Introducing Cloud Native Applications

Presented by: Dr. Vipul Chudasama

Session Objectives

- Understand the definition of Cloud Native Applications
- Explore benefits and challenges
- Understand design principles and architecture
- Learn about tools and technologies
- Discuss real-world examples

What are Cloud Native Applications?

- Apps designed to run in cloud environments
- Built using microservices and containers
- Emphasize scalability, resilience, agility

Cloud Native Computing Foundation : -
Cloud Native is building software applications as a collection of independent, loosely coupled, business capability oriented services (microservices) that can run on dynamic environments(public ,private , hybrid, muti-cloud) in an automated , scalable, resilient, manageable and observable way.

Why Cloud Native Applications?

- Rapid delivery of features
- Scalability on-demand
- Automated management and deployment
- Better fault isolation

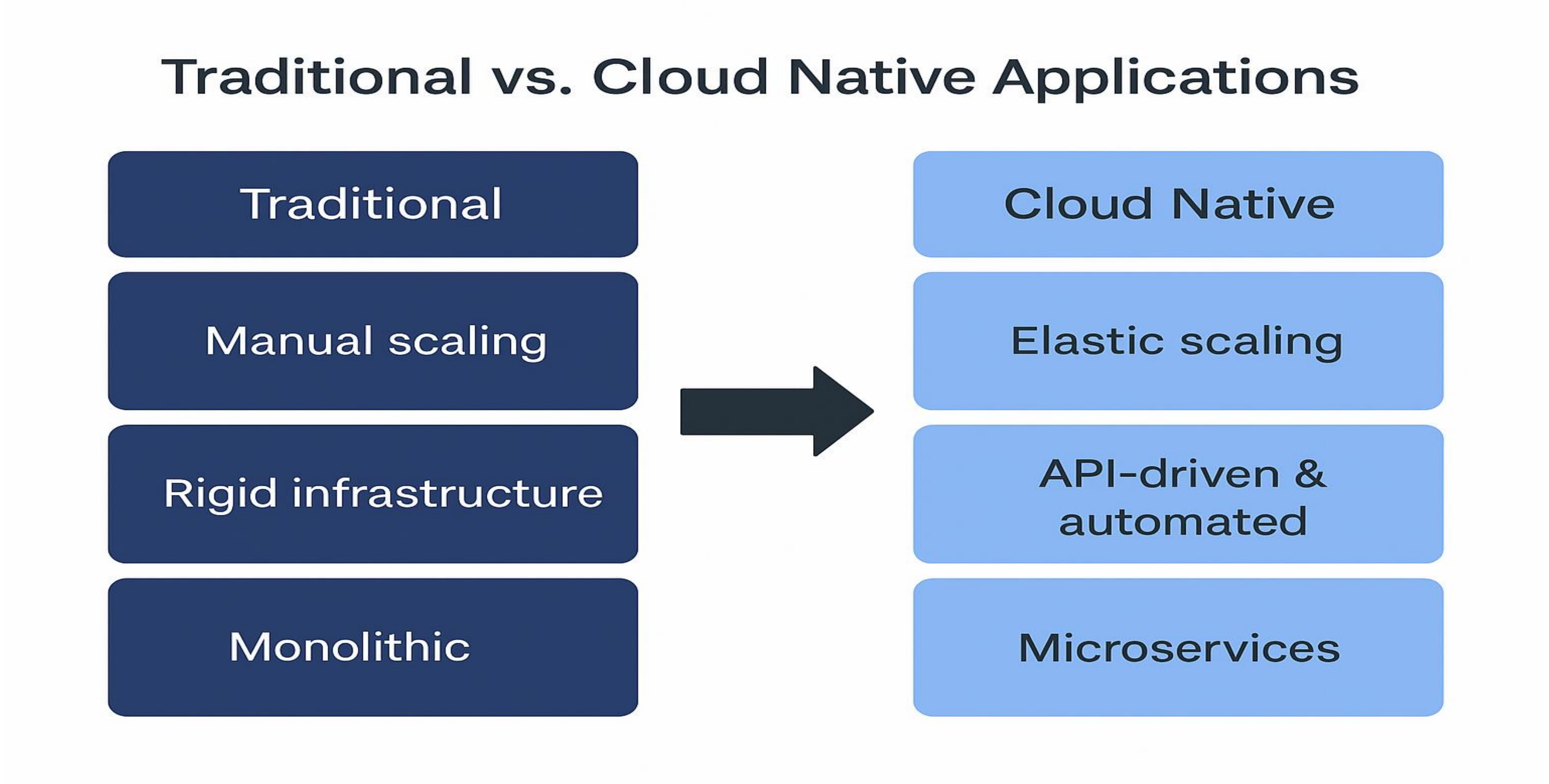
Software Application Journay

Monolithic

SOA

Microservices

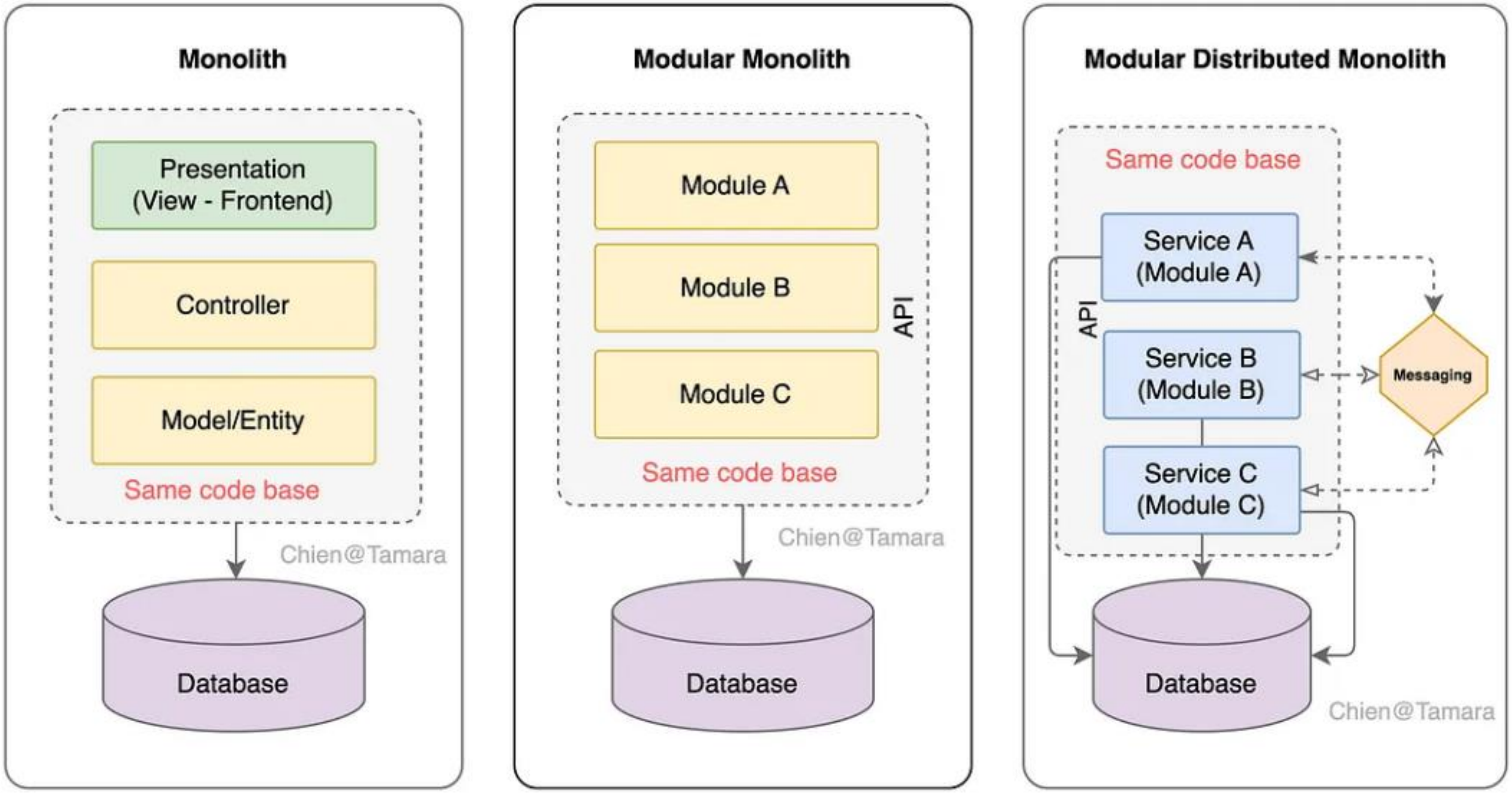
Traditional vs. Cloud Native Applications



