



# Kubernetes cluster with High Availability and Scalability of Web server



Guided by:  
Ms. Tejaswini Apate

Presented by:  
Pranjali Patil  
Sushmita Diwakar  
Sukanya Mane  
Trishna Dhruw  
Tarun Shori

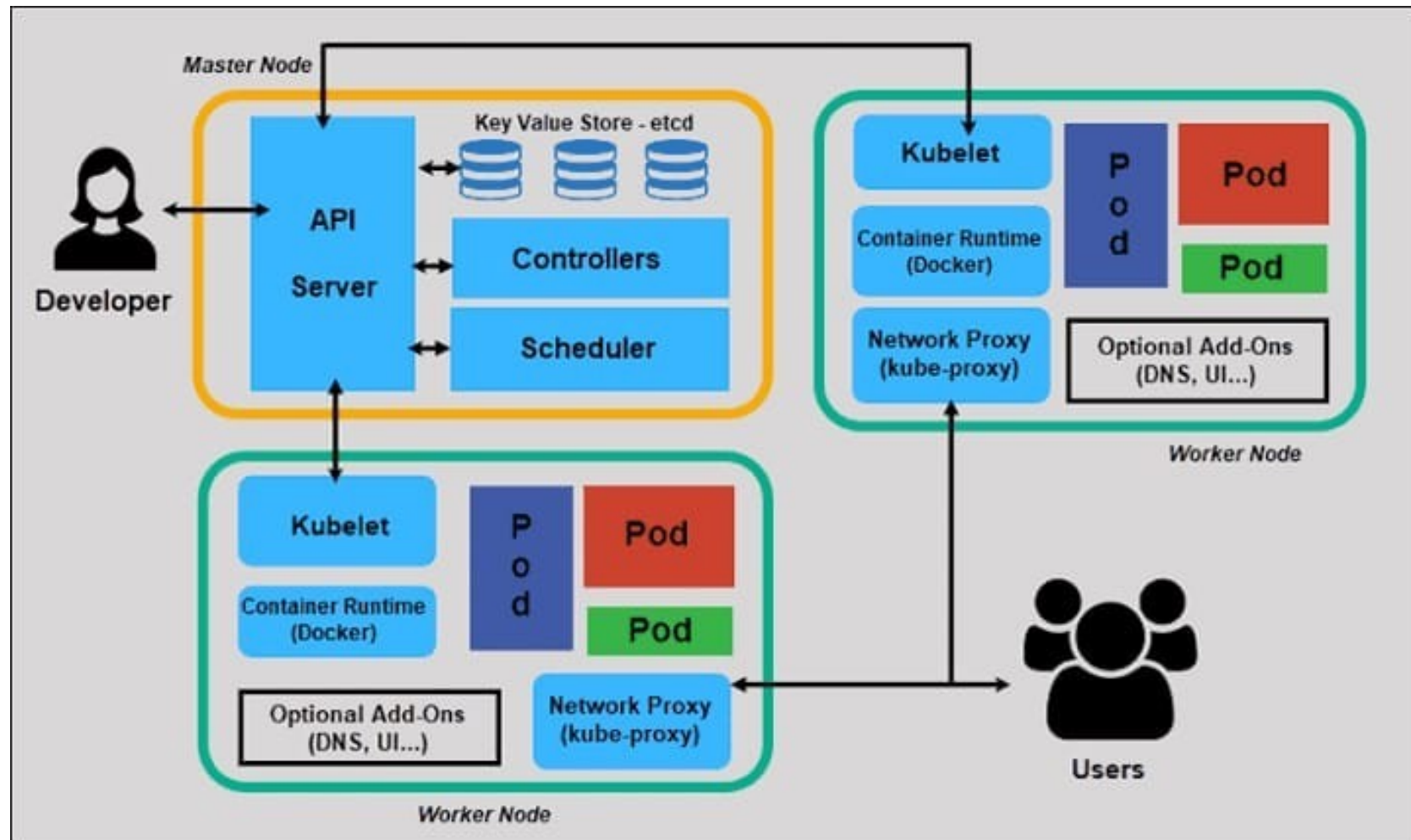


# Introduction

In today's era of cloud-native applications, ensuring high availability and scalability of web services is crucial for businesses to meet the demands of their users while maintaining reliability and performance. Kubernetes, as a powerful container orchestration platform, provides robust solutions for deploying, managing, and scaling applications in a cloud-native environment.

