# Learning AKS Lab 2

Deploying the Voting App Using kubectl With ConfigMaps, ENV Vars, PV/PVC, Probes, and Blue/Green.

## Prerequisites

- Have an AKS cluster installed (see https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal)
- Install the kubectl command line tool (see https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-deploy-cluster#install-the-kubernetes-cli)
- Connect kubectl to your AKS cluster (see https://docs.microsoft.com/en-us/azure/aks/tutorial-kubernetes-deploy-cluster#connect-to-cluster-using-kubectl)
- VSCode installed for YAML editing (see https://code.visualstudio.com/Download).
    - You can use the YAML editing IDE of your choice. This lab uses VSCode as it has YAML support along with an integrated terminal.

## Set Up Manifest Directory

We're going to create a few YAML files, so let's create a directory where we'll store them.

In our case, we'll create a directory at

```
C:\code\git\aks-training\part2\yaml\voting-app-lab
```

We'll use this directories in the rest of this lab. If you create a different location, please use that.

We'll also use VSCode for this lab

## Create Namespace

Let's create a namespace YAML file for our namespace. We'll use the name voting-lab and we'll use kubectl to create this for us:

```
kubectl create ns voting-lab --dry-run=client -o yaml > sample-voting-ns.yaml
```

**Note:** *`--dry-run=client` is for `kubectl` version 1.18 and greater; if you're version of `kubectl` is earlier, just use `--dry-run`.*

After running the command, your YAML file for the namespace will look like this:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: null
  name: voting-lab
spec: {}
```

```
status: {}
```

You can then apply this to your AKS cluster by issuing this command:

```
kubectl apply -f sample-voting-ns.yaml
```

You'll see this output:

```
namespace/voting-lab configured
```

Confirm the namespace exists by executing this command:

```
kubectl get ns
```

Which will produce this output, which should include the voting-lab namespace (highlighted):

```
NAME              STATUS   AGE
default           Active   11d
kube-node-lease   Active   11d
kube-public       Active   11d
kube-system       Active   11d
voting-lab        Active   10d
```

## Create ConfigMap for Redis Location

Let's create a ConfigMap object to hold our Redis service location

- Using the command line, issue this command to generate a ConfigMap YAML file
  kubectl create cm redis-map --from-literal=REDIS=azure-vote-back.voting-lab -n voting-lab --dry-run=client -o yaml > redis-map.yaml
- Examine the generated YAML file, as it should look similar to this:

```
apiVersion: v1
data:
  REDIS: azure-vote-back.voting-lab
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: redis-map
  namespace: voting-lab
```

- Apply the ConfigMap to your AKS cluster using the `kubectl` tool:
  `kubectl apply -f redis-map.yaml`
- You can confirm that the ConfigMap was created by issuing this command:
  `kubectl get cm -n voting-lab`

Which produces:

```
NAME        DATA    AGE
redis-map   1       11s
```

## Create PersistentVolume for Redis Persistence

Now we'll create a PersistentVolume for Redis Persistence. This will create an underlying Azure Storage Account for your Redis data.

- We'll create a YAML file named redis-pvc.yaml that will create a PersistentVolumeClaim (PVC).

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: redis-store
  namespace: voting-lab
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile
  resources:
    requests:
      storage: 5Gi
```

