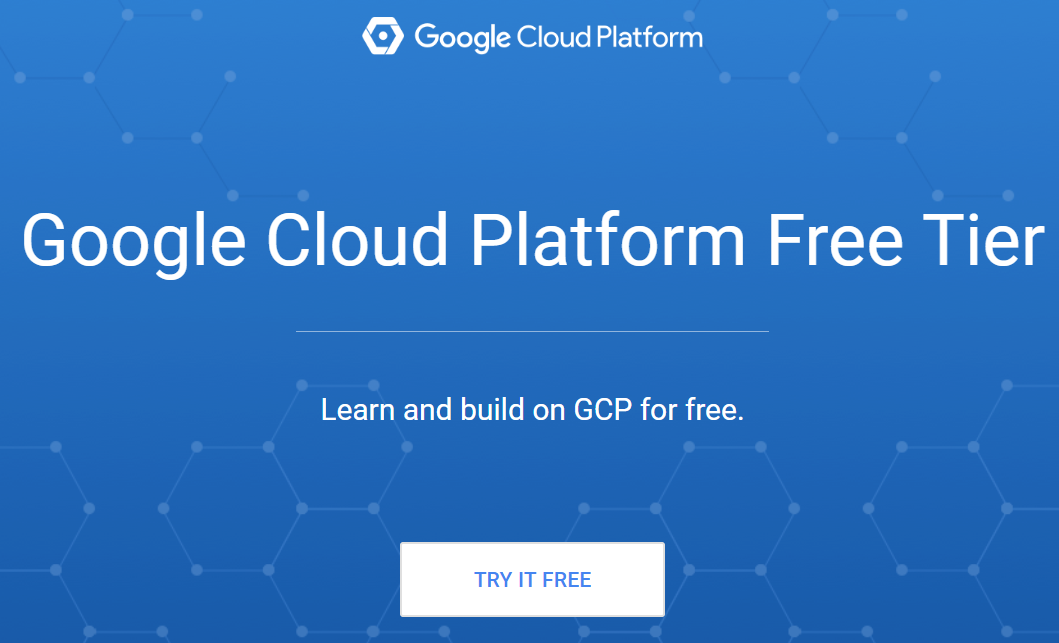
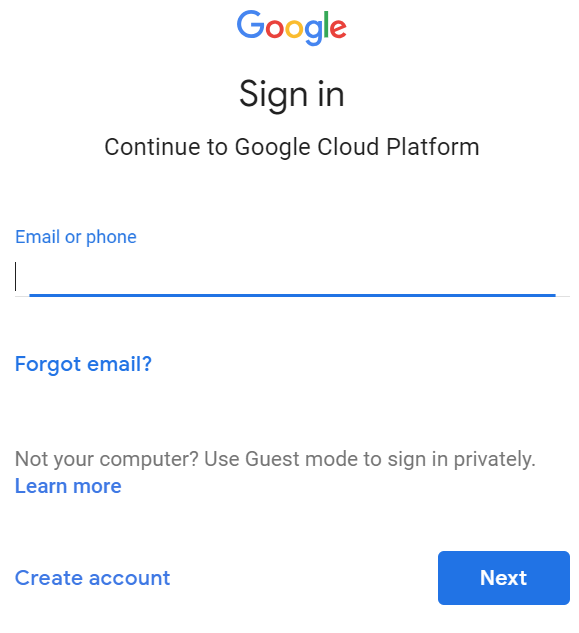
# PAS and PKS Hands-on (Kuala Lumpur TRS, Sept 2018)

## Pre-requisites (GCP and PWS setup)

In order to minimize the network issues and the set up complexity we will use Google Cloud Platform to do the labs. Prior to attending the hands-on session please create a free account in GCP following this link: <https://cloud.google.com/free/>

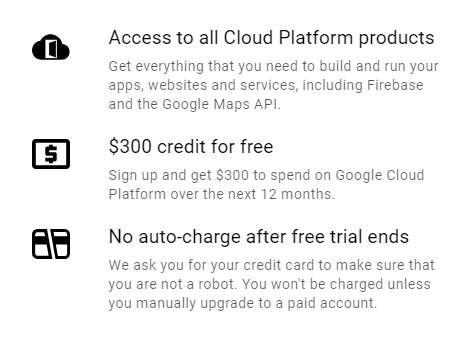


For authentication GCP uses your Gmail account. Essentially the same account that gives you access to Gmail, Android, Google Drive, Youtube etc is what you will use for GCP. It uses the same Single-Sign-On for all Google services, including GCP.

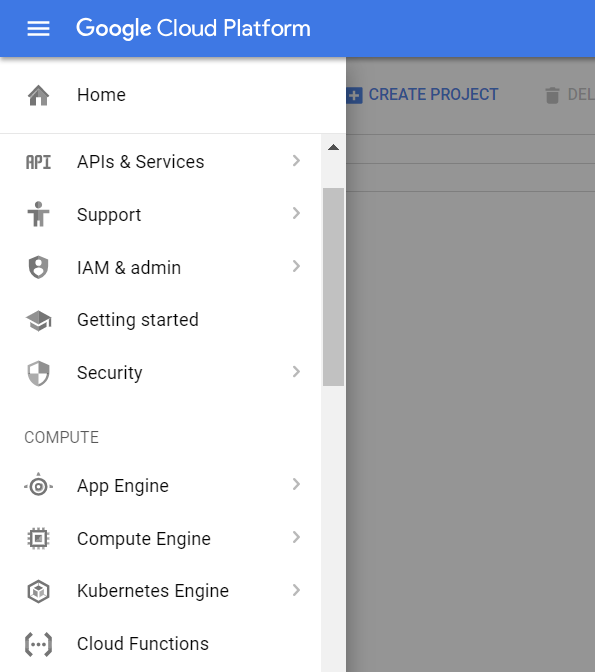


If you don’t have any account with Google please create one.

