📘 Design Decisions, Implementation Notes & Answers to All Assignment Questions

This section explains **why each architectural choice was made** and provides explicit **answers to all required “suggest and implement”** tasks from the DevOps challenge.

✅ 1. Kubernetes Cluster (Kind)

Reasoning

- Local, lightweight Kubernetes cluster.
- Supports multi-node topology.
- No cloud accounts required.
- Works perfectly with PVCs, StatefulSets, CronJobs, and NetworkPolicies.

Implementation

A 3-node cluster is created using `kind-config.yaml`:

- 1 control-plane node
- 2 worker nodes
- Host port `30080` mapped for browser-based testing

✅ 2. Database Cluster — MariaDB StatefulSet (Persistent)

Reasoning

- StatefulSets provide stable DNS names (`mysql-0`, `mysql-1`).
- Each replica gets its own PVC for durability.
- Ideal for database consistency and crash recovery.

Implementation

- MariaDB deployed as a StatefulSet.
- VolumeClaimTemplates ensure persistent storage.
- A headless service provides stable DNS for replicas.

✅ 3. Web Server (Nginx) With Multi-Replica Deployment

Reasoning

- Nginx is fast, small, and stable.
- Great for demonstrating configuration injection and init containers.

Implementation

- Deployment with multiple replicas.
- Custom Nginx config mounted via ConfigMap.
- Init container modifies HTML before Nginx starts.
- HTML page displays:
  - Pod IP
  - `serving-host = Host-xxxxx` (last 5 chars of pod name)

Example:

```
web-server-7f89cf47bf-25gxj → Host-5gxj
```

✅ 4. Restrict DB Access — Only Web Pods Can Reach MySQL

Assignment Requirement

> “Suggest and implement a way to only allow the web server pods to initiate connections to the database pods on port 3306.”

Choice Made: Kubernetes NetworkPolicy

Reasoning

- Native Kubernetes security control.
- Blocks all east-west traffic except what you explicitly allow.
- No service mesh required.

Implementation

NetworkPolicy allows:

```
from:
  - podSelector:
      matchLabels:
        app: devops-demo-web
ports:
  - port: 3306
```

All other traffic → **denied**.

✅ 5. Disaster Recovery for Database

Assignment Requirement

> “Suggest and implement a disaster recovery solution for the DB.”

Choice Made: mysqldump CronJob → backup PVC

Reasoning

- CronJobs automate recurring backups.
- `mysqldump` is universally compatible.
- Backup PVC stores dumps separate from DB storage.

Implementation

CronJob:

- Runs `mysqldump -A` daily (or more frequently).
- Dumps stored in `/var/lib/mysql-backups` on a PVC.
- Supports manual execution.

✅ 6. Flexible Way to Attach Pods to a Secondary Network

Assignment Requirement

> “Find and implement a flexible way to connect the Pod to a new network other than the Pods network with proper routes. No LoadBalancer needed.”

Choice Made: Multus CNI secondary network

Reasoning

- Multus allows pods to attach to **multiple network interfaces**.

- Industry-standard for multi-NIC Kubernetes workloads.
- Clean, declarative, and optional on a per-pod basis.

Implementation

Pods add:

```
annotations:
  k8s.v1.cni.cncf.io/networks: macvlan-conf
```

This attaches the pod to an additional network with its own routing table.

✅ 7. Schedule Specific DB Replicas on Specific Nodes

Assignment Requirement

> “Schedule specific replicas of the database cluster on specific K8s nodes.”

Choice Made: Node labels + NodeAffinity

Reasoning

- StatefulSets ensure stable replica numbering (`mysql-0`, `mysql-1`).
- NodeAffinity allows deterministic scheduling.
- Common in HA DB architectures (one replica per node).

Implementation

Nodes labeled:

```
kubectl label node worker-1 db=node-1
kubectl label node worker-2 db=node-2
```

StatefulSet includes:

```
nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
      - matchExpressions:
          - key: db
            operator: In
            values:
              - node-1
```

This pins DB replicas to specific nodes.

✅ 8. Golang Pod Monitoring Application

Reasoning

- Uses the official Kubernetes client-go library.
- Informers provide **real-time**, event-driven updates.
- Required events:
  - Pod created
  - Pod updated
  - Pod deleted

Implementation

- Watches all pods in all namespaces.
- Prints structured logs:
  - Timestamp
  - Event type
  - Pod namespace/name

    - Pod IP (if available)
    - Phase changes for update events

Example:

```
2025-11-23T20:22:29Z - ADDED: web-75546bd9dd-9t4bm (IP:10.244.2.14)
2025-11-23T20:40:23Z - UPDATED: web-75546bd9dd-vtsd4 (phase=Failed)
2025-11-23T20:39:54Z - DELETED: web-75546bd9dd-hgnwg
```

🏗️ Architecture Diagram

Placed inside the repository as `architecture.png`.