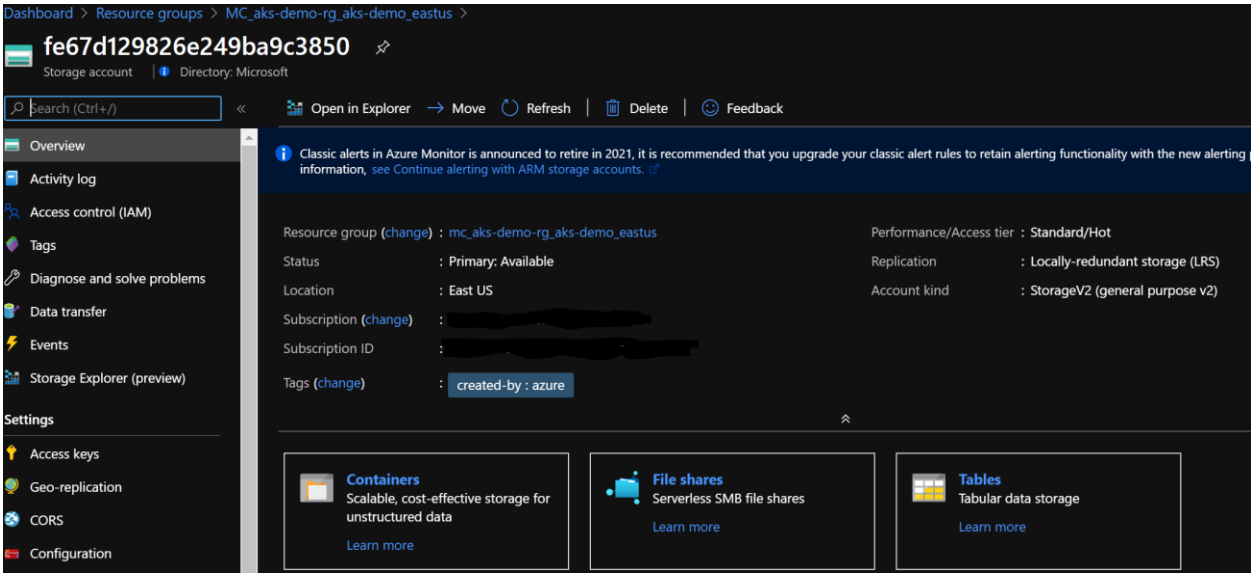
- Apply the PVC YAML file to your AKS cluster:
  kubectl apply -f redis-pvc.yaml
- Check the status that the PersistentVolume was created:

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
pvc-b805b214-2574-4ebc-af31-ac8e5ce78367   5Gi        RWX            Delete           Bound    voting-lab/redis-store  azurefile              10d
```

- Check the status of the PersistentVolumeClaim:

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get pvc -n voting-lab
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
redis-store   Bound    pvc-b805b214-2574-4ebc-af31-ac8e5ce78367   5Gi        RWX            azurefile      10d
```

- You should also see a new Storage Account show up in your MC_XXX resource group for your AKS cluster:




## Deploy the Backend

Now we'll deploy the Redis backend to our cluster. Use the following YAML and save it to sample-voting-back-deploy.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
  namespace: voting-lab
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      containers:
        - name: azure-vote-back
          image: redis
          ports:
            - containerPort: 6379
              name: redis
          volumeMounts:
            - mountPath: /data
              name: data
      volumes:
```

```
        - name: data
          persistentVolumeClaim:
            claimName: redis-store
```

Note that in the YAML, we're using the PVC we created in the previous step for the data mount. Also note that this deployment is targeting the voting-lab namespace, just like the other resources we've created.

Apply this YAML to your cluster using `kubectl`:

```
kubectl apply -f sample-voting-back-deploy.yaml
```

Confirm that the deploy was successful by issuing this `kubectl` command:

```
kubectl get all -n voting-lab
```

You should see one pod in a Running state, one deployment, and one replicaset.

## Create Backend Service

Now that we've created the Redis pod and replicaset, now we'll create an internal service so that other Kubernetes objects can call it.

In VSCode, copy this YAML and save it as sample-voting-back-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
  namespace: voting-lab
spec:
  ports:
    - port: 6379
  selector:
    app: azure-vote-back
  type: ClusterIP #default value
```

Apply the YAML to your cluster by using kubectl:

kubectl apply -f sample-voting-back-svc.yaml

And check the status of your newly created service by using kubectl:

kubectl get svc -n voting-lab

You should see one service running:

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ k get svc -n voting-lab
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
azure-vote-back    ClusterIP      10.0.188.163   <none>          6379/TCP       10d
```

You can also check the Endpoint created by issuing this command:

kubectl get endpoints -n voting-lab

And you should see the following endpoints:

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get endpoints -n voting-lab
NAME               ENDPOINTS                                                  AGE
azure-vote-back    10.244.1.37:6379                                           10d
```

And the endpoint should match the IP of your running Redis pod:

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get pod -n voting-lab -l app=azure-vote-back -o wide
NAME                               READY   STATUS    RESTARTS   AGE   IP            NODE
azure-vote-back-69569fbb68-8jqbc   1/1     Running   0          10d   10.244.1.37   aks-agentpool-51725664-vmss000000
```

## Deploy the Frontend

Now we'll do something similar with the Frontend of our application. Let's use the following YAML to create our frontend web app:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front-blue
  namespace: voting-lab
spec:
  replicas: 1     #NOTE 0 - replica count
  selector:
    matchLabels:
      app: azure-vote-front
      env: blue
  template:
    metadata:
      labels:  #NOTE 1 - multiple labels
        app: azure-vote-front
        env: blue
    spec:
      containers:
        - name: azure-vote-front
          image: microsoft/azure-vote-front:v1
          ports:
```

```
        - containerPort: 80
      livenessProbe:    #NOTE 2 - liveness probe
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 5
        periodSeconds: 5
      readinessProbe:   #NOTE 3 - readiness probe
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 10
        periodSeconds: 30
      resources:          #NOTE 4 - resource specification
        requests:
          cpu: 250m
        limits:
          cpu: 500m
      env:                #NOTE 5 - ENV Variable settings
        - name: REDIS
          valueFrom:
            configMapKeyRef:
              key: REDIS
              name: redis-map
```

There's a lot going on here, so let's take things one step at a time by referring to the "NOTE" comments in the YAML:

- Note 0 - just noting the replica count as we'll be modifying this later on.
- Note 1 - multiple labels are applied to this deployment. Notice that one is a pretty standard "app" label, but there's also an "env" label. This is used to specify what environment we're deploying to this -- blue or green. We'll use this in Challenge #1 below.
- Note 2 - Liveness Probe definition is so that Kubernetes knows whether this pod is healthy or not. The probe hits that root of the website 5 seconds after the pod starts up ("initialDelaySeconds") and then every 5 seconds after that ("periodSeconds").
- Note 3 - Readiness Probe definition is so that Kubernetes knows if this pod is healthy to serve traffic via a Service (which we'll create in the next step). Like the liveness probe, this probe hits the root of the web site but waits for 10 seconds after the pod is ready and then hits the root every 30 seconds thereafter
- Note 4 - Resource specifications allow Kubernetes to allocate space in the node pool for the pods it's scheduling. Knowing how much CPU (specified in the example) and Memory (not specified) allows the Kubernetes Scheduler to determine how many pods to place on nodes and when Kubernetes needs to scale out the nodes. This is a useful setting that we'll discuss more in week 4.

- Note 5 - Environment Variables being set from the ConfigMap we created earlier. Note that we're specifying the ENV Var we want to set explicitly from the ConfigMap. In Challenge #2, you'll find other ways to specify ENV Vars.

Let's apply this Deployment using kubectl:

```
kubectl apply -f sample-voting-front-deploy-blue.yaml
```

Confirm that the deployment, pods, and replicaset were created using:

```
kubectl get all -n voting-lab
```

You'll see your front and back end pods, deployments and replicasets.

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get all -n voting-lab
NAME                                        READY   STATUS    RESTARTS   AGE
pod/azure-vote-back-69569fbb68-8jqbc        1/1     Running   0          10d
pod/azure-vote-front-blue-5d554c77b-wc6sh   1/1     Running   0          23m

NAME                       TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/azure-vote-back    ClusterIP   10.0.188.163   <none>        6379/TCP   10d

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/azure-vote-back         1/1     1            1           10d
deployment.apps/azure-vote-front-blue   1/1     1            1           6d13h

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/azure-vote-back-69569fbb68        1         1         1       10d
replicaset.apps/azure-vote-front-blue-5d554c77b   1         1         1       23m
replicaset.apps/azure-vote-front-blue-7454d5c8d9   0         0         0       6d13h
```

## Create the Frontend Service

Now that we have a front end and back end deployment, let's expose the front end service so that we can hit it from a web browser. We'll do this by creating a service that will create an Azure Public IP address -- we'll do this by creating a service of type `LoadBalancer`.

In VSCode, create a file named sample-voting-front-svc.yaml and use this YAML:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
  namespace: voting-lab
spec:
  type: LoadBalancer
  ports:
    - port: 80
  selector:
    app: azure-vote-front
    env: blue
```

Notice that we're specifying two selectors to this service

- app is equal to 'azure-vote-front'
- env is equal to 'blue'

Type is set to `LoadBalancer`, and we're exposing this service over port 80. Let's apply this to our AKS cluster via `kubectl`:

```
kubectl apply -f sample-voting-front-svc.yaml
```

