Configure a Webserver in a Pod using Kubernetes

What is pod?

- Un Pod est la plus petite unité déployable qui peut être créée et gérée dans Kubernetes, contenant un ou plusieurs conteneurs.

What is kubelet?

Le kubelet est un agent qui s'exécute sur chaque nœud d'un cluster Kubernetes et assure que les conteneurs sont en cours d'exécution dans un pod.

What is kubernetes?

- Kubernetes est un système open-source de gestion de conteneurs qui permet d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées.
- Kubernetes est un système pour gérer des conteneurs à grande échelle.
- Il organise les conteneurs en clusters en utilisant un nœud principal pour contrôler des nœuds secondaires.
- Kubeadm aide à installer Kubernetes qui, dans certains cas, peut fonctionner sur un seul nœud faisant office à la fois de maître et d'ouvrier.
- AWS est utilisé pour l'infrastructure de base, avec EC2 pour les ressources informatiques et Amazon Linux 2 comme système d'exploitation.
- Les pods sont les plus petites unités de Kubernetes contenant les conteneurs, qui peuvent être multiples dans un seul pod.
- Kubernetes facilite la gestion des applications conteneurisées, même dans des environnements étendus.
- Un pod sera créé pour exécuter un serveur web Apache dans ce laboratoire spécifique.
- L'utilitaire de ligne de commande kubectl est utilisé pour interagir avec les conteneurs, indépendamment du moteur de conteneur utilisé.

- Kubernetes permet des déploiements flexibles et économiques, souvent sans interruption de service.

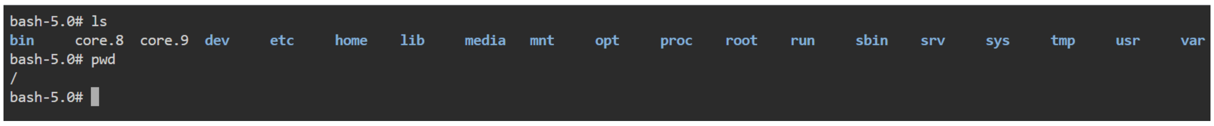
Task Details

1. Launching Lab Environment.
2. SSH into the EC2 instance through the Whiz terminal
3. Install an Apache Server
4. Create a Pod with Apache server
5. Create and publish the page
6. Validation Test

Lab Steps

Task 1: Launching Lab Environment

1. Launch the lab environment by clicking on the **Start Lab** button.
2. Please wait until the lab environment is provisioned. It will take around 2 minutes to provision the lab environment which contains the single node Kubernetes cluster. The unique Username, Public IP address, and the Password are displayed on the right corner of the screen.
3. Alongside the EC2 instance, the Whizlabs terminal is also being provisioned.



```
bash-5.0# ls
bin     core.8  core.9  dev     etc     home    lib     media   mnt     opt     proc    root    run     sbin    srv     sys     tmp     usr     var
bash-5.0# pwd
/
bash-5.0# 
```

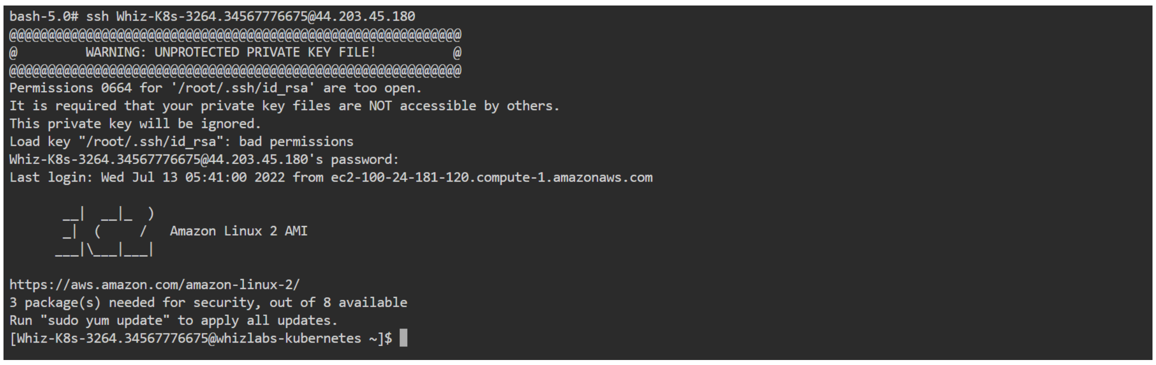
Note : If you face any issues, please go through FAQs and Troubleshooting for Labs.

