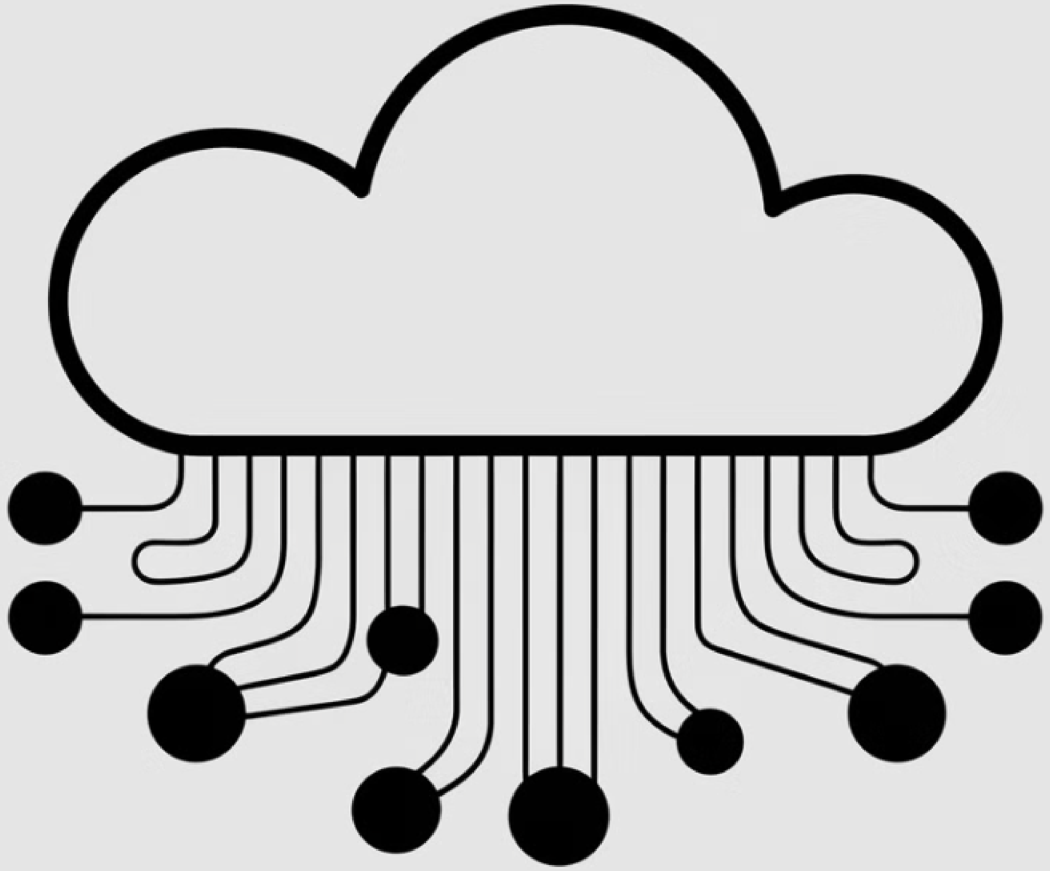


Microservice-Based Architecture in SAP BTP

We'll examine what microservices really are, how they're implemented in SAP BTP, and the real-world benefits and challenges you might encounter when adopting this architecture.

por Mayko Silva

What Are Microservices, Really?

Independent Services

Instead of building one massive application where everything is connected and interdependent, microservice architecture breaks things down into smaller, independent services.

Single Responsibility

Each microservice is focused on doing one thing really well - like handling user authentication, managing product inventory, or processing payments.

API Communication

These services communicate with each other through well-defined APIs, allowing them to work together while remaining independent.

I worked with a large retail client whose monolithic e-commerce system required testing the entire application for any update. After moving to microservices on SAP BTP, they could update just the payment service without touching anything else, saving weeks of testing for each release.

Domain-Driven Boundaries

Customer Accounts

Manages user profiles, authentication, and preferences

Order Management

Handles order creation, processing, and history

Shipping

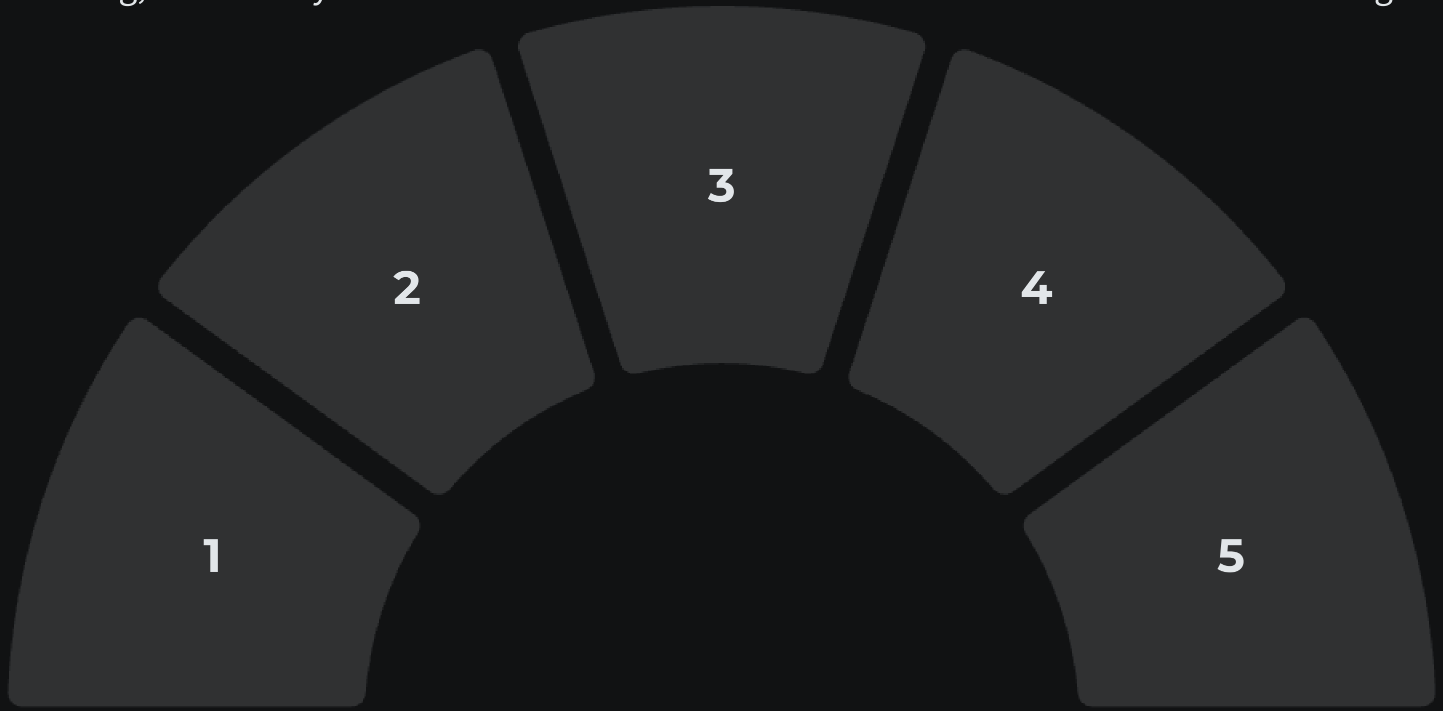
Coordinates delivery logistics and tracking

