Deploying WordPress and MySQL with Persistent Volumes Using K8s

t.me/vishalk17 github.com/vishalk17

Let us understand, what is wordpress:

WordPress is a versatile and widely-used content management system (CMS) that allows users to create and manage websites, blogs, and online stores. Here are some of the main uses and benefits of WordPress:

1. Website Creation: WordPress provides an intuitive interface and a wide range of themes and templates that enable users to create professional-looking websites without any coding knowledge.

2. Blogging: Originally developed as a blogging platform, WordPress remains a popular choice for bloggers. It offers robust features for managing and publishing blog content, including categories, tags, commenting systems, and RSS feeds.

3. E-commerce: With the help of plugins like WooCommerce, WordPress can be transformed into a full-fledged e-commerce platform. It allows users to set up online stores, manage products, process payments, and handle inventory.

4. Customization: WordPress offers a vast library of themes and plugins that allow users to customize the appearance and functionality of their websites. Themes control the layout and design, while plugins add specific features and functionality.

5. SEO-Friendly: WordPress is highly optimized for search engines, making it easier for websites to rank well in search engine results. It provides various SEO tools and plugins to enhance website visibility and improve search engine rankings.

6. User Management: WordPress allows multiple users to collaborate on a website, each with their own roles and permissions. This feature is especially useful for businesses, organizations, and multi-author blogs.

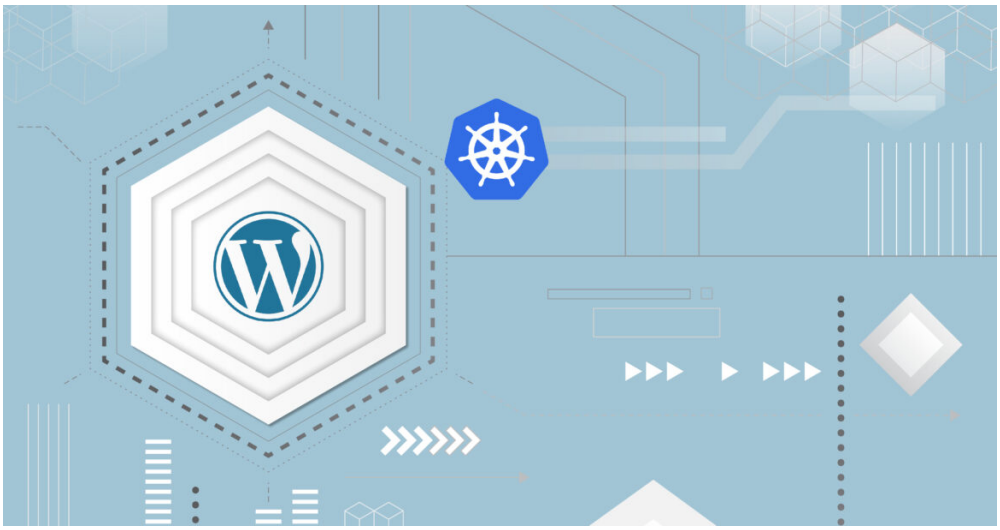
7. Mobile Responsiveness: With the increasing use of mobile devices, having a mobile-friendly website is crucial. WordPress offers responsive themes that automatically adjust the layout and design to fit different screen sizes and devices.

8. Community Support: WordPress has a large and active community of developers, designers, and users who contribute to its continuous improvement. This community provides support, resources, and updates, ensuring that WordPress remains a reliable and secure platform.

Overall, WordPress is a versatile CMS that can be used for various purposes, including website creation, blogging, e-commerce, and more. Its user-friendly interface, customization options,

and extensive plugin ecosystem make it a popular choice for individuals, businesses, and organizations of all sizes.

Relation between wordpress and k8s:



WordPress is a popular open-source content management system (CMS) that allows users to create and manage websites and blogs. It provides a user-friendly interface and a wide range of plugins and themes to customize the website's appearance and functionality.

Kubernetes, on the other hand, is an open-source container orchestration platform. It automates the deployment, scaling, and management of containerized applications. Kubernetes provides features such as load balancing, automatic scaling, self-healing, and rolling updates, making it easier to manage and scale applications.

The relation between WordPress and Kubernetes comes into play when deploying and managing WordPress applications at scale. Kubernetes can be used to deploy and manage WordPress instances as containers, allowing for easy scaling, high availability, and resilience. Kubernetes provides features like horizontal pod autoscaling, rolling updates, and service discovery, which can be leveraged to manage WordPress deployments effectively.

By using Kubernetes, you can ensure that your WordPress application is highly available, scalable, and resilient, handling traffic spikes and providing a seamless experience to your users. Kubernetes allows you to manage the underlying infrastructure and resources efficiently, making it easier to deploy and manage WordPress in a production environment.

Deploying WordPress and MySQL with Persistent Volumes Using K8s

t.me/vishalk17 github.com/vishalk17

In summary, Kubernetes can be used to deploy, scale, and manage WordPress applications, ensuring high availability and scalability. It provides a platform to run WordPress containers and offers various features to simplify the management of WordPress deployments.

Requirements :

- Wordpress image
- Mysql database : for wordpress data to store
    - Password and username req.
- Service : nodeport for accessing anywhere
- secrete (for mysql credential)
- Persist volumes for both wordpress and mysql

Create PersistentVolumeClaims and PersistentVolumes

- MySQL and Wordpress each require a PersistentVolume to store data. Their PersistentVolumeClaims will be created at the deployment step.
- Many cluster environments have a default StorageClass installed. When a StorageClass is not specified in the PersistentVolumeClaim, the cluster's default StorageClass is used instead.
- When a PersistentVolumeClaim is created, a PersistentVolume is dynamically provisioned based on the StorageClass configuration.

vi mysql-pv.yml .... For mysql

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    application: wordpress
    mode: mysql
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```

- `apiVersion` specifies the version of the Kubernetes API being used.
- `kind` defines the type of resource, in this case, a PersistentVolumeClaim.
- `metadata` contains metadata about the PVC, including its name and labels for identification.
- `spec` defines the specification for the PVC, including access modes and resource requests.
- `accessModes` specify how the volume can be accessed. In this case, it is set to `ReadWriteOnce`, meaning the volume can be mounted as read-write by a single node.
- `resources` define the resource requirements for the PVC.
- `requests` specify the minimum amount of storage required for the PVC. In this case, it is set to 4Gi (4 gigabytes).

This PVC is named `mysql-pv-claim` and has labels `application: wordpress` and `mode: mysql`. It requests a minimum of 4 gigabytes of storage with read-write access from a single node.

vi wordpress-pv.yml …. For wordpress

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
        name: wp-pv-claim
        labels:
                application: wordpress
                mode: wordpress-app
spec:
        accessModes:
                - ReadWriteOnce
        resources:
                requests:
                        storage: 4Gi
```

This PVC is named `wp-pv-claim` and has labels `application: wordpress` and `mode: wordpress-app`. It requests a minimum of 4 gigabytes of storage with read-write access from a single node.

Service object:

Here are some of the uses of service objects in Kubernetes:

- **Abstraction:** Services provide an abstraction for a group of pods, which makes it easier to manage and scale applications.
- **Load balancing:** Services can be used to load balance traffic across pods, which improves the performance of applications.
- **External access:** Services can be exposed to the outside world, which makes it possible to access applications from outside the cluster.
- **Health checking:** Services can be used to health check pods, which helps to ensure that applications are available.
- **Name resolution:** Services can be used to resolve names to IP addresses, which makes it easier to access applications.

Here are some examples of how service objects can be used in Kubernetes:

- A web application might be deployed as a set of pods. A service object can be used to expose the pods to the outside world, and to load balance traffic across them.
- A database might be deployed as a set of pods. A service object can be used to access the database from other pods in the cluster.
- A microservice architecture might be implemented using a set of services. Services can be used to communicate with each other, and to load balance traffic across them.

vi mysql-pv.yml …. For mysql

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    application: wordpress
    mode: mysql
  clusterIP: None
```

- The name of the Service: `wordpress-mysql`

- The labels for the Service: `app=wordpress`
- The port that the Service exposes: `3306`
- The selector for the Service: `application=wordpress` and `mode=mysql`
- The type of Service: `ClusterIP`

The `ClusterIP` type of Service exposes the Service on an internal IP address in the cluster. This means that the Service can only be accessed from within the Kubernetes cluster.

The `selector` field specifies the labels that the pods must have in order to be part of the Service. In this case, the pods must have the labels `application=wordpress` and `mode=mysql`.

The `ports` field specifies the ports that the Service exposes. In this case, the Service exposes port 3306.

vi wordpress-service.yml …. For wordpress to expose outside world

```
kind: Service                   #    --> Defines to create service type object
apiVersion: v1
metadata:
  name: wordpress-service
spec:
  ports:
    - port: 80                  #    --> Containers port exposed
      #targetPort: 80           #--> Pods port
  selector:
    application: wordpress              #--> Apply this service to any pods which has the
specific label
    mode: wordpress-app
  type: LoadBalancer             #  --> Specifies the service type i.e. ClusterIP or NodePort
```

- The `kind` field specifies that the resource is a Service.
- The `apiVersion` field specifies the version of the Kubernetes API that the Service is using.
- The `metadata` field specifies the name and labels for the Service.
- The `spec` field specifies the ports, selector, and type for the Service.

The `ports` field specifies the ports that the Service exposes. In this case, the Service exposes port 80.

The `selector` field specifies the labels that the pods must have in order to be part of the Service. In this case, the pods must have the labels `application=wordpress` and `mode=wordpress-app`.

The `type` field specifies the type of Service. In this case, the Service is of type `LoadBalancer`. This means that the Service will be exposed on a public IP address.

Deployment object:

Deployment object is a higher-level abstraction that manages the creation and scaling of ReplicaSets. It is a declarative way to define and manage the desired state of a set of identical Pods.

key properties of a deployment object:

- Name: The name of the deployment.
- Labels: A set of labels that can be used to identify the deployment.
- Replicas: The number of pods that the deployment should create.
- Pod template: A template that defines the pod that should be created by the deployment.
- Strategy: The strategy that the deployment should use to update pods.
- Rollback: The strategy that the deployment should use to roll back to a previous version of the application.

The key components of a Deployment object in Kubernetes are:

- `apiVersion`: Specifies the version of the Kubernetes API being used.
- `kind`: Defines the type of resource, which is "Deployment" in this case.
- `metadata`: Contains metadata about the Deployment, including its name, labels, and annotations for identification and categorization.
- `spec`: Defines the specification for the Deployment, including its selector, replicas, template, and update strategy.
    - `selector`: Specifies the labels used to select the Pods that the Deployment manages.
    - `replicas`: Defines the desired number of Pod replicas to be created and maintained by the Deployment.
    - `template`: Specifies the template for creating the Pods. It includes the Pod's metadata, labels, and container specifications.
    - `strategy`: Defines the update strategy for the Deployment, including options for rolling updates and recreating Pods.

Deploying WordPress and MySQL with Persistent Volumes Using K8s

t.me/vishalk17 github.com/vishalk17

Deployments enable you to define and manage the lifecycle of your application by automatically creating and scaling Pods based on the desired state defined in the Deployment object. They also handle rolling updates, allowing you to update your application without downtime by gradually replacing the old Pods with the updated ones.

vi deploymysql.yml …. For mysql

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    application: wordpress
    mode: mysql
spec:
  selector:
    matchLabels:
      application: wordpress
      mode: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        application: wordpress
        mode: mysql
    spec:
      containers:
      - image: mysql:8.0
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        - name: MYSQL_DATABASE
          value: wordpress
        - name: MYSQL_USER
          value: wordpress
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

- The apiVersion field specifies the version of the Kubernetes API being used, in this case, apps/v1.
- The kind field defines the type of resource, which is "Deployment" in this case.
- The metadata field contains metadata about the Deployment, including its name and labels for identification.
- The spec field defines the specification for the Deployment, including its selector, strategy, and template for creating Pods.
  - The selector field specifies the labels used to select the Pods that the Deployment manages.
  - The strategy field defines the update strategy for the Deployment. In this case, it is set to Recreate, which means that all Pods will be replaced when an update is made.
  - The template field specifies the template for creating the Pods.
  - The metadata field contains labels for identification.
  - The spec field defines the specification for the Pod, including its containers, environment variables, ports, and volume mounts.
    - The containers field specifies the containers to run in the Pod. In this case, there is one container named "mysql" using the mysql:8.0 image.
    - The env field defines the environment variables for the container, including the MySQL root password, database name, user, and password. Some values are retrieved from a secret named "mysql-pass".
    - The ports field defines the container port to expose for MySQL, which is set to 3306.
    - The volumeMounts field specifies the volume mount for persistent storage, which will be mounted at /var/lib/mysql.
  - The volumes field defines the volumes to be used by the Pod. In this case, there is one volume named "mysql-persistent-storage" referencing a PersistentVolumeClaim named "mysql-pv-claim".

This Deployment configuration creates and manages Pods running MySQL with the specified environment variables, ports, and volume mounts. The Deployment ensures that the desired number of replicas is maintained and handles updates using the Recreate strategy.

Deploying WordPress and MySQL with Persistent Volumes Using K8s

t.me/vishalk17 github.com/vishalk17

---

vi deploy-wordpress.yml …. For wordpress

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: deployment-wordpress
spec:
  selector:
    matchLabels:
      application: wordpress
      mode: wordpress-app
  strategy:
    type: Recreate
  template:
    metadata:
      name: web-vishalk17-pod
      labels:
        application: wordpress
        mode: wordpress-app
    spec:
      containers:
        - name: wordpress
          image: wordpress:6.2.1-apache
          env:
            - name: WORDPRESS_DB_HOST
              value: wordpress-mysql
            - name: WORDPRESS_DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
            - name: WORDPRESS_DB_USER
              value: wordpress
          ports:
            - containerPort: 80
              name: wordpress
          volumeMounts:
            - name: wordpress-persistent-storage
              mountPath: /var/www/html
      volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
            claimName: wp-pv-claim
```

- **kind:** defines the type of resource, which is "Deployment" in this case.
- **apiVersion:** specifies the version of the Kubernetes API being used, in this case, apps/v1.
- **metadata:** contains metadata about the Deployment, including its name.
- **spec:** defines the specification for the Deployment, including its selector, strategy, and template for creating Pods.

- **selector:** specifies the labels used to select the Pods that the Deployment manages.
- **matchLabels:** defines the labels used to match the Pods. In this case, it matches Pods with the labels application: wordpress and mode: wordpress-app.
- **strategy:** defines the update strategy for the Deployment. In this case, it is set to Recreate, which means that all Pods will be replaced when an update is made.
- **template:** specifies the template for creating the Pods.

  metadata: contains labels for identification.

  - **name:** specifies the name of the Pod.
  - **labels:** define the labels for the Pod. In this case, it has labels application: wordpress and mode: wordpress-app.

  spec: defines the specification for the Pod, including its containers, environment variables, ports, and volume mounts.

  - **containers:** specifies the containers to run in the Pod. In this case, there is one container named "wordpress" using the wordpress:6.2.1-apache image.
    - **env:** defines the environment variables for the container, including the WordPress database host, password, and user. Some values are retrieved from a secret named "mysql-pass".
  - **ports:** defines the container port to expose for WordPress, which is set to 80.
  - **volumeMounts:** specifies the volume mount for persistent storage, which will be mounted at /var/www/html.

  volumes: defines the volumes to be used by the Pod. In this case, there is one volume named "wordpress-persistent-storage" referencing a PersistentVolumeClaim named "wp-pv-claim".

This Deployment configuration creates and manages Pods running WordPress with the specified environment variables, ports, and volume mounts. The Deployment ensures that the desired number of replicas is maintained and handles updates using the Recreate strategy.

Secret object:

A secret object in Kubernetes is a way to store sensitive data, such as passwords, database credentials, and API keys. Secrets are stored in a secure manner and are not accessible to users or applications by default.

Secrets can be used in a variety of ways in Kubernetes, such as:

- **As environment variables:** Secrets can be mounted as environment variables in Pods. This allows you to store sensitive data in a secure location and access it from your applications without having to expose it in plain text.
- **As config maps:** Secrets can be mounted as config maps in Pods. This is similar to using secrets as environment variables, but config maps allow you to store more complex data, such as JSON or YAML files.
- **As volume mounts:** Secrets can be mounted as volume mounts in Pods. This allows you to store sensitive data in a persistent volume, which is a way to store data that persists even if the Pod is terminated.

Here are some of the benefits of using secrets in Kubernetes:

- **Security:** Secrets are stored in a secure manner and are not accessible to users or applications by default. This helps to protect sensitive data from unauthorized access.
- **Convenience:** Secrets can be easily created and managed using the `kubectl` command-line tool. This makes it easy to store and access sensitive data in your Kubernetes deployments.
- **Flexibility:** Secrets can be used in a variety of ways in Kubernetes, such as as environment variables, config maps, or volume mounts. This gives you flexibility in how you store and access sensitive data in your deployments.

Kustomization file, which is a way to customize Kubernetes resources. Kustomization files are used to define how Kubernetes resources should be generated, including the parameters that should be passed to them.

vi kustomization.yaml

```
secretGenerator:
- name: mysql-pass
  literals:
  - password=Vishal@1995
resources:
  - mysql-pv.yml
  - wordpress-pv.yml
  - mysql-service.yml
  - wordpress-service.yml
  - deploymysql.yml
  - deploy-wordpress.yml
```

The `secretGenerator` section of the Kustomization file defines a secret generator named `mysql-pass`. This secret generator will generate a secret object with the name `mysql-pass` that contains the literal value `password=Vishal@1995`. This secret object can then be used in other Kubernetes resources, such as Pods or Deployments, to store the password for a MySQL database.

The `resources` section of the Kustomization file defines the Kubernetes resources that should be generated. In this case, the Kustomization file defines the following resources:

- `mysql-pv.yml`: This resource defines a PersistentVolumeClaim for MySQL data.
- `wordpress-pv.yml`: This resource defines a PersistentVolumeClaim for WordPress data.
- `mysql-service.yml`: This resource defines a Service for MySQL.
- `wordpress-service.yml`: This resource defines a Service for WordPress.
- `deploymysql.yml`: This resource defines a Deployment for MySQL.
- `deploy-wordpress.yml`: This resource defines a Deployment for WordPress.

Deploying WordPress and MySQL with Persistent Volumes Using K8s

t.me/vishalk17 github.com/vishalk17

Lets deploy all things I have written:

- The `kustomization.yaml` contains all the resources for deploying a WordPress site and a MySQL database. You can apply the directory by

- The command `kubectl apply -k ./` is used to apply a Kustomization file to a Kubernetes cluster. The `-k` flag tells the `kubectl` command to look for Kustomization files in the current directory. The `./` specifies the current directory.

kubectl apply -k ./

```
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ ls -l
total 28
-rw-rw-r-- 1 vishal vishal 1092 Jul 13 18:13 deploymysql.yml
-rw-rw-r-- 1 vishal vishal 1062 Jul 13 18:29 deploy-wordpress.yml
-rw-rw-r-- 1 vishal vishal  215 Jul 13 14:49 kustomization.yaml
-rw-rw-r-- 1 vishal vishal  216 Jul 13 22:03 mysql-pv.yml
-rw-rw-r-- 1 vishal vishal  197 Jul 13 18:22 mysql-service.yml
-rw-rw-r-- 1 vishal vishal  311 Jul 13 17:32 wordpress-pv.yml
-rw-rw-r-- 1 vishal vishal  430 Jul 13 16:41 wordpress-service.yml
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl apply -k ./
secret/mysql-pass-t879c75gdf created
service/wordpress-mysql created
service/wordpress-service created
persistentvolumeclaim/mysql-pv-claim created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/deployment-wordpress created
deployment.apps/wordpress-mysql created
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$
```

```
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl get secret
NAME                  TYPE     DATA   AGE
mysql-pass-t879c75gdf   Opaque   1      41s
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl get deploy
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment-wordpress   1/1     1            1           48s
wordpress-mysql        1/1     1            1           47s
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM                     STORAGECLASS   REASON   AGE
pvc-119bd981-09d5-4c16-9996-943ae410d797   4Gi        RWO            Delete           Bound      default/wp-pv-claim       standard                50s
pvc-42221a7c-aec8-47b2-9481-6fc68e2ebd36   20Gi       RWO            Delete           Released   default/mysql-pv-claim    standard                4h31m
pvc-95d3d462-41b3-4d2f-b1bb-18743e2b148d   4Gi        RWO            Delete           Released   default/wp-pv-claim       standard                4h31m
pvc-9b91b5a3-f969-4797-be0c-2fc3df601d4b   4Gi        RWO            Delete           Released   default/wp-pv-claim       standard                8h
pvc-ad98ac36-b58c-4cd0-b954-7248ab0fd98c   4Gi        RWO            Delete           Released   default/mysql-pv-claim    standard                8h
pvc-f6369aea-6bdc-4cb0-af9f-a5028d94d415   4Gi        RWO            Delete           Bound      default/mysql-pv-claim    standard                50s
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
deployment-wordpress-57f5849687-zkccc   1/1     Running   0          72s
wordpress-mysql-76bdc64d57-5q9vt        1/1     Running   0          72s
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl get svc
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP        8h
wordpress-mysql   ClusterIP      None            <none>        3306/TCP       88s
wordpress-service   LoadBalancer   10.107.123.28   <pending>     80:31977/TCP   88s
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$
```

Lets access it from web

`public-Ip-or-dns:port` – paste this link in web browser to check whether wordpress site is accessible or not

—-- only minikube user follow this things all —--

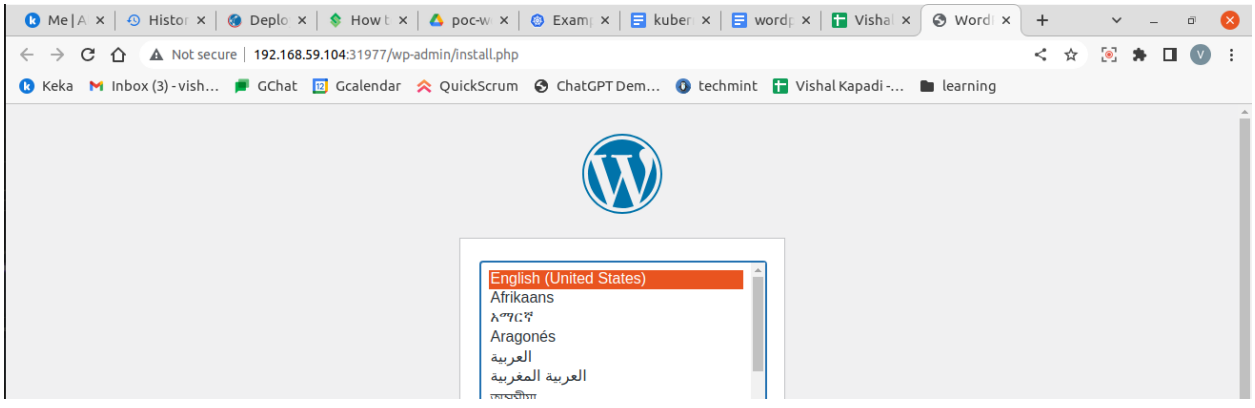
minikube service service-name --url .. to get link to access app

minikube service wordpress-service --url

```
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ kubectl get svc
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes          ClusterIP      10.96.0.1       <none>        443/TCP        8h
wordpress-mysql     ClusterIP      None            <none>        3306/TCP       4m43s
wordpress-service   LoadBalancer   10.107.123.28   <pending>     80:31977/TCP   4m43s
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$ minikube service wordpress-service --url
http://192.168.59.104:31977
vishal@vishal-HP-245-G8:~/kubernetes/wordpress$
```

—---------- end —----------

Let try to access link



Its working

Deploying WordPress and MySQL with Persistent Volumes Using K8s

t.me/vishalk17 github.com/vishalk17

Sourcecode :

- https://github.com/vishalk17/k8s-wordpress-deployment

My devops repo :

- https://github.com/vishalk17/devops

My telegram channel:

- https://t.me/vishalk17_devops

Contact:

Telegram : t.me/vishalk17

Refrences :

- https://www.cloudsigma.com/how-to-deploy-wordpress-with-persistent-volume-on-kubernetes-cluster/#:~:text=WordPress%20is%20one%20of%20the,platform%20on%20the%20Kubernetes%20cluster
- https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/