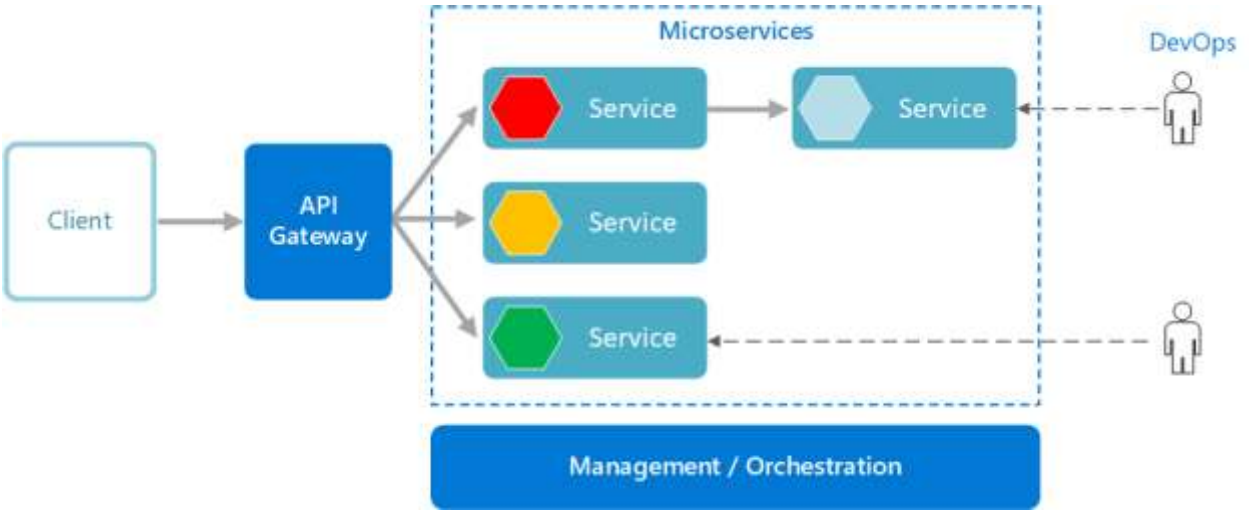
Microservice Architecture



Microservices are small, independent, and loosely coupled. A single small team of developers can write and maintain a service.

Each service is a separate codebase, which can be managed by a small development team.

Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.

Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.

Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.

Advantages

1. Because microservices are deployed independently, it's easier to manage bug fixes and feature releases. You can update a service without redeploying the entire application, and roll back an update if something goes wrong
2. **Small, focused teams**. A microservice should be small enough that a single feature team can build, test, and deploy it. Small team sizes promote greater agility. Large teams tend be less productive, because communication is slower, management overhead goes up, and agility diminishes.
3. Scalability. Services can be scaled independently, letting you scale out subsystems that require more resources, without scaling out the entire application.
4. Data isolation. It is much easier to perform schema updates, because only a single microservice is affected.

Best Practices

1. Model services around the business domain.
2. Decentralize everything. Individual teams are responsible for designing and building services. Avoid sharing code or data schemas.
3. Avoid coupling between services. Causes of coupling include shared database schemas and rigid communication protocols.

https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices