Assume the web application (mock or real) is a simple service that connects to the PostgreSQL database using the provided credentials. We will create Kubernetes manifests (deployment.yaml) to define the app deployment in Minikube.

**Example Kubernetes Deployment for the Web Application:**

--------------------------------------------script starts here--------------------------------------------------------

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp

spec:

replicas: 1

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: myapp-container

image: myapp\_image:latest

env:

- name: DB\_HOST

value: "localhost" # If DB\_HOST needs to be dynamic, consider storing it in a secret or ConfigMap

- name: DB\_PORT

value: "5432" # Same as above for DB\_PORT

- name: DB\_USER

valueFrom:

secretKeyRef:

name: db-credentials # Using the db-credentials secret

key: username # Fetching the DB username from the secret

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-credentials # Using the db-credentials secret

key: password # Fetching the DB password from the secret

- name: DB\_NAME

value: "myapp\_db" # You can leave this in the deployment if it doesn't change dynamically

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: myapp-service

spec:

ports:

- port: 8080

selector:

app: myapp

--------------------------------------------script ends here----------------------------------------------------------

**Explanation of Key Updates:**

1. **DB\_USER and DB\_PASSWORD from Kubernetes Secret**:
   * Both DB\_USER and DB\_PASSWORD are now set from the db-credentials Kubernetes secret using secretKeyRef. This ensures that the database credentials are securely managed and not hardcoded in the deployment YAML.
   * The DB\_HOST and DB\_PORT environment variables can still be hardcoded if they are fixed (e.g., localhost and 5432), but if they are dynamic, you can store them in a Kubernetes secret or ConfigMap.
2. **Service Configuration**:
   * The service remains unchanged, exposing port 8080 to allow communication with the application.

**Creating the Kubernetes Secret (db-credentials):**

Since you're using Kubernetes secrets to store the database credentials, you need to ensure the db-credentials secret is created in your cluster. Here’s how you can create it with kubectl using the generated credentials from Terraform:

bash

Copy

kubectl create secret generic db-credentials \

--from-literal=username=myapp\_user \

--from-literal=password=<your\_generated\_password\_here>

You can replace <your\_generated\_password\_here> with the actual password returned by Terraform. This secret will be used to inject the database credentials into the DB\_USER and DB\_PASSWORD environment variables.

**Summary of Key Points:**

* **Secrets**: The database username (DB\_USER) and password (DB\_PASSWORD) are securely injected into the deployment via Kubernetes secrets (db-credentials).
* **DB\_HOST & DB\_PORT**: If they are static, they can remain hardcoded. Otherwise, consider moving them to a ConfigMap or secret for dynamic configuration.
* **Service**: The service configuration for exposing the application remains the same.

This setup ensures secure handling of database credentials and follows Kubernetes best practices for managing secrets and environment variables.