Traditional vs. Cloud Native Applications

Traditional	Cloud Native
Manual scaling	Elastic scaling
Rigid infrastructure	API-driven & automated
Monolithic	Microservices

Monolithic


Monolith
Presentation (View - Frontend)
Controller
Model/Entity
Same code base
Chien@Tamara
Database
Modular Monolith
Module A
Module B
Module C
API
Same code base
Chien@Tamara
Database
Modular Distributed Monolith
Same code base
Service A (Module A)
Service B (Module B)
Service C (Module C)
API
Messaging
Database
Chien@Tamara

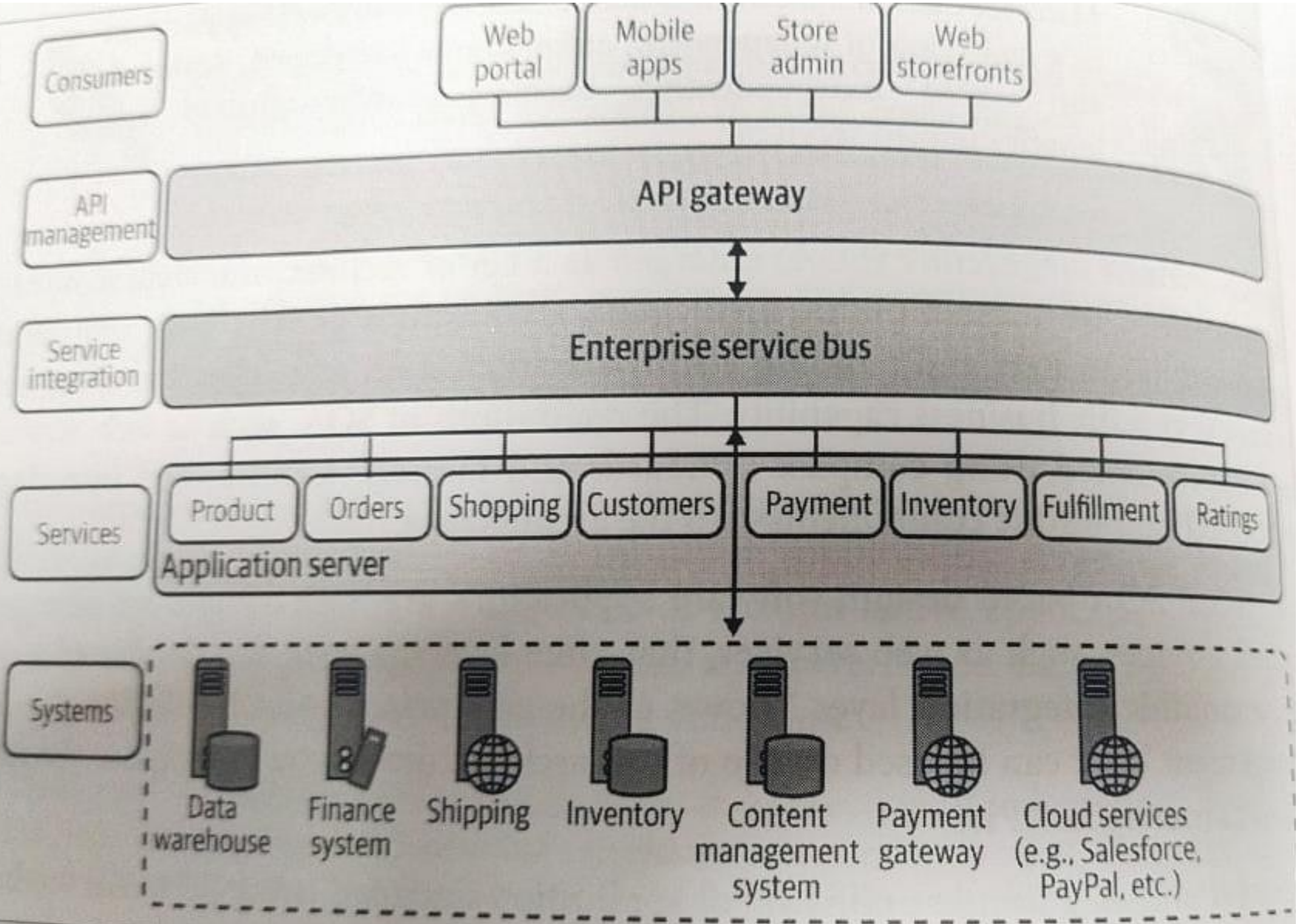


Figure 1-1. An online retail application scenario built using an SOA/ESB with API management

Microservices

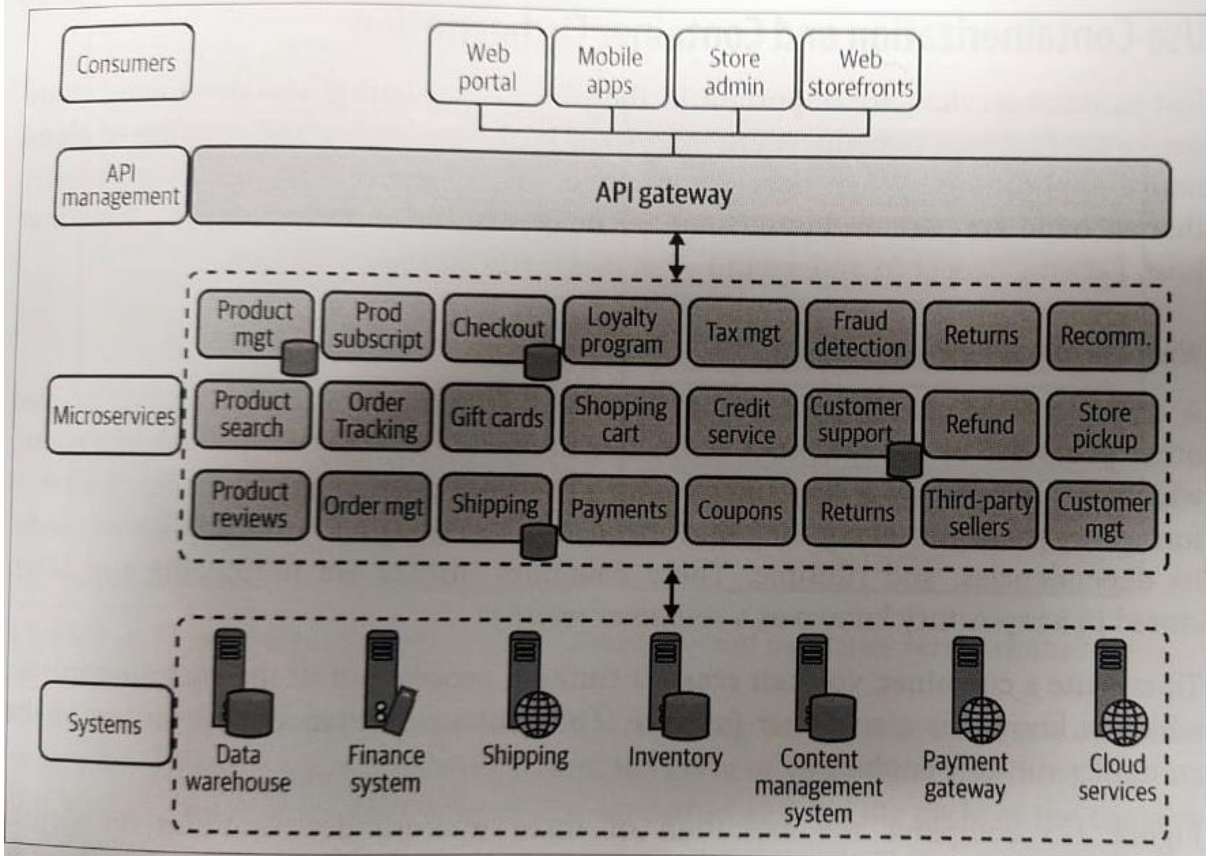
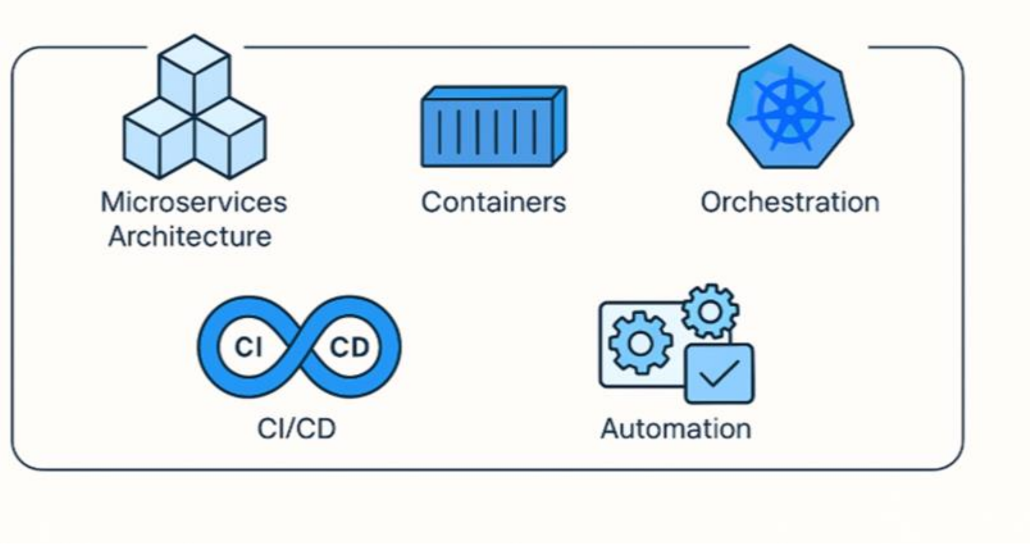