Product Catalog

Manages all product information, categories, and search functionality

Payment Processing

Handles transactions and financial operations



In domain-driven design, you organize your microservices around business domains or capabilities rather than technical functions. Each service owns its data and the business logic for a specific domain.

I worked with a manufacturing company that initially tried to create microservices based on technical layers. It was a mess! When we reorganized around business domains, teams could work independently and make progress much faster.

Identifying Good Domain Boundaries

Natural Business Divisions

Look for natural divisions in your business processes that can function independently

Independent Scaling

Find areas that need to scale independently based on different usage patterns

Change Patterns

Group together functions that typically change for the same reasons or at the same time

Team Structures

Consider existing team structures and responsibilities when defining boundaries

In SAP BTP, you can implement these domain-based microservices using the Cloud Application Programming Model (CAP) or the Kyma runtime depending on your specific requirements and technical preferences.

Loosely Coupled Components

1

Well-defined APIs

Communicate through clearly specified interfaces that hide implementation details

2

Database Independence

Avoid sharing databases between services to maintain autonomy

3

Asynchronous Communication

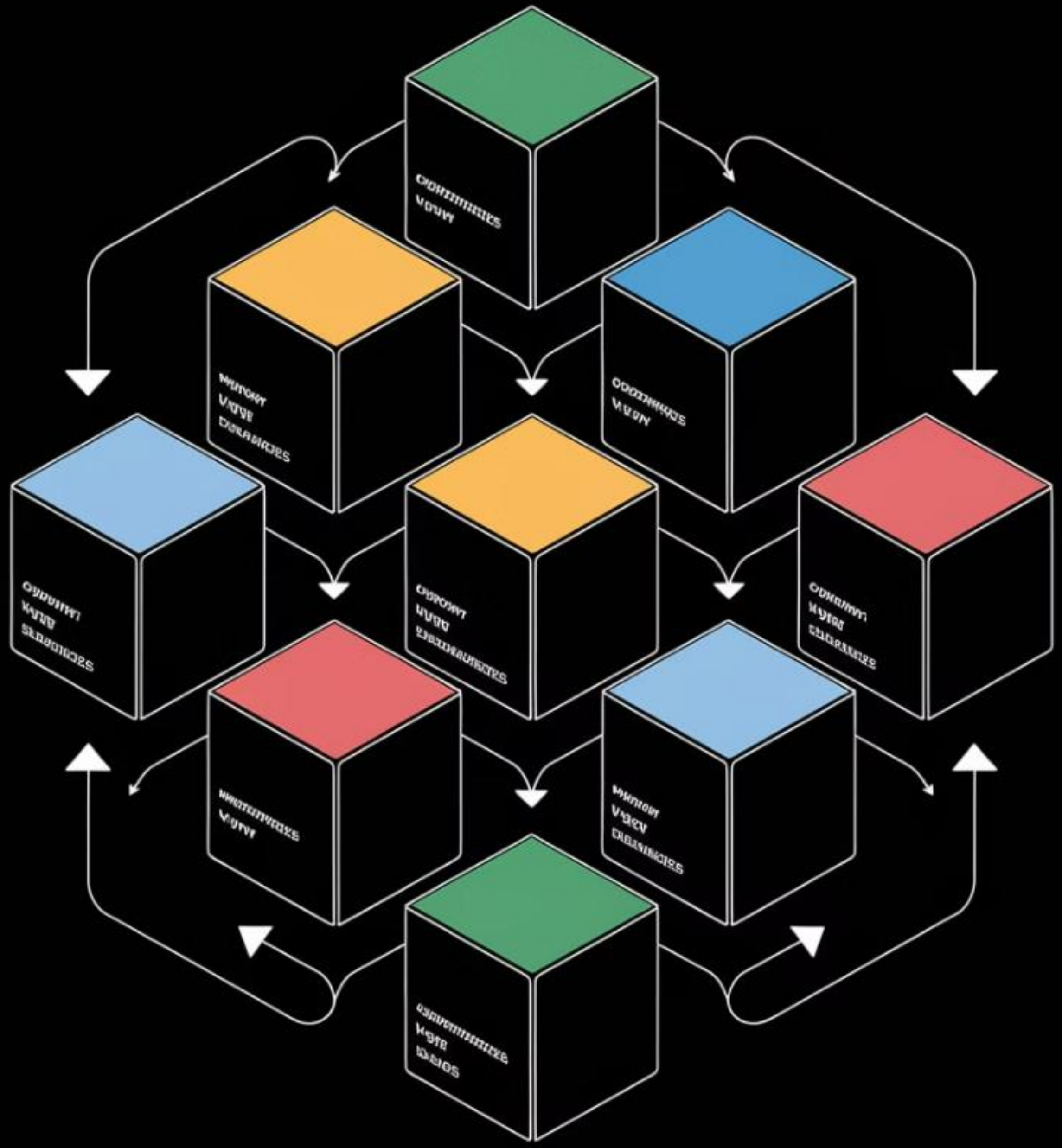
Minimize synchronous communication where possible to reduce dependencies

4

Event-Driven Notifications

Use events to notify other services about changes without tight coupling

I recall a project where two different teams were building microservices that needed to work together. They spent time defining their API contract upfront, allowing them to develop independently for months. When they finally integrated their services, everything worked the first time!



