

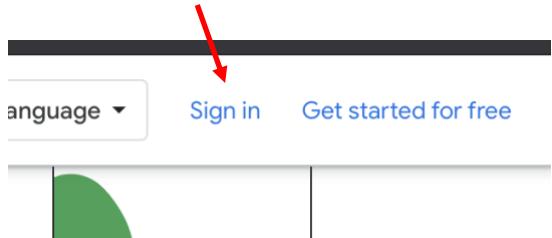
# Intro to DevOps, Containers & Orchestration



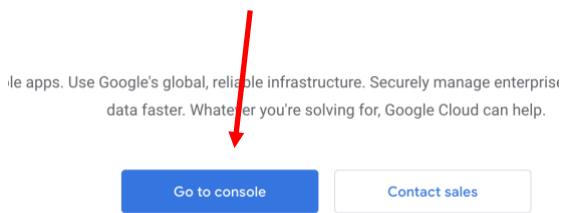
# Lab Section A – Access Google Cloud Platform

Login to GCP - <https://cloud.google.com>

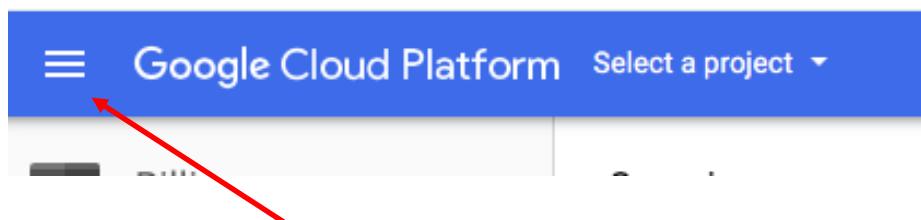
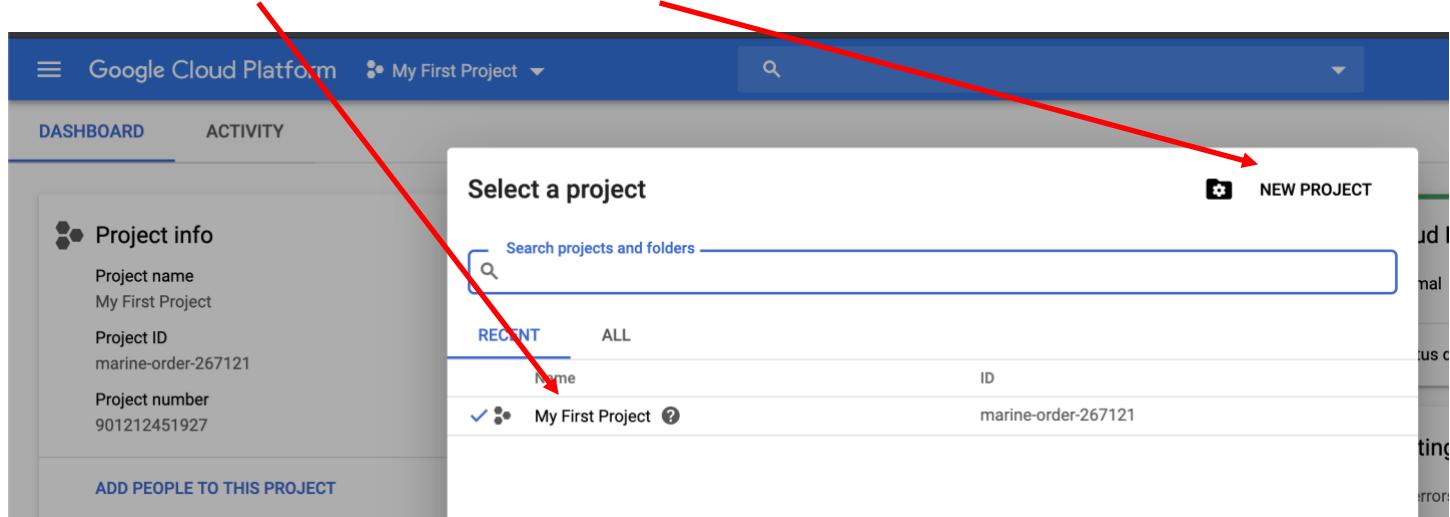
Click “Sign In” button



Click Go to console



Select an existing project or create a new one:

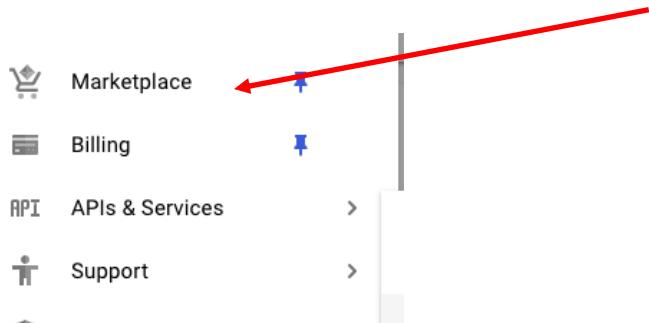


Remember this ‘hamburger’ icon – it will show the main menu in GCP

**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”

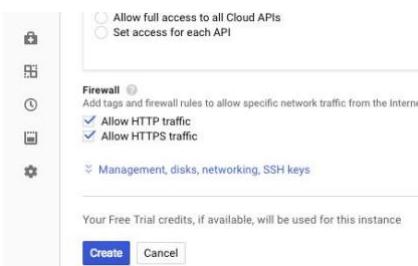


type “Centos” in the search bar and double click “Centos 7”



Click “Launch on Compute Engine”

Accept all defaults **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 these options)



Now click create

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

VM Instances					
<input type="checkbox"/>	Name ^	Zone	Recommendation	Internal IP	External IP
<input checked="" type="checkbox"/>	centos-7-1	us-east1-b		10.142.0.2	104.196.105.65 



**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 something similar to the following –

```
[kubefy2020@centos-7-1 ~]$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
  Active: active (running) since Tue 2020-02-04 14:36:37 UTC; 9min ago
    Docs: https://docs.docker.com
 Main PID: 1717 (dockerd)
   Tasks: 8
  Memory: 39.9M
 CGroup: /system.slice/docker.service
         └─1717 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

*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

```
[kubefy2020@centos-7-1 ~]$ docker info
Client:
  Debug Mode: false

Server:
  Containers: 0
    Running: 0
    Paused: 0
    Stopped: 0
  Images: 0
  Server Version: 19.03.5
  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-file
```

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

```
[kubefy2020@centos-7-1 ~]$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
5c939e3a4d10: Pull complete
c63719cdbe7a: Pull complete
19a861ea6baf: Pull complete
651c9d2d6c4f: Pull complete
Digest: sha256:8d31dad0c58f552e890d68bbfb735588b6b820a46e459672d96e585871acc110
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Docker is pulling the layered image

type `docker images` command again

```
[kubefy2020@centos-7-1 ~]$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
ubuntu          latest   ccc6e87d482b   3 weeks ago  64.2MB
[kubefy2020@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 ccc6e87d482b`

## Examine the output

