Case Study: Kubernetes Deployment & Service

Developed By: Ashoka Naidu, Devanshu Sharma, Hemanth Singh Kumar, Molshree Singhal, & Shushovan Chakraborty

Table of Contents

Deployment

Deployment Controller changes the actual state to the desired state at a controlled rate. We can create new ReplicaSets, remove existing Deployments and adopt all their resources with new Deployments.

Step 1: Create Deployment File

Create a deployment.yml in a vi editor on your master machine

```
vi deployment.yml
```

Get into the deployment.yml file

```
#the scripts are running from top to bootom and left to right fashion

apiVersion: apps/v1   # in kubernetes all the communcation happen through api version
kind: Deployment      # kind of api we are using
metadata:             # using this we provide the name of the api inside cluister
  name: myapp-deployment     # giving name to the deployment
spec:                        # specification
  replicas: 3                # this will create 3 pods inside the deployment
  selector:                  # to select resources based on the value of labels of pods
    matchLabels:             # it tells what pods the deployment will apply
      app: myapp3            # label format will be in key value format
  template:                  # the template defines reusable content, logic, and parameters
    metadata:
```

```
    name: myapp3-pod       # name of the pod
    labels:                # maps with the pods in form of key:value pairs
      app: myapp3
  spec:                    # indicates that the Pods run one container
    containers:            # stores in form of list
      - name: myapp3-container         # name of the container
        image: piuma/phpsysinfo        # it search/checks for an image with this name
        ports:                         # exposes the Kubernetes service on the specified port within the cluster
          - containerPort: 80          # exposed port 80
```

Step 2: Deploy it & verify it

- When the Deployment file is created . We need to deploy it in our vm with the help of command

```
kubectl apply -f deployment.yml
```

To verify whether the file is deployed or not, we can use can use the following commands

- By getting pod that are available

```
kubectl get pod
```

Output as follows

```
root@master:~# kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE

nginx-deployment-747757dc89-jj5j5   1/1     Running   0          143m
```

- By describing the pod with the hrlp of command

```
kubectl describe pod {name} (nginx-deployment-747757dc89-sr7wx)
```

The describing of pod provides us the following data about that particular pod

```
root@master:~# kubectl describe pod nginx-deployment-747757dc89-sr7wx

Name:         nginx-deployment-747757dc89-sr7wx
Namespace:    default
Priority:     0
Node:         ip-172-31-75-169/172.31.75.169
Start Time:   Wed, 17 Nov 2021 06:46:06 +0000
Labels:       app=nginx
pod-template-hash=747757dc89
Annotations:  <none>
Status:       Running
IP:           10.224.1.15
IPs:
IP:           10.224.1.15
Controlled By:  ReplicaSet/nginx-deployment-747757dc89
Containers:
nginx:
Container ID:   containerd://10e2113764d165b8cf25e0b391e47551a13f6e0d519baf312b883150ff616d93
Image:          piuma/phpsysinfo
Image ID:       sha256:8594ee49d529eb313cbb2f26e29e955cf29577ce55834a7cd85586219c486475
Port:           80/TCP
Host Port:      0/TCP
State:          Running
Started:      Wed, 17 Nov 2021 06:46:07 +0000
Ready:          True
Restart Count:  0
Environment:    <none>
Mounts:
/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-2rkqg (ro)
Conditions:
Type              Status
Initialized       True
```

```
Ready             True
ContainersReady   True
PodScheduled      True
Volumes:
kube-api-access-2rkqg:
Type:                     Projected (a volume that contains injected data from multiple sources)
TokenExpirationSeconds:  3607
ConfigMapName:           kube-root-ca.crt
ConfigMapOptional:       <nil>
DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:                      <none>
```

the above pod describes that the file has been deployed successfully.

- by the help of replica set

```
kubectl get rs
```

Output as follows

```
root@master:~# kubectl get rs

NAME                                     DESIRED   CURRENT   READY   AGE

nginx-deployment-66b6c48dd5                 0         0         0    3h38m
```

- Describing a particular replicaSet

```
kubectl describe rs {name} (nginx-deployment-66b6c48dd5)
```

Output as follows

```
root@master:~# kubectl describe rs nginx-deployment-66b6c48dd5
Name:           nginx-deployment-66b6c48dd5
Namespace:      default
Selector:       app=nginx,pod-template-hash=66b6c48dd5
Labels:         app=nginx
pod-template-hash=66b6c48dd5
Annotations:    [deployment.kubernetes.io/desired-replicas:](http://deployment.kubernetes.io/desired-replicas:) 3
[deployment.kubernetes.io/max-replicas:](http://deployment.kubernetes.io/max-replicas:) 4
[deployment.kubernetes.io/revision:](http://deployment.kubernetes.io/revision:) 1
Controlled By:  Deployment/nginx-deployment
Replicas:       0 current / 0 desired
Pods Status:    0 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
Labels:  app=nginx
pod-template-hash=66b6c48dd5
Containers:
nginx:
Image:        nginx:1.14.2
Port:         80/TCP
Host Port:    0/TCP
Environment:  <none>
Mounts:       <none>
Volumes:        <none>
Events:           <none>
```

Describing of replica set provides the above information including image, port, labels etc...

Service

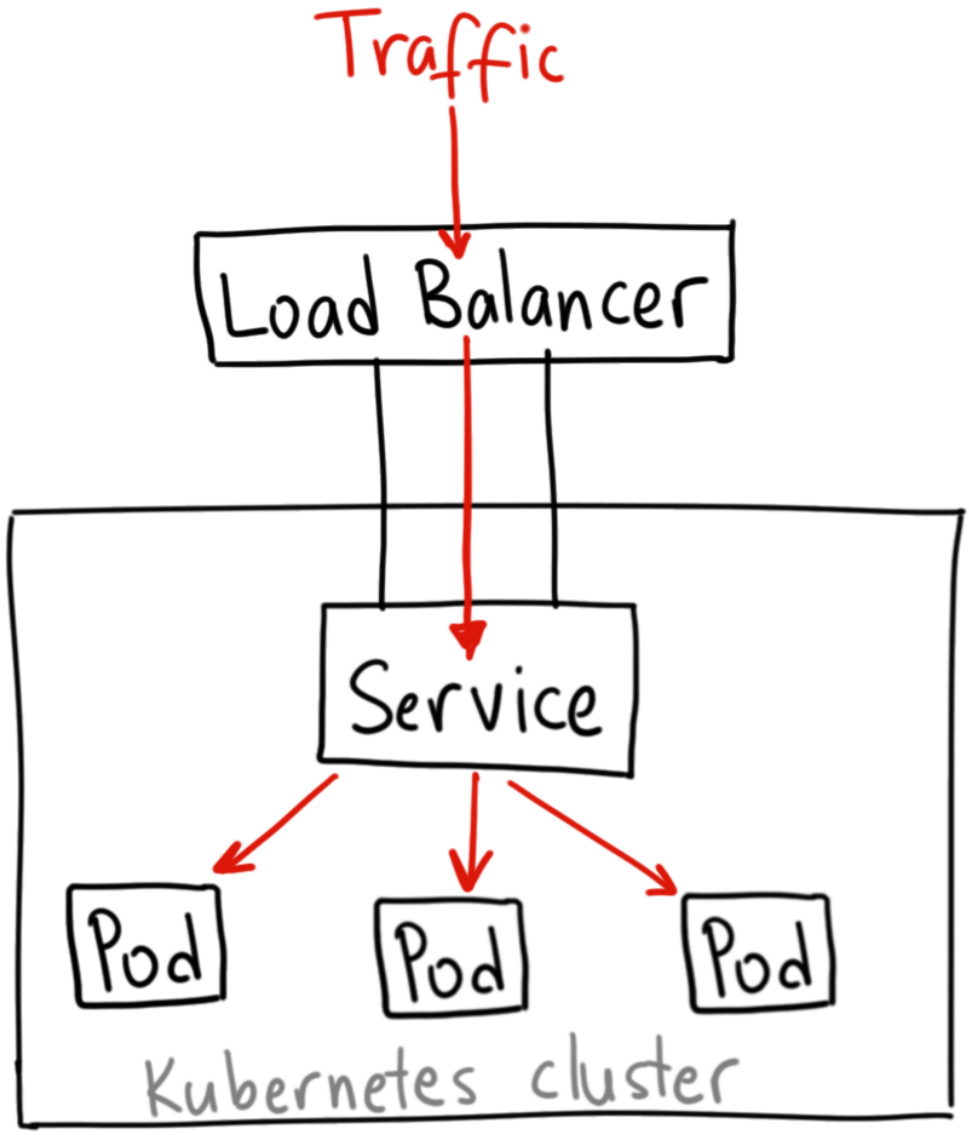
Step 3: Create Service and Expose it

After creating a successful deployment we want to expose our application to the outside world. To do this, we make use of Kubernetes Service.

> According to the Kubernetes documentation, Service is an abstract way to expose an application running on a set of pods as a network service.

What does it mean?

