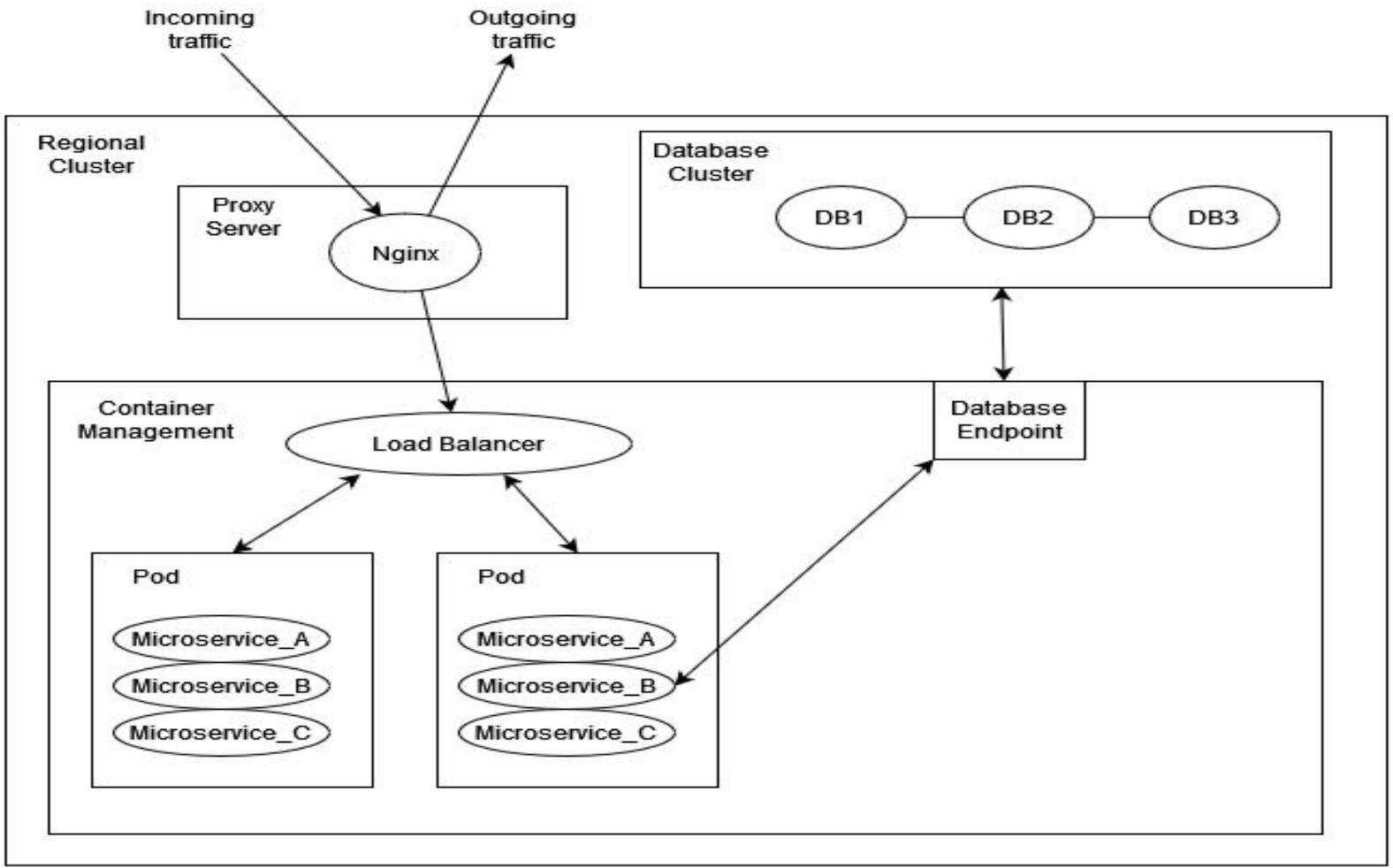
Solution Architecturing

Multi-region e-commerce platform


Incoming traffic
Outgoing traffic
Regional Cluster
Proxy Server
Nginx
Database Cluster
DB1
DB2
DB3
Container Management
Load Balancer
Database Endpoint
Pod
Microservice_A
Microservice_B
Microservice_C
Pod
Microservice_A
Microservice_B
Microservice_C

Openshift Container Platform vs Kubernetes

Openshift Container Platform (OCP)

Pros:

- **Ease** of use/implementation
- Enterprise Solution
- Built-in **security** measures

Cons:

- **Premium** pricing
- **OS limited** to Red Hat Enterprise Linux (RHEL)

Kubernetes (K8s)

Pros:

- Build infra-as-code from scratch
- Highly Customizable
- **Free** (Open source)

Cons:

- More **complex** configuration & management
- Takes **time and experience**
- Security based on own infra design

Openshift Container Platform vs Kubernetes

Both provides functionality of Platform as a Service (PaaS)

- Able to provide for **load balancing**
  - Covered by **Nginx** as a proxy-server
- Able to provide for **auto scaling** of microservices
  - Pods can be provisioned when **necessary**

High Availability

Fault Tolerance for various parts of our architecture

- In event of network outage, or component down

Extent of **solution** depends on our **Service-Level Agreement** (SLA)

- Microservice can work with cached data
- Database with built-in fault tolerance (Eg. etcd)

Disaster Recovery

Backup & Restore strategy:

- Subjected to the **Service-Level-Agreement** (SLA)
  - Eg: 99% uptime = only allow to down for ~100mins/week
- Hot/cold site service
  - Hot site = **Immediately available** copy of application that can take over as a **failover** solution
  - Cold site = Infrastructure & application ready to **start-up and take over**

Difference lies in **cost of maintaining** the backup/restore strategy

Disaster Recovery (application / microservices)

On application-level with our microservices, in event of crash

1. Container management level **spin up new pods** with the microservices
    a. Current pods could be kept alive, or killed/removed to save on cloud resources, depending on what the application logging strategy is
2. Roll-back application & microservices back to a previous version
    a. **Repeatable** step until a stable running version is serving

Spinning new containers / pods is a relatively **quick process**

Disaster Recovery (Database)

Backup strategy:

Asynchronous replication (hot backup):

- **Replicate** all current transactions to a 2nd/3rd site
- Backup site will then take over fully in disaster event **immediately**

Transaction-based replication (cold backup):

- Copy only the **transactions** being done to a 2nd/3rd site
- Keep backup on current database state
- Recovery event will then load previous backup database and **apply transactions** from backup time until current time