at the 'frontend-deployment.yaml' file (3 pages back) you will see we defined 3 replicas in our deployment file.
It is recommended using Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates at all.

For the sake of this lab we will increase the 3 replicas we specified to 5 using the kubectl command.

kubectl scale deployment frontend --replicas=5

- scale up the number of PODs to 5

kubectl get pods

You should now see 5 PODs

```
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
frontend-678d98b8f7-5t8gm   1/1     Running   0          27m
frontend-678d98b8f7-7zmxf   1/1     Running   0          27m
frontend-678d98b8f7-95vq5   1/1     Running   0          21m
frontend-678d98b8f7-g99gj   1/1     Running   0          21m
frontend-678d98b8f7-nx984   1/1     Running   0          27m
redis-master-77bb4946d6-w27dz   1/1     Running   0          38m
redis-slave-84548fdbc-9ffq9     1/1     Running   0          30m
redis-slave-84548fdbc-pq6fq     1/1     Running   0          30m
klowenst@cloudshell:~ (gcp-klowenst041320-nprd-68609)$
```

Let's Cleanup

kubectl delete service frontend

- delete the service

gcloud compute forwarding-rules list

- verify deleted

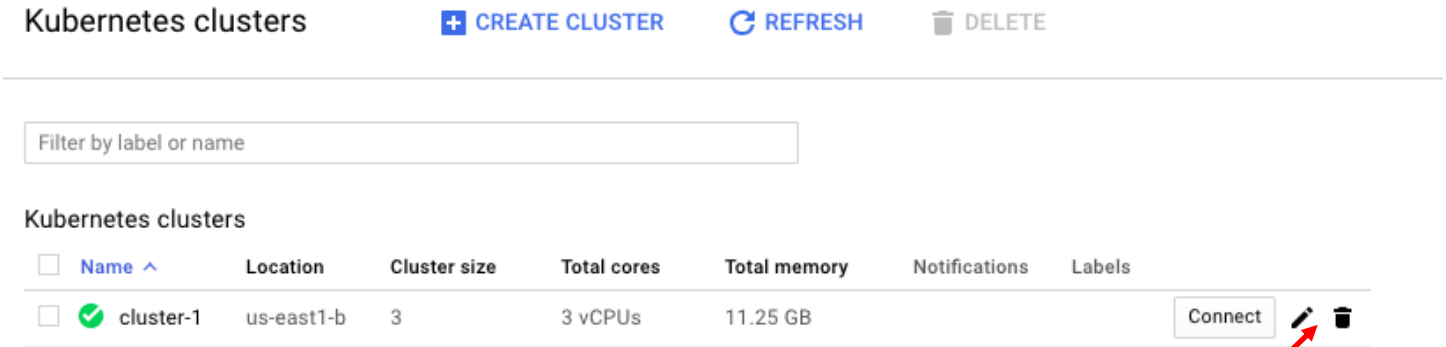
```
kubeouser@k8s-cisco-lab1:~$ gcloud compute forwarding-rules list
NAME                             REGION       IP_ADDRESS       IP_PROTOCOL  TARGET
a0a7bce70f21611e7b13c42010a8e01b  us-east1     104.196.185.172  TCP          us-east1/targetPools/a0a7bce70f21611e7b13c42010a8e01b
a302e292ff3d711e79cc942010a8e006  us-east1     35.196.99.109    TCP          us-east1/targetPools/a302e292ff3d711e79cc942010a8e006
a5ffa3f17f21611e7b13c42010a8e01b  us-east1     35.190.140.202   TCP          us-east1/targetPools/a5ffa3f17f21611e7b13c42010a8e01b
a5b4f32bef17a11e79c1b42010a80009  us-central1  35.226.129.173   TCP          us-central1/targetPools/a5b4f32bef17a11e79c1b42010a80009
a71b31f22f17811e79c1b42010a80009  us-central1  35.225.194.254   TCP          us-central1/targetPools/a71b31f22f17811e79c1b42010a80009
kubeouser@k8s-cisco-lab1:~$ gcloud compute forwarding-rules list
NAME                             REGION       IP_ADDRESS       IP_PROTOCOL  TARGET
```

gcloud container clusters delete guestbook

- delete the container cluster

When CLI commands are finished running you will see guestbook cluster deleted from GCP dash

Now let's delete our original cluster



Return to GCP > Compute Engine - delete Centos 7 VM

- - - DONE - - -

If you're interested - more k8s hands-on…

Go to Cisco devnet site:
https://devnetsandbox.cisco.com/RM/Topology?c=0a71d70b-8fa3-4bcb-94dd-6c379dd11c24