Figure 1-2. An online retail application built using microservices architecture

Book: Design pattens for Cloud Native Applications

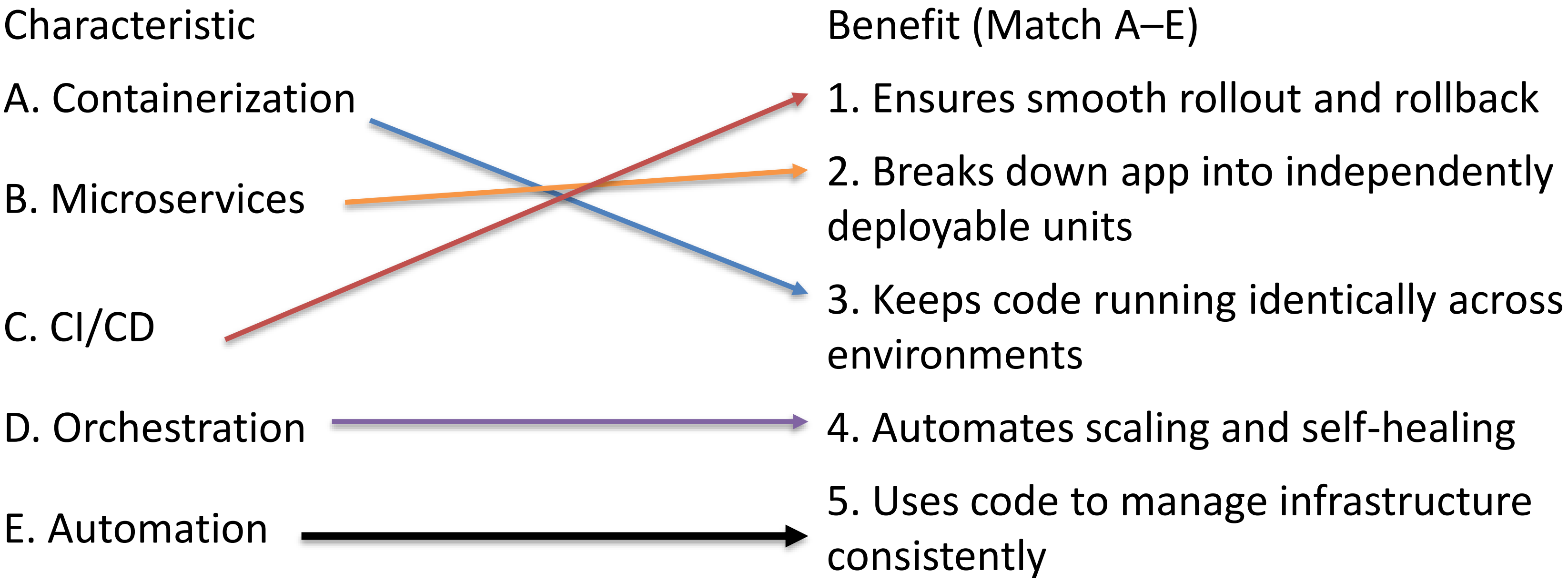
Key Characteristics

- Containerization (e.g., Docker)
- Microservices architecture
- CI/CD enabled
- Dynamic orchestration (e.g., Kubernetes)
- Infrastructure as Code



