# FULL STEP BY STEP DOCUMENT

# 1: Prepare an application for Azure Kubernetes Service (AKS)

Simple docker application run

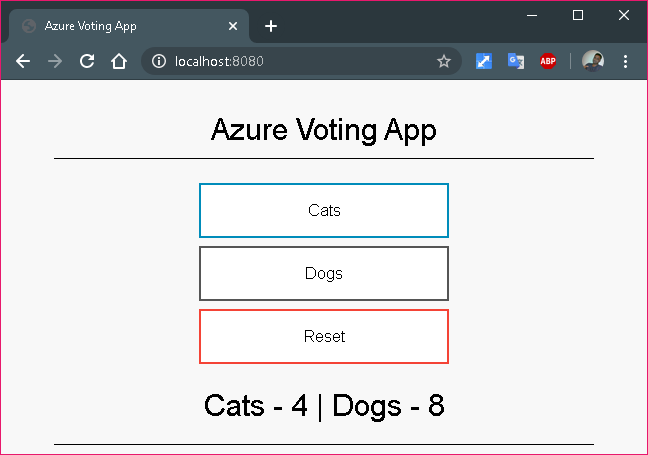
git clone https://github.com/Azure-Samples/azure-voting-app-redis.git

$ docker-compose up -d

$ docker images

$ docker ps

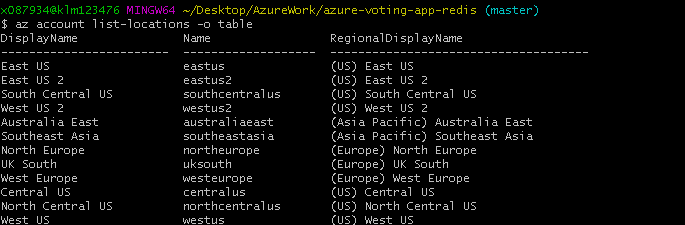
$ docker-compose down



# 2: Deploy and use Azure Container Registry

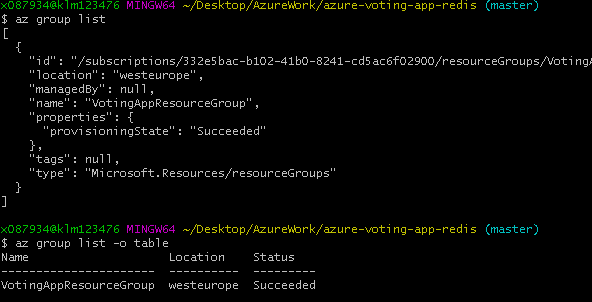
An Azure resource group is a logical group in which Azure resources are deployed and managed. When you create a resource group, you are asked to specify a location. This location is where resource group metadata is stored, it is also where your resources run in Azure if you don't specify another region during resource creation. Create a resource group using the az group create command.

* List available regions



Create resource group

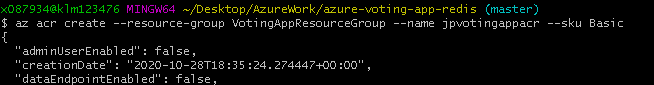
az group create --name VotingAppResourceGroup --location westeurope



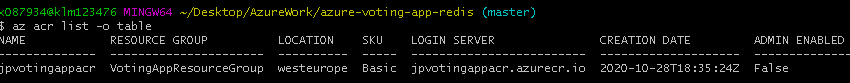
**Create container registry**

$ az acr create --resource-group myResourceGroup --name <acrName> --sku Basic

$ az acr create --resource-group VotingAppResourceGroup --name jpvotingappacr --sku Basic



$ az acr list -o table



**Log in to the container registry**

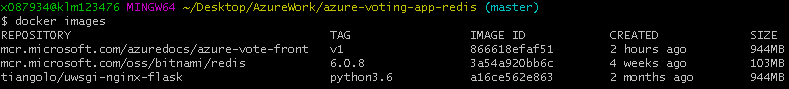
$ az acr login --name jpvotingappacr



**List local container images**

This command requires running docker dameon.

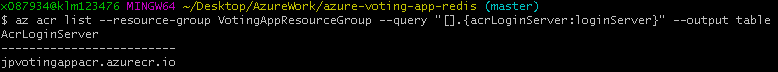
$ docker images



**Tag a container image**

To use the azure-vote-front container image with ACR, the image needs to be tagged with the login server address of your registry. This tag is used for routing when pushing container images to an image registry.

