# KUBERNETES (K8s)

Kubernetes is an open-source container management platform that automates the deployment, management, and scaling of container-based applications in different kinds of environments like physical, virtual, and cloud-native computing foundations.

Containers are isolated from each other so that multiple containers can run on the same machine without interrupting anyone else. It allows us to deploy and manage container-based applications across a Kubernetes cluster of machines.

Before Kubernetes, developers used Docker to package and run their applications inside containers. Docker made creating and running a single container easy, but it became hard to manage many containers running across different machines. For example, what if one container crashes? How do you restart it automatically? Or how do you handle hundreds of containers that need to work together?

That's why Kubernetes was created. It helps manage and organize containers automatically. Kubernetes makes sure your containers keep running, can scale up when there's more traffic, and can move to healthy machines if something goes wrong.

## What is Kubernetes?

**Kubernetes** (also called **K8s**) is an open-source platform that helps you **automates the deployment, scaling, and management of containerized applications**. In simple words, if you're running a lot of apps using containers (like with Docker), Kubernetes helps you organize and control them efficiently just like a traffic controller for your apps.

You tell Kubernetes what your app should look like (how many copies to run, what to do if something fails, etc.), and Kubernetes takes care of the rest making sure everything is up and running properly.

**Example**

Now after discussing about the Kubernetes let's take an example to understand how Kubernetes help us in real-world:

Let's say you created a food delivery app and it runs in a container using Docker. When your app becomes popular and thousands of users start placing orders, you need to run more copies of it to handle the load.

Instead of doing this manually, you use Kubernetes to say:

*"Hey Kubernetes, always keep 5 copies of my app running. If one stops, replace it. And if more users come, increase the number of copies automatically."*

Kubernetes listens to this instruction and makes it happen automatically.

**Key Terminologies**

Think of Kubernetes as a well-organized company where different teams and systems work together to run applications efficiently. Here's how the key terms fit into this system:

**1. Pod**

A Pod is the smallest unit you can deploy in Kubernetes. It wraps one or more containers that need to run together, sharing the same network and storage. Containers inside a Pod can easily communicate and work as a single unit.

**2. Node**

A Node is a machine (physical or virtual) in a Kubernetes cluster that runs your applications. Each Node contains the tools needed to run Pods, including the container runtime (like Docker), the Kubelet (agent), and the Kube proxy (networking).

**3. Cluster**

A Kubernetes cluster is a group of computers (called nodes) that work together to run your containerized applications. These nodes can be real machines or virtual ones.

There are two types of nodes in a Kubernetes cluster:

1. **Master node (Control Plane):**
    - Think of it as the brain of the cluster.
    - It makes decisions, like where to run applications, handles scheduling, and keeps track of everything.
2. **Worker nodes:**
    - These are the machines that actually run your apps inside containers.
    - Each worker node has a Kubelet (agent), a container runtime (like Docker or containerd), and tools for networking and monitoring.

**4. Deployment**

A Deployment is a Kubernetes object used to manage a set of Pods running your containerized applications. It provides declarative updates, meaning you tell Kubernetes what you want, and it figures out how to get there.

**5. ReplicaSet**

A ReplicaSet ensures that the right number of identical Pods are running.

**6. Service**

A Service in Kubernetes is a way to connect applications running inside your cluster. It gives your Pods a stable way to communicate, even if the Pods themselves keep changing.

**7. Ingress**

Ingress is a way to manage external access to your services in a Kubernetes cluster. It provides HTTP and HTTPS routing to your services, acting as a reverse proxy.

**8. ConfigMap**

A ConfigMap stores configuration settings separately from the application, so changes can be made without modifying the actual code.

Imagine you have an application that needs some settings, like a database password or an API key. Instead of hardcoding these settings into your app, you store them in a ConfigMap. Your application can then read these settings from the ConfigMap at runtime, which makes it easy to update the settings without changing the app code.

**9. Secret**

A Secret is a way to store sensitive information (like passwords, API keys, or tokens) securely in a Kubernetes cluster.

**10. Persistent Volume (PV)**

A Persistent Volume (PV) in Kubernetes is a piece of storage in the cluster that you can use to store data — and it doesn't get deleted when a Pod is removed or restarted.

