INTRODUCTION TO THE MICROSERVICES

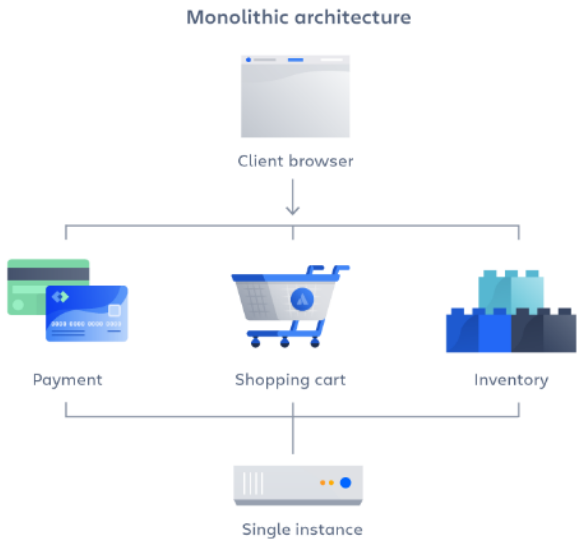
A monolithic application is built as a single unified unit while a microservices architecture is a collection of smaller, independently deployable services.

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 in part due to this new infrastructure that internalized DevOps.

What is a 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.

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.



Advantages of a monolithic architecture

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

- The disadvantages of a monolith include:

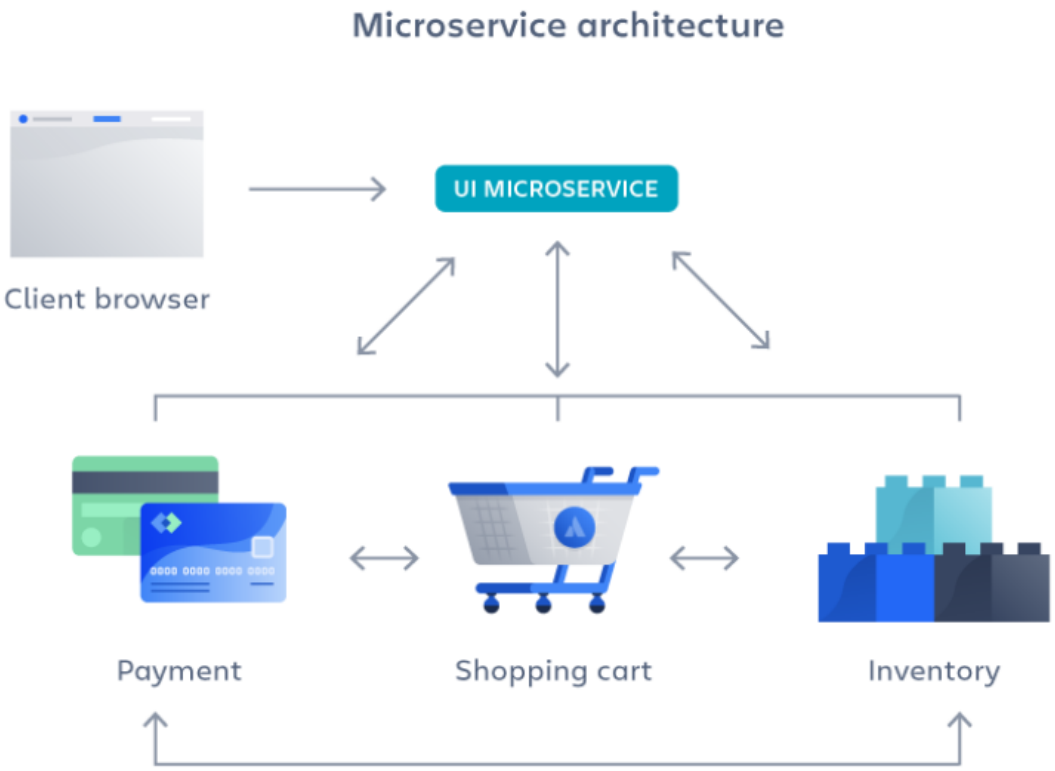
- **Slower development speed** – A large, monolithic application makes development more complex and slower.
- **Scalability** – You can't scale individual components.
- **Reliability** – If there's an error in any module, it could affect the entire application's availability.
- **Barrier to technology 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.

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.



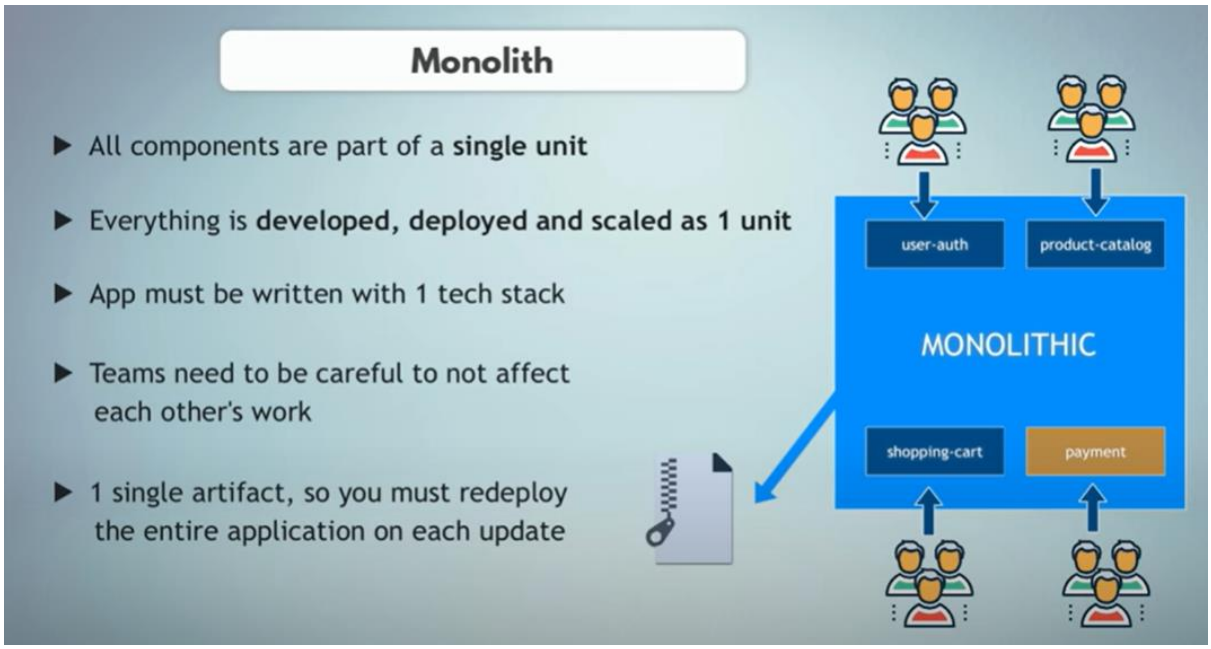
Advantages of microservices

- **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. 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

- **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.



Challenges of monolithic architecture

- Application is too large and complex
- Parts are more tangled into each other
- You can only scale the entire app, instead of a specific service
- Difficulty if services need different dependency versions

Challenges of monolithic architecture

- Release process takes longer
- On every change, the entire application needs to be tested
- Entire application needs to be built and deployed
- Bug in any module can potentially bring down the entire application

What are Microservices?

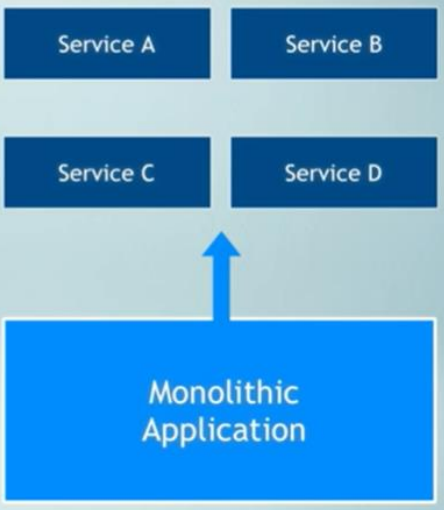
Microservices Architecture

- Split application into **smaller, independent services**

How to break down the application?

What code goes where?

How many services do we create?



Microservices Architecture

- Split application into **smaller, independent services**

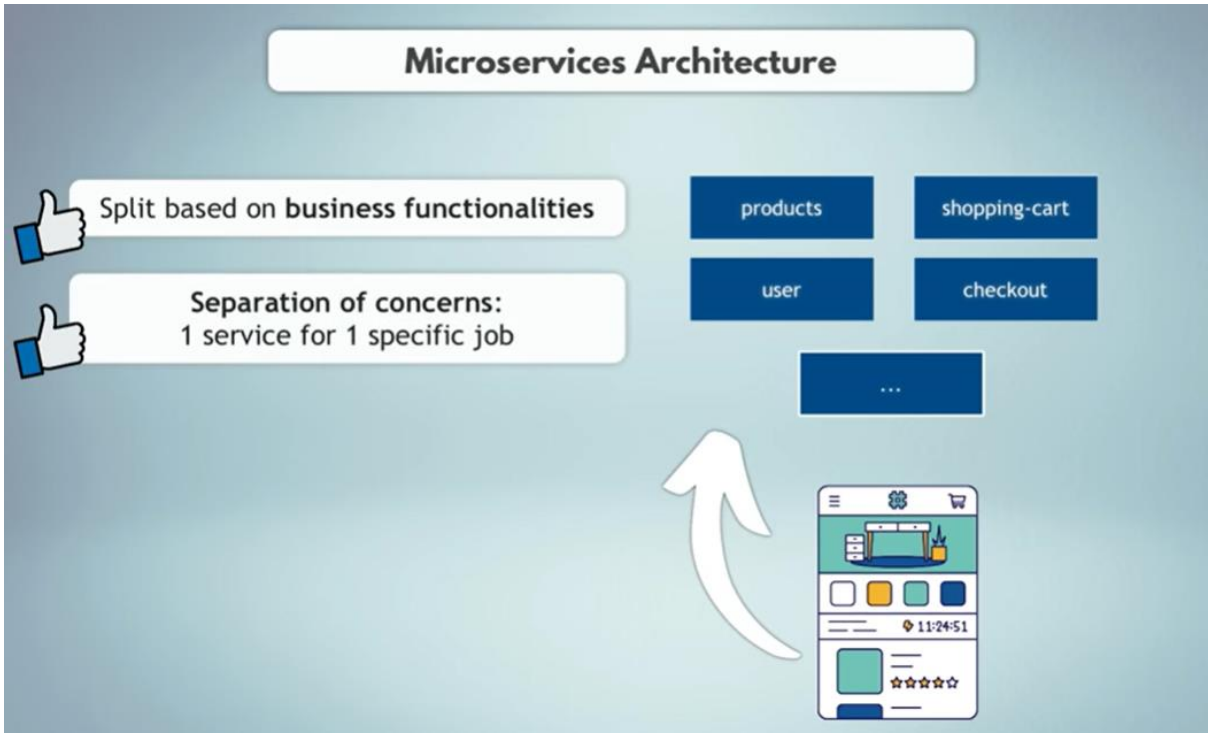
How to break down the application?

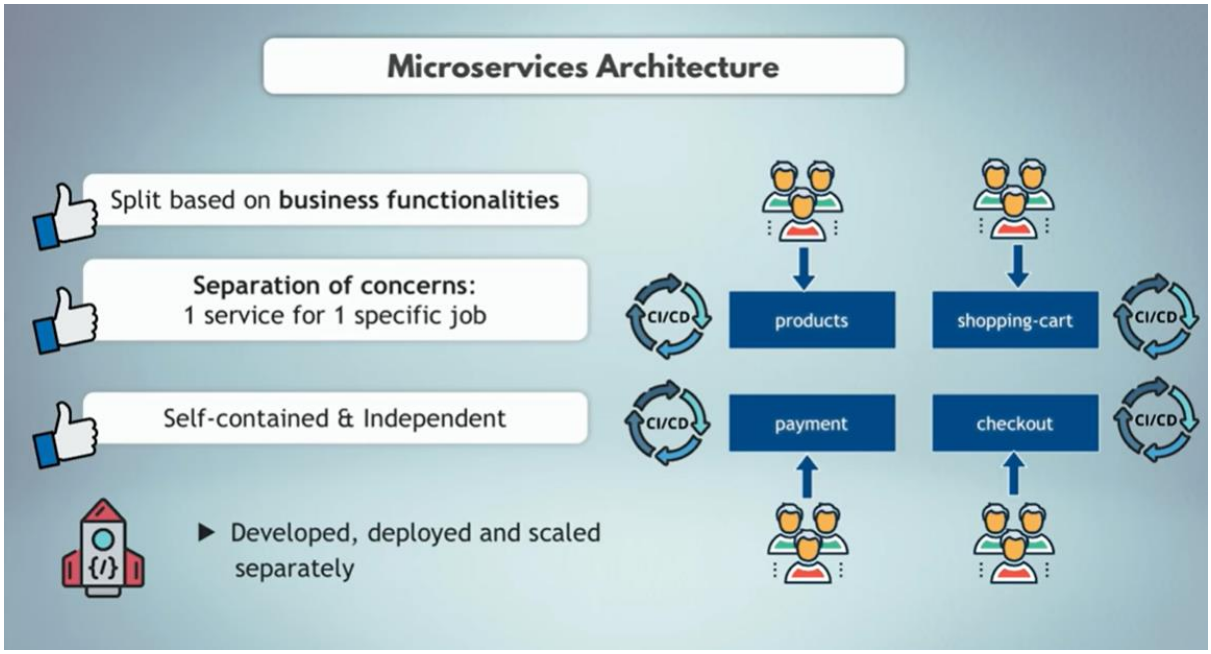
What code goes where?

How many services do we create?

How big/small should they be?

How do they communicate?


Microservices Architecture
Split based on business functionalities
Separation of concerns:
1 service for 1 specific job
products
shopping-cart
user
checkout
...
11:24:51


Microservices Architecture
Split based on business functionalities
Separation of concerns:
1 service for 1 specific job
Self-contained & Independent
Developed, deployed and scaled separately
CI/CD
products
shopping-cart
payment
checkout

Microservices Architecture

- Split based on **business functionalities**
- **Separation of concerns:** 1 service for 1 specific job
- Self-contained & Independent

▶ Apps should be **loosely coupled!**

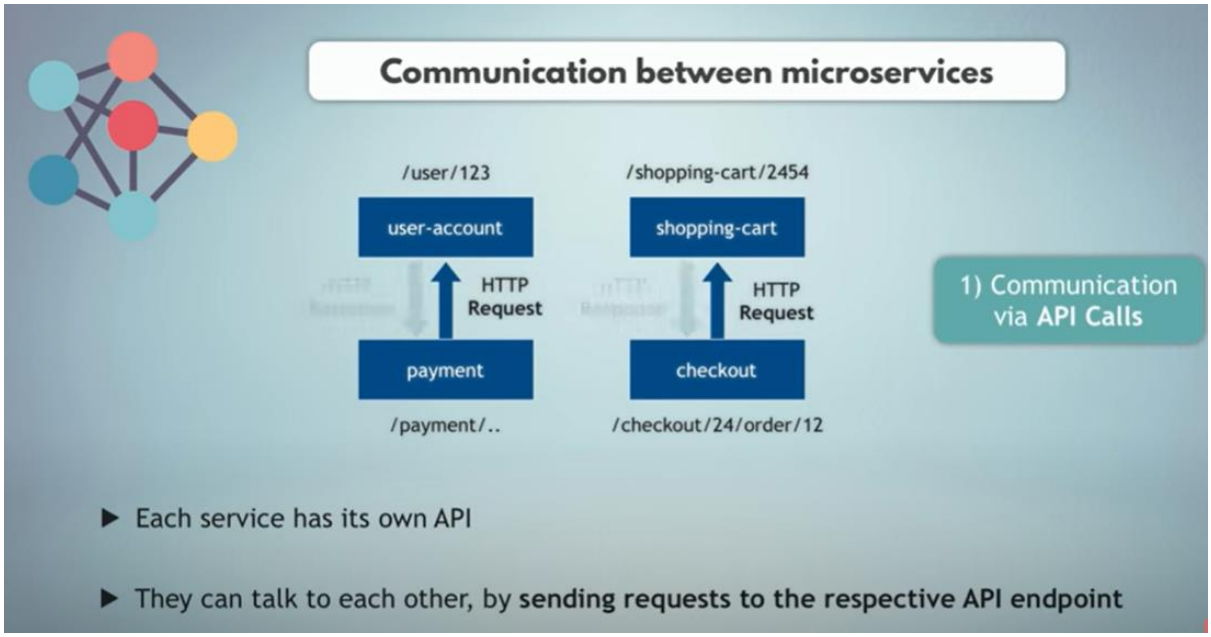
Microservices Architecture

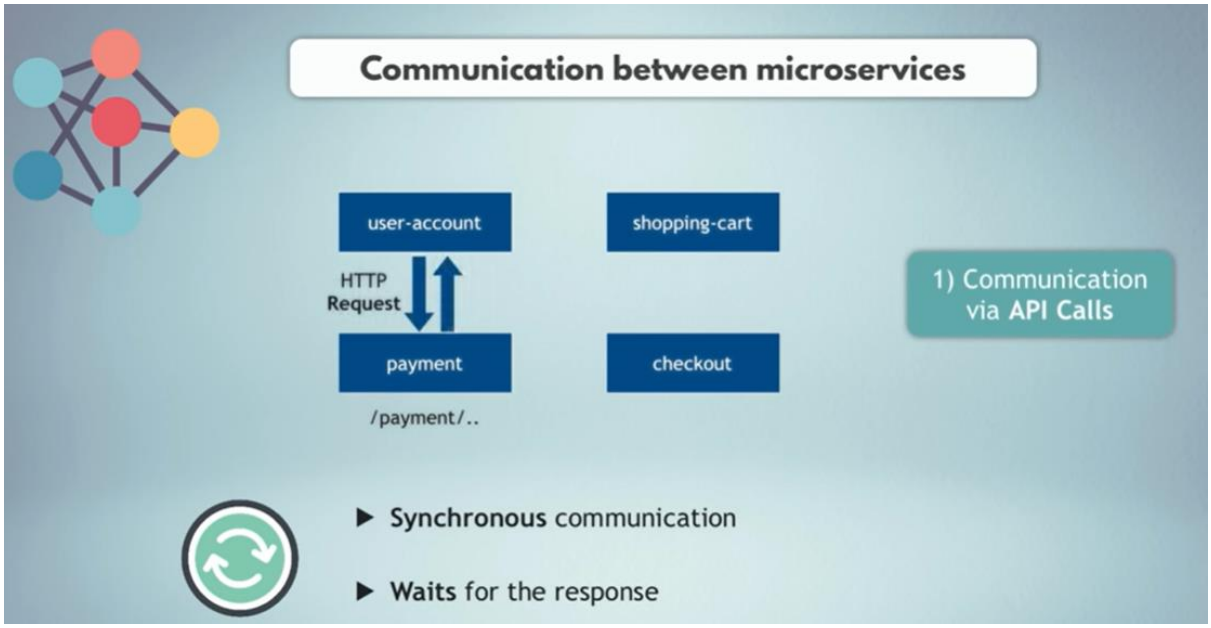
▶ Services can be built and deployed separately