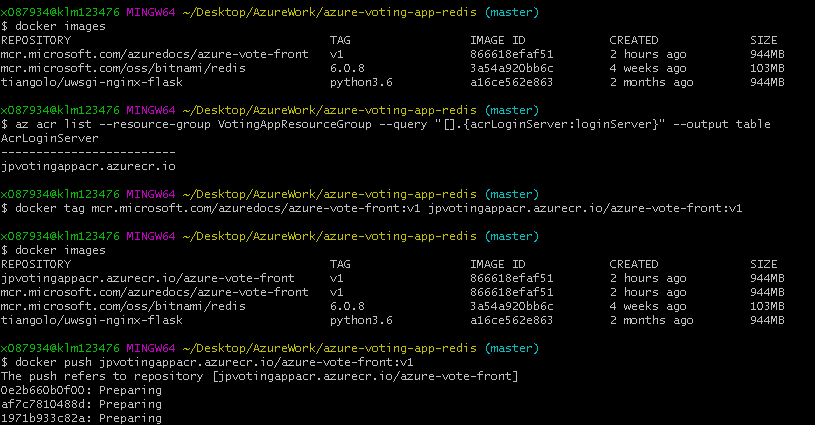
$ az acr list --resource-group VotingAppResourceGroup --query "[].{acrLoginServer:loginServer}" --output table



$ docker tag mcr.microsoft.com/azuredocs/azure-vote-front:v1 jpvotingappacr.azurecr.io/azure-vote-front:v1

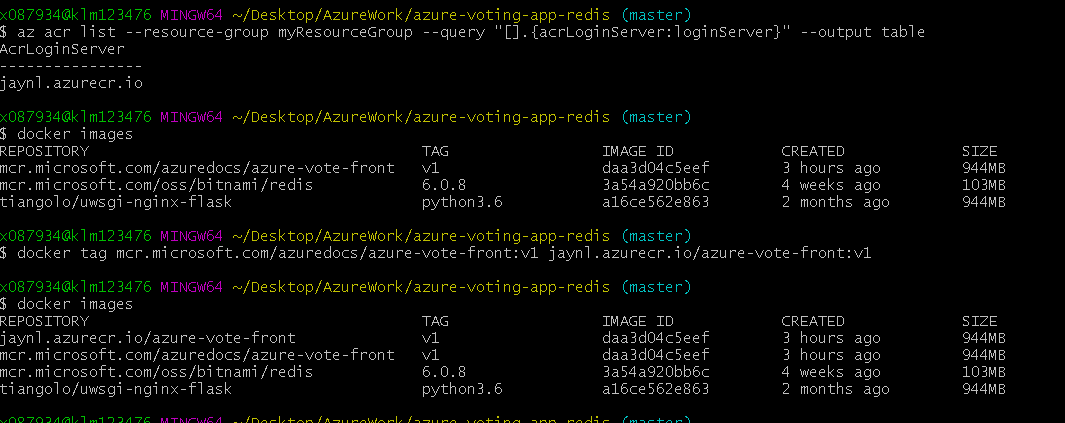
**Push image to registry**

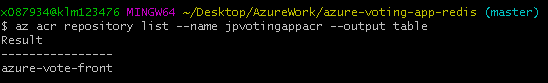
$ docker push jpvotingappacr.azurecr.io/azure-vote-front:v1

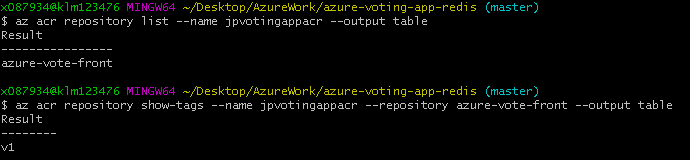


**List images in the registry**

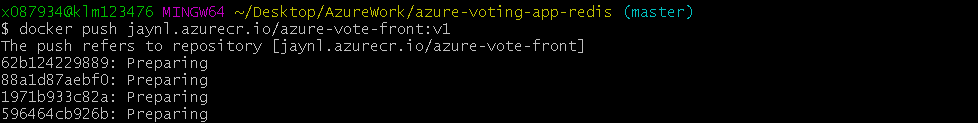
$ az acr repository list --name jpvotingappacr --output table

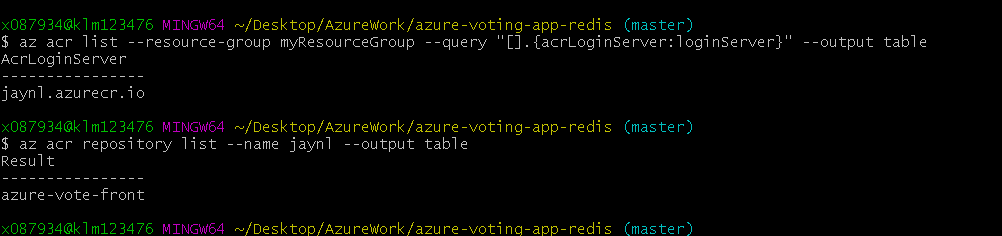






**An image is tagged with the ACR instance address and a version number.**





# 3: Deploy an Azure Kubernetes Service (AKS) cluster

**Create a Kubernetes cluster**

$ az aks create \

--name jpnlAKSCluster \

--resource-group VotingAppResourceGroup \

--attach-acr jpvotingappacr \

--node-count 2 \

--generate-ssh-keys \

--node-vm-size Standard\_B2s \

--load-balancer-sku basic \

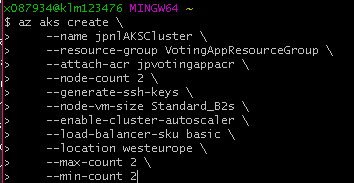
--location westeurope \

--enable-cluster-autoscaler \

--max-count 2 \

--min-count 2

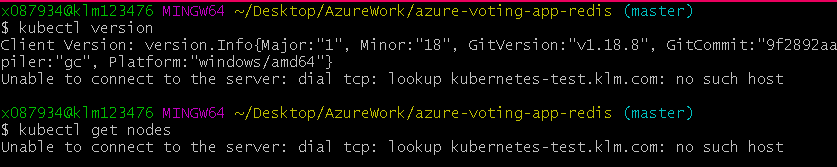
<https://azure.microsoft.com/hu-hu/blog/introducing-burstable-vm-support-in-aks/>



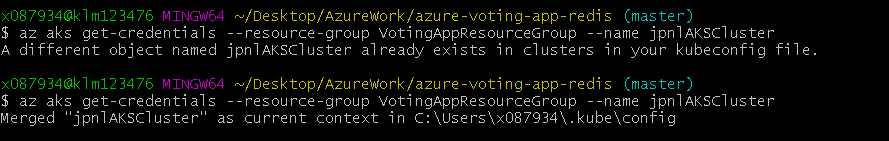
Now cluster is created.

Try to perform kubectl.

It connected to my local kubeconfig file, which is not connecting to Azure’s network.

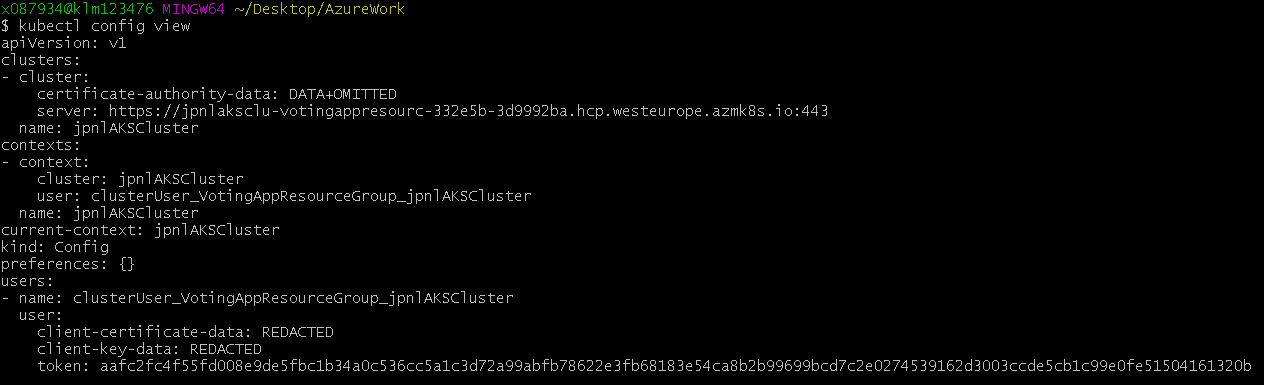


So I can call aks get-credentials with Override command. Alternatively, I can delete the kubeconfig file and recreate new kube config file



Once merged, check the kubeconfig

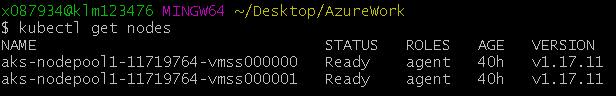
$ kubectl config view





List the k8s nodes

$ kubectl get nodes

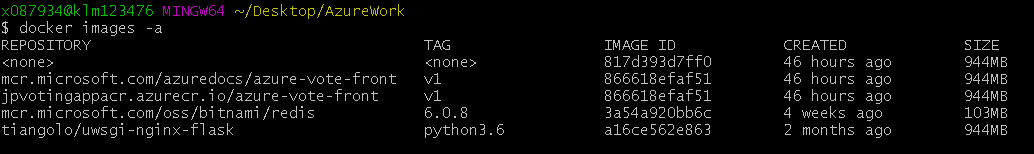


These 2 nodes are created at the start of cluster creation

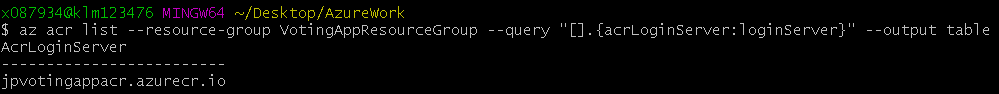
# 4: Run applications in Azure Kubernetes Service (AKS)

ACR has the container image for the application.

