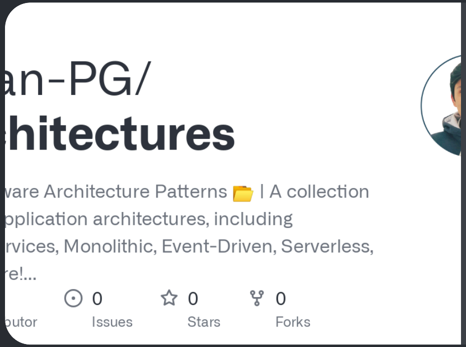
Microservices Architecture

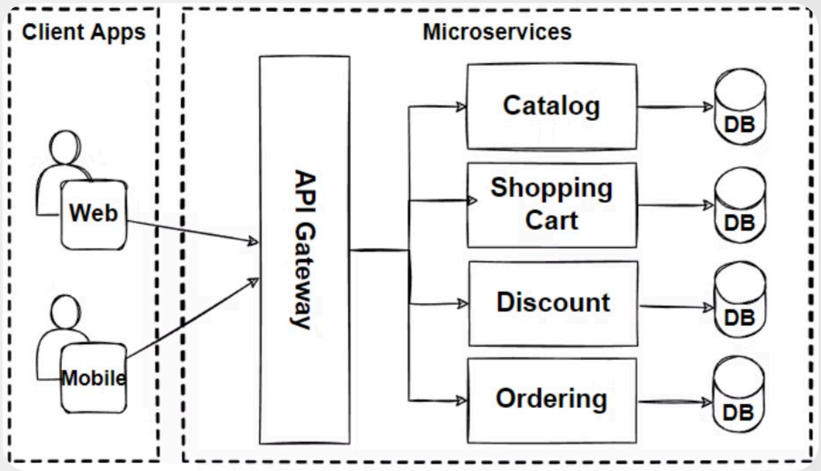
Microservices architecture structures an application as a collection of small, loosely coupled services. Each service handles a specific business function. Services communicate via APIs.


an-PG/
hitectures
vare Architecture Patterns 📁 | A collection
pplication architectures, including
rvices, Monolithic, Event-Driven, Serverless,
re!...
utor 0 Issues 0 Stars 0 Forks

GitHub

GitHub - Ryan-PG/architectures: 🚀 Softw...

🚀 Software Architecture Patterns 📂 | A collection of 10+ application architectures,...



Key Characteristics

Independence

Services are independently developed, deployed, and scaled.

Decentralized Data

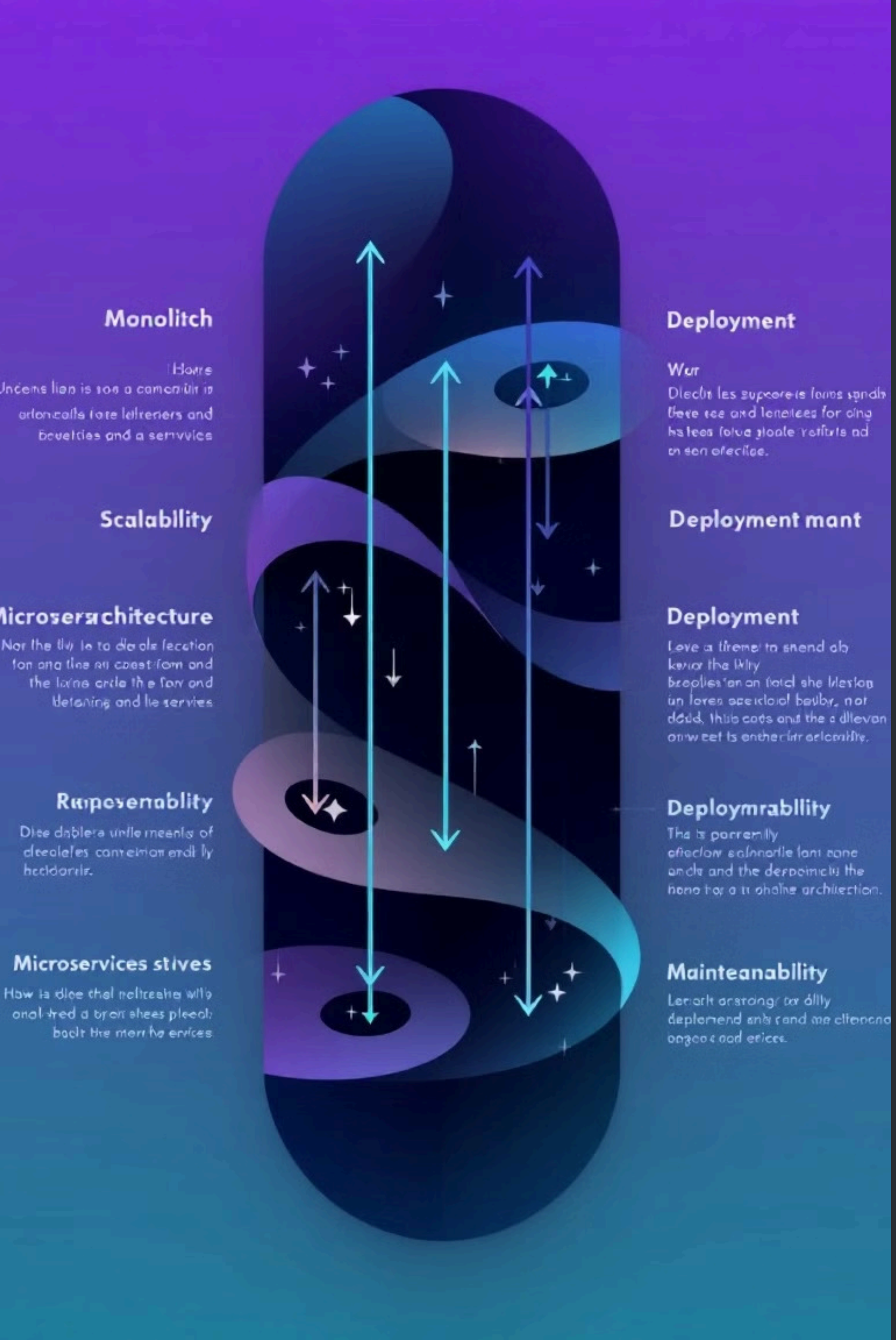
Each microservice manages its own database.

