Kubernetes with Containers and DevOps Workshop

Hands-on lab step-by-step

Aralık 2018

Kubernetes with Containers and DevOps hands-on lab step-by-step

## Abstract and learning objectives

Build a PoC to deliver a multi-tenant web app hosting solution leveraging Azure Container Service (AKS), Docker containers, and Linux nodes.

Attendees will be better able to deploy Docker-based applications and scale them with Azure Container Service and Kubernetes orchestration. In addition,

* Create and run a Docker Application
* Create and run a Jenkins server on an Azure Linux VM
* Deploy to the Azure Kubernetes Service (AKS) using Helm Charts
* Monitor and scale the application and test availability

## Overview

## This workshop will guide you through migrating an application from "on-premises" to containers running in Azure Kubernetes Service using Jenkins.

## The labs are based upon a node.js application that allows for voting on the Justice League Superheroes. Data is stored in MongoDB.

## Note: These labs are designed to run on a Linux CentOS VM running in Azure (jumpbox) along with Azure Cloud Shell. They can potentially be run locally on a Mac or Windows, but the instructions are not written towards that experience. ie - "You're on your own."

## Note: Since we are working on a jumpbox, note that Copy and Paste are a bit different when working in the terminal. You can use Shift+Ctrl+C for Copy and Shift+Ctrl+V for Paste when working in the terminal. Outside of the terminal Copy and Paste behaves as expected using Ctrl+C and Ctrl+V.

## Solution architecture

Below is a diagram of the solution architecture you will build in this lab. Please study this carefully, so you understand the whole of the solution as you are working on the various components.

The solution will use Azure Kubernetes Service (AKS) which means that the container cluster topology is provisioned according to the number of requested nodes. The proposed containers deployed to the cluster are illustrated below, with MongoDB remaining as a managed service:

## 

## Requirements

1. Microsoft Azure subscription must be pay-as-you-go or MSDN.
   1. Trial subscriptions will *not* work.
   2. You must have rights to create a service principal as discussed in Task 9: Create a Service Principal — and this typically requires a subscription owner to log in. You may have to ask another subscription owner to login to the portal and execute that step ahead of time if you do not have the rights.
   3. You must have enough cores available in your subscription to create the build agent and Azure Container Service cluster in Task 5: Create a build agent VM and Task 10: Create an Azure Container Service (AKS) cluster. You’ll need eight cores if following the exact instructions in the lab, more if you choose additional agents or larger VM sizes. If you execute the steps required before the lab, you will be able to see if you need to request more cores in your sub.
2. Local machine or a virtual machine configured with:
   1. A browser, preferably Chrome for consistency with the lab implementation tests
   2. Command prompt
      1. On Windows, you will be using Bash on Ubuntu on Windows, hereon referred to as WSL.
      2. On Mac, all instructions should be executed using bash in Terminal.
3. You will be asked to install other tools throughout the exercises.

**VERY IMPORTANT: You should be typing all of the commands as they appear in the guide, except where explicitly stated in this document. Do not try to copy and paste from Word to your command windows or other documents where you are instructed to enter information shown in this document. There can be issues with Copy and Paste from Word that result in errors, execution of instructions, or creation of file content.**

## Lab- Guide

## 0. [Setup Lab environment]

## 1. [Run app locally to test components]

## 2. [Create Docker images for apps and push to Azure Container Registry]

## 3. [Create an Azure Kubernetes Service (AKS) cluster]

## 4. [Deploy application to Azure Kubernetes Service]

## 5. [CI/CD Automation using Jenkins Pipeline]

## 6. [Kubernetes UI Overview]

## 7. [Operational Monitoring and Log Management]

## 8. [Application and Infrastructure Scaling]

## 9. [Upgrade an Azure Kubernetes Service (AKS) cluster]

## 10. [Update and Deploy New Version of Application]

## Self-guided

It is possible to use your own machine outside of the classroom. You will need the following in order to complete these labs:

* Azure subscription
* Linux, Mac, or Windows with Bash
* Jenkins
* Docker
* Azure CLI
* Visual Studio Code
* Helm
* Kubernetes CLI (kubectl)
* GitHub account and git tools

## Setup Lab environment

Use Prepare Lab Environment document.

https://github.com/comparexoss/cpx-oss-workshop/labs/PrepareLabEnvironment-OSS-2018.pdf

## Run app locally to test components

The lab files must be cloned to the local machine to complete all of the exercises for the day.

* Open a terminal on the jumpbox to bring up a command line
* Clone the Github repo via the command line

git clone https://github.com/comparexoss/cpx-oss-workshop.git

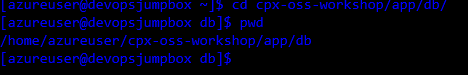


**Database layer - MongoDB**

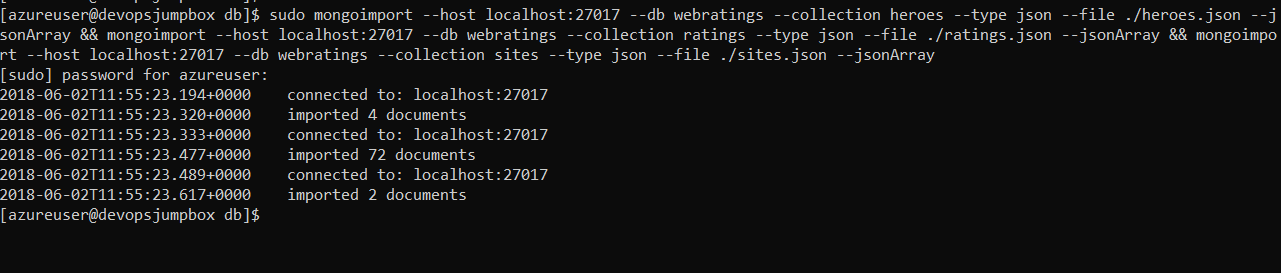
The underlying data store for the app is [MongoDB](https://www.mongodb.com/). It is already running. We need to import the data for our application.

* Import the data via the command line

cd ~/cpx-oss-workshop/app/db/



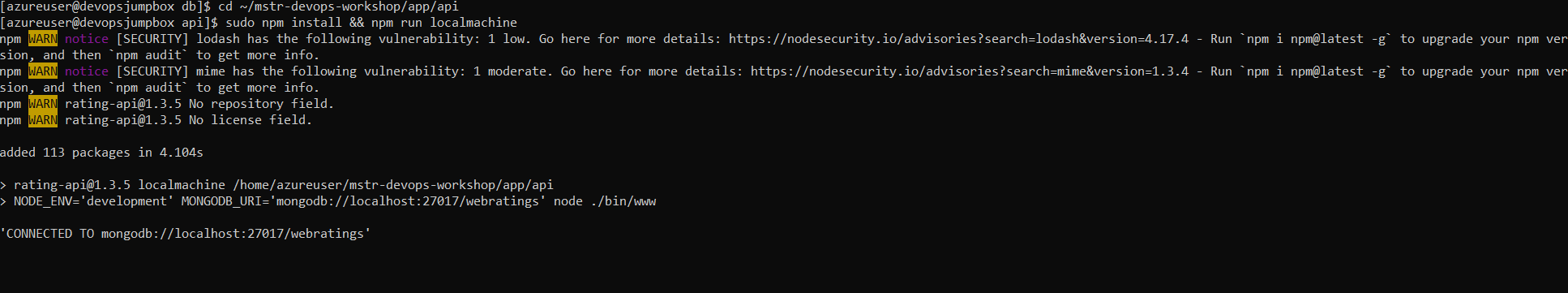
sudo mongoimport --host localhost:27017 --db webratings --collection heroes --type json --file ./heroes.json --jsonArray && mongoimport --host localhost:27017 --db webratings --collection ratings --type json --file ./ratings.json --jsonArray && mongoimport --host localhost:27017 --db webratings --collection sites --type json --file ./sites.json –-jsonArray



**API Application layer - Node.js**

The API for the app is written in javascript, running on [Node.js](https://nodejs.org/en/) and [Express](http://expressjs.com/)

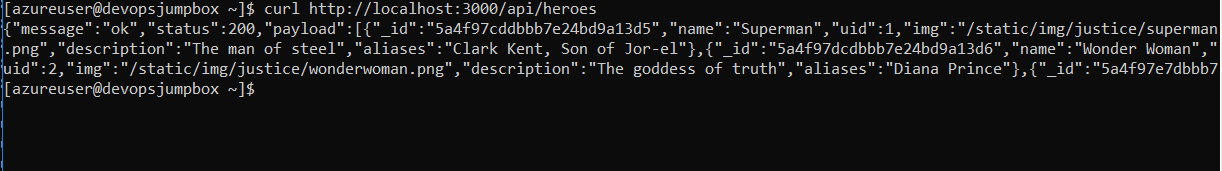
* Update dependencies and run app via node in the command line
  + 1. cd ~/cpx-oss-workshop/app/api
    2. npm install && npm run localmachine



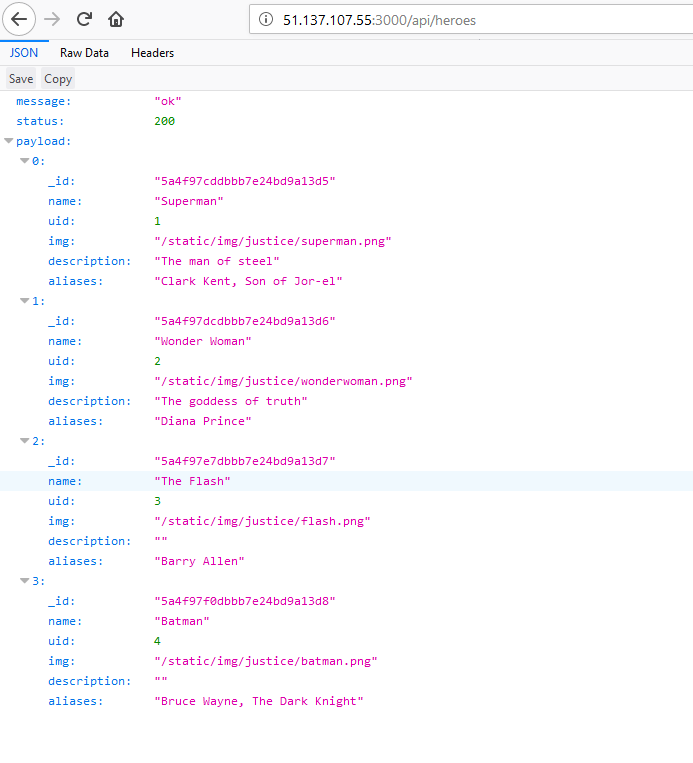
**Open a new terminal session** on the **jumpbox** and test the API (Do not forget to change {PUBLIC\_IPADDRESS\_VM} with your jumpbox vm’s ip address)

using curl in the command line

* + 1. ssh azureuser@{PUBLIC\_IPADDRESS\_VM}
    2. curl <http://localhost:3000/api/heroes>



or use the browser on your local computer and navigate to [http:// {PUBLIC\_IPADDRESS\_VM}:3000/api/heroes](http://localhost:3000/api/heroes)



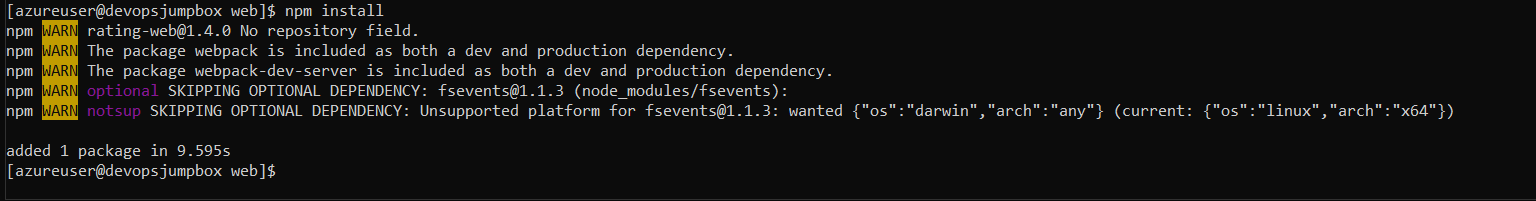
**Web Application layer - Vue.js, Node.js**

The web frontend for the app is written in [Vue.js](https://vuejs.org/Vue), running on [Node.js](https://nodejs.org/en/) with [Webpack](https://webpack.js.org/)

* Continue new Terminal session
* Update dependencies and run app via node

cd ~/cpx-oss-workshop/app/web/

npm install && npm run localmachine



* Test web locally

use the browser on the jumpbox and navigate to http://{PUBLIC\_IPADDRESS\_VM}:8080

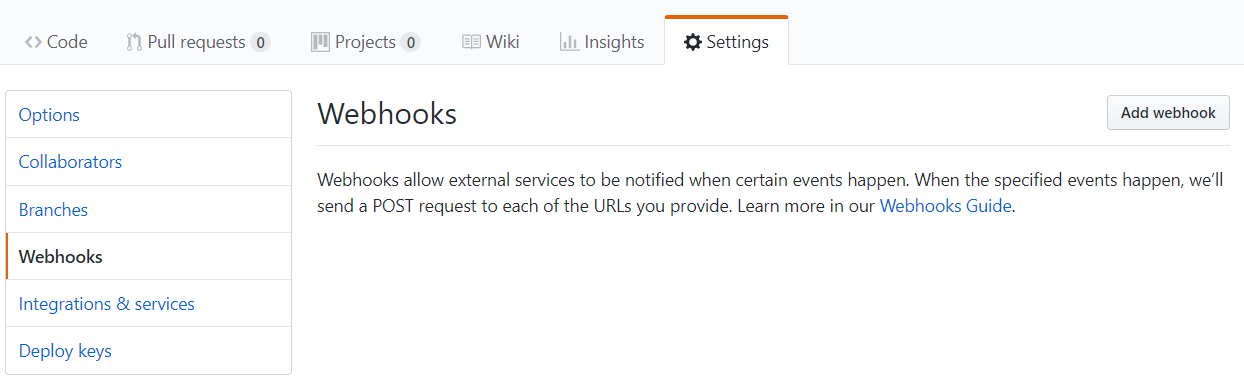
**Clean-up**

* Close the web and api apps in the terminal windows by hitting ctrl-c in each of the corresponding terminal windows

**Create GitHub webhook**

To configure the integration with GitHub, open the <https://github.com/comparexoss/cpx-oss-workshop> from sample repo. To fork the repo to your own GitHub account, select the Fork button in the top right-hand corner.

Once you fork the repo follow these steps to create web hook.

* Go to the settings page of your repository or organization. From there, click **Webhooks**, then **Add webhook**.
* Typehttp://{PUBLIC\_IPADDRESS\_VM}:8081/github-webhook/ into the Payload URL field. DO NOT forget to change PUBLIC\_IPADDRESS\_VM with your jumpbox public ip address.
* Choose **application/json** from the Content Type field.
* Leave blank the Secret field.
* Click **Send me everything** to trigger all events.
* Once you finished, be sure that you have clicked **Active** to enable this webhook and Click **Add webhook**.
  1. Create Docker images for apps and push to Azure Container Registry

**Build Container Images**

For the containers, the Dockerfiles are provided.

**Web Container**

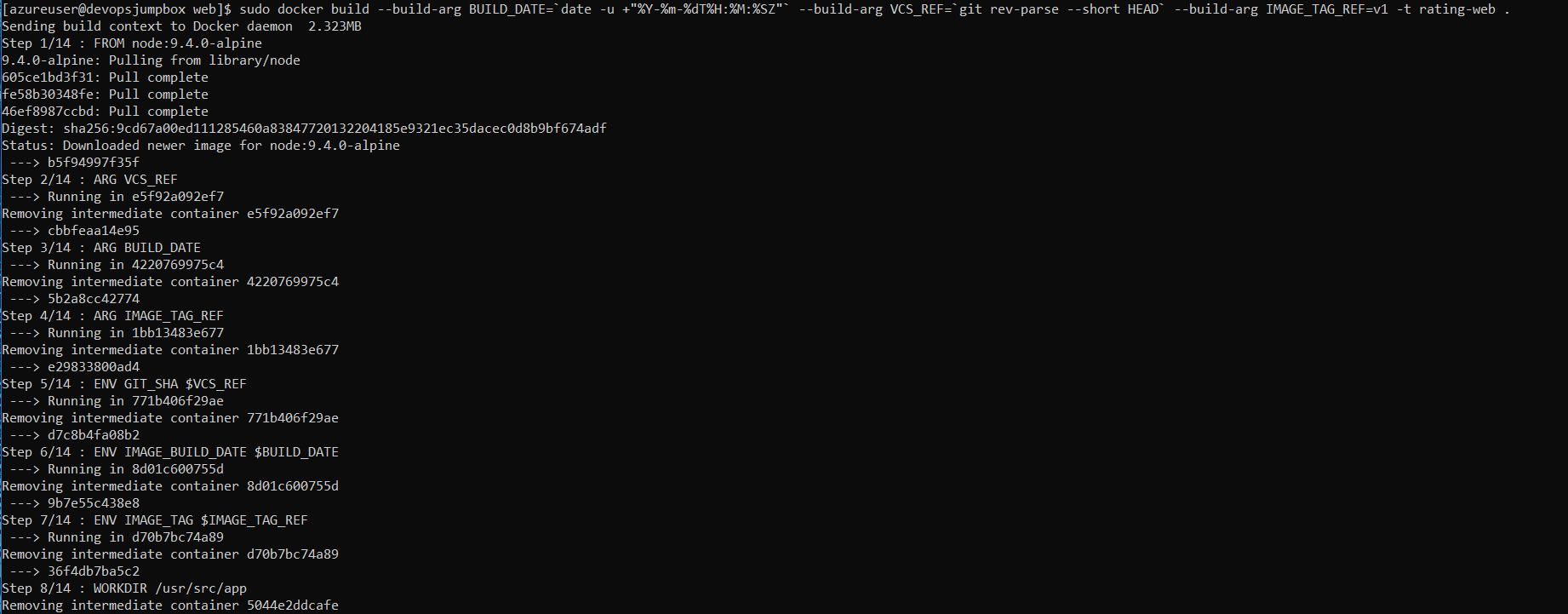
1. Change Directory to web app directory.
   * In the cd ~/cpx-oss-workshop/app/web/ directory,

cd ~/cpx-oss-workshop/app/web

1. Create a container image for the node.js Web app. Ignore npm warnings.

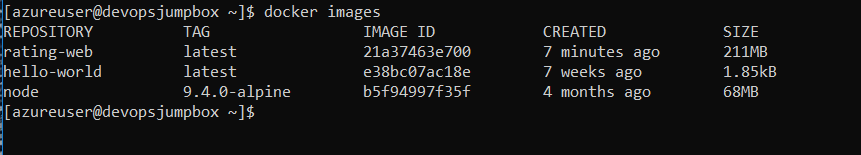
From bash shell:

sudo docker build --build-arg BUILD\_DATE=`date -u +"%Y-%m-%dT%H:%M:%SZ"` --build-arg VCS\_REF=`git rev-parse --short HEAD` --build-arg IMAGE\_TAG\_REF=v1 -t rating-web .



Validate image was created with

docker images

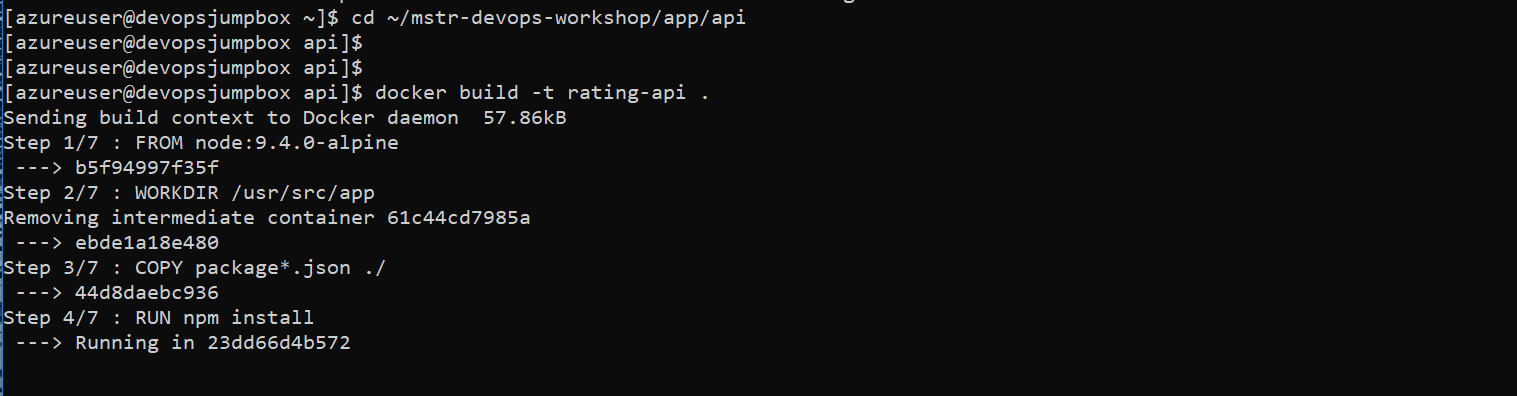


**API Container**

1. Create a container image for the node.js API app. And ignore npm warnings.

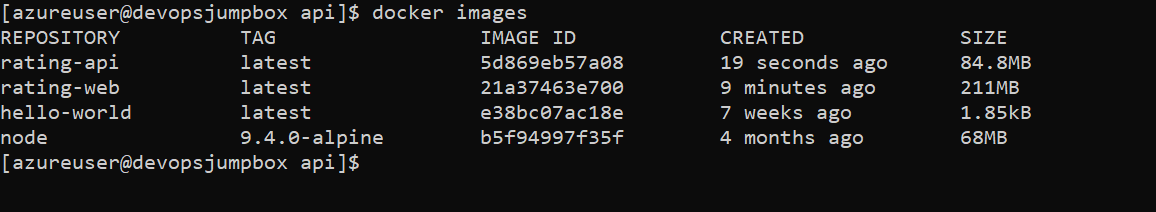
cd ~/cpx-oss-workshop/app/api

docker build -t rating-api .



Validate image was created with

docker images

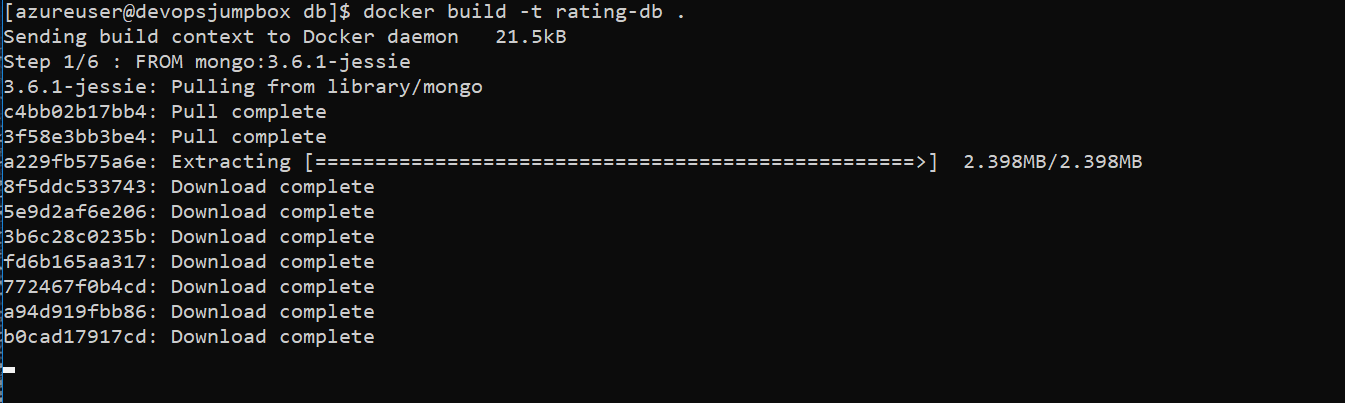


**MongoDB Container**

1. Create a MongoDB image with data files

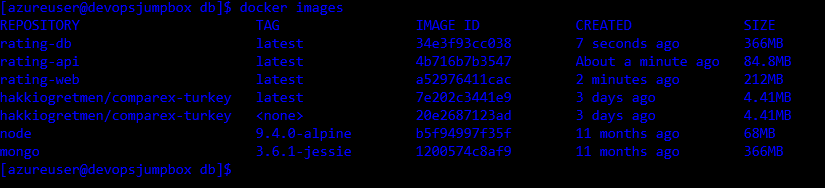
cd ~/cpx-oss-workshop/app/db

docker build -t rating-db .