Task 2 : SSH into the EC2 instance through the Whiz terminal:

1. Copy the IP address and the Password for SSH to the EC2 instance configured with the Kubernetes cluster.
2. Run the following command to SSH into the system:

```
ssh username@ip_address
```

1. Replace the unique username and the IP address with the details of your own lab environment.
2. Type yes if prompted for the SSH permission for the first time. Then, Insert the password in the prompt as it is.



```
bash-5.0# ssh Whiz-K8s-3264.34567776675@44.203.45.180
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0664 for '/root/.ssh/id_rsa' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/root/.ssh/id_rsa": bad permissions
Whiz-K8s-3264.34567776675@44.203.45.180's password:
Last login: Wed Jul 13 05:41:00 2022 from ec2-100-24-181-120.compute-1.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
3 package(s) needed for security, out of 8 available
Run "sudo yum update" to apply all updates.
[Whiz-K8s-3264.34567776675@whizlabs-kubernetes ~]$
```

3. Now switch the user to root with the command below:

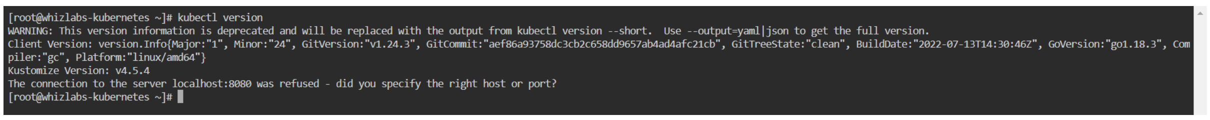
```
sudo su root
```

This ec2 instance is configured with Kubernetes packages and all prerequisites are fulfilled. This is a single node cluster and we will use the same node as the client node to communicate and control the containers.

Task 3: Verify the cluster configuration

1. Run the below command to check the version of `kubectl` package. Kubectl allows us to control the cluster via a command-line interface. One can run the kubectl commands followed by the resource arguments to send API requests to the master node and run the intended task over the worker node.

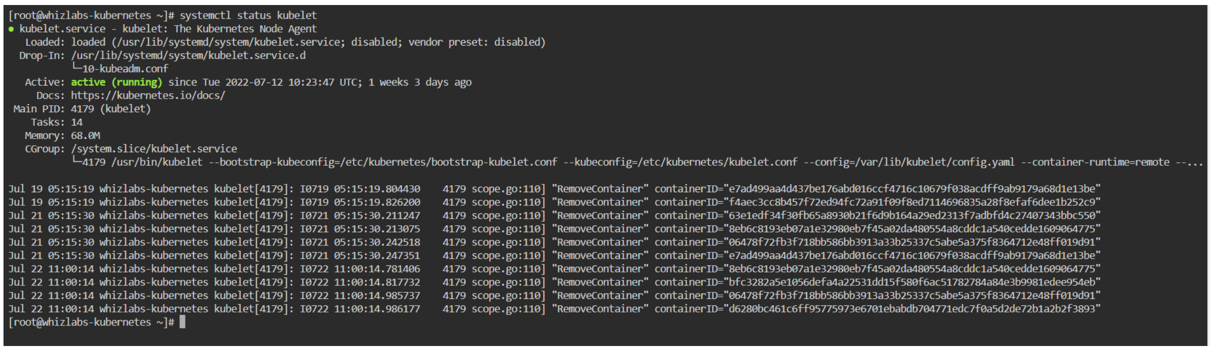
   ```
   kubectl version
   ```



```
[root@whizlabs-kubernetes ~]# kubectl version
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short.  Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.3", GitCommit:"aef86a93758dc3cb2c658dd9657ab4ad4afc21cb", GitTreeState:"clean", BuildDate:"2022-07-13T14:30:46Z", GoVersion:"go1.18.3", Com
piler:"gc", Platform:"linux/amd64"}
Kustomize Version: v4.5.4
The connection to the server localhost:8080 was refused - did you specify the right host or port?
[root@whizlabs-kubernetes ~]#
```

2. Now, verify the kubelet service is running by the following command. Kubelet service pods are running on both the master and worker nodes and this service is responsible for sending and receiving API requests within the cluster.

```
systemctl status kubelet
```