Technology Agnostic

Microservices can use diverse technologies.

Scalability

Services can be scaled independently.



Monolithic vs. Microservices

Feature	Monolithic	Microservices
Scalability	Limited	High
Deployment	Entire app redeployed	Independent services deployable
Technology	Single stack	Multiple technologies
Fault Isolation	Low	High
Codebase Complexity	Simple (at the start)	Complex (requires governance)

Benefits

1. **Faster Development**

   Independent teams work concurrently.

2. **Improved Tolerance**

   Single failure doesn't crash the system.

3. **Easier Maintenance**

   Smaller codebases simplify debugging.

4. **Better Resource Utilization**

   Can scale individual services independently.

Challenges

Complexity

Managing services increases system complexity. / Deployment Complexity

Data Consistency

Databases require careful synchronization.

Communication

Inter-service calls introduce latency.

Best Practices

1 **Domain-Driven Design**

Clear boundaries based on business domains.

2 **API Gateway**

Manage and secure service communication.

3 **Containerization**

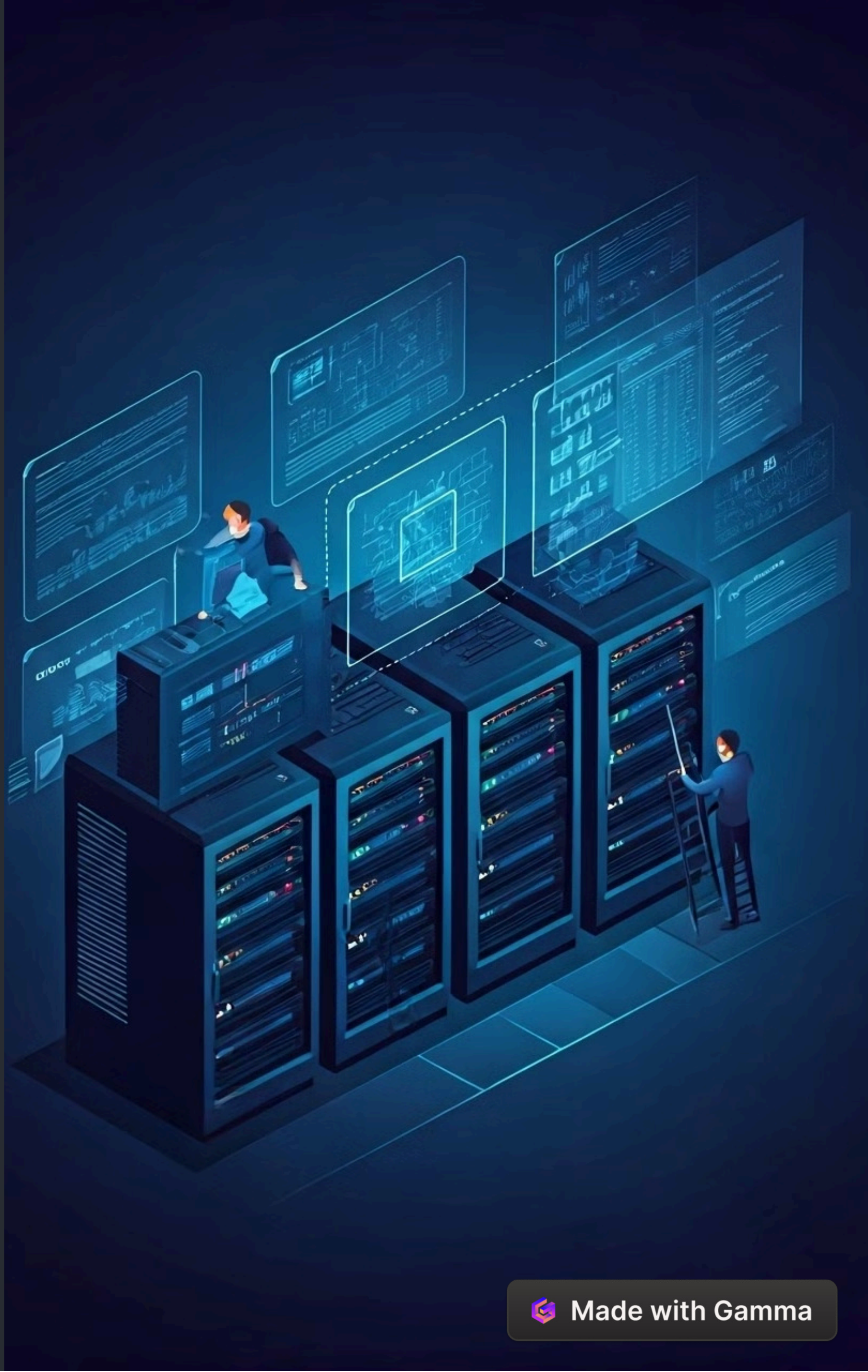
Docker and Kubernetes for deployment.

4 **Observability**

Logging, monitoring, and tracing

5 **Automated CI/CD**

Ensure fast and reliable deployments using Jenkins, GitHub Actions, or GitLab CI/CD.



Tools & Technologies

Containers

Docker

Orchestration

Kubernetes

Communication

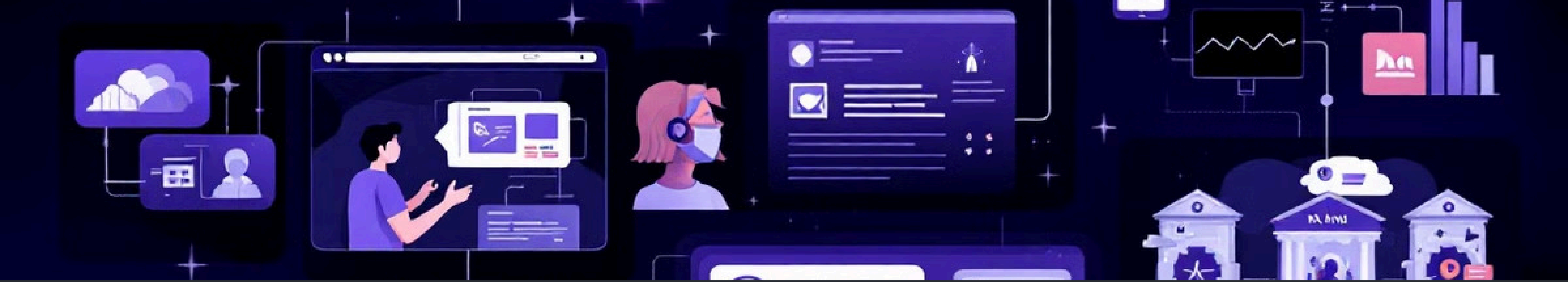
gRPC, Kafka

Monitoring & Logging

Prometheus, Grafana, ELK Stack

Security

OAuth2, OpenID Connect, JWT



Use Cases

1. **E-commerce**

   Orders, payments, and inventory services.

2. **Streaming**

   Video processing and authentication.

3. **FinTech**

   Secure services for transactions.

Case Study: Netflix

Faster Development

High Availability

Efficient Scaling

Netflix transitioned to microservices for scalability. They achieved faster development and deployment.

Conclusion

Microservices offer scalable, resilient applications.

Careful planning and proper tooling are necessary.

Adhere to best practices for success.