This project involves the creation of a strong Kubernetes cluster deployed on AWS instances. We use Docker containers within Kubernetes pods to manage Apache Web server. This server is orchestrated to ensure both high availability and scalability. With careful configuration and management. This deployment architecture promises to elevate the reliability and efficiency of web services while maximizing resource utilization.

# Work flow





# Use Cases

## 1. Implement High Availability:

- Configured Kubernetes Pod Replication Controllers or Deployments to ensure redundancy

## 2. Enable Scaling:

- Implement Kubernetes Horizontal Pod Autoscaler (HPA) to automatically scale the web server based on demand.
- Adjust HPA settings for CPU or memory utilization thresholds.

## 3. Configure Load Balancing:

- Used Kubernetes Services to expose the web server, automatically an creating internal load balancer.



# System And Software Requirements

## System requirement:

RAM: 4 GB

HDD: 15 GB

Processors: 2 cores

OS: AWS Linux

## Software requirement:

Kubernetes Tools: Kubeadm

Container Runtime: Containerd

Networking Plugin: Calico

Web Server: Apache2 httpd

Cloud Provider: AWS

Repository: Docker hub

# Instance Snapshots

## Setting up AWS Instance:

The screenshot shows the AWS Management Console interface for EC2 Instances. The top navigation bar includes the AWS logo, Services menu, a search bar, and user information (Mumbai, Tarun shori). The left sidebar lists various AWS services, with 'Instances' selected under the 'EC2' category. The main content area displays a table of three EC2 instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
master	i-0e7b963c54f417345	Running	t2.medium	2/2 checks passed	View alarms	ap-south-1b	ec2-65-1-248-38.ap-so...	65.1.248.38	-
worker1	i-000d2037ceb603034	Running	t2.medium	2/2 checks passed	View alarms	ap-south-1b	ec2-3-109-59-217.ap-s...	3.109.59.217	-
worker2	i-096c070e044ea7450	Running	t2.medium	2/2 checks passed	View alarms	ap-south-1b	ec2-35-154-136-92.ap-...	35.154.136.92	-

Below the table, there is a section titled 'Select an instance' with a search bar and a list of instance IDs. The page also includes a 'Launch instances' button and a 'Connect' button.

# Create Security group for instances of master and worker nodes.

## For master node :

The screenshot displays the AWS Management Console interface. The left-hand navigation pane shows the 'Instances' section expanded, with 'Instances' selected. The main content area shows the configuration for a security group named 'sg-099ef03a825deed89 (launch-wizard-15)'. Under the 'Inbound rules' tab, a table lists ten rules, all of which are inbound, using the TCP protocol, and allowing traffic from the source '0.0.0.0/0'. The rules are defined by their Security group rule IDs and port ranges. Below the inbound rules, the 'Outbound rules' section is visible but currently empty.

Name	Security group rule ID	Port range	Protocol	Source
-	sgr-09ae0c1b3b1e08dff	80	TCP	0.0.0.0/0
-	sgr-0e87af341cb95ba5a	10257	TCP	0.0.0.0/0
-	sgr-0312c6cf66bbf1a4b	22	TCP	0.0.0.0/0
-	sgr-0328b2e33929691b5	179	TCP	0.0.0.0/0
-	sgr-01ffe8dce8b7d0c8d	10259	TCP	0.0.0.0/0
-	sgr-09264034b2d6d971c	443	TCP	0.0.0.0/0
-	sgr-09dbeb7617b2fd715	2380	TCP	0.0.0.0/0
-	sgr-0e94436fdcb1b9bae	8001	TCP	0.0.0.0/0
-	sgr-0e1039b0858cdc916	10252	TCP	0.0.0.0/0
-	sgr-042fbf898b14a9108	6443	TCP	0.0.0.0/0

