

# **Kubernetes Lesson 6**

## Configuring Applications & Deployments

### Video 1. “ConfigMap”

### ConfigMap

- ConfigMaps allow you to separate your application configurations from your application code, which helps keep your containerized applications portable
- ConfigMaps are useful for storing and sharing non-sensitive, unencrypted configuration information which you can change at runtime
- It help makes their configurations easier to change and manage, and prevents hardcoding configuration data to Pod specifications

### Video 2. “ConfigMap from Literals”

- ***kubectl create configmap myfirstcm --from-literal=fname=Rizwan --from-literal=lname=Sheikh***

*Through this command we create a resource from-literal tag with the string values*

### Video 3. “ConfigMap from File”

- ***kubectl create cm myfirstcmwithfile --from-file=user.txt***

*This command is used to create the CM from file. But in in this case we not mentioned the Key for our values indeed these values are in the file and by default the CM choose the key name that we used as a file name.*

***But if we want to give the a specific name for so this below command is used.***

- ***kubectl create cm myfirstcmwithfile2 --from-file=Bio=user.txt***

#### Video 4. “ConfigMap from env File”

*If we want to make sure that we have not to give the key for values as externally so we used cm from env file except cm from file. So firstly create a file with the extension of .env because it is best practice for creating the cm from env file **Other extensions** are valid but best practice is to use .env **e.g. cmfromenvfile.env***

- **kubectl create cm cmfromenvfile --from-env-file=cm.env**

*This is the command that which is used for creating the configmap resource from env file.*

#### Video 5. “ConfigMap as Volume”

- **kubectl exec pod podwithcmvol -it sh**

*This command is deprecated and now the other new one updated command is useful for go into the container’s filesystem. You can see from picture below.*

```
allii@kubernetes:~$ kubectl exec pod podwithcmvol -it sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
Error from server (NotFound): pods "pod" not found
allii@kubernetes:~$ kubectl exec --stdin --tty shell-demo -- /bin/bash
```

**Note:** Now this command is used to go into the container’s inside filesystem.

Get a shell to the running container:

```
kubectl exec --stdin --tty shell-demo -- /bin/bash
```

**Note:** The double dash (--) separates the arguments you want to pass to the command from the kubectl arguments.

- ***kubect exec --stdin --tty podwithcmvol -c container1 -- /bin/bash***

```
allii@kubernetes: ~  
allii@kubernetes:~$ kubect exec --stdin --tty podwithcmvol -c container1 -- /bin/bash  
root@podwithcmvol:/# ls  
bin  data  docker-entrypoint.d  etc  lib  media  opt  root  sbin  sys  usr  
boot dev  docker-entrypoint.sh  home lib64 mnt  proc  run  srv  tmp  var  
root@podwithcmvol:/# cd data/  
root@podwithcmvol:/data# ls  
cm  
root@podwithcmvol:/data# cd cm  
root@podwithcmvol:/data/cm# ls  
fname  lname  
root@podwithcmvol:/data/cm# cat fname  
Rizwanroot@podwithcmvol:/data/cm# cat lname  
Sheikhroot@podwithcmvol:/data/cm#
```

***So it is proved*** that files that we stored in the configmap through any method like from literal, from file or from env file, is now accessible by container that which is in the pod through **configMap Volume resource**.

***So now we*** can make access through our programming language that will be use for out application and can be use for the configuration settings.

***This one is the method one*** that how to reach our files under the access of the our container the second method will be discuss in the next video.