```
        },
        "Name": "overlay2"
    },
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:43c67172d1d182ca5460fc962f8f053f33028e0a3a1d423e05d91b532429e73d",
            "sha256:21ec61b65b20ec53a1b7f069fd04df5acb0e75434bd3603c88467c8bfc80d9c6",
            "sha256:1d0dfb259f6a31f95efcba61f0a3afa318448890610c7d9a64dc4e95f9add843",
            "sha256:f55aa0bd26b801374773c103bed4479865d0e37435b848cb39d164ccb2c3ba51"
        ]
    },
    "Metadata": {
        "LastTagTime": "0001-01-01T00:00:00Z"
    }
}
]
[kubefy2020@centos-7-1 ~]$ docker inspect ccc6e87d482b
```

Now type

```
ps -aux
```

Notice all the processes currently running on the linux OS

```
root      1006  0.0  0.2 218548  7540 ?          Ssl  14:30  0:00 /usr/sbin/rsyslogd -n
root      1248  0.0  0.3 214788 12700 ?          Ss   14:30  0:00 /usr/bin/python /usr/bin/google_network_daemon
root      1250  0.0  0.3 217320 13156 ?          Ss   14:30  0:00 /usr/bin/python /usr/bin/google_accounts_daemon
root      1254  0.0  0.3 214776 12624 ?          Ss   14:30  0:00 /usr/bin/python /usr/bin/google_clock_skew_daemon
root      1256  0.0  0.1 112920  4344 ?          Ss   14:30  0:00 /usr/sbin/sshd -D
root      1313  0.0  0.0 89700  2084 ?          Ss   14:30  0:00 /usr/libexec/postfix/master -w
postfix   1316  0.0  0.1 89804  4056 ?          S   14:30  0:00 pickup -l -t unix -u
postfix   1317  0.0  0.1 89872  4080 ?          S   14:30  0:00 qmgr -l -t unix -u
root      1716  0.0  1.0 489976 37764 ?          Ssl  14:36  0:00 /usr/bin/containerd
root      1717  0.1  2.4 570244 88212 ?          Ssl  14:36  0:02 /usr/bin/dockerd -H fd:// --containerd=/run/contai
root      1904  0.0  0.1 158920  5756 ?          Ss   14:37  0:00 sshd: kubefy2020 [priv]
kubefy2+  1907  0.0  0.0 158920  2436 ?          S   14:37  0:00 sshd: kubefy2020@pts/0
kubefy2+  1908  0.0  0.1 117072  3708 pts/0       Ss   14:37  0:00 /bin/bash -l
kubefy2+  1909  0.0  0.0 72248  2676 ?          Ss   14:37  0:00 /usr/libexec/openssh/sftp-server
root      2054  0.0  0.0      0     0 ?          S   14:45  0:00 [kworker/0:2]
root      2139  0.0  0.0      0     0 ?          R   14:50  0:00 [kworker/0:1]
root      2362  0.0  0.0      0     0 ?          S   14:55  0:00 [kworker/0:0]
root      2448  0.0  0.1 112920  4284 ?          Ss   14:58  0:00 sshd: [accepted]
kubefy2+  2449  0.0  0.0 155372  1864 pts/0       R+  14:58  0:00 ps -aux
[kubefy2020@centos-7-1 ~]$
```

## Creating the first container

type

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

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

again type

```
ps -aux
```

```
[kubefy2020@centos-7-1 ~]$ docker run -it ubuntu
root@df9c25bde234:/# ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0  18496  2020 pts/0    Ss   14:59   0:00 /bin/bash
root         10  0.0  0.0  34388  1472 pts/0    R+   15:00   0:00 ps -aux
root@df9c25bde234:/# █
```

Only 2 processes are running in this container – “/bin/bash” & the command you just entered, “ps aux”

Let's exit the container but leave it running

Press and hold “CTRL” key then “p-q”

You should now be back at linux prompt

type:

```
docker ps
```

```
[kubefy2020@centos-7-1 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
df9c25bde234        ubuntu              "/bin/bash"        About a minute ago   Up About a minute
epic_shtern
[kubefy2020@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

```
[root@centos-7-1 chuckduffy]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
[root@centos-7-1 chuckduffy]#
```

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)

```
[kubefy2020@centos-7-1 ~]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS
CRTS              NAMES
46b745dc809e      ubuntu              "/bin/bash"
[kubefy2020@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

```
[kubefy2020@centos-7-1 ~]$ docker images
REPOSITORY          TAG           IMAGE ID            CREATED          SIZE
ubuntu_traceroute  latest        de9c41794641    38 seconds ago  93.1MB
ubuntu              latest        ccc6e87d482b    3 weeks ago     64.2MB
[kubefy2020@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 4 layers

```
"Name": "overlay2",
},
"RootFS": {
  "Type": "layers",
  "Layers": [
    "sha256:43c67172d1d182ca5460fc962f8f053f33028e0a3a1d423e05d91b532429e73d",
    "sha256:21ec61b65b20ec53a1b7f069fd04df5acb0e75434bd3603c88467c8bfc80d9c6",
    "sha256:1d0dfb259f6a31f95efcba61f0a3afa318448890610c7d9a64dc4e95f9add843",
    "sha256:f55aa0bd26b801374773c103bed4479865d0e37435b848cb39d164ccb2c3ba51"
]
```

Type `docker inspect ubuntu_traceroute`

We now see the original 4 layers plus the layer we added

```
        "Name": "overlay2"
    },
    "RootFS": {
        "Type": "layers",
        "Layers": [
            "sha256:43c67172d1d182ca5460fc962f8f053f33028e0a3a1d423e05d91b532429e73d",
            "sha256:21ec61b65b20ec53a1b7f069fd04df5acb0e75434bd3603c88467c8bfc80d9c6",
            "sha256:1d0dfb259f6a31f95efcba61f0a3afa318448890610c7d9a64dc4e95f9add843",
            "sha256:f55aa0bd26b801374773c103bed4479865d0e37435b848cb39d164ccb2c3ba51",
            "sha256:d55bc1197085dbe0cb5f8e3d315eaf8dcf3c9a1eec198e9a7eef399dfd78f4a1"
        ]
    }
```

## 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.

VM instances		 CREATE INSTANCE	 IMPORT VM	 R
Filter VM instances				
<input type="checkbox"/>	Name	Zone	Recommendation	Internal IP
<input checked="" type="checkbox"/>	centos-7-1	us-east1-c		10.142.0.2

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

The NGINX default web page should show.



## Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](#)  
Commercial support is available at [nginx.com](#)

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

```
[kubefy2020@centos-7-1 ~]$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
testweb              latest   7f86d4da09de  9 seconds ago  131MB
nginx               latest   2073e0bcb60e  2 days ago   127MB
ubuntu              latest   ccc6e87d482b  2 weeks ago   64.2MB
```

type

*docker history testweb*

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

```
[kubefy2020@centos-7-1 ~]$ docker history testweb
IMAGE          CREATED      CREATED BY
MENT
0697bee8272d  7 minutes ago /bin/sh -c #(nop) COPY file:1bd25f1a7f80d235...
6678c7c2e56c   3 weeks ago  /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon...
<missing>       3 weeks ago  /bin/sh -c #(nop) STOPSIGNAL SIGTERM
<missing>       3 weeks ago  /bin/sh -c #(nop) EXPOSE 80
<missing>       3 weeks ago  /bin/sh -c ln -sf /dev/stdout /var/log/nginx...
<missing>       3 weeks ago  /bin/sh -c set -x && addgroup --system ...
<missing>       3 weeks ago  /bin/sh -c #(nop) ENV PKG_RELEASE=1~buster
<missing>       3 weeks ago  /bin/sh -c #(nop) ENV NJS_VERSION=0.3.9
<missing>       3 weeks ago  /bin/sh -c #(nop) ENV NGINX_VERSION=1.17.9
<missing>       4 weeks ago  /bin/sh -c #(nop) LABEL maintainer=NGINX Do...
<missing>       4 weeks ago  /bin/sh -c #(nop) CMD ["bash"]
<missing>       4 weeks ago  /bin/sh -c #(nop) ADD file:e5a364615e0f69616...
[kubefy2020@centos-7-1 ~]$
```

Type *docker history nginx* to view original image we copied from

- notice there is one less “layer”

```
See docker -help
[kubefy2020@centos-7-1 ~]$ docker history nginx
IMAGE          CREATED      CREATED BY
MENT
6678c7c2e56c   3 weeks ago  /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon...
<missing>       3 weeks ago  /bin/sh -c #(nop) STOPSIGNAL SIGTERM
<missing>       3 weeks ago  /bin/sh -c #(nop) EXPOSE 80
<missing>       3 weeks ago  /bin/sh -c ln -sf /dev/stdout /var/log/nginx...
<missing>       3 weeks ago  /bin/sh -c set -x && addgroup --system ...
<missing>       3 weeks ago  /bin/sh -c #(nop) ENV PKG_RELEASE=1~buster
<missing>       3 weeks ago  /bin/sh -c #(nop) ENV NJS_VERSION=0.3.9
<missing>       3 weeks ago  /bin/sh -c #(nop) ENV NGINX_VERSION=1.17.9
<missing>       4 weeks ago  /bin/sh -c #(nop) LABEL maintainer=NGINX Do...
<missing>       4 weeks ago  /bin/sh -c #(nop) CMD ["bash"]
<missing>       4 weeks ago  /bin/sh -c #(nop) ADD file:e5a364615e0f69616...
[kubefy2020@centos-7-1 ~]$
```

Now run the container

*docker run -d -p 80:80 testweb*

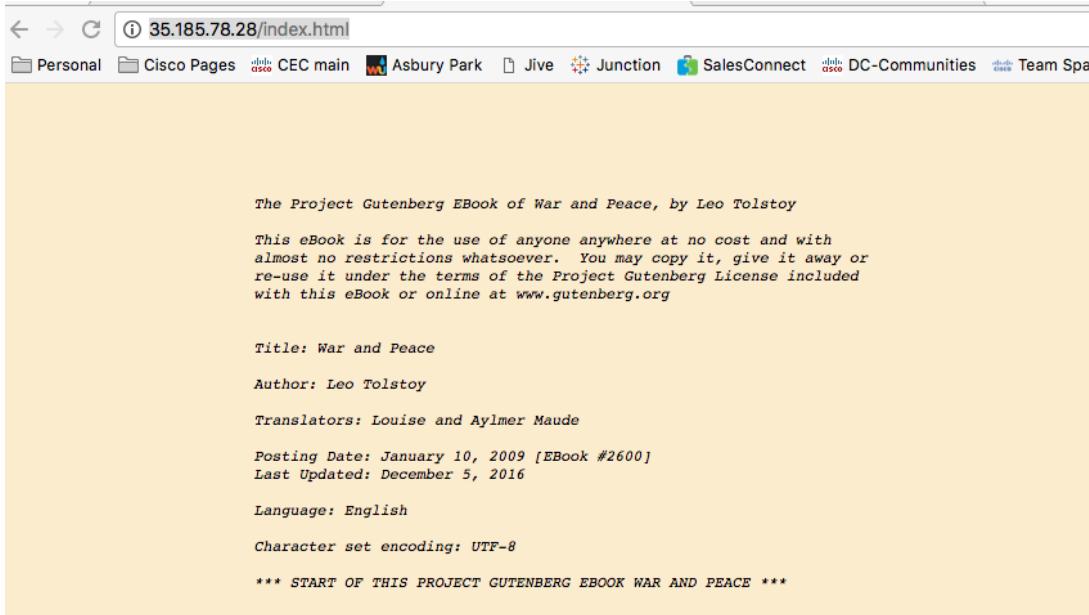
Retrieve the IP address of VM from GCP console (VM instances)

The screenshot shows the Google Cloud Platform Compute Engine interface. On the left, a sidebar menu includes 'Compute Engine', 'VM instances' (which is selected and highlighted in blue), 'Instance groups', 'Instance templates', 'Disks', and 'Snapshots'. The main content area is titled 'VM instances' and features a 'CREATE INSTANCE' button and an 'IMPORT VM' button. Below these are buttons for 'Filter VM instances' and 'External IP'. A table lists the VM instances, showing columns for Name, Zone, Recommendation, Internal IP, and External IP. The 'centos-7-1' instance is selected, and a red arrow points to its External IP value, which is 35.185.78.28.

Name	Zone	Recommendation	Internal IP	External IP
centos-7-1	us-east1-b		10.142.0.2	35.185.78.28

Verify web server is running by opening browser with this address - adding  
"/index.html"      [http://<your\\_external\\_ip>/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.**



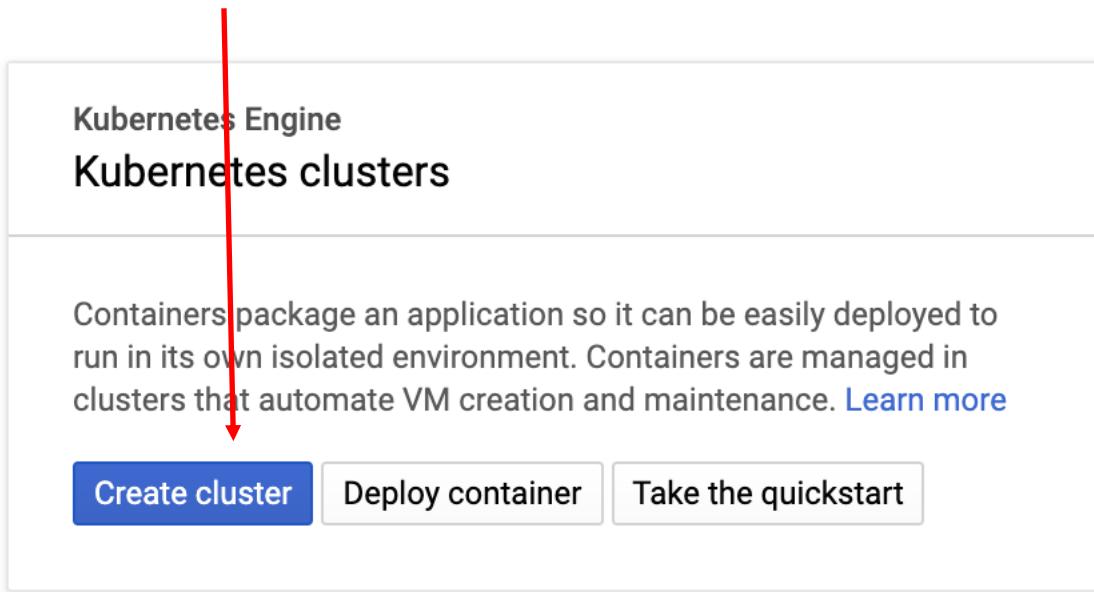
```
type
docker ps
docker stop <container_ID>
```

