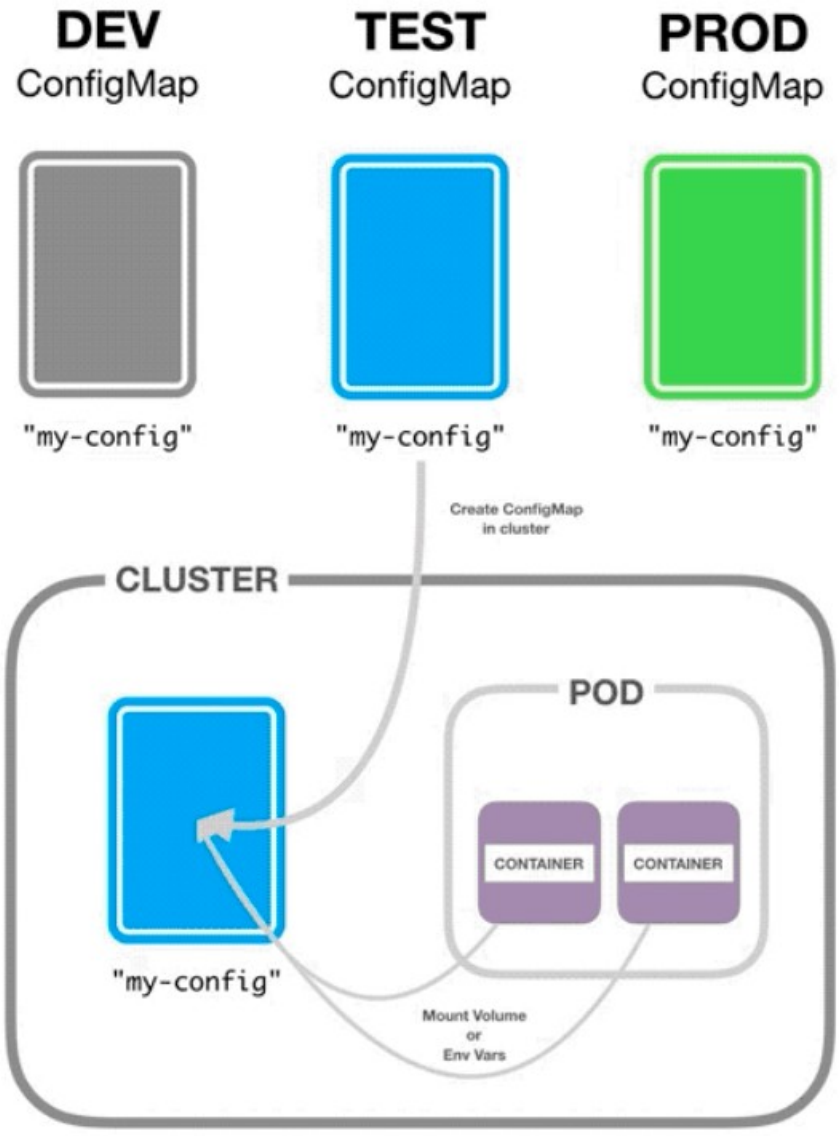
ConfigMaps and Secrets

What are ConfigMaps?

A **ConfigMap** is an API object used to store non-confidential data in key-value pairs. How do you manage your application's configuration? For a Python or Node.js application, where do you store the configuration? ConfigMaps allow us to make a single change to the file itself, which can then effect many different pods using the same ConfigMap, thus cutting down on user error and deployment times.



Kubernetes pods can use ConfigMaps as configuration files, environment variables or command-line arguments. ConfigMaps allow you to decouple environment-specific configurations from containers to make applications portable. Notice in the picture below we're creating a ConfigMap for a video game. The player lives, types of

enemies, and even colors are defined and would be applied to the application running on any pods using this particular ConfigMap.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

What are Secrets?

Secrets are a Kubernetes object intended for storing a small amount of sensitive data. To create a Secret ad-hoc choose a password and convert it to base64.

This is an imperative method to create a Secret, followed be the declarative YAML file. The command `echo -n 'KubernetesRocks!' | base64` is taking the string and converting it into a hashed state.

```
$ echo -n 'KubernetesRocks!' | base64
S3ViZXJuZXRlc1JvY2tzIQ==
```