**11. Namespace**

A Namespace is like a separate environment within your Kubernetes cluster. It helps you organize and isolate your resources like Pods, Services, and Deployments.

**12. Kubelet**

A **Kubelet** runs on each Worker Node and ensures Pods are running as expected.

**13. Kube-proxy**

**Kube-proxy** manages networking inside the cluster, ensuring different Pods can communicate.

## Benefits of using Kubernetes

**1. Automated Deployment and Management**

- If you are using Kubernetes for deploying the application then no need for manual intervention kubernetes will take care of everything like automating the deployment, scaling, and containerizing the application.
- Kubernetes will reduce the errors that can be made by humans which makes the deployment more effective.

**2. Scalability**

- You can scale the application containers depending on the incoming traffic Kubernetes offers Horizontal pod scaling the pods will be scaled automatically depending on the load.

**3. High Availability**

- You can achieve high availability for your application with the help of Kubernetes and also it will reduce the latency issues for the end users.

**4. Cost-Effectiveness**

- If there is unnecessary use of infrastructure the cost will also increase kubernetes will help you to reduce resource utilization and control the overprovisioning of infrastructure.

**5. Improved Developer Productivity**

- Developers can focus on writing code rather than managing deployments, as Kubernetes automates much of the deployment and scaling process

## Key Components of Kubernetes

### 1. Kubernetes-Master Node Components

Kubernetes master is responsible for managing the entire cluster, coordinates all activities inside the cluster, and communicates with the worker nodes to keep the Kubernetes and your application running. This is the entry point of all administrative tasks. When we install

Kubernetes on our system we have four primary components of Kubernetes Master that will get installed. The components of the Kubernetes Master node are:

***API Server***

The API server is the entry point for all the REST commands used to control the cluster. All the administrative tasks are done by the API server within the master node. If we want to create, delete, update or display in Kubernetes object it has to go through this API server.API server validates and configures the API objects such as ports, services, replication, controllers, and deployments and it is responsible for exposing APIs for every operation. We can interact with these APIs using a tool called **kubectl**. *'kubectl' is a very tiny go language binary that basically talks to the API server to perform any operations that we issue from the command line. It is a command-line interface for running commands against Kubernetes clusters*

***Scheduler***

It is a service in the master responsible for distributing the workload. It is responsible for tracking the utilization of the working load of each worker node and then placing the workload on which resources are available and can accept the workload. The scheduler is responsible for scheduling pods across available nodes depending on the constraints you mention in the configuration file it schedules these pods accordingly. The scheduler is responsible for workload utilization and allocating the pod to the new node.

***Controller Manager***

Also known as controllers. It is a daemon that runs in a non terminating loop and is responsible for collecting and sending information to the API server. It regulates the Kubernetes cluster by performing lifestyle functions such as namespace creation and lifecycle event garbage collections, terminated pod garbage collection, cascading deleted garbage collection, node garbage collection, and many more. Basically, the controller watches the desired state of the cluster if the current state of the cluster does not meet the desired state then the control loop takes the corrective steps to make sure that the current state is the same as that of the desired state. The key controllers are the replication controller, endpoint controller, namespace controller, and service account, controller. So in this way controllers are responsible for the overall health of the entire cluster by ensuring that nodes are up and running all the time and correct pods are running as mentioned in the specs file.

***etcd***

It is a distributed key-value lightweight database. In Kubernetes, it is a central database for storing the current cluster state at any point in time and is also used to store the configuration details such as subnets, config maps, etc. It is written in the Go programming language.

**2. Kubernetes-Worker Node Components**

Kubernetes Worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled. The components of the Kubernetes Worker node are:

***Kubelet***

It is a primary node agent which communicates with the master node and executes on each worker node inside the cluster. It gets the pod specifications through the API server and executes the container associated with the pods and ensures that the containers described in the pods are running and healthy. If kubelet notices any issues with the pods running on the worker nodes then it tries to restart the pod on the same node. If the issue is with the worker node itself then the Kubernetes master node detects the node failure and decides to recreate the pods on the other healthy node.