# 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 an **eastern** zone

Zone  
us-east1-b

Specify node locations [?](#)

Choose **1.14.10-gke.27 (default)** cluster version  
(note: this version may vary as default is updated often)

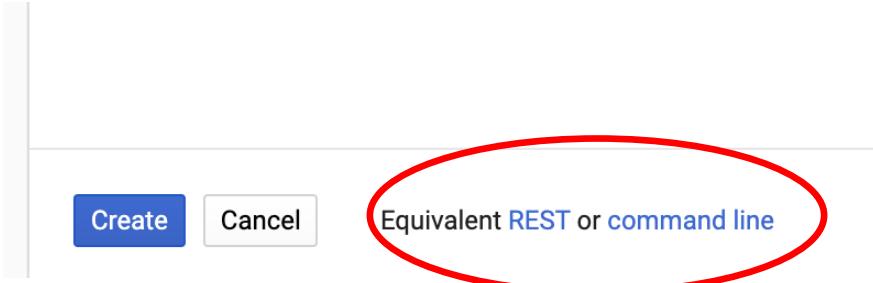
### Master version

Choose Release Channel to get automatic GKE upgrades as new versions are ready. Choose a static version to upgrade manually in the future. [Learn more](#)

Release channel  
 Static version

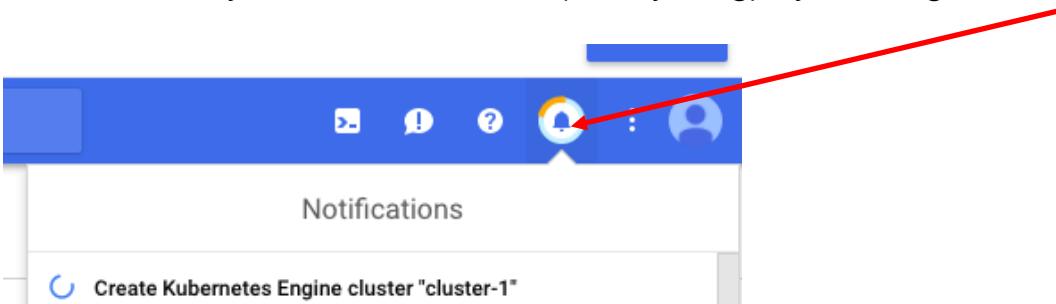
Static version  
1.14.10-gke.27 (default)

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

Name	Location	Cluster size	Total cores	Total memory	Notifications	Labels	
<input checked="" type="checkbox"/> cluster-1	us-central1-a	3	3 vCPUs	11.25 GB			<button>Connect</button>

A dialog box will open –

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

Connect to the cluster

You can connect to your cluster via command line or using a dashboard.

#### Command-line access

Configure `kubectl` command line access by running the following command:

```
$ gcloud container clusters get-credentials standard-cluster-1 --zone us-central1-a -
```

[Run in Cloud Shell](#)

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*

NAME	READY	STATUS	ROLES	AGE	VERSION
kube-system/fluentd-gcp-v3.2.0-jpv1d	2/2	Running	0	24m	
kube-system/fluentd-gcp-v3.2.0-m85hz	3/3	Running	0	24m	
kube-system/heapster-7f47454698-dtgct	4/4	Running	0	25m	
kube-system/kube-dns-79868f54c5-49nzm	4/4	Running	0	24m	
kube-system/kube-dns-79868f54c5-zrpvq	1/1	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-1jx2n	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

`service/load-balancer-example.yaml`

```
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 replicaset`

`kubectl describe replicaset`

- 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)

```
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:

```
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
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`

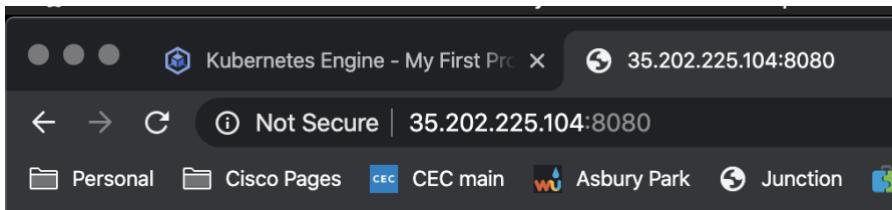
### Output:

```
kubefy2020@cloudshell:~ (marine-order-267121)$ 
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](http://tiny.cc/ourlab)

Download yaml.zip

yaml

yaml.zip

Yesterday by Ken Lowenstein... 6 Files

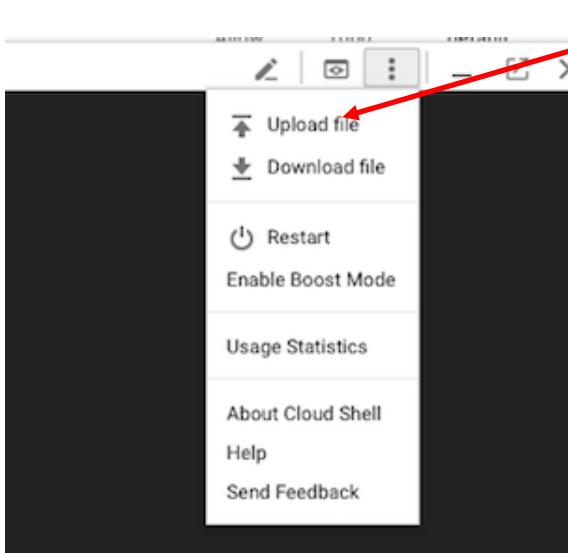
Today by Ken Lowenstein 6.7 KB

...

- Open with The Unarchiver
- Share
- Upload New Version
- Download
- Favorite

Unzip the folder to your desktop

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



type

/s

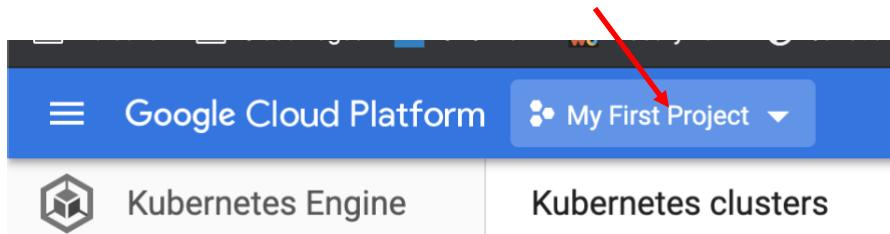
- in console to verify yaml files - then continue with lab

In your Cloud Shell console type:

`gcloud config set project xxxxxxxx`

- where “xxxxxxxx” is your project ID.