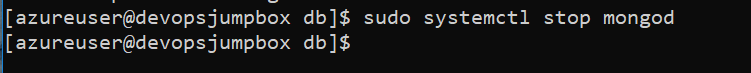


1. Validate image was created with

docker image ls



sudo systemctl stop mongod

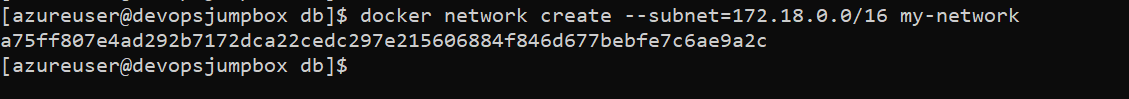


**Run Containers**

**Setup Docker Network**

Create a docker bridge network to allow the containers to communicate internally.

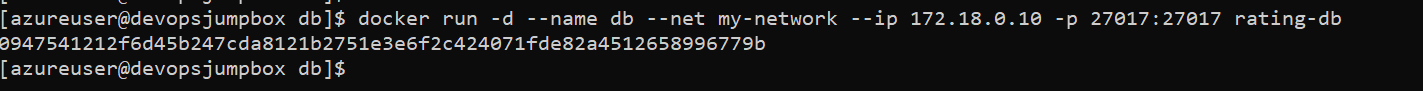
docker network create --subnet=172.18.0.0/16 my-network



**MongoDB Container**

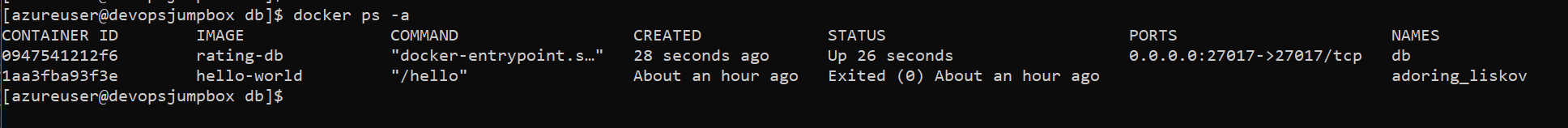
1. Run mongo container

docker run -d --name db --net my-network --ip 172.18.0.10 -p 27017:27017 rating-db



Validate by running

docker ps -a



1. Import data into database. Attach to the container by running bash process on it.

docker exec -it db bash

You will have a prompt inside the mongo container. From that prompt, run the import script which will import the documents to collection

./import.sh

root@61f9894538d0:/# **./import.sh**

2018-01-10T19:26:07.746+0000 connected to: localhost

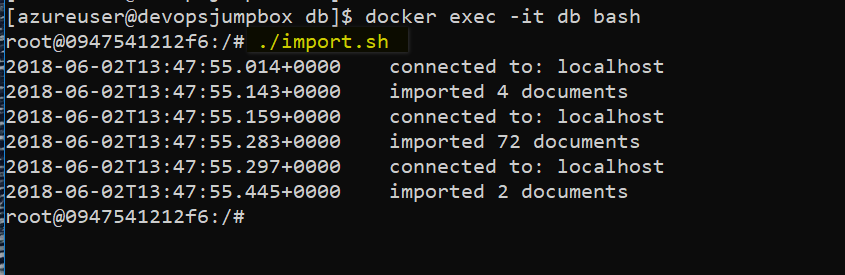
2018-01-10T19:26:07.761+0000 imported 4 documents

2018-01-10T19:26:07.776+0000 connected to: localhost

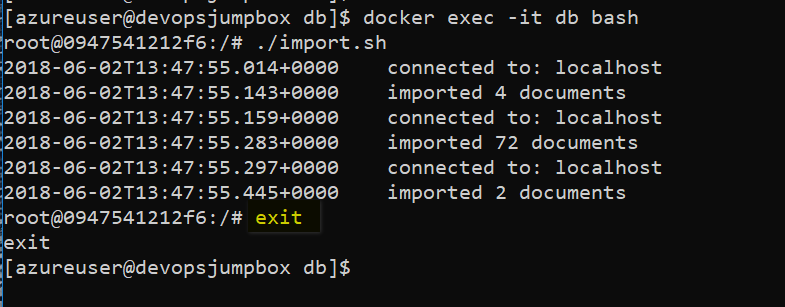
2018-01-10T19:26:07.787+0000 imported 72 documents

2018-01-10T19:26:07.746+0000 connected to: localhost

2018-01-10T19:26:07.761+0000 imported 2 documents



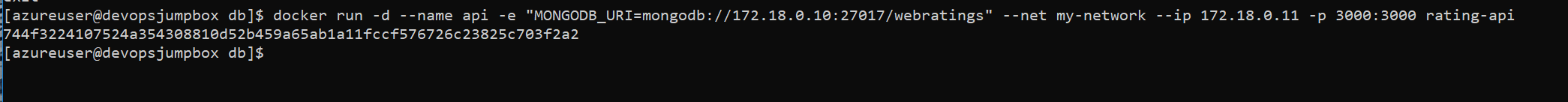
1. Type exit to exit out of container. Typing exit will kill the bash process we have created. Container will be still alive after exiting.



**API Container**

1. Run api app container

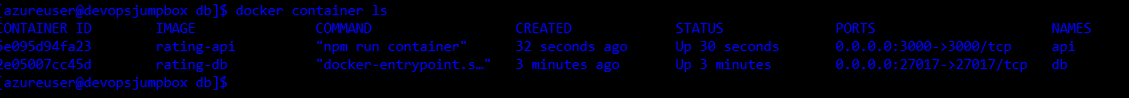
docker run -d --name api -e "MONGODB\_URI=mongodb://172.18.0.10:27017/webratings" --net my-network --ip 172.18.0.11 -p 3000:3000 rating-api



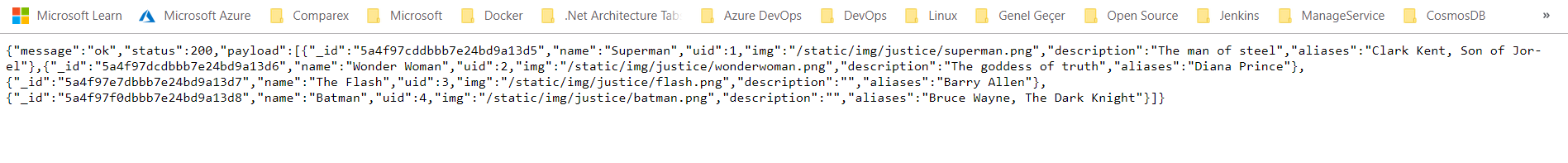
Note that environment variables are used here to direct the api app to mongo.

Validate by running (same command with docker ps)

docker container ls



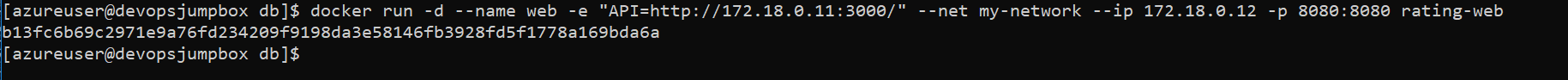
1. Test api app by browsing to http://{PUBLIC\_IPADDRESS\_VM}:3000/api/heroes



**Web Container**

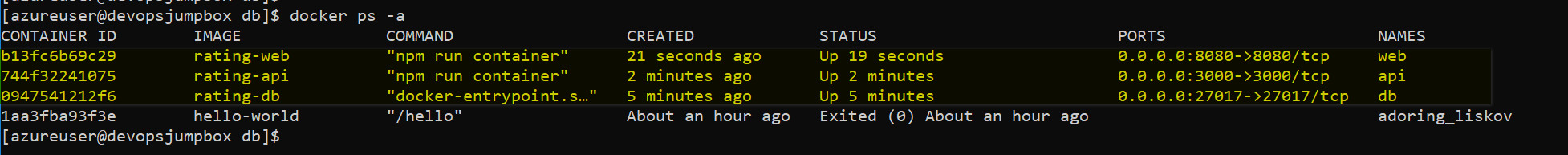
1. Run web app container

docker run -d --name web -e "API=http://172.18.0.11:3000/" --net my-network --ip 172.18.0.12 -p 8080:8080 rating-web

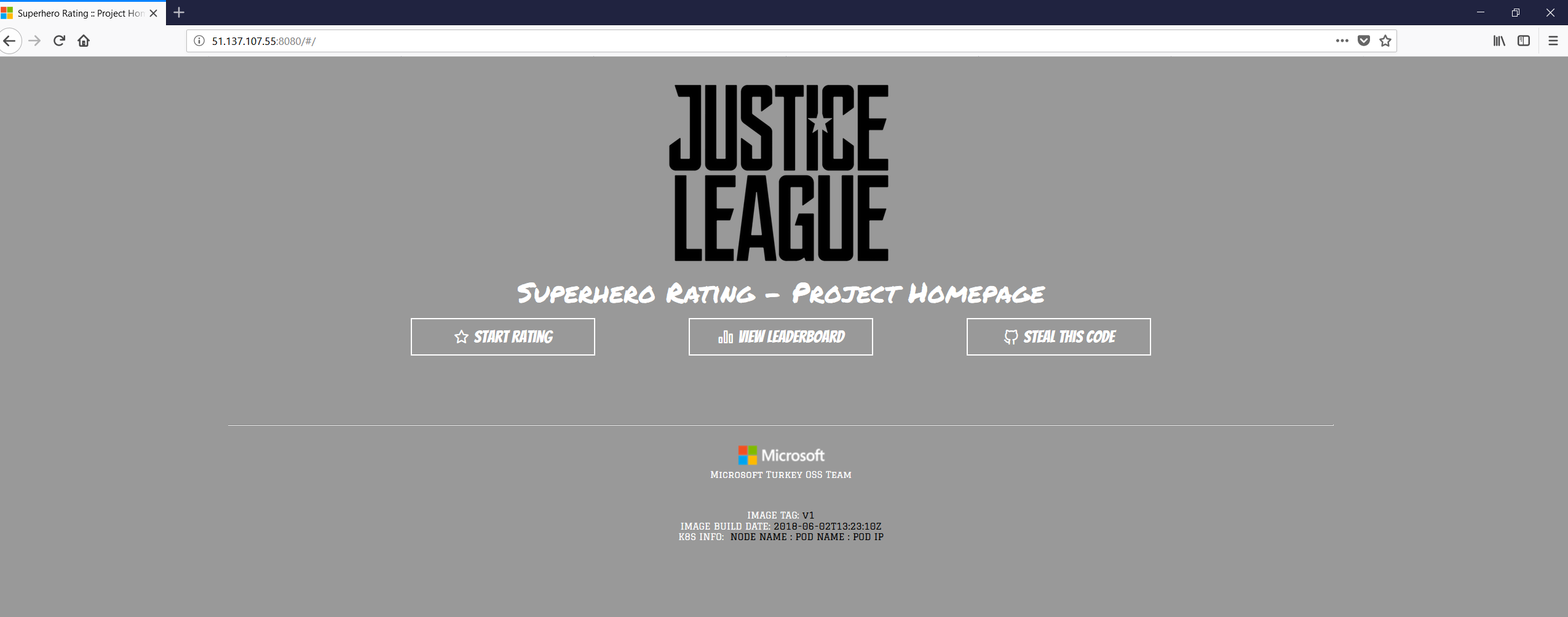


Validate by running

docker ps -a



1. Test web app by browsing to http://{PUBLIC\_IPADDRESS\_VM}:8080

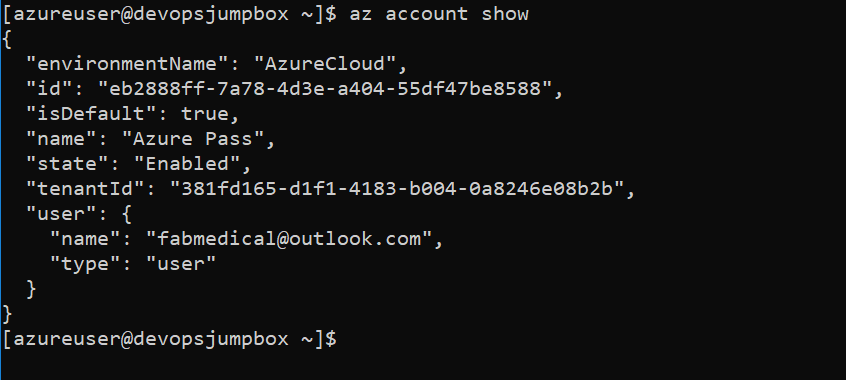


**Azure Container Registry (ACR)**

Now that we have container images for our application components, we need to store them in a secure, central location. In this lab we will use Azure Container Registry for this.

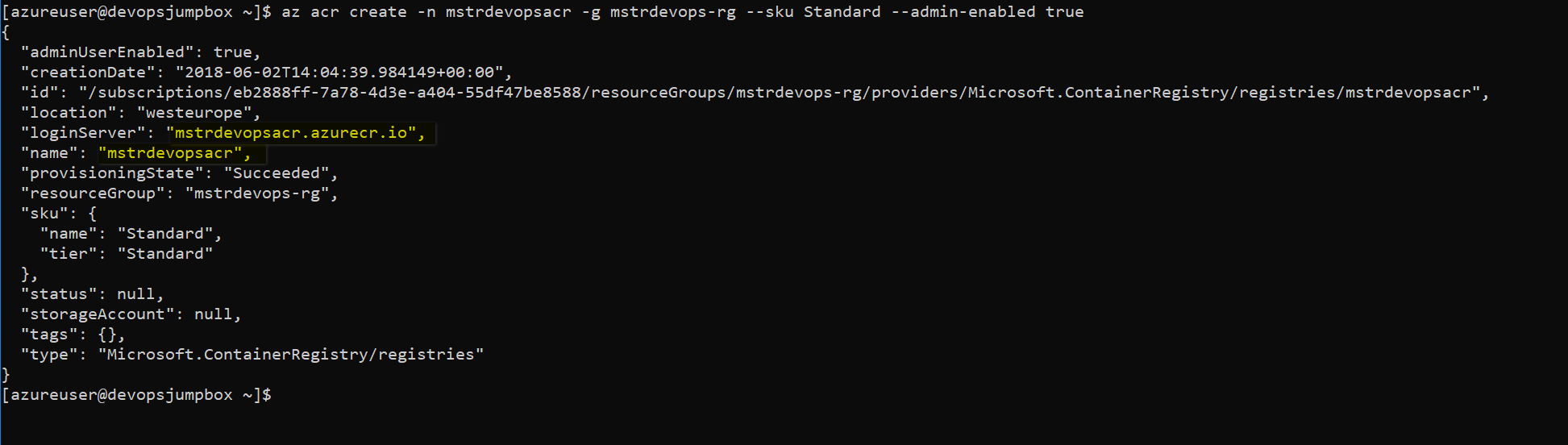
**Create Azure Container Registry instance**

az account show



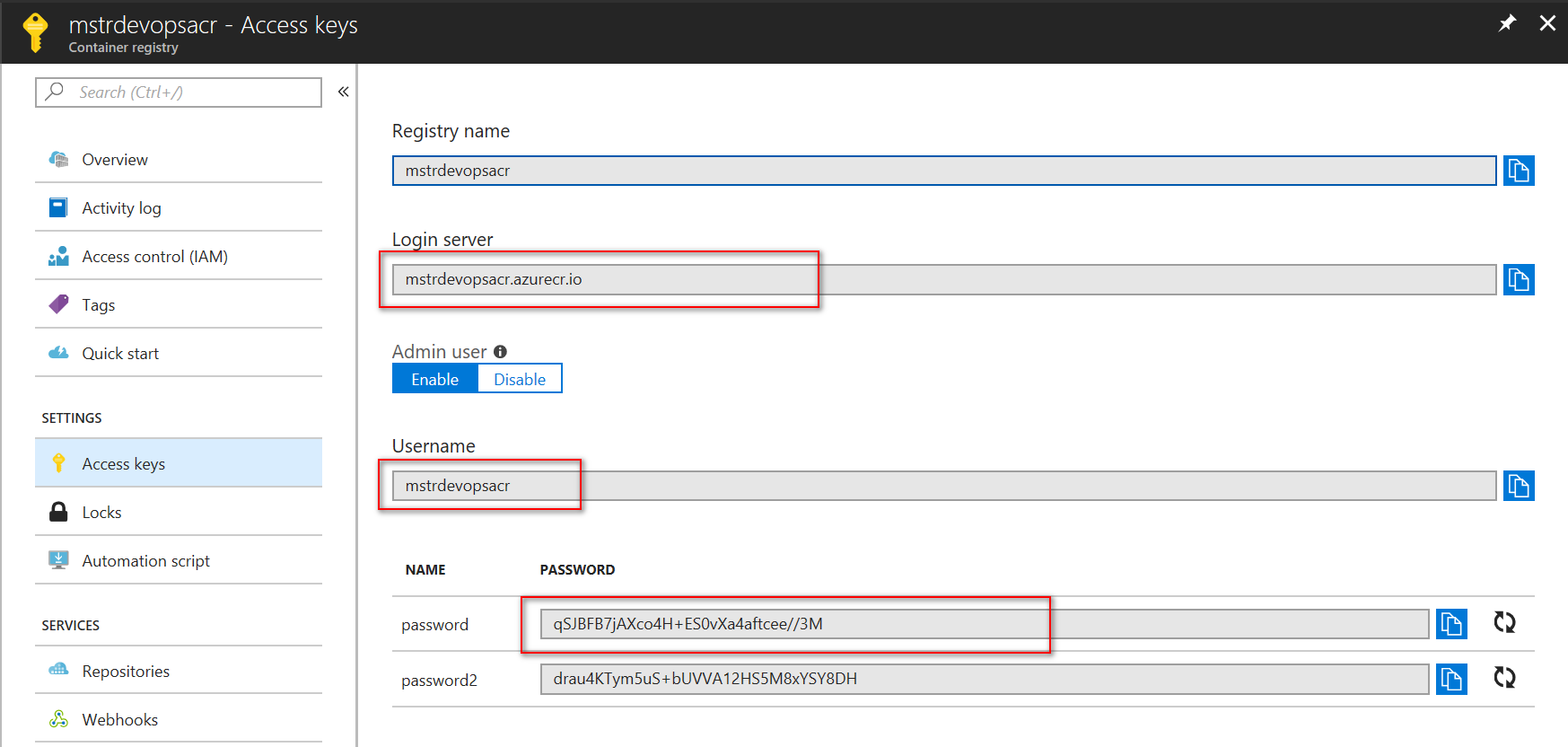
To create Azure Container Registry run the command. Do not forget to change acr name (You may use mstrdevopsacr{yourname} naming format).

az acr create -n mstrdevopsacr -g mstrdevops-rg --sku Standard --admin-enabled true

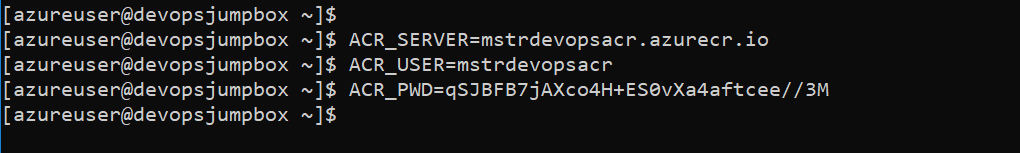


**Login to your ACR with Docker**

1. Browse to your Container Registry in the Azure Portal
2. Click on "Access keys"

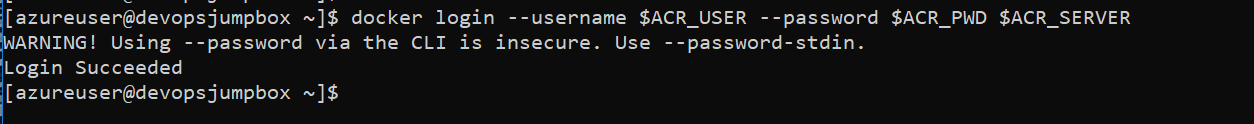


1. Make note of the "Login server", "Username" and "password"
2. Set each value to a variable as shown below.
3. Login in your Bash shell:
   * 1. # set these values to yours
     2. ACR\_SERVER=mstrdevopsacr132.azurecr.io
     3. ACR\_USER=mstrdevopsacr132
     4. ACR\_PWD=/GYkfWoZYIUKjWzgKSkXDxNw0pECLeHQ



docker login --username $ACR\_USER --password $ACR\_PWD $ACR\_SERVER

\*\* Ignore “using --password via the CLI is insecure” warning.



