Assignment 2

Monolithic and Microservices

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.

Monoliths can be convenient early on in a project's life for ease of code management, cognitive overhead, and deployment. This allows everything in the monolith to be released at once.



Advantages of a monolithic architecture

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.

The advantages of a monolithic architecture include:

**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

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.

The disadvantages of a monolith include:

**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.

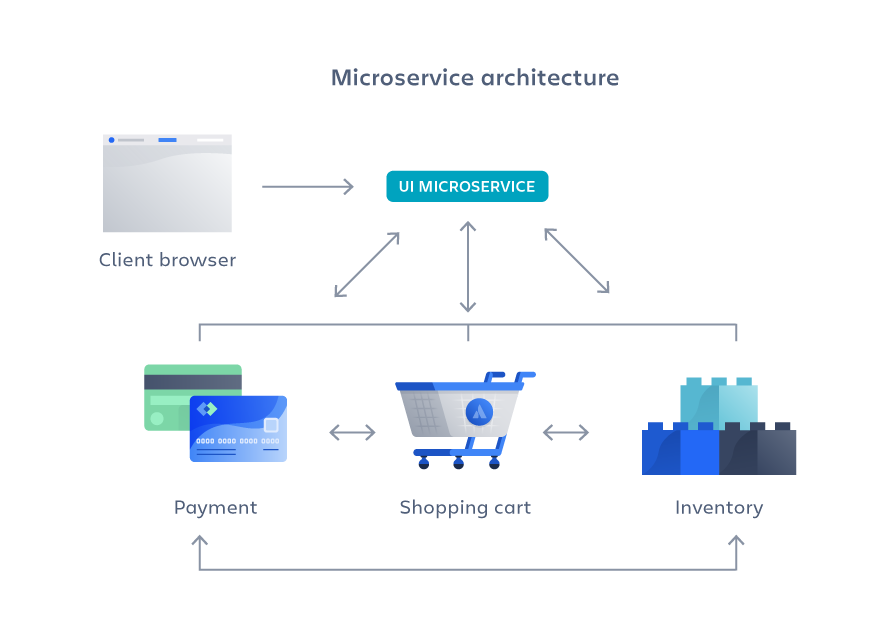
**Barrier to technology adoption** – Any changes in the framework or language affects the entire application, making changes often expensive and time-consuming.

**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.

**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.   
  
Adopting microservices often goes hand in hand with DevOps, since they are the basis for [continuous delivery](https://www.atlassian.com/continuous-delivery) practices that allow teams to adapt quickly to user requirements.



Advantages of microservices

Microservices are by no means a silver bullet, but they solve a number of problems for growing software and companies. Since a microservices architecture consists of units that run independently, each service can be developed, updated, deployed, and scaled without affecting the other services. Software updates can be performed more frequently, with improved reliability, uptime, and performance. We went from pushing updates once a week, to two to three times a day.

As Atlassian grows, microservices enable us to scale teams and geographic locations more reliably by splitting along lines of service ownership. Before we started Vertigo, Atlassian had five different development centers around the world. These distributed teams were constrained by a centralized monolith and we needed to support them in an autonomous fashion. Microservices allow us to do so.

The benefits of Vertigo include increased deployment speed, disaster recovery, reduced cost, and higher performance. This allows us to get to our target faster while delivering more incremental value to customers along the way.

Plus, more generally, microservices make it easier for teams to update code and accelerate release cycles with [continuous integration and continuous delivery (CI/CD)](https://www.atlassian.com/continuous-delivery). Teams can experiment with code and roll back if something goes wrong.

In short, the advantages of microservices are:

**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. We are now multi-tenanant and stateless with customers spread across multiple instances. 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

When we moved from a small number of monolithic codebases to many more distributed systems and services powering our products, unintended complexity arose. We initially struggled to add new capabilities with the same velocity and confidence as we had done in the past. Microservices can add increased complexity that leads to development sprawl, or rapid and unmanaged growth. It can be challenging to determine how different components relate to each other, who owns a particular software component, or how to avoid interfering with dependent components.

With Vertigo, we built a common functionality that would power our existing products and future products we acquire and build. If you are a single product company, microservices may not be necessary.