## **Video 6. “ConfigMap as env Variables”**


***There are two*** method to reaches the configurationmap to the container’s filesystem firstly using the environmental variables and second is the volumes.

```
allii@kubernetes:~$ cat podcmenv.yaml  
apiVersion: v1  
kind: Pod  
metadata:  
  name: podcmenv  
  labels:  
    app: envexample  
spec:  
  containers:  
  - name: container1  
    image: aamirpinger/node-app-image  
    envFrom:  
    - configMapRef:  
      name: cmfromenvfile
```

```
alli@kubernetes:~$ cat podwithcm.yaml
apiVersion: v1
kind: Pod
metadata:
  name: podwithcmvol
spec:
  volumes:
  - name: cmvol
    configMap:
      name: myfirstcm
  containers:
  - name: container1
    image: nginx
    volumeMounts:
    - name: cmvol
      mountPath: /data/cm
    imagePullPolicy: IfNotPresent
```

## Video 7. “Secrets”

### Secret



- Secrets are also used to pass key/value data to application dynamically like configMaps
- Secrets are secure objects which stores sensitive data, such as passwords, OAuth tokens, and SSH keys, in your clusters
- Storing sensitive data in Secrets is more secure than 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
- The Secret values are Base64 encoded in Kubernetes

Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)

**Secrets are used to** send the most sensitive data (passwords, SSH Keys etc.) in encrypted form to the container. This information does not send in the string form.

*It works like the **configmap** but little bit different form that the **configmap** prints the data over the terminal and every user can easily see the data but in the secrets whole data send in the encrypted form and only authorized users can use that data.*

### Video 8. “Secret from Literals”

## DNS Subdomain Names

Most resource types require a name that can be used as a DNS subdomain name as defined in [RFC 1123](#). This means the name must:

- contain no more than 253 characters
- contain only lowercase alphanumeric characters, '-' or '.'
- start with an alphanumeric character
- end with an alphanumeric character

## Creating your own Secrets

### Creating a Secret Using `kubectl`

Secrets can contain user credentials required by Pods to access a database. For example, a database connection string consists of a username and password. You can store the username in a file `./username.txt` and the password in a file `./password.txt` on your local machine.

```
# Create files needed for the rest of the example.  
echo -n 'admin' > ./username.txt  
echo -n '1f2d1e2e67df' > ./password.txt
```

The `kubectl create secret` command packages these files into a Secret and creates the object on the API server. The name of a Secret object must be a valid [DNS subdomain name](#).

- ***kubectl create secret generic myfirstsecret --from-literal=username=Rizwansheikh --from-literal=api=test\_db-aws.com***

*This is the command that used to send data to the pod in encrypted form through from literal.*

- ***echo dGVzdF9kYi1hd3MuY29t | base64 -d***

*This is the command that is used to dencrypt the data.*

### **Video 9. “Secret from File”**

- ***kubectl create secret generic secretfromfile --from-file=Bio=keyval.txt***

*This command used to create secret form file*

### **Video 10. “Secret from env File”**

- ***kubectl create secret generic secretfromenvfile --from-env-file=keyval.txt***

*This command used to create secret form env file*

### **Video 11. “Secret as Volume”**

***apiVersion: v1***

***kind: Pod***

***metadata:***

***name: podwithsecvol***

***spec:***

***volumes:***

***- name: secvol***

***secret:***

***secretName: myfirstsecret***

***containers:***

***- name: container1***

***image: nginx***

***imagePullPolicy: IfNotPresent***

***volumeMounts:***

***- name: secvol***

***mountPath: /usr/share/nginx/html***

*Through this configuration we make secret volume type and send the sensitive data to the pod and then the container use the data.*

#### **Video 12. "Secret as env Variable"**

```
apiVersion: v1
kind: Pod
metadata:
  name: podwithenv
spec:
  containers:
  - name: container1
    image: nginx
    imagePullPolicy: IfNotPresent
    env:
    - name: USERNAME
      value: alliiriizzvvii
    - name: COURSENAME
      value: kubernetes
```

#### **Video 13. "Environment Variable"**

```
apiVersion: v1
kind: Pod
metadata:
  name: podwithenv
spec:
  containers:
  - name: container1
    image: nginx
    imagePullPolicy: IfNotPresent
    env:
    - name: USERNAME
      value: alliiriizzvvii
    - name: COURSENAME
      value: kubernetes
```

