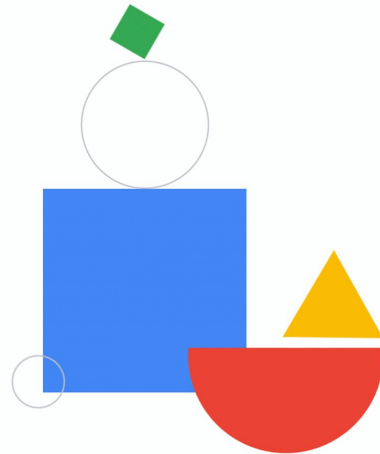
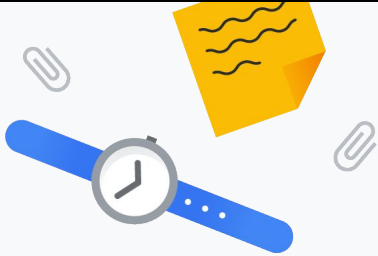


Service Orchestration and Choreography on Google Cloud

Module 1: Introduction to Microservices



Welcome to module 1 of Service Orchestration and Choreography on Google Cloud:
Introduction to Microservices.



Module agenda



- 01 What are microservices?
- 02 Benefits of microservices architectures
- 03 Challenges of microservices architectures

In this module, you learn about microservices, an architectural style for developing applications.

We discuss the benefits you gain by developing your applications using microservices.

We also discuss some challenges that a microservices architecture can introduce.



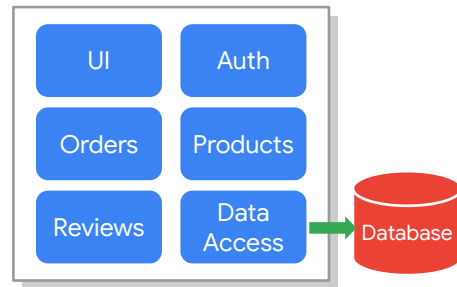
**What are
microservices?**

Today, many new enterprise applications are being designed using a microservices architecture. What are microservices?

Traditional monolithic applications

A monolithic application is:

- ✓ A large, self-contained application.
- ✓ Complex.
- ✓ Tightly coupled.



First, we should understand how applications were traditionally designed. Early enterprise applications were developed as large, self-contained applications. These applications included the user interface, business logic, and data access code, with data persisted in a large, relational database.

These applications were designed to handle many tasks, and the codebase was necessarily complex. Each major change to the application tended to make the codebase even more complex.

And, because the code was all in a single application, the application code was often tightly coupled. This interdependent code was hard to maintain, making it difficult to fix bugs without introducing new ones.

We now call these monolithic applications, or monoliths.

Service-Oriented Architecture (SOA)



- ✓ Architectural style designed around reusable software components called services
- ✓ Each service executes a discrete business function
- ✓ Communication between services by messaging over defined interfaces
- ✓ Enterprise scope

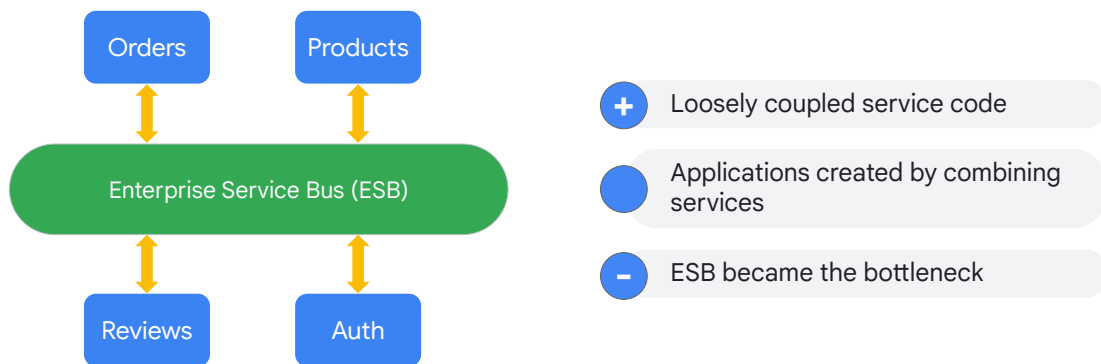
Service-Oriented Architecture, or SOA, was an attempt to solve the challenges of monolithic applications.

SOA is an architectural style that focuses on building reusable software components called services.

Each service in a service-oriented architecture should execute a discrete business function, and communication between services was implemented using messaging over defined service interfaces.

SOA was typically implemented at an enterprise level. Organizations would map business activities into services, and mandate interoperability and discoverability standards for their services.

In practice, SOA produced mixed results



SOA did provide tangible benefits, but it typically produced mixed results.

One clear benefit was that services were smaller and more loosely coupled than in large monolithic applications. Smaller services often led to smaller development teams focused on a single service and a smaller problem domain, which could improve efficiency.

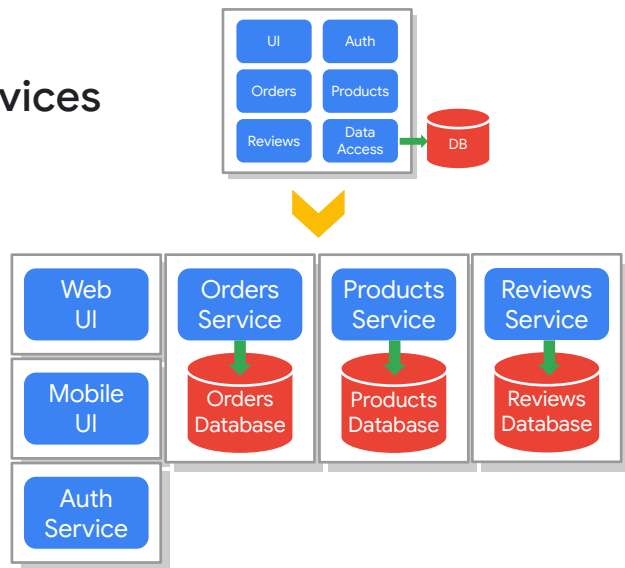
Service reuse was also a large focus of SOA. Applications were created by combining services. The messages between services were handled by a messaging middleware component called the Enterprise Service Bus, or ESB. By managing connectivity, security, and message routing and transformation, the ESB enabled applications to be integrated, even for applications outside the organization.

Applications would connect to the ESB, and the ESB would transform protocols, route messages between services, and transform data.

Though SOA reduced complexity in the service code, the complexity was typically shifted to ESB integrations. ESB integrations became the bottleneck in successfully launching and updating applications. The ESB was typically managed by a central team, and ESB integration work could face significant delays because all application and service teams needed ESB work. Changing an integration for one application might destabilize other applications using the integration. Even updates to the ESB software itself could break existing integrations, so ESB updates required significant testing.

Decompose monolithic applications into microservices

- Microservices are:
- ✓ Separate services, limited in scope.
 - ✓ Loosely coupled.



Microservices are an alternative, decentralized approach to decomposing applications into services.

Microservices are separate services, limited in scope. In this example, each of the business domains (orders, products, and reviews) is in its own microservice with its own database. A microservice specifies an interface, typically an API, that is used by other services when calling the operations of the microservice.

The separation of microservices tends to lead to loose coupling between the microservices. Loosely coupled services are easier to maintain, update, and deploy.