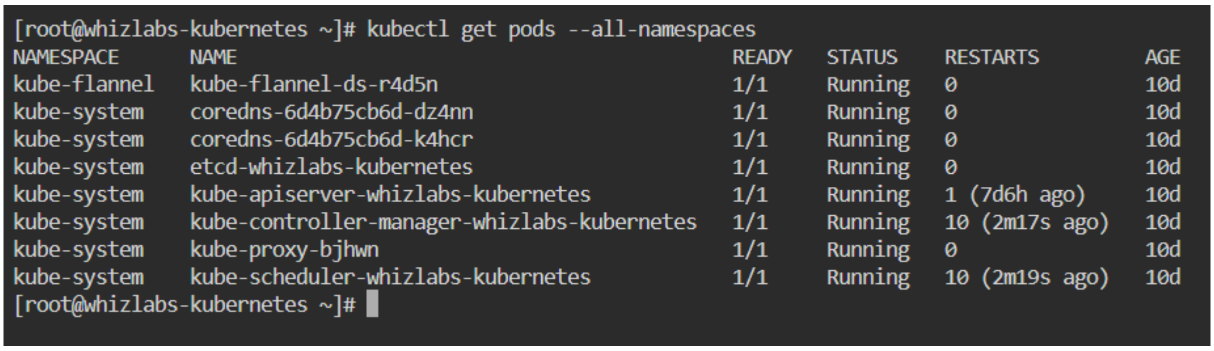


```
[root@whizlabs-kubernetes ~]# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; disabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: active (running) since Tue 2022-07-12 10:23:47 UTC; 1 weeks 3 days ago
     Docs: https://kubernetes.io/docs/
 Main PID: 4179 (kubelet)
    Tasks: 14
   Memory: 68.0M
   CGroup: /system.slice/kubelet.service
           └─4179 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --config=/var/lib/kubelet/config.yaml --container-runtime=remote --...

Jul 19 05:15:19 whizlabs-kubernetes kubelet[4179]: I0719 05:15:19.804430    4179 scope.go:110] "RemoveContainer" containerID="e7ad499aa4d437be176abd016ccf4716c10679f038acdff9ab9179a68d1e13be"
Jul 19 05:15:19 whizlabs-kubernetes kubelet[4179]: I0719 05:15:19.826200    4179 scope.go:110] "RemoveContainer" containerID="f4aec3cc8b457f72ed94fc72a91f09f8ed7114696835a28f8efaf6dee1b252c9"
Jul 21 05:15:30 whizlabs-kubernetes kubelet[4179]: I0721 05:15:30.211247    4179 scope.go:110] "RemoveContainer" containerID="63e1edf34f30fb65a8930b21f6d9b164a29ed2313f7adbfd4c27407343bbc550"
Jul 21 05:15:30 whizlabs-kubernetes kubelet[4179]: I0721 05:15:30.213075    4179 scope.go:110] "RemoveContainer" containerID="8eb6c8193eb07a1e32980eb7f45a02da480554a8cddc1a540cedde1609064775"
Jul 21 05:15:30 whizlabs-kubernetes kubelet[4179]: I0721 05:15:30.242518    4179 scope.go:110] "RemoveContainer" containerID="06478f72fb3f718bb586bb3913a33b25337c5abe5a375f8364712e48ff019d91"
Jul 21 05:15:30 whizlabs-kubernetes kubelet[4179]: I0721 05:15:30.247351    4179 scope.go:110] "RemoveContainer" containerID="e7ad499aa4d437be176abd016ccf4716c10679f038acdff9ab9179a68d1e13be"
Jul 22 11:00:14 whizlabs-kubernetes kubelet[4179]: I0722 11:00:14.781406    4179 scope.go:110] "RemoveContainer" containerID="8eb6c8193eb07a1e32980eb7f45a02da480554a8cddc1a540cedde1609064775"
Jul 22 11:00:14 whizlabs-kubernetes kubelet[4179]: I0722 11:00:14.817732    4179 scope.go:110] "RemoveContainer" containerID="bfc3282a5e1056defa4a22531dd15f580f6ac51782784a84e3b9981edee954eb"
Jul 22 11:00:14 whizlabs-kubernetes kubelet[4179]: I0722 11:00:14.985737    4179 scope.go:110] "RemoveContainer" containerID="06478f72fb3f718bb586bb3913a33b25337c5abe5a375f8364712e48ff019d91"
Jul 22 11:00:14 whizlabs-kubernetes kubelet[4179]: I0722 11:00:14.986177    4179 scope.go:110] "RemoveContainer" containerID="d6280bc461c6ff95775973e6701ebabdb704771edc7f0a5d2de72b1a2b2f3893"
[root@whizlabs-kubernetes ~]#
```