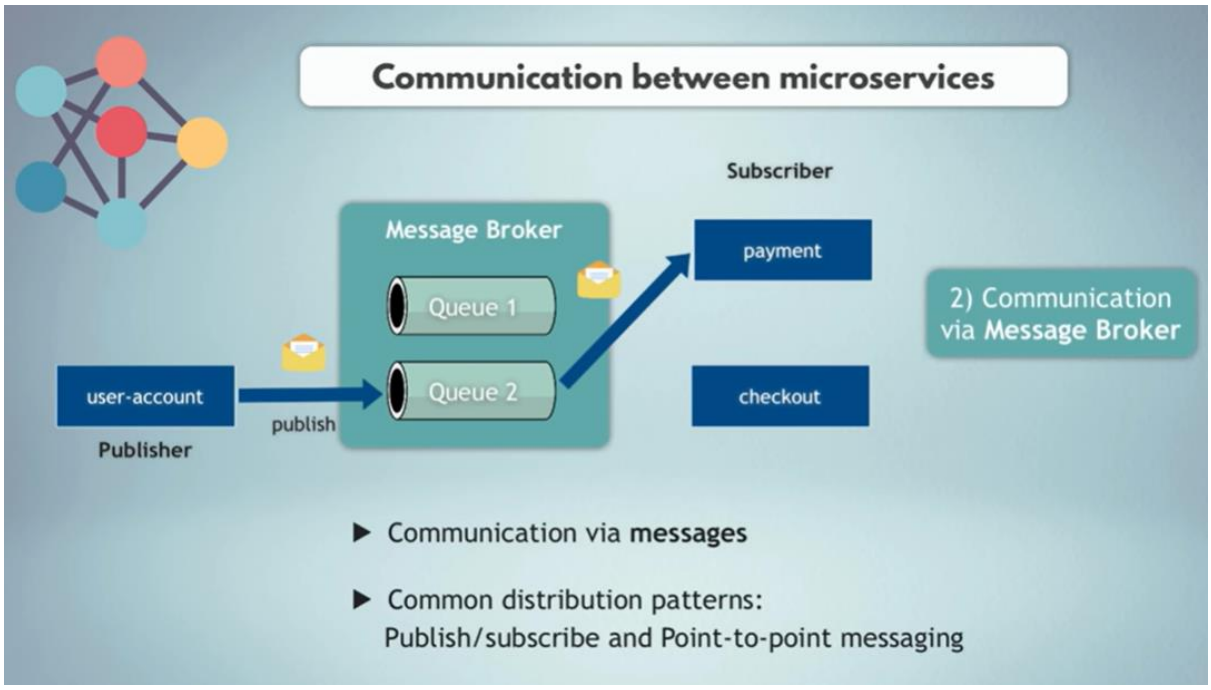
Microservices Architecture

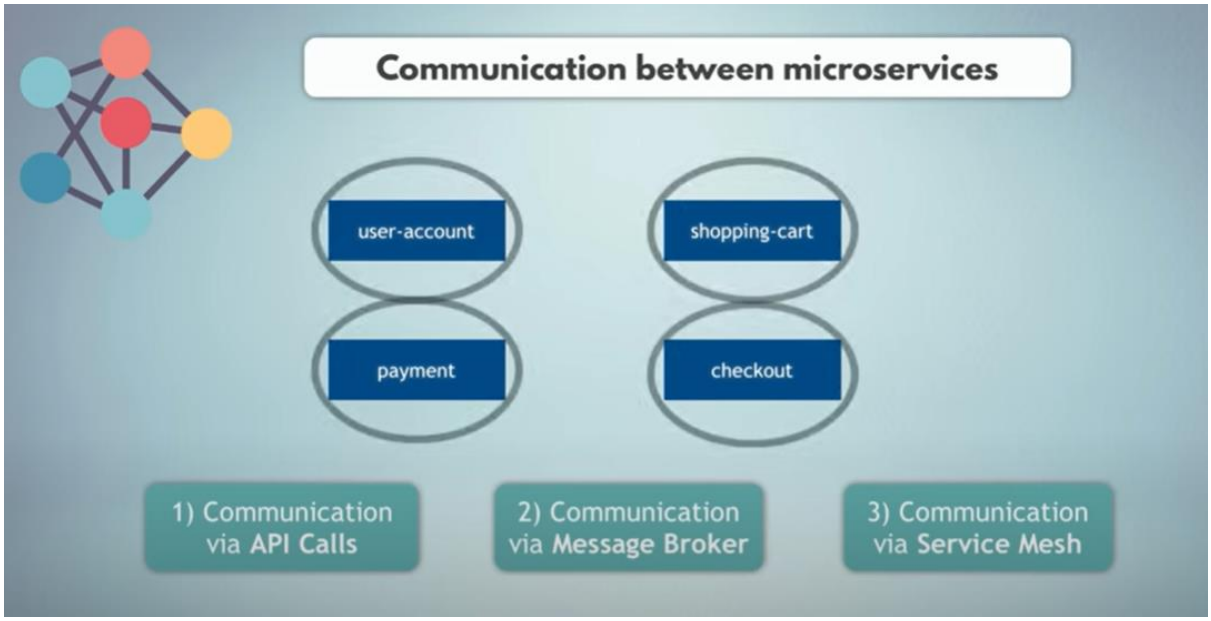
▶ Each microservice has its own version

Communication between Services



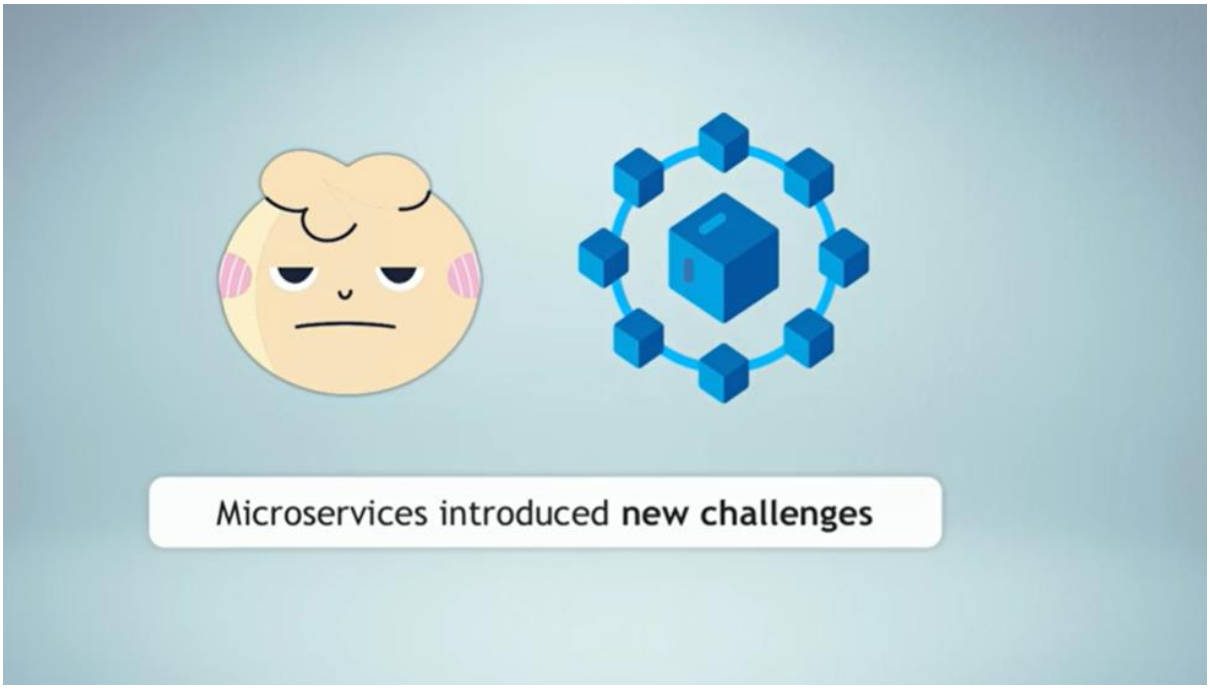

Communication between microservices
user-account
shopping-cart
HTTP Request
payment
checkout
/payment/..
1) Communication via API Calls
Synchronous communication
Waits for the response

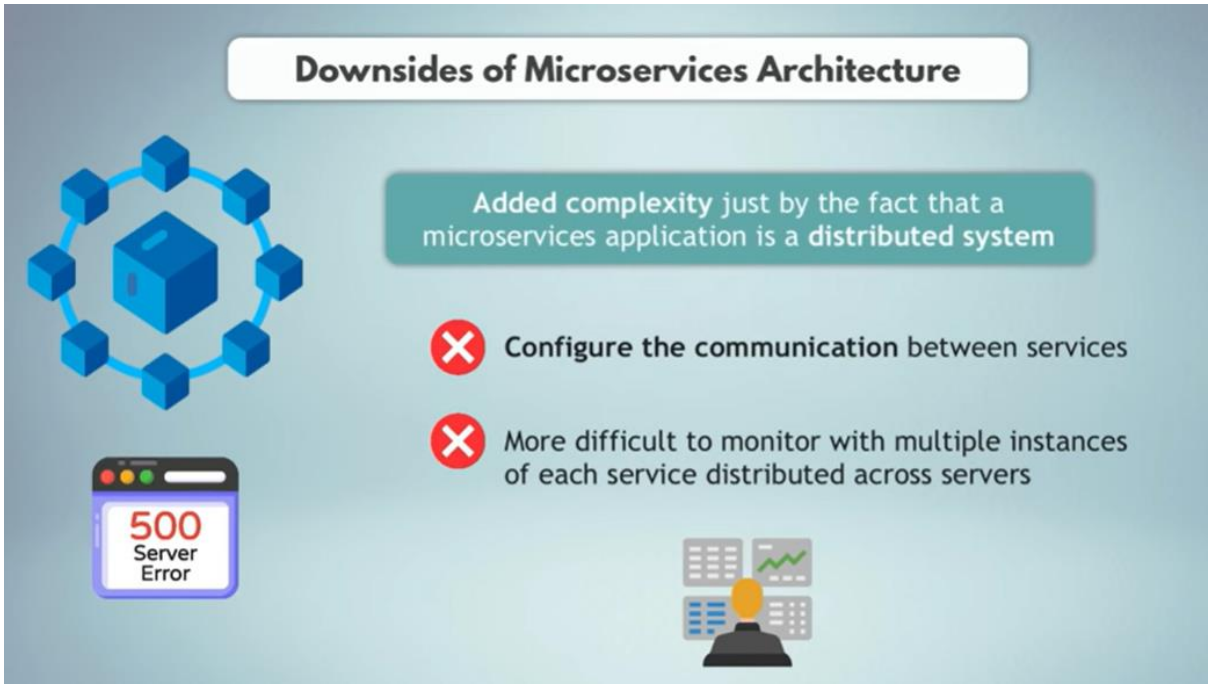

Communication between microservices
Subscriber
Message Broker
payment
Queue 1
Queue 2
2) Communication via Message Broker
user-account
publish
checkout
Publisher
Communication via messages
Common distribution patterns:
Publish/subscribe and Point-to-point messaging