Activity

Characteristic	Benefit (Match A–E)
A. Containerization	1. Ensures smooth rollout and rollback
B. Microservices	2. Breaks down app into independently deployable units
C. CI/CD	3. Keeps code running identically across environments
D. Orchestration	4. Automates scaling and self-healing
E. Automation	5. Uses code to manage infrastructure consistently



Characteristic

A. Containerization

B. Microservices

C. CI/CD

D. Orchestration

E. Automation

Benefit (Match A–E)

1. Ensures smooth rollout and rollback

2. Breaks down app into independently deployable units

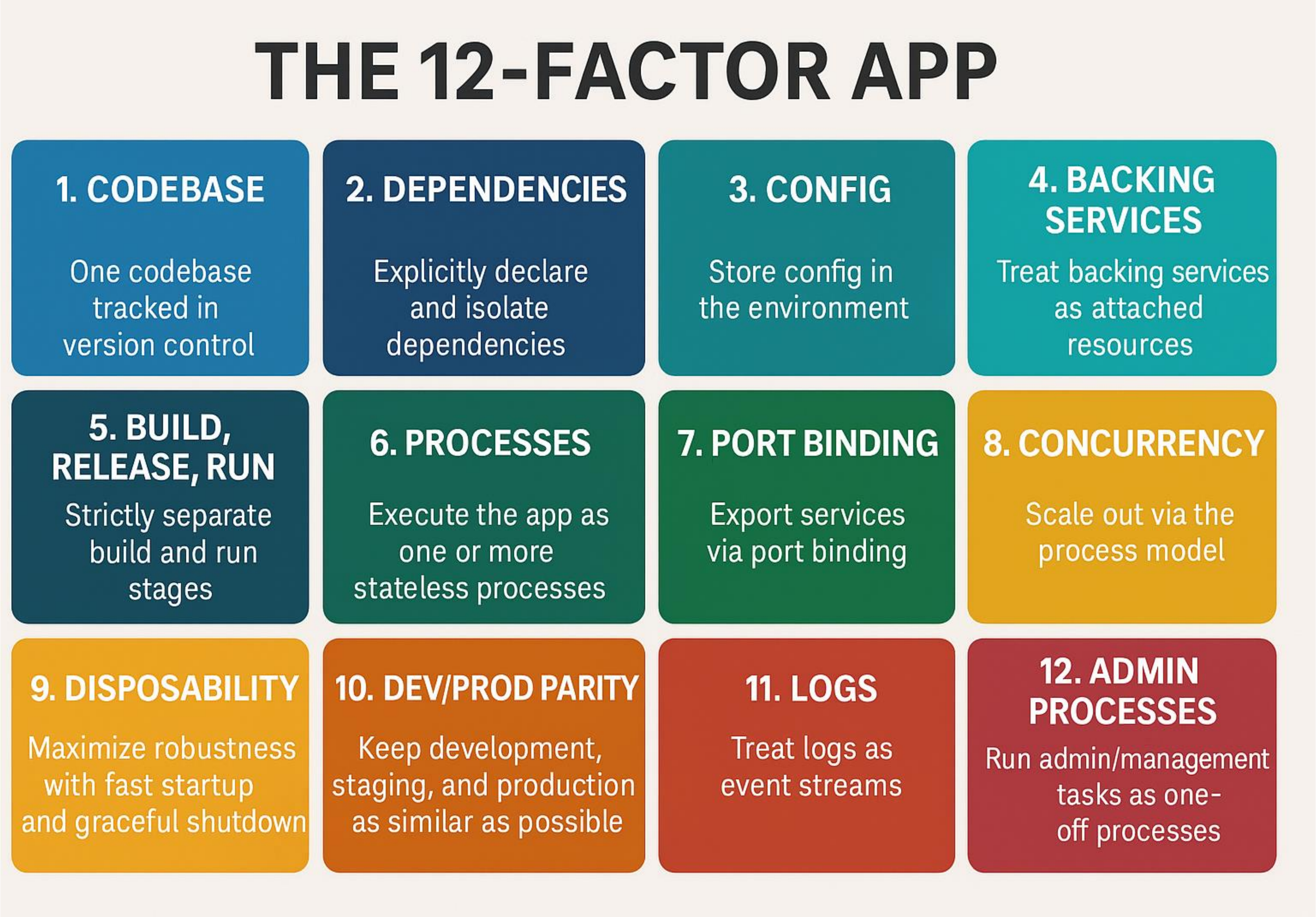
3. Keeps code running identically across environments