The disadvantages of microservices can include:

**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.

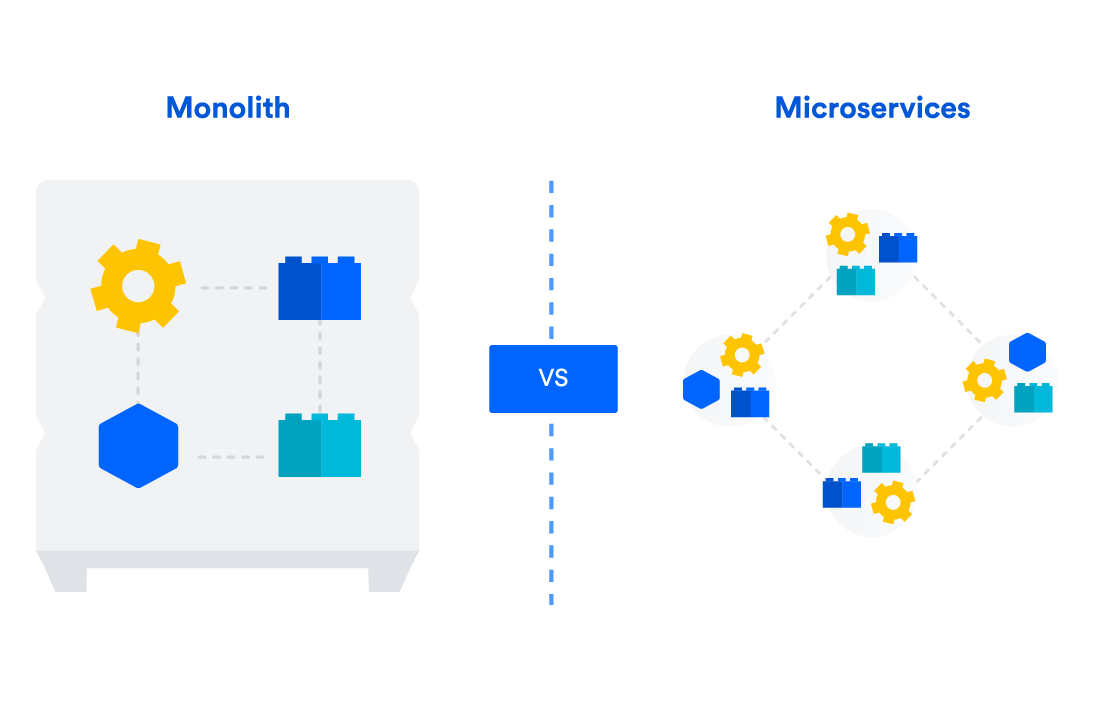
**Exponential infrastructure costs** – Each new microservice can have its own cost for test suite, deployment playbooks, hosting infrastructure, monitoring tools, and more.

**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.



**REST and SOAP API**

* SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol and REST stands for **RE**presentational **S**tate **T**ransfer.
* Since SOAP is a protocol, it follows a strict standard to allow communication between the client and the server whereas REST is an architectural style that doesn’t follow any strict standard but follows six constraints defined by Roy Fielding in 2000. Those constraints are – Uniform Interface, Client-Server, Stateless, Cacheable, Layered System, Code on Demand.
* SOAP uses only XML for exchanging information in its message format whereas REST is not restricted to XML and its the choice of implementer which Media-Type to use like XML, JSON, Plain-text. Moreover, REST can use SOAP protocol but SOAP cannot use REST.
* On behalf of services interfaces to business logic, SOAP uses @WebService whereas REST instead of using interfaces uses URI like @Path.
* SOAP is difficult to implement and it requires more bandwidth whereas REST is easy to implement and requires less bandwidth such as smartphones.
* Benefits of SOAP over REST as SOAP has ACID compliance transaction. Some of the applications require transaction ability which is accepted by SOAP whereas REST lacks in it.
* On the basis of Security, SOAP has SSL( **S**ecure **S**ocket **L**ayer) and WS-security whereas REST has SSL and HTTPS. In the case of Bank Account Password, Card Number, etc. SOAP is preferred over REST. The security issue is all about your application requirement, you have to build security on your own. It’s about what type of protocol you use.
* SOAP cannot make use of REST since SOAP is a protocol without any architectural pattern. REST can make use of SOAP because it is an architectural pattern having protocol.