***Kube-Proxy***

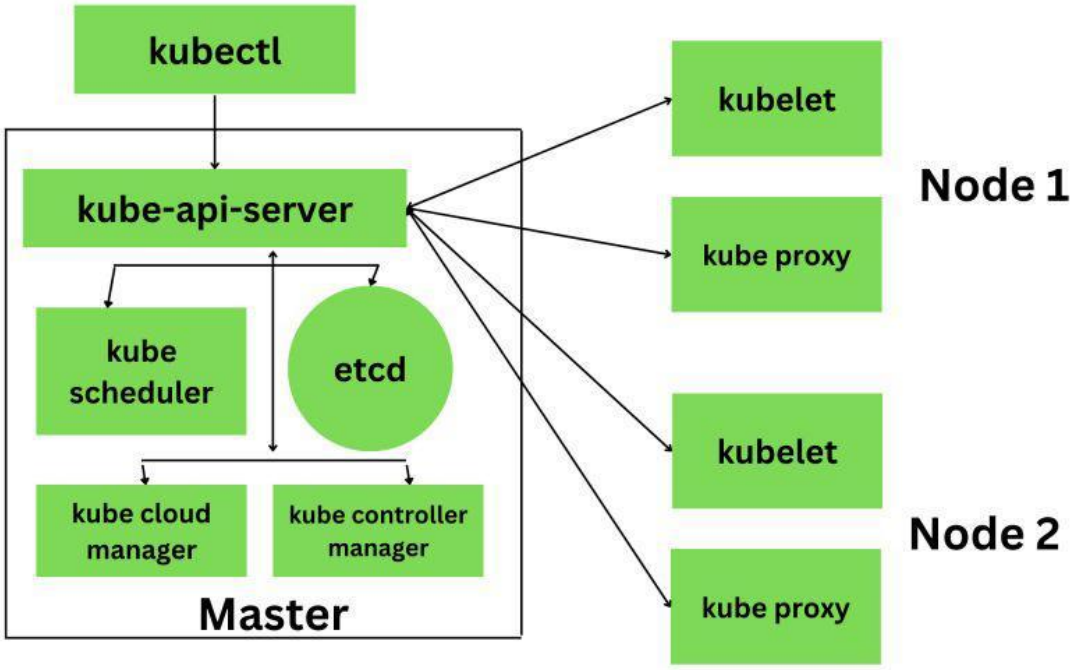
It is the core networking component inside the Kubernetes cluster. It is responsible for maintaining the entire network configuration. Kube-Proxy maintains the distributed network across all the nodes, pods, and containers and exposes the services across the outside world. It acts as a network proxy and load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets. It listens to the API server for each service endpoint creation and deletion so for each service endpoint it sets up the route so that you can reach it.

***Pods***

A pod is a group of containers that are deployed together on the same host. With the help of pods, we can deploy multiple dependent containers together so it acts as a wrapper around these containers so we can interact and manage these containers primarily through pods.

***Docker***

Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be development or test or production. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Docker is the world's leading software container platform. It was launched in 2013 by a company called Dot cloud. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VMs. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of systems. Docker comes into play at the deployment stage of the software development cycle.




**Application of Kubernetes**

The following are some application of Kubernetes:

1. **Microservices Architecture**

Kubernetes is well-suited for managing microservices architectures, which involve breaking down complex applications into smaller, modular components that can be independently deployed and managed.

2. **Cloud-native Development**

Kubernetes is a key component of cloud-native development, which involves building applications that are designed to run on cloud infrastructure and take advantage of the scalability, flexibility, and resilience of the cloud.

3. **Continuous Integration and Delivery**

Kubernetes integrates well with CI/CD pipelines, making it easier to automate the deployment process and roll out new versions of your application with minimal downtime.

4. **Hybrid and Multi-Cloud Deployments**

Kubernetes provides a consistent deployment and management experience across different cloud providers, on-premise data centers, and even developer laptops, making it easier to build and manage hybrid and multi-cloud deployments.

5. **High-Performance Computing**

Kubernetes can be used to manage high-performance computing workloads, such as scientific simulations, machine learning, and big data processing.

6. **Edge Computing**

Kubernetes is also being used in edge computing applications, where it can be used to manage containerized applications running on edge devices such as IoT devices or network appliances.

