

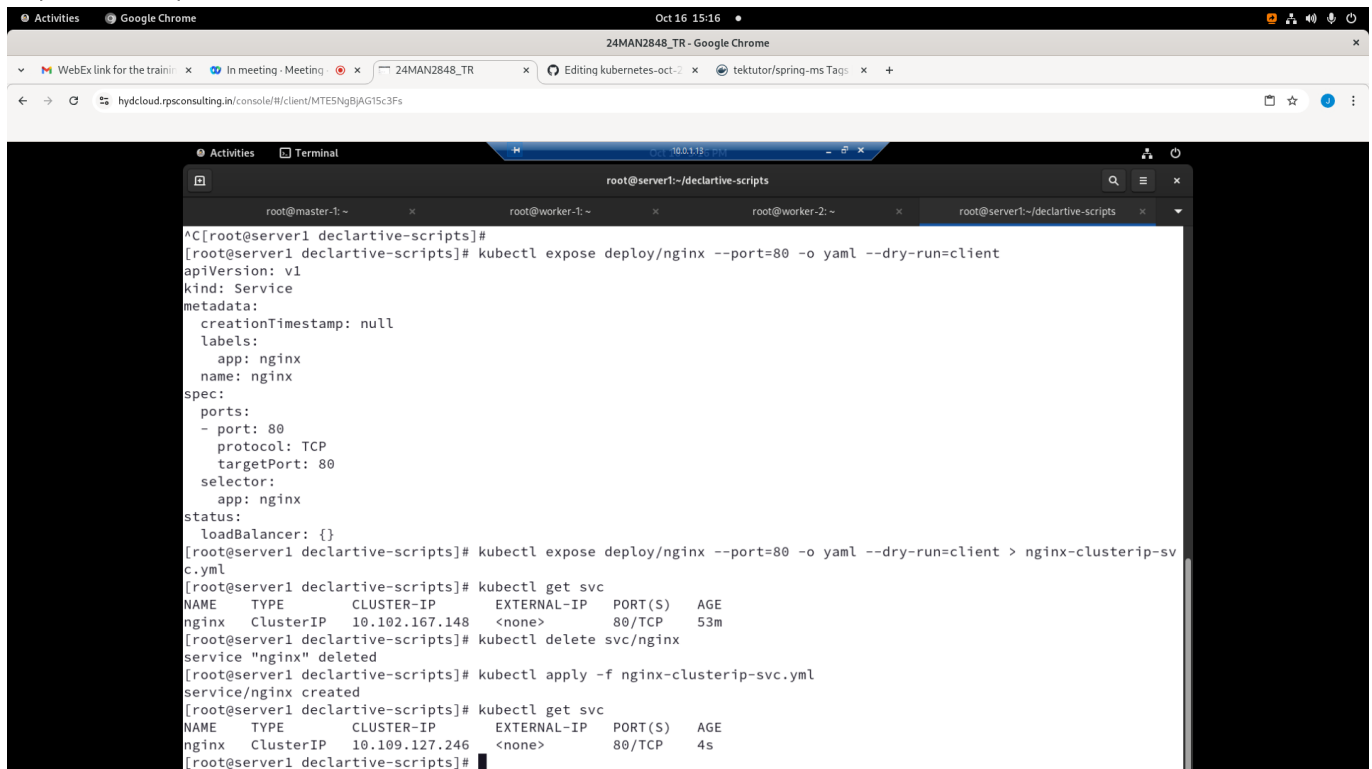
Day 3

Lab - Declaratively creating a clusterip internal service

```
kubectl expose deploy/nginx --type=ClusterIP --port=80 -o yaml --dry-run=client
kubectl expose deploy/nginx --type=ClusterIP --port=80 -o yaml --dry-run=client > nginx-clusterip-svc.yaml
kubectl apply -f nginx-clusterip-svc.yaml

kubectl get svc
kubectl describe svc/nginx
```

Expected output



```
^C[root@server1 declarative-scripts]#
[root@server1 declarative-scripts]# kubectl expose deploy/nginx --port=80 -o yaml --dry-run=client
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: nginx
    name: nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
status:
  loadBalancer: {}
[root@server1 declarative-scripts]# kubectl expose deploy/nginx --port=80 -o yaml --dry-run=client > nginx-clusterip-svc.yaml
[root@server1 declarative-scripts]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx     ClusterIP  10.102.167.148   <none>            80/TCP        53m
[root@server1 declarative-scripts]# kubectl delete svc/nginx
service "nginx" deleted
[root@server1 declarative-scripts]# kubectl apply -f nginx-clusterip-svc.yaml
service/nginx created
[root@server1 declarative-scripts]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx     ClusterIP  10.109.127.246   <none>            80/TCP        4s
[root@server1 declarative-scripts]#
```

Lab - Declaratively creating nodeport external service

Let's delete the clusterip service in declarative style

```
kubectl delete -f nginx-clusterip-svc.yaml
```

Let's create the nodeport external service in declarative style

```
kubectl expose deploy/nginx --type=NodePort --port=80 -o yaml --dry-run=client
```

```
kubectl expose deploy/nginx --type=NodePort --port=80 -o yaml --dry-run=client > nginx-nodeport-svc.yaml
kubectl apply -f nginx-nodeport-svc.yaml

kubectl get svc
kubectl describe svc/nginx
```

Expected output

```
root@server1:~/declarative-scripts
root@master-1: ~
root@worker-1: ~
root@worker-2: ~
root@server1:~/declarative-scripts

app: nginx
status:
  loadBalancer: {}
[root@server1 declarative-scripts]# kubectl expose deploy/nginx --port=80 -o yaml --dry-run=client > nginx-clusterip-svc.yaml
[root@server1 declarative-scripts]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx     ClusterIP  10.102.167.148   <none>           80/TCP       53m
[root@server1 declarative-scripts]# kubectl delete svc/nginx
service "nginx" deleted
[root@server1 declarative-scripts]# kubectl apply -f nginx-clusterip-svc.yaml
service/nginx created
[root@server1 declarative-scripts]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx     ClusterIP  10.109.127.246   <none>           80/TCP       4s
[root@server1 declarative-scripts]#
[root@server1 declarative-scripts]# kubectl delete -f nginx-clusterip-svc.yaml
service "nginx" deleted
[root@server1 declarative-scripts]# kubectl expose deploy/nginx --type=NodePort --port=80 -o yaml --dry-run=client > nginx-nodeport-svc.yaml
[root@server1 declarative-scripts]# ls -l
total 12
-rw-r--r--. 1 root root 223 Oct 16 15:15 nginx-clusterip-svc.yaml
-rw-r--r--. 1 root root 391 Oct 16 15:10 nginx-deploy.yaml
-rw-r--r--. 1 root root 240 Oct 16 15:17 nginx-nodeport-svc.yaml
[root@server1 declarative-scripts]# kubectl apply -f nginx-nodeport-svc.yaml
service/nginx created
[root@server1 declarative-scripts]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx     NodePort   10.101.215.83    <none>           80:30314/TCP 4s
[root@server1 declarative-scripts]#
```

Lab - Creating a Pod in declarative style

Create a file with below content and save the file as pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: frontend
spec:
  containers:
  - name: my-container
    image: tektutor/spring-ms:1.0
```

Let's create the pod

```
kubectl create -f pod.yaml --save-config
kubectl get po
```

Let's modify the pod.yml as shown below

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: web
    tier: frontend
spec:
  containers:
  - name: my-container
    image: tektutor/spring-ms:1.0
```