**Tag images with ACR server and repository**

# Be sure to replace the login server value

docker tag rating-db $ACR\_SERVER/mstrdevopsworkshop/rating-db:v1

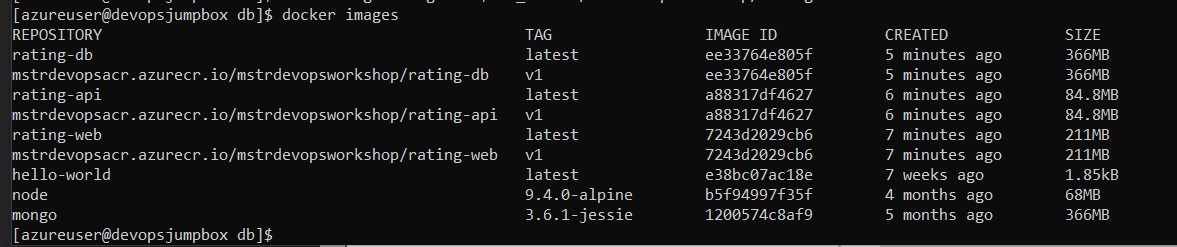
docker tag rating-api $ACR\_SERVER/mstrdevopsworkshop/rating-api:v1

docker tag rating-web $ACR\_SERVER/mstrdevopsworkshop/rating-web:v1



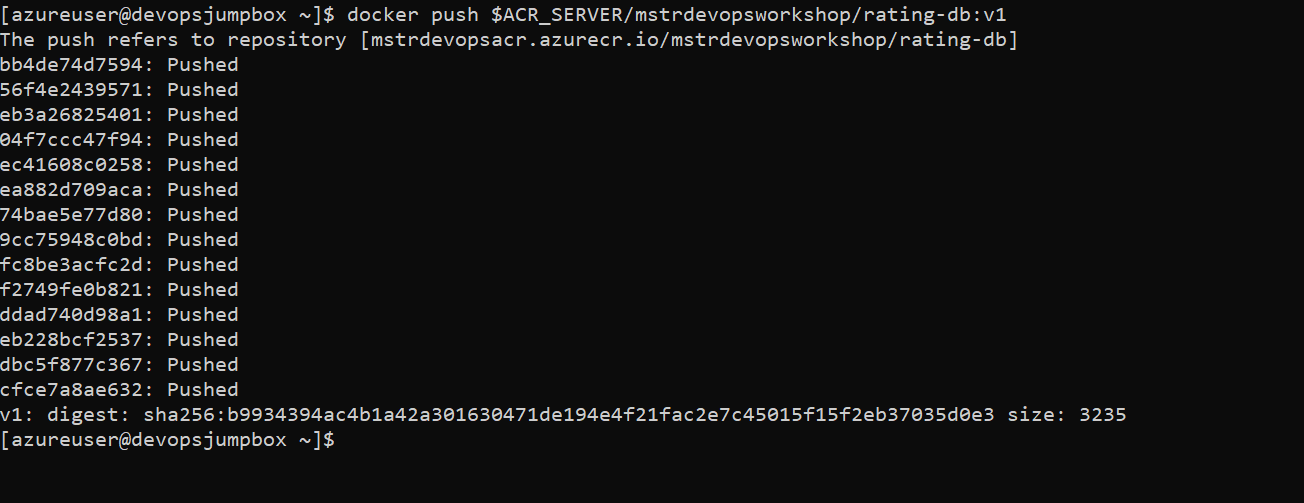
Validate images

docker images



**Push images to registry**

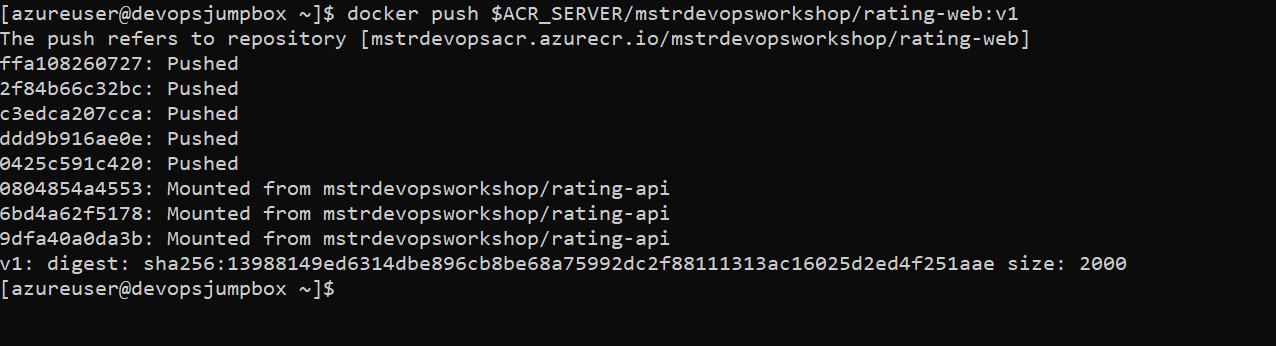
docker push $ACR\_SERVER/mstrdevopsworkshop/rating-db:v1



docker push $ACR\_SERVER/mstrdevopsworkshop/rating-api:v1

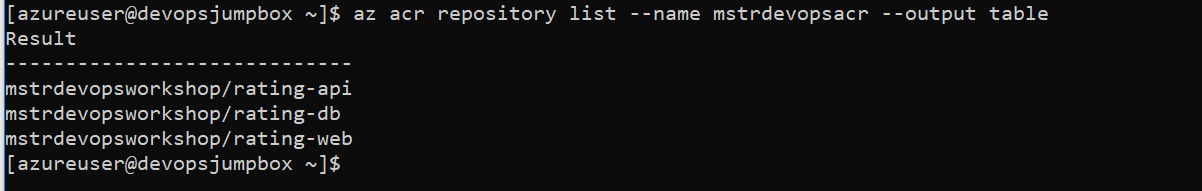


docker push $ACR\_SERVER/mstrdevopsworkshop/rating-web:v1



**Validate images in Azure**

az acr repository list --name mstrdevopsacr --output table



**You can see container images in Container Repositories**

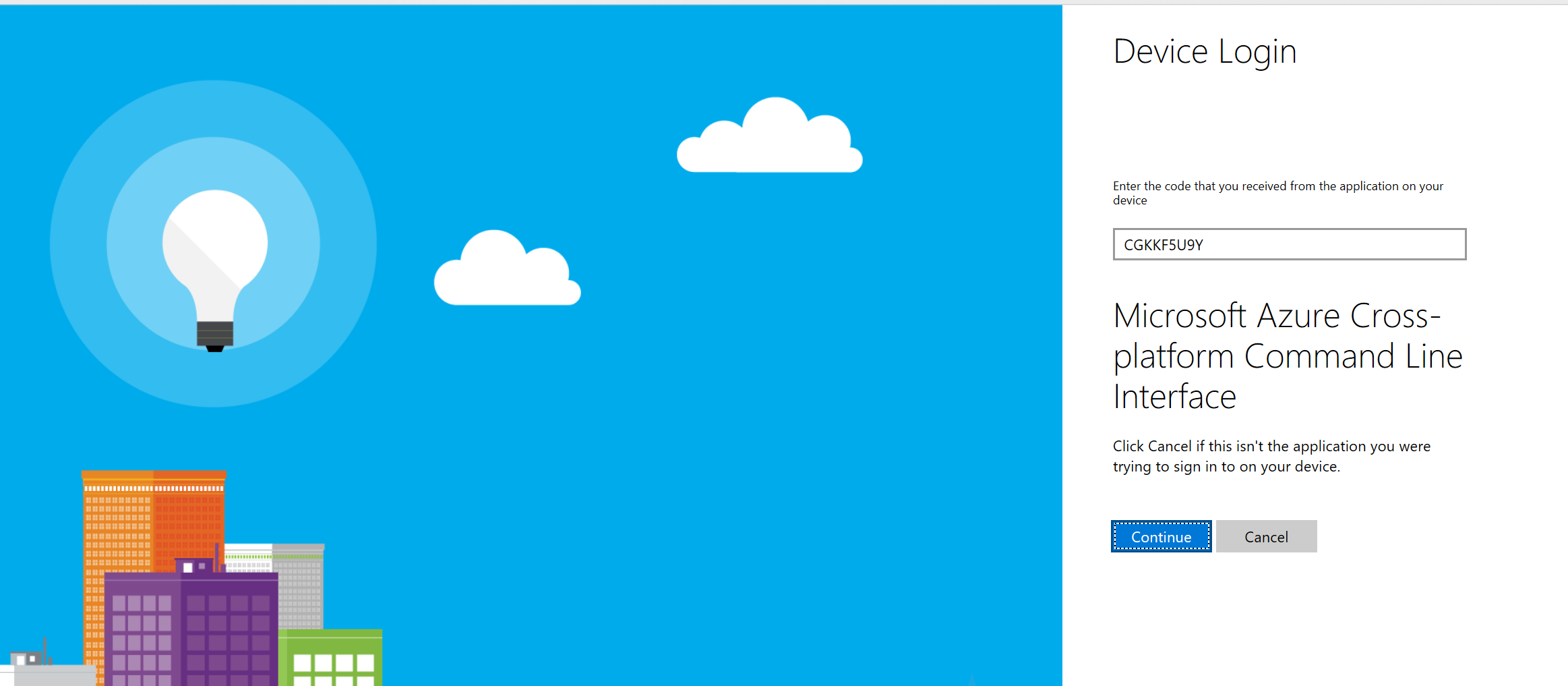


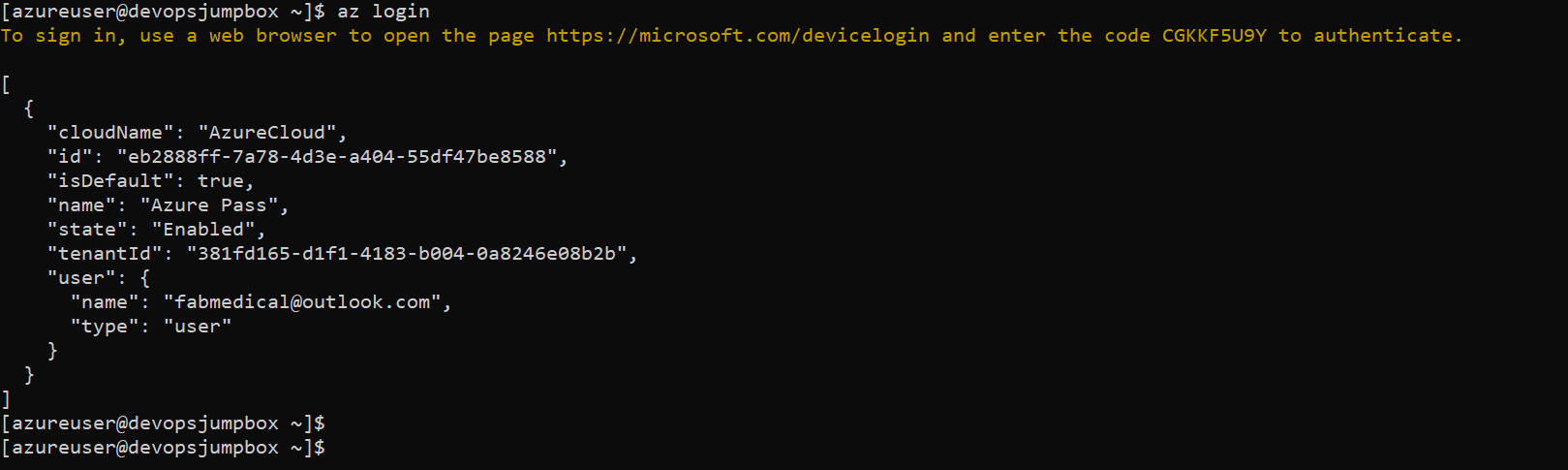
## Create an Azure Kubernetes Service (AKS) cluster

Open your Bash Shell and login to your Azure subscription using the az cli

az login

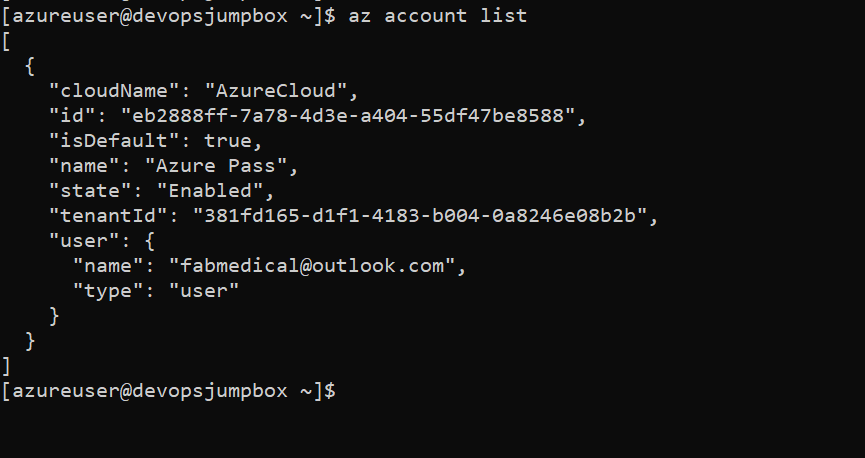






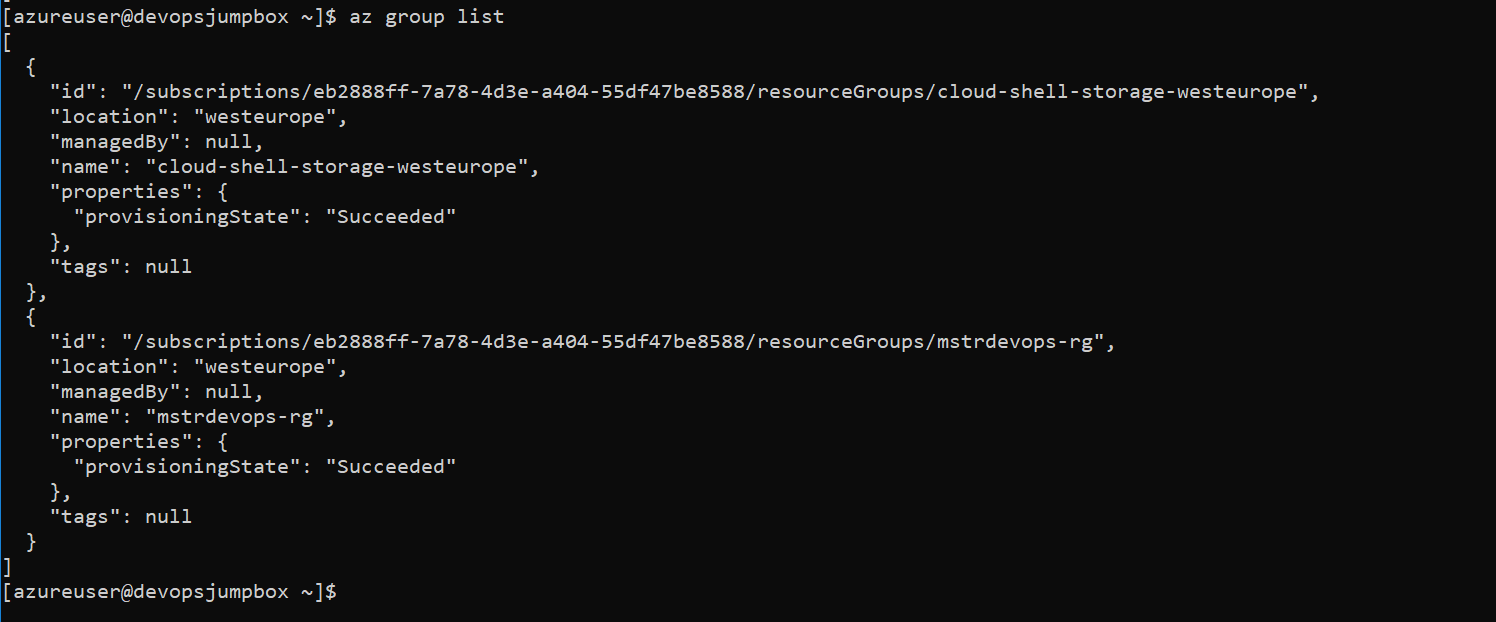
Verify your subscription is correctly selected as the default

az account list



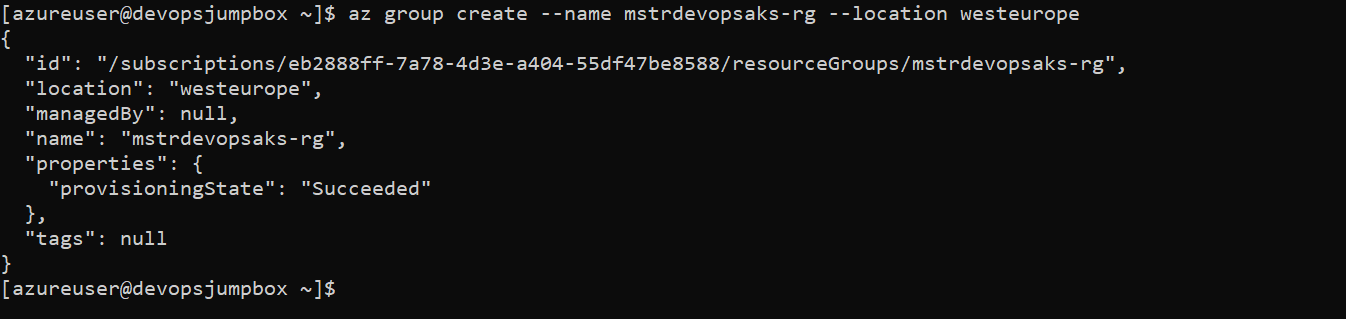
Find your RG name

az group list

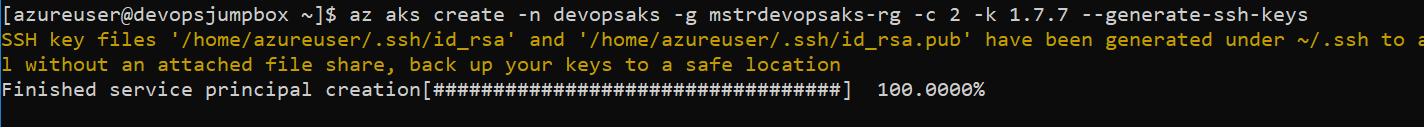


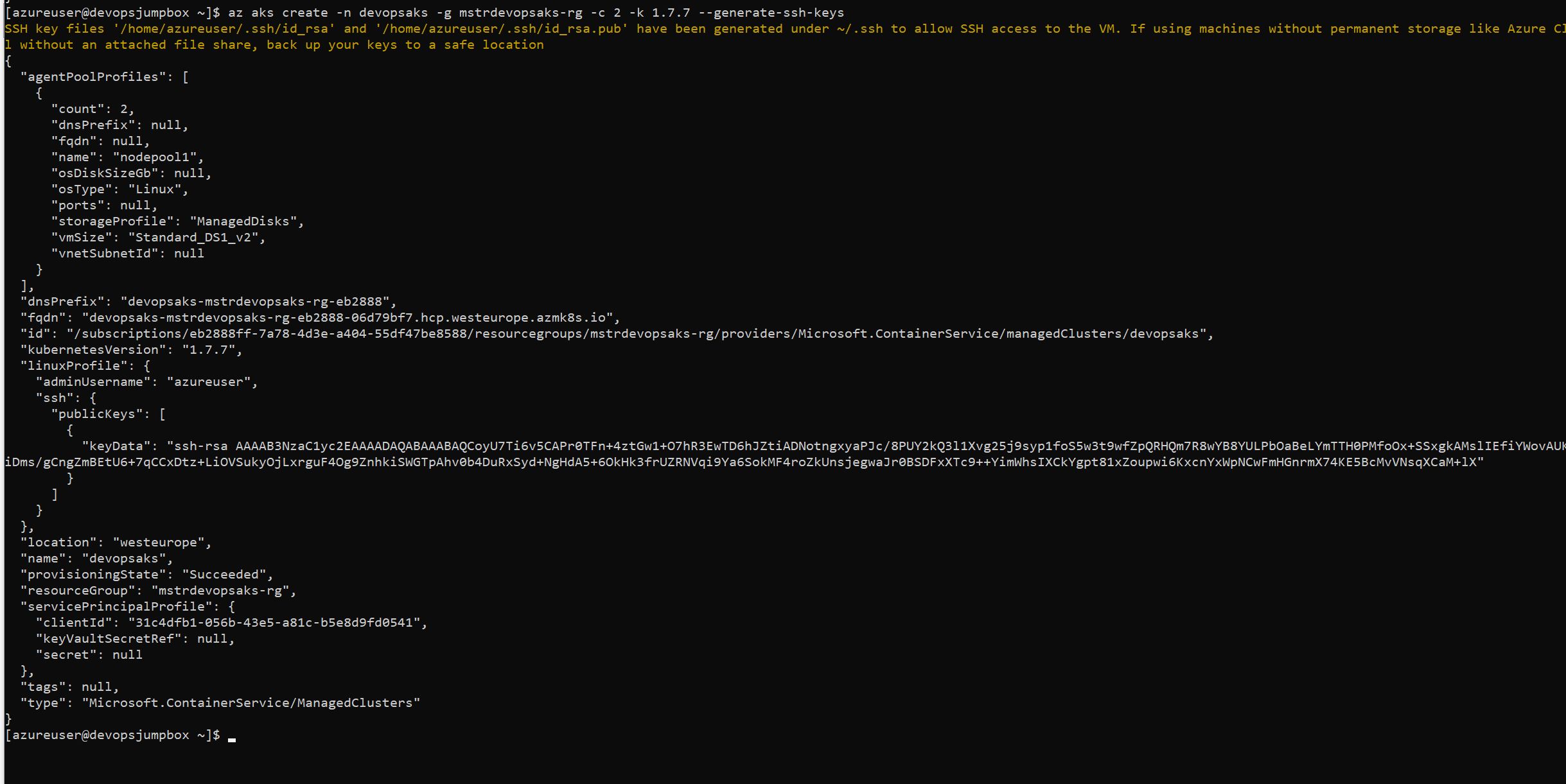
Create your AKS cluster in the resource group created above with 2 nodes, targeting Kubernetes version 1.8.6

az group create --name mstrdevopsaks-rg --location westeurope

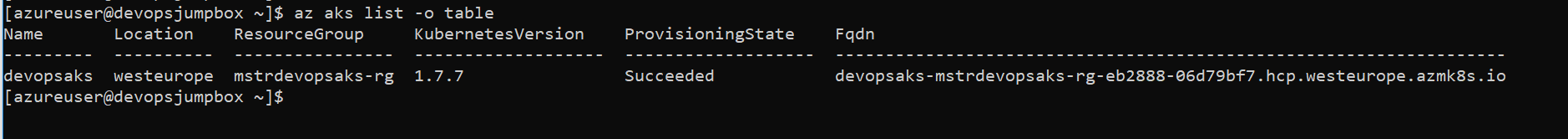


az aks create -n devopsaks -g mstrdevopsaks-rg -c 2 --generate-ssh-keys

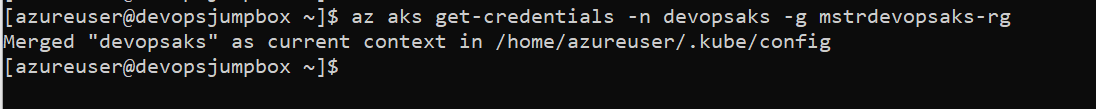




az aks list -o table

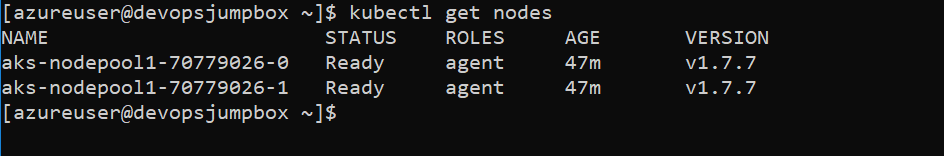


az aks get-credentials -n devopsaks -g mstrdevopsaks-rg

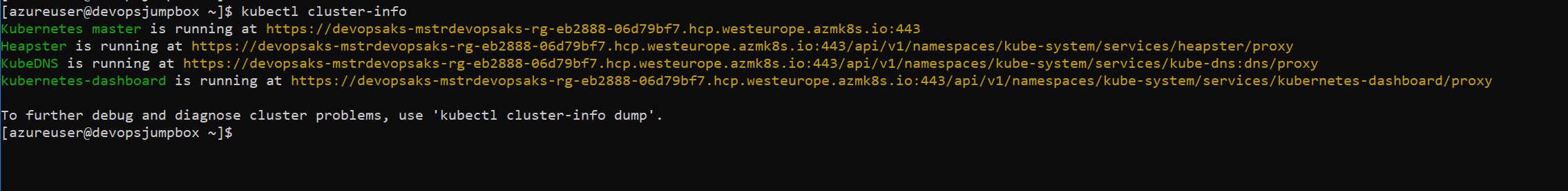


Verify you have API access to your new AKS cluster

kubectl get nodes

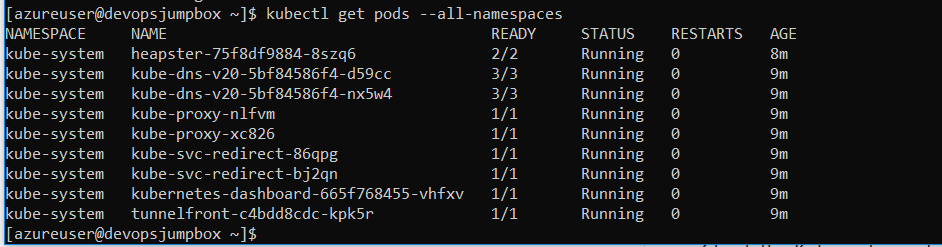


kubectl cluster-info



You should now have a Kubernetes cluster running with 2 nodes. You do not see the master servers for the cluster because these are managed by Microsoft. The Control Plane services which manage the Kubernetes cluster such as scheduling, API access, configuration data store and object controllers are all provided as services to the nodes.

kubectl get pods --all-namespaces



## Deploy application to Azure Kubernetes Service

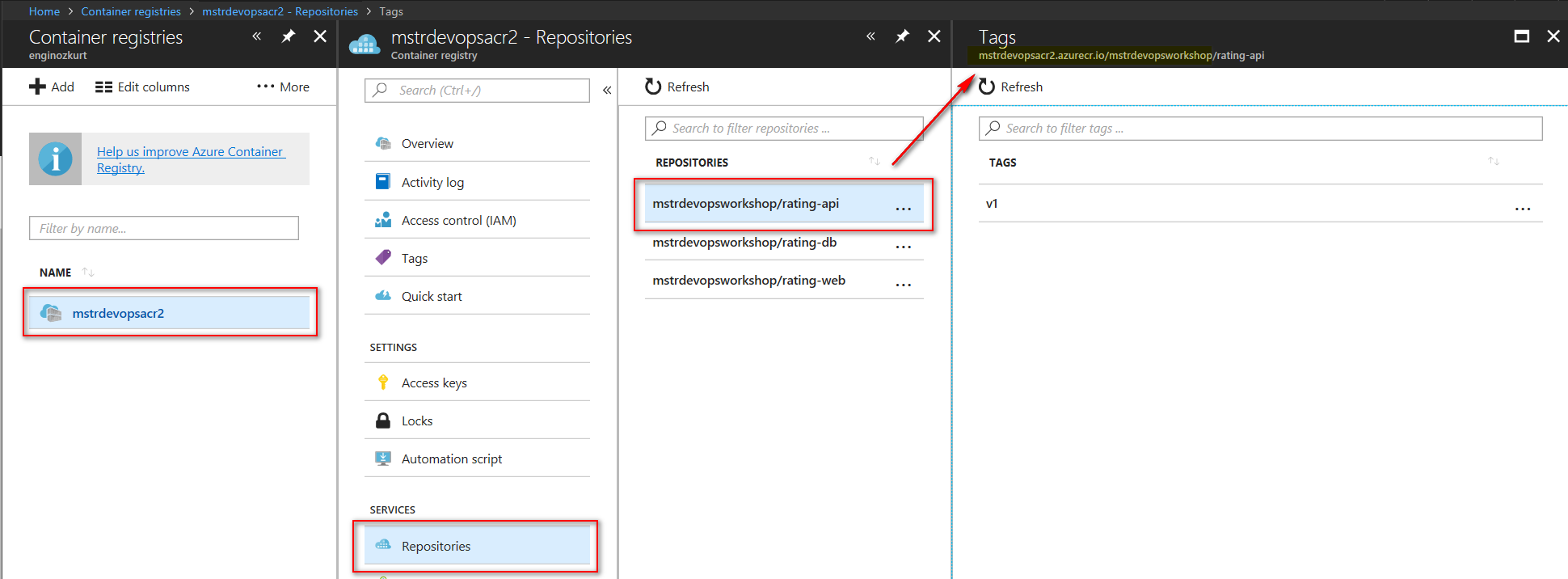
**Review/Edit the YAML Config Files**

* In VS Code (or vi), open the helper-files directory and review the yaml files:

cd ~/cpx-oss-workshop/labs/helper-files



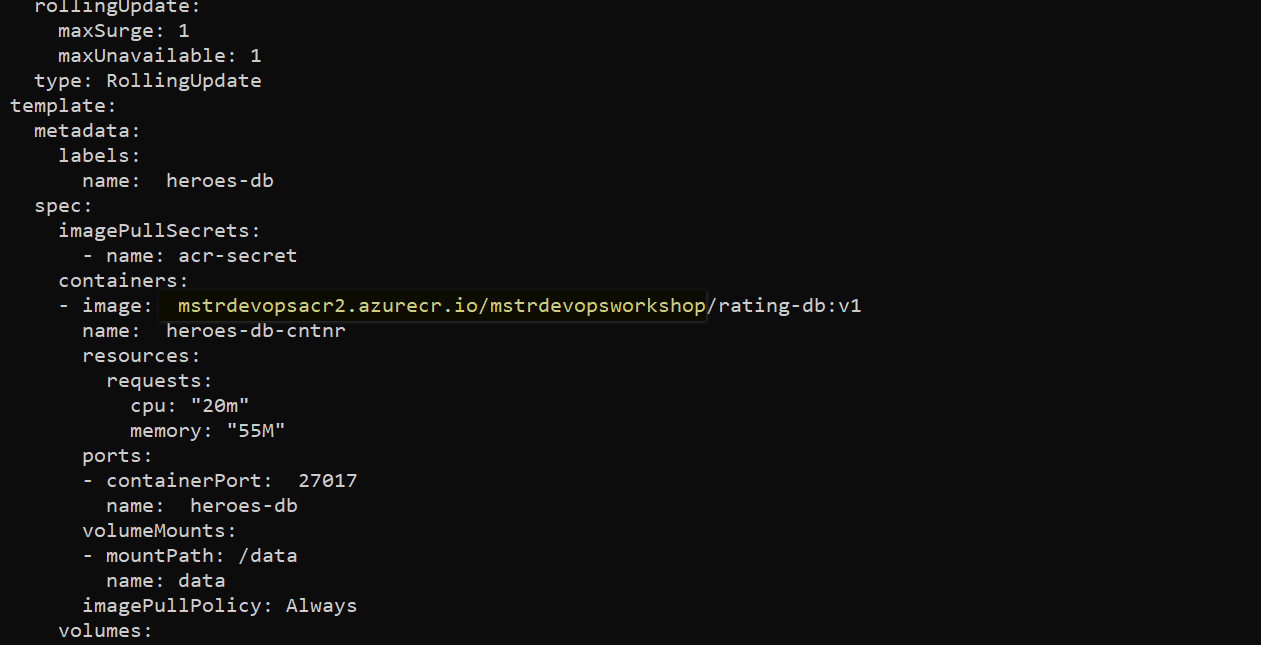
* Note the environment variables that direct each app to other services.
* Update the yaml file for the proper container image names.

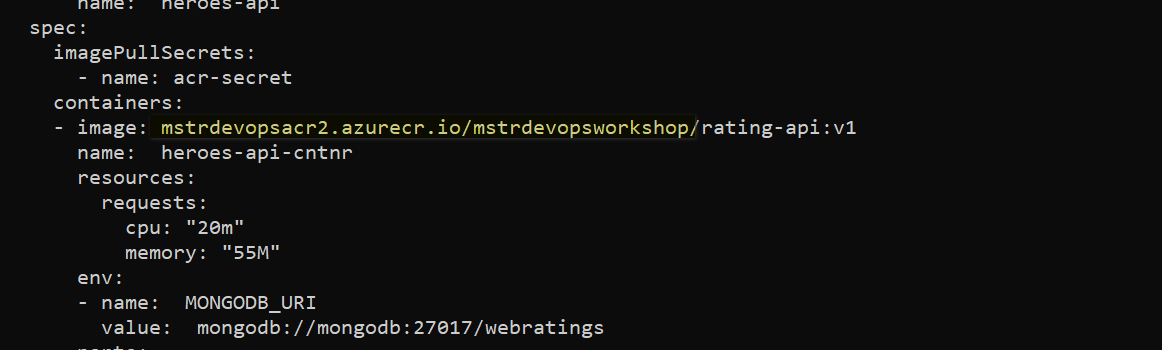


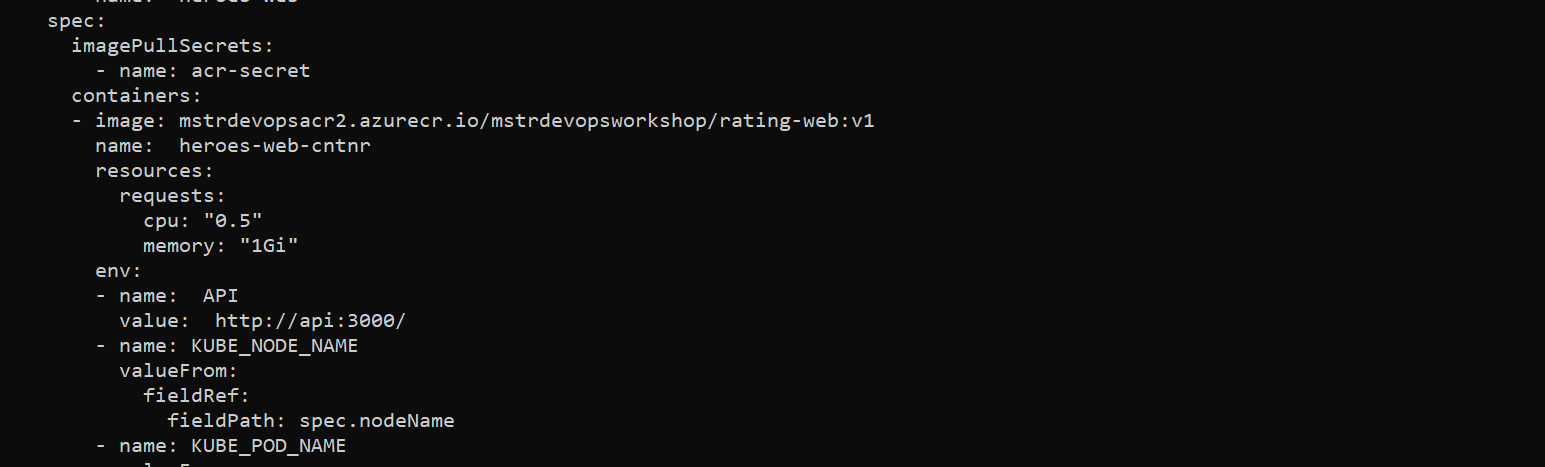
sudo vi heroes-db.yaml

sudo vi heroes-web-api.yaml

* + You will need to replace the <login server> with the ACR login server created in Lab 2.
  + Repeat this **THREE** times in the heroes yaml files (for the web, api, and db images). Example:
  + spec:
  + containers:
  + - image: mstrdevopsacr.azurecr.io/mstrdevopsworkshop/rating-web:v1
  + name: heroes-web-cntnr







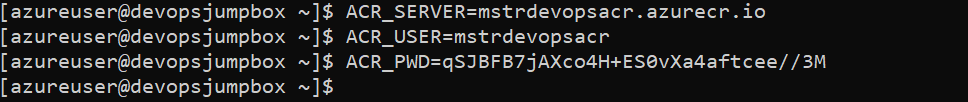
**Setup AKS with access to Azure Container Registry**

There are a few ways that AKS clusters can access your private Azure Container Registry. Generally the service account that kubernetes utilizes will have rights based on its Azure credentials. In our lab config, we must create a secret to allow this access.

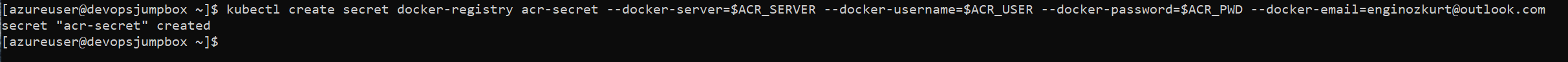
ACR\_SERVER=mstrdevopsacr.azurecr.io

ACR\_USER=mstrdevopsacr

ACR\_PWD=qSJBFB7jAXco4H+ES0vXa4aftcee//3M



kubectl create secret docker-registry acr-secret --docker-server=$ACR\_SERVER --docker-username=$ACR\_USER --docker-password=$ACR\_PWD --docker-email=comparexoss@gmail.com

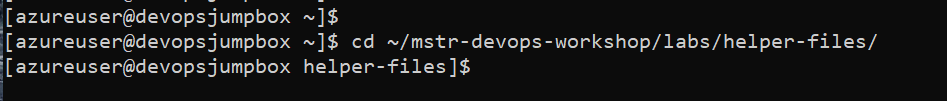


Note: You can review the heroes-db.yaml and heroes-web-api.yaml to see where the imagePullSecrets are configured.

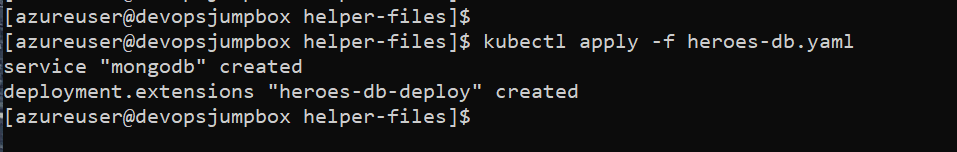
**Deploy database container to AKS**

* Use the kubectl CLI to deploy each app

cd ~/cpx-oss-workshop/labs/helper-files/

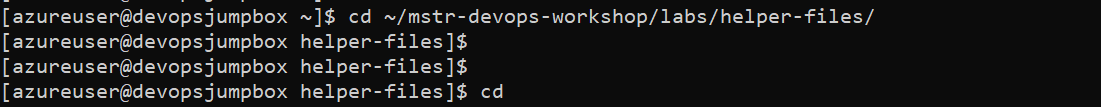


kubectl apply -f heroes-db.yaml

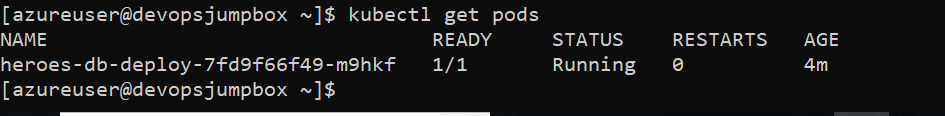


* Get mongodb pod name. Wait your pod to get ready.

cd



kubectl get pods



MONGO\_POD=heroes-db-deploy-2357291595-k7wjk



* Import data into MongoDB using script
* # ensure the pod name variable is set to your pod name
* # once you exec into pod, run the `**import.sh`** script

kubectl exec -it $MONGO\_POD bash

root@heroes-db-deploy-2357291595-xb4xm:/# **./import.sh**

2018-01-16T21:38:44.819+0000 connected to: localhost

2018-01-16T21:38:44.918+0000 imported 4 documents

2018-01-16T21:38:44.927+0000 connected to: localhost

2018-01-16T21:38:45.031+0000 imported 72 documents

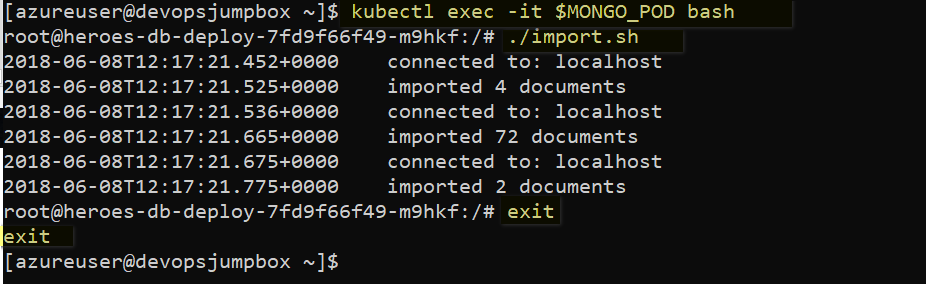
2018-01-16T21:38:45.040+0000 connected to: localhost

2018-01-16T21:38:45.152+0000 imported 2 documents

root@heroes-db-deploy-2357291595-xb4xm:/# exit

# be sure to exit pod as shown above

exit



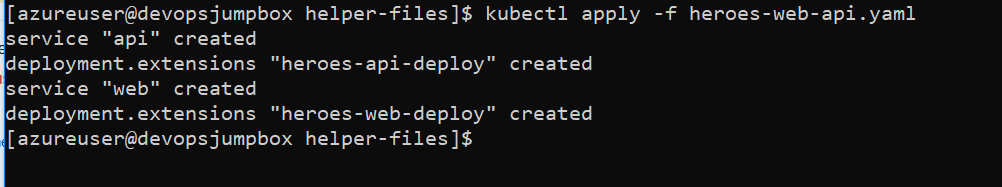
**Deploy the web and api containers to AKS**

* Use the kubectl CLI to deploy each app

cd ~/cpx-oss-workshop/labs/helper-files



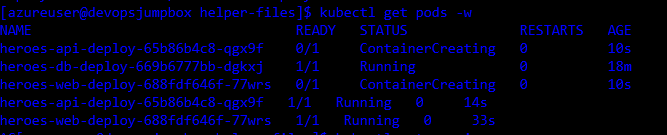
kubectl apply -f heroes-web-api.yaml



**Validate**

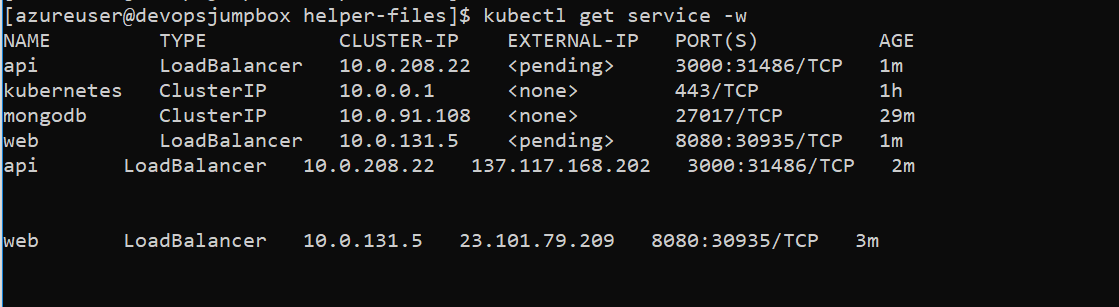
Check to see if pods are running in your cluster

kubectl get pods

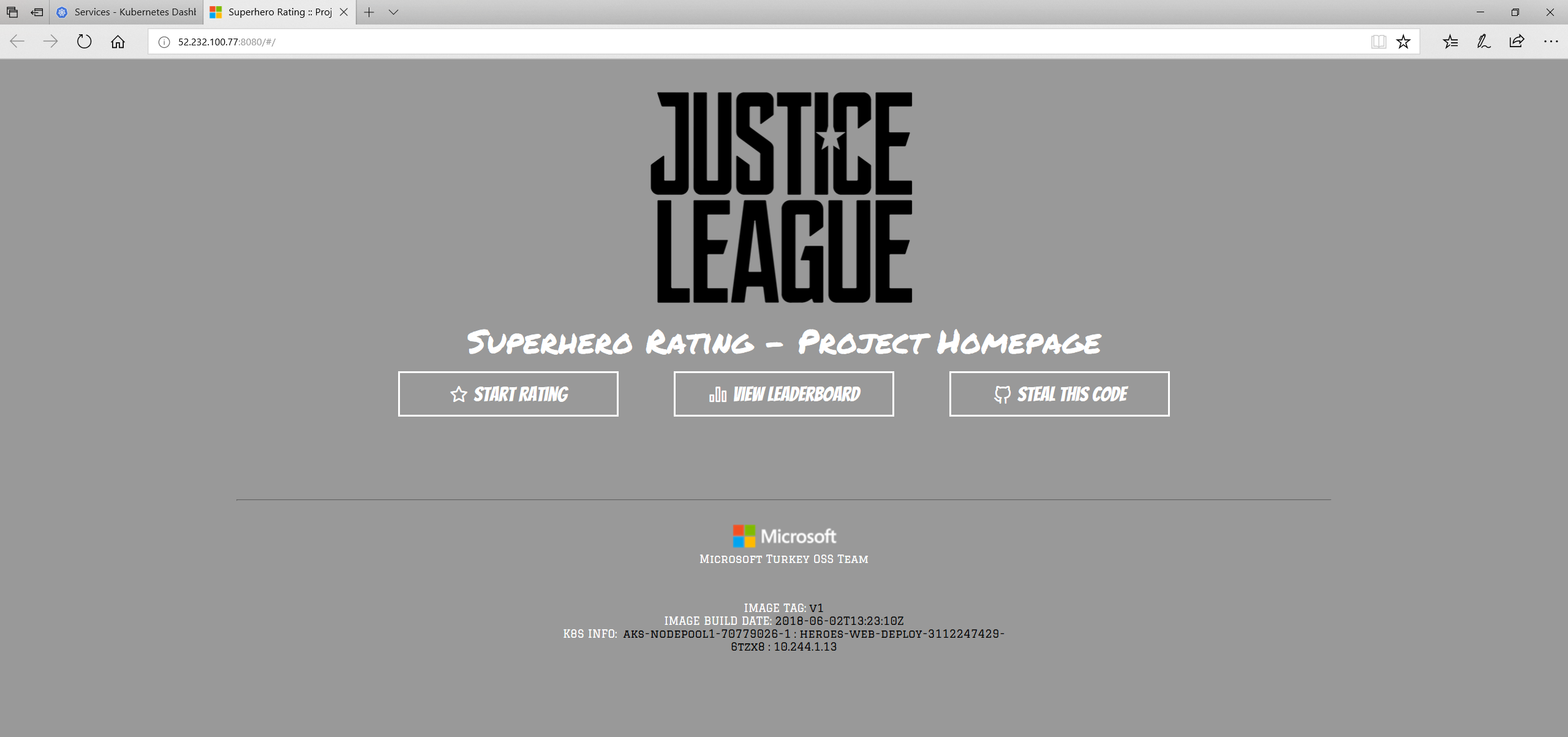


* Check to see if services are deployed.

kubectl get service -w



* Browse to the External IP for your web application (on port 8080) and try the app



The public IP can take a few minutes to create with a new cluster. Sit back and relax. Maybe check twitter.

## CI/CD Automation using Jenkins Pipeline

**Install Helm and Tiller**

Install Helm

cd /

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get | bash

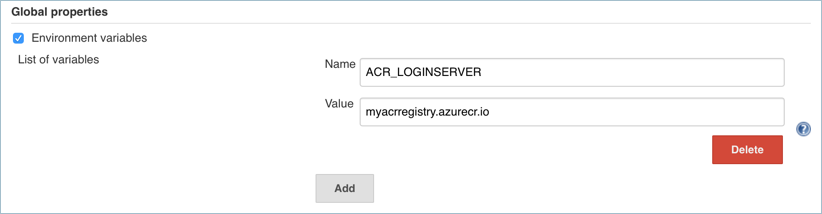
Run helm init to configure helm

helm init

**Create a Jenkins Environment Variable**

A Jenkins environment variable is used to hold the ACR login server name. This variable is referenced during the Jenkins build job. To create this environment variable, complete the following steps:

* On the left-hand side of the Jenkins portal, select Manage Jenkins > Configure System
* Under Global Properties, select Environment variables. Add a variable with the name **ACR\_LOGINSERVER** and the value of **your ACR login server** such as mycontainerregistry.azurecr.io



* When complete, click Save at the bottom of the Jenkins configuration page.

**Create a Jenkins Credential for ACR**

To allow Jenkins to build and then push updated container images to ACR, you need to specify credentials for ACR. This authentication can use Azure Active Directory service principals. In the pre-requisites, you configured the service principal for your AKS cluster with Reader permissions to your ACR registry. These permissions allow the AKS cluster to pull images from the ACR registry. During the CI/CD process, Jenkins builds new container images based on application updates, and needs to then push those images to the ACR registry. For separation of roles and permissions, now configure a service principal for Jenkins with Contributor permissions to your ACR registry.

**Create a Service Principal for Jenkins to use ACR**

* First, create SP:

az ad sp create-for-rbac --skip-assignment

* Make a note of the appId and password shown in your output. These values are used in following steps to configure the credential resource in Jenkins. (Replace 626dd8ea-042d-4043-a8df-4ef56273670f with your own appId)

APP\_ID=626dd8ea-042d-4043-a8df-4ef56273670f

* Get the resource ID of your ACR registry using the **az acr show** command, and store it as a variable. Do not forget to change ACR name.

ACR\_ID=$(az acr show --resource-group mstrdevops-rg --name mstrdevopsacr --query "id" --output tsv)

* Now create a role assignment to assign the service principal Contributor rights to the ACR registry. In the following example, provide your own appId noted before.

az role assignment create --assignee $APP\_ID --role Contributor --scope $ACR\_ID

**Create a Credential Resource in Jenkins for the ACR service principal**

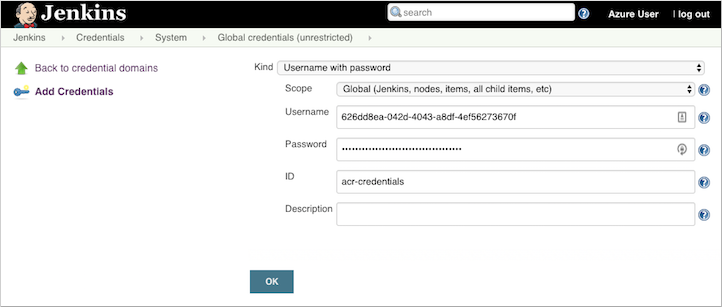
With the role assignment created in Azure, now store your ACR credentials in a Jenkins credential object. These credentials are referenced during the Jenkins build job.

Back on the left-hand side of the Jenkins portal, click Credentials > Jenkins > Global credentials (unrestricted) > Add Credentials

Ensure that the credential kind is Username with password and enter the following items:

* **Username:** The *appId* of the service principal created for authentication with your ACR registry.
* **Password:** The *password* of the service principal created for authentication with your ACR registry.
* **ID:** Credential identifier such as acr-credentials.

When complete, the credentials form looks like the following example:

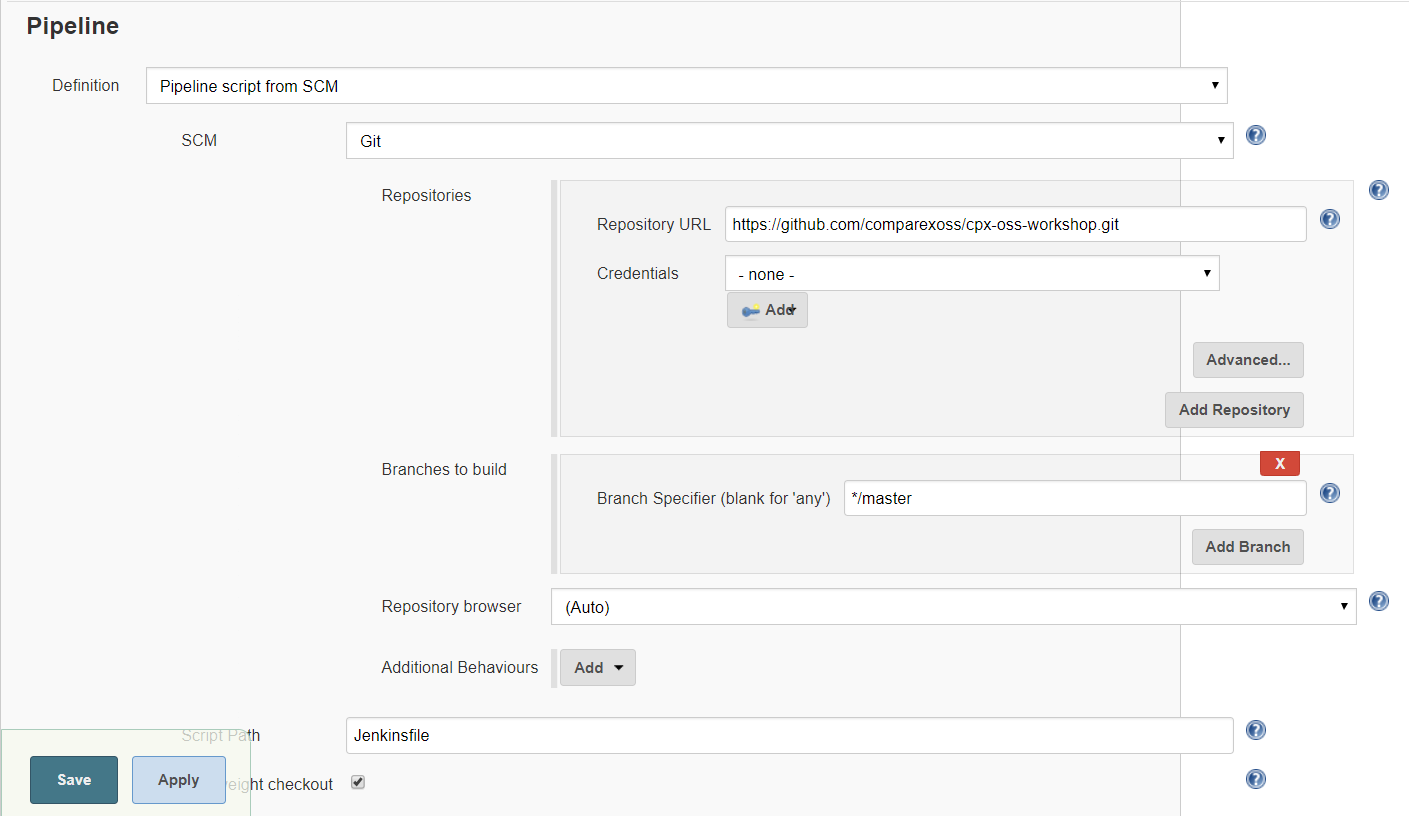


Click OK and return to the Jenkins portal.

**Create a Jenkins Project**

From the home page of your Jenkins portal, select **New item** on the left-hand side:

* Enter *superhero-ratings-pipe* as job name. Choose Pipeline project, then select OK.
* Under the **General** section, select **GitHub** project and enter your forked repo URL, such as <https://github.com/comparexoss/cpx-oss-workshop>
* Check “Do not allow concurrent builds”
* Under the **Build Triggers** section, select **GitHub hook trigger for GITscm polling**.
* Under the **Pipeline** section, select **Pipeline script from SCM** for the defitinion.

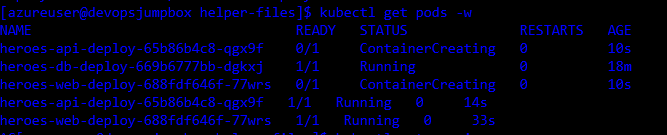


* Enter your GitHub project repository url. Do not forget to change github address with your own address.
* Select **Jenkinsfile** under the **Script Path** section.
* Once completed, click Save and build the project that you have created.

**Validate**

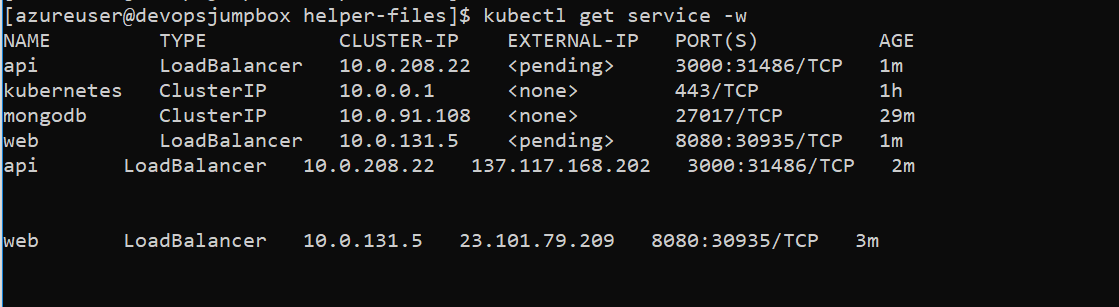
Check to see if pods are running in your cluster

kubectl get pods

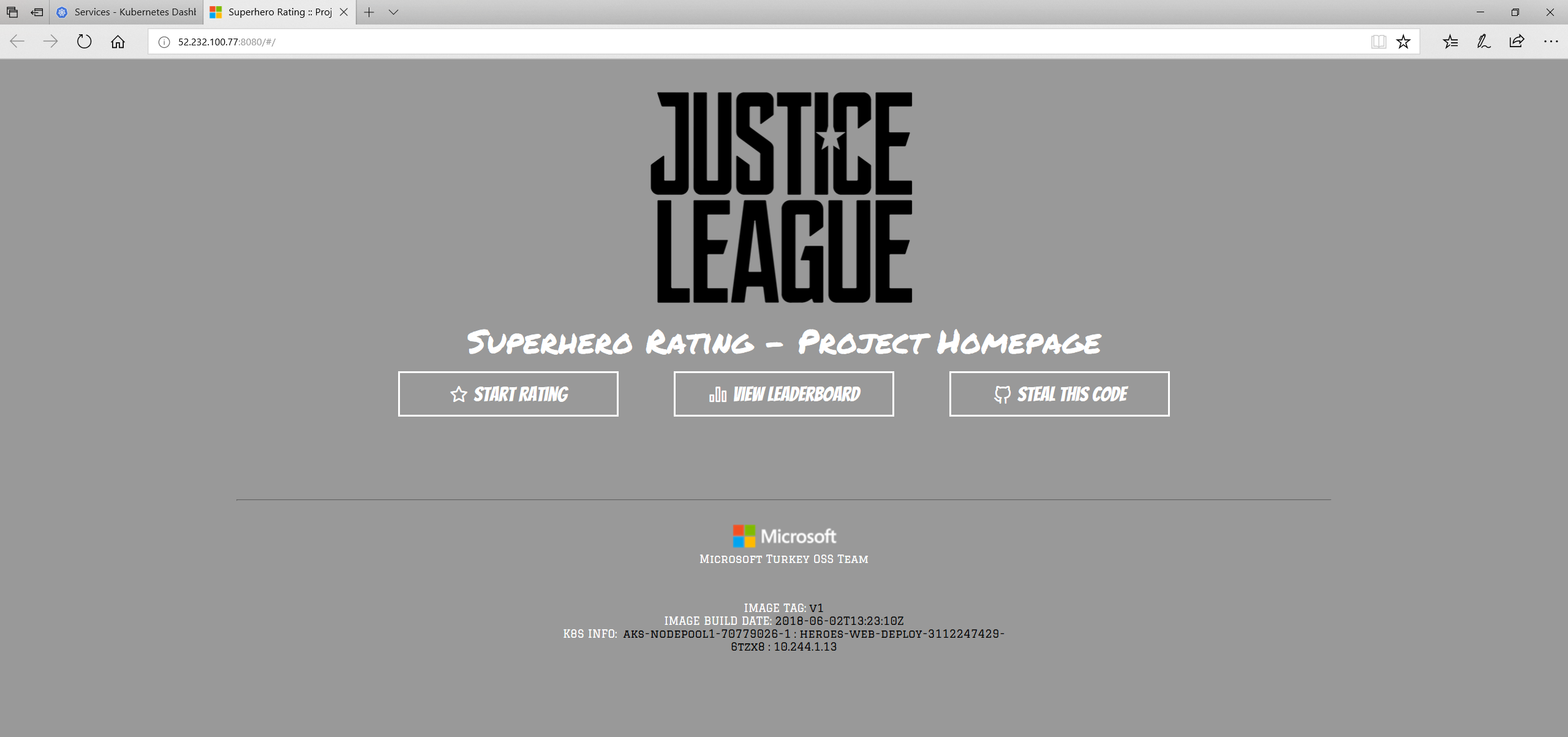


* Check to see if services are deployed.

kubectl get service -w



* Browse to the External IP for your web application (on port 8080) and try the app



The public IP can take a few minutes to create with a new cluster. Sit back and relax. Maybe check twitter.

1. Kubernetes UI Overview

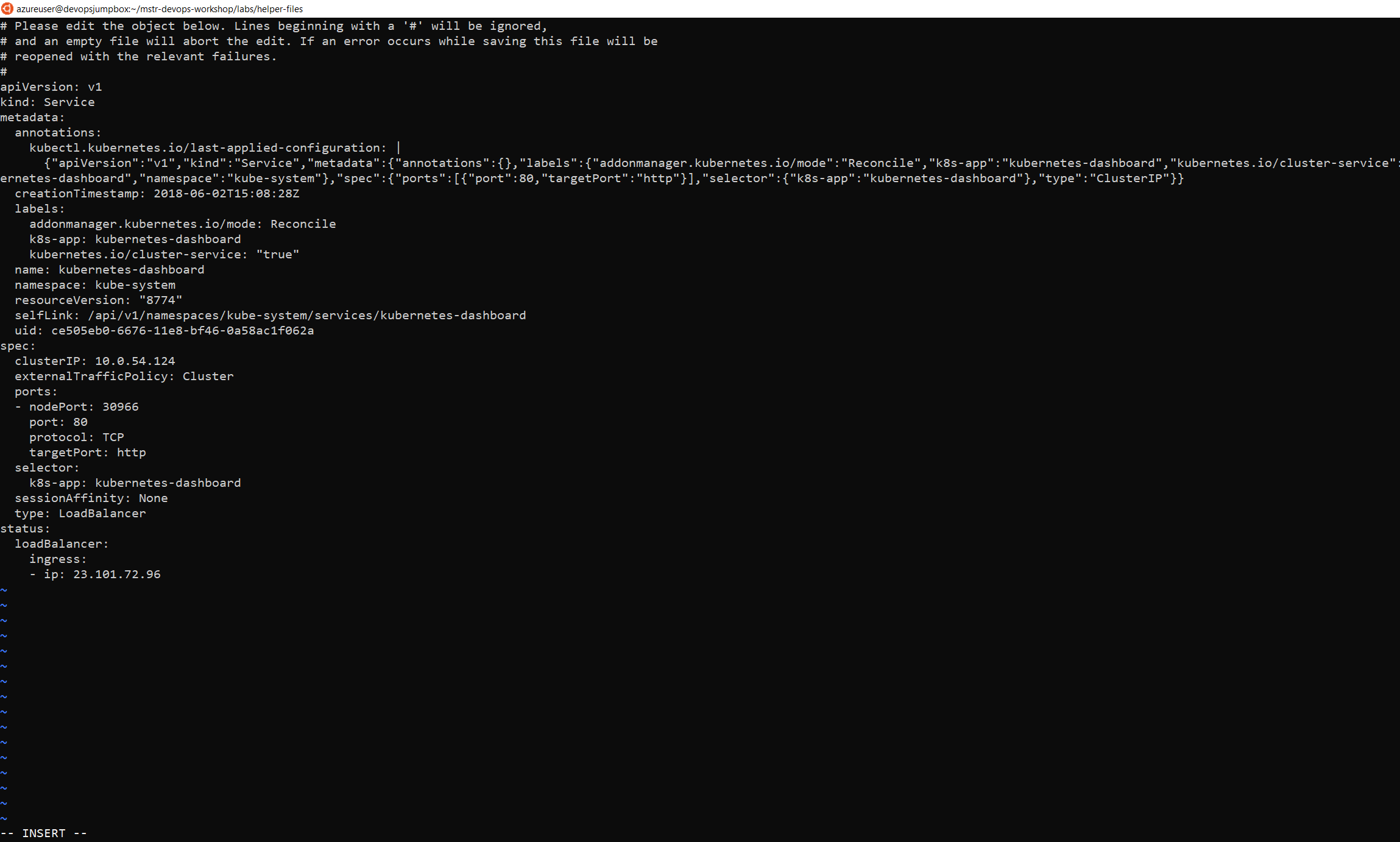
The Kubernetes dashboard is a web ui that lets you view, monitor, and troubleshoot Kubernetes resources.

**Accessing The Dashboard UI**

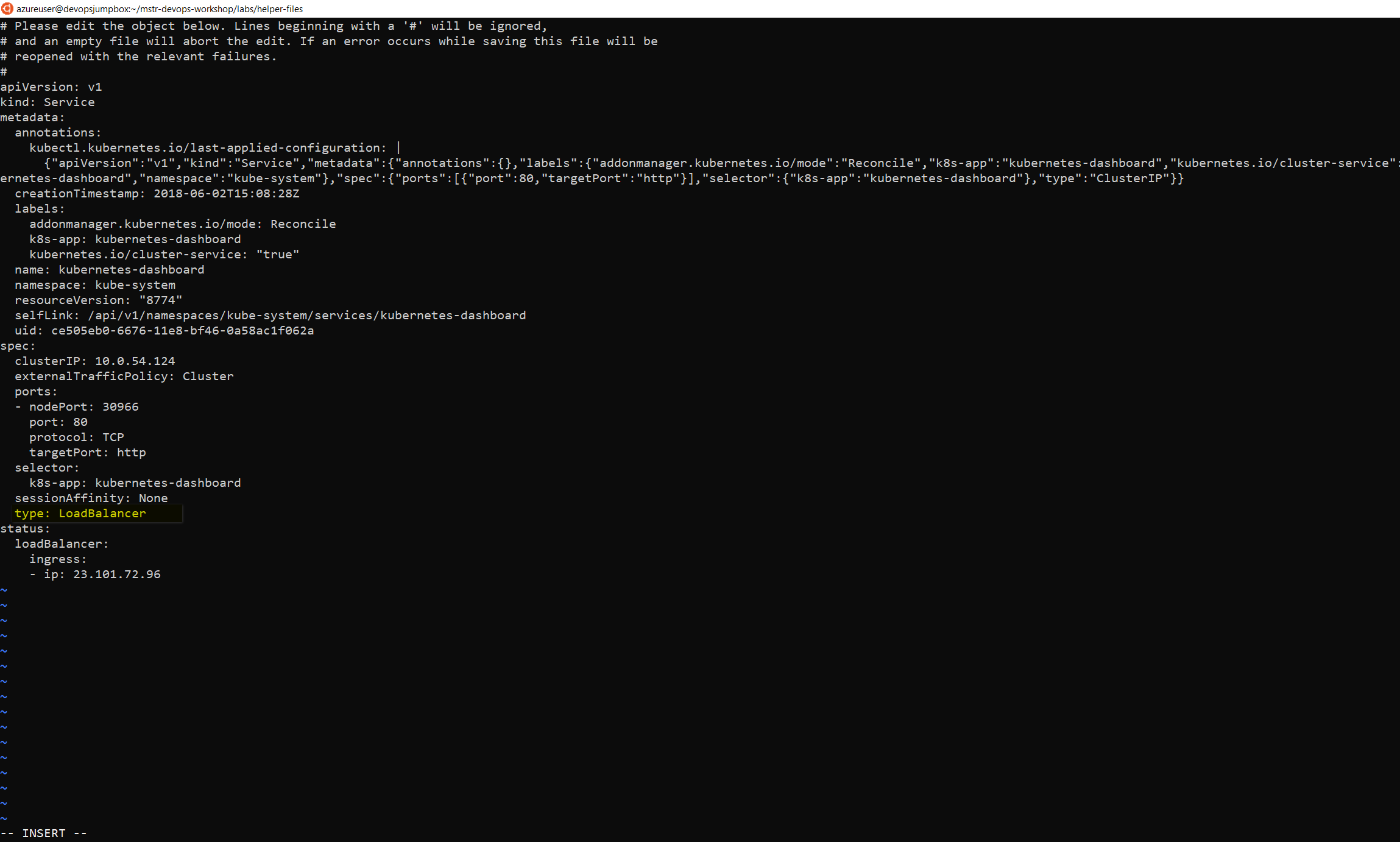
There are multiple ways of accessing Kubernetes dashboard. You can access through kubectl command-line interface or through the master server API. We'll be using kubectl, as it provides a secure connection, that doesn't expose the UI to the internet.

1. Command-Line Proxy
   * Open a Bash Shell

kubectl -n kube-system edit service kubernetes-dashboard

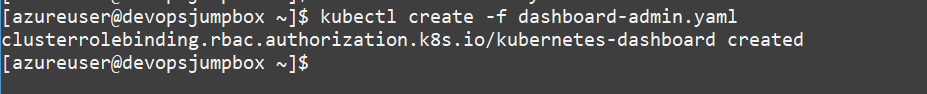


Change Type: ClusterIP to Type:LoadBalancer

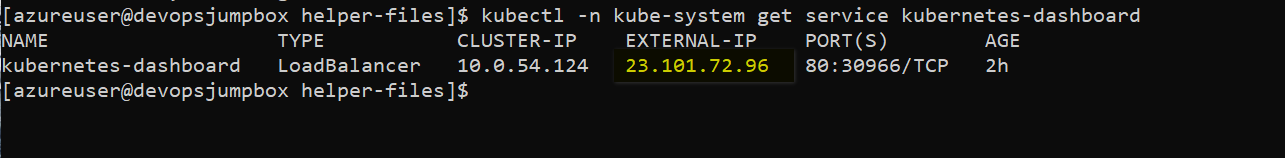


cd ~/cpx-oss-workshop/labs/helper-files/

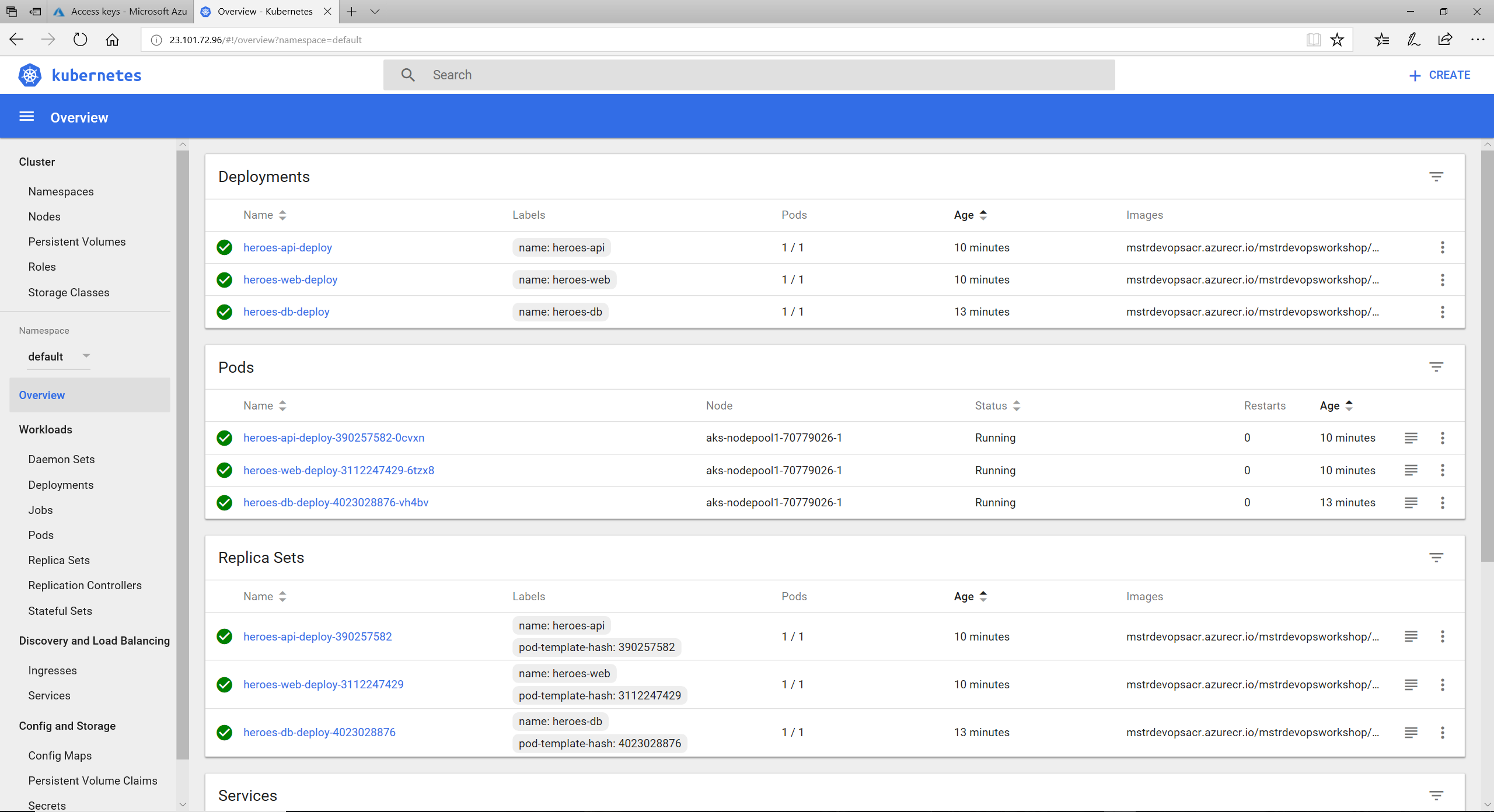
kubectl create -f dashboard-admin.yaml



kubectl -n kube-system get service kubernetes-dashboard

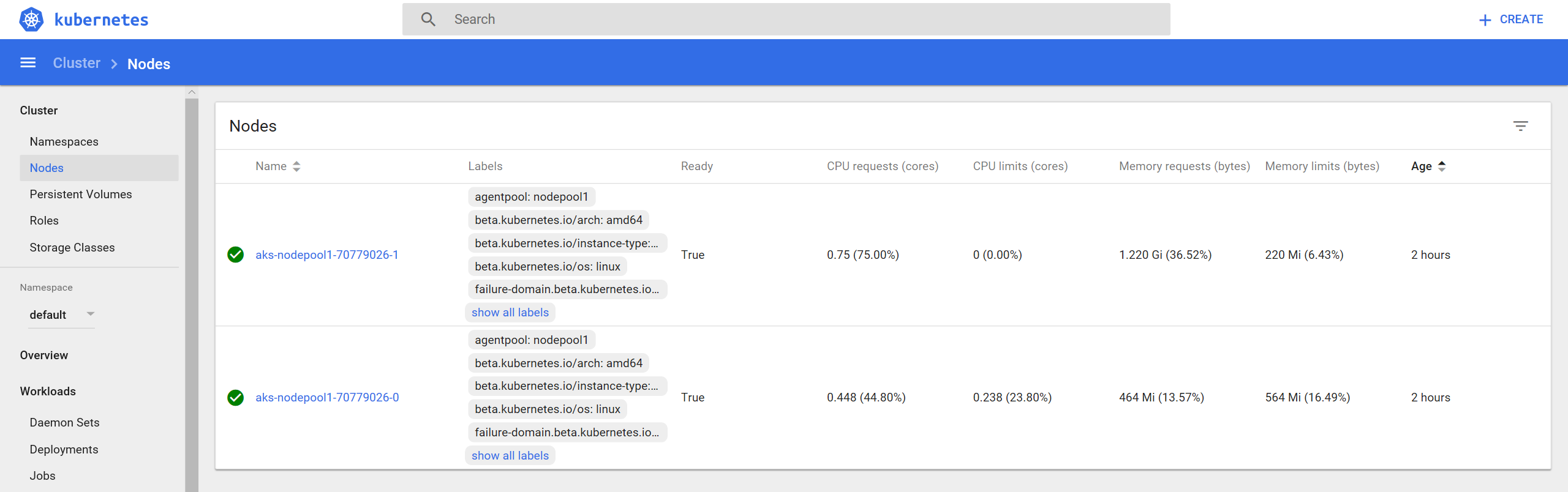


http://23.101.72.96

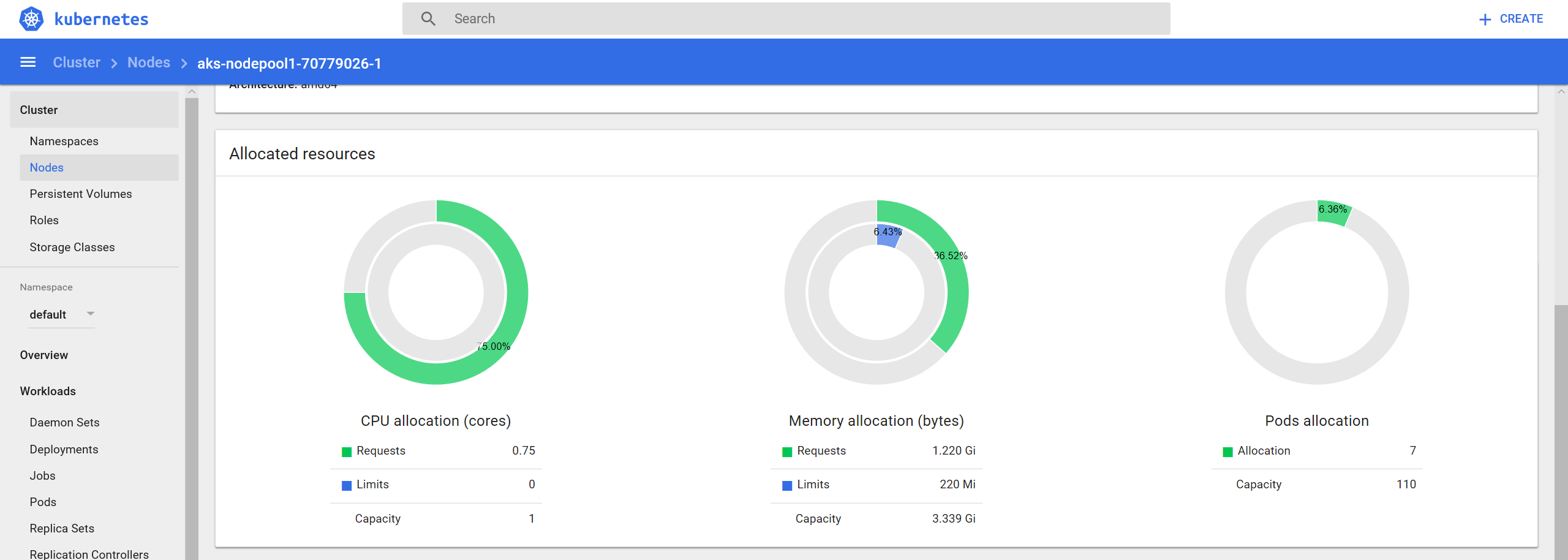


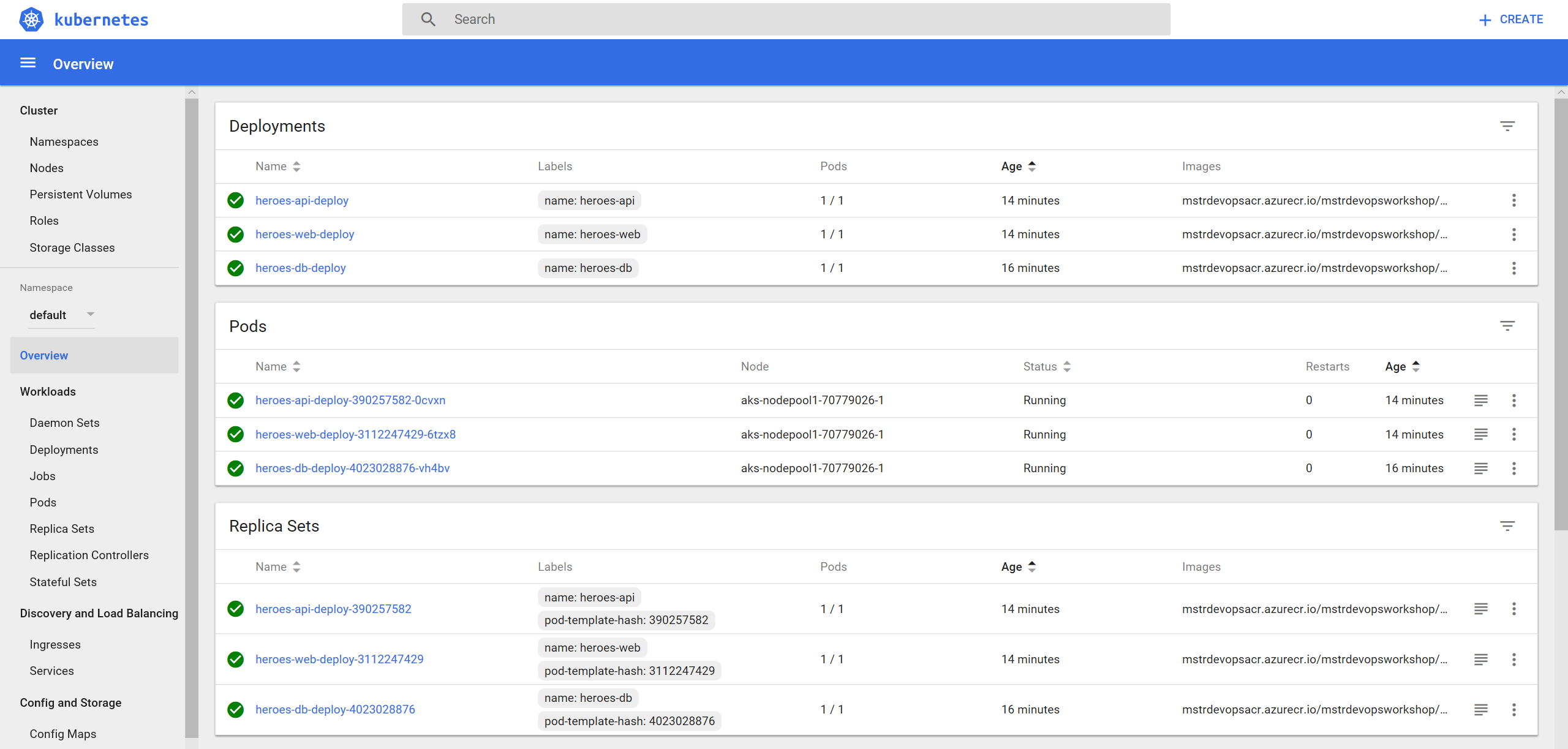
**Explore Kubernetes Dashboard**

1. In the Kubernetes Dashboard select nodes to view



1. Explore the different node properties avalabile through the dashboard



1. Explore the different pod properties avaliable through the dashboard 
2. In this lab feel free to take a look around other at other resources Kubernetes provides through the dashboard
3. Operational Monitoring and Log Management

There are a number of monitoring solutions available today. Here is a quick, but not exhaustive list for reference purposes:

* Datadog
* Sysdig
* Elastic Stack
* Splunk
* Operations Management Suite
* Prometheus

For the purposes of this lab we will be focusing in on Prometheus and using Grafana to provide a visual Dashboard of our Azure Kubernetes Service Cluster.

We are going to be installing Prometheus and Grafana into our K8s cluster using Helm and Tiller. You can think of Helm as a package manager for Kubernetes with Tiller being the server-side component.

**Install Prometheus using Helm**

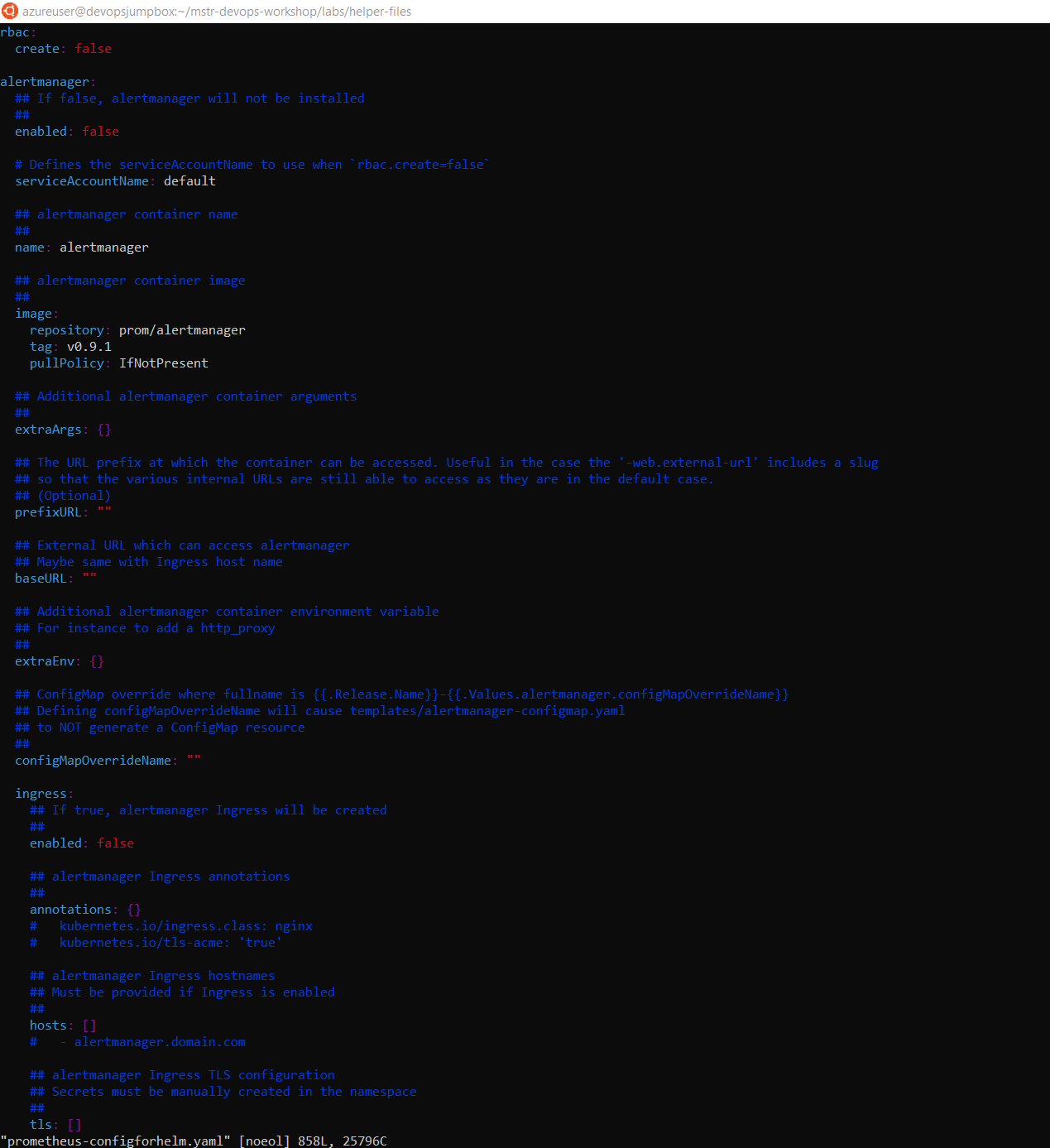
Prometheus is a Cloud Native Computing Foundation (CNCF) project used to collect and process metrics. It collects metrics from configured targets, in our case it is a Kubernetes Cluster.

1. Install Prometheus using Helm CLI

* Switch to the helper-files directory and view the prometheus-configforhelm.yaml file. This configures Helm to install Prometheus with our desired settings.

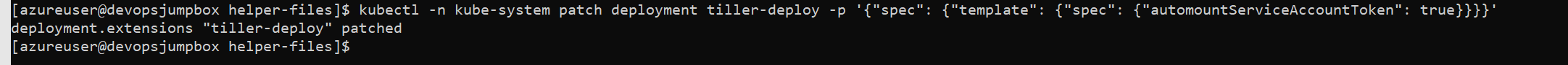
cd ~/cpx-oss-workshop/labs/helper-files/

cat prometheus-configforhelm.yaml



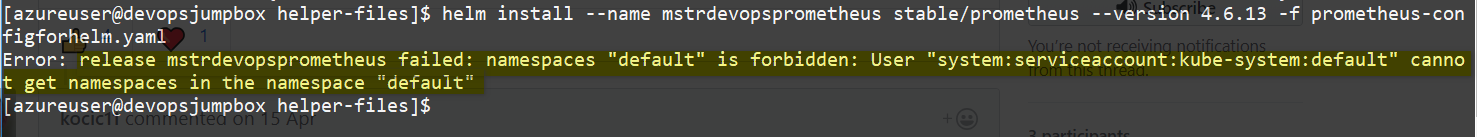
# The following command will install Prometheus into the K8s cluster using custom settings.

kubectl -n kube-system patch deployment tiller-deploy -p '{"spec": {"template": {"spec": {"automountServiceAccountToken": true}}}}'

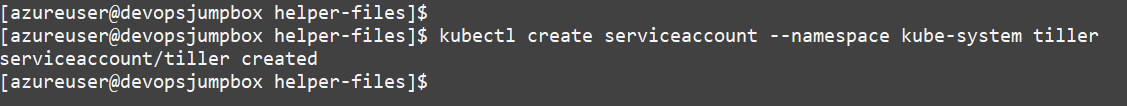


helm install --name mstrdevopsprometheus stable/prometheus --version 4.6.13 -f prometheus-configforhelm.yaml

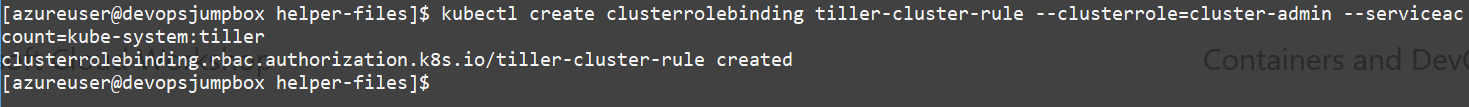
If you get an error, try the steps below.



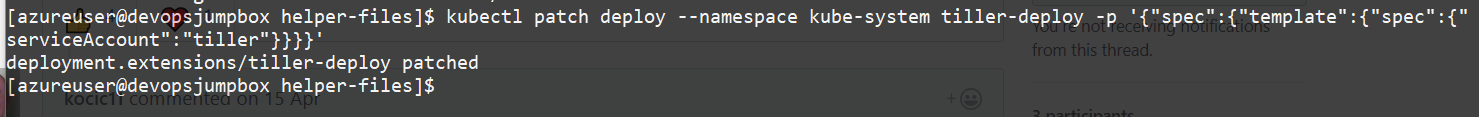
kubectl create serviceaccount --namespace kube-system tiller



kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --serviceaccount=kube-system:tiller

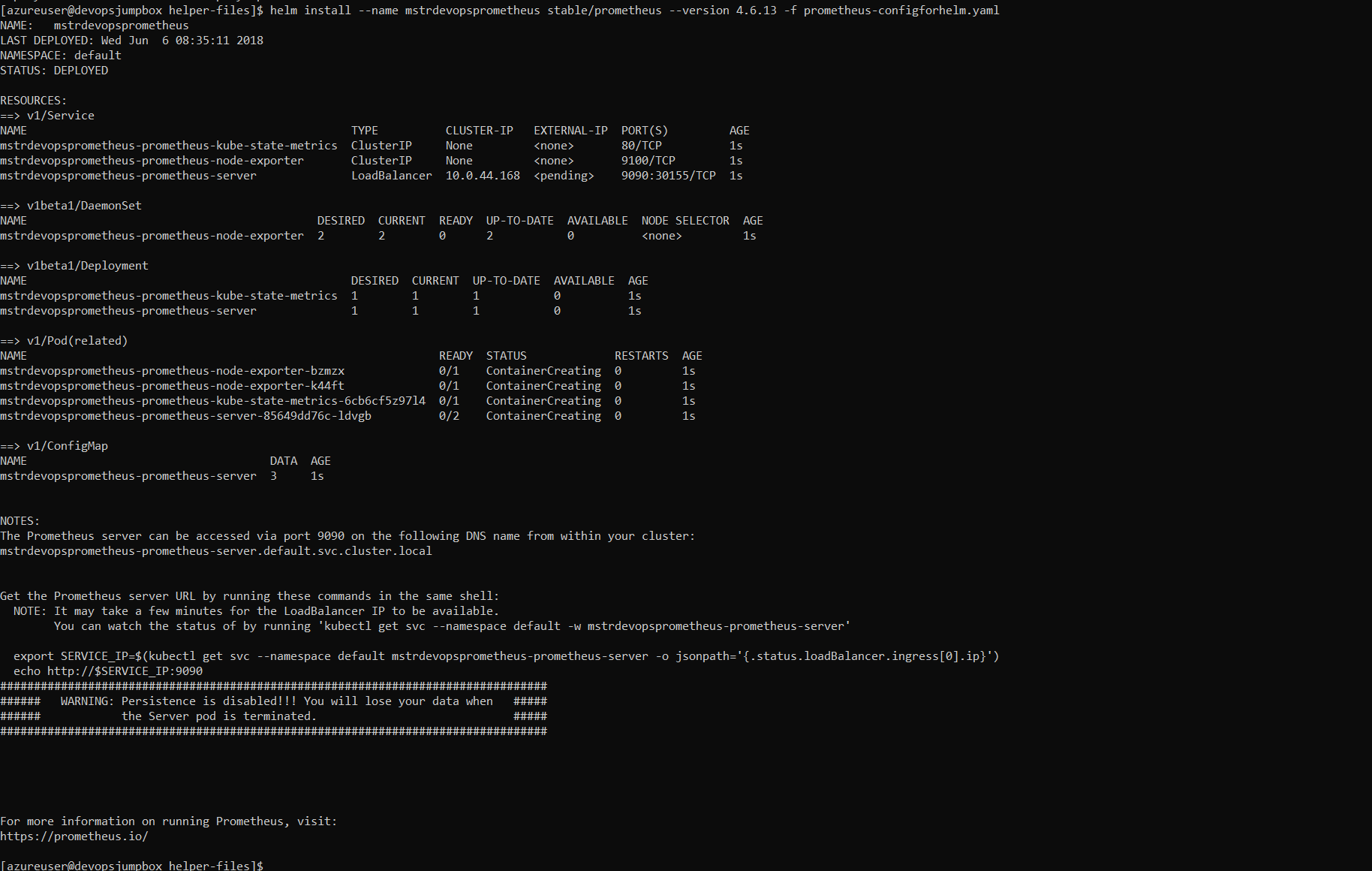


kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":{"template":{"spec":{"serviceAccount":"tiller"}}}}'



try it now again!

helm install --name mstrdevopsprometheus stable/prometheus --version 4.6.13 -f prometheus-configforhelm.yaml

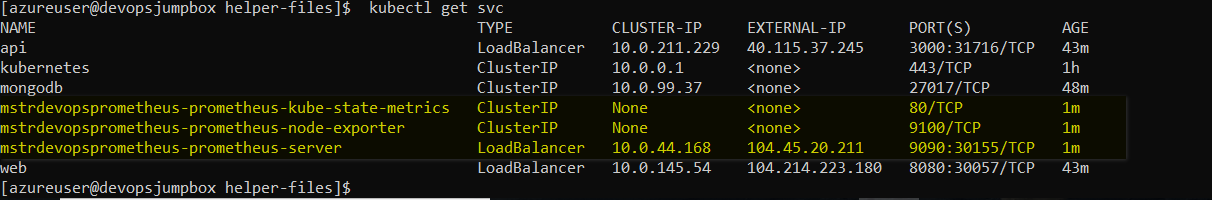


1. Validate that Prometheus was Installed

kubectl get pods



kubectl get svc



**Install Grafana**

Grafana is a dashboard visualization tool that can use all kinds of data sources. In our case, Prometheus will be used as the data source.

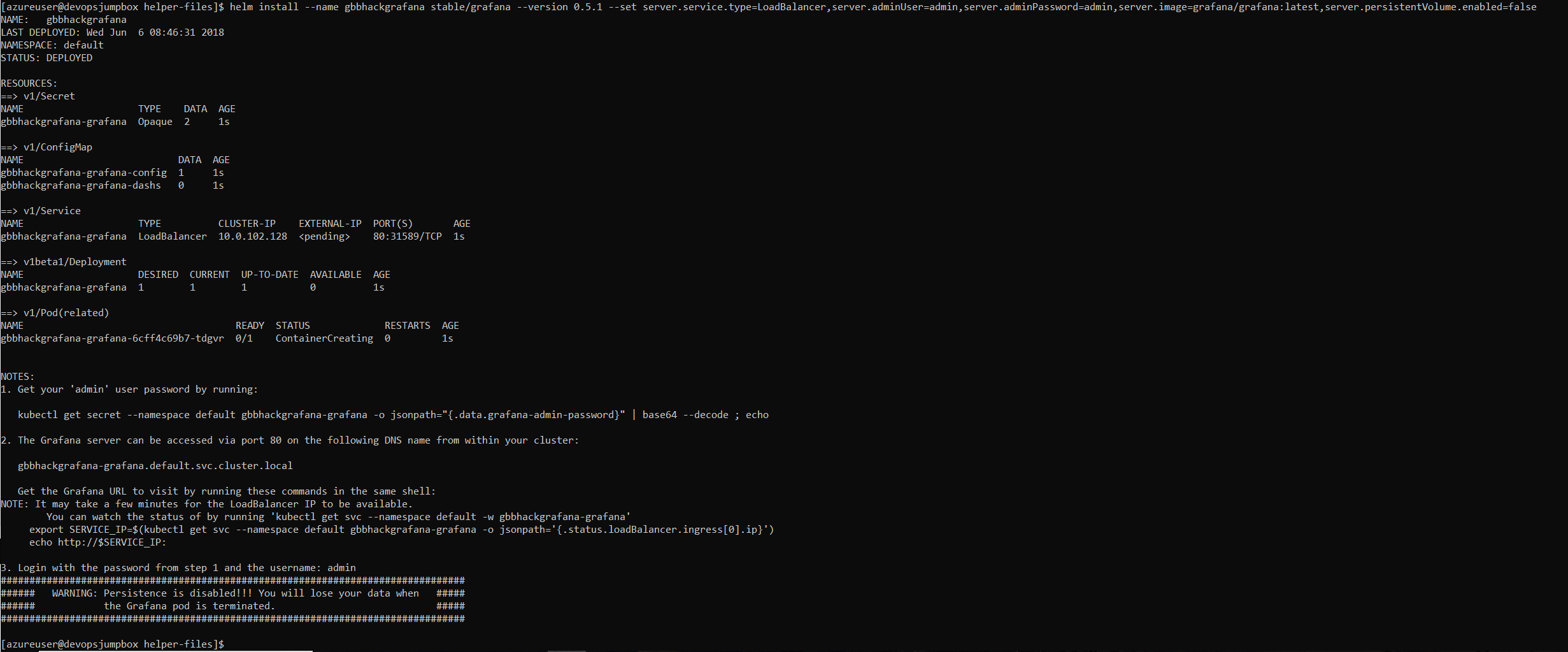
1. Install Grafana using Helm CLI

* The following command will install Grafana into the K8s cluster with a few custom settings to make it easier to access.

