COMP2511

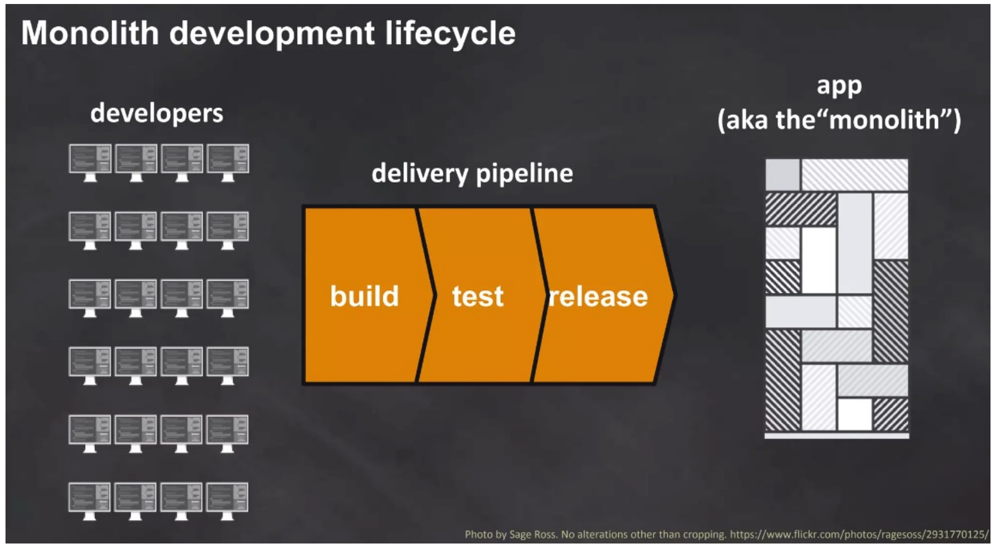
Microservices Software Design

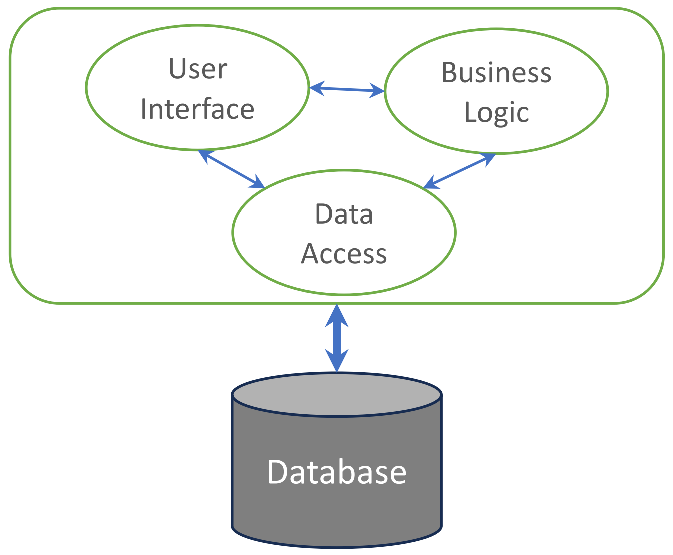
Prepared by

Dr. Ashesh Mahidadia

Monolithic Architecture

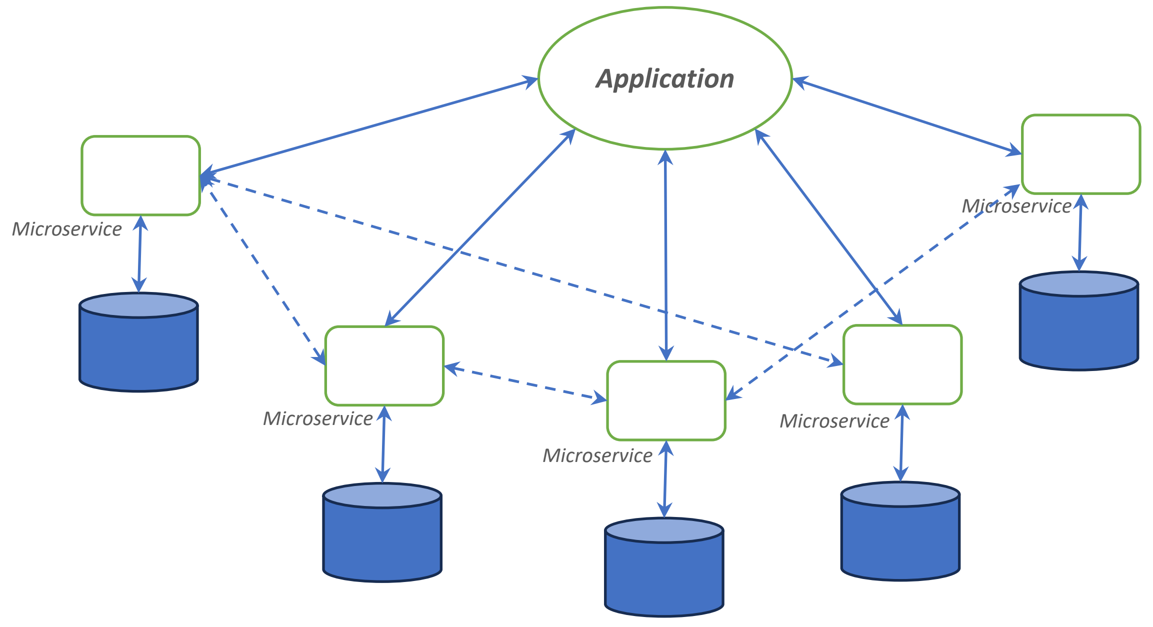
- Long cycle times for building, testing, and releasing.
- Lack of agility.
- The absence of agility hinders the progress of innovations.
- Due to significant coupling, reusability is difficult.
- Often difficult to scale.