Benefits of Microservice Architecture

Adopting microservice architecture requires significant effort, but the benefits can transform how your organization builds and maintains applications. Let's explore the key advantages that make this architectural approach worth considering.

These benefits directly address many of the pain points organizations face with traditional monolithic applications, particularly as they scale and evolve over time.

Scalability

10x

Peak Handling

Scale specific services during high demand

$$$

Cost Savings

Only scale what needs scaling

24/7

Availability

Maintain service even during scaling events

You can scale individual services based on their specific resource needs. For instance, your payment processing service might need more computing power during sales events, while your product catalog service remains stable.

A client in the financial services industry was able to scale their transaction processing service to handle 10x normal volume during peak periods without scaling anything else - saving a ton on infrastructure costs.

Development Agility

1. **Monolithic Releases**
   Quarterly updates requiring full system testing
2. **Transition Period**
   Moving to microservices with independent deployment pipelines
3. **Microservice Agility**
   Weekly or daily updates to individual services
4. **Continuous Innovation**
   Rapid feature delivery with minimal risk

Different teams can work on different services simultaneously without stepping on each other's toes. This speeds up development and allows for more frequent releases.

I've seen organizations go from quarterly releases to weekly or even daily updates for certain services after adopting microservices, dramatically improving their ability to respond to market changes.