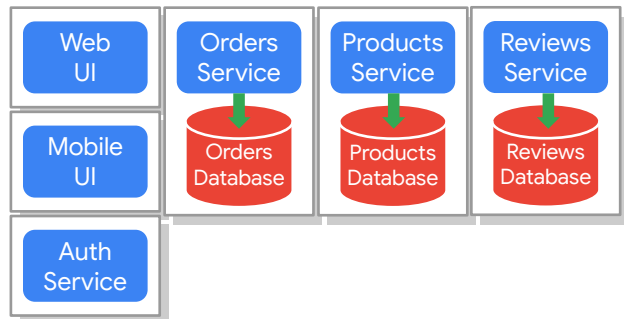
Microservices architecture:

<https://cloud.google.com/learn/what-is-microservices-architecture>

Design microservices from the start

Use microservices from the start if:

- ✓ You understand the problem domain.
- ✓ You require agile service delivery.
- ✓ You plan to grow the team.



When you're starting to design a new application, you may decide to start with microservices instead of a monolith.

One of the most difficult parts of architecting a new microservices application is designing the service boundaries. If you're designing an application and do not have expertise in the problem domain, choosing the separate services to create might be difficult. If you start with a monolithic application, you can build your application before you fully understand how to separate the services. You can migrate to a microservices architecture later as you gain more experience with the problem domain.

It's easier to deliver microservices in an agile fashion than it is to deliver monoliths. If you require the ability to quickly release changes to your services, it probably makes sense to start with microservices.

If you plan to grow the size of your team, microservices allow new team members to focus on smaller parts of the overall application. Team members aren't required to learn a large monolithic codebase. A microservices application provides natural service boundaries that allow for smaller teams, with each team having reduced scope.

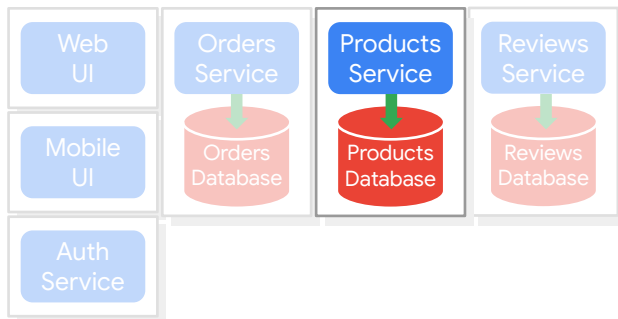
When starting with a monolith, design your application to be modular so it's easier to migrate to microservices in the future.



Benefits of microservices architectures

Developing your applications as microservices has many benefits.

Microservices are simpler to develop



- ✓ Smaller codebase
- ✓ Easier to understand and update
- ✓ Easier to unit test
- ✓ Separately deployable
- ✓ More agile

One obvious benefit of microservices is that each microservice is simpler than the large, monolithic application.

Each microservice has a smaller and simpler codebase, which generally allows for a single, small team to focus on the internal details of that microservice.

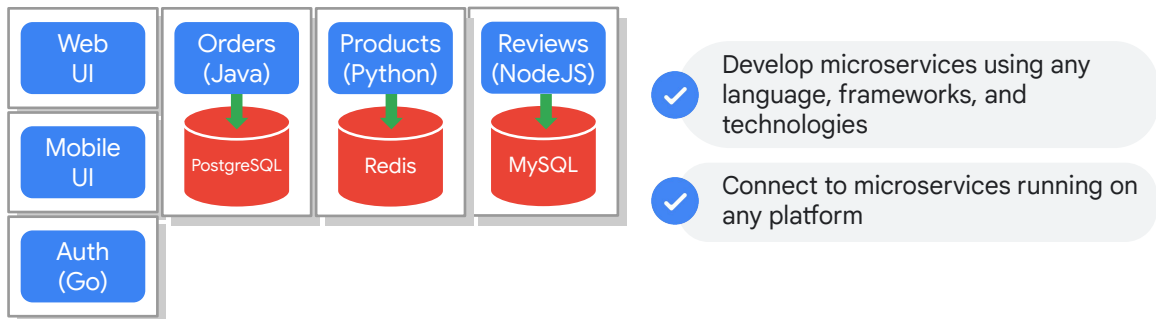
It's easier for the members of that team to understand the microservice and update it without causing issues in other parts of the application.