## Installation Methods of Kubernetes

There are several methods to install Kubernetes based on your needs and resources. **Play-with-K8s** offers a ready-made online Kubernetes cluster for quick learning or testing without any installation. **Minikube** is ideal for limited system resources, packaging all Kubernetes components into a single setup that acts as both master and worker nodes. **Kubeadm** is perfect for real-time setups, enabling the creation of multi-node Kubernetes Cluster. It is a popular choice for configuring master and node components, with cloud-based VMs recommended for systems with limited resources.

## Kubernetes Minikube:

Minikube is a one-node Kubernetes cluster where master processes and work processes both run on one node. According to the official documentation of Minikube, Minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes. A prerequisite to use Minikube is a Docker container or a Virtual Machine environment. Minikube provides you a way to easily deploy application on Kubernetes for learning and developing purposes even if you don't have enough resources like memory, CPU etc.

## What Does Minikube Do?

Minikube is a one node Kubernetes cluster that runs in VirtualBox on your local machine in order to test Kubernetes on your local set up. In order to setup a production cluster, we need multiple master notes and multiple worker nodes. Master nodes and worker nodes have their own separate responsibilities. In order to test something on our local environment for example - deploying a new application or a new component and to test it on our local machine, it will be very difficult and would consume a lot of resources like CPU, memory etc.

Minikube comes into play here. Minikube is an open-source tool that is basically like a one node cluster where the master processes and the work processes both run on one node. This node must have a Docker container runtime pre-installed to run the containers or pods with containers on this node.

## Kubernetes - Kubectl

Kubectl is a command-line software tool that is used to run commands in command-line mode against the Kubernetes Cluster. Kubectl runs the commands in the command line, Behind the scenes it authenticates with the master node of Kubernetes Cluster with API

calls and performs the operations on Kubernetes Resources such as Creation, Deletion, Modification, and Viewing Operations. Here the Resources include such as Pod, Service, Deployment, Secrets, ReplicationController, ReplicationSet, Persistent Volumes, etc.

## What is kubectl?

Kubectl is command-line software that helps to run commands in the Kubernetes Cluster. It acts as a bridge between clients and the Kubernetes cluster, providing communication through API requests using http or https protocols. It converts the commands entered in the command line from a user into API requests and pass to the API server of the Kubernetes master Node with proper Authentication. It helps to manage the containers and deploy apps and other resources to run smoothly.

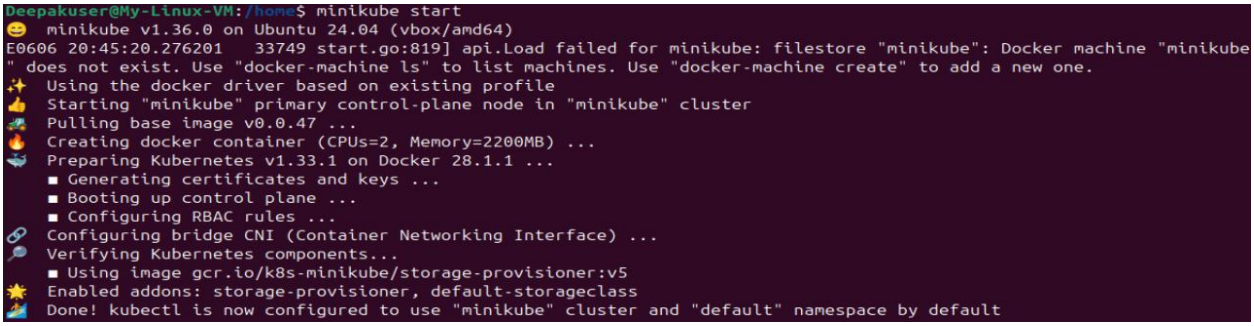
## Deploying a Simple Service Using Minikube and Kubectl

Kubectl is the Kubernetes CLI tool. Follow these steps to deploy a service using Minikube and Kubectl:

**Step 1.** Enter the following command to start the Kubernetes cluster:

**minikube start**

you will get a similar output, wait for some time until the installation of dependencies gets completed:




**Step 2.** Now if we check the number of nodes running, we will get the minikube node running.

**kubectl get nodes**

this will give you the following output:


or you can also check the status by running the following command:

**minikube status**

you will get a similar output on the terminal:

```
Deepakuser@My-Linux-VM:/home$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

**Step 2.** To find out the services that are currently running on Kubernetes, enter the following command:

**kubectl get services**

you will only see the default Kubernetes service running.