Technology Flexibility

Java Services

Perfect for complex business logic and enterprise integration

Node.js Services

Ideal for lightweight, high-throughput API services

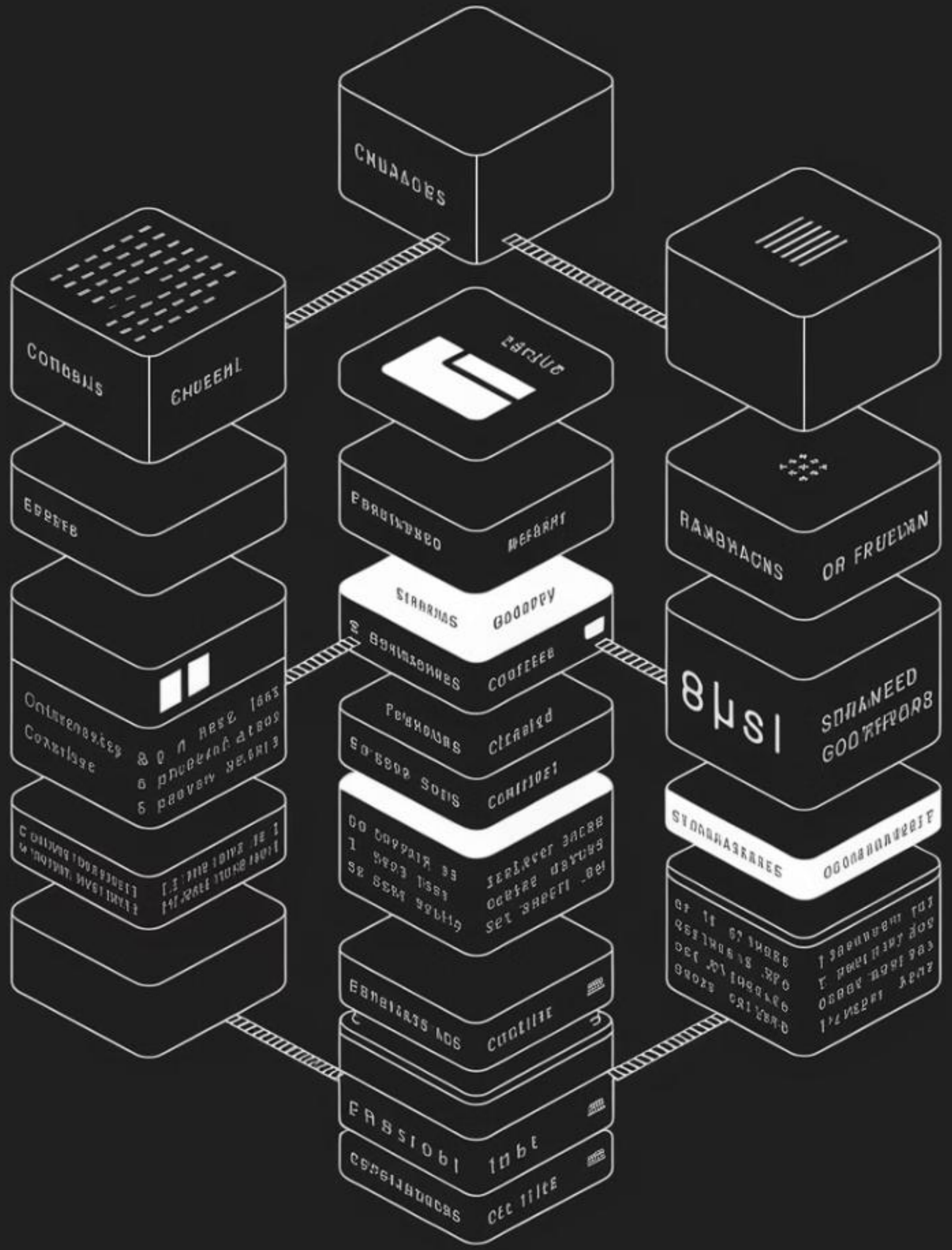
Python Services

Great for data processing and machine learning components

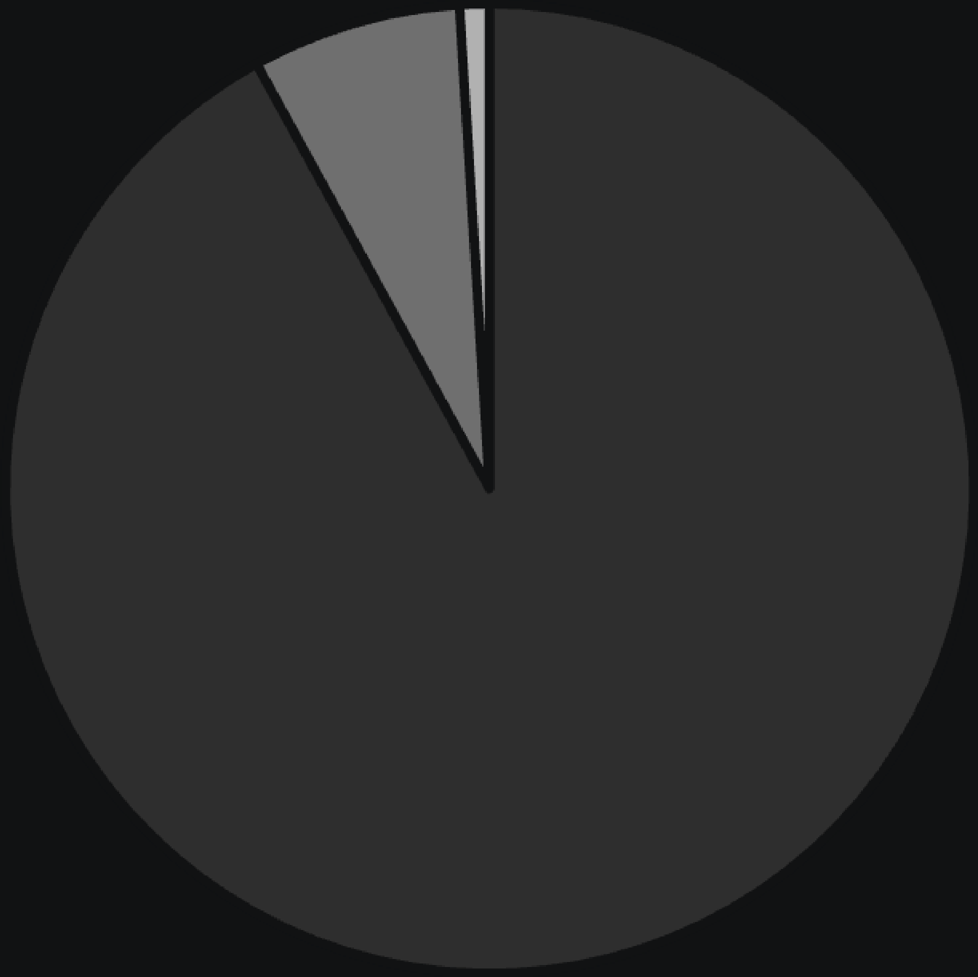
Specialized Databases

Choose the right database for each service's specific needs

Each service can use the technology stack that makes the most sense for its specific function. This flexibility allows teams to leverage the best tools for each job rather than being constrained by a single technology choice.



Resilience

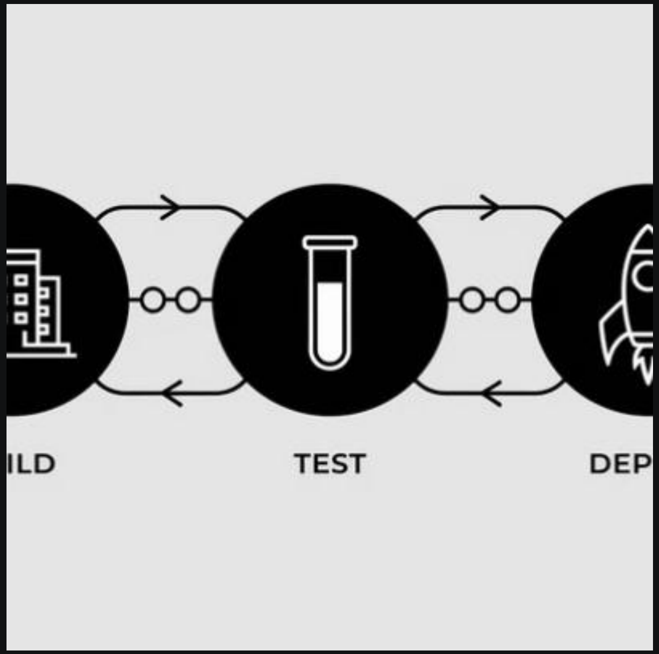


⬛ Fully Operational ▩ Partial Degradation ⬜ Complete Outage

If one service fails, the others can continue operating. This prevents the entire application from crashing due to a problem in one area, significantly improving overall system reliability.

With proper circuit breaker patterns and fallback mechanisms, microservices can gracefully handle failures in dependent services, providing a degraded but functional experience rather than a complete outage.

Easier Maintenance



Smaller codebases are easier to understand, test, and maintain. New team members can get up to speed more quickly when they only need to understand a single service rather than an entire monolithic application.

This modularity also makes it easier to refactor or even completely rewrite individual services when necessary, without disrupting the entire system.

Challenges of Microservice Architecture

While microservices offer significant benefits, they also introduce new challenges that organizations must be prepared to address. Understanding these challenges is crucial for successful implementation.

Many of these challenges stem from the distributed nature of microservices and require different approaches than those used in monolithic applications. Let's explore the key challenges you might face.