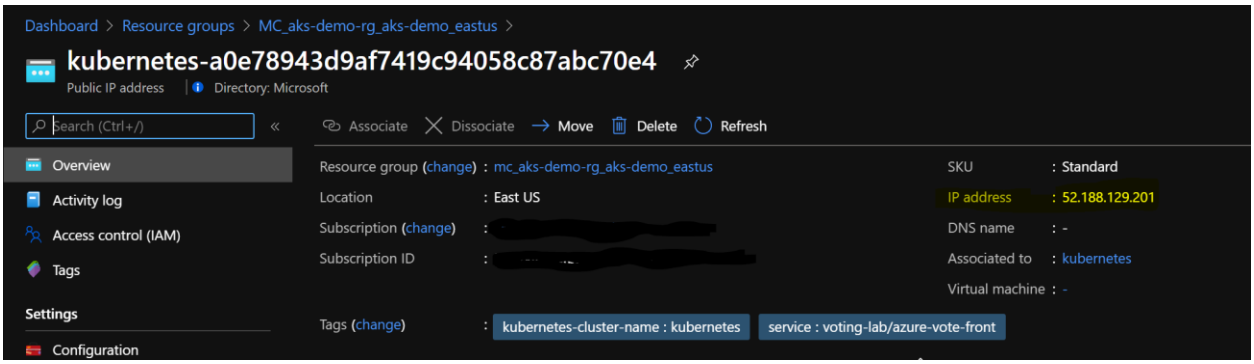
```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl apply -f sample-voting-front-svc.yaml
service/azure-vote-front created
```

You'll see the service was successful by using kubectl again:

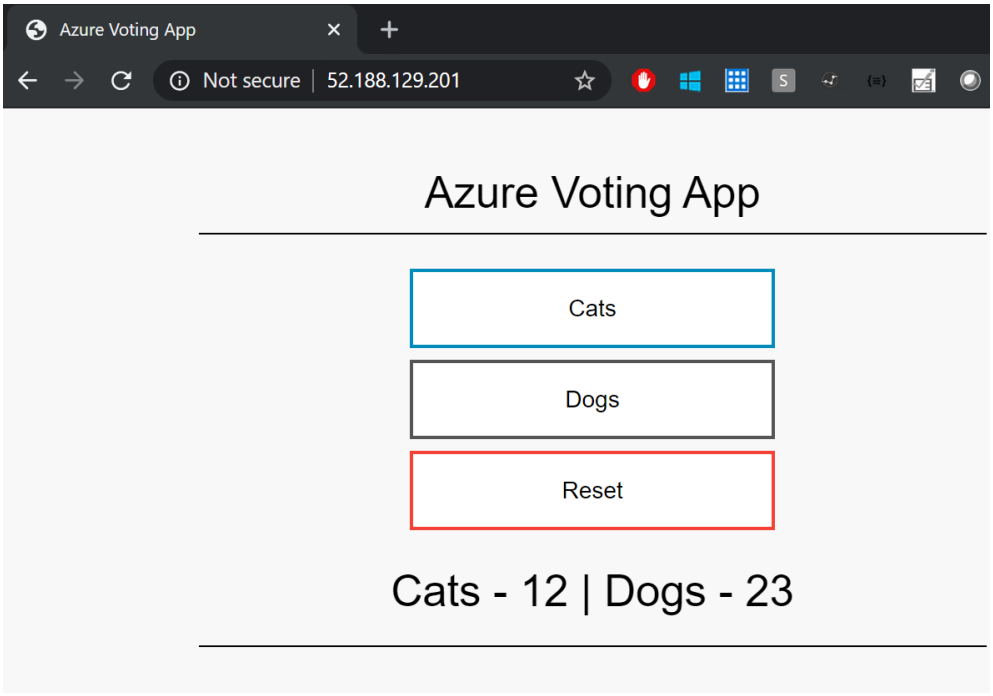
```
kubectl get svc -n voting-lab
```

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get svc -n voting-lab
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
azure-vote-back    ClusterIP      10.0.188.163   <none>           6379/TCP       10d
azure-vote-front   LoadBalancer   10.0.230.76    52.188.129.201   80:31563/TCP   38s
```

You'll also see that, in your MC_XXX resource group in Azure, a new Public IP was created that will match the External IP from the kubectl output above.




After this has been created, open up a browser window and go to the public/external IP address for your service (in this case, I'm going to http://52.188.129.201). You should see the voting page:




## Scale the Frontend

Now that we've deployed our Cats and Dogs voting app, it's become very popular. As a result, we need to scale out the front end in order to handle the volume. We decide we'll scale the front end from 1 pod to 3 pods. We can accomplish that easily by modifying the YAML for our front end deployment (sample-voting-front-deploy-blue.yaml), as such:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front-blue
  namespace: voting-lab
spec:
  replicas: 3
  selector:
    matchLabels:
      app: azure-vote-front
      env: blue
```