from GCP console click on Project Name



to view project ID

Name	ID
My First Project	marine-order-267121

`gcloud config set compute/zone xxxxxxxx`

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

Name	Location	Cluster size	Total core
standard-cluster-1	us-east1-b	3	3 vCPUs

## Create the Kubernetes Engine via CLI

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

Note: You should now notice 2 clusters in the GCP dashboard

<input type="checkbox"/>	Name ^	Location	Cluster size	Total cores	Total memory
<input type="checkbox"/>	guestbook	us-east1-b	3	3 vCPUs	11.25 GB
<input type="checkbox"/>	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 list 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:

```
examples/guestbook/redis-master-deployment.yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: redis-master
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
        - name: master
          image: gcr.io/google_containers/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
```

- verify Redis master POD is running

```
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get pods  
NAME                      READY   STATUS    RESTARTS   AGE  
redis-master-74474c9cd-v66xl   1/1     Running   0          34s  
kubefy2020@cloudshell:~ (marine-order-267121)$
```

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

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

```
examples/guestbook/redis-master-service.yaml
```

```
apiVersion: v1  
kind: Service  
metadata:  
  name: redis-master  
  labels:  
    app: redis  
    role: master  
    tier: backend  
spec:  
  ports:  
  - 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 create -f redis-master-service.yaml
```

- Start the Redis master's Service

```
kubectl get service
```

- Verify the service is created

```
SERVICE          NAME        CLUSTER-IP      EXTERNAL-IP    PORT(S)      AGE
kubernetes       kubernetes   10.15.240.1    <none>        443/TCP     4m28s
redis-master     redis-master 10.15.251.61   <none>        6379/TCP    7s
kubefy2020@cloudshell:~ (marine-order-267121)$ █
```

## 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
```

```
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 create -f redis-slave-deployment.yaml
```

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

```
kubectl get pods
```

```
deployment.extensions/redis-slave created
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
redis-master-74474c9cd-v66xl   1/1     Running   0          3m27s
redis-slave-74ccb764fc-4xcsv   1/1     Running   0          10s
redis-slave-74ccb764fc-cxbp7   1/1     Running   0          10s
kubefy2020@cloudshell:~ (marine-order-267121)$ █
```

## 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

```
examples/guestbook/redis-slave-service.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

```
service/redis-slave created
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get service
NAME         TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
kubernetes   ClusterIP 10.15.240.1 <none>        443/TCP     5m49s
redis-master  ClusterIP 10.15.251.61 <none>        6379/TCP    88s
redis-slave   ClusterIP 10.15.246.222 <none>        6379/TCP    7s
kubefy2020@cloudshell:~ (marine-order-267121)$ █
```

### Step 3: Set up the guestbook web frontend

Now that the Redis storage of your guestbook is 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

```
examples/guestbook/frontend-deployment.yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  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
              # If your cluster config does not include a dns service, then to
              # instead access environment variables to find service host
              # info, comment out the 'value: dns' line above, and uncomment the
              # line below:
              # value: env
          ports:
            - containerPort: 80
```

```
kubectl create -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

```
deployment.extensions/frontend created
kubefy2020@cloudshell:~ (marine-order-267121)$ kubectl get pods -l app=guestbook -l tier=frontend
NAME           READY   STATUS            RESTARTS   AGE
frontend-74b4665db5-25w9v  0/1    ContainerCreating  0          6s
frontend-74b4665db5-ptmpn  0/1    ContainerCreating  0          6s
frontend-74b4665db5-qcvkc  0/1    ContainerCreating  0          6s
kubefy2020@cloudshell:~ (marine-order-267121)$ █
```

## 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

```
examples/guestbook/frontend-service.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # if your cluster supports it, uncomment the following to automatically create
  # an external load-balanced IP for the frontend service.
  # 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

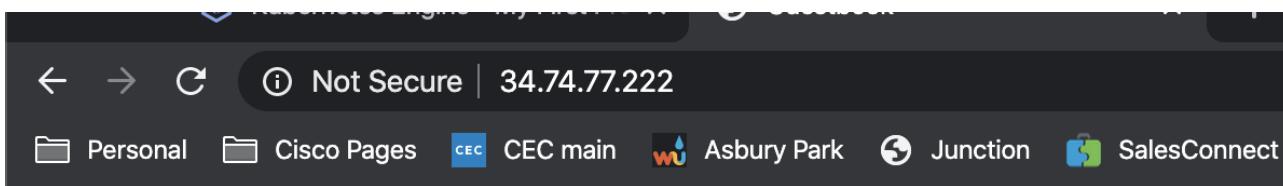
To access the guestbook Service, you need to find the external IP of the Service you just set up by running the command (this may take a minute or so to populate external IP address)

```
kubectl get service frontend
```

- find the external IP of the Service you just set up

```
kubeouser@k8s-cisco-lab1:~$ kubectl get service frontend
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
frontend   LoadBalancer   10.15.251.127   35.196.99.109   80:32035/TCP   1m
kubeouser@k8s-cisco-lab1:~$
```

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



## Guestbook

Messages

Submit

type:

```
kubectl get pods
```

- you should see the 3 frontend PODs we created earlier

```
Frontend   LoadBalancer   10.15.252.87   34.74.77.222   80:32035/TCP
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

deployment.extensions/frontend scaled				
NAME	READY	STATUS	RESTARTS	AGE
frontend-74b4665db5-25w9v	1/1	Running	0	4m21s
frontend-74b4665db5-4dw82	1/1	Running	0	8s
frontend-74b4665db5-c8qfp	1/1	Running	0	8s
frontend-74b4665db5-ptmpn	1/1	Running	0	4m21s
frontend-74b4665db5-qcvkc	1/1	Running	0	4m21s
redis-master-74474c9cd-v66xl	1/1	Running	0	9m15s
redis-slave-74ccb764fc-4xcsv	1/1	Running	0	5m58s
redis-slave-74ccb764fc-cxbp7	1/1	Running	0	5m58s

## Let's Cleanup

`kubectl delete service frontend`

- delete the service

`gcloud compute forwarding-rules list`

- verify deleted

```
kubeuser@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
kubeuser@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

The screenshot shows the GCP Kubernetes Clusters page. At the top, there are buttons for 'CREATE CLUSTER', 'REFRESH', and 'DELETE'. Below is a search bar labeled 'Filter by label or name'. The main table lists 'Kubernetes clusters' with columns: Name, Location, Cluster size, Total cores, Total memory, Notifications, and Labels. A row for 'cluster-1' is selected, indicated by a green checkmark. To the right of this row are three icons: a pencil for edit, a trash can for delete, and a 'Connect' button. A red arrow points from the bottom of the slide towards the delete icon.

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

Return to GCP > Compute Engine - delete Centos 7 VM

The screenshot shows the GCP Compute Engine VM list. A table displays VM details: Name, Zone, Recommendation, Internal IP, External IP, and Connect. A row for 'centos-7-1' is selected, indicated by a green checkmark. To the right of this row is a vertical ellipsis menu. A red arrow points from the bottom of the slide towards the 'Delete' option in this menu. The menu also includes options for Start, Stop, Reset, New instance group, and View logs.