Monolith development lifecycle
developers
delivery pipeline
build
test
release
app
(aka the"monolith")
Photo by Sage Ross. No alterations other than cropping. https://www.flickr.com/photos/ragesoss/2931770125/



Monolithic Software Architecture

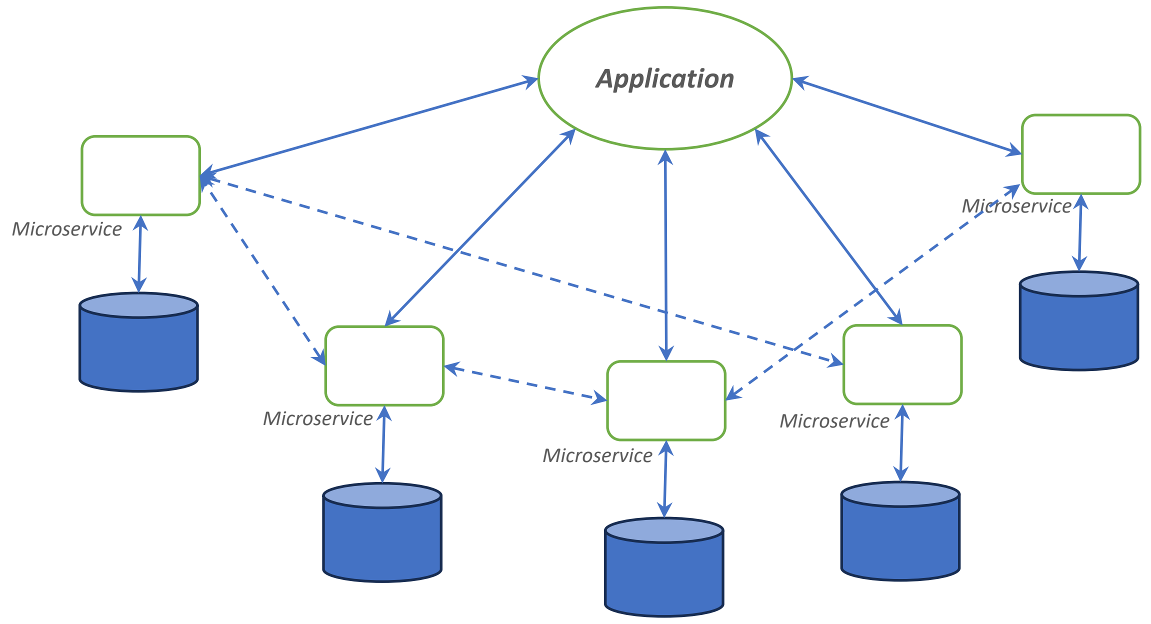
Microservices Architecture



Microservices Software Architecture

- Microservices architecture is an architectural pattern that arranges an application as a collection of loosely coupled services.
- Each service is independently designed, developed, deployed, and maintained.
- Microservices are often developed based on functionality. For example, a service to manage shipping, an order management service, an inventory management service, and so on.

Microservices Architecture



Microservices Software Architecture

- To accomplish loose coupling, services only utilise the appropriate APIs to communicate with other srvices.
- To enable the service to be utilised in a variety of ways, patterns such as adapter and facade are often used to offer multiple interfaces for the same service.
- A service offers encapsulation and abstraction.

Advantages of Microservices

- Individual services can be added, updated or replaced without affecting other services, provided that the service contracts (APIs) are upheld.
- Different software and hardware platforms can be used by different services; for example, Java on Windows 10 on Azure, Python on Linux on AWS, Javascript on Nodejs on local server, etc.
- Only the most in-demand services need to be scaled, there is no need to scale the entire system.
- A service could be reused easily.
- Software complexity could be minimised.

Things to Consider

- Interservice communication latency.
- Idempotency must be considered in design. That is, performing the same action several times leads in the same outcome.
- Avoid using shared data repositories/databases and instead design for data locality.
- The final system should handle individual failures gracefully.
- It is necessary to plan for *eventual consistency*.
- Maintaining a diverse set of services could be a challenge, and we need to orchestrate deployment and maintenance carefully.

End