Microservices are typically easier to unit test, because there are clear boundaries between different services.

Microservices are also separately deployable, which allows teams to update their microservices on their own schedules. Other microservices are only affected when a breaking change is made to the interface of a microservice.

These traits of microservices lead to more agile development, because microservices can be separately updated and deployed without affecting other services.

Microservices can use any programming languages and technologies

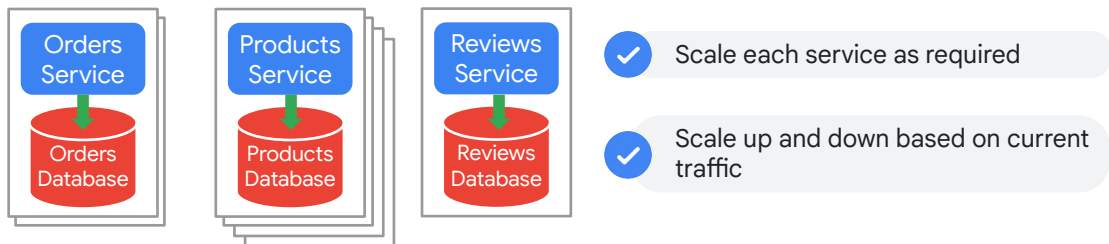


Another benefit is that microservices can use different programming languages and technologies.

Microservices connect using an API interface. The programming languages, frameworks, and technologies used by the microservice code do not affect calling services. A team can choose the languages and technologies that best suit their service.

Microservices running on one platform can also connect to microservices running on a different platform. Microservices, which typically connect using HTTP APIs, can call each other without worrying about where the service is located.

Microservices can be scaled separately



A third benefit of microservices is the ability to separately scale microservices.

Each microservice can be scaled separately, dedicating more resources only to those services that require more capacity.

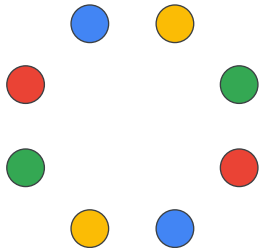
Microservices are typically easy to scale up and down based on fluctuations in traffic. Instead of building the infrastructure required when the load is highest, the infrastructure, and therefore cost, can be optimized based on the current traffic requirements.



Challenges of microservices architectures

The microservices architecture is a common architecture for new enterprise applications being developed today. However, microservices architectures also provide significant challenges.

More services means more deployments and more points of failure



- ✓ More operational skill is required
- ✓ Automated builds, testing, and deployments are vital
- ✓ Consistent logging, reporting, security, and authorization must be maintained

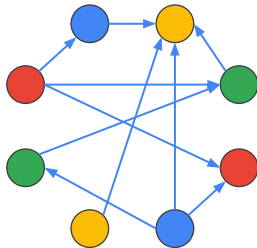
In a microservices architecture, each microservice tends to be simpler and easier to understand than the monolithic application. Each service can be developed, deployed, and tested separately.

However, having more deployable entities creates a greater operational burden for an organization. The operations team must manage tens, hundreds, or thousands of microservices.

Automated builds, testing, and deployments are crucial to maintaining the health and efficiency of your applications and your operational team.

With so many services, it's also important to maintain consistent logging, reporting, security, and authorization for your services.

Each microservice is simpler, but communication between services can be complicated



- ✓ Spider web of point-to-point communication
- ✓ Increased communication latency
- ✓ Integration testing is difficult
- ✓ Debugging is challenging

A microservices architecture includes the communication between microservices, which can be complex.