Distributed System Complexity

Network Reliability

Services must handle network failures gracefully

Data Consistency

Maintaining consistent data across services

Transaction Management

Coordinating operations across multiple services

Latency Concerns

Managing performance with inter-service communication

Microservices create a distributed system, which introduces complexity in terms of network communication, data consistency, and transaction management.

I worked with a company that had to implement the Saga pattern to manage transactions across multiple services when they realized that traditional database transactions wouldn't work in their distributed architecture.

Operational Overhead

1. **Deployment Complexity**

   Managing deployments for dozens of services

2. **Monitoring Challenges**

   Tracking performance across distributed systems

3. **Configuration Management**

   Maintaining consistent configurations

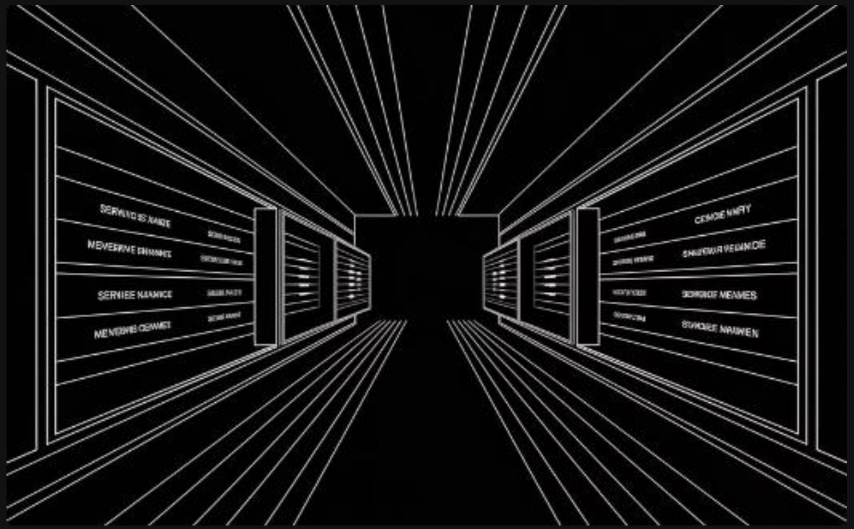
4. **Resource Allocation**

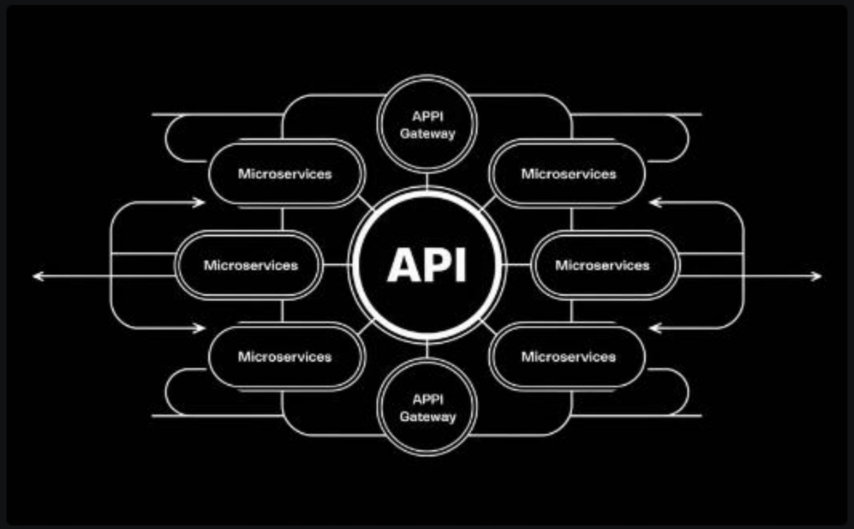
   Efficiently distributing computing resources

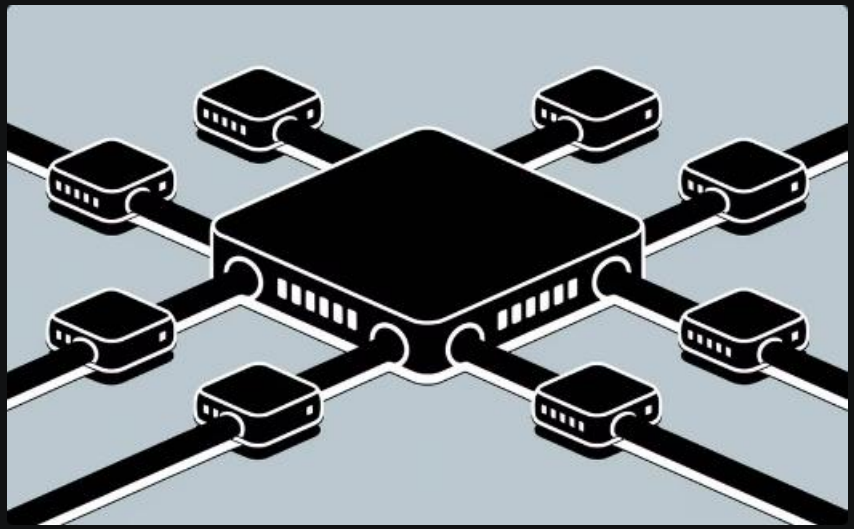
More services mean more things to deploy, monitor, and maintain. This requires robust DevOps practices and tools to manage effectively, often necessitating specialized skills and tooling.

Organizations typically need to invest in automation and monitoring solutions to handle the increased operational complexity of a microservice architecture.

Service Discovery and Communication







Service Registry

Central repository where services register their location and capabilities, allowing other services to find them dynamically.