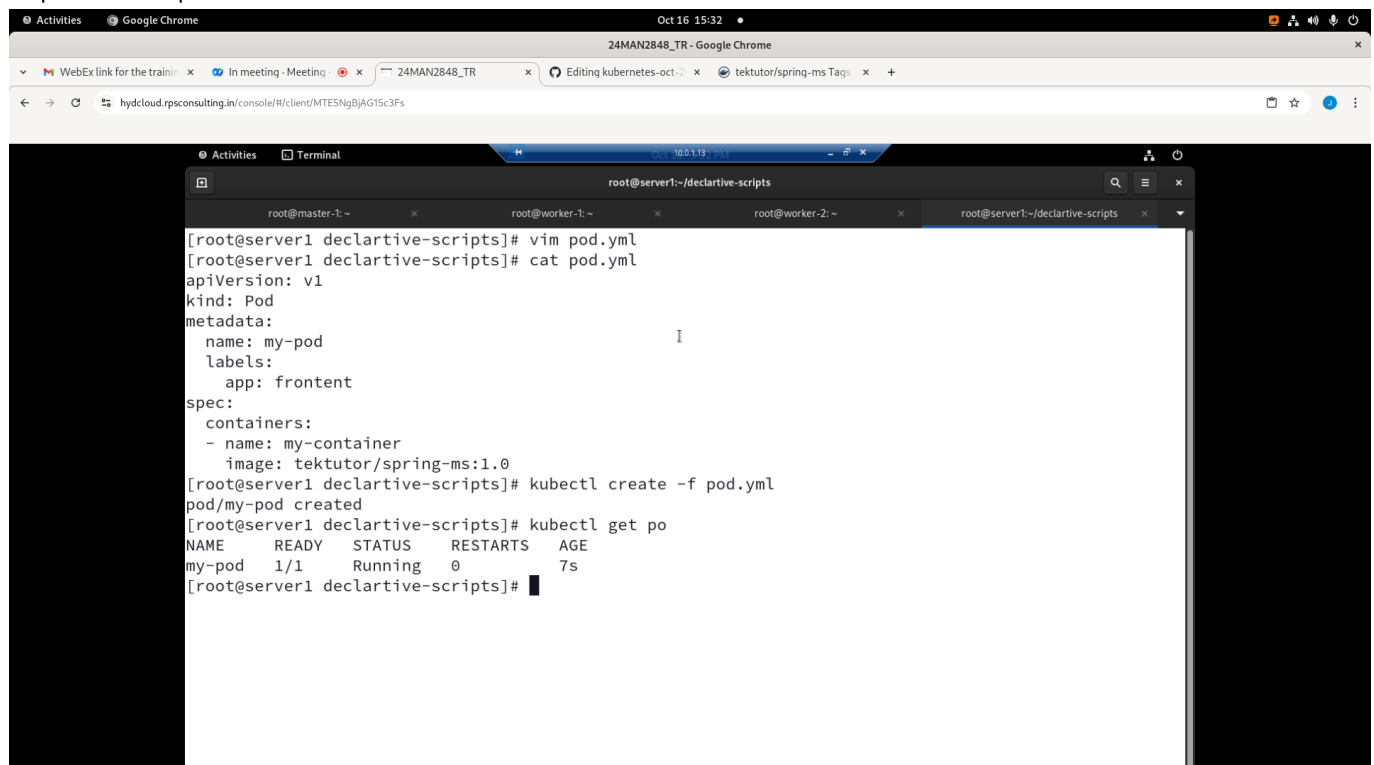
Let's apply the delta changes on the already existing kubernetes resource

```
kubectl apply -f pod.yml
kubectl get po
```

Once you are done with this exercise, you may delete it as shown below

```
kubectl delete -f pod.yml
```

Expected output

A screenshot of a terminal window with a dark background. The terminal shows the following commands and output:

```
[root@server1 declarative-scripts]# vim pod.yml
[root@server1 declarative-scripts]# cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: frontend
spec:
  containers:
  - name: my-container
    image: tektutor/spring-ms:1.0
[root@server1 declarative-scripts]# kubectl create -f pod.yml
pod/my-pod created
[root@server1 declarative-scripts]# kubectl get po
NAME      READY   STATUS    RESTARTS   AGE
my-pod    1/1     Running   0           7s
[root@server1 declarative-scripts]#
```

The terminal window has several tabs open at the top, including 'root@master-1: ~', 'root@worker-1: ~', 'root@worker-2: ~', and 'root@server1: ~/declarative-scripts'. The active tab is 'root@server1: ~/declarative-scripts'.

```

[root@server1 declarative-scripts]# kubectl get po
NAME      READY   STATUS    RESTARTS   AGE
my-pod    1/1     Running   0           7s
[root@server1 declarative-scripts]# vim pod.yml
[root@server1 declarative-scripts]# cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: web
    tier: frontend
spec:
  containers:
  - name: my-container
    image: tektutor/spring-ms:1.0
[root@server1 declarative-scripts]# kubectl apply -f pod.yml
Warning: resource pods/my-pod is missing the kubectrl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
pod/my-pod configured
[root@server1 declarative-scripts]# kubectl get po
NAME      READY   STATUS    RESTARTS   AGE
my-pod    1/1     Running   0          5m15s
[root@server1 declarative-scripts]#

```

Lab - Deploying your custom application into Kubernetes cluster

We need to clone the source code first

```

cd ~
git clone https://github.com/tektutor/spring-ms.git
rm *.yml *.yaml

```

Let's build the custom docker image

```

docker build -t tektutor/hello-spring-microservice:1.0 .
docker images
docker login
docker push tektutor/hello-spring-microservice:1.0

```

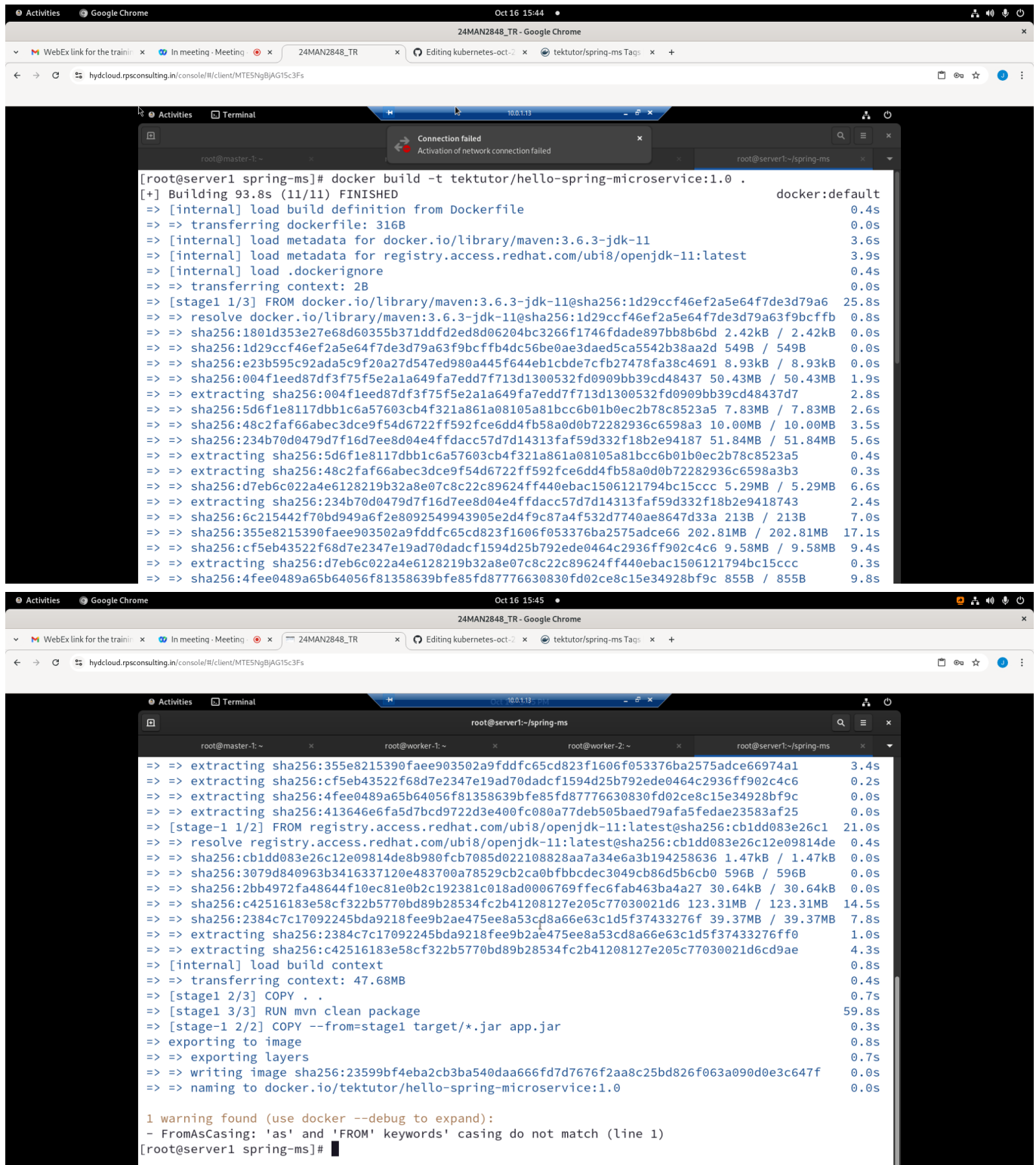
Let's deploy our custom application into K8s cluster

```

kubectl create deployment hello --image=tektutor/hello-spring-microservice:1.0 ---replicas=2
kubectl get deploy,rs,po

```

Expected output



```
[root@server1 spring-ms]# docker build -t tektutor/hello-spring-microservice:1.0 .
[+] Building 93.8s (11/11) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default
=> => transferring dockerfile: 316B                                              0.4s
=> [internal] load metadata for docker.io/library/maven:3.6.3-jdk-11            0.0s
=> [internal] load metadata for registry.access.redhat.com/ubi8/openjdk-11:latest 3.6s
=> [internal] load .dockerignore                                                 3.9s
=> => transferring context: 2B                                                  0.4s
=> [stage1 1/3] FROM docker.io/library/maven:3.6.3-jdk-11@sha256:1d29ccf46ef2a5e64f7de3d79a6 0.0s
=> => resolve docker.io/library/maven:3.6.3-jdk-11@sha256:1d29ccf46ef2a5e64f7de3d79a63f9bcffb 0.8s
=> => sha256:1801d353e27e68d60355b371ddfd2ed8d06204bc3266f1746fdade897bb8b6bd 2.42kB / 2.42kB 0.0s
=> => sha256:1d29ccf46ef2a5e64f7de3d79a63f9bcffb4dc56be0ae3daed5ca5542b38aa2d 549B / 549B 0.0s
=> => sha256:e23b595c92ada5c9f20a27d547ed980a445f644eb1cbde7cfb27478fa38c4691 8.93kB / 8.93kB 0.0s
=> => sha256:004f1eed87df3f75f5e2a1a649fa7edd7f713d1300532fd0909bb39cd48437 50.43MB / 50.43MB 1.9s
=> => extracting sha256:004f1eed87df3f75f5e2a1a649fa7edd7f713d1300532fd0909bb39cd48437d7 2.8s
=> => sha256:5d6f1e8117dbb1c6a57603cb4f321a861a08105a81bcc6b01b0ec2b78c8523a5 7.83MB / 7.83MB 2.6s
=> => sha256:48c2faf66abec3dce9f54d6722ff592fce6dd4fb58a0d0b72282936c6598a3 10.00MB / 10.00MB 3.5s
=> => sha256:234b70d0479d7f16d7ee8d04e4ffdac57d7d14313faf59d332f18b2e94187 51.84MB / 51.84MB 5.6s
=> => extracting sha256:5d6f1e8117dbb1c6a57603cb4f321a861a08105a81bcc6b01b0ec2b78c8523a5 0.4s
=> => extracting sha256:48c2faf66abec3dce9f54d6722ff592fce6dd4fb58a0d0b72282936c6598a3b3 0.3s
=> => sha256:d7eb6c022a4e6128219b32a8e07c8c22c89624ff440ebac1506121794bc15ccc 5.29MB / 5.29MB 6.6s
=> => extracting sha256:234b70d0479d7f16d7ee8d04e4ffdac57d7d14313faf59d332f18b2e9418743 2.4s
=> => sha256:6c215442f70bd949a6f2e8092549943905e2d4f9c87a4f532d7740ae8647d33a 213B / 213B 7.0s
=> => sha256:355e8215390faee903502a9fddfc65cd823f1606f053376ba2575adce66 202.81MB / 202.81MB 17.1s
=> => sha256:c5f5eb43522f68d7e2347e19ad70dadcf1594d25b792ede0464c2936ff902c4c6 9.58MB / 9.58MB 9.4s
=> => extracting sha256:d7eb6c022a4e6128219b32a8e07c8c22c89624ff440ebac1506121794bc15ccc 0.3s
=> => sha256:4fee0489a65b64056f81358639bf85fd87776630830fd02ce8c15e34928bf9c 855B / 855B 9.8s

1 warning found (use docker --debug to expand):
- FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)
[root@server1 spring-ms]#
```

```

Oct 16 15:51
root@tektutor:/home/jegan/spring-ms

jegan@tektutor:~/kubernetes-oct-2024/Day3/declarative-manifest-scripts
root@tektutor:/home/jegan/spring-ms

ide-work-tree

1 warning found (use docker --debug to expand):
- FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)