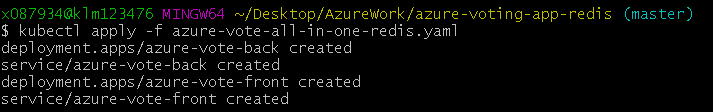
The k8s yaml file (deployment) retrieves image from Microsoft mcr, which must be updated to our own Azure’s ACR.



$ az acr list --resource-group VotingAppResourceGroup --query "[].{acrLoginServer:loginServer}" --output table

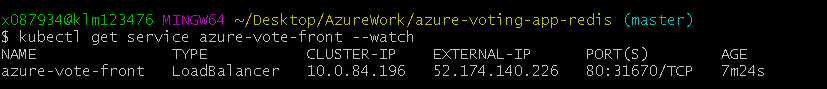


$ kubectl apply -f azure-vote-all-in-one-redis.yaml

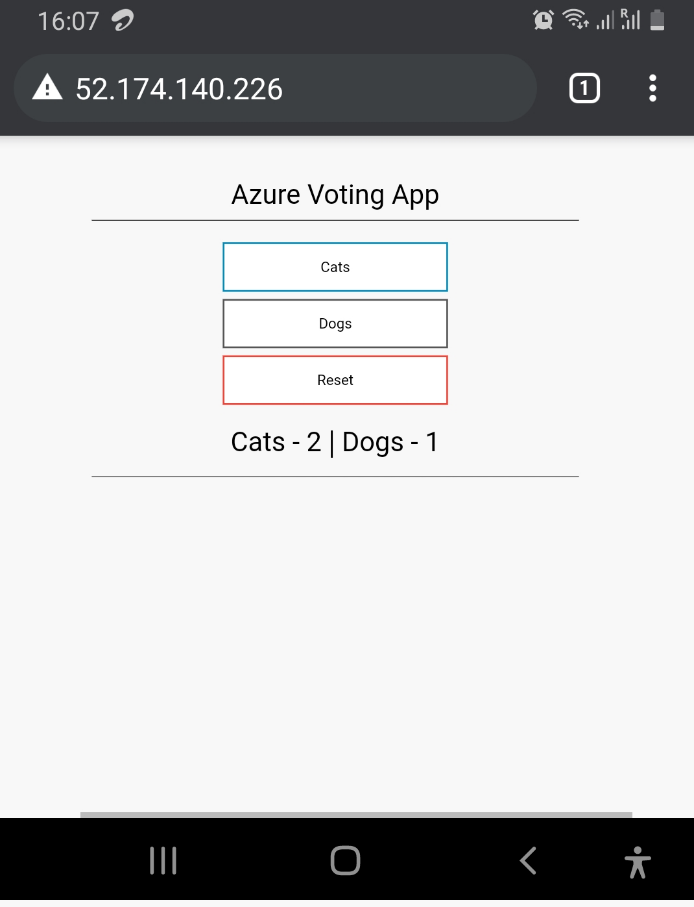


kubectl get service command with the --watch argument provides monitoring and exposes the external IP address where the application hosted

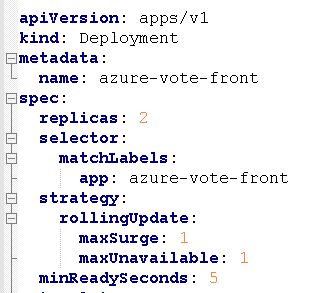
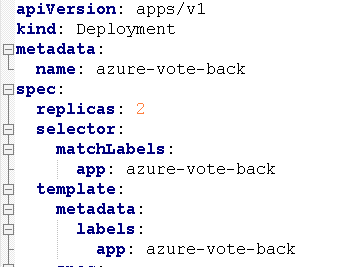
$ kubectl get service azure-vote-front --watch



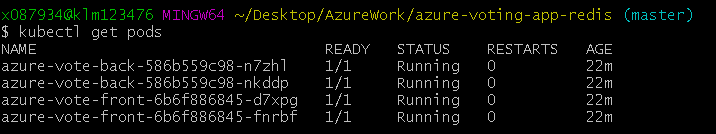
Now you will see the application running with the dedicated IP.



In the deployment YAML, I updated the replica count to 2.