Secrets let us store and manage information, such as API keys, SSH keys, OAuth token, and more.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mariadb-root-password
type: Opaque
data:
  password: S3ViZXJuZXRlc1JvY2tzIQ==
```

Storing sensitive data in Secrets is more secure than in plain text ConfigMaps or in Pod specifications. Using Secrets gives you control over how sensitive data is used, and reduces the risk of exposing the data to unauthorized users.

You can also encrypt Secrets at the application layer using a key you manage in Cloud KMS.

Hands On – Create a ConfigMap and Secret

I. Create a ConfigMap and a pod that will utilize it:

Let's start creating a ConfigMap, applying it to the cluster, then verifying that its running. Notice that this ConfigMap is applying certain values to video game within our redis DB. Look in the data field and remember that *player.maximum-lives* is currently set to 5.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: yourname-demo
data:
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-configmap.yaml
configmap/yourname-demo unchanged
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get configmap yourname-demo
NAME            DATA   AGE
yourname-demo   2      5m50s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

The *data* field would change characteristics of the app itself. Once we create our pod, we will then make a change to the ConfigMap to ensure that everything is working as expected.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-app.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: mypod
        image: redis
        resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
        volumeMounts:
        - name: yournamefoo
          mountPath: "/etc/yournamefoo"
          readOnly: true
      volumes:
      - name: yournamefoo
        configMap:
          name: yourname-demo
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-app.yaml
deployment.apps/myapp created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
myapp-f76dcbd4-m85lv     1/1     Running   0          4s
```

II. SSH to your Pod and verify the ConfigMap:

Use the kubectl exec -it <yourpod> – */bin*/bash command to SSH into your new pod. Once inside, navigate to the *etc/<yournamefoo>* folder. Inside, you will find fame files based on the data from our ConfigMap. Notice that max player lives is 5. Once you've verified this exit back to the cloudshell.

```
root@myapp-f76dcbd4-m85lv:/data# pwd
/data
root@myapp-f76dcbd4-m85lv:/data# cd ..
root@myapp-f76dcbd4-m85lv:/# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@myapp-f76dcbd4-m85lv:/# cd etc/yournamefoo/
root@myapp-f76dcbd4-m85lv:/etc/yournamefoo# ls
game.properties  user-interface.properties
root@myapp-f76dcbd4-m85lv:/etc/yournamefoo# cat game.properties
enemy.types=aliens,monsters
player.maximum-lives=5
root@myapp-f76dcbd4-m85lv:/etc/yournamefoo#
```

III. Change the ConfigMap:

Edit the *yourname-configmap.yaml* file to reflect only 2 lives instead of 5.

```
  GNU nano 5.4
apiVersion: v1
kind: ConfigMap
metadata:
  name: yourname-demo
data:
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=2
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

Verify your changes and then apply the new changes. Next, delete you pod and wait for it to redeploy so that the changes take effect.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: yourname-demo
data:
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=2
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-configmap.yaml
configmap/yourname-demo configured
```

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl delete pods --all
pod "myapp-f76dcbd4-m85lv" deleted
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
myapp-f76dcbd4-vcchn    1/1     Running   0          8s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

IV. Check your Pod to verify:

Using the same method as step 2, SSH into your new pod. Check the same directory and ensure the changes have taken place. You should now have 2 max lives instead of 5.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl exec -it myapp-f76dcbd4-vcchn -- /bin/bash
root@myapp-f76dcbd4-vcchn:/data# cd ..
root@myapp-f76dcbd4-vcchn:/# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@myapp-f76dcbd4-vcchn:/# cd etc/yournamefoo/
root@myapp-f76dcbd4-vcchn:/etc/yournamefoo# cat game.properties
enemy.types=aliens,monsters
player.maximum-lives=2
root@myapp-f76dcbd4-vcchn:/etc/yournamefoo#
```

VI. Create a secret

First, lets create a base64 secret and copy it for the YAML file. You need to copy down your hash so that we can paste it into your secrets.yaml file.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ echo -n 'CoditIT' | base64
Q29kaXRJVA==
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

Create a *secrets.yaml* file and ensure that the *password.file:* field contains your new secret. Finally, apply your secret to the cluster.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: temp-secret
type: Opaque
data:
  password.file: |
    Q29kaXRJVA==
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f secrets.yaml
secret/temp-secret created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

Your secrets deployment should be named *temp-secret*. Now, create your Pod with the *temp-secret* name referenced in the *volumes* field.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-pod.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myappcontainer
        image: nginx:latest
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 1883
        volumeMounts:
          - name: myapp-secret
            mountPath: "etc/foo"
            readOnly: true
      volumes:
        - name: myapp-secret
          secret:
            secretName: temp-secret
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-pod.yaml
```

After you apply this deployment ssh into your pod with the kubectl exec -it <yourpod> – */bin/bash* command. Find the etc/foo/ directory and cat the password.file. You should notice 'CoditIT' printed on the screen. The output may come out strangely in cloudshell. So long as it shows, you have completed this training.

```
root@myapp-5d6645954d-b572q:~# cd ..
root@myapp-5d6645954d-b572q:/# ls
bin  boot  dev  docker-entrypoint.d  docker-entrypoint.sh  etc
root@myapp-5d6645954d-b572q:/# cat /etc/foo/password.file
CoditITroot@myapp-5d6645954d-b572q:/#
```