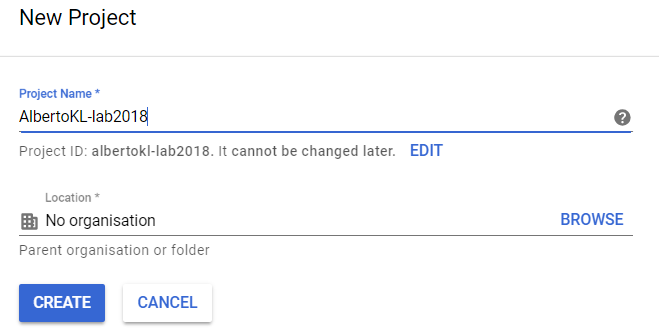
The Free Trial account comes with $300 credit for free for 1 year. However at some point it will ask you for your credit card details. Please note that as the sign says they don’t intend to charge you after that credit expires. You will have to manually upgrade to “paid” account



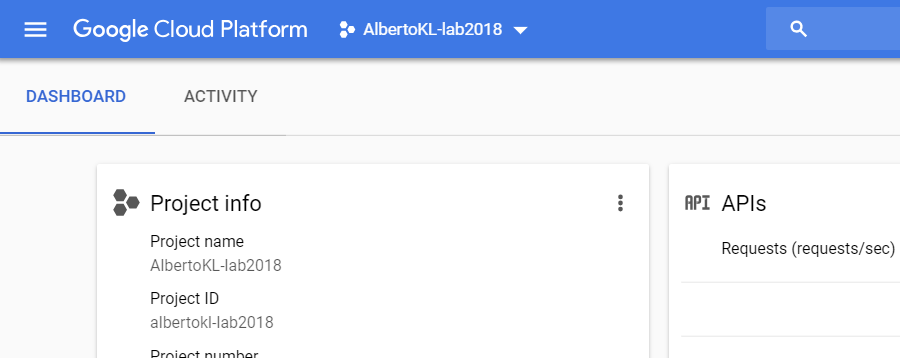
Follow the prompts and eventually you will arrive to the GCP Console, which will look like this:



In GCP all resources you use need to be part of a “Project”. Click on “Create Project” and give it a name

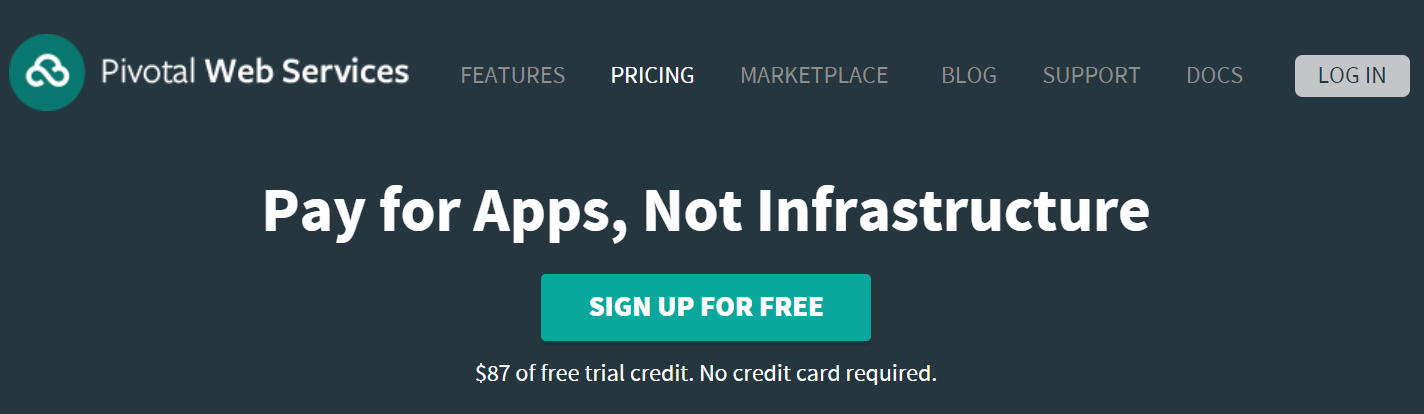


After 1 minute or so your project will be ready. If you click in “Google Cloud Platform” at the top-left you will be go back to the console and your project will be selected



This concludes the setup portion on GCP. Now you will need to create a free account with Pivotal Web Services (PWS). Go to: <https://run.pivotal.io/pricing/>

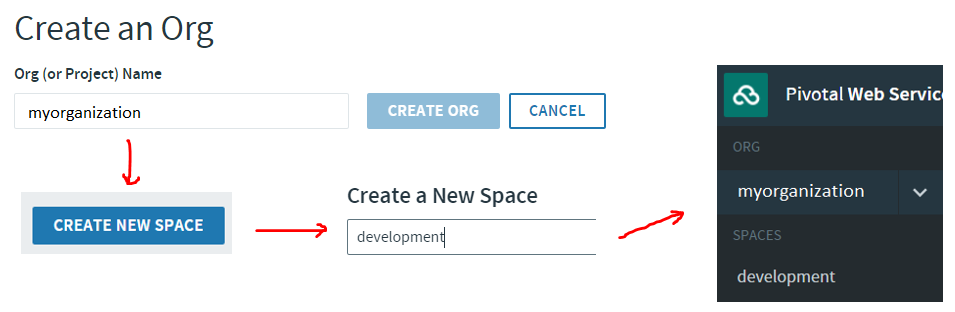
The free account includes $87 credit for 1 year. It might ask you for your credit card details.