API Gateway

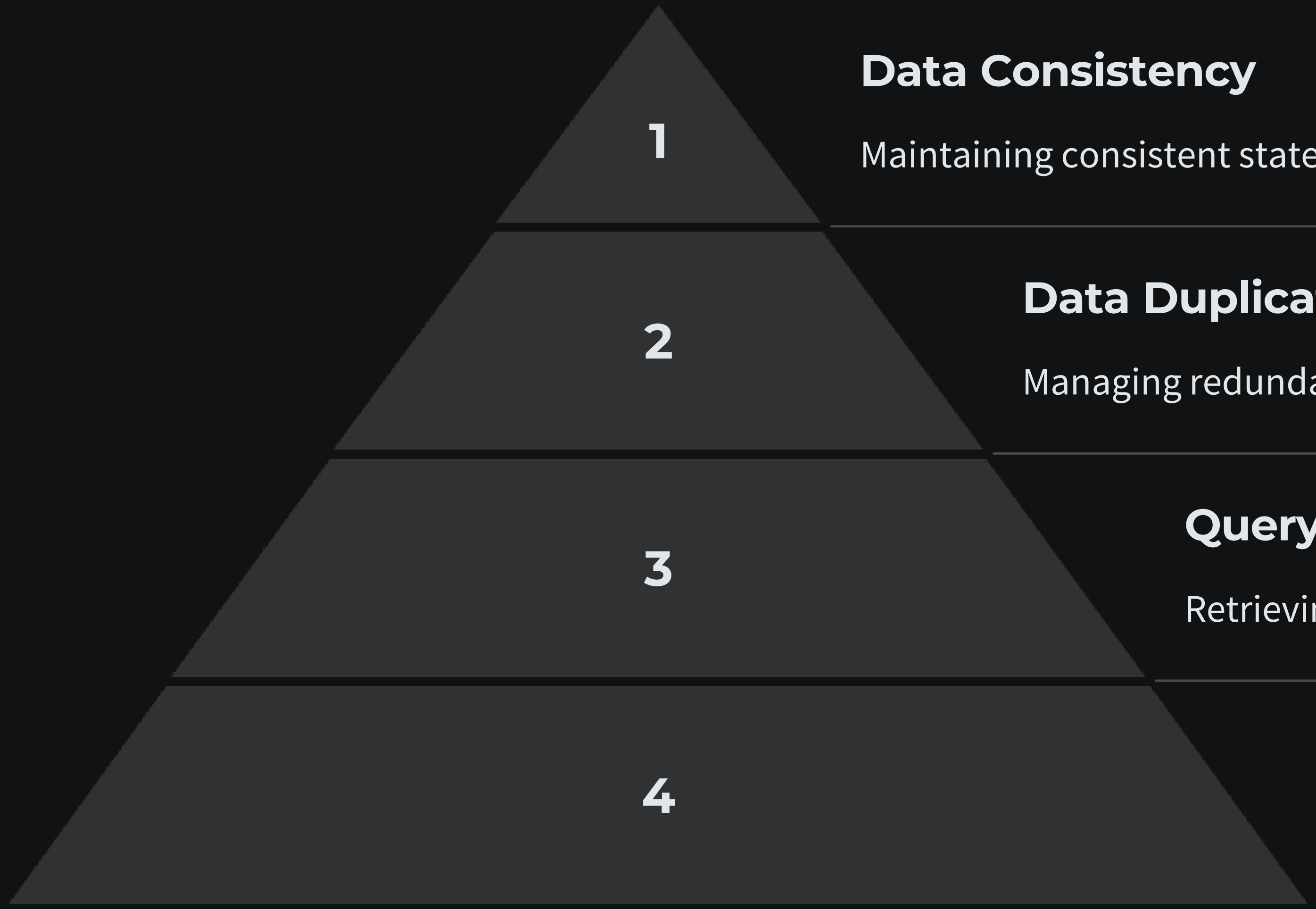
Central entry point that routes requests to appropriate services, handling cross-cutting concerns like authentication and rate limiting.

Load Balancing

Distributing requests across multiple instances of the same service to improve reliability and performance.

Services need to find and communicate with each other reliably, often requiring additional infrastructure components like service registries, API gateways, and load balancers.

Data Management



Data Consistency

Maintaining consistent state across services

Data Duplication

Managing redundant data across services

Query Complexity

Retrieving data that spans multiple services

Database Per Service

Each service with its own data store

Each service typically has its own database, which can make it challenging to maintain data consistency and perform queries across services. This often requires implementing patterns like CQRS (Command Query Responsibility Segregation) or event sourcing.

A healthcare client struggled with this until we implemented an event-driven approach to keep data synchronized between services, ensuring critical patient information remained consistent across their system.

Testing Complexity

1 **Unit Testing**

Testing individual services in isolation becomes simpler, but doesn't verify integration points.

2 **Integration Testing**

Testing how services work together becomes more complex, requiring specialized approaches.

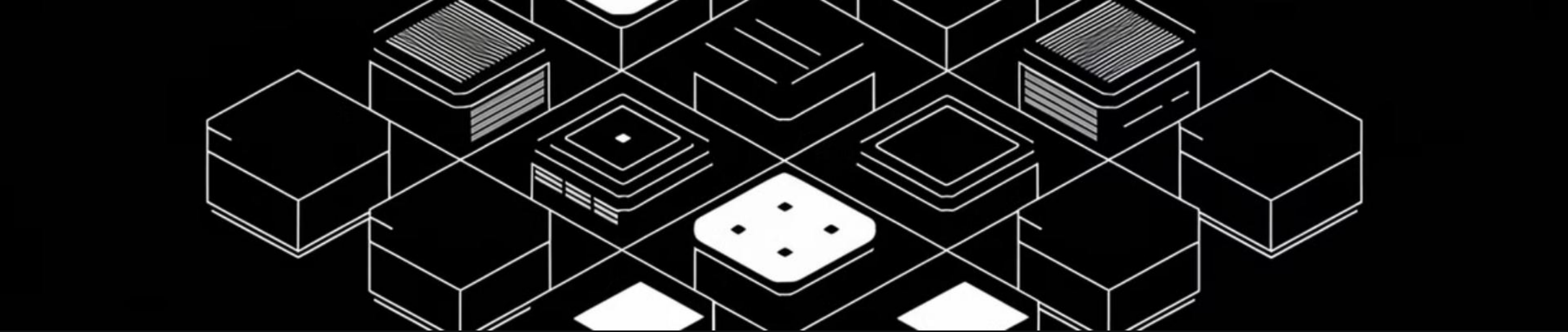
3 **Contract Testing**

Verifying that services adhere to their API contracts is essential for system stability.

