ConfigMap

Create ConfigMap for Redis Cache

```
kubectl create configmap example-redis-config --from-file=config/redis-config
```

Verify ConfigMap looks correct

```
kubectl get configmap example-redis-config -o yaml
```

It should look something like

```
apiVersion: v1
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
kind: ConfigMap
metadata:
  creationTimestamp: 2016-03-30T18:14:41Z
  name: example-redis-config
  namespace: default
  resourceVersion: "24686"
  selfLink: /api/v1/namespaces/default/configmaps/example-redis-config
  uid: 460a2b6e-f6a3-11e5-8ae5-42010af00002
```

Create Redis POD that uses ConfigMap

```
kubectl create -f manifests/redis-pod.yaml
```

Confirm Redis POD launched with ConfigMap settings

```
kubectl exec -it redis redis-cli
CONFIG GET maxmemory
```

Should see something like this

```
1) "maxmemory"
2) "2097152"
127.0.0.1:6379> CONFIG GET maxmemory-policy
1) "maxmemory-policy"
2) "allkeys-lru"
```

Cleanup

```
kubectl delete pod redis
```

ConfigMap Nginx

Now that you've had some experience creating a ConfigMap, let's build on that and setup an Nginx reverse proxy that sends traffic to a NodeJS web application.

Doing things on your own

Create ConfigMap

Look in the `config` directory and you will see `reverseproxy.conf` which contains the configuration for our Nginx reverse proxy.

1. Create a ConfigMap with this file called `nginx-config`
2. Create `nginx.yaml` to create a Pod with the following attributes:
   - Container 1
     - name: `helloworld-nginx`
     - label: `app: helloworld-nginx`
     - image: `nginx:1.15`
     - port: `80`
     - Mount configMap `nginx-config` to `/etc/nginx/conf.d` with

        - key of `reverseproxy.conf`
        - path of `reverseproxy.conf`
    - Container 2
        - name: `k8s-demo`
        - image: `aslaen/k8s-demo`
        - port `3000`
3. Create `nginx-service.yaml` to expose the application
    - name: `helloworld-nginx-service`
    - port: `80`
    - protocol: `TCP`
    - Selector: `app: helloworld-nginx`
    - type: `NodePort`

Validate

Now run `curl` to connect to server on port 80 and confirm you get

```
* Rebuilt URL to: 0:80/
*   Trying 0.0.0.0...
* TCP_NODELAY set
* Connected to 0 (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: 0
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.15.9
< Date: Thu, 28 Feb 2019 05:55:55 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 12
< Connection: keep-alive
< X-Powered-By: Express
< ETag: W/"c-7Qdih1MuhjZehB6Sv8UNjA"
<
* Connection #0 to host 0 left intact
Hello World!
```

We can see that it connects to Nginx and then returns `Hello World!` which is what we've

defined in our Node.js app in the `k8s-demo` container.

Now to confirm the traffic is actually being passed through the proxy update the ConfigMap to pass traffic to port `3001` and redeploy. You should see it fail to connect.

Lab Complete!