Please make sure you keep a record of your credentials so that you can log back to do the lab

Once the account is created, log in to the PWS console: <https://console.run.pivotal.io/>

In order to get ready for the lab, you will need to create an “organization” and a “space”



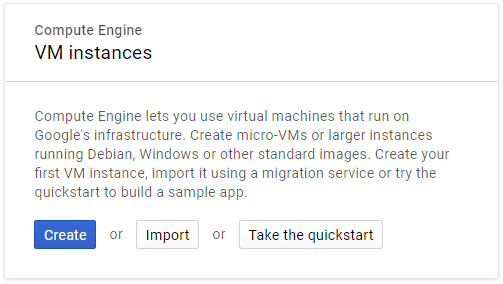
Congrats!! You are ready for the Lab

## Docker Lab

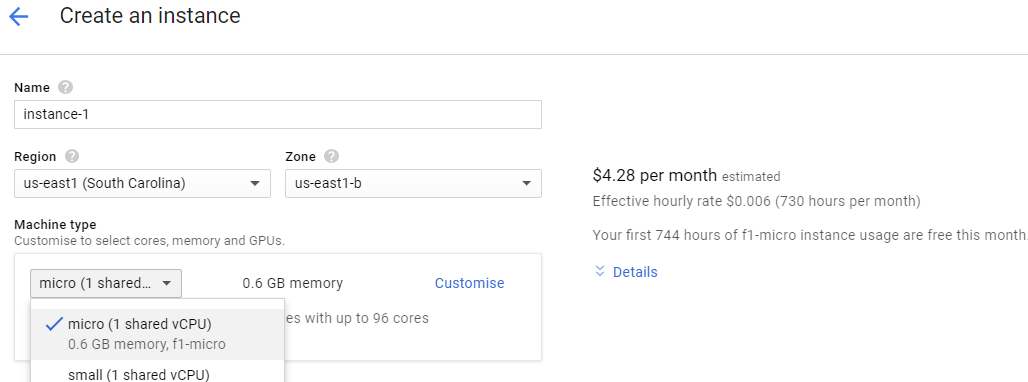
Firstly we are going to create a virtual machine in GCP. We will use this VM to play with Docker and to push our first app to Pivotal Cloud foundry at the end of the session

Go to your GCP console expand the left navigation menu and click “Compute Engine”

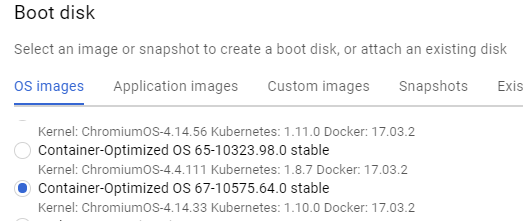
After a few moments it will show you this menu. Click “Create”.



In the Create an instance menu” select a machine type of “micro”. On the right you can see the cost per month.



In the “Boot disk” click “Change” and from the list of available Virtual Machines select COS the latest stable Container Optimized OS (COS)



Leave the “Boot disk type” to the default “Standard persistent disk” with 10GB

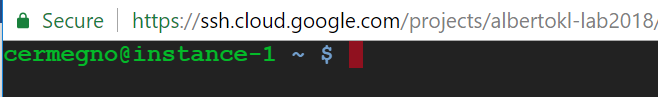
In the “Firewall” click on “HTTP traffic”

Click “Create”. After a few moments it will be ready. If the “spinning circle” is still going after a minute you can refresh the page. Sometimes it finishes but it keeps spinning

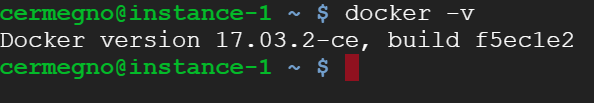
In order to connect to it you can select “Open in browser window” from the “SSH” drop-down menu.



A new browser window will open with your Virtual Machine console. If you were questioning why we are using GCP for the lab … this is one of the reasons, how handy is this? SSH over HTTPS



We chose a VM that comes with Docker pre-installed, which saves some time too.



**Important TIP** : You can work more efficiently with GCP consoles by using these two tricks:

* If you need to copy anything from the output, just select it with the mouse
* Tto past something in the console (ex: all the commands in this guide) simply press CTRL+V

The files we need to complete the lab are sitting in a repository in Github. Github is a web-based hosted service for version control using Git. It offers distributed version control and source code management functionality. You can explore the repo by opening this link in your laptop’s web browser and clicking on the “*KL-Summit-18*” link

<https://github.com/cermegno>

We need these files inside our virtual machine. To move them there we are going to use the “git” command line tool to clone the Github repository

git clone https://github.com/cermegno/KL-Summit-18.git

A directory should be created in your VM. Go into the “docker” directory

ls

cd KL-Summit-18

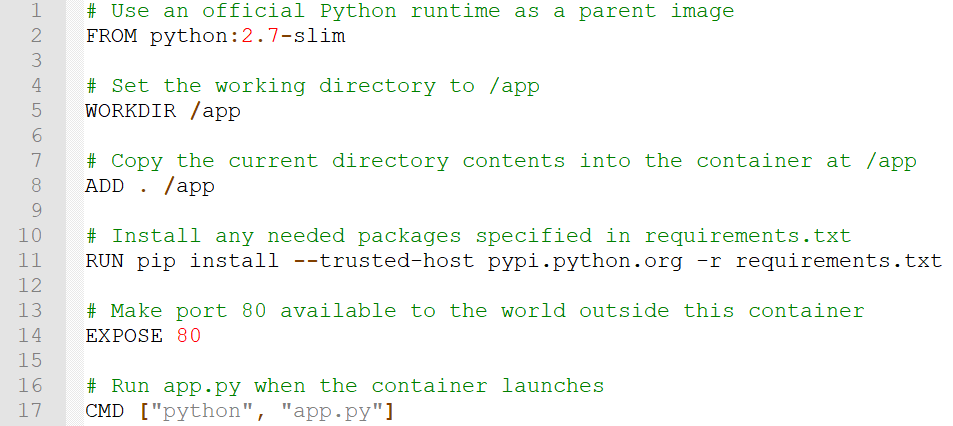
cd docker

It should contain 3 files. You can see what they contain by using the “cat” tool

cat app.py

cat Dockerfile

In particular you can see the Dockerfile. This is the file where we give Docker instructions on how we want to create a container



For example in line 2 we are specifying we want to start with another base image and install our bits and pieces on top. That image is an image that comes with Python installed because our app is written in Python. And in Line 8 we are adding our application files.

One thing that is important to understand is that a container image are a hierarchy of file systems layered on top of each other. Each new command in your Dockerfile adds a new layer to the image and the resulting containers you create later have a file system that is the “union” of all those layers.

One implication of these “layers” is that the size of container image is much smaller than a Virtual Machine because it leverages the file system of the host.

Now we are ready to build our first container

**IMPORTANT**: Don't forget the “.” at the end of the command to specify the directory that contains the build files

docker build -t myfirstcontainer .

Docker distinguishes between containers and container images:

* Images are the templates we use to create containers. They are often organized in “container registries”
* Containers are in-memory instances of the image. With a single container image you can create many containers

This is how you check what container images you have in your Docker. We should have the one we just created

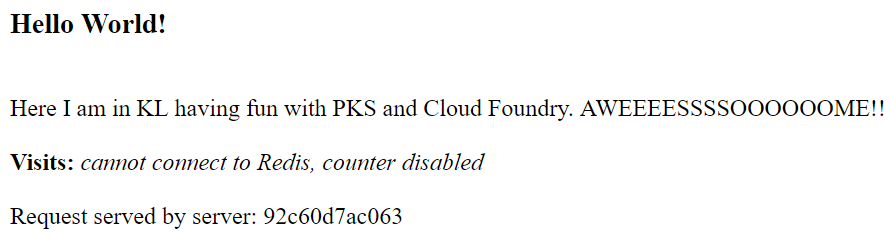
docker image ls

Finally you use the “docker run” command to spin up a container. In this instance we are mapping port 80 from the container to port 80 on the VM so that it can be accessible from the outside world

docker run -p 80:80 myfirstcontainer

The file “app.py” is a python script that spins up a little web app with a single page. You should be able view that page from your laptop because we allowed HTTP traffic when we created the VM

Go to the GCP console and find the “external” IP address of your VM. Then open a browser and go to your VM's IP address: http://<your\_vm\_ip\_address>



There is always a first time for everything. Congratulations on your first container!!

Back in your VM **press CTRL+C to stop** the web server and regain access to your command line.

If you need to run more than one container simultaneously you will need to run them in “detached” mode. This allows you to regain control of the command line. This is achieved with the “-d” flag

But if you run them detached, is there a way of gaining access to its console? Try the following commands to open an interactive terminal session into a detached container

docker run -d myfirstcontainer

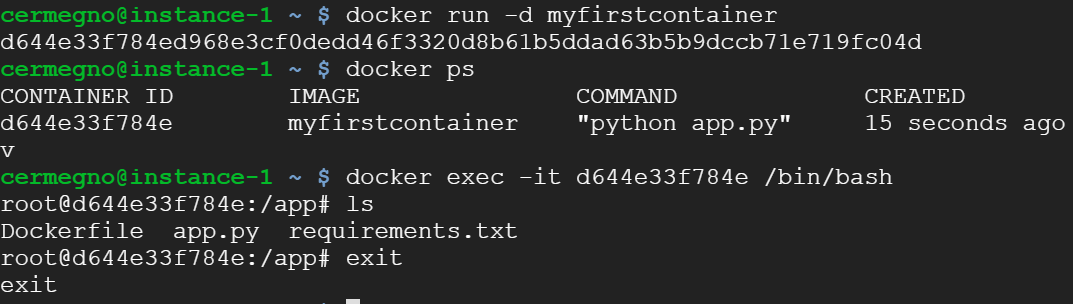
docker ps

docker exec -it <container\_id> /bin/bash

Look up you <container id> with “docker ps” or “docker container ls” and use it in your “docker exec” command.

If you issue an “ls” command you will see the 3 files we had in our directory when we created the container.