For worker nodes :

The screenshot displays the AWS Management Console interface. The top navigation bar includes the AWS logo, a 'Services' menu, a search bar with the placeholder 'Search' and a keyboard shortcut '[Alt+S]', and user information for 'Mumbai' and 'Tarun shori'. The left-hand navigation pane lists various AWS services, including 'EC2 Dashboard', 'EC2 Global View', 'Events', 'Instances', 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Capacity Reservations', 'Images', 'AMI Catalog', and 'Elastic Block Store'. The main content area is titled 'Inbound rules' and features a search bar labeled 'Filter rules' and a pagination control showing '1'. Below this is a table with five columns: 'Name', 'Security group rule ID', 'Port range', 'Protocol', and 'Source'. The table contains seven rows of inbound rules. Below the inbound rules section is the 'Outbound rules' section, which also has a search bar labeled 'Filter rules' and a pagination control showing '1'. It contains a single row in a table with five columns: 'Name', 'Security group rule ID', 'Port range', 'Protocol', and 'Destination'.

Name	Security group rule ID	Port range	Protocol	Source
-	sgr-0398c61da985da718	443	TCP	0.0.0.0/0
-	sgr-0aea357a56ef5072f	30000 - 32767	TCP	0.0.0.0/0
-	sgr-0b9d4d90786661a1b	4789	UDP	0.0.0.0/0
-	sgr-0d91f86626ebf3574	80	TCP	0.0.0.0/0
-	sgr-0888eb2d0609257de	179	TCP	0.0.0.0/0
-	sgr-0ae5603342cf700e1	10250	TCP	0.0.0.0/0
-	sgr-0200583437d955795	22	TCP	0.0.0.0/0

Name	Security group rule ID	Port range	Protocol	Destination
-	sgr-07ef5af25e4da5a4b	All	All	0.0.0.0/0



# Kubernetes Installation

## Step 1: Disable swap space

To disable swap space and to make the changes persistent:

```
$ sudo swapoff -a  
$ sudo sed -i '/swap/d' /etc/fstab
```

## Step 2: Disable SELinux

To achieve this, open the SELinux configuration file, change the SELINUX value from enforcing to permissive or alternatively you can use the sed command as follows.

```
$ sudo vi /etc/selinux/config  
SELINUX=permissive  
$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'  
/etc/selinux/config
```

## Step 3: Configure networking in master and worker node and install traffic control utility and allow specific port on master and worker nodes.

```
$ sudo vi /etc/hosts  
$ sudo dnf install -y iproute-tc
```

## Step 5: Install Containerd container runtime:

First, create a modules configuration file for Kubernetes, then load both the modules using the modprobe command.

```
$ sudo tee /etc/modules-load.d/containerd.conf <<EOF
$ sudo modprobe overlay
$ sudo modprobe br_netfilter
```

Next, configure the required sysctl parameters as follows:

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

$ sudo sysctl --system
```

## Step 6: Install containerd runtime

```
$ sudo dnf config-manager --add-repo
```

```
https://download.docker.com/linux/centos/docker-ce.repo
```

```
$ sudo yum install containerd.io -y
```

```
$ sudo containerd config default | sudo tee /etc/containerd/config.toml  
>dev/null 2>&1
```

```
$ sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g'  
/etc/containerd/config.toml
```

Start and enable the containerd service.

```
$ sudo systemctl start containerd
```

```
$ sudo systemctl enable containerd
```

```
$ sudo systemctl status containerd
```

## Step 7: Install Kubernetes Packages

```
$ sudo vi /etc/yum.repos.d/kubernetes.repo
```

Finally, install k8s package as follows.