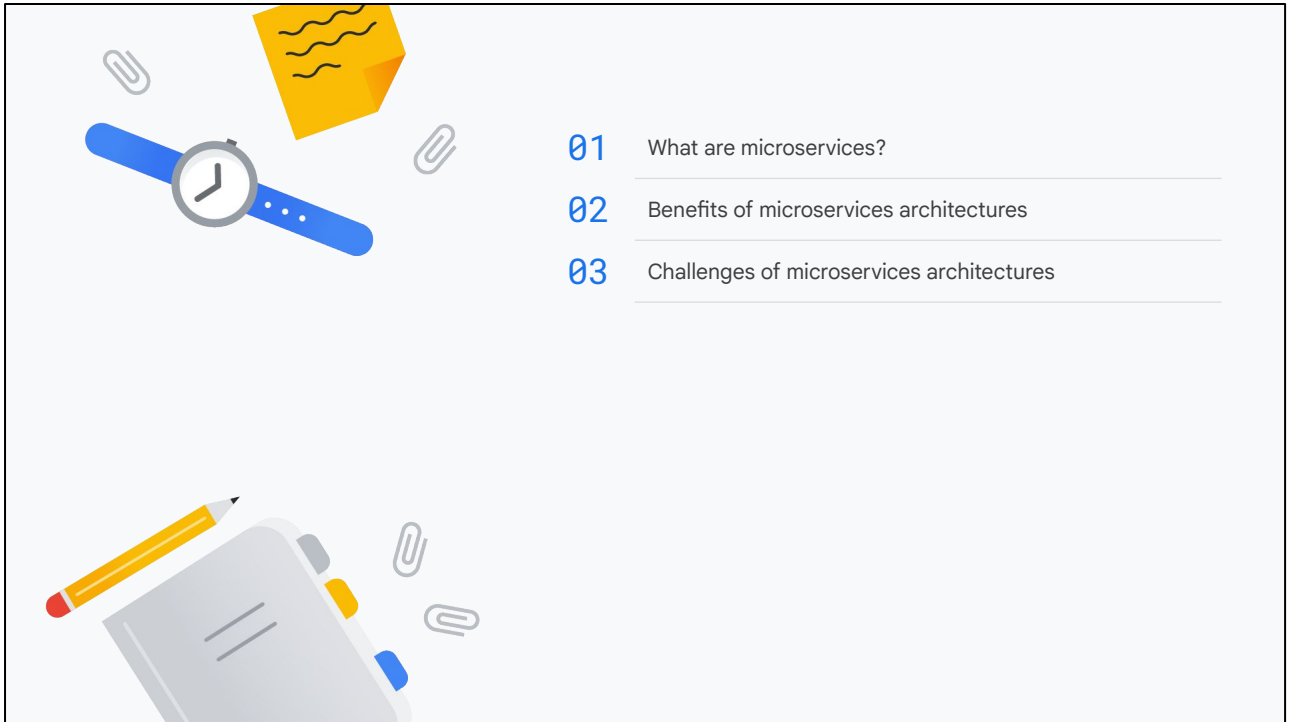
If you do not design your systems well, it can be difficult to understand the "spider web" of communication between microservices.

Microservices can also introduce communication latency. For a monolith, calls between components are typically in the same process running on the same hardware. With microservices, calls between services happen across the network, which can be thousands of times slower. When a business operation requires many microservice calls, the latency can be significant.

While unit testing for each microservice can be straightforward, integration testing is typically more challenging. The distributed nature of microservices often means that testing the entire system requires modeling the entire production deployment.

Debugging a microservices architecture can also be difficult. If an application consists of many microservices, and each microservice creates its own logs, tracing calls that span many microservices can be challenging.

Building microservices requires a commitment to automation and operational excellence. The benefits of microservices generally outweigh the challenges, and we will see how service orchestration and choreography can reduce the complexity of microservice-based applications.



This is the end of the first module of the course "Service Orchestration and Choreography in Google Cloud."

In this module, "Introduction to Microservices," you learned about microservices, which were compared to monolithic and SOA applications. You also learned about the benefits and challenges of using a microservices architecture.