```
Deepakuser@My-Linux-VM:/home$ kubectl get services
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP   73m
```

**Step 3.** Enter the following command on your terminal to create a deployment

**kubectl create deployment nginxpod --image=nginx**

**Step 4.** Enter the following command to expose the deployment to port 6000:

**kubectl expose deployment nginxpod --type=NodePort --port=80**

**Step 5.** Now when you check out the list of services

**kubectl get services**

you will find the *nginxpod* service:

```
Deepakuser@My-Linux-VM:~$ kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP          20d
mysql-service       ClusterIP   10.97.255.176    <none>        3306/TCP         2d17h
nginxpod            NodePort    10.96.93.134     <none>        80:31034/TCP     75m
wordpress-service   NodePort    10.97.210.104    <none>        80:30007/TCP     2d16h
```

**Step 6.** Now to run the service, enter the following minikube command:

**minikube service nginxpod**

It will display the following result:

```
Deepakuser@My-Linux-VM:~/Desktop$ minikube service nginxpod
|-----------|-----------|-------------|---------------------------|
| NAMESPACE |   NAME    | TARGET PORT |            URL            |
|-----------|-----------|-------------|---------------------------|
| default   | nginxpod |          80 | http://192.168.49.2:31034 |
|-----------|-----------|-------------|---------------------------|
  Opening service default/nginxpod in default browser...
```

You have to just copy the address (*http://192.168.49.2:31034 for me*) and paste it to your browser to see the application website. It looks like the following:


## Installing WordPress Application on Kubernetes Cluster using yaml files

WordPress is a popular Content Management System (CMS) that allows you to create and manage websites easily. Kubernetes is a powerful container orchestration platform that can help you deploy and manage WordPress with high availability and scalability.

For deploying the WordPress application on Kubernetes Cluster I have created several files such as MySQL StatefulSet for database storage, deployment files, secret files, Persistent Volume (PV), PersistentVolumeClaim (PVC), and service files.

```
Deepakuser@My-Linux-VM:~/Desktop$ ls
Dockerfiles  mysql-pv-pvc.yaml  mysql-service.yaml      wordpress_deployment.yaml
kubectl      mysql-secret.yaml  mysql-statefulset.yaml  wordpress-pv-pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
   name: mysql-pv
spec:
   capacity:
     storage: 5Gi
   accessModes:
     - ReadWriteOnce
   hostPath:
     path: /mnt/data/mysql

---

apiVersion: v1



kind: PersistentVolumeClaim


metadata:
   name: mysql-pvc
spec:
   accessModes:
     - ReadWriteOnce
   resources:
      requests:
        storage: 5Gi
   selector:
     matchLabels:
```

```
Deepakuser@My-Linux-VM:~/Desktop$ cat mysql-secret.yaml


apiVersion: v1
kind: Secret
metadata:

   name: mysql-secret
type: Opaque
data:
   MYSQL_ROOT_PASSWORD: bXlwYXNzd29yZAo=
```

```
Deepakuser@My-Linux-VM:~/Desktop$ cat mysql-statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  replicas: 1
  serviceName: mysql
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: database
        image: mysql:5.7
        args:
          - "--ignore-db-dir=lost+found"
        envFrom:
          - secretRef:
              name: mysql-secret
        ports:
        - containerPort: 3306
        volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
      volumes:
        - name: mysql-data
```

You can see that I have created the yaml files, and above are some of the screenshots of the content of those yaml files.

**After creating all those files apply the configurations to your Kubernetes cluster using the following commands:**

**kubectl apply -f mysql-secret.yaml**
**kubectl apply -f mysql-pv-pvc.yaml**
**kubectl apply -f mysql-statefulset.yaml**
**kubectl apply -f mysql-service.yaml**
**kubectl apply -f wordpress-pv-pvc.yaml**
**kubectl apply -f wordpress-deployment.yaml**

You can apply all your configuration files at a time as well by calling your directory like:

**kubectl apply -f <directory_name>**

**Now you can check for the pods and services that have been created in the cluster using:**

**Kubectl get pods**