#### ● **kubectl exec podwithenv env**

*This command is used to check the env from the container's filesystem. We can use the ls with this command for checking the list of the files and folders inside the container e.g. **kubectl exec podwithenv ls***



## Video 14. “Deployment”

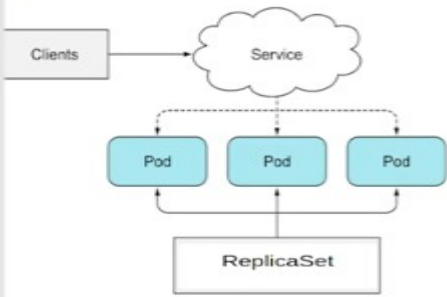
### Deployment

- Till the moment we have learned
  - How to group our containerized app into pods
  - Ways to provide them with temporary or permanent storage
  - How to pass both secret and non-secret config data to them
  - Allowed pods to find and talk to each other
  - How to run a full-fledged system composed of independently running smaller components (microservices)

Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)

## Video 15. “Deployment Use Case”

### Deployment



The diagram illustrates a deployment use case. At the top, a box labeled 'Clients' has an arrow pointing to a cloud labeled 'Service'. Below the 'Service' cloud, three light blue boxes labeled 'Pod' are arranged horizontally. A dashed line connects the 'Service' cloud to the first 'Pod'. Below the 'Pods', a box labeled 'ReplicaSet' has arrows pointing up to each of the three 'Pods'. Dashed lines also connect the 'Pods' to each other, indicating communication within the pod set.

- Let's say you created your pod with a container having any image e.g. aamirpinger/helloworld:v1
- After sometime you want you running pods to update the image with aamirpinger/helloworld:v2
- Because you can't change an existing pod's image after the pod is created, you need to remove the old pods and replace them with new ones running the new image

Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)

### Theory Based Video.

## Video 16. “Deployment Strategies Pros and Cons”

This Video is theory Based.



## Video 17. “What Strategy to Choose-”

*This Video is theory Based.*

## Video 18. “Deployment Example Part 1”

apiVersion: apps/v1

kind: Deployment

metadata:

name: deployexample

spec:

replicas: 4

selector:

matchLabels:

app: frontend

template:

metadata:

labels:

app: frontend

spec:

containers:

- name: container1

image: aamirpinger/hi

ports:

- containerPort: 80

imagePullPolicy: IfNotPresent

## Updating Images

An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.

The default pull policy is `IfNotPresent` which causes the `kubelet` to skip pulling an image if it already exists. If you would like to always force a pull, you can do one of the following:

- set the `imagePullPolicy` of the container to `Always`.
- omit the `imagePullPolicy` and use `:latest` as the tag for the image to use.
- omit the `imagePullPolicy` and the tag for the image to use.
- enable the `AlwaysPullImages` admission controller.

When `imagePullPolicy` is defined without a specific value, it is also set to `Always`.

### **Video 19. “Deployment Example Part 2”**

- ✓ In this video we will discuss that how to **change/update** our image and that updated image automatically reach under the access of the users.

For doing this we will change the image in the deployment template and then the deployment resource will automatically change the image using one strategy out of three.

We can change only the tag of the image for example initially V1 was as a tag of the image and after that we can replace with V2 tag

**OR**

We can change the image totally.

- **kubectrl set image deploy deployexample container1=aamirpinger/flag**

By using this command we will assign the new image and then the kubernetes will change the Image with the help of the deployment resource.

**NOTE:** The **kubernetes** uses Rolling Update as the default strategy. Firstly deletes the one pod and then the replicaSet makes the new pod for the replacement of the missing one pod.

So when the user accesses the app through provided external IP the replicaset diverts the some of the user on the old pods and some of them on the new pods and the replicaset do this till the all the old pods delete and the new pod with updated version are not created.

