Implementation of Docker and DevOps Principles in the Django-React Web Project

Paper - https://ieeexplore.ieee.org/document/10117715
Date - 15 Nov 2024

Introduction

This report outlines the implementation of principles discussed in ***"Architecture Design of Enterprise Information System Based on Docker and DevOps Technology"*** within a Django-React-based web application. The project integrates Docker for containerization, Kubernetes for orchestration, and DevOps methodologies to ensure a scalable, efficient, and maintainable deployment architecture.

System Overview

The project architecture includes:

1. **Backend**: Django for server-side logic and PostgreSQL database for persistent storage.
2. **Frontend**: React for a responsive user interface.
3. **Containerization**: All components are encapsulated in Docker containers.
4. **Orchestration**: Kubernetes is used for managing container clusters, ensuring high availability and scalability.

Alignment with Paper

Below is a detailed alignment of the project's implementation with the paper's recommendations:

Aspect	Paper Recommendations	Implementation in Project
Containerization	Use Docker to package applications and dependencies.	Docker images for Django, React, and PostgreSQL.
Microservices Architecture	Modularize into independent services.	Backend and frontend in independent containers; potential for further modularization.
Orchestration	Use Kubernetes for distributed container management.	Kubernetes manages clusters and load balancing.
Distributed Gateways	Deploy a gateway for load balancing and routing.	Kubernetes ingress handles load balancing.
High Availability	Implement redundancy and automated failover.	Kubernetes ensures redundancy and self-healing.
Monitoring	Use tools like Zabbix for system health monitoring.	Monitoring implemented using Prometheus/Grafana.

Key Features of the Implementation

1. **Containerized Deployment**:
    - Docker images ensure isolated environments for the Django backend and React frontend.
    - PostgreSQL and other services are containerized, simplifying cross-environment compatibility.
2. **Kubernetes Orchestration**:
    - Kubernetes clusters enable distributed deployment, ensuring system availability.
    - Scaling is achieved dynamically, with Kubernetes managing resource allocation.
3. **Load Balancing and Gateway**:
    - Kubernetes ingress serves as the gateway for routing and balancing user requests.
    - High concurrency and efficient request handling are achieved.
4. **Monitoring and Maintenance**:
    - Prometheus and Grafana monitor application health, logs, and metrics.
    - Alerts notify the team of performance bottlenecks or failures.

Benefits Achieved

1. **Scalability**:
    - The system dynamically adjusts to workload variations.
2. **Reliability**:
    - Redundant nodes and Kubernetes self-healing ensure high availability.
3. **Efficiency**:
    - Docker reduces resource overhead, and Kubernetes optimizes resource usage.
4. **Maintainability**:
    - DevOps practices streamline updates and minimize downtime.
5. **Observability**:
    - Monitoring tools provide actionable insights into system health.

Kubernetes Configuration for Course Recommendation System (CRS)

This documentation provides an overview of the various Kubernetes YAML configurations for deploying the Course Recommendation System (CRS) in a containerized environment using Docker, PostgreSQL, Django, and React.

---

1. **Namespace**
    - File: *namespace.yaml*
    - Purpose: Creates an isolated namespace `crs-namespace` for all CRS resources.
2. **Django Backend**
    - *django-config.yaml*: Stores environment variables for Django.
    - *django-secret.yaml*: Securely stores sensitive database credentials.
    - *django-deployment.yaml*: Deploys the Django backend application.
    - *django-service.yaml*: Exposes the Django backend as a service.
3. **React Frontend**
    - *react-deployment.yaml*: Deploys the React frontend application.
    - *react-service.yaml*: Exposes the React frontend as a service.
4. **PostgreSQL Database**
    - *postgres-pv.yaml*: Defines persistent storage for the database.
    - *postgres-pvc.yaml:* Claims storage for PostgreSQL using the PersistentVolume.
    - *postgres-secret.yaml:* Securely stores PostgreSQL credentials.
    - *postgres-deployment.yaml:* Deploys the PostgreSQL database container.
    - *postgres-service.yaml:* Exposes PostgreSQL for communication with the Django backend.
5. **Ingress**
    - File: *ingress.yaml*
    - Purpose: Configures routing for the React frontend and Django backend.

---

1. django-config.yaml

- **Type**: ConfigMap
- **Purpose**: Contains environment variables for configuring the Django application.
- **Key Variables**:
    - `DJANGO_SETTINGS_MODULE`: Specifies the Django settings module.
    - `SECRET_KEY`: Secret key for Django.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: django-config
  namespace: crs-namespace
data:
  DJANGO_SETTINGS_MODULE: "course_recommendation_system.settings"
  SECRET_KEY:
"django-insecure-md#ke9-2o61v_z*dbp-h3bzmu&*b7bv!vxtg*f)^k=6%x&ouq%"
```

---

2. django-secret.yaml

- **Type**: Secret
- **Purpose**: Stores sensitive data, such as database credentials, for the Django application.
- **Encoded Data**:
    - `DATABASE_NAME`: Database name (base64 encoded).
    - `DATABASE_USER`: Database user (base64 encoded).
    - `DATABASE_PASSWORD`: Database password (base64 encoded).
    - `DATABASE_HOST`: Database host (base64 encoded).

```
apiVersion: v1
kind: Secret
metadata:
  name: django-secret
  namespace: crs-namespace
type: Opaque
data:
  DATABASE_NAME: Y3JzX2Ri # base64 encoded 'crs_db'
```

```
  DATABASE_USER: Y3JzX3VzZXI= # base64 encoded 'crs_user'
  DATABASE_PASSWORD: Q3JzQDU0MzIx # base64 encoded 'Crs@54321'
  DATABASE_HOST: cG9zdGdyZXM= # base64 encoded 'postgres'
```

---

3. django-deployment.yaml

- **Type**: Deployment
- **Purpose**: Deploys the Django backend container.
- **Key Configuration**:
  - Uses `django-config` for non-sensitive config values.
  - Uses `django-secret` for sensitive environment variables (database credentials).
  - Exposes the application on port `8000`.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: django-backend
  namespace: crs-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: django-backend
  template:
    metadata:
      labels:
        app: django-backend
    spec:
      containers:
      - name: django
        image: sntiwari07/course_recommendation_system_web:latest
        ports:
        - containerPort: 8000
        envFrom:
        - configMapRef:
```

```
      name: django-config
  - secretRef:
      name: django-secret
```

---

4. django-service.yaml

- **Type**: Service
- **Purpose**: Exposes the Django backend application to other services.
- **Key Configuration**:
  - Exposes port `8000` for internal communication.

```
apiVersion: v1
kind: Service
metadata:
  name: django-service
  namespace: crs-namespace
spec:
  ports:
  - port: 8000
  selector:
    app: django-backend
```

---

5. ingress.yaml

- **Type**: Ingress
- **Purpose**: Defines the routing rules for incoming HTTP requests.
- **Key Configuration**:
  - Directs traffic from `localhost` to the React frontend on port `3000`.
  - Routes API traffic to the Django service on port `8000`.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: crs-ingress
  namespace: crs-namespace
```

```
spec:
  rules:
  - host: localhost
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: react-service
            port:
              number: 3000
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: django-service
            port:
              number: 8000
```

---

6. postgres-pv.yaml

- **Type**: PersistentVolume
- **Purpose**: Defines storage for the PostgreSQL database.
- **Key Configuration**:
    - 1Gi storage with `ReadWriteOnce` access mode.
    - Mounted from `/mnt/data`.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv
  namespace: crs-namespace
spec:
  capacity:
```

```
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data" # Change this path as needed
```

---

7. postgres-pvc.yaml

- **Type**: PersistentVolumeClaim
- **Purpose**: Claims storage from the defined `postgres-pv`.
- **Key Configuration**:
    - Requests `1Gi` storage with `ReadWriteOnce` access mode.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
  namespace: crs-namespace
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

---

8. postgres-secret.yaml

- **Type**: Secret
- **Purpose**: Stores PostgreSQL credentials.
- **Encoded Data**:
    - `POSTGRES_DB`: PostgreSQL database name (base64 encoded).
    - `POSTGRES_USER`: PostgreSQL username (base64 encoded).
    - `POSTGRES_PASSWORD`: PostgreSQL password (base64 encoded).

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: postgres-secret
  namespace: crs-namespace
type: Opaque
data:
  POSTGRES_DB: Y3JzX2Ri # base64 encoded 'crs_db'
  POSTGRES_USER: Y3JzX3VzZXI= # base64 encoded 'crs_user'
  POSTGRES_PASSWORD: Q3JzQDU0MzIx # base64 encoded 'password'
```

---

9. postgres-deployment.yaml

- **Type**: Deployment
- **Purpose**: Deploys the PostgreSQL container.
- **Key Configuration**:
    - Uses `postgres-secret` for environment variables.
    - Mounts storage using `postgres-pvc`.
    - Exposes the database on port `5432`.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
  namespace: crs-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:latest
        ports:
```

```
        - containerPort: 5432
        env:
        - name: POSTGRES_DB
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: POSTGRES_DB
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: POSTGRES_USER
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: POSTGRES_PASSWORD
        volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgres-storage
      volumes:
      - name: postgres-storage
        persistentVolumeClaim:
          claimName: postgres-pvc
```

---

10. postgres-service.yaml

- **Type**: Service
- **Purpose**: Exposes PostgreSQL to other services within the cluster.
- **Key Configuration**:
    - Exposes port 5432 for internal communication.

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
  namespace: crs-namespace
```

```
spec:
  ports:
  - port: 5432
  selector:
    app: postgres
```

---

11. react-deployment.yaml

- **Type**: Deployment
- **Purpose**: Deploys the React frontend container.
- **Key Configuration**:
    - Exposes the React app on port `3000`.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-frontend
  namespace: crs-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: react-frontend
  template:
    metadata:
      labels:
        app: react-frontend
    spec:
      containers:
      - name: react
        image: sntiwari07/course_recommendation_system_frontend:latest
        ports:
        - containerPort: 3000
```

---

12. react-service.yaml

- **Type**: Service
- **Purpose**: Exposes the React frontend application to the users.
- **Key Configuration**:
    - Exposes port `3000` for communication.

```
apiVersion: v1
kind: Service
metadata:
  name: react-service
  namespace: crs-namespace
spec:
  ports:
  - port: 3000
  selector:
    app: react-frontend
```

---

13. namespace.yaml

- **Type**: Namespace
- **Purpose**: Defines the Kubernetes namespace `crs-namespace` for the Course Recommendation System.
- **Key Configuration**:
    - Used to isolate resources for the application.

```
apiVersion: v1
kind: Namespace
metadata:
  name: crs-namespace
```

It includes configurations for Django, PostgreSQL, React, and associated services like PersistentVolumes, Secrets, ConfigMaps, and Ingress routing. These YAML files help create a robust, containerized, and scalable application.