Communication between microservices
user-account
shopping-cart
payment
checkout
1) Communication via API Calls
2) Communication via Message Broker
3) Communication via Service Mesh


user-account
shopping-cart
payment
checkout
Java
GO
Each microservice team can choose its own tech stack
Each team can develop the service independently, without affecting others

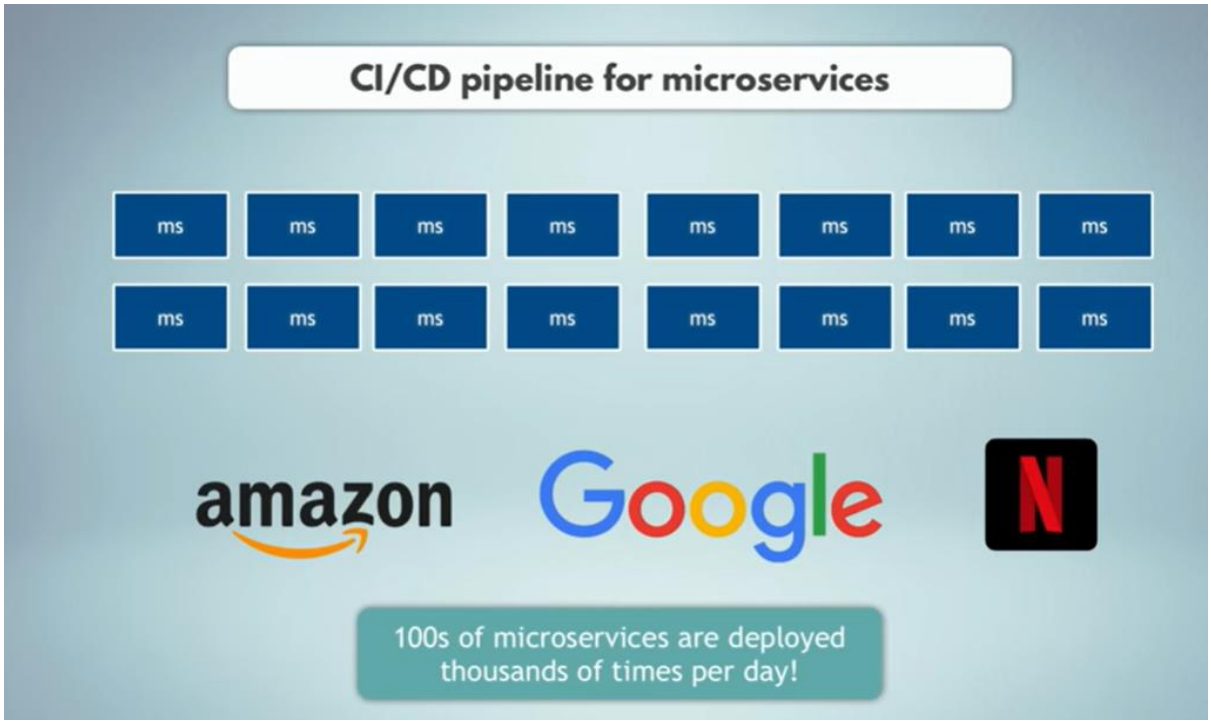
Downsides of Microservices




Downsides of Microservices Architecture
Added complexity just by the fact that a microservices application is a distributed system
Configure the communication between services
More difficult to monitor with multiple instances of each service distributed across servers
500
Server
Error


Tools are being developed to tackle these challenges
Security
Messaging
Orchestration
Service Mesh
Containers
Monitoring

CI/CD Pipeline for Microservices

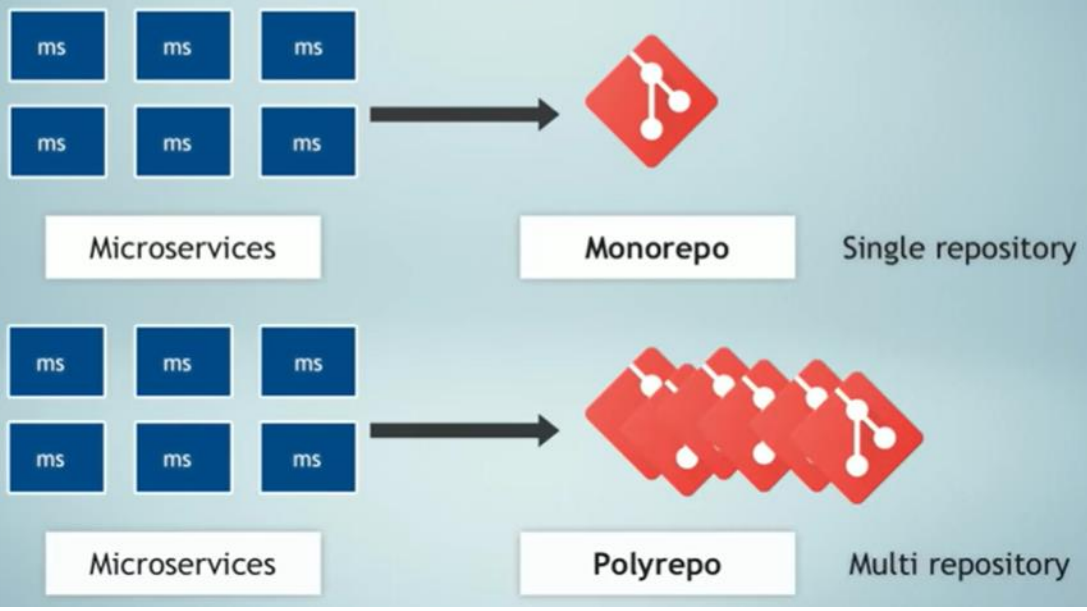

CI/CD pipeline for microservices
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
ms
amazon
Google
N
100s of microservices are deployed
thousands of times per day!