### **Video 20. “Deployment Example Part 3”**

**Theory Based..**

### **Video 21. “Deployment Helper Commands”**

- **kubectrl rollout pause deployment deployexample**

**This command** will **pause** the process of the deployment.

- **kubectrl rollout resume deployment deployexample**

**This command** will **resume** the process of the deployment.

- **kubectrl rollout status deployment deployexample**

**This command** will show the **status** of process of the deployment.

- **kubectrl rollout history deployment deployexample**

*This command will show deployment history*

- **kubectrl rollout history deployment deployexample --revision=1**

*By using this command we will see the the revisions history and make the difference from each other.*


- **kubectrl rollout undo deployment deployexample --to-revision=1**

***Note:** If we will not mention the number of the revisions so by default the kubernetes will choose last time used revision **But we** can choose by our own wish for example there are 7 time revisions have made so we can choose 3 to roll back.*

## **Video 22. “Kubernetes Best Practices”**

***Always avoid form latest tag** because if we use the latest tag it's not meaning that the docker will pull the targeted image form the registry. The docker understands latest means the most recent image if the tag of the image is v4, latest means last image you pushed the tag is v4. It is possible tht there are available more one images with latest tag and over write too. Always choose v1,v2,v3 and so on for giving the tag to image simultaneously.*

### Kubernetes Best Practices



---

- Try avoid using latest tag instead us proper tags
- imagePullPolicy should be used wisely
- If the imagePullPolicy is IfNotPresent and you push updated image with the same previous tag, container will not updated as it will find image already present so won't pull again
- If the imagePullPolicy is Always it will pull image everytime pod instance will created, this will slow down the initialization phase of container

*Be careful when choosing the **imagePullPolicy**.*

### **1. ImagePullPolicy: Always**

*This means that the kubernetes will always download the new image when creating new instance of the application for example if we created the **replicaset count: 3** so the kubernetes will download the image three times for making three pods with same image. **But this policy** has a drawback that the image will always be download newly and out resources will consume many*

## 2. ImagePullPolicy: IfNotPresent

If the image is available on our system so the kubernetes will not download the image again. **This Policy has one drawback** means if you pushed 2<sup>nd</sup> image with same tag so the kubernetes will not download the most recent image from the repository because the kubernetes will understand that you said that the same tag that is already downloaded. **But this policy** has a drawback if the image has latest tag that which is downloaded already and you have uploaded three more images with latest tag so the kubernetes will not download the new image because it sense that the image is available with this tag already, it has no concern that the code in existence image is old

**Kubernetes Depends All About On Tags So Choose Wisely During Building An Image And When Describing The ImagePullPolicy.**