4. Automates scaling and self-healing

5. Uses code to manage infrastructure consistently

12-Factor App Methodology

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release, Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/prod parity
11. Logs
12. Admin processes



THE 12-FACTOR APP

1. CODEBASE

One codebase tracked in version control

2. DEPENDENCIES

Explicitly declare and isolate dependencies

3. CONFIG

Store config in the environment

4. BACKING SERVICES

Treat backing services as attached resources

5. BUILD, RELEASE, RUN

Strictly separate build and run stages

6. PROCESSES

Execute the app as one or more stateless processes

7. PORT BINDING

Export services via port binding

8. CONCURRENCY

Scale out via the process model

9. DISPOSABILITY

Maximize robustness with fast startup and graceful shutdown

10. DEV/PROD PARITY

Keep development, staging, and production as similar as possible

11. LOGS

Treat logs as event streams

12. ADMIN PROCESSES

Run admin/management tasks as one-off processes

Codebase

One codebase tracked in revision control, many deploys.“

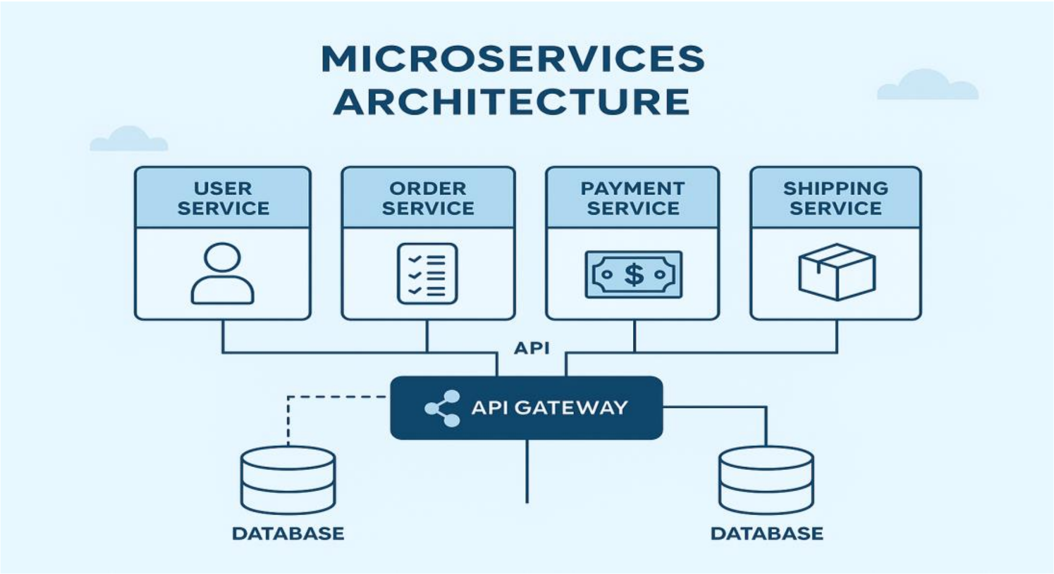
The **LMS** should have a **single codebase** in version control (e.g., GitHub).

It can be **deployed to multiple environments** (development, testing, staging, production), but all these share the same codebase.

