Externalize Configuration in Kubernetes

Learning Outcomes

After completing the lab, you will be able to:

1. Use environment variables to configure an application running locally
2. Describe how to configure a Spring Boot application on Kubernetes with ConfigMap

Before starting the lab, cherry pick the config-start tag.

```
git cherry-pick config-start --strategy-option theirs
```

In case you get an error when you cherry-pick, open `intellij`, right-click on the `project`, select `git → resolve-conflicts → accept theirs`

Externalize the welcome message

1. As a result of cherry-pick some test cases have been updated. Have a look at HomeControllerTest class.
2. Externalize the message being displayed into an environment variable called `PAGE_CONTENT`
3. The goal is to ensure that your test cases are passing before dockerizing.
4. Update `HomeController` class to externalize the welcome message through constructor injection

```
    private String pageContent;

    public HomeController(@Value("${page.content}") String pageContent){
        this.pageContent=pageContent;
    }

    @GetMapping
    public String getPage(){
        return "Hello from page : "+pageContent+" ";
    }
```

5. Set the environment variable in `build.gradle` file for test and dev environments

```
    bootRun.environment([
                "PAGE_CONTENT": "YellowPages",
    ])

    test.environment([
                "PAGE_CONTENT": "YellowPages",
    ])
```

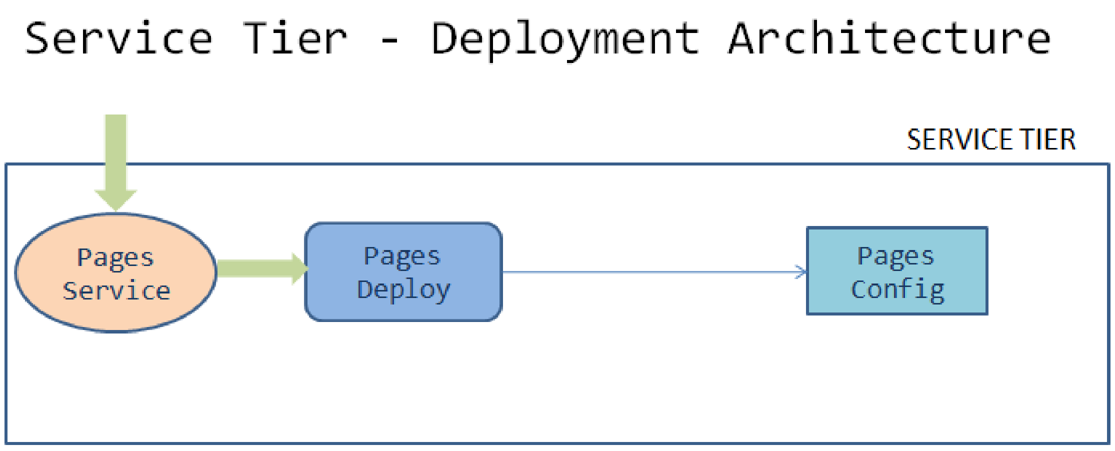
6. Build and test the application just like you did in previous labs.
7. Dockerize the application and tag it appropriately

```
docker build -t [docker_username]/pages:[tag] .
```

8. Push the image to docker

```
docker push [docker_username]/pages:[tag]
```

Kubernetize the application



Below are the 2 tasks to be completed within the kubernetes cluster

1. Create a config map object with the following specifications.

```
name -> page-config-map
namespace -> [student-name]
```

```
config data key-value pair:
PAGE_CONTENT -> "Green-Pages from Yellow World!"
```

2. Edit the pages deployment in the namespace [student-name] & add an environment variable to the container

```
name -> PAGE_CONTENT
value -> fetch it from configmap name page-config-map [key
is PAGE_CONTENT]
```

Implement the above tasks by creating manifest/yaml files

1. Create `deployment/pages-config.yaml`

```
apiVersion: v1
data:
  PAGE_CONTENT: Green-Pages coming from Yellow-World!
kind: ConfigMap
metadata:
  name: pages-config-map
  namespace: [student-name]
```

2. Update `deployment/pages-deployment.yaml`

```
      containers:
        - image: [docker-username]/pages:[tag]
          name: pages
          ports:
            - containerPort: 8080
          env:
            - name: PAGE_CONTENT
              valueFrom:
                configMapKeyRef:
                  name: pages-config-map
                  key: PAGE_CONTENT
```

3. Start minikube locally `minikube start --driver=virtualbox`
4. Verify the kubectl context `kubectl config get-contexts` is set to minikube. If not, set it to minikube `kubectl config use-context minikube`
5. Deploy and test the application in minikube. Refer to Deployment Guide

Deploy the application to production cluster

1. Follow Production Cluster Guide to login/connect to the production cluster.
2. Deploy and test the application in production cluster. Refer to Deployment Guide
3. Commit code changes to the github repository

```
git add .
git commit -m "Externalized config"
git push -u origin master
```

Deployment Guide

1. Set up `[student-name]` namespace to point to the current context

```
kubectl config set-context --current --namespace=[student-
name]
```

2. Create kubernetes objects

```
kubectl apply -f deployment/pages-config.yaml
kubectl apply -f deployment/pages-deployment.yaml
```

3. Verify the created objects

```
kubectl get deployment pages
kubectl get configmap pages-config-map
kubectl get pods -o wide
```

4. Access the pages application by port-forwarding using kubectl, enabling the application can be served via `localhost` on port `8080`

```
kubectl port-forward svc/pages 8080:8080

curl localhost:8080
```

Advanced usecase challenges

1. Enable refresh scope for automatic synchronization of config map properties in spring boot
2. Watch for changes in kubernetes api resources and objects, for dynamic loading
3. Security concerns