```
$ sudo dnf install -y kubelet kubeadm kubectl disableexcludes=kubernetes
```

```
$ sudo systemctl enable kubelet
```

```
$ sudo systemctl start kubelet
```

# Kubernetes Cluster Installation:

\$ sudo kubeadm init --control-plane-endpoint=master

Join the worker nodes to run token:

This output confirms that the master node is ready. Additionally, you can check the pod namespaces:

```
[root@master /]#  
[root@master /]#  
[root@master /]# kubectl get nodes -o wide  
NAME          STATUS    ROLES          AGE    VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE          KERNEL-VERSION    CONTAINER-RUNTIME  
master        Ready     control-plane  4d7h   v1.28.6    172.31.13.236  <none>         Amazon Linux 2023  6.1.75-99.163.amzn2023.x86_64  containerd://1.7.11  
worker1       Ready     <none>         4d7h   v1.28.6    172.31.13.128  <none>         Amazon Linux 2023  6.1.75-99.163.amzn2023.x86_64  containerd://1.7.11  
worker2       Ready     <none>         4d7h   v1.28.6    172.31.0.223   <none>         Amazon Linux 2023  6.1.75-99.163.amzn2023.x86_64  containerd://1.7.11  
[root@master /]#  
[root@master /]#
```

```
[root@master /]# kubectl get nodes  
NAME          STATUS    ROLES          AGE    VERSION  
master        Ready     control-plane  4d7h   v1.28.6  
worker1       Ready     <none>         4d7h   v1.28.6  
worker2       Ready     <none>         4d7h   v1.28.6  
[root@master /]#
```

# Create Docker Image And Push To Docker Hub

Create a Dockerfile: Create a Dockerfile in our project directory with the necessary instructions to build your Docker image.

Build the Docker image: navigate to the directory containing your Dockerfile and execute the following command to build the Docker image.

```
[root@master finalwebsite]# docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
hpcsaweb             latest          e9101332018d    12 hours ago    168MB
tarunshori/hpcsaweb <none>         c0cf8eb4d4f7    3 days ago      167MB
[root@master finalwebsite]#
[root@master finalwebsite]# docker tag hpcsaweb tarunshori/hpcsaweb:5
[root@master finalwebsite]#
[root@master finalwebsite]# docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
hpcsaweb             latest          e9101332018d    12 hours ago    168MB
tarunshori/hpcsaweb 5               e9101332018d    12 hours ago    168MB
tarunshori/hpcsaweb <none>         c0cf8eb4d4f7    3 days ago      167MB
[root@master finalwebsite]#
[root@master finalwebsite]# docker push tarunshori/hpcsaweb:5
The push refers to repository [docker.io/tarunshori/hpcsaweb]
188e1442e17f: Layer already exists
83e84e8d7bd1: Layer already exists
ee9a39d3b67e: Layer already exists
70d67450158d: Layer already exists
5f70bf18a086: Layer already exists
8f562cbc866f: Layer already exists
ceb365432eec: Layer already exists
5: digest: sha256:867510099f12c48c9932c451671d5daa372b8eb7767c0d69b564056d43045dd9 size: 1782
[root@master finalwebsite]#
```

Verify on Docker Hub: After pushing the image, verify that it's available on Docker Hub by visiting repository's page on the Docker Hub website.

The screenshot shows the Docker Hub interface for the repository `tarunshori/hpcsaweb`. The page is titled "Tags" and displays a list of image tags. The top navigation bar includes the Docker Hub logo, "Explore", "Repositories" (selected), "Organizations", a search bar, and user controls. The breadcrumb trail is `tarunshori / Repositories / hpcsaweb / Tags`. Below the breadcrumb, there are tabs for "General", "Tags" (selected), "Builds", "Collaborators", "Webhooks", and "Settings". A control bar at the top of the tag list includes a checkbox, "Sort by" (set to "Newest"), a "Filter Tags" input, and a "Delete" button. The tag list contains two entries:

Tag	Last pushed	OS/ARCH	Last Pull	Compressed Size
<code>5</code> Last pushed 34 minutes ago by <a href="#">tarunshori</a> DIGEST: <a href="#">867510099f12</a>		linux/amd64	13 hours ago	61.69 MB
<code>4</code> Last pushed 13 hours ago by <a href="#">tarunshori</a> DIGEST: <a href="#">867510099f12</a>		linux/amd64	13 hours ago	61.69 MB


Each tag entry includes a checkbox on the left and a "docker pull" command button on the right. The bottom of the page shows the start of a third tag entry.

## Create YAML Manifest:

run an application by creating a Kubernetes Deployment, Services, Horizontal Pod Autoscaler and other object. we describe all configurations in a YAML file.

Create a Deployment based on the YAML file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web
  namespace: project-website
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-web
  template:
    metadata:
      labels:
        app: my-web
    spec:
      containers:
        - name: web-container
          image: docker.io/tarunshori/hpcsaweb
          ports:
            - containerPort: 80
          volumeMounts:
            - name: data-volume
              mountPath: /data/
      volumes:
        - name: data-volume
          persistentVolumeClaim:
            claimName: web-pvc # Reference to the PVC
```