Finally you can get out of the container by typing “exit”



An alternative way of creating container images is by using an existing container. For example you could try to install any other packages or make any other changes you need and the package it with the “commit” command. But the Dockerfile is the recommended method

docker commit --help

We exited but the container is still running in the background. You can see it with any of these two commands

docker ps

docker container ls

When you no longer need your container you can stop it by referring to its <container\_id>

docker container stop <container\_id>

If we run the “ps” command again we shouldn’t see any containers running

docker container ls

But are we sure there is nothing there? You can see any containers that were created but are no longer running with this option

docker container ls -a

You could use “docker container rm <CONTAINER\_ID>” to remove them one by one or you can delete all inactive containers at once like this

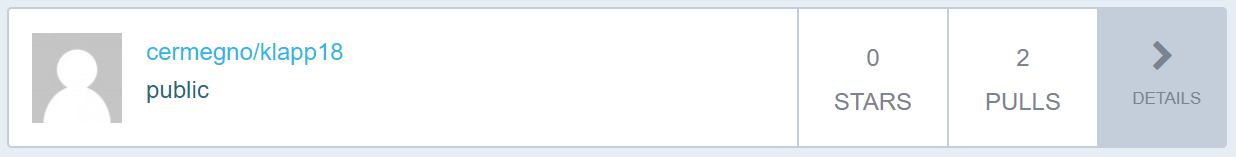
docker container prune

If you are happy with your container, you could decide to store for future use or for distribution. Container images typically get stored in a "registry" like Docker Hub. This would require you to open an account first. It's free and you can get a feel for how it’s done by looking at the help of the “docker push” command but in the interest of time we are not going to do it.

docker push --help

Instead I have already pushed the same image to Docker Hub under my own account so that we can use it later in the Kubernetes section

https://hub.docker.com/r/cermegno/klapp18/



You noticed that the counter in the app wasn’t working because it expected to have a Redis database but we hadn’t configured one. So this app has 2 tiers, let’s deploy both of them now.

Docker Hub contains not just container images from individuals like you and me. It also contains official images for many commercial products and open source projects. Use the search box in Docker Hub to find some of these: mysql, cassandra, wordpress, redis, …

Container registries also do version control. It uses the concept of tags to represent the different versions. You can see all the versions of a given image. For example in the Docker Hub official Redis repository you can click on the tags tab to see what versions are available:

<https://hub.docker.com/r/library/redis/tags/>

We are going to spin up a redis container for our 2-tier app. If we don’t specify a specific version it will pick the image with the “latest” tag

docker run --rm --name myredis -d redis

We have done a few things differently this time:

* Instead of pulling an image and then running a container, you can use “docker run” to do both at one go
* “-d” means we are running in “detached” mode, ie in the background. You will notice that the control of the prompt is returned immediately
* “--rm" means it will remove the container automatically when it exits
* We use “--name" to give our container a name instead of letting Docker assign one of those two-words funny ones

Now you can check your image was downloaded and your container is running

docker image ls

docker container ls

Now we are going to bring up the web app container again but we need to add “--link" to make sure the front-end app can access the backend Redis database. It helps that we gave it a short friendly name ;-)

docker run -p 80:80 --link myredis:redis -d myfirstcontainer

docker container ls

Do you see both containers running? In that case it is time to test it. Use the IP address of your VM in your laptop’s web browser

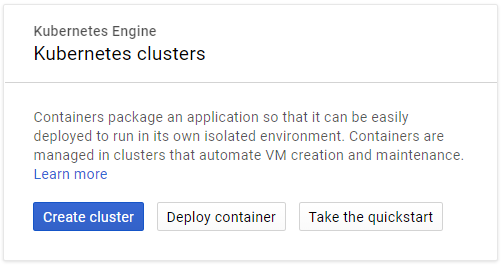
http://<your\_vm\_ip\_address>

Refresh the page a few times. Is the counter incrementing? If so, well done! You have just successfully deployed your first containerized 2-tier app

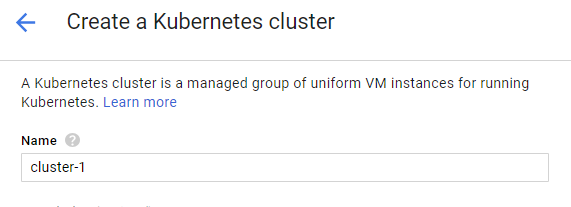
## Kubernetes Lab

What if you wanted to deploy at once many containers on each tier and get them working as a team in one single deployment? The original solution from the Docker Company included concepts such as Swarms and Stacks but eventually the market has chosen to do such things through Kubernetes. Luckily for us GCP is potentially the best place to run Kubernetes … they invented it.

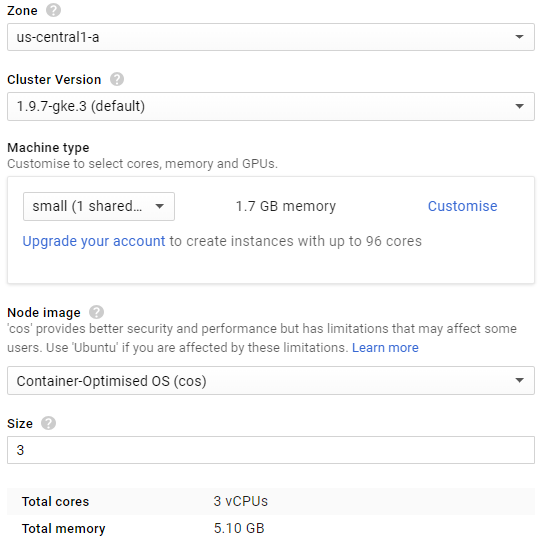
Go to your GCP console and from the left menu select “Kubernetes Engine”. When it is ready you will see:



Click “Create cluster”

