Monolithic vs. Microservices Architecture

Dr. Vipin Pal

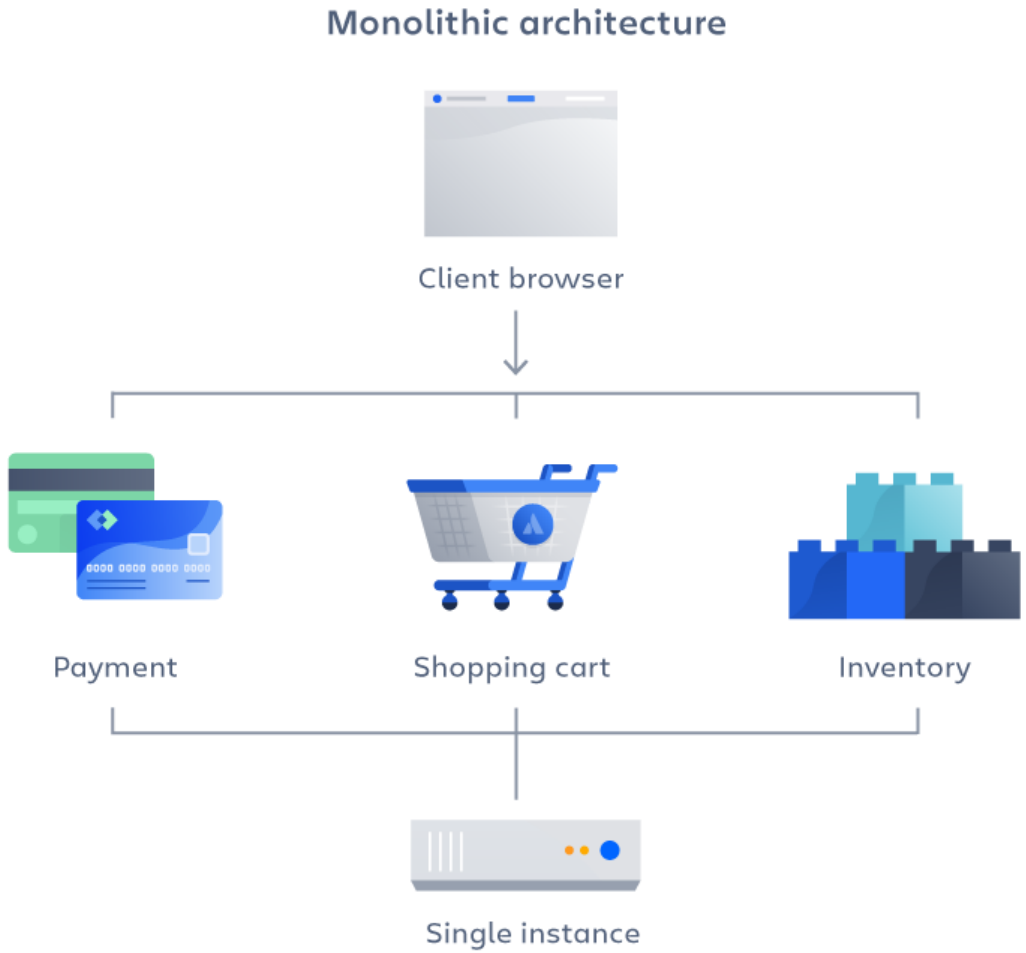
Netflix

- In 2009 Netflix faced growing pains.
- Its infrastructure couldn’t keep up with the demand for its rapidly growing video streaming services.
- The company decided to migrate its IT infrastructure from its private data centers to a public cloud and replace its monolithic architecture with a microservices architecture.
- The only problem was, the term “microservices” didn’t exist and the structure wasn’t well-known.
- Netflix became one of the first high-profile companies to successfully migrate from a monolith to a cloud-based microservices architecture.
- It won the **2015 JAX Special Jury award.**

- A monolithic application is built as a single unified unit while a microservices architecture is a collection of smaller, independently deployable services.
- Which one is right for you?