[root@tektutor.org spring-ms]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tektutor/hello-spring-microservice  1.0                2e2aa7f7ab8e       13 seconds ago     428MB
ubuntu               latest             dc4c1391d370       6 days ago         78.1MB
[root@tektutor.org spring-ms]# docker login

USING WEB-BASED LOGIN
To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: F6CF-MLJM
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
[root@tektutor.org spring-ms]# docker push tektutor/hello-spring-microservice:1.0
The push refers to repository [docker.io/tektutor/hello-spring-microservice]
028438a2b34a: Pushed
92bcaa529413: Pushed
59a5c510c4c9: Pushed
1.0: digest: sha256:c01e5e5729f41bb121ea2653f9c408240d990731a77376db63759ea119860949 size: 955
[root@tektutor.org spring-ms]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tektutor/hello-spring-microservice  1.0                2e2aa7f7ab8e       3 minutes ago      428MB
ubuntu               latest             dc4c1391d370       6 days ago         78.1MB
[root@tektutor.org spring-ms]#

Oct 16 15:56
24MAN2848_TR - Google Chrome
WebEx link for the traini... In meeting - Meeting x 24MAN2848_TR x kubernetes-oct-2024/D... x tektutor/spring-ms Tags x login.docker.com/device... x tektutor/hello-spring-mi... x +
hydccloud.rpsconsulting.in/console/#/client/MTESNgBjAG15c3Fs

root@master1:~
root@worker1:~
root@worker2:~
root@server1:~/spring-ms

tektutor/ubuntu      2.0      4c3bdb7b0a91    2 days ago      878MB
tektutor/ubuntu      24.04    fa66ca801d2a    2 days ago      121MB
ubuntu               24.04    dc4c1391d370    6 days ago      78.1MB
ubuntu               latest    dc4c1391d370    6 days ago      78.1MB
nginx                latest    7f553e8bbc89    13 days ago     192MB
gcr.io/k8s-minikube/kicbase v0.0.45   aeed0e1d4642    6 weeks ago     1.28GB
mysql                latest    c757d623b190    2 months ago    586MB
[root@server1 spring-ms]# kubectl get all
NAME                READY   STATUS    RESTARTS   AGE
pod/my-pod           1/1     Running   0           25m
[root@server1 spring-ms]# kubectl create deploy hello --image=tektutor/hello-spring-microservice:1.0 --replicas=3
deployment.apps/hello created
[root@server1 spring-ms]# kubectl get po
NAME                READY   STATUS    RESTARTS   AGE
hello-7cffc688f-5274w  0/1     ContainerCreating   0           4s
hello-7cffc688f-jlk6v  0/1     ContainerCreating   0           4s
hello-7cffc688f-vpjjt  0/1     ContainerCreating   0           4s
my-pod               1/1     Running            0           26m
[root@server1 spring-ms]# kubectl expose deploy/hello --type=NodePort --port=8080
service/hello exposed
[root@server1 spring-ms]# kubectl get svc
NAME    TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
hello   NodePort    10.102.84.169 <none>      8080:30793/TCP  37s
[root@server1 spring-ms]# curl http://master-1:30793
Hello Microservice 1.0 ![root@server1 spring-ms]#