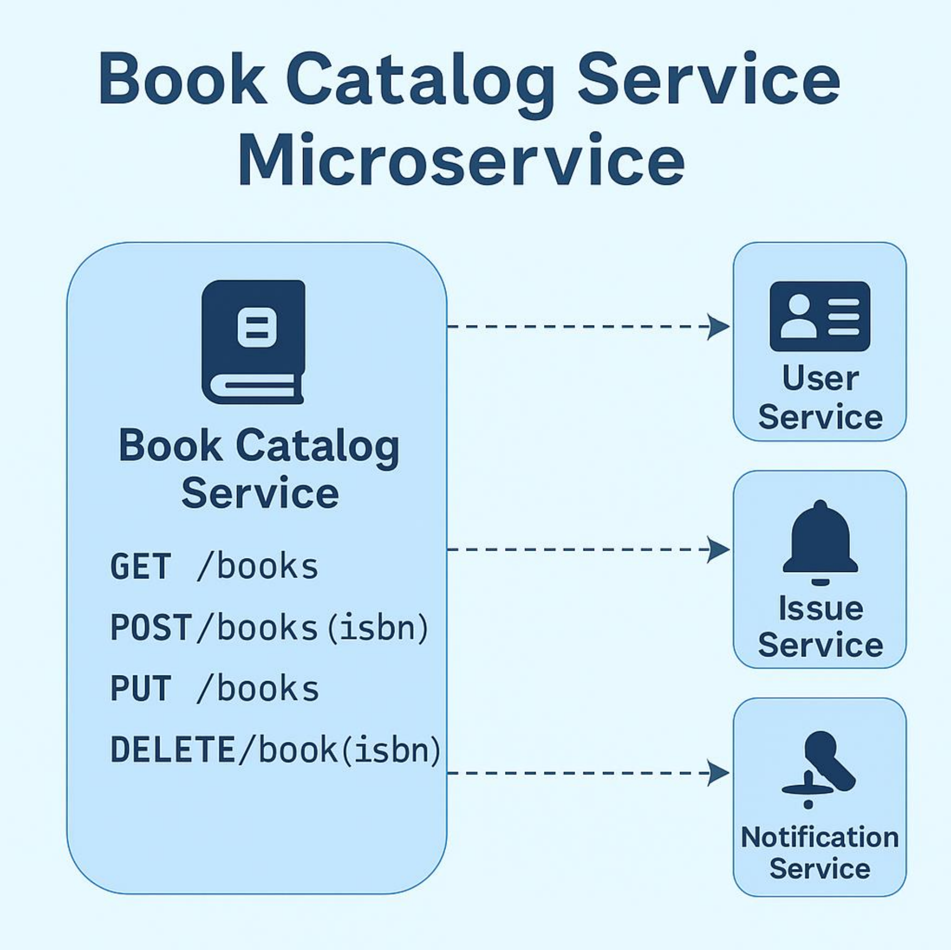
Example:
Your LMS project on GitHub is deployed to staging for testing and production for real use—but the code is the same.

Microservices in Cloud Native

- Independently deployable services
- Isolated failure domains
- Easier to scale and maintain



Book Catalog



```
GET  /books
POST/books(isbn)
PUT  /books
DELETE/book(isbn)
```

Containers & Kubernetes

- Containers package app and dependencies
- Kubernetes orchestrates containers
- Ensures availability, load balancing, rollout/rollback

DevOps & CI/CD

- Continuous Integration: Automated testing & builds
- Continuous Delivery: Frequent releases to staging/production
- Tools: Jenkins, GitHub Actions, GitLab CI

Observability & Monitoring

- Tools: Prometheus, Grafana, ELK Stack
- Metrics, logs, and traces
- Enables proactive issue detection

Security in Cloud Native Apps

- Secure APIs & secrets management
- Network policies
- Role-Based Access Control (RBAC)
- DevSecOps practices

Challenges of Cloud Native

- Complexity in orchestration
- Monitoring distributed systems
- Cultural shift in teams
- Data consistency & latency

Real-World Example – Netflix

- Migrated from monolith to microservices
- Uses containers, orchestration, and chaos engineering
- Rapid feature releases and global scale

Real-World Example – Uber

- Scalable ride-matching and maps
- Built with microservices and DevOps
- Observability and fast iteration

Summary

- Cloud Native Apps are modern, scalable, agile solutions
- Key concepts: Microservices, Containers, DevOps, Observability
- Embrace change, automate everything, scale intelligently

Quick Recap – Quiz

- What is a Cloud Native App?
- Name two benefits of containerization.
- What role does Kubernetes play?

Q&A

- Feel free to ask your questions!

Further Reading

- https://12factor.net
- https://kubernetes.io
- https://cloudnative.io
- CNCF Projects

Thank You

Prepared by: Dr. Vipul Chudasama

Department of Computer Engineering

Feedback and Questions Welcome