```
Deepakuser@My-Linux-VM:~$ kubectl get pods
NAME                         READY   STATUS    RESTARTS      AGE
mysql-0                      1/1     Running   5 (92m ago)   2d16h
nginxpod-64c6d8485d-9k6pp    1/1     Running   1 (92m ago)   132m
wordpress-587d7bff58-528p7   1/1     Running   4 (92m ago)   2d15h
wordpress-587d7bff58-qzfdw   1/1     Running   4 (92m ago)   2d15h
```

In above image the pods are highlighted which got created in the cluster.

**You can check for deployments using the command:**

**Kubectl get deployments**

```
Deepakuser@My-Linux-VM:~$ kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
nginxpod    1/1     1            1           133m
wordpress   2/2     2            2           2d15h
```

**You can check for services created in the cluster using the command:**

**Kubectl get services**

```
Deepakuser@My-Linux-VM:~$ kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP          20d
mysql-service       ClusterIP   10.97.255.176    <none>        3306/TCP         2d17h
nginxpod            NodePort    10.96.93.134     <none>        80:31034/TCP     75m
wordpress-service   NodePort    10.97.210.104    <none>        80:30007/TCP     2d16h
```

In above image you can see for the wordpress-service is listening on port 30007.

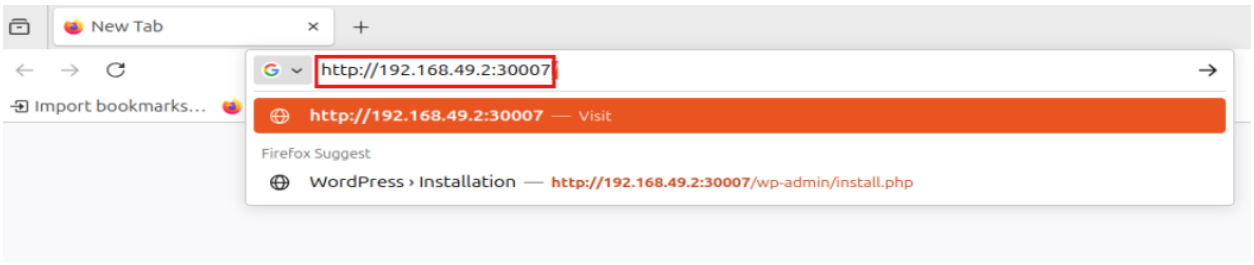
**You can check for the ip address of your node using command:**

**Kubectl get node -o wide**

```
Deepakuser@My-Linux-VM:~$ kubectl get no -o wide
NAME       STATUS   ROLES           AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION
  CONTAINER-RUNTIME
minikube   Ready    control-plane   20d   v1.33.1   192.168.49.2   <none>        Ubuntu 22.04.5 LTS   6.11.0-26-generic
  docker://28.1.1
```

You can now access the application by going to the browser and search for [http://192.168.49.2:3007](http://192.168.49.2:3007)

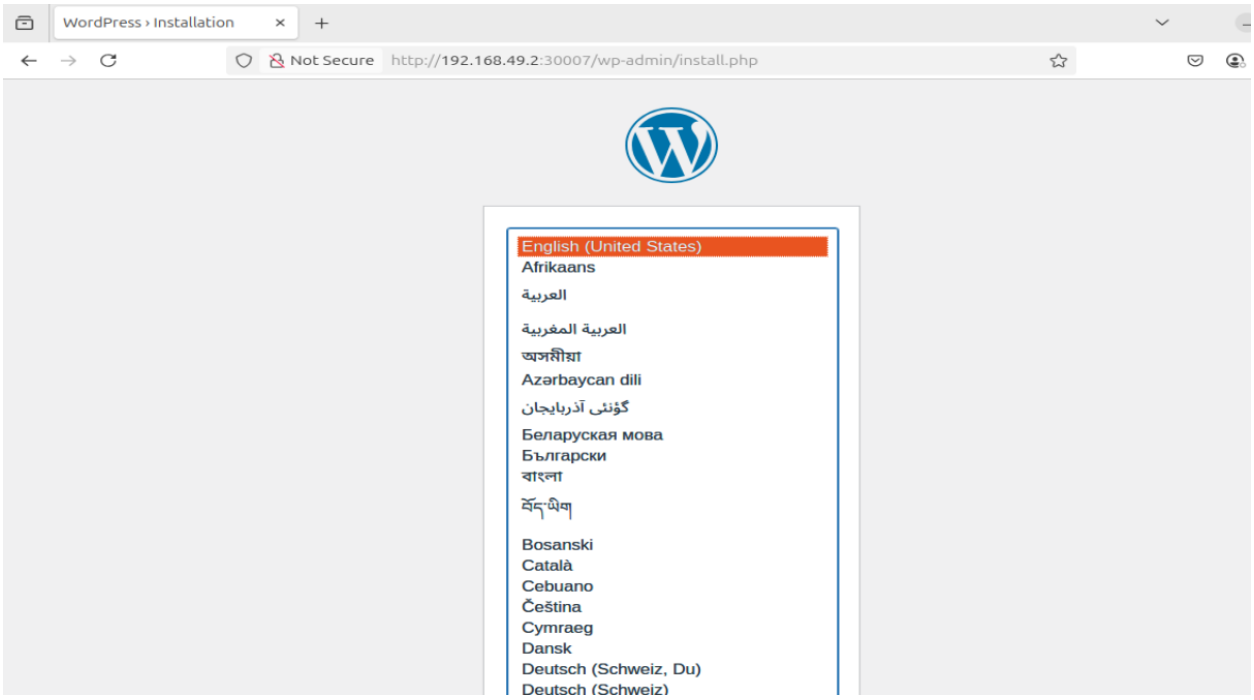
Now you can access the application.




**OR**

To access your WordPress site, obtain the external IP address of the WordPress Service with the following command:

**kubectl get svc wordpress-service**




# Some common Minikube command

**1. Deleting The Minikube Cluster**

You just have to simply enter the following command to delete the minikube cluster.

**minikube delete**

**2. To Pause Kubernetes Without Impacting Deployed Applications.**

**minikube pause**

**3. To Unpause The Instance**

**minikube unpause**

**4. To Stop The Cluster**

**minikube stop**

**5. To Delete All Minikube Clusters**

**minikube delete –all**

## How to Install Minikube on Ubuntu

Follow the steps below to install Minikube on a local Ubuntu system. The steps show how to set up the necessary dependencies, download the Minikube binary, and enable its execution.

### Step 1: Install Required Packages

Minikube installation requires the curl command to download the Minikube installer and the **apt-transport-https** package to handle HTTPS connections during package installation.

Proceed with the steps below to install the packages:

1. Update the package information on the system by entering the following command:

**sudo apt update**

2. Install **curl** and **apt-transport-https**:

**sudo apt install curl apt-transport-https**

```
marko@phoenixnap:~$ sudo apt install curl apt-transport-https
[sudo] password for marko:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  apt-transport-https curl
0 upgraded, 2 newly installed, 0 to remove and 3 not upgraded.
```

### Step 2: Download Minikube Binary

Minikube binaries for all major systems and architectures are available on the Minikube website. The following procedure downloads the binary for X86-64 Ubuntu systems:

1. Use **curl** to download the latest Minikube binary:

**curl -O https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64**

```
marko@phoenixnap:~$ curl -O https://storage.googleapis.com/minikube/releases/la
test/minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  119M  100  119M    0     0  8849k      0  0:00:13  0:00:13 --:--:-- 10.4M
marko@phoenixnap:~$
```

2. Copy the downloaded file and store it in the */usr/local/bin/* directory:

**sudo cp minikube-linux-amd64 /usr/local/bin/minikube**

The command prints no output.

**Step 3: Enable Minikube Binary Execution**

By default, binaries do not have the permission to execute. Give the execute permission to the Minikube binary using the **chmod command**:

**sudo chmod 755 /usr/local/bin/minikube**

Verify the installation by checking the Minikube version:

**minikube version**

The output displays the software version number.