**PersistentVolume (PV):** A PersistentVolume is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using a StorageClass. In our case we provisioned 4 gb.

**PersistentVolumeClaim (PVC):** When a user creates a PVC, Kubernetes finds a matching PV that satisfies the claim's requirements (capacity, access mode, etc.) and binds the claim to the PV. In our case we binds 2 gb .

**livenessProbe:** This probe checks whether the container is alive or not. In this case, it sends an HTTP GET request to / on port 80 of the container. If the container responds successfully, indicating that it's alive, Kubernetes considers it healthy. If the probe fails, Kubernetes restarts the container after a specified period.

**readinessProbe:** This probe checks whether the container is ready to serve traffic. Similar to the liveness probe, it sends an HTTP GET request to / on port 80. If the container responds successfully, indicating that it's ready to serve traffic, Kubernetes considers it ready. If the probe fails, Kubernetes stops sending traffic to the container until it becomes ready. Again,






**Deployment:** A Deployment manages a set of replicated Pods in the cluster. It provides declarative updates to applications, such as rolling out new features or updating container images. Deployments ensure that desired state is maintained by managing replica sets and controlling the rollout process.

**Service:** A Service defines a set of Pods and a policy to access them. It provides an abstract way to expose an application running on a set of Pods as a network service. Services enable external traffic to access applications running inside the cluster and allow intercommunication between different parts of the application. In our case we use NodePort service