```

Lab - Creating a loadbalancer service in declarative style

Let's create an external loadbalancer service for hello deployment

```

kubectl expose deploy/hello --type=LoadBalancer --port=8080 -o yaml --dry-run=client
kubectl expose deploy/hello --type=LoadBalancer --port=8080 -o yaml --dry-run=client > hello-lb-svc.yaml
kubectl apply -f hello-lb-svc.yaml
kubectl get svc

```

But in order for the loadbalancer service to acquire an external IP, the kubernetes administrator has to install an operator called MetallB. The Metallb operator configures our local k8s cluster to work like it works in AWS/Azure or any public cloud.

Each time someone creates a LoadBalancer service, the metallb controller will be watching, when it detects some one creating a new loadbalancer service, it configures the metallb load balancer to route the traffic to our pods just like how it works in AWS/Azure.

Let's install the metallb operator

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.8/config/manifests/
metallb-native.yaml
```

Lab - Let's add a new type of resource to K8s cluster

Create a file named training-crd.yml with the below content

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: trainings.tektutor.org
spec:
  group: tektutor.org
  scope: Namespaced
  names:
    kind: Training
    listKind: TrainingList
    plural: trainings
    singular: training
    shortNames:
      - train
  version:
    - name: v1
      served: true
      storage: true
  schema:
    openAPIV3Schema:
      type: object
      properties:
        training:
          type: string
        duration:
          type: string
        from:
          type: string
        to:
          type: string
```

Let's create a training resource

```
apiVersion: tektutor.org/v1
kind: Training
metadata:
  name: devops-training
spec:
  training: "Advanced DevOps"
  duration: "5 Days"
  from: "4th Nov 2023"
  to: "8th Nov 2023"
```

Info - Kubernetes Operator Overview

- Kubernetes Operator helps us extend the Kubernetes API or used to add new functionality to Kubernetes
- Operator is a combination of one or more Controllers and Custom Resources

Demo - Configure the metallb operator

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: default
  namespace: metallb-system
spec:
  ipAddressPools:
    - tektutor-metallb-addresspool
```

Lab - Scale up nginx deployment

```
kubectl create namespace jegan
kubectl create deployment nginx --image=nginx:latest --replicas=3
kubectl get po
kubectl scale deploy/nginx --replicas=5
kubectl get po
```

Expected output

Lab - Scale down nginx deployment

```
kubectl get po
kubectl scale deploy/nginx --replicas=3
```



```
kubectl get po
```

Expected output

Lab - Rolling update

Let's create a namespace and deploy nginx v1.8