Monolithic architecture

- A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications.
- The word “monolith” is often attributed to something large and glacial, which isn’t far from the truth of a monolith architecture for software design.
- A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together.
- To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface.
- This makes updates restrictive and time-consuming.


Monolithic architecture
Client browser
Payment
Shopping cart
Inventory
Single instance

- Organizations can benefit from either a monolithic or microservices architecture, depending on a number of different factors.
- When developing using a monolithic architecture, the primary advantage is fast development speed due to the simplicity of having an application based on one code base

Advantages of a monolithic architecture

- **Easy deployment** – One executable file or directory makes deployment easier.
- **Development** – When an application is built with one code base, it is easier to develop.
- **Performance** – In a centralized code base and repository, one API can often perform the same function that numerous APIs perform with microservices.
- **Simplified testing** – Since a monolithic application is a single, centralized unit, end-to-end testing can be performed faster than with a distributed application.
- **Easy debugging** – With all code located in one place, it’s easier to follow a request and find an issue.

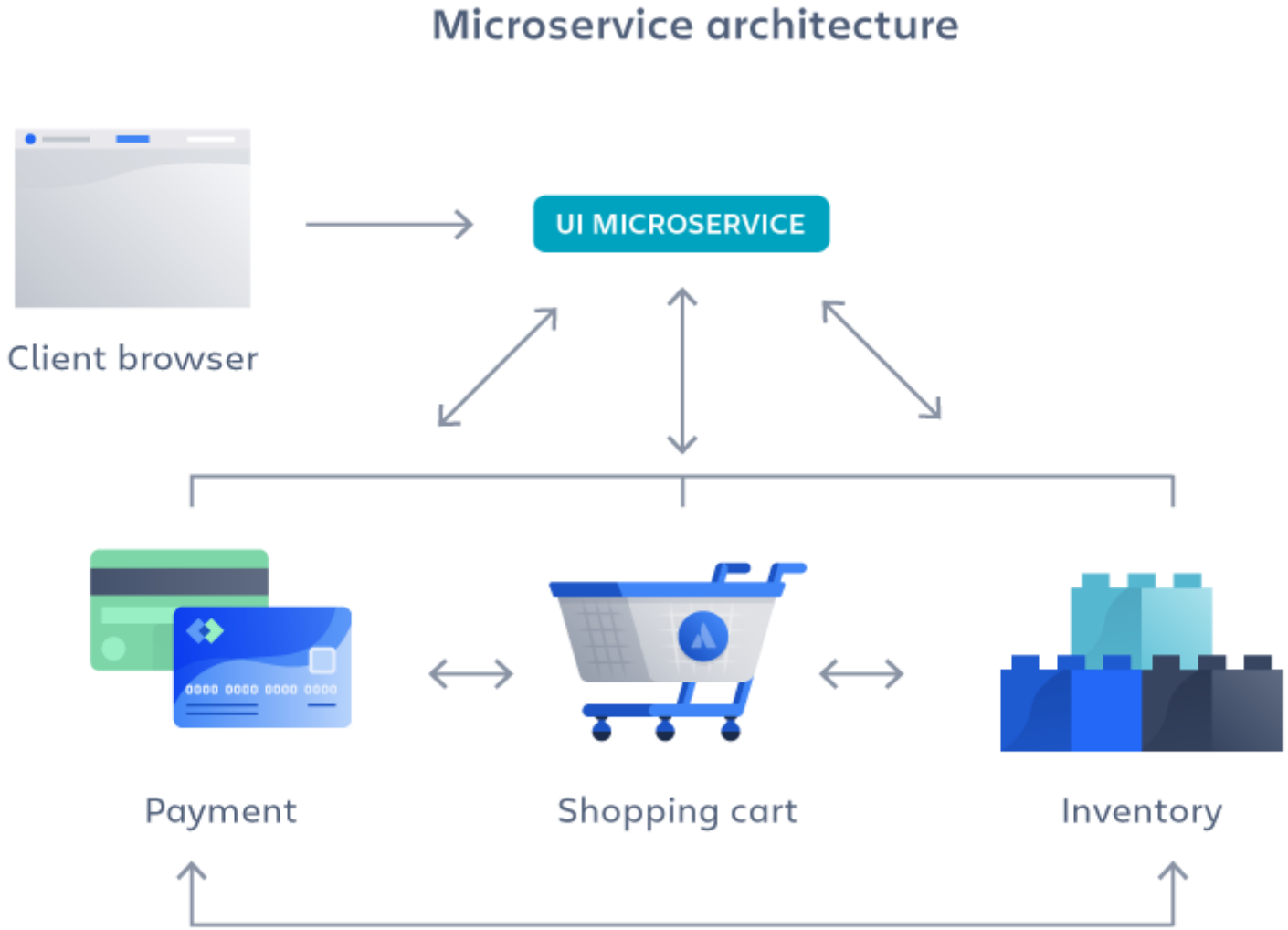
Disadvantages of a monolithic architecture

- **Slower development speed** – A large, monolithic application makes development more complex and slower.
- **Scalability** – You can’t scale individual components.
- **Reliability** – If there’s an error in any module, it could affect the entire application’s availability.
- **Barrier to technology adoption** – Any changes in the framework or language affects the entire application, making changes often expensive and time-consuming.
- **Lack of flexibility** – A monolith is constrained by the technologies already used in the monolith.
- **Deployment** – A small change to a monolithic application requires the redeployment of the entire monolith.

- As with the case of Netflix, monolithic applications can be quite effective until they grow too large and scaling becomes a challenge.
- Making a small change in a single function requires compiling and testing the entire platform, which goes against the agile approach today’s developers favor.

What are microservices?

- A microservices architecture, also simply known as microservices, is an architectural method that relies on a series of independently deployable services.
- These services have their own business logic and database with a specific goal. Updating, testing, deployment, and scaling occur within each service.
- Microservices decouple major business, domain-specific concerns into separate, independent code bases.
- Microservices don’t reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole.


Microservice architecture
UI MICROSERVICE
Client browser
Payment
Shopping cart
Inventory

Advantages of microservices

- **Agility** – Promote agile ways of working with small teams that deploy frequently.
- **Flexible scaling** – If a microservice reaches its load capacity, new instances of that service can rapidly be deployed to the accompanying cluster to help relieve pressure. Now we can support much larger instance sizes.
- **Continuous deployment** – We now have frequent and faster release cycles. Before we would push out updates once a week and now we can do so about two to three times a day.
- **Highly maintainable and testable** – Teams can experiment with new features and roll back if something doesn’t work. This makes it easier to update code and accelerates time-to-market for new features. Plus, it is easy to isolate and fix faults and bugs in individual services.

- **Independently deployable** – Since microservices are individual units they allow for fast and easy independent deployment of individual features.
- **Technology flexibility** – Microservice architectures allow teams the freedom to select the tools they desire.
- **High reliability** – You can deploy changes for a specific service, without the threat of bringing down the entire application.
- **Happier teams** – The Atlassian teams who work with microservices are a lot happier, since they are more autonomous and can build and deploy themselves without waiting weeks for a pull request to be approved.

