Docker Compose to Kubernetes

Migrating from Docker Compose to Kubernetes involves translating the Docker Compose configuration into Kubernetes resource definitions. Here’s a step-by-step example with comments explaining the YAML files involved.

Example Scenario

Let's assume we have a Docker Compose file that defines a simple web application with a single service (a Python Flask app) and a database (PostgreSQL).

Docker Compose File (`docker-compose.yml`):

```
version: '3'
services:
  web:
    image: my-web-app:latest
    ports:
      - "5000:5000"
    environment:
      - DATABASE_URL=postgres://user:password@db:5432/mydatabase
    depends_on:
      - db
  db:
    image: postgres:13
    environment:
      POSTGRES_DB: mydatabase
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
```

Migration to Kubernetes

1. **Create a Namespace (optional):** This isolates the resources for the application in a Kubernetes namespace.

```
# namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-app
```

2. **Define ConfigMap for Configuration (optional):** Stores non-sensitive configuration data.

```
# configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: web-config
  namespace: my-app
data:
  DATABASE_URL: postgres://user:password@db:5432/mydatabase
```

3. **Define Secrets for Sensitive Data:** Securely store sensitive information like database passwords.

```
# secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: my-app
type: Opaque
data:
  POSTGRES_PASSWORD: cGFzc3dvcmQ= # Base64 encoded password
```

4. **Define Persistent Volume and Persistent Volume Claim for the Database:** For stateful applications like databases.

```
# persistent-volume.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: db-pv
  namespace: my-app
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  hostPath:
    path: /mnt/data
```

```
# persistent-volume-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
  namespace: my-app
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

5. **Define Deployments for the Web App and Database:**

```
# web-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
  namespace: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: my-web-app:latest
          ports:
            - containerPort: 5000
          env:
            - name: DATABASE_URL
              valueFrom:
                configMapKeyRef:
                  name: web-config
                  key: DATABASE_URL
      restartPolicy: Always
```

```
# db-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db-deployment
  namespace: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
        - name: db
          image: postgres:13
          env:
            - name: POSTGRES_DB
              value: mydatabase
            - name: POSTGRES_USER
              value: user
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: db-secret
                  key: POSTGRES_PASSWORD
          volumeMounts:
            - name: db-storage
              mountPath: /var/lib/postgresql/data
      volumes:
        - name: db-storage
          persistentVolumeClaim:
            claimName: db-pvc
```

6. **Define Services for the Web App and Database:**

```
# web-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: web-service
  namespace: my-app
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
```

```
# db-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: db-service
  namespace: my-app
spec:
  selector:
    app: db
  ports:
    - protocol: TCP
      port: 5432
```

Applying the Configuration

1. Create the namespace:

```
kubectl apply -f namespace.yaml
```

2. Apply the ConfigMap and Secret:

```
kubectl apply -f configmap.yaml
kubectl apply -f secret.yaml
```

3. Create the Persistent Volume and Persistent Volume Claim:

```
kubectl apply -f persistent-volume.yaml
kubectl apply -f persistent-volume-claim.yaml
```

4. Deploy the web application and database:

```
kubectl apply -f web-deployment.yaml
kubectl apply -f db-deployment.yaml
```

5. Create the services:

```
kubectl apply -f web-service.yaml
kubectl apply -f db-service.yaml
```

This setup provides a basic migration path from Docker Compose to Kubernetes. Depending on your application's complexity and requirements, you might need additional configurations or adjustments.