```
kubectl create namespace jegan  
kubectl config set-context --current --namespace=jegan  
kubectl create deployment nginx --image=nginx:1.18 --replicas=3  
kubectl get po -o yaml | grep image  
kubectl get rs  
kubectl get deploy
```

Let's perform rolling update (upgrade nginx image from 1.18 to 1.19)

```
kubectl set image deploy/nginx nginx=nginx:1.19
```

Let's observe if 2 replicaset are created under nginx deployment

```
kubectl get rs
```

Let's check the pod

```
kubectl get po
```

Let's check the status of the rolling update

```
kubectl rollout status deploy/nginx
```

Let's check the image used by nginx pods after rolling update completed successfully

```
kubectl get po -o yaml | grep image
```

Rolling back to previous version of image

```
kubectl rollout undo deploy/nginx
kubectl rollout status deploy/nginx
kubectl get po -o yaml | grep image
```

Info - Ingress

- a routing rule
- for an Ingress to work, basically 3 things are required
 1. Ingress rule (we will be writing this a yaml file)
 2. Ingress Controller (Nginx Controller or HAProxy Ingress Controller)
 3. Either Nginx Load Balancer or HAProxy Load Balancer
- For instance,
 - We have a bank website with login, balance enquiry, fundtransfer, cheque request, logout, etc.,
 - assume we have developed each of the above features as a microservice, hence each one will be a separate deployment
 - Using ingress we will get a public url, so base path we can route the traffic to different microservices
 - E.g
 - www.somebank.com - home page
 - www.somebank.com/login - this should be forwarded to the login microservice K8s service
 - www.somebank.com/fundtransfer - this should be forwarded to the fundtransfer microservice K8s service

Info - ReplicationController vs ReplicaSet

- In older version of Kubernetes, the only way we could deploy stateless application is via ReplicationController
- The ReplicationController supports both Rolling Update and Scale up/down
- One Controller does two things, which violates Single Responsibility Principle (SOLID - SRP Principle)
- In latest version of kubernetes, they refactored(broken down) ReplicationController functionality into Deployment and ReplicaSet
- The Deployment supports rolling update to stateless applications, while the ReplicaSet supports scale up/down
- ReplicationController is still supported for backward compatibility and legacy application
- We should strictly avoid using ReplicationController for deploying new application

Info - Persistent Volume (PV)

- is a external storage that can be used by the applications running within Pod

- this can be provisioned by Administrator either manually or dynamically
- created on the cluster scope, which any pod running in any project namespace can claim and use PV
- In case the PV is manually provisioned, the administrator will have create Persistent volume
 - with a specific size capacity
 - with specific access modes
 - ReadWriteOnce
 - ReadWriteMany
 - etc
 - StorageClass(optional)
 - Labels (optional)

Info - Persistent Volume Claims (PVC)

- is the way your application can request for external storage
- PV will have to define
 - the size of the storage required
 - storageclass(optional)
 - access mode
 - labels (optional)

Info - What is Ingress?

- routing/forwarding rules
- Ingress helps in forwarding the calls to multiple different services pointing to different deployments
- Ingress is not a service
- We can declaratively create ingress rules, which are retrieved by Ingress Controller, which then configures the load balancer with the forwarding rules we listing in the ingress
- For Ingress to work, we need the below
 - Ingress (rules)
 - Ingress Controller
 - Load Balancer

Info - What is Ingress Controller?

- Ingress Controller is Controller like Deployment Controller, ReplicaSet Controller
- Ingress Controller keeps an eye on every new Ingress created in any project namespace
- Ingress Controller monitors any change done to existing Ingress resources under any project namespace
- Ingress Controller also will monitor when Ingress is deleted in any project namespace

- Ingress Controller picks the rules we mentioned in the Ingress resource and configures the load balancer accordingly
- There are two popular ingress controllers
 - Nginx Ingress Controller
 - HAProxy Ingress Controller
- In our lab setup, we are using HAProxy Load Balancer, hence we need to use HAProxy Ingress Controller