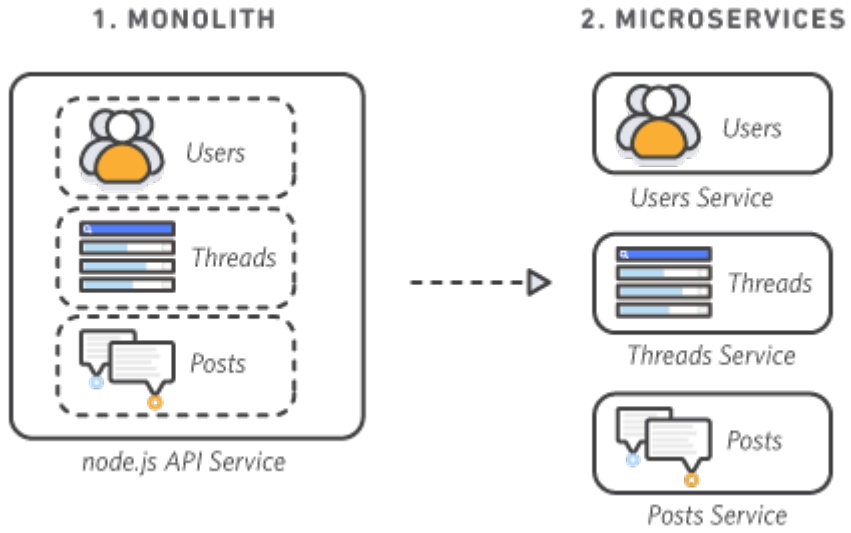
Disadvantages of microservices

- **Development sprawl** – Microservices add more complexity compared to a monolith architecture, since there are more services in more places created by multiple teams. If development sprawl isn't properly managed, it results in slower development speed and poor operational performance.
- **Exponential infrastructure costs** – Each new microservice can have its own cost for test suite, deployment playbooks, hosting infrastructure, monitoring tools, and more.
- **Added organizational overhead** – Teams need to add another level of communication and collaboration to coordinate updates and interfaces.
- **Debugging challenges** – Each microservice has its own set of logs, which makes debugging more complicated. Plus, a single business process can run across multiple machines, further complicating debugging.

- **Lack of standardization** – Without a common platform, there can be a proliferation of languages, logging standards, and monitoring.
- **Lack of clear ownership** – As more services are introduced, so are the number of teams running those services. Over time it becomes difficult to know the available services a team can leverage and who to contact for support.

Monolithic vs. Microservices

- Monolithic applications typically consist of a client-side UI, a database, and a server-side application.
- Developers build all of these modules on a single code base.
- On the other hand, in a distributed architecture, each microservice works to accomplish a single feature or business logic.
- Instead of exchanging data within the same code base, microservices communicate with an API.



Development process

- Monolithic applications are easier to start with, as not much up-front planning is required. You can get started and keep adding code modules as needed. However, the application can become complex and challenging to update or change over time.
- A microservice architecture requires more planning and design before starting. Developers must identify different functions that can work independently and plan consistent APIs. However, the initial coordination makes code maintenance much more efficient. You can make changes and find bugs faster. Code reusability also increases over time.

Deployment

- Deploying monolithic applications is more straightforward than deploying microservices. Developers install the entire application code base and dependencies in a single environment.
- In contrast, deploying microservice-based applications is more complex, as each microservice is an independently deployable software package. Developers usually containerize microservices before deploying them. Containers package the code and related dependencies of the microservice for platform independence.

Debugging

- When debugging monolith architecture, the developer can trace data movement or examine code behavior within the same programming environment. Meanwhile, identifying coding issues in a microservice architecture requires looking at multiple loosely coupled individual services.
- It can be more challenging to debug microservice applications because several developers might be responsible for many microservices.

Modifications

- A small change in one part of a monolithic application affects multiple software functions because of the tightly coupled coding. In addition, when developers introduce new changes to a monolithic application, they must retest and redeploy the entire system on the server.
- In contrast, the microservices approach allows flexibility. It’s easier to make changes to the application. Instead of modifying all the services, developers only change specific functions. They can also deploy particular services independently. Such an approach is helpful in the continuous deployment workflow where developers make frequent small changes without affecting the system's stability.

Scaling

- Monolithic applications face several challenges as they scale. The monolithic architecture contains all functionalities within a single code base, so the entire application must be scaled as requirements change. For example, if the application's performance degrades because the communication function experiences a traffic surge, you must increase the compute resources to accommodate the entire monolithic application. This results in resource wastage because not all parts of the application are at peak capacity.
- Meanwhile, the microservices architecture supports distributed systems. Each software component receives its own computing resources in a distributed system. These resources can be scaled independently based on current capacities and predicted demands. So, for example, you can allocate more resources to a geographic location service instead of the whole system.

When to use monolithic vs. microservices architecture

- Both monolithic and microservices architecture help developers to build applications with different approaches.
- It's important to understand that microservices don't reduce the complexity of an application.
- Instead, the microservices structure reveals underlying complexities and allows developers to build, manage, and scale large applications more efficiently.

Application size

- The monolithic approach is more suitable when designing a simple application or prototype. Because monolithic applications use a single code base and framework, developers can build the software without integrating multiple services. Microservice applications may require substantial time and design effort, which doesn't justify the cost and benefit of very small projects.
- Meanwhile, microservices architecture is better for building a complex system. It provides a robust programming foundation for your team and supports their ability to add more features flexibly. For example, Netflix uses AWS Lambda to scale its streaming infrastructure and save development time.

Team competency

- Despite its flexibility, developing with microservices requires a different knowledge set and design thinking. Unlike monolithic applications, microservices development needs an understanding of cloud architecture, APIs, containerization, and other expertise specific to modern cloud applications. Furthermore, troubleshooting microservices may be challenging for developers new to the distributed architecture.

Infrastructure

- A monolithic application runs on a single server, but microservices applications benefit more from the cloud environment. While it's possible to run microservices from a single server, developers typically host microservices with cloud service providers to help ensure scalability, fault tolerance, and high availability.
- You need the right infrastructure in place before you can start with microservices. You require more effort to set up the tools and workflow for microservices, but they are preferable for building a complex and scalable application.

Monolith Vs. Microservices

Scenario	Monolith	Microservices
Small, simple application	✓	✗
Limited development team resources	✓	✗
Low operational complexity	✓	✗
Frequent changes are not expected	✓	✗
Focus on rapid deployment & iteration	✗	✓
High scalability & fault isolation needed	✗	✓
Leveraging different technology stacks	✗	✓
Dynamically evolving product or market	✗	✓

Category	Monolithic architecture	Microservices architecture
Design	Single code base with multiple interdependent functions.	Independent software components with autonomous functionality that communicate with each other using APIs.
Development	Requires less planning at the start, but gets increasingly complex to understand and maintain.	Requires more planning and infrastructure at the start, but gets easier to manage and maintain over time.
Deployment	Entire application deployed as a single entity.	Every microservice is an independent software entity that requires individual containerized deployment.
Debugging	Trace the code path in the same environment.	Requires advanced debugging tools to trace the data exchange between multiple microservices.
Modification	Small changes introduce greater risks as they impact the entire code base.	You can modify individual microservices without impacting the entire application.
Scale	You have to scale the entire application, even if only certain functional areas experience an increase in demand.	You can scale individual microservices as required, which saves overall scaling costs.
Investment	Low upfront investment at the cost of increased ongoing and maintenance efforts.	Additional time and cost investment to set up the required infrastructure and build team competency. However, long-term cost savings, maintenance, and adaptability.

How can AWS support your microservices architecture requirements?

- You can build modern applications on Amazon Web Services (AWS) with modular architectural patterns, serverless operational models, and agile development processes. We offer a complete platform for building highly available microservices of any scope and scale.

- For example, you can use these AWS services to set up and maintain a microservice architecture:

- Amazon Elastic Container Service (Amazon ECS) to build, isolate, and run secure microservices in managed containers to simplify operations and reduce management overhead
- AWS Lambda to run your microservices without provisioning and managing servers
- AWS App Mesh to monitor and control microservices
- AWS X-Ray to monitor and troubleshoot complex microservice interactions

- In Conclusion, if you’re building a small project, a monolithic architecture is like having everything in one big box, which can be easier to manage at first.
- However, as the project gets bigger, it’s like trying to fit more and more things into that same box, which can become difficult.
- On the other hand, with a microservices architecture, you have different smaller boxes, each handling a specific part of your project.
- This makes it easier to manage and scale as your project grows, but it requires more planning and coordination to make sure all the boxes work together smoothly.

References

- https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith
- https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/
- https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/
- https://www.linkedin.com/pulse/microservices-architecture-case-study-from-various-suryawanshi/