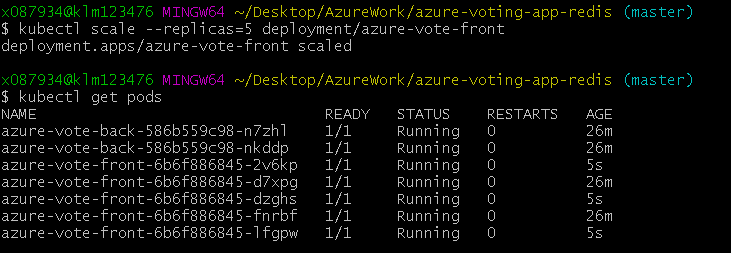


So you will see total 4 pods (2\*2)



# 5: Scale applications in Azure Kubernetes Service (AKS)

$ kubectl scale --replicas=5 deployment/azure-vote-front



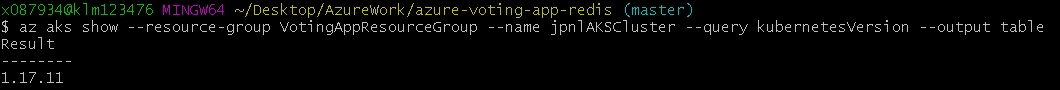
## Autoscale pods

K8s supports horizontal pod autoscaling

The Metrics Server is used to provide resource utilization to Kubernetes, and is automatically deployed in AKS clusters versions 1.10 and higher.

So first we need to find the current version of k8s used by our AKS.

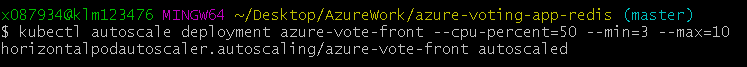
$ az aks show --resource-group VotingAppResourceGroup --name jpnlAKSCluster --query kubernetesVersion --output table