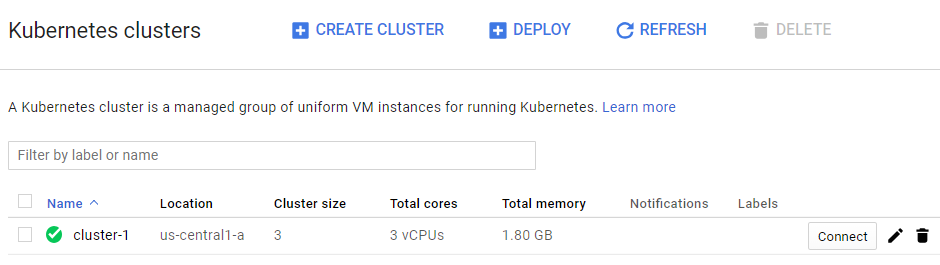


You can leave everything to default. Change only the “Machine type”. Let’s use the “small” with 1 shared CPU.

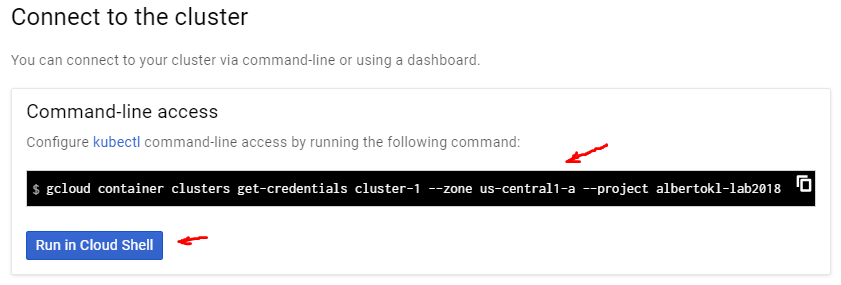


**IMPORTANT**: Please write down the “Name” of your cluster and the “Zone” because you will need those to connect to and manage the cluster. As you can see in my case I am using the “us-central1-a” zone

Give it a few moments and your cluster will be ready. If the “spinning circle” is still going after a minute you can refresh the page. Sometimes it finishes but it keeps spinning

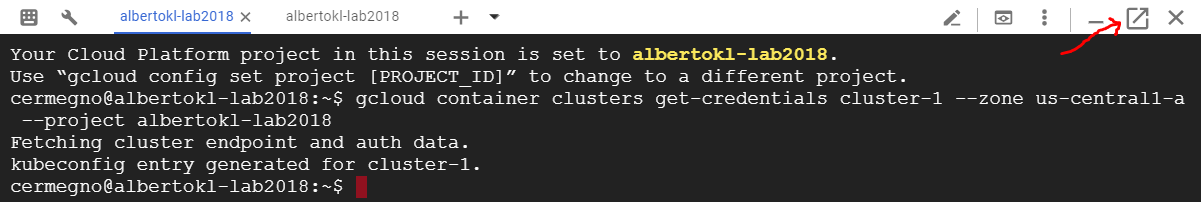


Click “Connect”



In the windows that opens, select the “gcloud” command and CTRL+C to copy. Then click on “Run in Cloud Shell”. This will open a shell from where you can manage your Kubernetes cluster.

It will take a few moments to get ready. When ready paste the “gcloud” command with CTRL+V and press enter. This will make all the necessary configuration changes so that you can use the “kubectl” tool. “Kubectl” is the main tool you use to manage Kubernetes clusters



By default the shell opens at the bottom of the page. If you want to open it in a different tab click in the square arrow symbol in the right.

Let's start by obtaining information about our cluster

kubectl cluster-info

kubectl get nodes

In Kubernetes we deploy pods instead of containers. Pods can contain multiple containers and other objects like storage volumes. There should be no pods at this stage

kubectl get pods

Let's deploy our app in Kubernetes. You could have a different container registry but by default it will look up for the image in Docker Hub. In this command we are pulling the container image I pushed for you in the previous section.

kubectl run klapp18 --image=cermegno/klapp18 --port 80

kubectl get pods

If you issue the “get pods” commands a few times you will see how it will change state from “Creating” to “Running”. Mine created in 15 seconds.

Kubernetes uses deployments for deploying multiple identical pods. Your app might need multiple instances for resilience or performance. Kubernetes also uses the "deployment" to manage upgrades of your application in a controlled way, performing a rolling update by default.

kubectl get deployments

As you can see the run command has created a “deployment” with a “desired” state of 1 running pod. If we were to scale the app to let’s say 2, we would see “desired” changing to 2

At this stage you cannot still access your app. There is something else we have to do. We need to expose it also referred to as “creating a service”. Services in Kubernetes are a set of pods where an application or component is running, such as one tier of a multi-tier application. Services define network properties like ports and access methods.

kubectl expose deployment klapp18 --type=LoadBalancer --port 80 --target-port 80

Passing in the “--type" LoadBalancer flag creates a “Compute Engine” load balancer for your container

The “--port" flag initializes public port 80 to the Internet and “--target-port” routes the traffic to port 80 of the application.

kubectl get service

This will display the connectivity details of our app. Initially will show as pending and eventually show an external IP. In my case it took 50 seconds. Once you see an external IP address, copy it by selecting it with your mouse, open a new tab then paste the IP address in your browser

It is very easy is to add more pods to your deployment. Once it is scaled refresh your app in the browser and see how every few clicks the load balancer is directing you to a different server

kubectl scale deployment klapp18 --replicas=2

kubectl get pods

kubectl get deployments

Now let's run the backend container. Redis uses port 6379 by default

kubectl run redis --image=redis --port 6379

kubectl get pods

It is important that the name of the deployment and service match what the code is looking for. Essentially this becomes the hostname. If you revisit the code this was the line that contained the exact spelling (see it in red):

redis = Redis(host="**redis**", db=0, ....

kubectl expose deployment redis --type=ClusterIP --port 6379

kubectl get services

This time we are using "ClusterIP" instead of "LoadBalancer". We don't need our Redis database to be expose externally so by using "ClusterIP" we expose it internally within the Kubernetes cluster to other apps, and to our frontend app in particular

Now open again the external IP address of your frontend app and check that the counter is increasing

### Working with YAML files

So far we have been creating pods, deployments, services etc by using commands and passing parameters but that’s not how enterprises work with Kubernetes. Kubernetes has the ability to import and export all configuration as YAML files. This is a very common format for configuration files. This allows organizations to leverage version control tools like Git to maintain their cluster configuration.

You can see any piece of configuration in YAML format by using the “-o yaml” flag, ex:

kubectl get pods -o yaml

kubectl get deployment -o yaml

kubectl get service -o yaml

If the output is too long to fit into the screen you can pipe it through “more” and scroll down by pressing enter

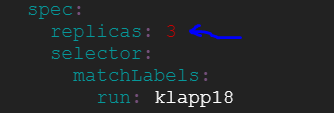
kubectl get pods -o yaml | more

You can then edit a YAML file and use it to create a resource with “kubectl create -f file.yaml”. For example you can create a YAML file out of your deployment, then edit the amount of replicas, save the files and use it to create a deployment

kubectl get deployments -o yaml > mydep.yaml

vi mydep.yaml

Modify the number of replicas



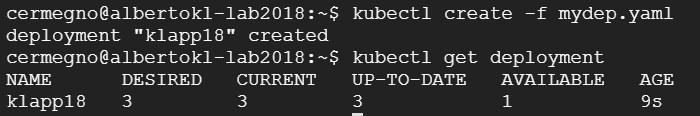
Before we get to apply the deployment we will delete it to prevent it from throwing an error:

kubectl delete deployment klapp18

kubectl get deployment

kubectl create -f mydep.yaml

kubectl get deployment



The ability to use YAML files means that the container management can be done more efficiently but the main conclusion is that with CaaS one has to manage the containers and everything they imply. This is in contrast to PaaS where developers have to worry only about the code, which is what ultimately delivers business value

### Clean-up (Optional)

In order to tear down the cluster, first delete the services and the cluster or you can simply delete the cluster from the GUI.

kubectl delete service klapp18

kubectl delete service redis

gcloud container clusters delete cluster-1 --zone us-central1-a

## Pivotal Cloud Foundry Lab

In order connect to Pivotal Web Services we will need a Linux machine with the CF-CLI tool. Unfortunately the COS Virtual Machine we created at the beginning doesn't have a package manager to install any additional software. However now we have learned a thing or two about containers so spinning up what we need is very easy. Let's go back to our original Virtual Machine (the Container Optimized System) to create a container running CentOS Linux. The two additional flags give you an interactive terminal session

docker run -i -t centos

You can verify you are actually running CentOS like this

cat /etc/\*release

You can wonder about the fact that you are running CentOS Linux on top of Chromium Linux!!

We are going to need a bunch of files. The easiest is to pull the repo onto our new container, but before we need to install git

yum -y install git

Clone the lab repository

git clone https://github.com/cermegno/KL-Summit-18.git

The repository includes the cf-cli package (cf.rpm). Let's install cf-cli

cd KL-Summit-18

rpm -i cf.rpm

Verify it is installed

cf -v

Go to the directory that contains the same app for Cloud Foundry

cd cf

ls

Login to your PWS account

cf login -a https://api.run.pivotal.io

Select your Organization and Space. If you only have one of each it might select them for you automatically

Now it is time to push your app. As a developer in Cloud Foundry this is the only thing you care about. You will have to give your application a name. This has to be a unique name within PWS namespace which corresponds to URL’s ending in “.cfapps.io” so avoid names like “test” or “test123”. You could try to use your name as part of the app name to help you make it unique

cf push -m 64M <your\_app\_name>

The "-m 64M" specifies the amount of RAM. PWS charges by RAM in use. If you don't specify an amount it will default to 1GB

This is how you see what apps are running in your space

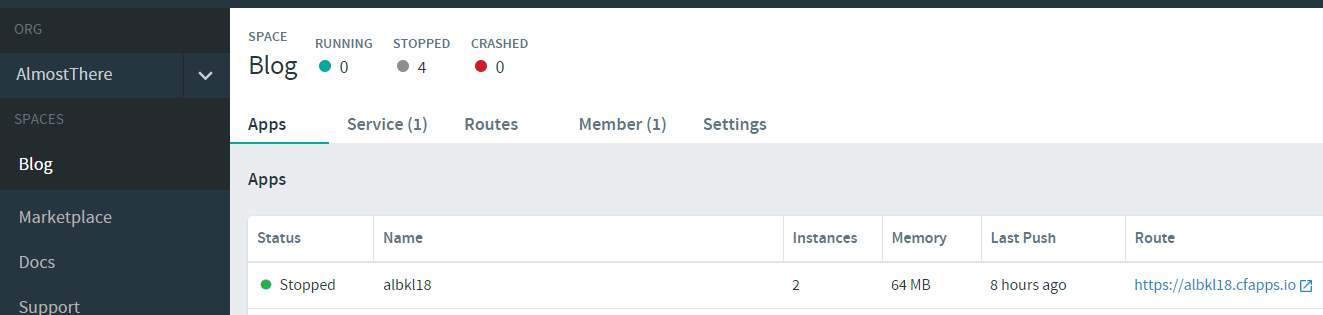
cf apps

Now go to your smartphone and open a browser to http://<your\_app\_name>.cfapps.io