When we want to expose our application to the world, we only need to create a new service in Kubernetes. **It will automatically map the pods of a deployment to the network and generate its own load balancer.**



It does the mapping based on the **pod labels.** After successful execution, **we get endpoints** for each pod which can be seen in the steps for verification.

Execution

In your master machine you can type the following command,

```
kubectl expose deploy myapp-deployment --type=NodePort
```

Let's understand the anatomy.

- kubectl expose deploy - this tells the Kubernetes cluster to **expose a deployment**
- myapp-deployment - this is the name of our deployment manifest file, this will serve as the specification for our service exposure.
- - -type=NodePort - this argument tells the service creator to use the worker machine node to expose the deployment.

The expected output after running the command :-

```
service/myapp-deployment exposed
```

This means our application has been successfully exposed.

Step 4: Verify it

After successful execution, we can verify the status of the service with "kubectl get svc"

```
root@ip-172-31-72-71:~/ kubectl get svc

NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP        5d2h
my-frist-service   NodePort    10.101.97.235   <none>        80:31506/TCP   24h
myapp-deployment   NodePort    10.102.103.44   <none>        80:30914/TCP   157m
```

Here we can see that our service "myapp-deployment" has been auto-generated from the deployment file. It has been exposed at port 80:30914.

From here we can copy the Port number 30914. This port number should be later attached with the public IP address of the slave/node machine to access the exposed application.

```
root@ip-172-31-72-71:~/capgeminimanifestnov# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS      AGE     IP            NODE              NOMINATED NODE   READIN
my-first-deployment-f5dc549d6-x24fw   1/1     Running   0             23h     10.244.1.12   ip-172-31-78-206   <none>           <none>
my-second-pod                       1/1     Running   1 (29h ago)   4d21h   10.244.1.7    ip-172-31-78-206   <none>           <none>
myapp-deployment-7bb646b9dc-2zngx   1/1     Running   0             136m    10.244.1.24   ip-172-31-78-206   <none>           <none>
myapp-deployment-7bb646b9dc-69tm7   1/1     Running   0             136m    10.244.1.23   ip-172-31-78-206   <none>           <none>
myapp-deployment-7bb646b9dc-7pvgs   1/1     Running   0             136m    10.244.1.22   ip-172-31-78-206   <none>           <none>
nginx-deployment-747757dc89-c5tpb   1/1     Running   0             148m    10.244.1.19   ip-172-31-78-206   <none>           <none>
nginx-deployment-747757dc89-ddn8x   1/1     Running   0             148m    10.244.1.20   ip-172-31-78-206   <none>           <none>
nginx-deployment-747757dc89-lh9r4   1/1     Running   0             148m    10.244.1.21   ip-172-31-78-206   <none>           <none>
testpod                             1/1     Running   0             28h     10.244.1.10   ip-172-31-78-206   <none>           <none>
```

Here we can see that there are 3 services for myapp-deployment attached with randomly generated string values.they have different IP addres which are also called **endpoints.**

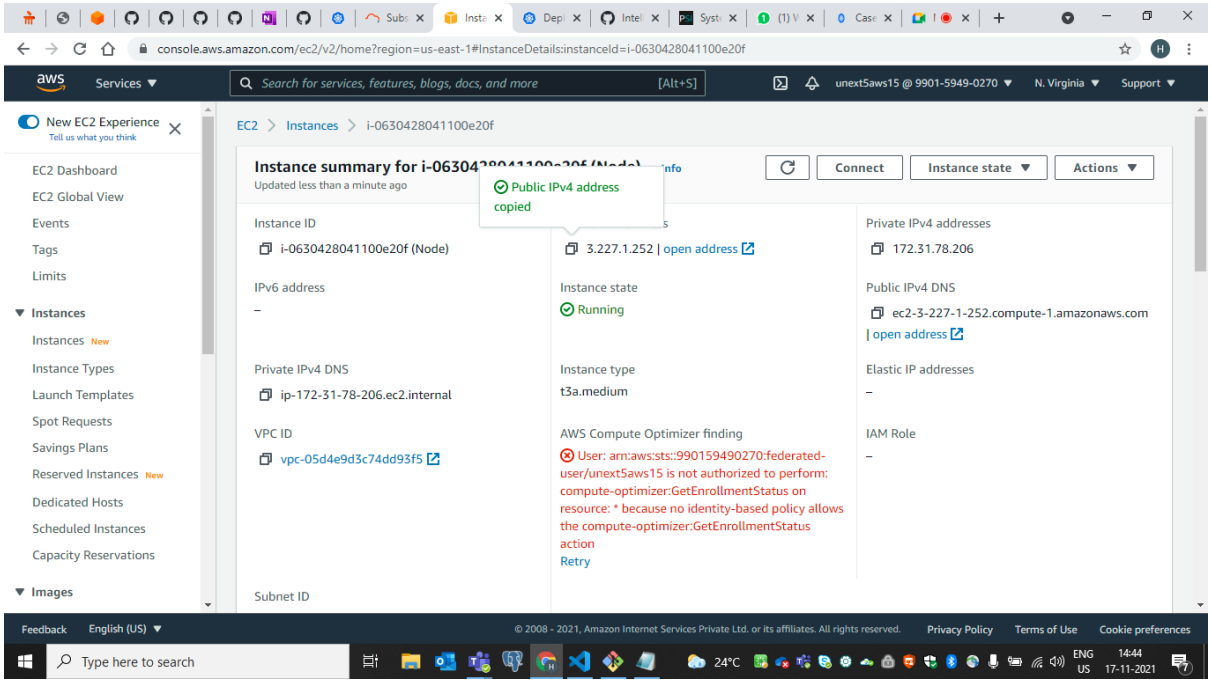
```
myapp-deployment-7bb646b9dc-2zngx        10.244.1.24   ip-172-31-78-206
myapp-deployment-7bb646b9dc-69tm7        10.244.1.23   ip-172-31-78-206
myapp-deployment-7bb646b9dc-7pvgs         10.244.1.22   ip-172-31-78-206
```