# We are setting the default username and password to \*\*admin\*\* to make it easier to remember.

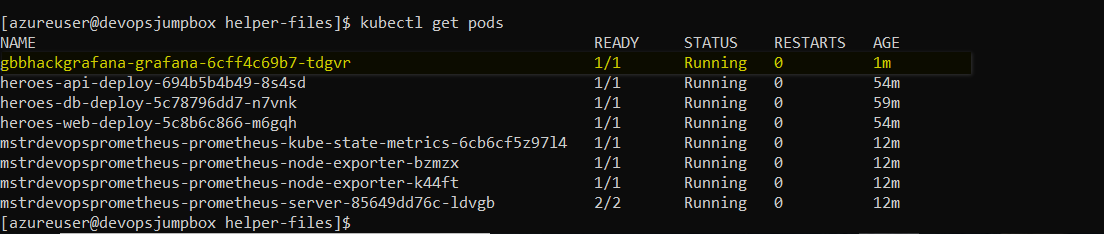
# We are also setting the service type to \*\*LoadBalancer\*\* to expose the service outside of the cluster and make it accessible via the Internet.

helm install --name mstrdevopsgrafana stable/grafana --version 0.5.1 --set server.service.type=LoadBalancer,server.adminUser=admin,server.adminPassword=admin,server.image=grafana/grafana:latest,server.persistentVolume.enabled=false

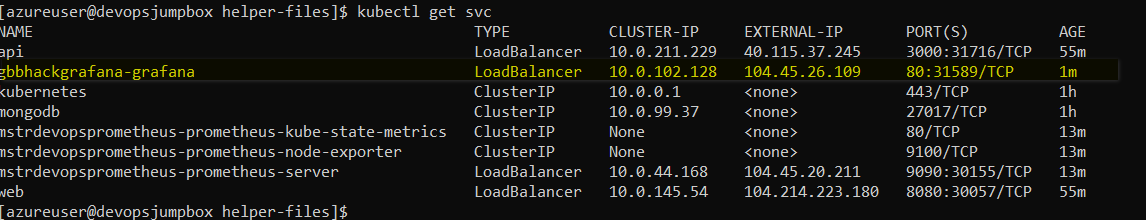


1. Validate that Grafana was Installed

kubectl get pods

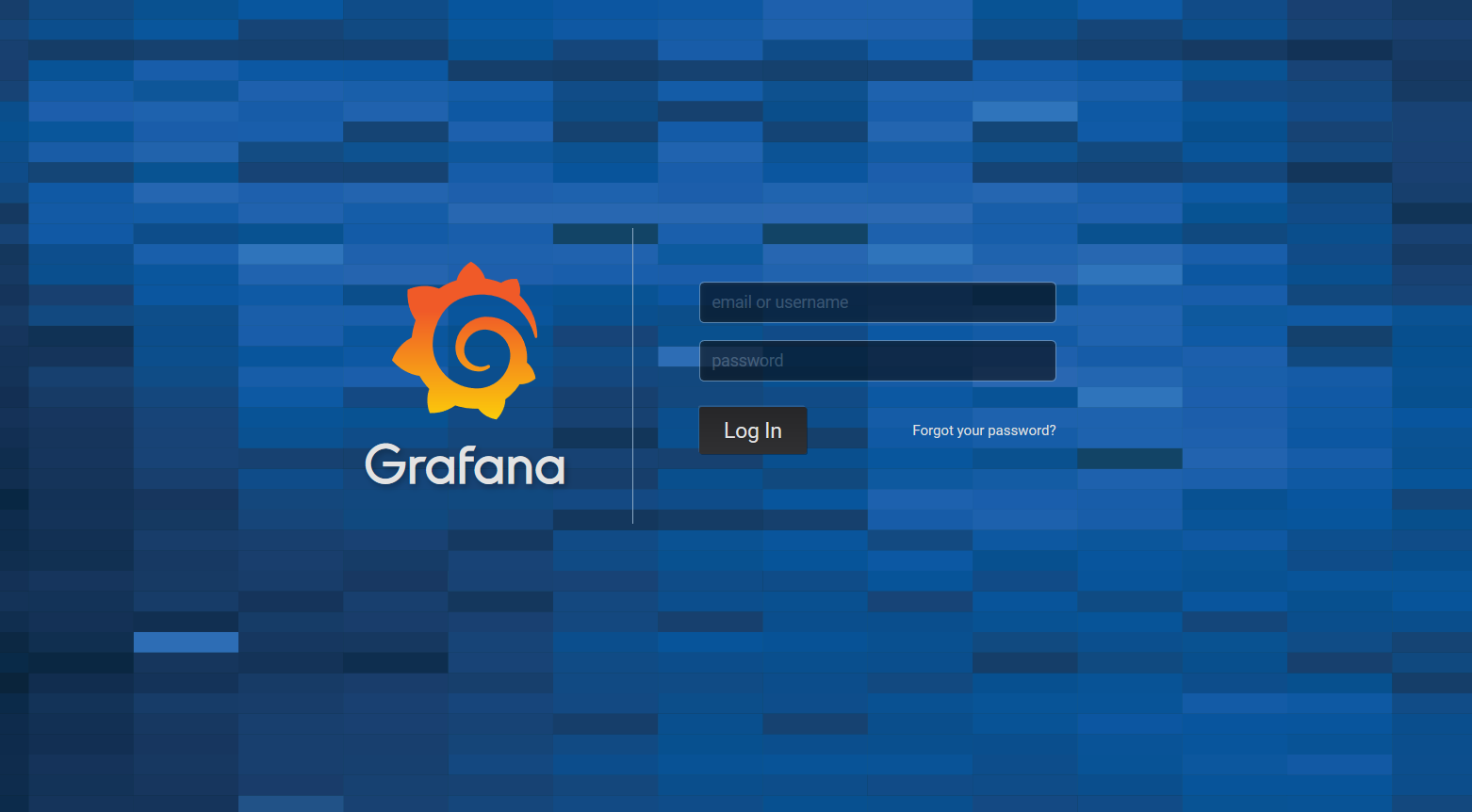


kubectl get svc



Test Grafana UI Comes Up Use the EXTERNAL-IP value from the previous step and put that into your browser:

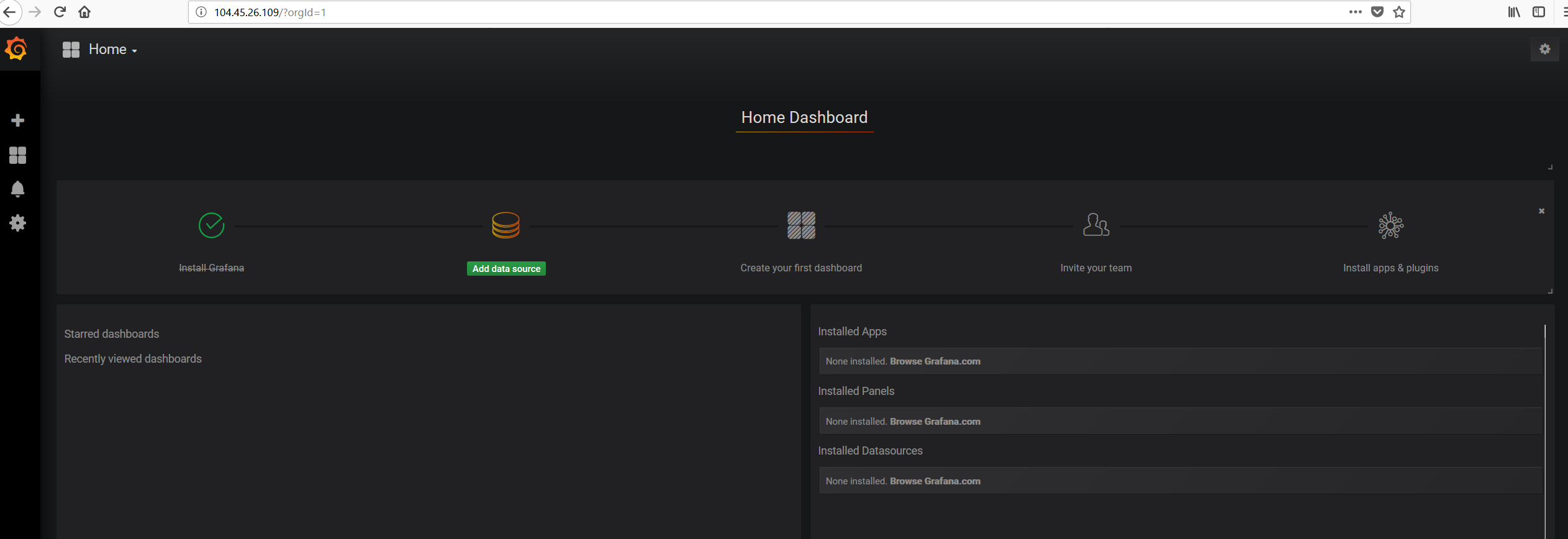
* eg. [http://52.226.75.38](http://52.226.75.38/), EXTERNAL-IP column from above. You should see something like the following come up, be patient it will take a moment or two:



**Setting up Grafana**

1. Log into Grafana Dashboard using **admin** for the username and password

* You should see something like the following:



1. Add Prometheus as a Data Source

* If you recall from above, we exposed a number of K8s services, one of those services was the Prometheus Server. We are going to use that Service endpoint in our Data Service configuration. The Add Data Source screen should look something like the below screen shot.

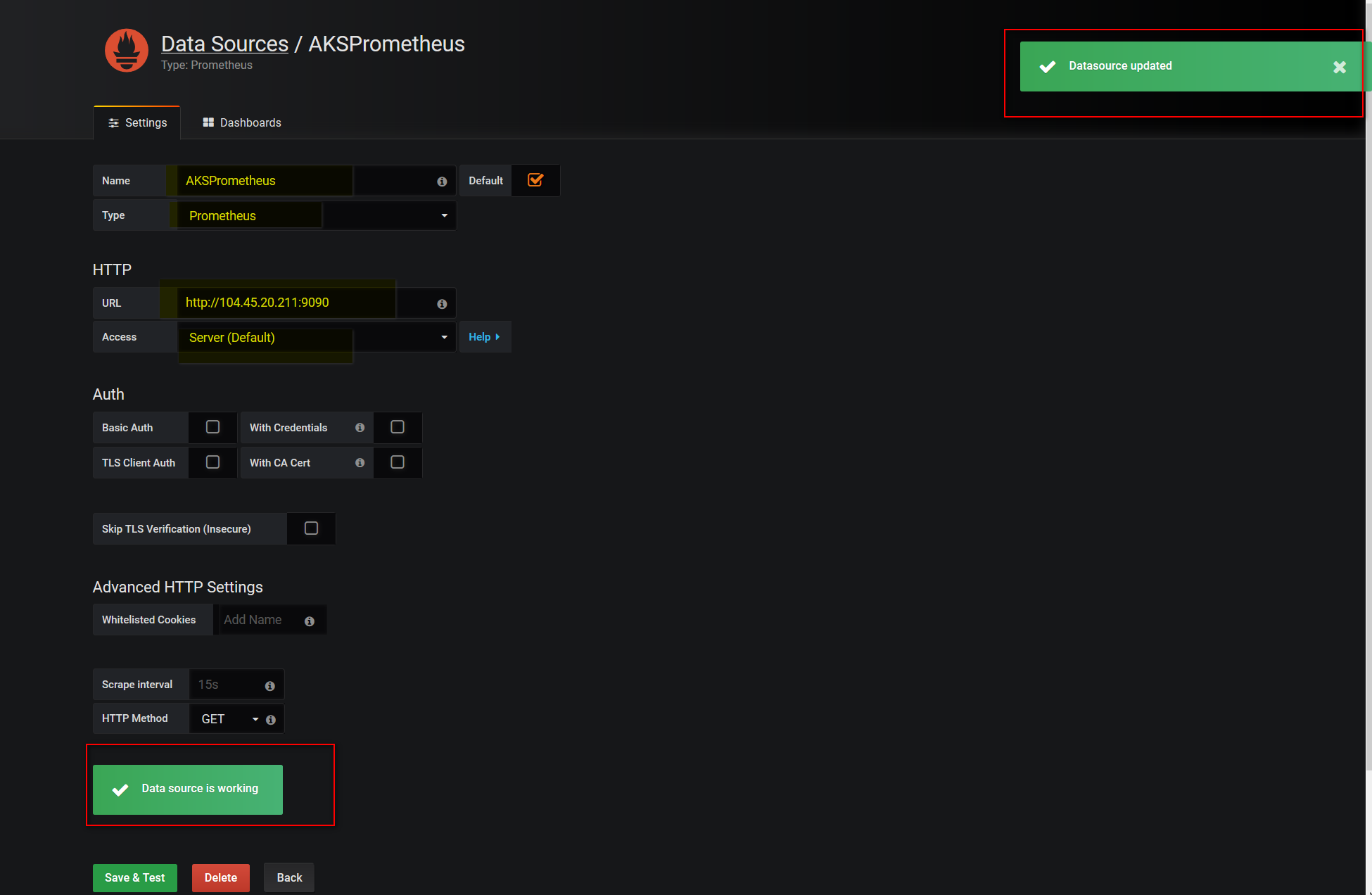
Use http://{PROMETHEUS\_PUBLIC\_IPADDRESS}:9090 for the URL in the HTTP settings.

1. Validate Prometheus Data Source

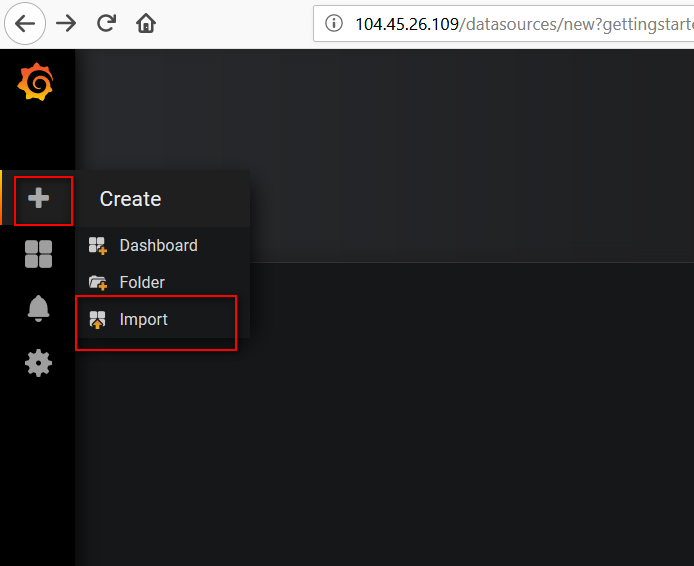
* Once you have filled in the values similar to the screenshot above, click the **Add** button and ensure no errors come back.

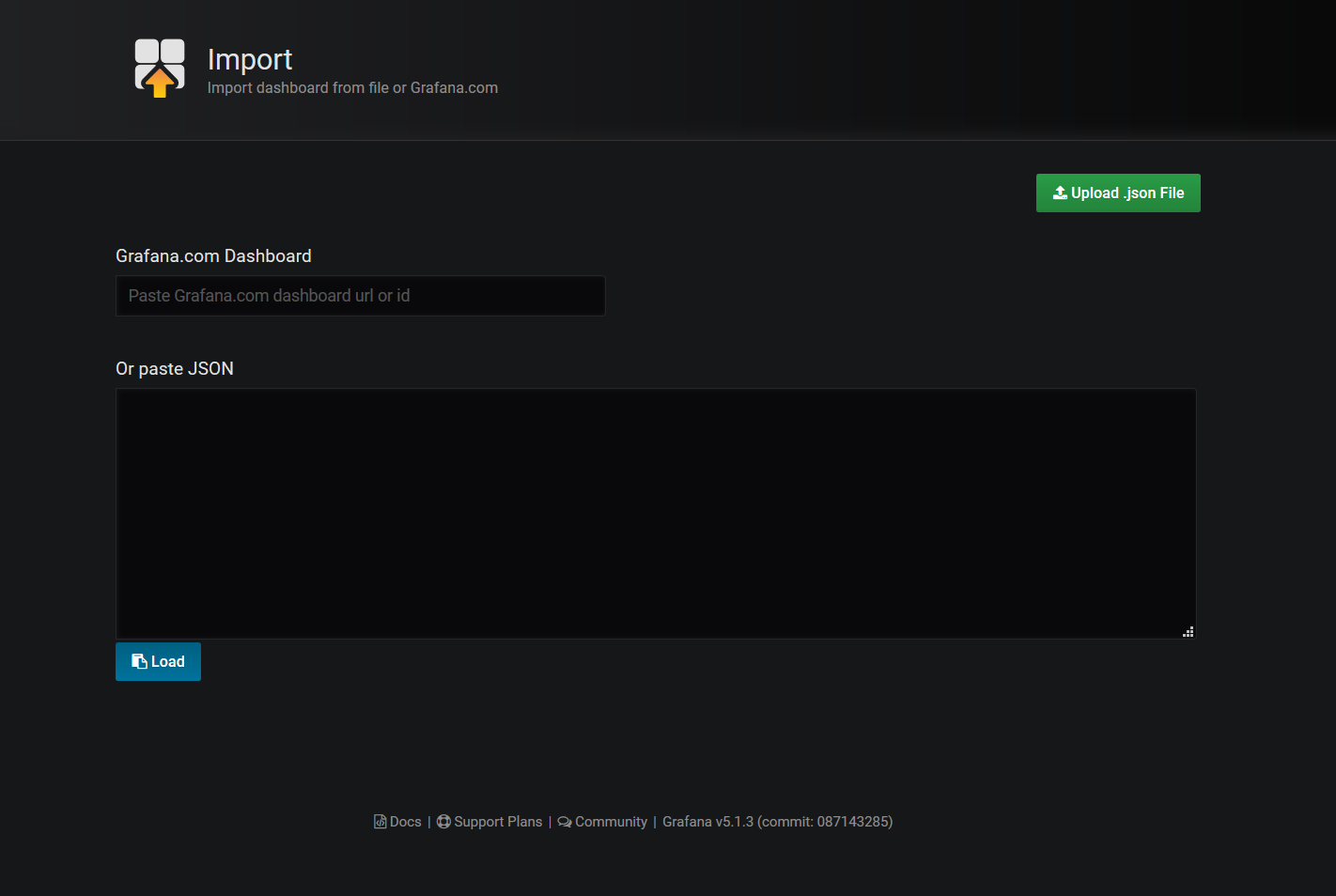
1. Add K8s Monitoring Dashboard to Grafana

* After the datasource has been added, it is now time to add a dashboard. Grafana dashboards can be shared on Grafana.com. Go to import dashboards viam the menu in the top left.

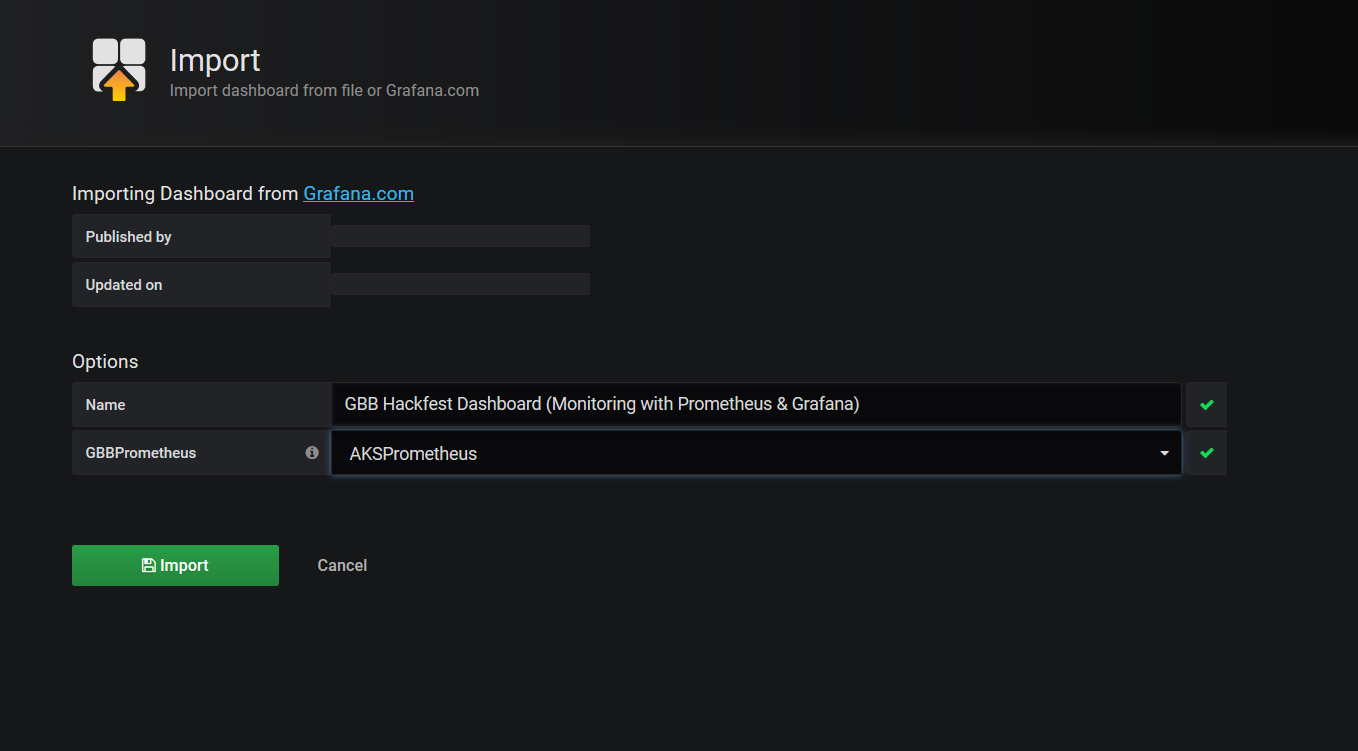


* Click on the **Upload File** button and browse to the **grafana-dashboard.json** in the **cpx-oss-workshop/labs/helper-files/** directory. You can also paste the contents of the json into the text box.