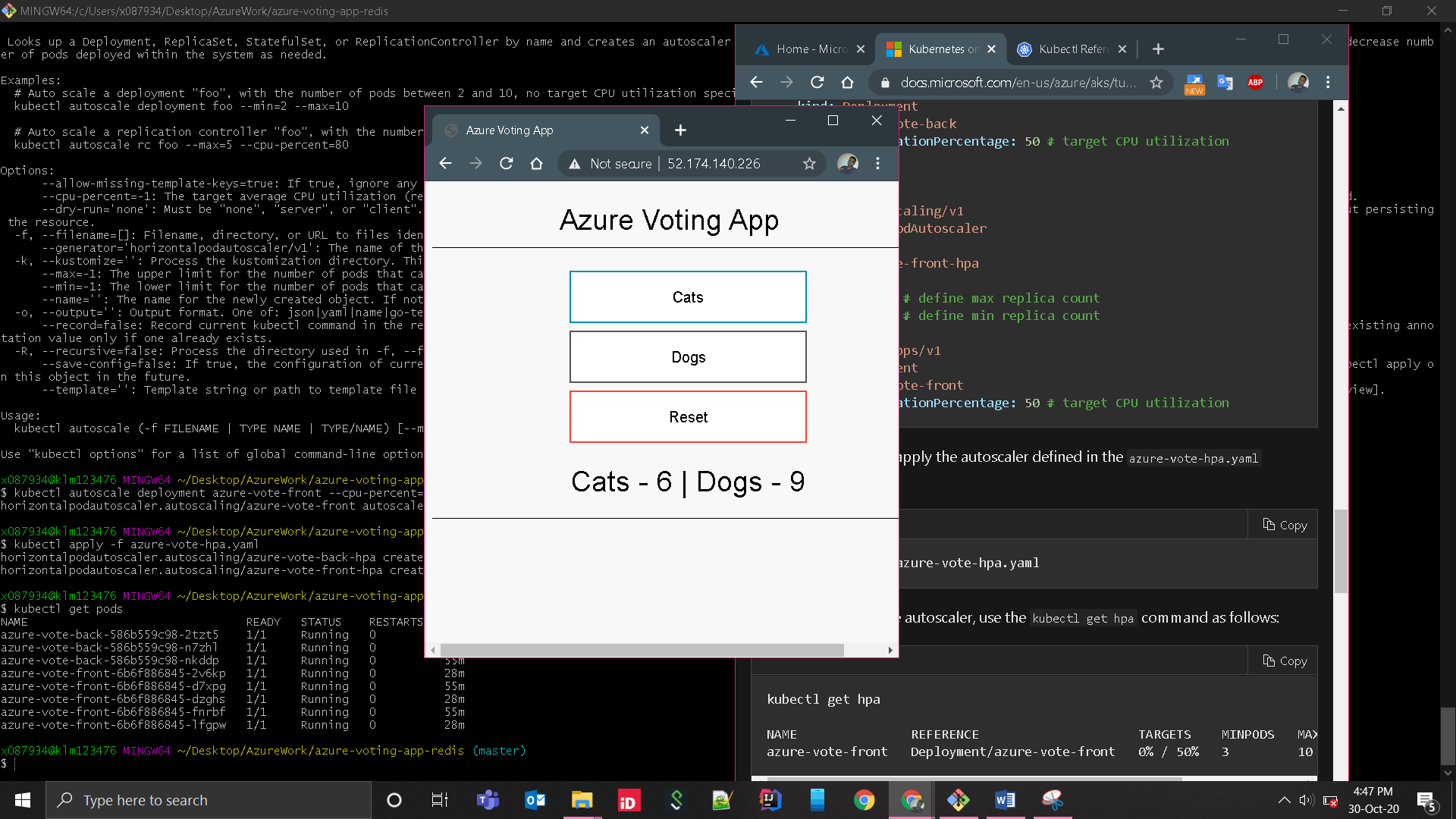


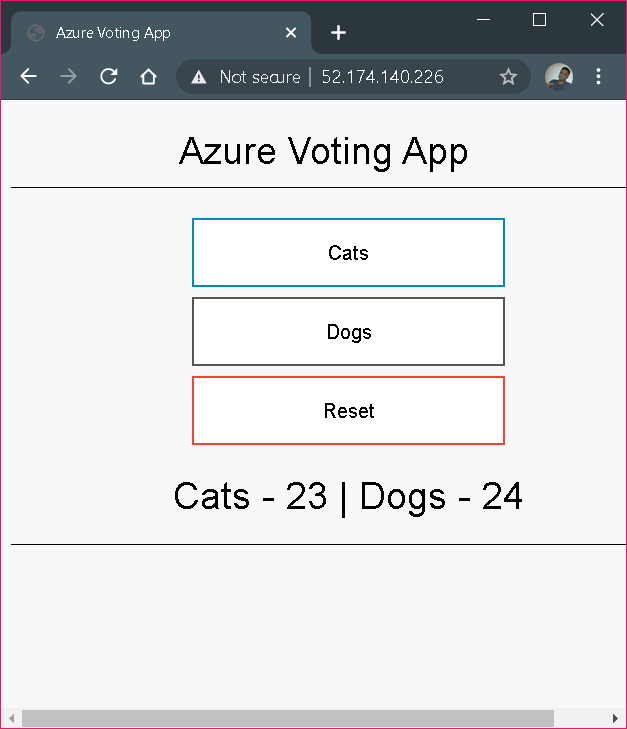
Examples:

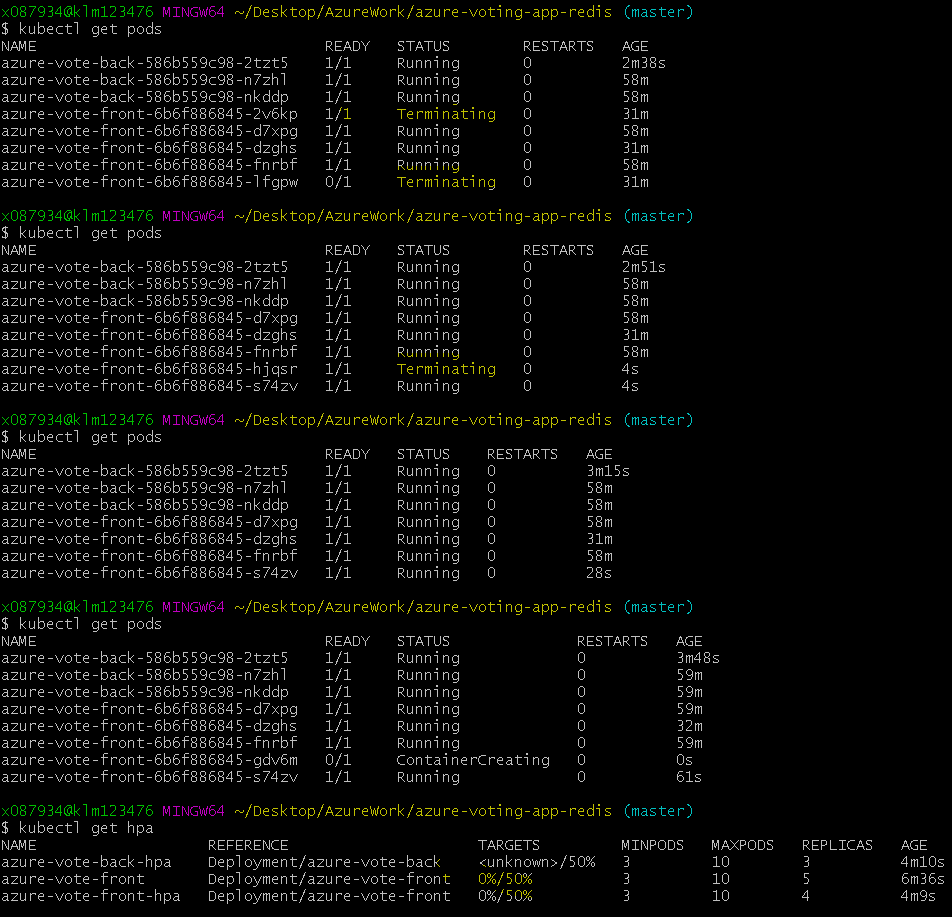
# Auto scale a deployment "foo", with the number of pods between 2 and 10, no target CPU utilization specified so a default autoscaling policy will be used:

kubectl autoscale deployment foo --min=2 --max=10

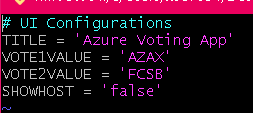


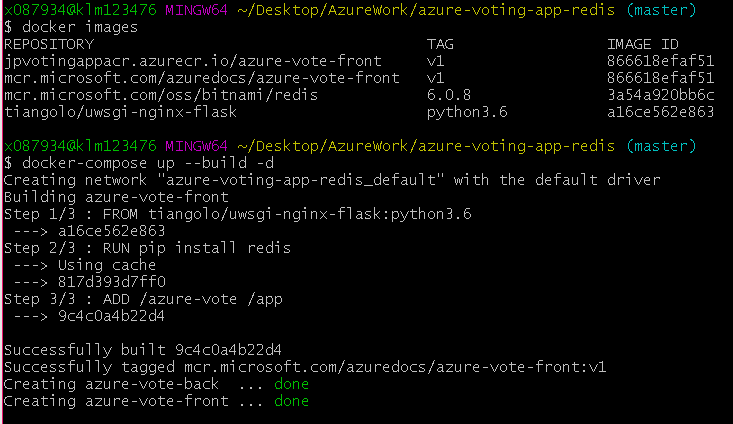




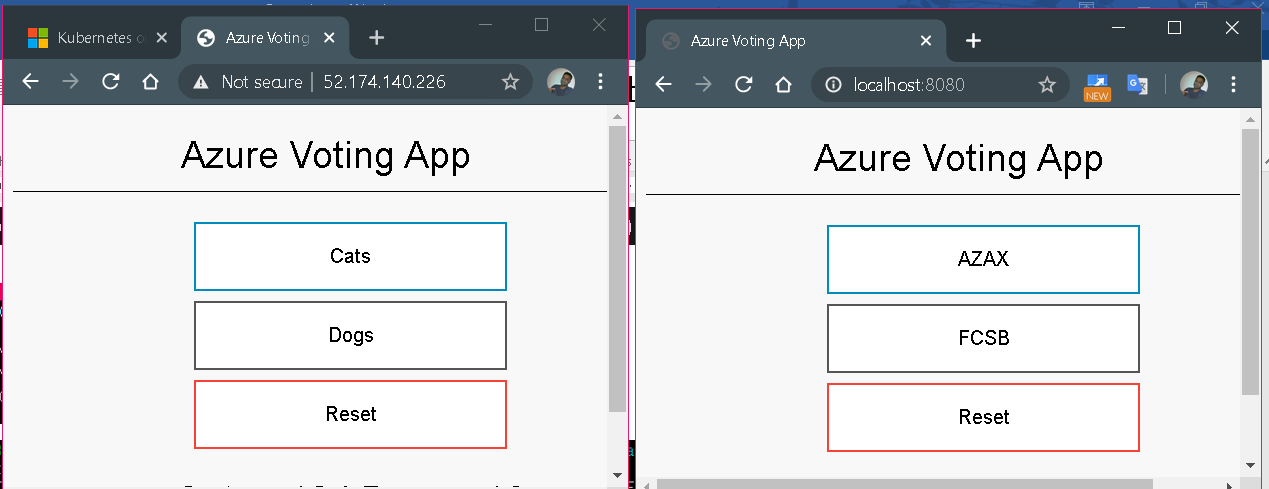


# 6: Update an application in Azure Kubernetes Service (AKS)

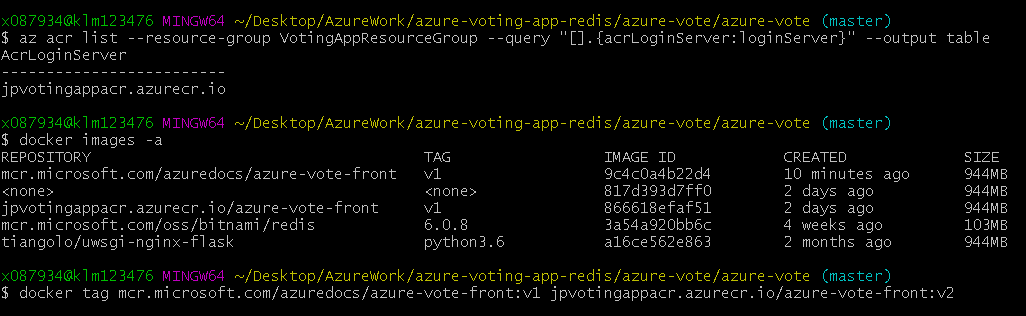




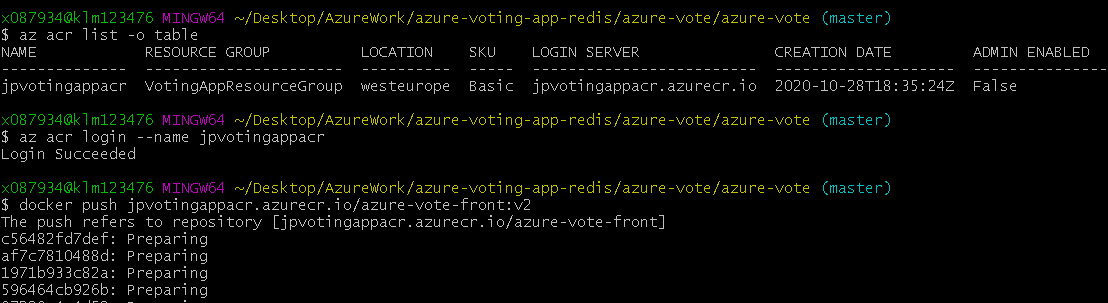
The localhost with updated value

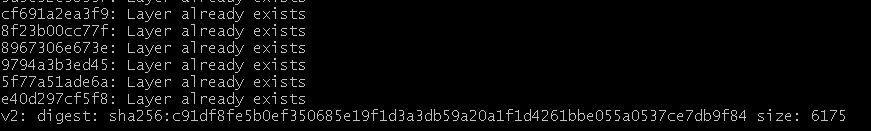