```
marko@phoenixnap:~$ minikube version
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
marko@phoenixnap:~$
```

**Step 4: Install kubectl**

**Kubectl** is the official command line tool for Kubernetes, which provides an interface for cluster deployment and management. Ubuntu has a kubectl **snap package** that can be installed by typing:

**sudo snap install kubectl --classic**

```
marko@phoenixnap:~$ sudo snap install kubectl --classic
kubectl 1.31.5 from Canonical✓ installed
marko@phoenixnap:~$
```

Alternatively, follow the steps below to download and install the kubectl binary directly from the maintainer:

1. Download kubectl with the following command:

**curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"**

2. Make the binary executable by typing:

**chmod +x ./kubectl**

3. Move the binary to the path with the following command:

**sudo mv ./kubectl /usr/local/bin/kubectl**

**Step 5: Start Minikube**

Start Minikube by running the command below:

**minikube start**

Minikube selects the default driver and begins pulling the latest available Kubernetes image.

```
marko@phoenixnap:~$ minikube start
😄  minikube v1.35.0 on Ubuntu 24.04 (vbox/amd64)
✨  Automatically selected the docker driver
📌  Using Docker driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.46 ...
💾  Downloading Kubernetes v1.32.0 preload ...
```

**Note**: To start Minikube with a non-default driver, use the following command: **minikube start --driver=[driver_name]**.

Once Minikube finishes initialization, the output shows the confirmation message, and the shell prompt reappears.

```
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" na
mespace by default
marko@phoenixnap:~$
```

# How to Install Minikube on Windows

Follow these steps to download Minikube via CLI, you can also check out Minikube's official website to download the .exe version.

**Step 1.** Enter the following command in your Windows PowerShell. Make sure to use PowerShell as an administrator.

```
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri
'https://github.com/kubernetes/minikube/
releases/latest/download/minikube-windows-amd64.exe' -UseBasicParsing
```

**Step 2.** Now enter the following command to add the minikube.exe binary to your **PATH:**

```
$oldPath = [Environment]::GetEnvironmentVariable('Path',
[EnvironmentVariableTarget]::Machine)
if ($oldPath.Split(';') -inotcontains 'C:\minikube'){
  [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath),
[EnvironmentVariableTarget]::Machine)
}
```

**Step 3.** Restart your terminal and enter 'minikube' command to check if Minikube is installed.

```
minikube
```

This will give you a similar output if Minikube is installed. If you face any other issues, checkout the official website of Minikube for resolving it.

```
PS C:\WINDOWS\system32> minikube
minikube provisions and manages local Kubernetes clusters optimized for development workflows.

Basic Commands:
  start            Starts a local Kubernetes cluster
  status           Gets the status of a local Kubernetes cluster
  stop             Stops a running local Kubernetes cluster
  delete           Deletes a local Kubernetes cluster
  dashboard        Access the Kubernetes dashboard running within the minikube cluster
  pause            pause Kubernetes
  unpause          unpause Kubernetes

Images Commands:
  docker-env       Provides instructions to point your terminal's docker-cli to the Docker Engine inside minikube.
(Useful for building docker images directly inside minikube)
  podman-env       Configure environment to use minikube's Podman service
  cache            Manage cache for images
  image            Manage images

Configuration and Management Commands:
  addons           Enable or disable a minikube addon
  config           Modify persistent configuration values
  profile          Get or list the current profiles (clusters)
  update-context   Update kubeconfig in case of an IP or port change

Networking and Connectivity Commands:
  service          Returns a URL to connect to a service
  tunnel           Connect to LoadBalancer services

Advanced Commands:
  mount            Mounts the specified directory into minikube
  ssh              Log into the minikube environment (for debugging)
  kubectl          Run a kubectl binary matching the cluster version
  node             Add, remove, or list additional nodes
  cp               Copy the specified file into minikube

Troubleshooting Commands:
  ssh-key          Retrieve the ssh identity key path of the specified node
  ssh-host         Retrieve the ssh host key of the specified node
  ip               Retrieves the IP address of the specified node
  logs             Returns logs to debug a local Kubernetes cluster
  update-check     Print current and latest version number
  version          Print the version of minikube
  options          Show a list of global command-line options (applies to all commands).

Other Commands:
  completion       Generate command completion for a shell
  license          Outputs the licenses of dependencies to a directory

Use "minikube <command> --help" for more information about a given command.
PS C:\WINDOWS\system32>
```