4 **End-to-End Testing**

Testing complete user journeys across multiple services requires sophisticated test environments.

Testing interactions between services is more complex than testing a monolithic application. You need strategies for integration testing and contract testing to ensure services work together correctly while maintaining independence.



Implementing Microservices in SAP BTP

SAP Business Technology Platform provides a comprehensive suite of services and tools specifically designed to address the challenges of microservice architecture. These capabilities make BTP an ideal platform for developing, deploying, and managing microservice-based applications.

Let's explore the key components that SAP BTP offers to support your microservice implementation across development, operations, and communication domains.

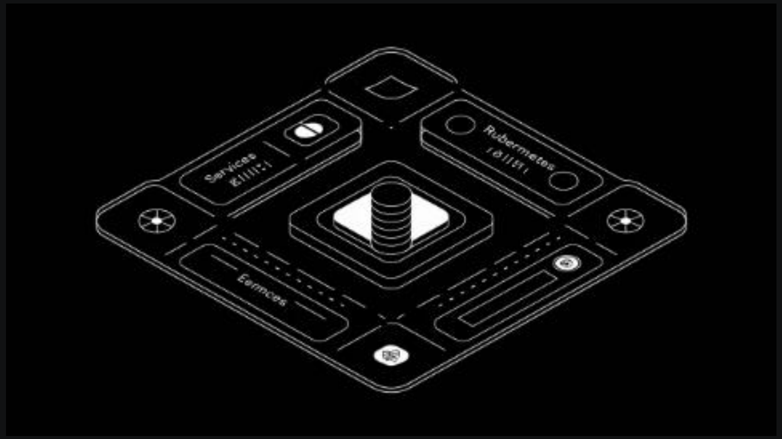
For Development

Cloud Application Programming Model (CAP)

A framework that simplifies the development of microservices by providing standard patterns for data models, services, and APIs. CAP accelerates development through code generation and built-in best practices.

SAP Business Application Studio

A cloud-based development environment specifically designed for building applications on SAP BTP. It provides specialized tools and templates for microservice development.



Kyma Runtime

A fully managed Kubernetes environment that enables containerized microservice deployment with advanced features like service mesh, serverless functions, and API gateways.

For Operations

SAP BTP Cockpit

Centralized monitoring and management console for all your microservices and platform services, providing visibility into performance, usage, and health.

Cloud Foundry Runtime

Platform-as-a-Service environment that simplifies deployment and scaling of microservices without managing the underlying infrastructure.

CI/CD Services

Integrated continuous integration and deployment tools that automate testing, building, and deploying your microservices to ensure quality and consistency.

These operational tools work together to reduce the management overhead typically associated with microservice architectures, allowing your teams to focus more on development and less on operations.

For Communication

Destination Service

Manages connection details to external systems and services, abstracting connectivity complexity from your microservices and providing secure credential management.

SAP API Management

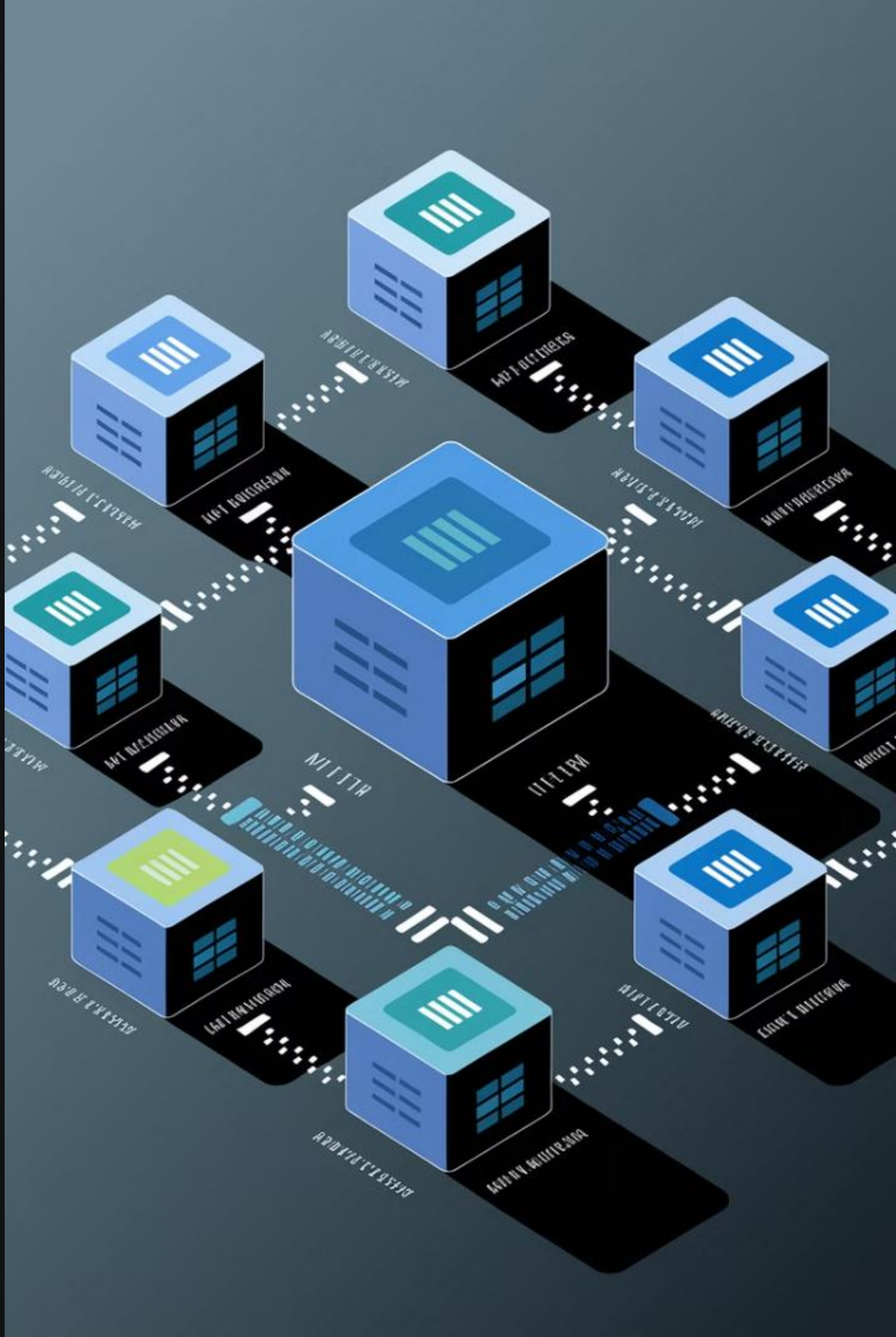
Provides tools for creating, publishing, and managing APIs that connect your microservices, including features for security, throttling, and monitoring.