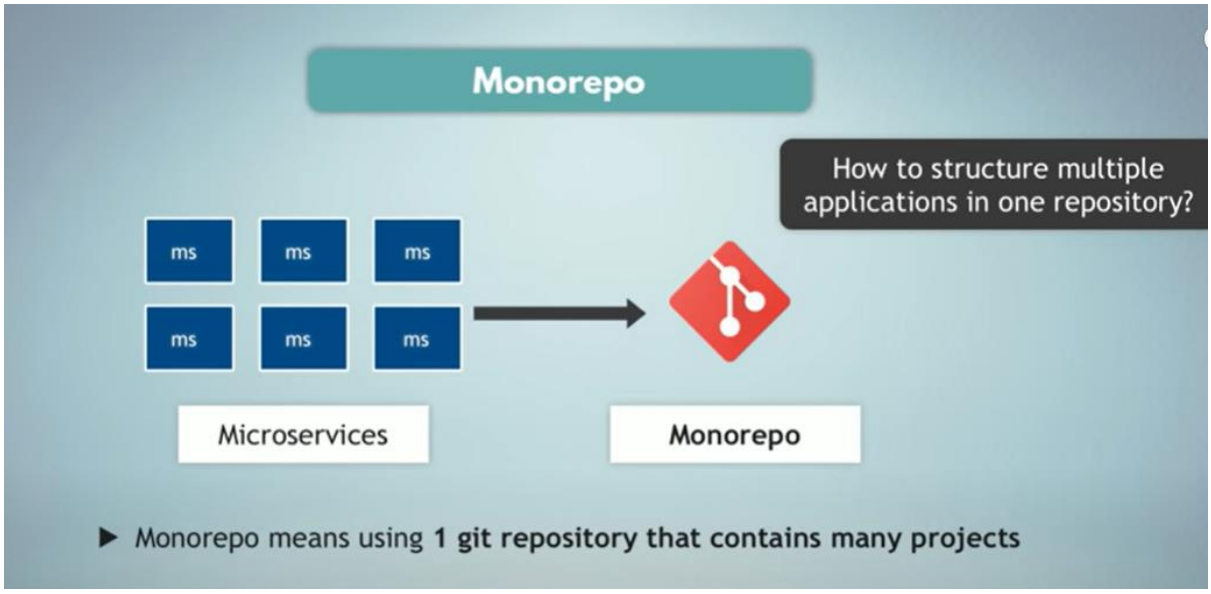

Monorepo vs
Polyrepo

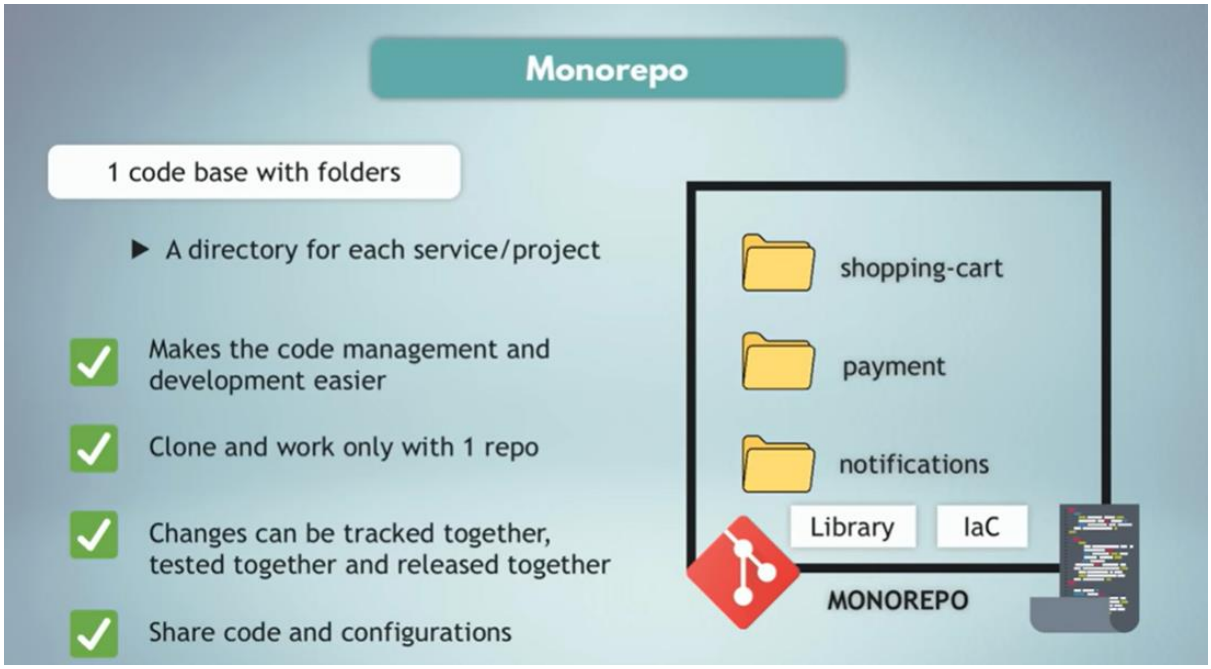
How do we manage the code?