## 3. ImagePullPolicy: Never

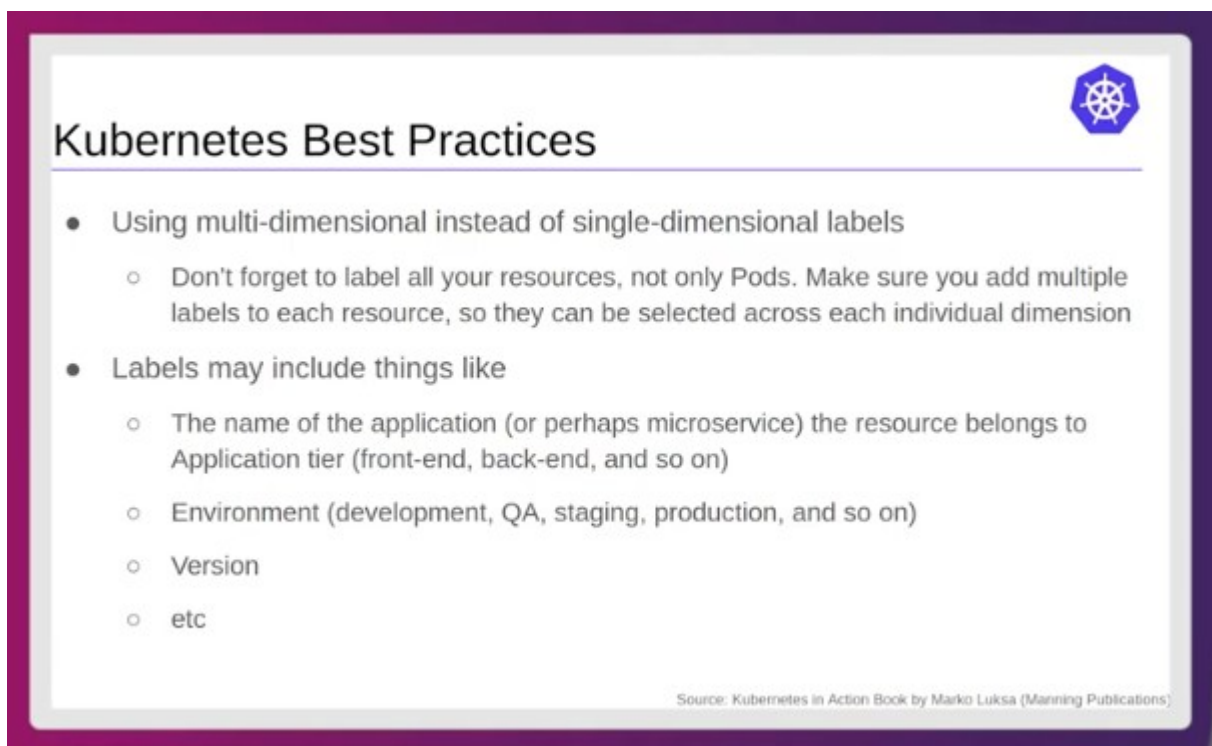
**This means that if** the image is available on our system means server so the kubernetes will use that image but if the image is not available so the throw the error that I can not make container due to none availability of image on the system..

**In This Policy The Kubernetes** will never download the image form any image registry like docker hub or rkt.

**But this policy** has a drawback if the image will not on our system the will not be download and the container will not run.

## Labels

*Always use labels when creating any resource.*

A presentation slide titled "Kubernetes Best Practices" with the Kubernetes logo in the top right corner. The slide lists two main bullet points: "Using multi-dimensional instead of single-dimensional labels" and "Labels may include things like". The first bullet point has a sub-bullet: "Don't forget to label all your resources, not only Pods. Make sure you add multiple labels to each resource, so they can be selected across each individual dimension". The second bullet point has sub-bullets: "The name of the application (or perhaps microservice) the resource belongs to", "Application tier (front-end, back-end, and so on)", "Environment (development, QA, staging, production, and so on)", "Version", and "etc". At the bottom right, there is a small text source: "Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)".

**Kubernetes Best Practices**

- Using multi-dimensional instead of single-dimensional labels
  - Don't forget to label all your resources, not only Pods. Make sure you add multiple labels to each resource, so they can be selected across each individual dimension
- Labels may include things like
  - The name of the application (or perhaps microservice) the resource belongs to
  - Application tier (front-end, back-end, and so on)
  - Environment (development, QA, staging, production, and so on)
  - Version
  - etc

Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)

## **Manageable Size Of Image**

*Your image's size should always be small means microservices based and we can easily build the communication between them through cluster ip..*