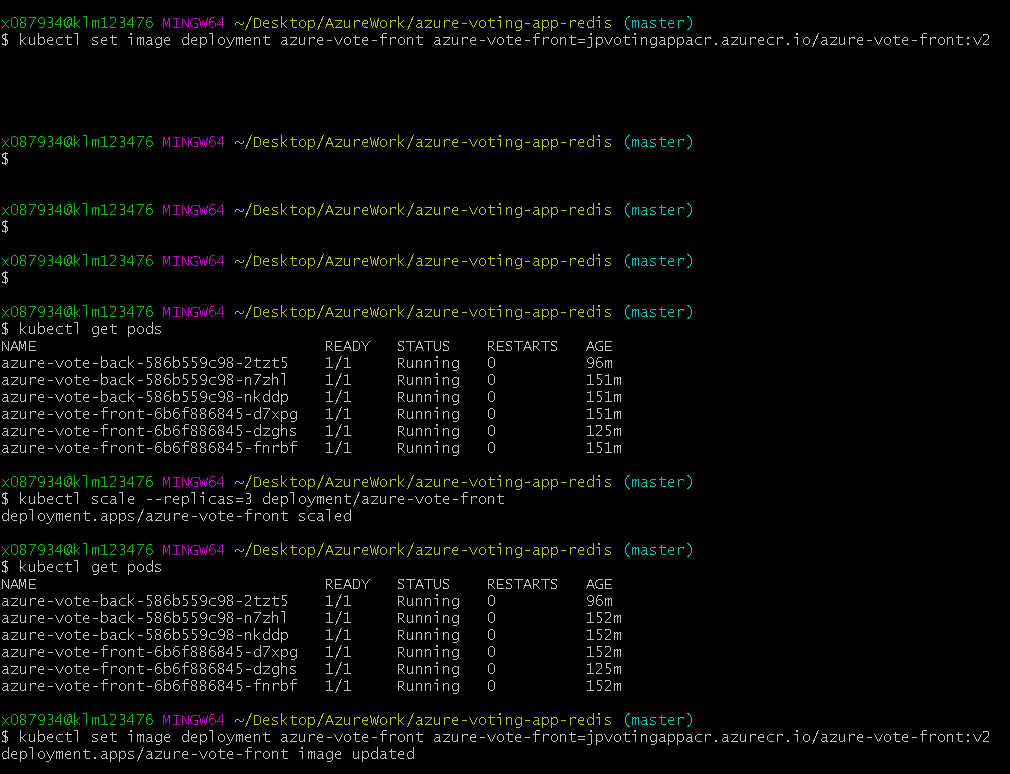
Tag the image with new version number



Push the image to Docker registry







When deploying v2 app, kubectl set image took very long time.

I performed some kubeclt scale operation (No need to set scale to same value)

kubectl scale --replicas=3 deployment/azure-vote-front

Then set image operation performed successfully and immediately





