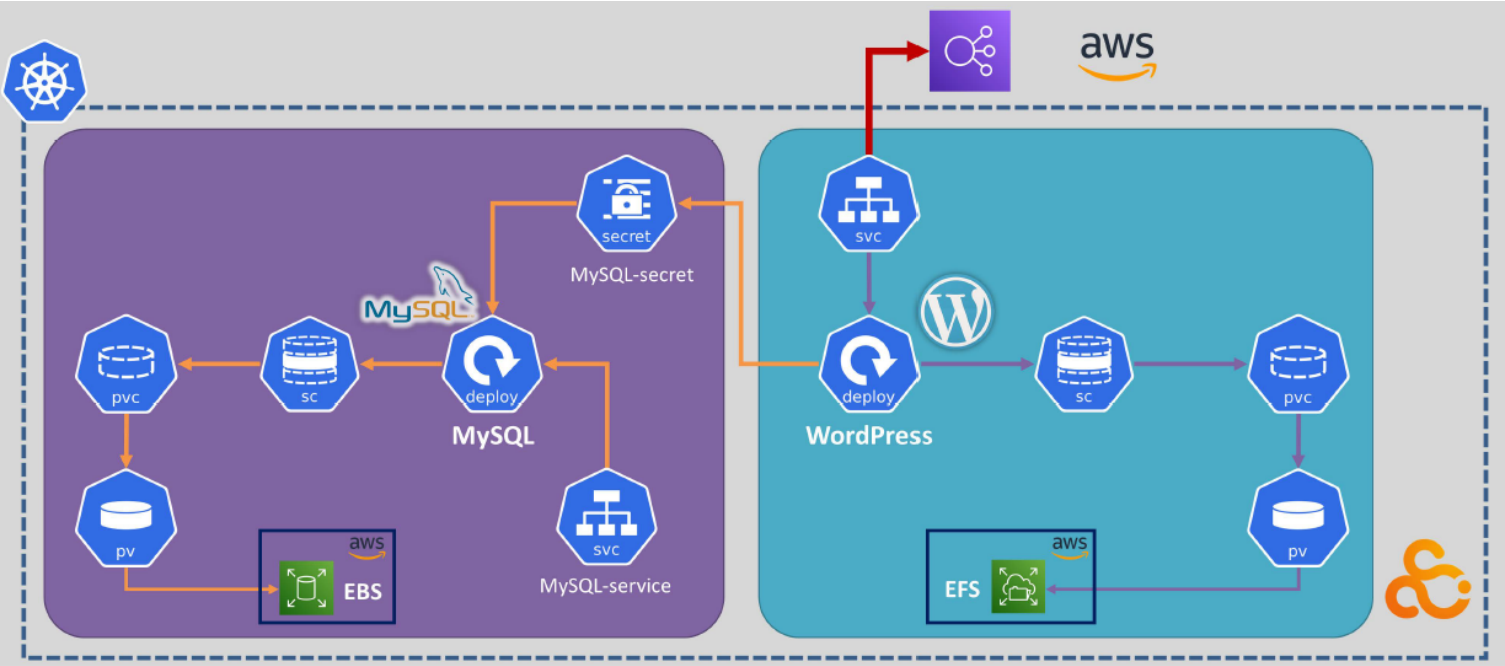
Deploying A WordPress Site with MySQL Database Using Kubernetes on AWS



Project Summary:

This project implements a scalable and cloud-native deployment of a WordPress website backed by a MySQL database, orchestrated using Kubernetes and integrated with AWS storage services for high availability and persistence.

Components:

a) MySQL Database:

- Deployed as a Kubernetes deployment.
- Secrets (MySQL-secret) are used for secure credential management.
- Exposed internally via a Kubernetes service (MySQL-service).
- Data persistence is ensured via a PersistentVolumeClaim (PVC) connected to an AWS Elastic Block Store (EBS) volume.

b) WordPress Application:

- Also deployed as a Kubernetes deployment.
- Communicates securely with the MySQL database using the MySQL-service.
- Exposes the application externally through a Kubernetes service which is fronted by an AWS Application Load Balancer (ALB).
- Persistent data such as uploads is stored using a PVC backed by AWS Elastic File System (EFS) for shared access and durability.

Benefits:

a) **Scalability:** Kubernetes enables horizontal scaling of WordPress pods to handle traffic spikes.
c) **High Availability:** Separate services for WordPress and MySQL promote fault tolerance and independent scaling.
d) **Persistent Storage:** EBS ensures reliable storage for MySQL, while EFS enables shared access to media files across WordPress pods.
e) **Security:** Sensitive database credentials are handled via Kubernetes Secrets.
f) **Cloud-Native Integration:** Deep integration with AWS services like EBS, EFS, and ALB allows for seamless cloud operation and resource management.

Use Cases:

a) Hosting dynamic content-driven websites and blogs using WordPress.
b) Demonstrating Kubernetes storage concepts with real-world AWS services.
c) Learning full-stack containerized application deployment on the cloud.

Deployment Steps:

1. Setup Kubernetes with kubeadm on AWS:

a) Launching **2 EC2 instances** (Master node – Worker node) with specs:
- **Instance Type:** t3.medium
- **OS:** ubuntu 22.04
- **Storage:** 20 GB (gp2)

b) Instances preparation and configuration via Bash script includes:
- Docker installation
- Hostname configuration
- Kubernetes installation

Script Link

c) Initialize Kubernetes on the Master node.

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config

#Deploy a Pod Network through the master node
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

#To Create a new token
sudo kubeadm token create --print-join-command
```

d) Joining Worker nodes to the Kubernetes Cluster by token which created by Master node.

e) Installing CSI – EFS driver:

```
kubectl apply -k "github.com/kubernetes-sigs/aws-efs-csi-
driver/deploy/kubernetes/overlays/stable/?ref=release-1.5"

kubectl get po -n kube-system

kubectl get csidriver

kubectl get csinode
```

f) Installing CSI – EBS driver:

```
kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-
driver/deploy/kubernetes/overlays/stable/?ref=release-1.14"

kubectl get pods -n kube-system

kubectl get csidriver

kubectl get csinode
```

2. IAM Configurations:

a) Creating a user with EBS & EFS permissions (efs-ebs-eslam-wp)

b) Creating and downloading access key for the user

c) Creating a role with EBS & EFS permissions (ebs-efs-eslam-wp)

d) Attaching created role to machines

e) Adding user credential to Kubernetes cluster:

```
kubectl create secret generic efs-ebs-eslam-wp \
     --namespace kube-system \
     --from-literal "key_id=AK************" \
     --from-literal "access_key=ie**********"

#list all secrets
Kubectl get secret -n kube-system
```

3. Initializing MySQL Database (BackEnd):

a) Creating a Secret for MySQL password:

```
echo -n 'myrootpassword' | openssl base64
bXlyb290cGFzc3dvcmQ=
```

b) Writing a **Secret** YAML file

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-password
type: Opaque
data:
  password: bXlyb290cGFzc3dvcmQ=
```

c) Writing a **Storage Class** YAML file for MySQL

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: mysql-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
```

d) Writing a **PVC** YAML file for MySQL

```
apiVersion: v1
kind: PersistentVolumeClaim

metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName:  mysql-sc
  resources:
    requests:
      storage: 5Gi
```

e) Writing a **Deployment** YAML file for MySQL

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-app
spec:
  selector:
    matchLabels:
      app: wp
      tier: mysql
  template:
    metadata:
      labels:
        app: wp
        tier: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.6
          env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                key: password
                name: mysql-password
          ports:
            - containerPort: 3306
          volumeMounts:
            - mountPath: /var/lib/mysql
              name: mysql-storage
      volumes:
        - name: mysql-storage
          persistentVolumeClaim:
            claimName: mysql-pvc
```

f) Writing a **Service** YAML file for MySQL

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc
spec:
  selector:
    app: wp
    tier: mysql
  ports:
    - port: 3306
```

4. Initializing WordPress (FrontEnd):

a) Writing a **Storage Class** YAML file for WordPress

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: wp-sc
provisioner: efs.csi.aws.com
```

b) Creating an **EFS** on AWS with access point specs:

- Name: efs
- Root dir: /wordpress
- User ID & Owner user ID: 1000
- Group ID & Owner Group ID: 1000
- Access Point permission: 777

c) Writing a **PV** YAML file for WordPress

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: wp-efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: wp-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-0b4115f0413669a96::fsap-08499189ba31d1961
```

d) Writing a **PVC** YAML file for WordPress

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-efs-pvc
spec:
  accessModes:
   - ReadWriteMany
  storageClassName: wp-sc
  resources:
```

```
    requests:
      storage: 5Gi
```

e) Writing a **Deployment** YAML file for WordPress

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wp-app
spec:
  selector:
    matchLabels:
      app: wp
      tier: frontend
  template:
    metadata:
      labels:
        app: wp
        tier: frontend
    spec:
      containers:
        - name: wordpress
          image: wordpress:php7.4-apache
          env:
            - name: WORDPRESS_DB_HOST
              value: mysql-svc
            - name: WORDPRESS_DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-password
                  key: password
          ports:
            - containerPort: 80
              name: wordpress
          volumeMounts:
            - name: wordpress-storage
              mountPath: /var/www/html
      volumes:
        - name: wordpress-storage
          persistentVolumeClaim:
            claimName: wp-efs-pvc

        - name: mysql-storage
          persistentVolumeClaim:
            claimName: mysql-pvc
```

f) Writing a **Service** YAML file for WordPress

```yaml
apiVersion: v1
kind: Service
metadata:
  name: wp-svc
spec:
  selector:
    app: wp
    tier: frontend
  ports:
    - port: 80
  type: LoadBalancer
```

g) Creating an **Application Load Balancer** and attaching machines to a **Target Group** working on port 30347

Final Structure: