Microservices and Microservices Architecture

Learn the pros and cons of microservices and how they differ from monoliths.

What are Microservices?

The term “Microservices” is a modern term used to describe a traditional “separation of concerns” pattern within a distributed, networked project. Microservices is an idea that follows an old fundamental unix philosophy “of small, sharp tools”. Both concepts build on another foundational computer science pattern of “composition” which means that complex systems are the sum of lower level composable entities.

READ ON BELOW

Microservices articles

ARTICLE

3 Secrets to Building Microservices

The first of rule of building microservices is that you shouldn't start for a green field project. Business requirements will change while you're building.

ARTICLE

What are containers?

Learn what containers are, how they work, and the different container platforms available.

TUTORIAL

Continuous Deployment Tutorial

This tutorial will show you how to get started with continuous deployment with Bitbucket Pipelines.

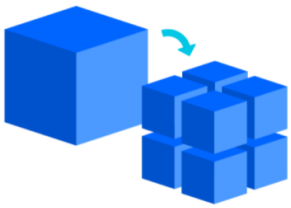
Try this tutorial →

[CONTINUED]

Composition occurs through all layers of a software project. At the lowest level, the “unit level”, individual independent code functions interact with each other over a shared interface to create collections or “libraries” of code. At the operating system shell level, shell commands can be composed to created a pipeline of higher-level functionality. Microservices are a level of composition that happens between Web Services. A Microservice is a web service that is responsible for one piece of domain logic. Microservices interact with each other via simple network protocols like REST to complete actions but they have no knowledge of how other services work internally. This harmonious interaction between microservices is a microservice architecture.

Microservices architecture or (MSA) has been getting a lot of attention as software teams are looking for new ways of improving their release workflows. Amazon, Netflix and Ebay are among the companies that are openly embracing this way of building software and they've contributed back to the community by publishing their own experience and developing tools that can help others to adopt.

> The guiding principle of microservices is to build an application by splitting its business components into small services that can be deployed and operated independently from each other.



Developers can then organise in smaller teams specialising on different services, with different stacks and decoupled deployments. This separation of concerns and decoupled independent function, enables streamlined agile software development practices like continuous delivery and integration.

Service Oriented Architecture (SOA) Vs Microservices

Service Oriented Architecture and Microservices are two types of higher order, web service architectures. Microservices can be thought of as a lite version of SOA. The distinction between the two architecture types is the bureaucratic classification of service types. SOA has 4 basic service types: Business, Enterprise, Application, and Infrastructure services. These types define the related domain specific responsibility of the underlying services. Comparatively, Microservices only have two service types; Functional and Infrastructure.

Both architectures share the same set of standards at different layers of an enterprise. The existence of MSA comes down to the success of SOA pattern. Hence, MSA pattern is a subset of SOA. Here the main focus is on the runtime autonomy of each service.

Monolith Application vs Microservices

Microservices decouple major business domain specific concerns into separate independent code bases. A Monolithic application architecture can be thought of as the inverse of Microservices. A Monolith is one code base that couples all of the business concerns together. Monoliths can be convenient early on in a projects life for ease of code management cognitive overhead, and deployment. This allows everything in the monolith to be released at once.

Many projects initially start out as a Monolith and then evolve into a Microservice architecture. As new features are added to a Monolith it may start to become cumbersome to have many developers working on a singular codebase. Code conflicts become more frequent and the risk of updates to one feature introducing bugs in an unrelated feature increases. When these undesirable patterns arise, it may be time for a discussion on a migration to Microservices.

How Microservices Architecture Works

Consider a hypothetical Ecommerce software project as an example. There are some clearly defined domain specific business features. Ecommerce sites have an authentication system for user login, logouts. A shopping cart to persist a list of products the user is interested in. A billing system allows users to pay for their purchases.

In a Microservice Architecture these example business domains would be independent services. Take the billing system as a specific example. Depending on company employee count there may be a dedicated “billing team” that owns the development and quality assurance of this billing microservice. The billing Microservice would have its own release schedule and deployment playbooks. The billing service would provide a documented and versioned API so other services could communicate and utilize it’s functionality.

Pros & Cons of Microservices

+ Horizontal scaling

Microservices are distributed by design and can be deployed in clusters. This enables dynamic horizontal scaling across the service boundaries. If a Microservice is maxing out its capacity for load, new instances of that service can rapidly be deployed to the accompanying cluster to help relieve the pressure.

+ Independent Team Execution

Microservice ownership teams can operate independently of other feature teams in the organization. This allows for more rapid execution and delivery of new functionality.

+ Deeper focus on quality

The separation of business concerns into independent microservices ensures that the service team owning that service is focused on the complete quality deliverable.

- Exponential Infrastructure Costs

Each new microservice an organization adds to its production deployment comes with its own cost of test suite, deployment playbooks, hosting infrastructure, monitoring tools and more.

- Added Organizational Overhead

An added level of communication and collaboration is needed to coordinate updates and interfaces between microservices architecture teams.

- Development environment complexity

When a project is broken into multiple Microservices it adds a challenge of reproducing the distributed architecture during local development setup.

Future of Microservices

Containerization and the deployment of containers, is a new pattern of distributed infrastructure. Tools like Docker and Kubernetes are used to package up a service into a complete "Container" which can be rapidly deployed and discarded. These new infrastructure tools are complementary to the Microservices architecture. Microservices can be containerized and easily deployed and managed using a container management system.

The adoption of microservices should be seen as a journey rather than the immediate goal for a team. Start small to understand the technical requirements of a distributed system, how to fail gracefully and scale individual components. Then you can gradually extract more and more services as you gain experience and knowledge.

The microservice architecture is still fairly young but it's a promising way of developing applications and it's definitely worth looking into but remember that it might not (yet) be a good fit for your team.

CLAIRE MAYNARD

Claire is an Atlassian marketing veteran who's worked across growth, performance, and product marketing throughout her tenure. Currently she drives brand, content, and go-to-marketing strategy for Confluence Cloud. Outside work, you can find Claire surfing, running, or trying out new restaurants in San Francisco or new cities across the globe.

TUTORIAL

Tips for scripting tasks with Bitbucket Pipelines

Learn our five tips for automating and scripting manual tasks with Bitbucket Pipelines.

Try this tutorial →

UP NEXT

3 Secrets to Building Microservices

The first of rule of building microservices is that you shouldn't start for a green field project. Business requirements will change while you're building.

Read this article →

CI/CD Topics

Continuous Delivery
Continuous Integration
Continuous Deployment
Pipelines
Software Testing
Microservices
Tutorials