# Introduction to Microservices

## Monolithic Architecture

* Both the backend and the frontend are tightly coupled
* Very easy to deploy
* Mostly used for small projects and prototypes
* Very hard to scale due to everything being tightly coupled
  + You can’t evolve frontend separately from the backend and vice versa

## Service Oriented Architecture

* The architecture style of breaking your backend into service that can be used for one or more frontend.
* Decoupled the frontend from the backend.
* Makes it easier to scale your project by having the backend and frontend be developed the same time.
* Easier to develop into more complex projects.

## Microservices Architecture

* Implementation of SOA that breaks down services even more to have their own responsibility and give a team of developers to just develop that one responsibility even further (Think of the S in SOLID).
* Starting this architecture is extremely complex and expensive.
* Very popular architectural style used by big IT companies such as Google, Amazon, Facebook, etc.

# MSA Characteristics

## Single Responsibility Principle

* One service must be responsible for one thing.

## Encapsulated

* Each service must encapsulate the data and behavior as a single unit.
* Each data for each service must be private and can only be seen by its own service.
* Therefore, each service must have each own private database it has accesses to.

## Independent

* Each service must be independent from each other.
* You can develop a service without worrying about breaking the other service that is being developed right now.
* Furthermore, each service can be developed in a completely different language if needed based on the responsibility of that service.

# Benefits

* Scalability
  + You can scale your application depending on how many people you have working at it.
  + It isn’t limited by how you constructed your entire architecture.
* Simplicity of developing new features
  + Once the infrastructure is setup.
  + Adding new features is a lot easier and you don’t have to worry about breaking other features from different services.
* Deployment of individual service
  + Easy to deploy the induvial services when scaling out.
  + Trying to redeploy multiple services at once will take forever to deploy
* Fault Tolerant
  + One service going down due to some external force or bad code won’t affect your other services.
  + A nice safety net that won’t crash or halt your entire operations for your whole company
* Language Agnostic
  + You can work with different languages if they all send the data in the same protocol (HTTP, HTTPS)
  + Enables you to work with the right language for different responsibilities
* Testable
  + Easy to test when each service serves only one purpose

# Drawbacks

* Deployment of a whole entire environment
  + Deploying everything at the same time (usually the initial setup) will take a while and might even cause problems due to deploying a whole ecosystem of services.
* Complexity when it comes to communication between services
  + So, with joins and aggregation of different data can become very convoluted
  + Very simple to develop each service separately but very difficult on making them communicate with each other to do some operation
* Monitoring
  + Checking on multiple services to see if there is an issue might take time (especially when your microservice ecosystem grows)
  + Thankfully there is an automated process that do it for us
* Consistency
  + Data consistency is a bit of a challenge to accomplish
* Communication is key
  + Since there are multiple services and they must have a stable communication or else things can fail and data inconsistency might happen
* Overall, it is a steep starting point for using MSA

# Key things to help make MSA work

## Service Discoverability

* You have a service registry that contains information about the services in your MSA ecosystem.
  + Think of a phone book or how DNS (Domain Name System) works
* They are responsible for automating the process of checking/monitory on your services
* They let you know which service is fine and which are down
* (comes built in with K8s)

## Gateway

* They are responsible for helping the services communicate with each other.
* It is essentially the hub of communication between services and user.
* So instead of services talking to each other directly, a service can just talk to a gateway and the gateway will take care of whoever it needs to talk to and accomplish what that service wanted.
* It essentially abstracts the entire MSA ecosystem for end users and make it appear like we are just talking to one service

## Load Balancing

* With multiple services running on different containers/servers/nodes/some sort of deployment artifact, you have a load balancer that handles that artifact
* They will balance the outgoing/internal traffic to the instances of the service

## Circuit Breaker

* Almost the same as what your circuit breaker does in your home.
* If something fails, it’ll make sure that it won’t create a cascading failure of your other services.
  + Like your electric circuit breaker that protects your other circuits from damage if another circuit causes a short circuit.

## Message queues

* Another mechanism to help with communications between services.
* It is a form of asynchronous service-to-service communication.
* It stores messages in a queue until they are process individually especially with spike workloads.