3. Last step is to verify all the prerequisite pods are up and running. These pods are the helping hand in performing cluster operations like scheduling, network policy and route management, API request handling, proxy service and so on. Run the following command to fetch the pods running in all the namespaces.

```
kubectl get pods  --all-namespaces
```



```
[root@whizlabs-kubernetes ~]# kubectl get pods --all-namespaces
NAMESPACE      NAME                                          READY   STATUS    RESTARTS         AGE
kube-flannel   kube-flannel-ds-r4d5n                         1/1     Running   0                10d
kube-system    coredns-6d4b75cb6d-dz4nn                      1/1     Running   0                10d
kube-system    coredns-6d4b75cb6d-k4hcr                      1/1     Running   0                10d
kube-system    etcd-whizlabs-kubernetes                      1/1     Running   0                10d
kube-system    kube-apiserver-whizlabs-kubernetes            1/1     Running   1 (7d6h ago)     10d
kube-system    kube-controller-manager-whizlabs-kubernetes   1/1     Running   10 (2m17s ago)   10d
kube-system    kube-proxy-bjhwn                              1/1     Running   0                10d
kube-system    kube-scheduler-whizlabs-kubernetes            1/1     Running   10 (2m19s ago)   10d
[root@whizlabs-kubernetes ~]#
```

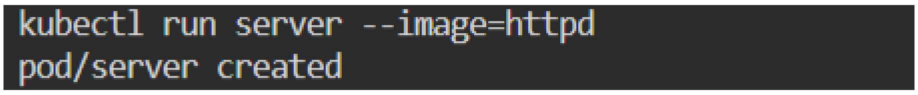
Make sure all pods display status **RUNNING** which implies all the required pods are up and running and we can perform the further steps.

Task 4: Create a Pod with apache server

1. In this step, we will launch a pod running the Apache web server inside a container. We will use the official container image of the web server which is

httpd. Run the following command to deploy a pod with httpd container image.

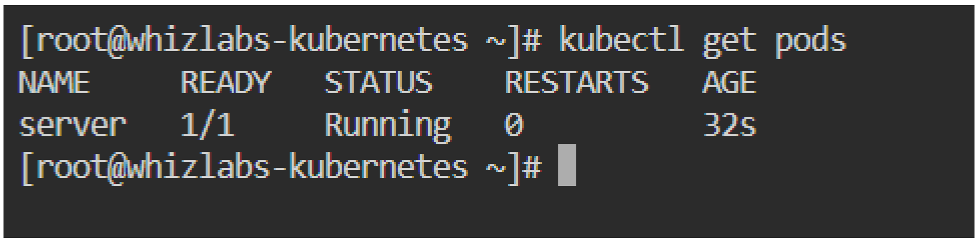
```
kubectl run server --image=httpd
```



```
kubectl run server --image=httpd
pod/server created
```

2. Verify the pod is running or not by the following command. Initially, the node will download the container image from the image registry and hence it will take a few seconds to spin up the container. The following command will stream the phases in which the pod gets deployed.

```
kubectl get pods
```



```
[root@whizlabs-kubernetes ~]# kubectl get pods
NAME     READY   STATUS    RESTARTS   AGE
server   1/1     Running   0          32s
[root@whizlabs-kubernetes ~]#
```

1. Observe the behind the scene process and terminate the command after the pod status is running. Press **cltr+c** to terminate the command execution.
2. Now let's go inside the container and configure web page on top of the container image downloaded from the registry

```
kubectl exec server -it  -- bash
```

You will see a change in the user and directory of the current terminal which implies that we are now inside the container.

Task 5 : Create and publish the page

1. First, navigate to the document root folder of the web server. The document root is the directory from which the content is being served. This simply means that whatever web pages are available inside this folder location, will be served to the end user.