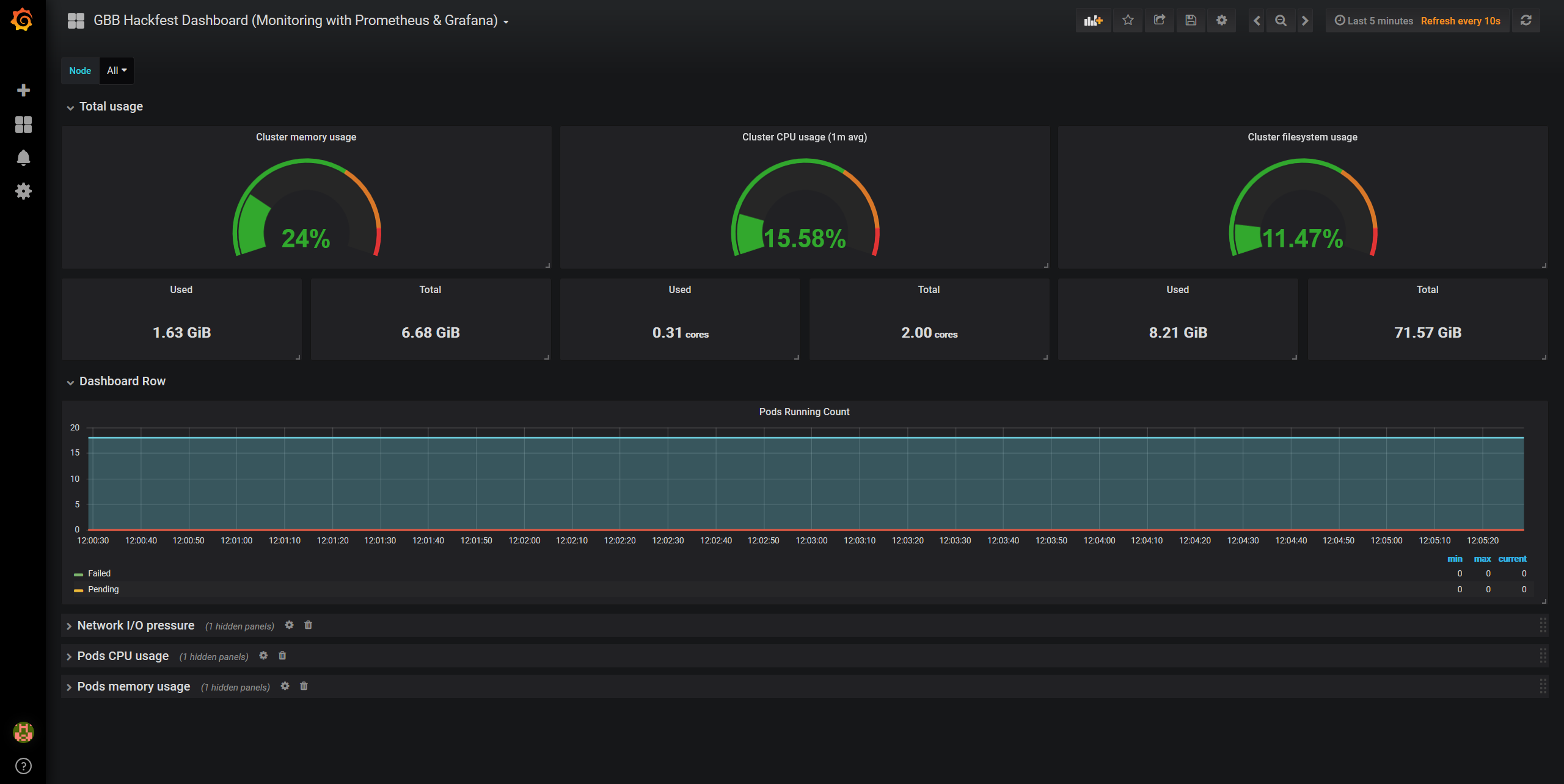




* Set the datasource dropdown to the "AKSPrometheus" that was created in the previous step.



* Click the **Import** button.



You should now have Prometheus and Grafana running in your Azure Kubernetes Service cluster and be able to see the Grafana Dashboard.

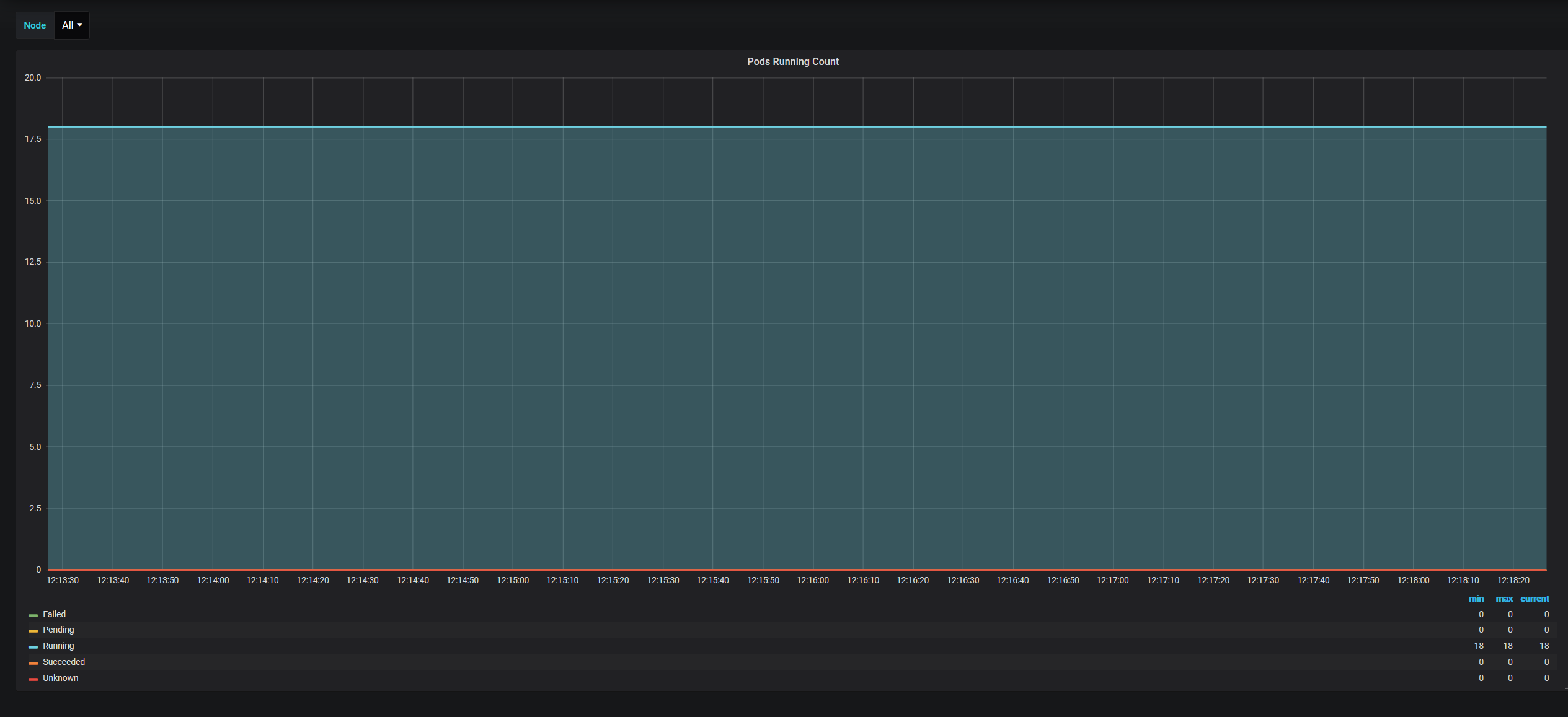
1. Application and Infrastructure Scaling

Imagine a scenario where your realize that your existing cluster is at capacity and you need to scale it out to add more nodes in order to increase capacity and be able to deploy more PODS.

**Exercise 1 - Scale Application**

1. Check to see current number of pods running via Grafana Dashboard.

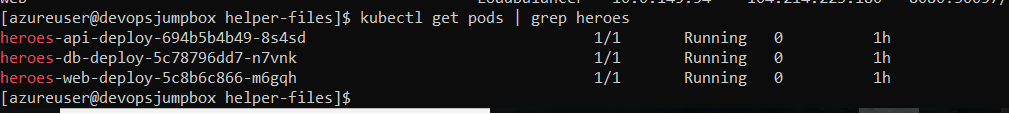
* Go to the same Grafana Dashboard from exercise 8 and look at the **Pods Running Count** section. You will see the total count of Pods and the various phases they are in.



1. Check to see current number of heroes pods running via K8s CLI.

kubectl get pods | grep heroes

# You should see something like the following as output (one replica of each pod):

3. Scale out the Web application

* To simulate a real-world scenario we are going to scale the web app to handle increased load.

# This command will create multiple replicas of the heroes-web pod to simulate additional load on the cluster.

kubectl scale deploy/heroes-web-deploy --replicas=4

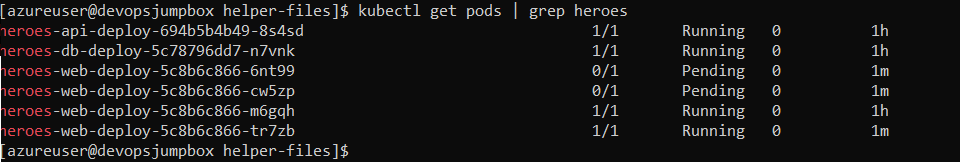
1. Check to see number of pods now running via Grafana Dashboard



1. Check to see number of heroes pods running via kubectl

kubectl get pod | grep heroes

# You should see something like the following as output (more than one heroes-web pod and some of them in different states):



1. Check up on Pods Running in Grafana dashboard

* As you can see we have a number of pods that are in the pending state which means they are trying to be scheduled to run. In this scenario the cluster is out of capacity so they are not able to be scheduled.

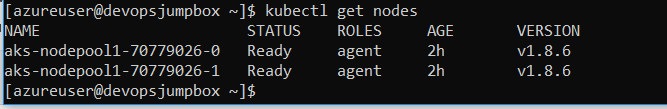


**Exercise 2 - Scale K8s Cluster**

1. Check to see number of current nodes running.

kubectl get nodes

# You should see something like the following as output (there is one node in the cluster):



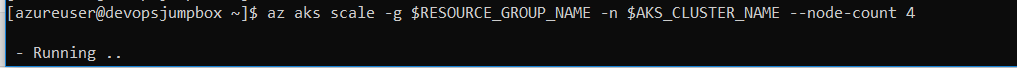
2 Scale out AKS cluster to accomodate the demand

# set these values to match yours (the cluster and the RG are the same name)

RESOURCE\_GROUP\_NAME=mstrdevopsaks-rg

AKS\_CLUSTER\_NAME=devopsaks

az aks scale -g $RESOURCE\_GROUP\_NAME -n $AKS\_CLUSTER\_NAME --node-count 4

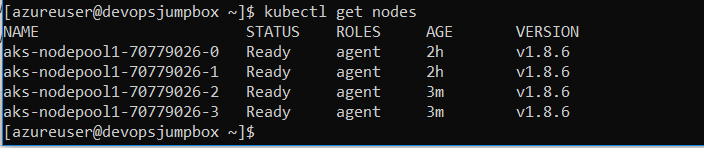


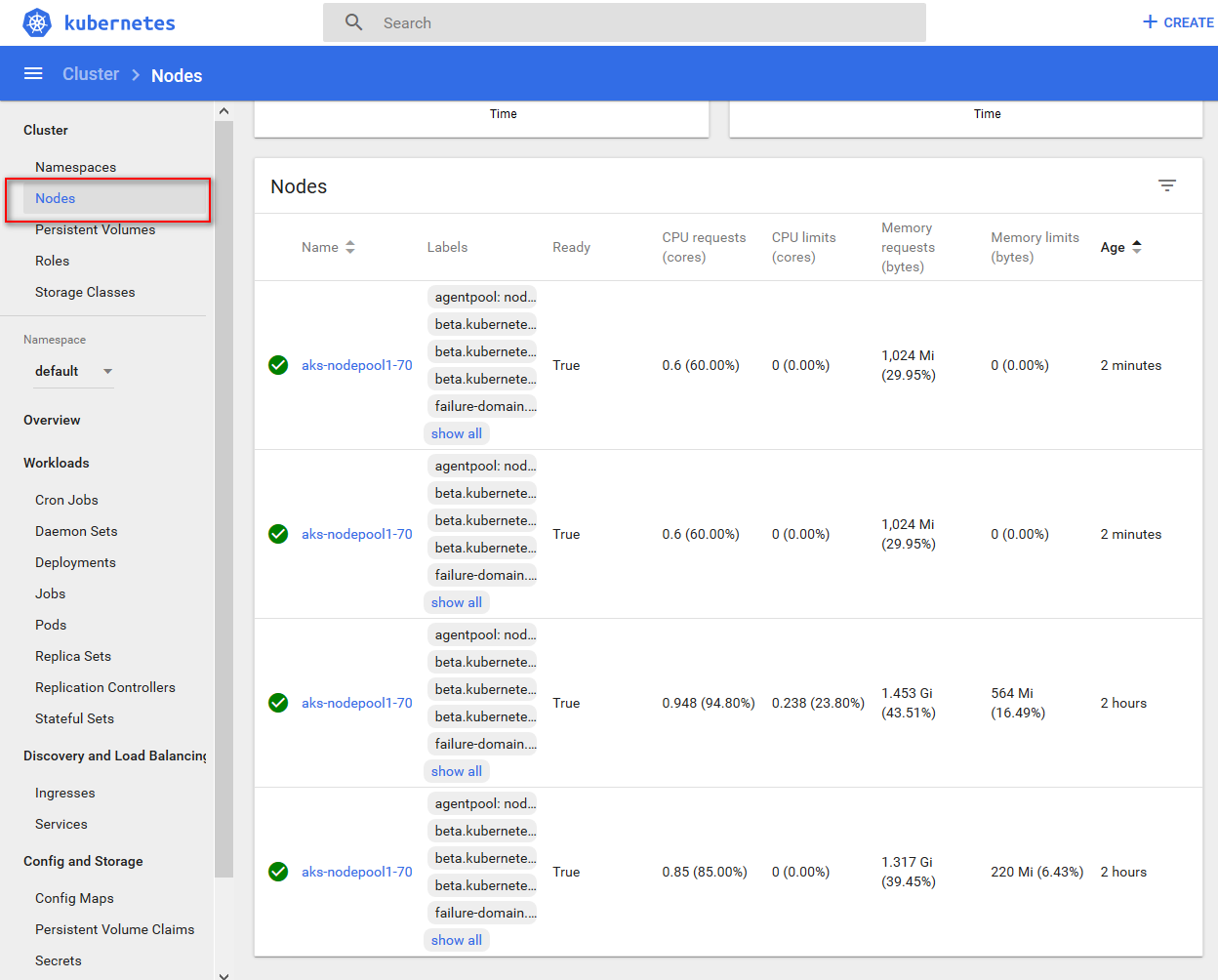
Note this may take some time. Good time to get some coffee.

1. Check to see if the new nodes are deployed and "Ready"

kubectl get nodes

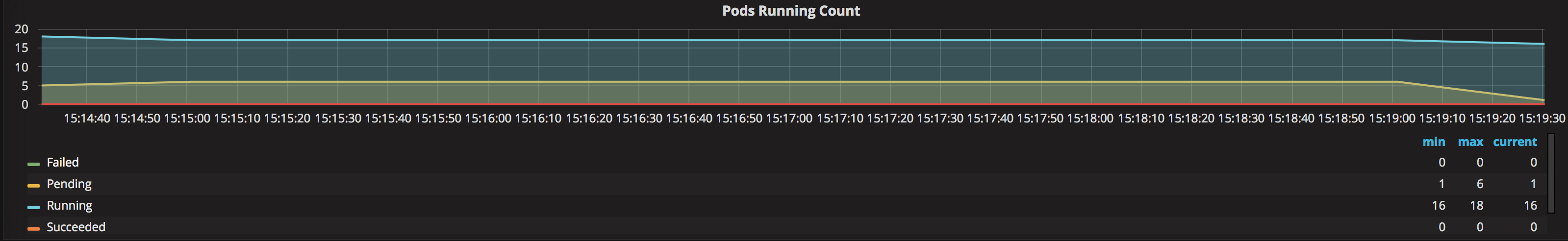
# You should see something like the following as output (there are now 4 nodes in the cluster):





4.Re-visit Grafana Dasboard to validate cluster scale is working.

* Take a look at the **Pods Pending Count** again and you should see that after a few minutes the number of pending pods is going down.

[](https://github.com/eozkurt/blackbelt-aks-hackfest/blob/master/linux-container-workshop/hol-content/img/9-grafana_podsscaling.png)

You now have additional node capacity in your Azure Kubernetes Service cluster to be able to provision more pods.

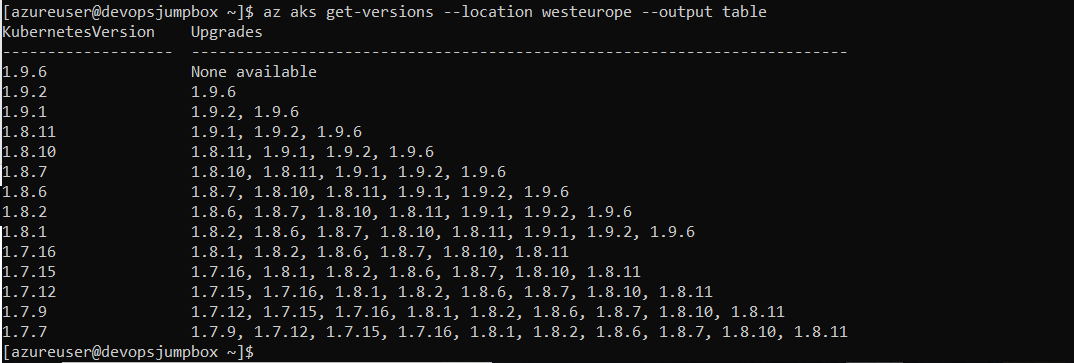
1. Upgrade an Azure Kubernetes Service (AKS) cluster

Azure Container Service (AKS) makes it easy to perform common management tasks including upgrading Kubernetes clusters.

**Upgrade an AKS cluster**

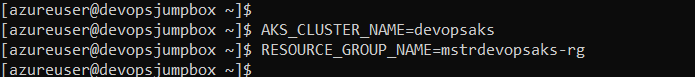
Before upgrading a cluster, use the az aks get-versions command to check which Kubernetes releases are available for upgrade.

az aks get-versions --location westeurope --output table



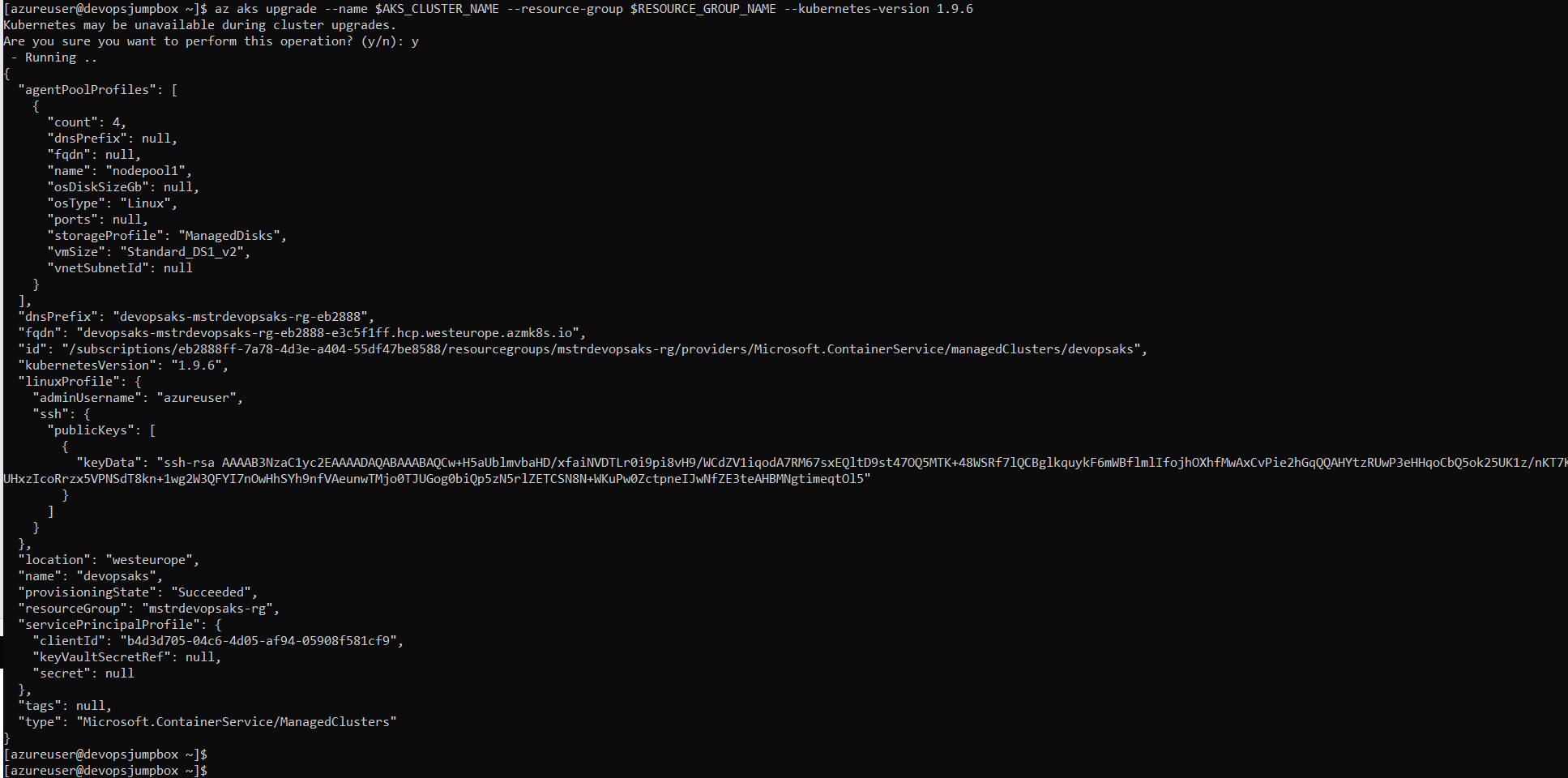
AKS\_CLUSTER\_NAME=devopsaks

RESOURCE\_GROUP\_NAME=mstrdevopsaks-rg



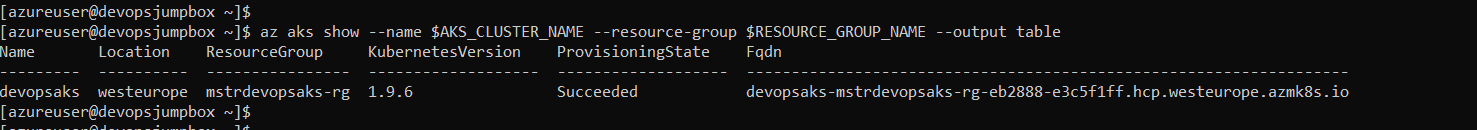
az aks scale -g $RESOURCE\_GROUP\_NAME -n $AKS\_CLUSTER\_NAME --node-count 2

az aks upgrade --name $AKS\_CLUSTER\_NAME --resource-group $RESOURCE\_GROUP\_NAME --kubernetes-version 1.10.5



You can now confirm the upgrade was successful with the az aks show command.

az aks show --name $AKS\_CLUSTER\_NAME --resource-group $RESOURCE\_GROUP\_NAME --output table



1. Update and Deploy New Version of Application

In this lab, we will make a change to the web application and then re-deploy the new container image into AKS.

**Update web application code**

1. Open your GitHub account and browse to cpx-oss-workshop/app/web/src/components/Footer.vue and edit this file
2. Find the snippet below *(line 13)* and change the text *"Microsoft & Comparex Turkey OSS Teams"* to your name or whatever you would like to display.

<div class="row at-row flex-center flex-middle">

<div class="col-lg-6">

</div>

<div class="col-lg-12 credits">

Microsoft & Comparex Turkey OSS Teams

</div>

<div class="col-lg-6">

</div>

</div>

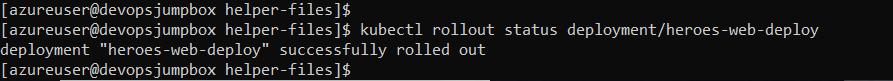
1. Commit your edits and check your Jenkins pipeline project

**Check status**

1. You can see the updates and history for the changes from above

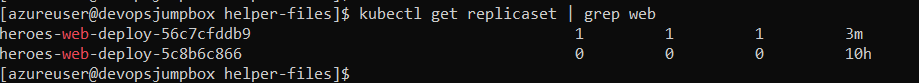
# this command will verify that latest deployment was successful

kubectl rollout status deployment/heroes-web-deploy



# each deployment creates a new replicaset

kubectl get replicaset | grep web



Browse to your newly deployed web application