To read complete description of the service, we use the "kubectl describe svc" function.

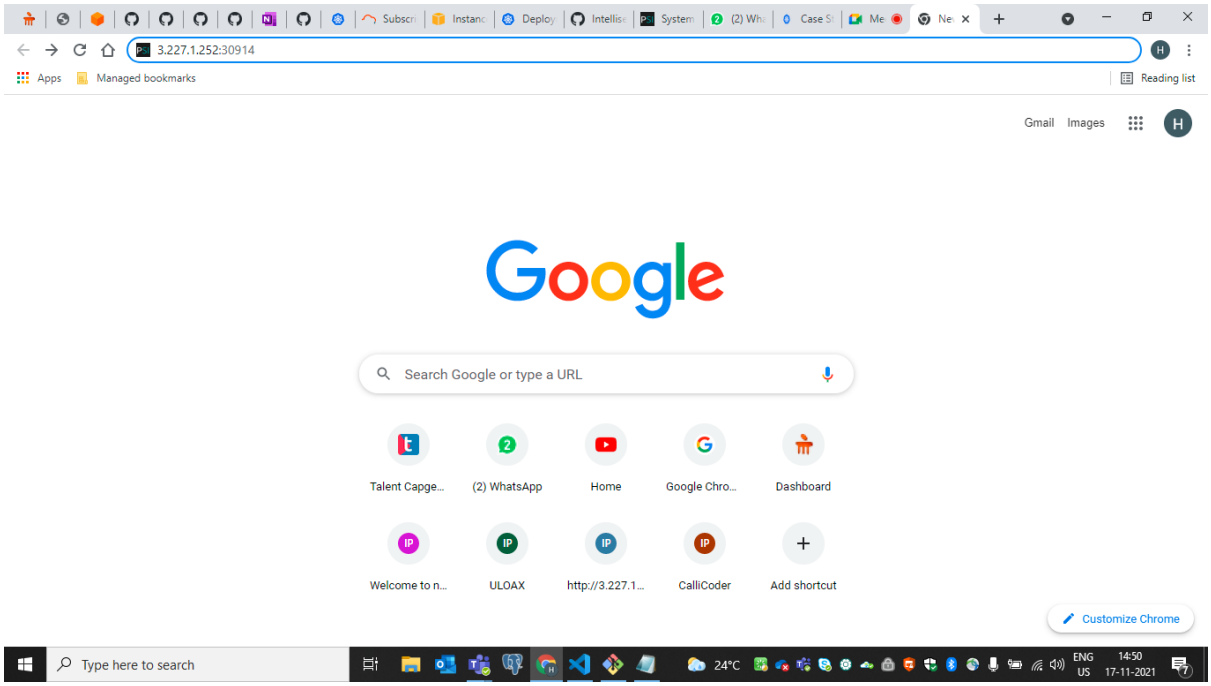
```
root@ip-172-31-72-71:~/ kubectl describe svc

Name:                     myapp-deployment
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=myapp3
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.102.103.44
IPs:                      10.102.103.44
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30914/TCP
Endpoints:                10.244.1.22:80,10.244.1.23:80,10.244.1.24:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

We can find various attributes related to our service here. We can see in the Endpoints that all the are added. These are randomly generated. We can also find our NodePort. In case we want to debug the application, we can also read the report in **events** tab.

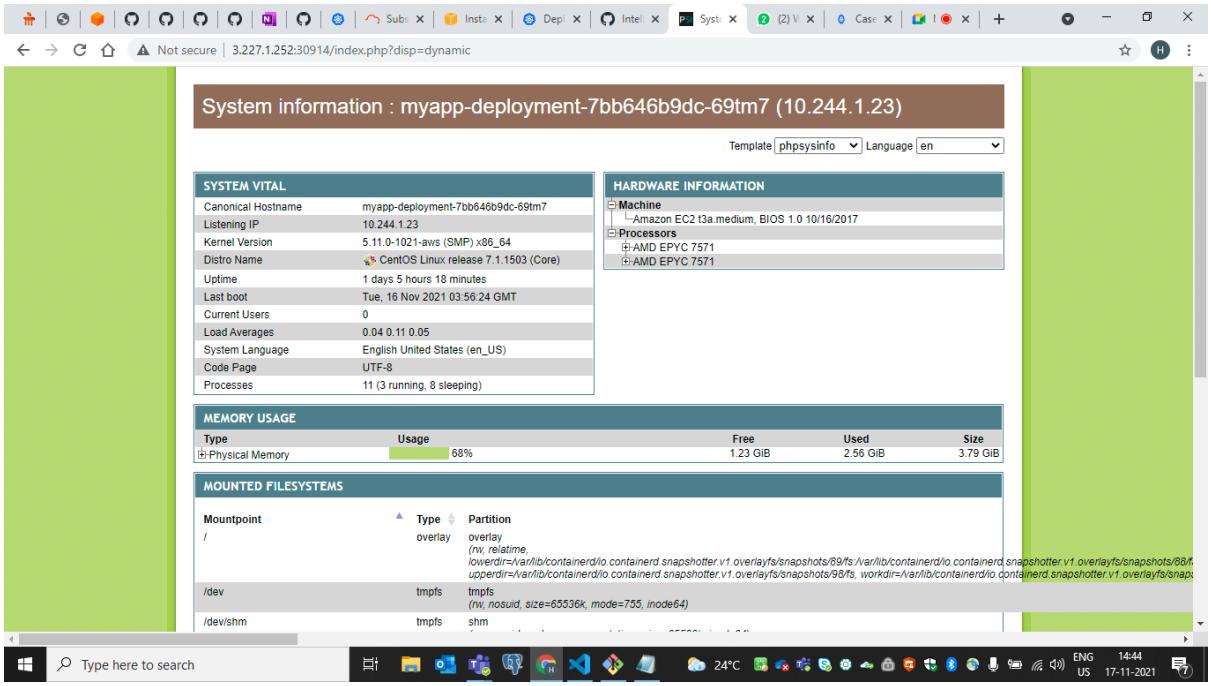
Now we have to copy the IP address of the slave/node machine from our AWS console.



As shown above copy the IP address and open a new tab in the browser and paste the IP address with the service IP address as shown below



Now since the web page is opening without any connection error we can say that the services are exposed. The output is as shown below


Not secure | 3.227.1.252:30914/index.php?disp=dynamic
System information : myapp-deployment-7bb646b9dc-69tm7 (10.244.1.23)
Template phpsysinfo Language en
SYSTEM VITAL
Canonical Hostname myapp-deployment-7bb646b9dc-69tm7
Listening IP 10.244.1.23
Kernel Version 5.11.0-1021-aws (SMP) x86_64
Distro Name CentOS Linux release 7.1.1503 (Core)
Uptime 1 days 5 hours 18 minutes
Last boot Tue, 16 Nov 2021 03:56:24 GMT
Current Users 0
Load Averages 0.04 0.11 0.05
System Language English United States (en_US)
Code Page UTF-8
Processes 11 (3 running, 8 sleeping)
HARDWARE INFORMATION
Machine
Amazon EC2 t3a.medium, BIOS 1.0 10/16/2017
Processors
AMD EPYC 7571
AMD EPYC 7571
MEMORY USAGE
Type Usage Free Used Size
Physical Memory 68% 1.23 GiB 2.56 GiB 3.79 GiB
MOUNTED FILESYSTEMS
Mountpoint Type Partition
/ overlay overlay
/dev tmpfs tmpfs
/dev/shm tmpfs shm
Type here to search
24°C
ENG
US
14:44
17-11-2021