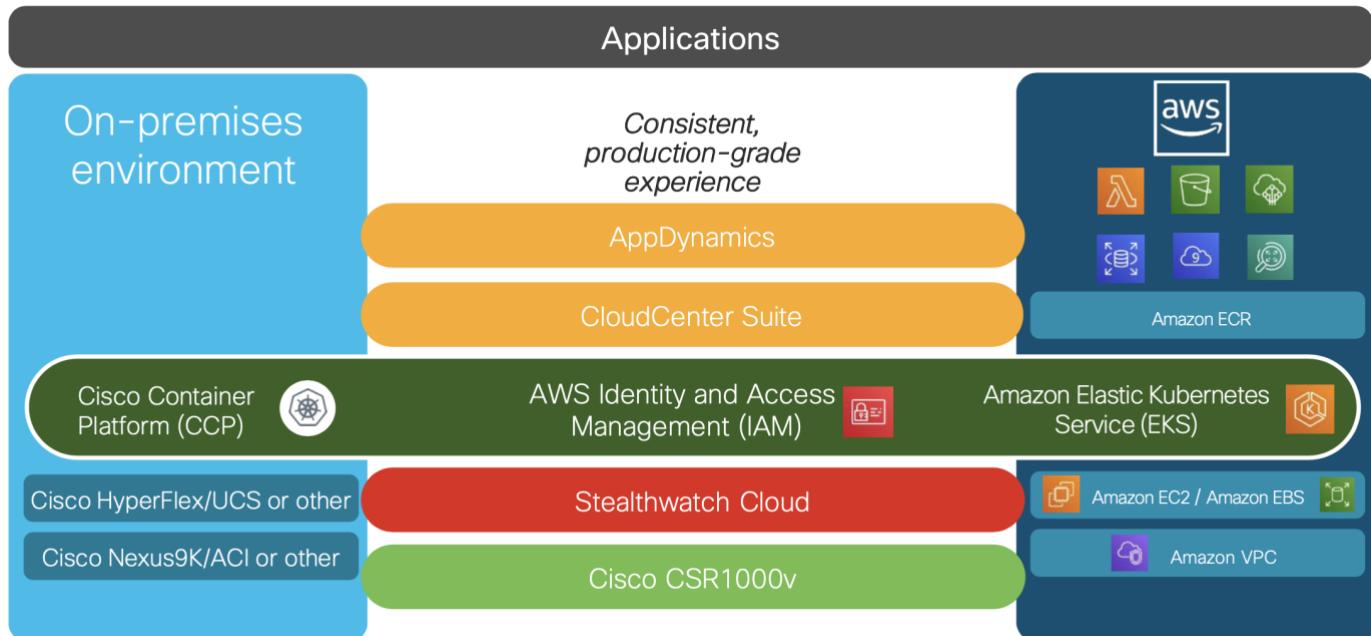
Name	Zone	Recommendation	Internal IP	External IP	Connect
centos-7-1	us-east1-b		10.142.0.2	35.185.78.28	SSH

⋮

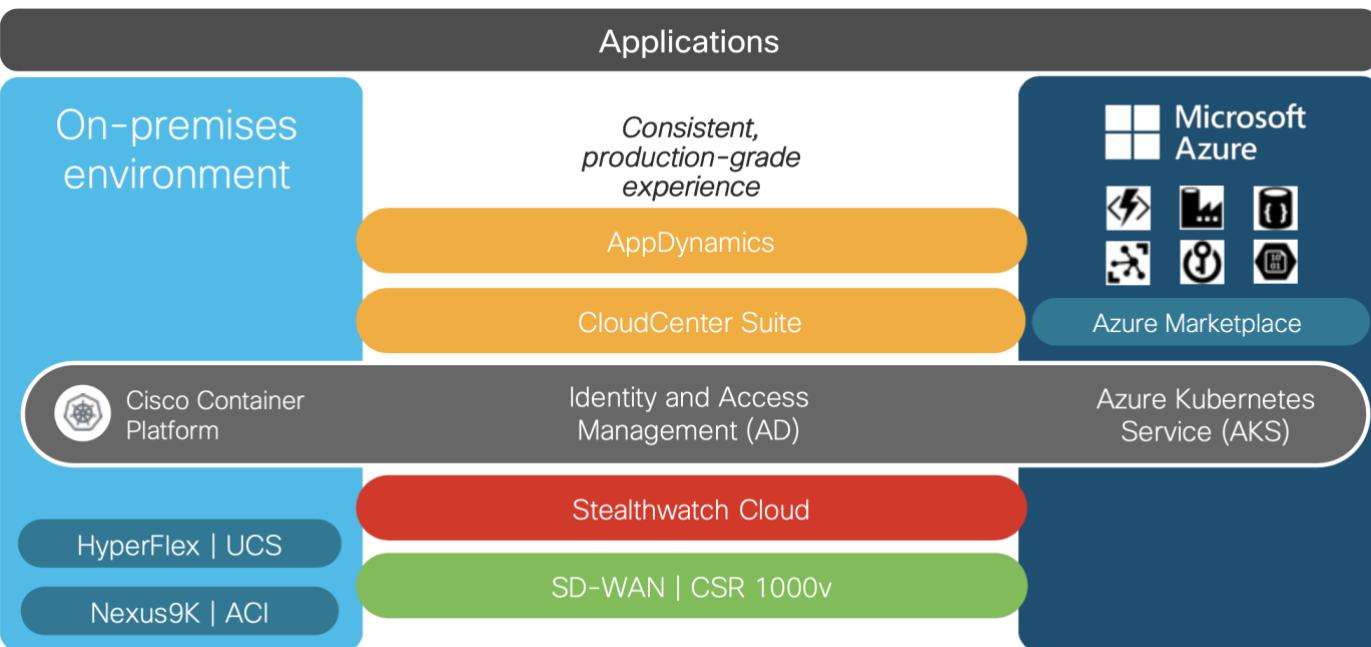
- Start
- Stop
- Reset
- Delete
- New instance group
- View logs

# Cisco Container Platform labs

# Cisco Hybrid Solution for Kubernetes on AWS



# Cisco Hybrid Architecture for Microsoft Azure



dCloud

# Cisco Hybrid Solution for Kubernetes on AWS v.1

<https://dcloud2-sjc.cisco.com/content/demo/142718?returnPathTitleKey=content-view>

## Overview

### Cisco Hybrid Solution for Kubernetes on AWS Lab v1

Schedule

Information Resources

## Overview

Cisco Hybrid Solution for Kubernetes on AWS offers one simple solution supported by Cisco. Now you can manage your hybrid applications lifecycle simply with one environment. With this Solution, you can deploy, connect, secure, and monitor your kubernetes applications across on-premise and AWS cloud. The Core of this solution is Cisco Container Platform (CCP). Cisco Hybrid Solution for Kubernetes on AWS can:

- Deploy kubernetes environments with a simplified and consistent CI/CD experience using common identity and authorization between on-premises and AWS - Cisco Container Platform.
- Model, deploy, and manage application across on-premise and AWS with one hybrid cloud management Platform - Cisco Cloud Center.
- Connect applications with scalable, robust, and secure connectivity with enterprise-class features - Cisco CSR1000v
- Secure applications with proactive monitoring and threat detection delivered by SaaS-based behavioral modelling - Stealthwatch Cloud.

## Scenarios

This demonstration includes the following scenarios:

- Scenario 1. Provision and Manage Kubernetes Cluster Life Cycle across Hybrid Clouds using Single-Control Plane
- Scenario 2. Common IAM Authentication and RBAC Authorization
- Scenario 3. ECR Repository for Container Images
- Scenario 4. Deploy and Manage Hybrid Applications
- Scenario 5. Monitoring the Kubernetes Cluster using Grafana/Prometheus
- Scenario 6. Unified and Secure Networking Across Hybrid Clouds
- Scenario 7. Verify ACI Multi-Tenant Policy-Driven Solution
- Scenario 8. Hyperflex and Multi-Cloud Software Defined Hyperconverged Solution
- Scenario 9 Verify Traffic Monitoring and Alerts using Stealthwatch Cloud

Click “Resources” for lab guide

### Cisco Hybrid Solution for Kubernetes on AWS Lab v1

Information

Resources

Documentation

[Cisco Hybrid Solution for Kubernetes on AWS v1 Guide](#)

If you run into issues with the Swagger backend not displaying the contents in the Petclinic App. You can run through the following steps to recover:

I have already fixed it for now so you should be good, till it happens again.

1. ssh [awsdemouser@198.19.193.214](mailto:awsdemouser@198.19.193.214) >>> swagger-API server
2. sudo -i
3. cd /root/spring-petclinic-rest/
4. `./mvnw spring-boot:run`

After you see the following message, you are good:

```
INFO PetClinicApplication - Started PetClinicApplication in 7.149 seconds  
(JVM running for 11.68)
```

Next, refresh your App and you should be good.

devnet

# Cisco Container Platform (local with HX)

Go to Cisco devnet site:

<https://devnetsandbox.cisco.com/RM/Topology>

Filter on 'Cloud'

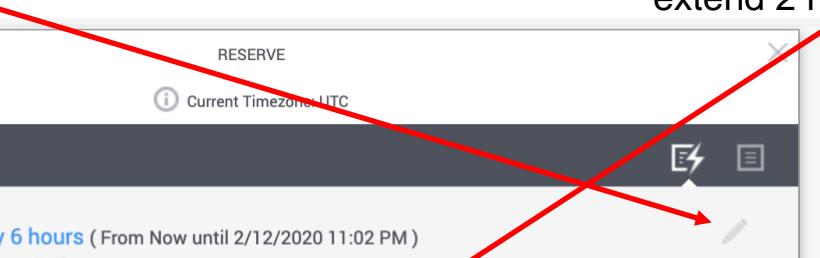
The screenshot shows the Cisco DevNet Learning Labs interface. On the left, there's a sidebar with a 'Sandbox Labs' section containing a 'Reservations' summary and a 'FILTER BY:' dropdown set to 'Available'. In the center, the 'LAB CATALOG (71)' section displays various lab offerings. On the right, a 'BROWSE BY CATEGORY' sidebar lists categories like Networking, Open Source, IoT, Data Center, Collaboration, Cloud, Analytics and Automation SW, and Security. A red arrow points from the 'Cloud' category in the sidebar to the 'Available' filter in the sidebar.

Lab Name	Version	Status	Action
ACI & Kubernetes	Version 3.1(2m)	RESERVE	
ACI Hardware Lab	Version 3.1(2v)	RESERVE	Contains a resource(s) that is currently unavailable
ACI Simulator	Version 4.1	ALWAYS-ON	ACI Simulator Always-On - V4
ACI Simulator Reservation	Version 4.1		
Aironet Developer Platform With Raspberry Pi	Version 1.0		DEMO
Cisco Partner Solution: Alleantia	Version 1.0		

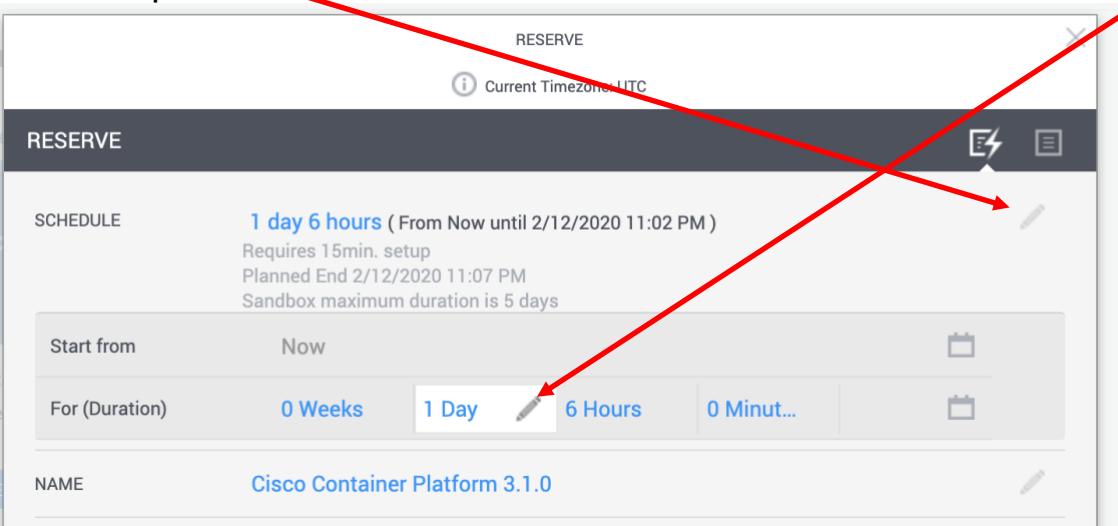
## Reserve Cisco Container Platform

The screenshot shows the 'CLOUD (2)' section of the Cisco DevNet Learning Labs interface. The sidebar still shows 'Available' selected under 'FILTER BY:'. The main area displays two available labs: 'Cisco Container Platform' (Version 3.1.0) and 'Istio' (Version 1.2). The 'Cisco Container Platform' entry has a red circle around its 'RESERVE' button.

Lab Name	Version	Action
Cisco Container Platform	Version 3.1.0	RESERVE
Istio	Version 1.2	RESERVE

Click the 'pencil' 

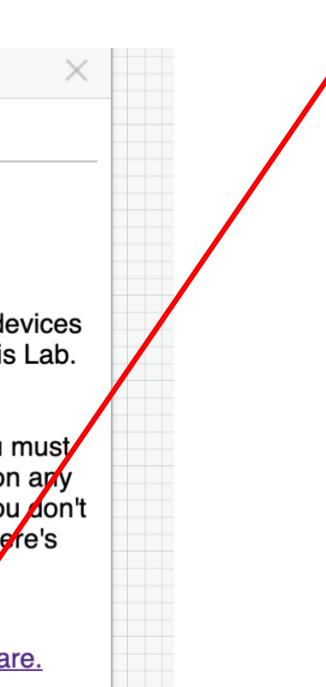
extend 2 hour default 

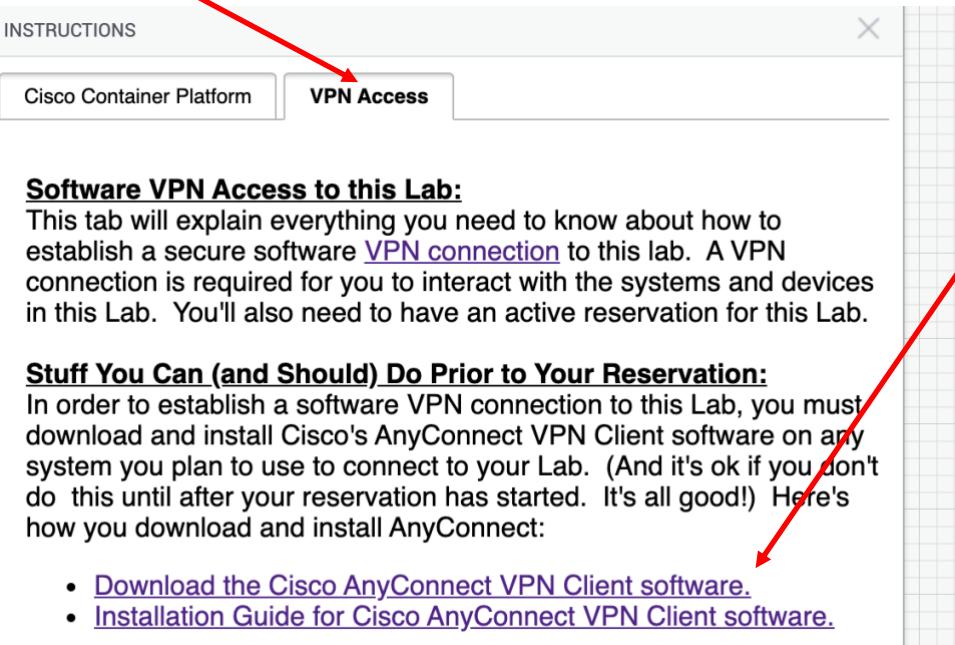


SCHEDULE	1 day 6 hours (From Now until 2/12/2020 11:02 PM)
Requires 15min. setup	
Planned End 2/12/2020 11:07 PM	
Sandbox maximum duration is 5 days	
Start from	Now
For (Duration)	0 Weeks <input checked="" type="radio"/> 1 Day <input type="radio"/> 6 Hours <input type="radio"/> 0 Minut...
NAME	Cisco Container Platform 3.1.0

While waiting for lab to initialize install AnyConnect if not already installed (you will also receive an email telling you lab is setting up with AnyConnect instructions)

Click VPN tab 

download and install the software 



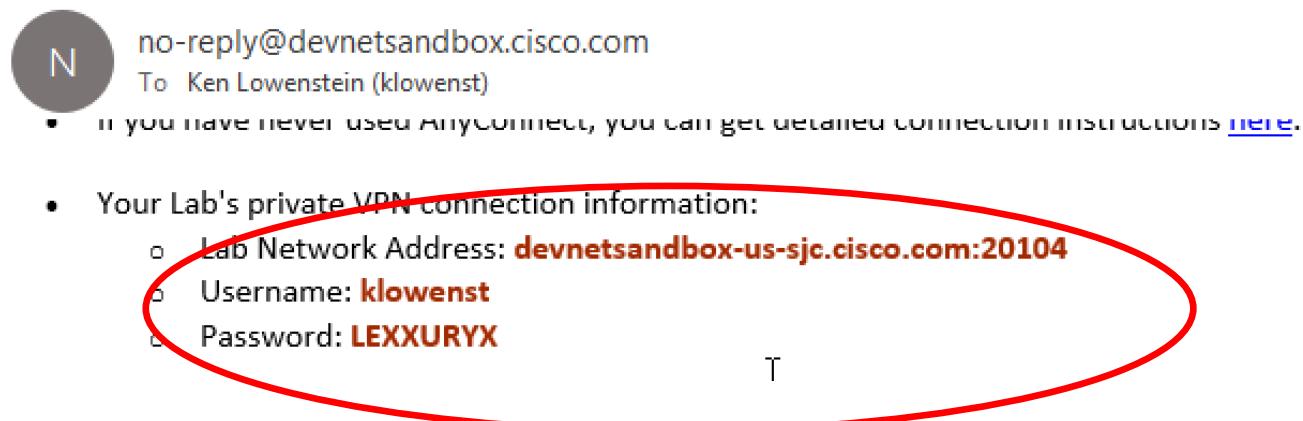
**Software VPN Access to this Lab:**  
 This tab will explain everything you need to know about how to establish a secure software [VPN connection](#) to this lab. A VPN connection is required for you to interact with the systems and devices in this Lab. You'll also need to have an active reservation for this Lab.

**Stuff You Can (and Should) Do Prior to Your Reservation:**  
 In order to establish a software VPN connection to this Lab, you must download and install Cisco's AnyConnect VPN Client software on any system you plan to use to connect to your Lab. (And it's ok if you don't do this until after your reservation has started. It's all good!) Here's how you download and install AnyConnect:

- [Download the Cisco AnyConnect VPN Client software.](#)
- [Installation Guide for Cisco AnyConnect VPN Client software.](#)

You will receive a second email with VPN settings

## Your Cisco DevNet Sandbox Lab is Ready



Enter this info into AnyConnect client then attach to lab as directed on devnet

Cisco Container Platform

VPN Access

### Internet Availability

Only the pre-configured default cluster i.e. **sandbox-demo-cluster-1** will have Internet access, though more clusters can be created in CCP dashboard.

### Getting Started!

If this is your first time using Cisco Container Platform, this Sandbox is designed for use with the following Learning Labs, which will guide you through the Platform, providing an understanding of its features and design. It will also have you get hands on with the platform UI and API's to create, view, access and then consume a Kubernetes cluster to deploy a sample application.

- (Learning Lab) Introduction to Cisco Container Platform.

### Sandbox Caveats

Unlike a real-world on premise deployment of Cisco Container

**IMPORTANT – see next page**

On step 3 of lab you are asked to cut and paste test into the BODY of a POST request. Unfortunately, the UUID is incorrect.

How To Setup Your Own Computer

## Step 3: API Cluster Creation

Time to create our second cluster! Re-Open the API docs if you closed them and re-authenticate like you did earlier in the Lab. Link: <https://10.10.20.110/2/swaggerapi/>.

Click on `Expand Operations` in the endpoint item entitled `2/clusters : List of cluster endpoints`.

Scroll down to the green `POST` request, which allows us to create a new cluster.

Use the following text in the BODY field, notice that these are all the same options (potentially in UUID form) which we specified from the UI.

```
{"is_harbor_enabled":false,"provider_client_config_uuid":"f610d6d4-5cd7-4a3c-937a-dbf69280dc5b","name":"API-Cluster","kubernetes_version":"1.12.3","ssh_key":"ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlzdHA1MjEAAAIBmlzdHA1MjEAAACFB AHLSb9ZkXQL5/GI12258c+AIKVhDN1p1YYjvJR5oliqoR/gN/65D04BfsZWE8nk00AtJzvEVbjenwLeWuvIQsFs5AHa5uM4Fpmw3Ylpt1tB/GHZ5Mg9sh1iLh5agSgNLWkAgCRvySmL03fSq0IKarnQrMqId2pGUlnZr/YPP4irTvU6w== sandbox@CCP_SANDBOX_NISTP521_KEY","description":"Cluster created via API","datacenter":"CCP","cluster":"CCP","resource_pool":"CCP/Resources","networks":[{"VMNetwork"}],"datastore":"CCPDatastore","storage_class":"vsphere","worker":true}
```

Please use the following json file instead

[https://github.com/klowenst/CCP-lab/blob/master/API-create\\_cluster](https://github.com/klowenst/CCP-lab/blob/master/API-create_cluster)

Done