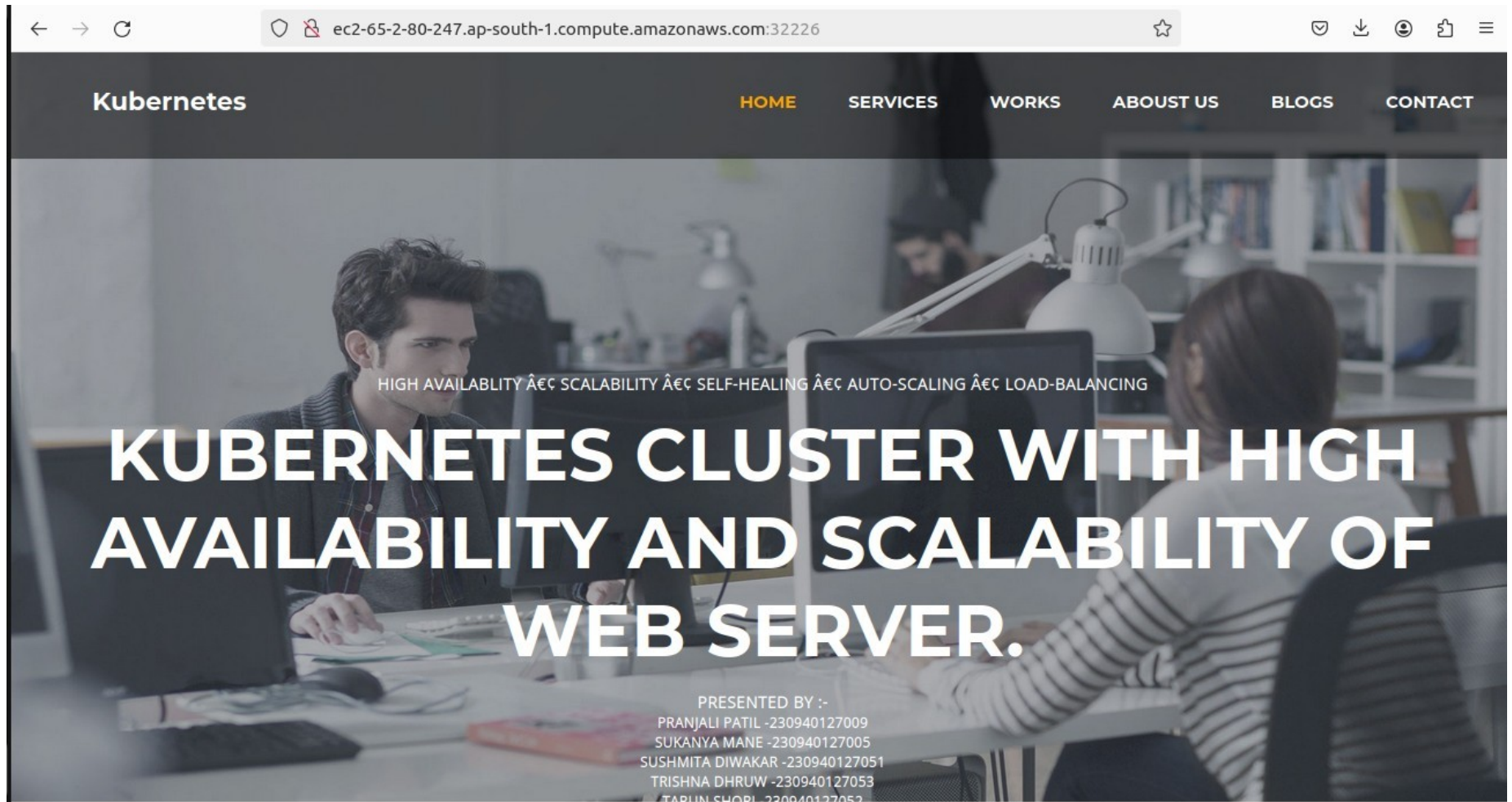
**RollingUpdate:** In this strategy the Deployment gradually replaces old Pods with new ones, ensuring that the application remains available throughout the update process. It allows fine-grained control over the rollout, including parameters such as the maximum number of Pods that can be unavailable and the maximum number of new Pods that can be created at once.



HorizontalPodAutoscaler : (HPA) Automatically scales the number of pods in a replication controller, deployment, or replica set based on observed CPU utilization (or, more recently, custom metrics). It ensures that your application has enough resources to handle incoming traffic efficiently while minimizing costs during periods of low demand.

In this project HorizontalPodAutoscaler continuously monitors the CPU utilization of the pods managed by the my-web Deployment. If the average CPU utilization exceeds 70%, it will scale up the number of replicas (pods) up to a maximum of 10. If the CPU utilization drops below 70%, it will scale down the number of replicas down to a minimum of 3. This dynamic scaling ensures that your application can handle varying levels of traffic efficiently.

# Access Web-Server From Outside The Cluster Using Node Port:



# Kubernetes Dashboard Setup

The screenshot displays the Kubernetes Dashboard interface. The top navigation bar includes the Kubernetes logo, a dropdown menu for 'project-website', a search bar, and icons for adding, notifying, and user management. The breadcrumb trail shows 'Workloads > Deployments > my-web'. The left sidebar lists various Kubernetes resources, with 'Deployments' highlighted under the 'Workloads' section. The main content area is divided into three sections: 'Metadata', 'Resource information', and 'Rolling update strategy'. The 'Metadata' section shows details for the 'my-web' deployment in the 'project-website' namespace, including its creation date, age, and UID. The 'Resource information' section shows the deployment strategy as 'RollingUpdate' with a revision history limit of 2. The 'Rolling update strategy' section is partially visible at the bottom.

Name	Namespace	Created	Age	UID
my-web	project-website	Feb 16, 2024	2 days ago	b2266f97-41af-43bd-a117-7f93d1f7bd16

Strategy	Min ready seconds	Revision history limit
RollingUpdate	0	2

Selector: app: my-web

The Kubernetes Dashboard provides a user-friendly interface for managing and monitoring Kubernetes clusters, making it easier for both novice and experienced users to work with Kubernetes resources effectively.