SAP Integration Suite

Offers comprehensive integration capabilities for connecting microservices with each other and with external systems through various protocols and formats.

SAP Event Mesh

Implements event-driven architecture patterns by providing reliable, scalable event distribution between microservices using publish-subscribe mechanisms.



Real-World Example: Order Processing System

1

Order Creation

Order Service creates a new order and checks inventory availability

2

Iventory Service

Checks and updates product availability

3

Customer Service

Handles customer information

4

Payment Service

Processes payments

5

Shipping Service

Arranges product delivery

6

Notification Service

Sends updates to customers

In this order processing system, we've broken functionality into six distinct services: Order, Inventory, Customer, Payment, Shipping, and Notification. Each service has its own database and can be developed, deployed, and scaled independently.

This architecture allows teams to focus on specific business domains while the event-driven communication pattern ensures loose coupling between services.

Best Practices for Microservice Success

1 **Start small**

Don't try to break everything into microservices at once.

2 **Define service boundaries**

Invest time in defining good service boundaries upfront.

3 **Standardize communication**

Standardize on communication patterns between services.

4 **Implement monitoring**

Implement robust monitoring and logging.

5 **Automate processes**

Automate testing and deployment.

6 **Handle data consistency**

Have a strategy for handling data consistency.

7 **Document APIs**

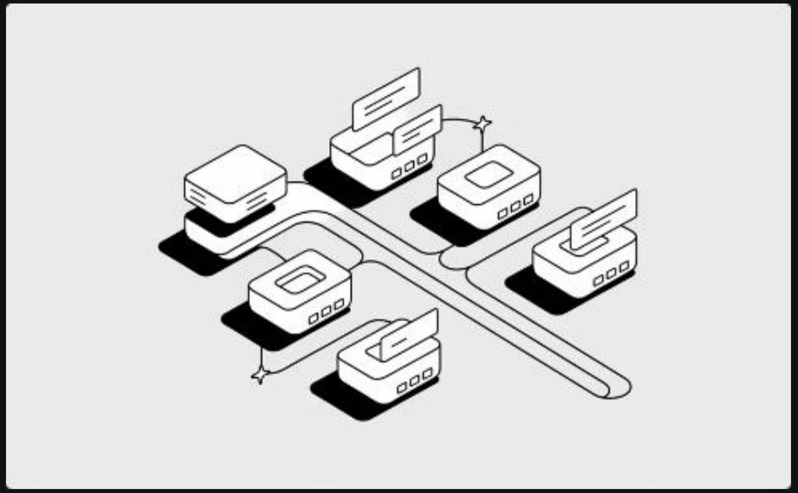
Document your APIs thoroughly.

8 **Create service templates**

Create service templates to ensure consistency.

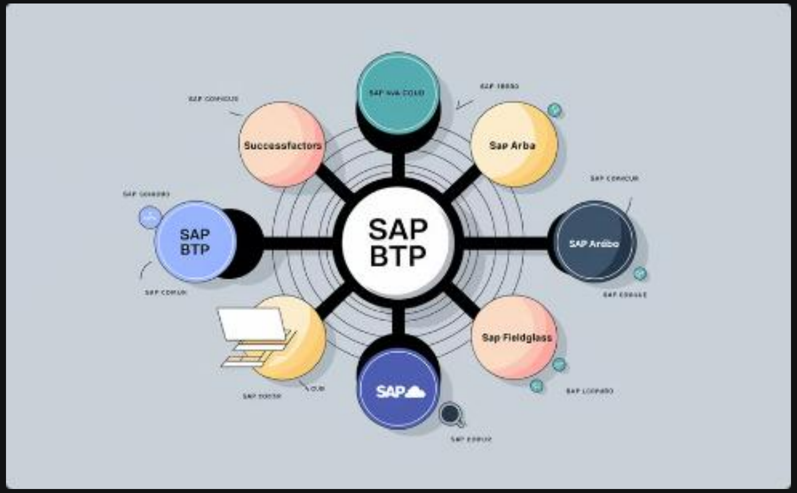
por Mayko Silva

Conclusion



Flexibility & Scalability

Microservice architecture offers powerful benefits for building flexible, scalable applications on SAP BTP that can evolve with changing business needs.



Comprehensive Platform

SAP BTP provides many services and tools to help address the challenges of microservices, from development to operations.

Thoughtful Approach

The key is to approach microservices thoughtfully and incrementally, rather than rushing to decompose everything at once.

I've seen organizations transform their ability to deliver new features and respond to market changes by adopting microservices on SAP BTP. With careful planning and the right implementation strategy, you can realize these benefits while minimizing the challenges.