We've made one change - we changed the replicas line from '1' to '3'. Once we do that in our editor, save the file and then apply it to our AKS cluster as such:

```
kubectl apply -f sample-voting-front-deploy-blue.yaml
```

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl apply -f sample-voting-front-deploy-blue.yaml
deployment.apps/azure-vote-front-blue configured
```

When we issue this command, we can check our deployment's status using `kubectl`:

`kubectl get all -n voting-lab`

```
c:\code\git\aks-training\part2\yaml\voting-app-lab
λ kubectl get all -n voting-lab
NAME                                       READY   STATUS    RESTARTS   AGE
pod/azure-vote-back-69569fbb68-8jqbc       1/1     Running   0          10d
pod/azure-vote-front-blue-5d554c77b-csnnf  1/1     Running   0          33s
pod/azure-vote-front-blue-5d554c77b-wc6sh  1/1     Running   0          40m
pod/azure-vote-front-blue-5d554c77b-xflr5  1/1     Running   0          33s

NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
service/azure-vote-back    ClusterIP      10.0.188.163   <none>           6379/TCP       10d
service/azure-vote-front   LoadBalancer   10.0.230.76    52.188.129.201   80:31563/TCP   11m

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/azure-vote-back       1/1     1            1           10d
deployment.apps/azure-vote-front-blue 3/3     3            2           6d13h

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/azure-vote-back-69569fbb68       1         1         1       10d
replicaset.apps/azure-vote-front-blue-5d554c77b  3         3         3       40m
replicaset.apps/azure-vote-front-blue-7454d5c8d9 0         0         0       6d13h
```

Notice that the front end service's IP hasn't changed (52.188.129.201 in this example), so you can continue to hit the site while the deployment scale out is going on, and the site will continue to run. You can do the same with scaling in the deployment, too. Just edit the deployment YAML file and apply it to the AKS cluster using kubectl apply.

## Challenge #1 – Create Blue/Green

Our first challenge is to create a second deployment that acts as the 'green' deployment. What you should do is:

- Create a new deployment YAML that is a copy of sample-voting-front-deploy-blue.yaml, but name it sample-voting-front-deploy-green.yaml. We'll call this new deployment file the "green" one.
- In the "green" deployment, update the image of the voting app to use the "v2" tag instead of "v1".
- In the "green" deployment, update the env label to have a value of "green"
- Apply this deployment file to your AKS cluster. Check that it was successfully deployed. Notice that you can't reach the pods via the service.
    - Check the service's endpoints (kubectl get endpoints -n voting-lab). Notice they only point to the pods from the "blue" deployment.

    - You can access one of the “green” pods by port-forwarding to it (kubectl port-forward azure-vote-front-green-<hash> -n voting-lab 8888:80) and then browsing to http://localhost:8888.
    - When you browse to the port-forwarded location, you see voting for Blue and Purple, not Cats and Dogs.
- The service (azure-vote-front) is only forwarding requests to the pods that match its selectors.
- CHALLENGE: Can you get the azure-vote-front service to point to the green pods instead of the blue pods?
    - Can you do this in a way you can then switch back to the “blue” pods if you don’t like the changes that “green” made?
    - Can you see how this is a useful deployment strategy for rolling back to previous versions of deployed applications?

## Challenge #2 - Set ENV Variables Differently

In this lab, we only had one ENV Var (“REDIS”) which was the location of the redis service (“azure-vote-back.voting-lab”). Instead of individually setting each ENV Var, can you map every key/value in our ConfigMap as ENV Vars in the Redis pod? Feel free to add a few more key/value pairs to the ConfigMap if you’d like, too.

As a hint, take a look at the `envFrom` keyword that you can use in your Pod spec. (https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#configure-all-key-value-pairs-in-a-configmap-as-container-environment-variables)

## Challenge #3 – Move Redis Deploy to a StatefulSet

Check out the documentation for a StatefulSet here:
https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

A StatefulSet allows for predictable pod creation and pod naming, which can be useful for a clustered application scenario where pod naming is important.

There are a few examples on the Internet where a Redis environment is set up as a StatefulSet. Can you create something similar for the voting app?