# Validation

1. Verify High Availability of Pods if worker node is Failed:
2. Verify Scalability of Pods - Horizontal Pod Autoscaling (HPA):
3. verify Manual Scaling:

```
[root@master /]# kubectl get pods -o wide -n=project-website
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
my-web-7f67bfb967-5rhrh	1/1	Running	0	26m	192.168.189.116	worker2	<none>		<none>	
my-web-7f67bfb967-9w95t	1/1	Running	0	26m	192.168.189.107	worker2	<none>		<none>	
my-web-7f67bfb967-kmdgb	1/1	Running	0	26m	192.168.189.112	worker2	<none>		<none>	

```
[root@master /]#  
[root@master /]# kubectl scale deploy my-web --replicas=5 -n=project-website  
deployment.apps/my-web scaled  
[root@master /]#  
[root@master /]# kubectl get pods -o wide -n=project-website
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
my-web-7f67bfb967-5rhrh	1/1	Running	0	26m	192.168.189.116	worker2	<none>		<none>	
my-web-7f67bfb967-9w95t	1/1	Running	0	26m	192.168.189.107	worker2	<none>		<none>	
my-web-7f67bfb967-kmdgb	1/1	Running	0	26m	192.168.189.112	worker2	<none>		<none>	
my-web-7f67bfb967-twrl5	0/1	ContainerCreating	0	3s	<none>	worker1	<none>		<none>	
my-web-7f67bfb967-v6dl8	1/1	Running	0	3s	192.168.235.145	worker1	<none>		<none>	

```
[root@master /]# kubectl get pods -o wide -n=project-website
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
my-web-7f67bfb967-5rhrh	1/1	Running	0	26m	192.168.189.116	worker2	<none>		<none>	
my-web-7f67bfb967-9w95t	1/1	Running	0	26m	192.168.189.107	worker2	<none>		<none>	
my-web-7f67bfb967-kmdgb	1/1	Running	0	26m	192.168.189.112	worker2	<none>		<none>	
my-web-7f67bfb967-twrl5	1/1	Running	0	7s	192.168.235.143	worker1	<none>		<none>	
my-web-7f67bfb967-v6dl8	1/1	Running	0	7s	192.168.235.145	worker1	<none>		<none>	

```
[root@master /]#
```



## References & Bibliography

1. Kubernetes Documentation:

<https://kubernetes.io/docs/home/>

2. Containerd Documentation:

<https://containerd.io/docs/>

3. AWS Cloud Documentation:

<https://docs.aws.amazon.com/>

4. Calico Documentation:

<https://docs.tigera.io/>

Project Link:

Github: <https://github.com/tarun-code/KUBERNETES-CLUSTER-PROJECT>



## Limitation's

1. Vendor lock-in: While Kubernetes itself is open-source and portable, using it on AWS may tie you to AWS-specific services and features, making it harder to migrate to another cloud provider in the future.
2. Service limitations: Some AWS services may not be fully compatible or optimized for Kubernetes. For example, AWS Load Balancers may require additional configuration to work seamlessly with Kubernetes services.
3. Regional availability: Not all AWS services and features are available in every AWS region, which could impact your Kubernetes deployment's flexibility and availability





## Conclusion

In conclusion, deploying a Kubernetes cluster on AWS for high availability, load balancing, and scaling of web server applications is a complex but rewarding endeavor. By addressing challenges, leveraging best practices, and embracing automation and scalability principles, organizations can build robust and resilient Kubernetes environments that meet the demands of modern cloud-native applications.

1. Benefits: Kubernetes provides a powerful platform for automating the deployment, scaling, and management of containerized applications. By leveraging AWS infrastructure, we can take advantage of scalable compute resources, managed services, and global reach to build resilient and efficient Kubernetes clusters.
2. Challenges : Deploying and managing a Kubernetes cluster on AWS involves addressing various challenges, including complexity, cost, resource management, networking considerations, and security concerns. It requires expertise in Kubernetes administration, AWS services, infrastructure management, and DevOps practices.