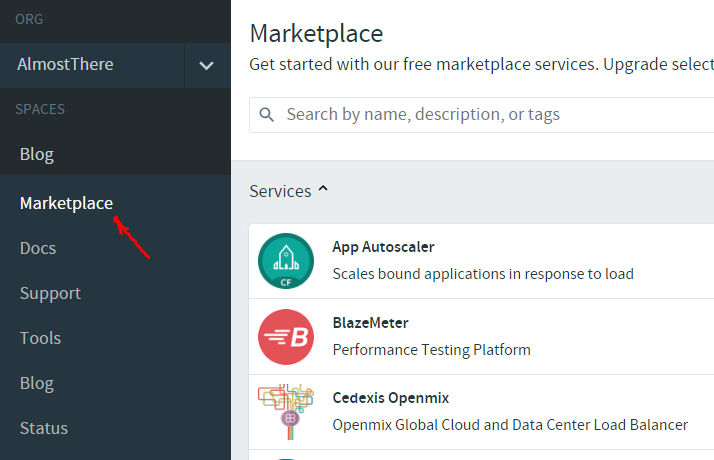
The app should be running but we haven’t provided any Redis database. The best candidate workloads for PAS are “stateless microservices”. This rules out pretty much every database. In order to deal with stateful requirements Cloud Native applications typically rely on “backing services” provided by the platform itself. PWS provides a large number of services through the Service Marketplace. Let’s explore it and see how we can add a Redis service to our app. Let’s use the PWS console for that instead of CLI

Open a new tab in your web browser and navigate to <https://console.run.pivotal.io>

Log in with your username and password. You will see your app

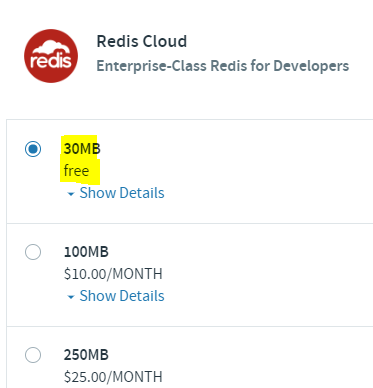


Now click on “Marketplace” on the left pane. Take a few moments to scroll down and see what services are available. Most services have various offerings available with varying service levels, ranging from free test environments to expensive highly available services.

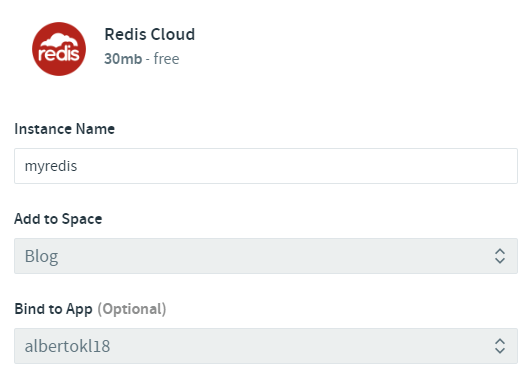


One of the best examples is the Redis service offerings. Select “Redis” and see the offerings available. You can click on “Show details” to see what they include. The point here is that Developers don’t need to worry about all this things they can focus on their code and just consume the services

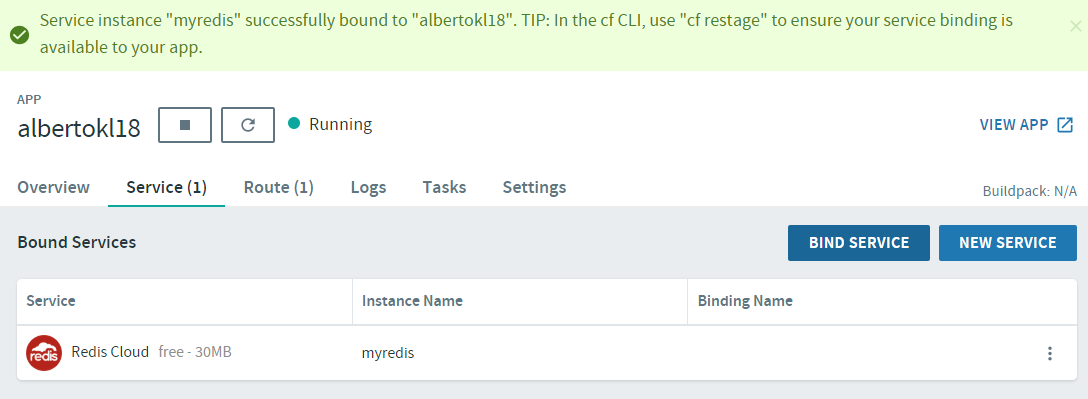
Now select the “free” 30MB Redis instance



In the next screen give it an instance name and “Bind” it to your app



Now your app will show it has a Redis service bound



**IMPORTANT**: You will have to restart the app so that it reads the Redis credentials from the environment. Once you do that view your app again on your phone and see if the counter works

### Clean-up (Optional)

You can delete your app the PWS console by clicking on “Settings” for the app, scrolling down and clicking “delete”

Or from the CF-CLI you can type “cf delete <your\_app\_name>”

At this point you can also delete the Virtual Machine we created at the beginning.