*There are some of benefits of small image.*

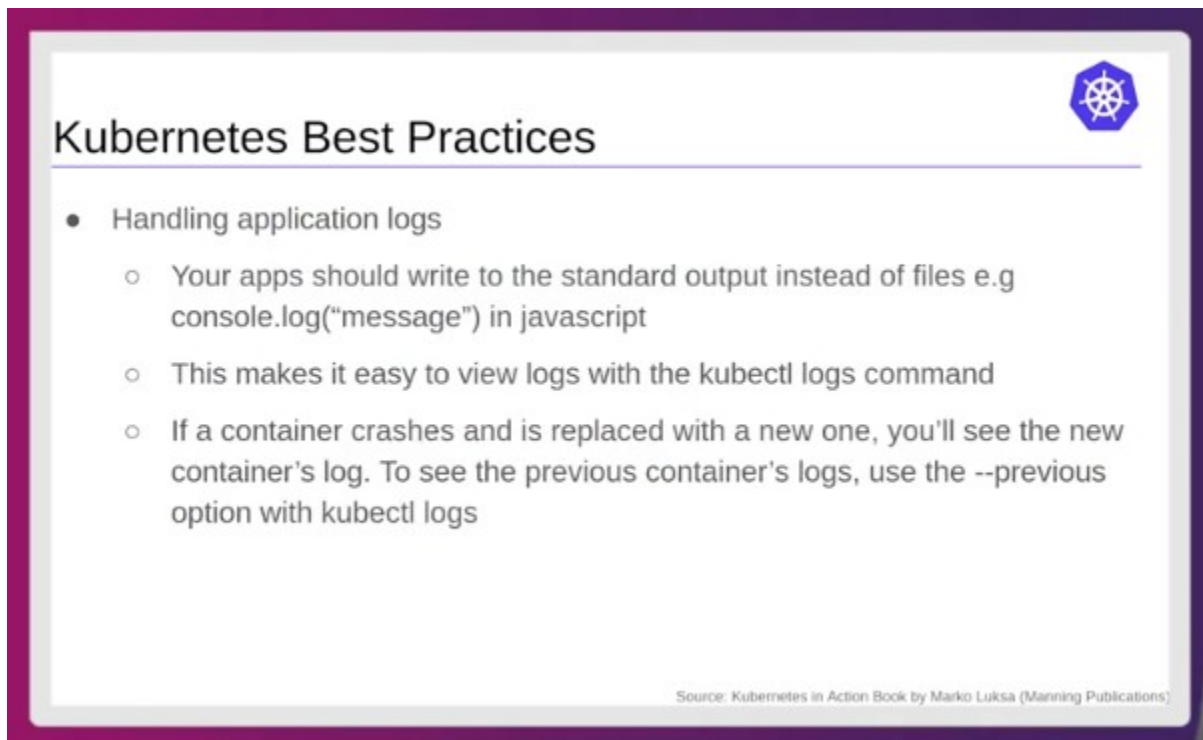
*For example if the container restart and this means the container will be create new so if the imagePullPolicy is Always so the image will be download hurry and load too.*

*And if the kubernetes shift our pod one worker node to another worker OR our replica count increases or decreases so the image Is small in size then image will be download and load hurry.*

## **Annotations**

*So Always use annotations in your app because these are helpful for fixing different problems and contact details of the maintainer/creator.*

## **Handling Application Logs**

A presentation slide titled "Kubernetes Best Practices" with a purple border and a Kubernetes logo in the top right corner. The slide lists three bullet points under the heading "Handling application logs". The first bullet point is "Your apps should write to the standard output instead of files e.g console.log('message') in javascript". The second bullet point is "This makes it easy to view logs with the kubectl logs command". The third bullet point is "If a container crashes and is replaced with a new one, you'll see the new container's log. To see the previous container's logs, use the --previous option with kubectl logs". At the bottom right, there is a small text source: "Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)".

**Kubernetes Best Practices**

- Handling application logs
  - Your apps should write to the standard output instead of files e.g `console.log("message")` in javascript
  - This makes it easy to view logs with the `kubectl logs` command
  - If a container crashes and is replaced with a new one, you'll see the new container's log. To see the previous container's logs, use the `--previous` option with `kubectl logs`

Source: Kubernetes in Action Book by Marko Luksa (Manning Publications)

**Here Kubernetes Course Completed.**

# ***Alhamdolillah!***

## ***Kubernetes Lesson6***

### ***Completed.***

*This Document Is Created By “Rizwan Sheikh”*

*Email: rizwansheikh7071@gmail.com*

*Docker ID: rizwansheikh7071*

*Due To Human Being The Mistake Can Be Done!*

*Thank You Happy Learning.*