Untitled

Source: https://medium.com/google-cloud/deploying-docker-images-to-google-cloud-using-kubernetes-engine-637af009e594
link *** > block quote

Requirements

This post assumes that you already have a working docker image, that you can build and run on your local machine, and obviously a Google Cloud account with all of the necessary APIs activated, and the Google Cloud SDK installed on your machine. If all of that is in order, you are good to go!

Source: https://www.r-bloggers.com/how-to-deploy-a-predictive-service-to-kubernetes-with-r-and-the-azurecontainers-packag

Fit the model

We'll fit a simple model for illustrative purposes, using the Boston housing dataset (which ships with R in the MASS package). To make the deployment process more interesting, the model we fit will be a random forest, using the randomForest package. This is not part of R, so we'll have to install it from CRAN.

```
data(Boston, package="MASS")
install.packages("randomForest")
library(randomForest)

# train a model for median house price
bos_rf <- randomForest(medv ~ ., data=Boston, ntree=100)

# save the model
saveRDS(bos_rf, "bos_rf.rds")
```

Scoring script for plumber

Now that we have the model, we also need a script to obtain predicted values from it given a set of inputs:

```
# save as bos_rf_score.R

bos_rf <- readRDS("bos_rf.rds")
library(randomForest)

#* @param df data frame of variables
#* @post /score
function(req, df)
{
    df <- as.data.frame(df)
    predict(bos_rf, df)
}
```

This is fairly straightforward, but the comments may require some explanation. They are plumber annotations that tell it to call the function if the server receives a HTTP POST request with the path /score, and query

parameter df. The value of the df parameter is then converted to a data frame, and passed to the randomForest predict method. For a fuller description of how Plumber works, see the Plumber website.[https://rplumber.io/]

Create a Dockerfile

Let's package up the model and the scoring script into a Docker image. A Dockerfile to do this is shown below. This uses the base image supplied by Plumber (trestletech/plumber), installs randomForest, and then adds the model and the above scoring script. Finally, it runs the code that will start the server and listen on port 8000.

example Dockerfile to expose a plumber service

FROM trestletech/plumber

install the randomForest package

RUN R -e 'install.packages(c("randomForest"))'

copy model and scoring script

RUN mkdir /data COPY bos_rf.rds /data COPY bos_rf_score.R /data WORKDIR /data

plumb and run server

EXPOSE 8000 ENTRYPOINT ["R", "-e",
"pr <- plumber::plumb('/data/bos_rf_score.R');
pr$run(host='0.0.0.0', port=8000)"]

```
docker build -t myimage .
```

Test it

#Push it to Google CLoud

Before pushing it to Google Cloud, we must quickly tag our image with the necessary information that Google Cloud needs. Make sure you replace with the correct project ID in the Google Cloud dashboard. Issue:

```
docker tag myimage gcr.io/virtual-machine-196412/myimage:v1
```

Bash in terminal

```
$docker login -u _json_key -p "$(cat OneDrive/7_DataScience/02_CloudComputing/X_Secrets/GCP/auth.json)"
```

$docker login -u_json_key -p "$(cat OneDrive/7_DataScience/02_CloudComputing/X_Secrets/GCP/auth.json)"
https://gcr.io

```
docker push gcr.io/virtual-machine-196412/myimage:v1
```

```
gcloud docker - push gcr.io/virtual-machine-196412/myimage:v1
```

Now if you head to your Google Cloud Console, and over to the Container Registry, you should see your new Docker image there. Good work!

We now need to get our image served in Kubernetes Engine. Kubernetes Engine is Google's hosted version of Kubernetes, which enables you to create a cluster of "nodes" to serve your containers among. To get started, create a cluster by following the prompts in the Kubernetes Engine dashboard. Then head to "workloads" and hit the "Deploy" button.

Make sure you click the "Select existing Google Container Registry image" button, and then choose your repository and tag (which will initially be v1). Click 'Done', give it a useful name, and then hit 'Deploy'.

In the panel on the right, you will see the option to 'Expose' your image to the internet using a load balancer. Go ahead and do this. You will then be provided with an internet-facing endpoint which you can test your deployment on. Congratulations, you have just deployed a docker image to Google Cloud using Kubernetes Engine!

Part 1 - Creating an R cluster

Using

```
library(googleKubernetesR)
```

Or gcloud

Using

```
library(googleKubernetesR)
```

Or gcloud

Follow the setup steps to authenticate with gcloud and kuberctl then create your cluster via:

This is a test. Here's some LaTeX{}:

gcloud container clusters create r-cluster –num-nodes=3 . . . or if using googleKubernetesR :

library(googleKubernetesR)

create a 3 node cluster called r-cluster with defaults

createCluster(projectId = gcp_get_project(), zone = "europe-west3-b") You'll need to wait a bit as it provisions all the VMs.

Most of the below will use the terminal/shell for working rather than R, but in the future a lot of this may be possible via googleKubernetesR within an R session.

Set up your shell to get the credentials for Kuberctl:

gcloud container clusters get-credentials r-cluster And we are all set to start setting up this cluster for R jobs! Here is a screenshot from the web UI of what it should look like: