Lecture 06 · SENG 41283 · Distributed and Cloud Computing

Monolithic Architecture vs Microservices

Monolithic Design · Benefits and Drawbacks · Microservices Architecture · Kubernetes

1. Monolithic Architecture

A monolithic application is a single-tiered software application in which different components are combined into a single program from a single platform. Despite different modules, the application is built and deployed as ONE unit, typically using a relational database as the data source.

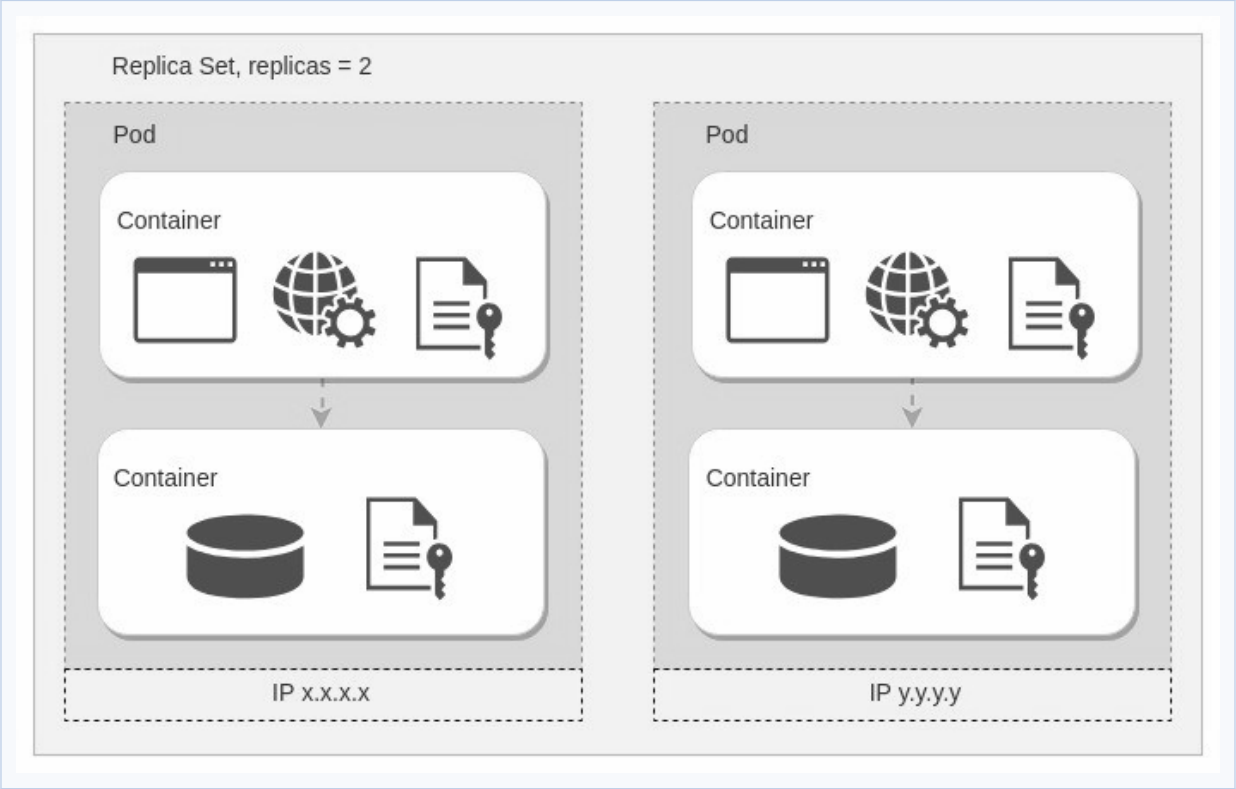

Figure 1.1 Monolithic architecture: all components built and deployed as a single unit sharing one database.

Components

- **Authorisation:** authenticates and authorises users.
- **Presentation:** handles HTTP requests; returns HTML or JSON/XML.
- **Business logic:** core application business logic.
- **Database layer:** data access objects for database access.
- **Application integration:** integration with other services via messaging or REST.
- **Notification module:** sends email notifications.

Benefits

- Simple to develop, test, and deploy at the beginning of a project.
- Simple to scale horizontally by running multiple copies behind a load balancer.

Drawbacks

- Maintenance becomes difficult as the application grows too large to understand entirely.
- Must redeploy the entire application on every update.
- Challenging to scale when different modules have conflicting resource requirements.
- A bug in any module (e.g. memory leak) can bring down the entire application.
- Difficult to adopt new technologies — changes affect the entire application.

2. Microservices Architecture

A large application built as a suite of modular, loosely coupled services. Each module supports a specific business goal with a simple, well-defined interface. Each microservice has its **own database** — essential for loose coupling.

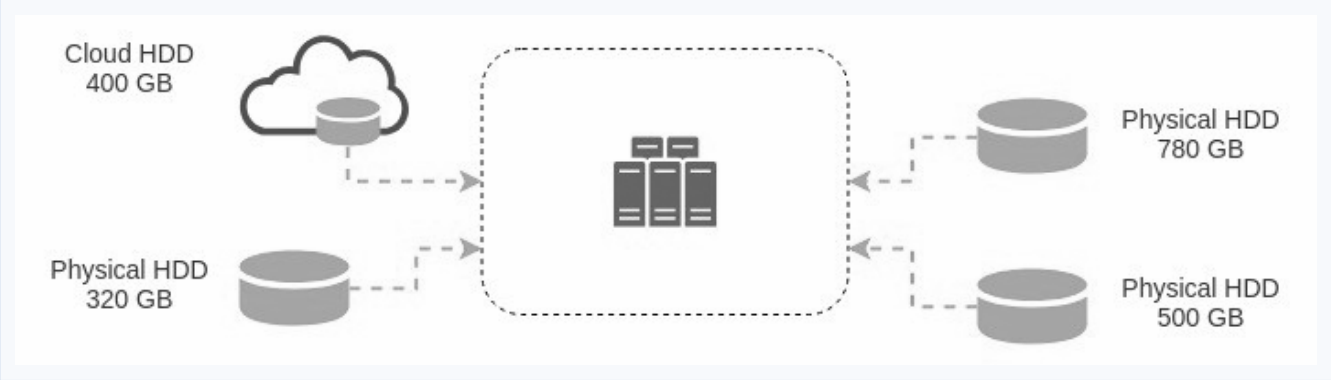

Figure 2.1 Microservices architecture: separate services with independent databases communicating via APIs.

E-Commerce Microservices

- Authorisation Service · Order Service · Catalogue Service · Cart Service · Payment Service · Shipping Service

Benefits

- Continuous delivery and deployment; better testability; independent deployment per team.
- Each service is small and easy to understand; improved fault isolation.
- No long-term commitment to a technology stack.

Drawbacks

- Additional complexity of creating a distributed system.
- Inter-service communication must be implemented; distributed transactions are hard.
- Deployment complexity; increased memory consumption.

3. Kubernetes

An open-source container orchestrator originally developed by Google. Provides velocity (immutability, declarative configuration, self-healing), scaling (decoupled architectures), infrastructure abstraction, and efficiency.

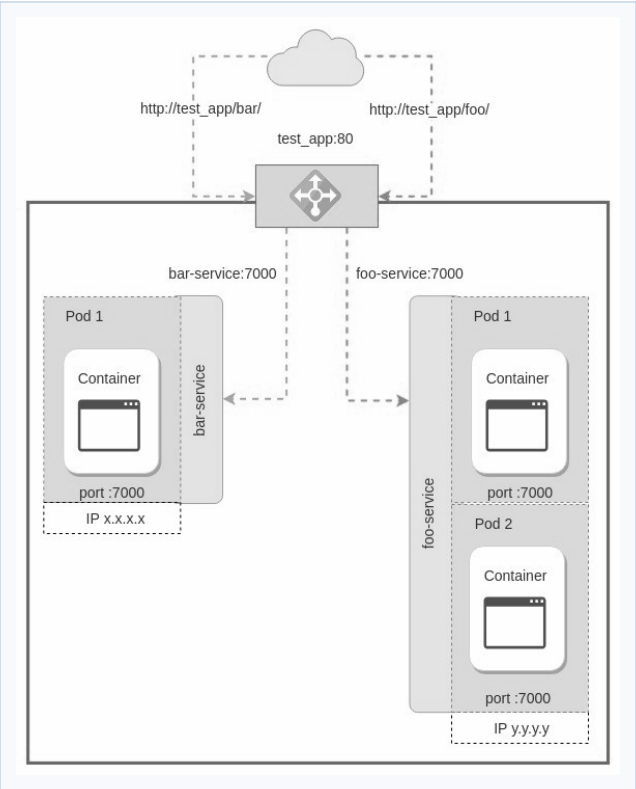

Figure 3.1 Kubernetes overview: cluster of nodes providing automated deployment, scaling, and management.

Hardware-Level Concepts

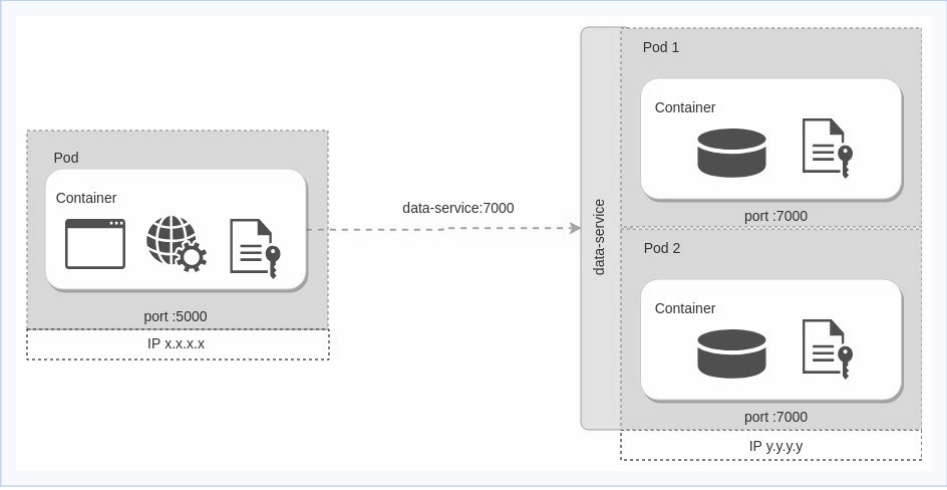

Figure 3.2 Kubernetes Node: smallest unit representing a single physical or virtual machine.

Figure 3.3 Kubernetes Cluster: collection of nodes; applications deployed without concern for underlying hardware.

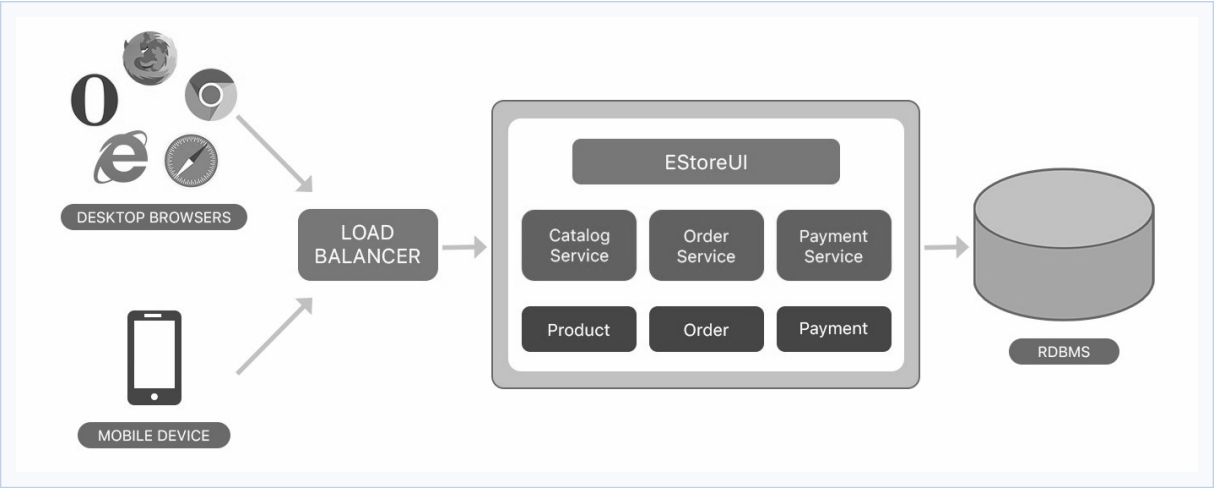

Figure 3.4 Kubernetes Persistent Volume: cluster-level storage not tied to any specific node.

Software-Level Concepts

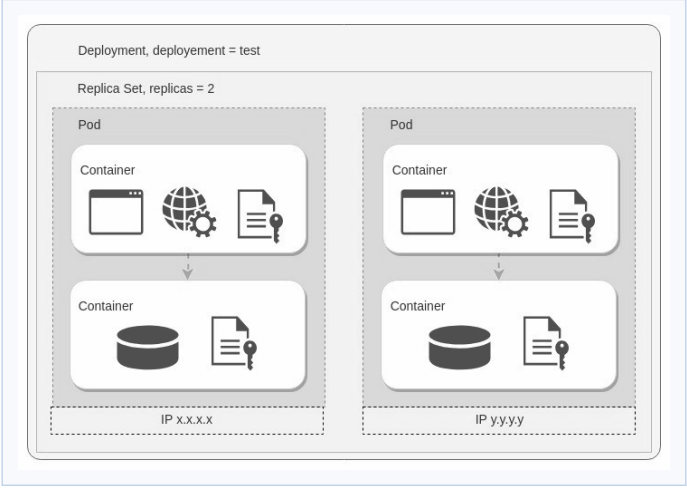

Figure 3.5 Kubernetes Pod: wraps one or more containers sharing resources and a network namespace.

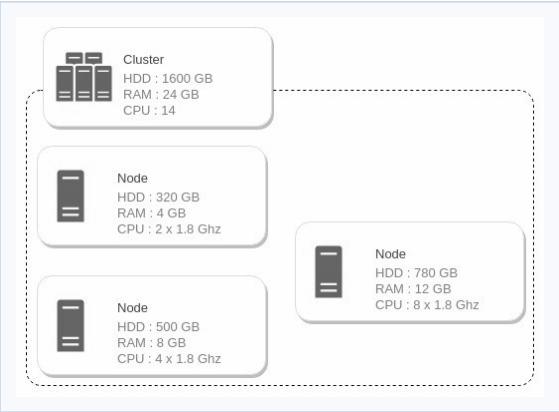

Figure 3.6 Kubernetes Replica Set: creates multiple Pod instances based on traffic load.

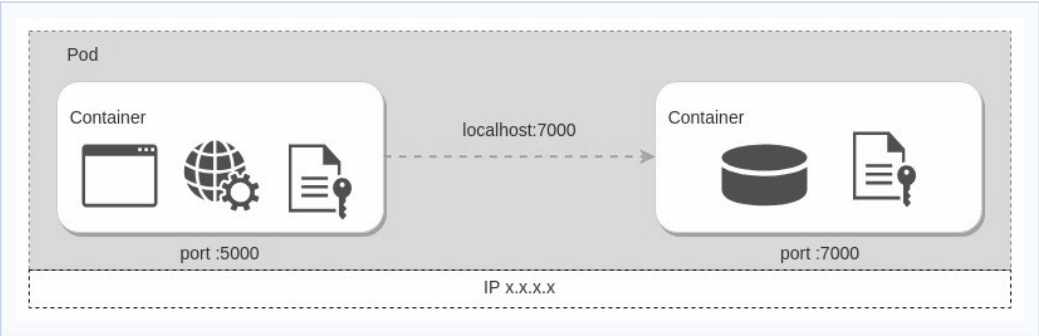

Figure 3.7 Kubernetes Deployment: highest-level abstraction; restarts pods if they fail.

Network-Level Concepts

• **Service:** pods assigned based on labels; service names used as hostnames within the cluster; Kubernetes proxy handles load balancing.

• **Ingress:** exposes internal services to external access; with Istio manages external traffic routing intelligently.