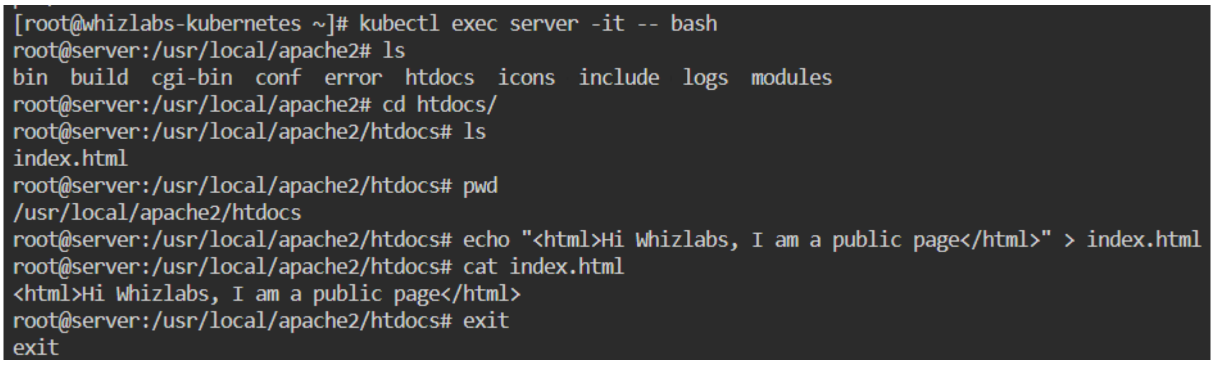
```
cd /usr/local/apache2/htdocs
```

3. To add the contents into the index.html file using echo, copy and paste the below command to the shell of the container.

```
echo "<html>Hi Whizlabs, I am a public page</html>" > index.h
```

4. Now, exit the container and return to the base Whiz Terminal

```
exit
```



```
[root@whizlabs-kubernetes ~]# kubectl exec server -it -- bash
root@server:/usr/local/apache2# ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
root@server:/usr/local/apache2# cd htdocs/
root@server:/usr/local/apache2/htdocs# ls
index.html
root@server:/usr/local/apache2/htdocs# pwd
/usr/local/apache2/htdocs
root@server:/usr/local/apache2/htdocs# echo "<html>Hi Whizlabs, I am a public page</html>" > index.html
root@server:/usr/local/apache2/htdocs# cat index.html
<html>Hi Whizlabs, I am a public page</html>
root@server:/usr/local/apache2/htdocs# exit
exit
```

Till here, we have launched and configured a pod and Apache web server is running inside it. And we configured a sample index page and pasted the same into the document root of the server.

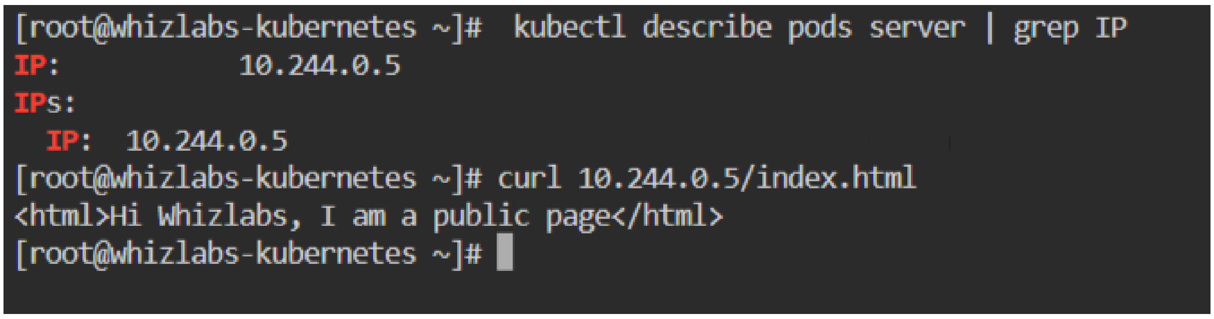
Task 6: Validation Test

1. Let’s fetch the IP address of the container by the following command:

```
kubectl describe pods server | grep IP
```

2. We will now put a curl request within the terminal to check whether the container is hosting the webserver or not

```
curl <ip-of-container>/index.html
```



```
[root@whizlabs-kubernetes ~]#  kubectl describe pods server | grep IP
IP:           10.244.0.5
IPs:
  IP:  10.244.0.5
[root@whizlabs-kubernetes ~]# curl 10.244.0.5/index.html
<html>Hi Whizlabs, I am a public page</html>
[root@whizlabs-kubernetes ~]# 
```

3. You will see the content from the index.html file on the command line which indicates that the webserver is successfully hosted on a container. The server is only accessible locally as we have not configured the networking and routing for the pod.

Completion and Conclusion

1. You have successfully verified Kubernetes cluster components.
2. You have successfully logged into the cluster through the Whiz terminal and deployed a sample web server pod inside it.
3. You have successfully created a webpage and published it backed by a container image named httpd.

End Lab

1. Exit from the terminal SSH session.
2. You have successfully completed the lab.
3. Once you have completed the steps, click on **End Lab** from your Whizlabs dashboard.