- Application components are developed and deployed **separately**

Monorepo and Polyrepo as 2 options




Monorepo
How to structure multiple applications in one repository?
ms
ms
ms
ms
ms
ms
Microservices
Monorepo
▶ Monorepo means using 1 git repository that contains many projects


Monorepo
1 code base with folders
▶ A directory for each service/project
Makes the code management and development easier
Clone and work only with 1 repo
Changes can be tracked together, tested together and released together
Share code and configurations
shopping-cart
payment
notifications
Library
IaC
MONOREPO

Monorepo

Challenges

- Tight coupling of projects
- Easier to break this criterion and develop more tightly coupled code
- Big source code, means git interactions becomes slow
- Additional logic necessary to make sure only service is built and deployed which had code changes

Monorepo

Challenges

- All projects and teams are affected, if there is some issue

Polyrepo

1 repository for each service

- Code is completely isolated
- Clone and work on them separately

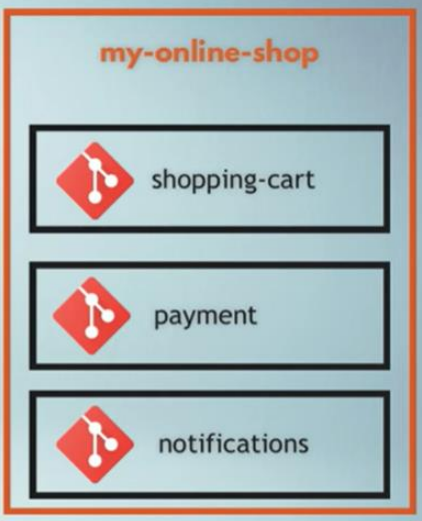
POLYREPO

Polyrepo

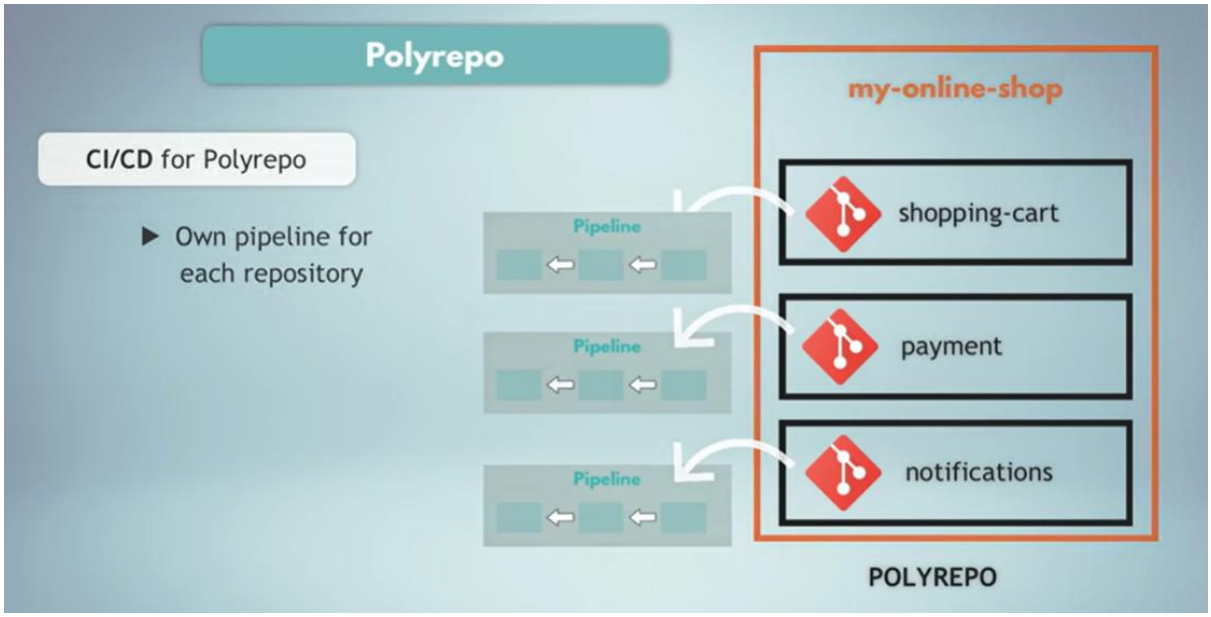
GitLab Groups

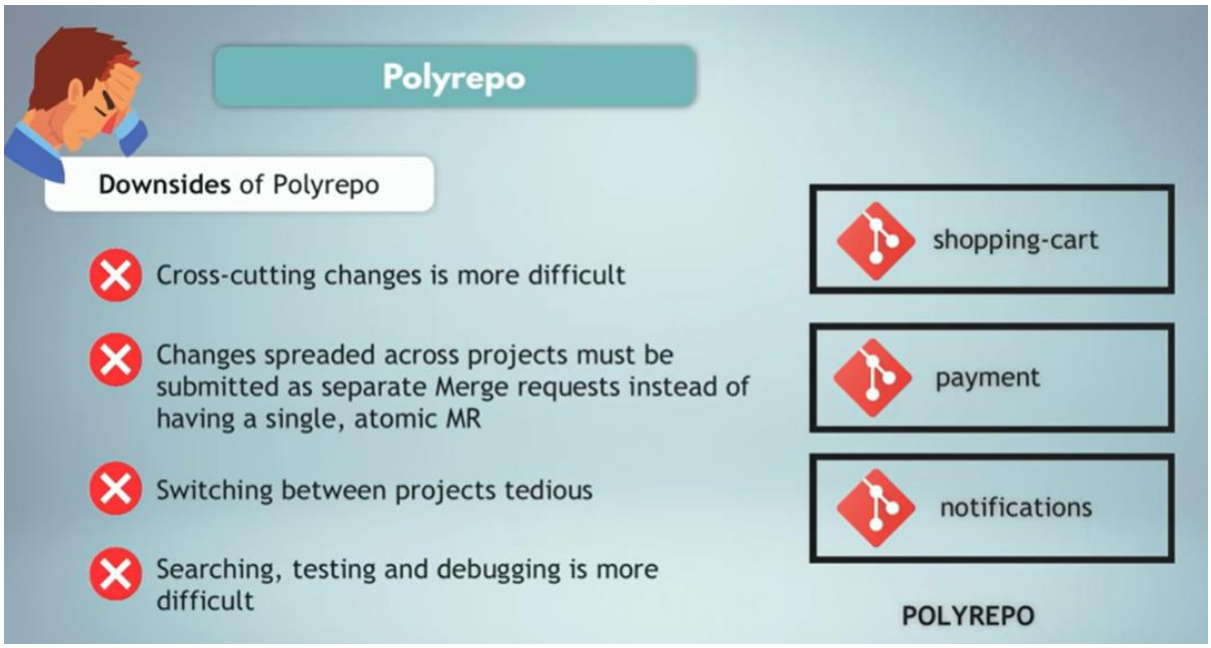
- In GitLab you have projects to configure connected projects together (**Groups**)

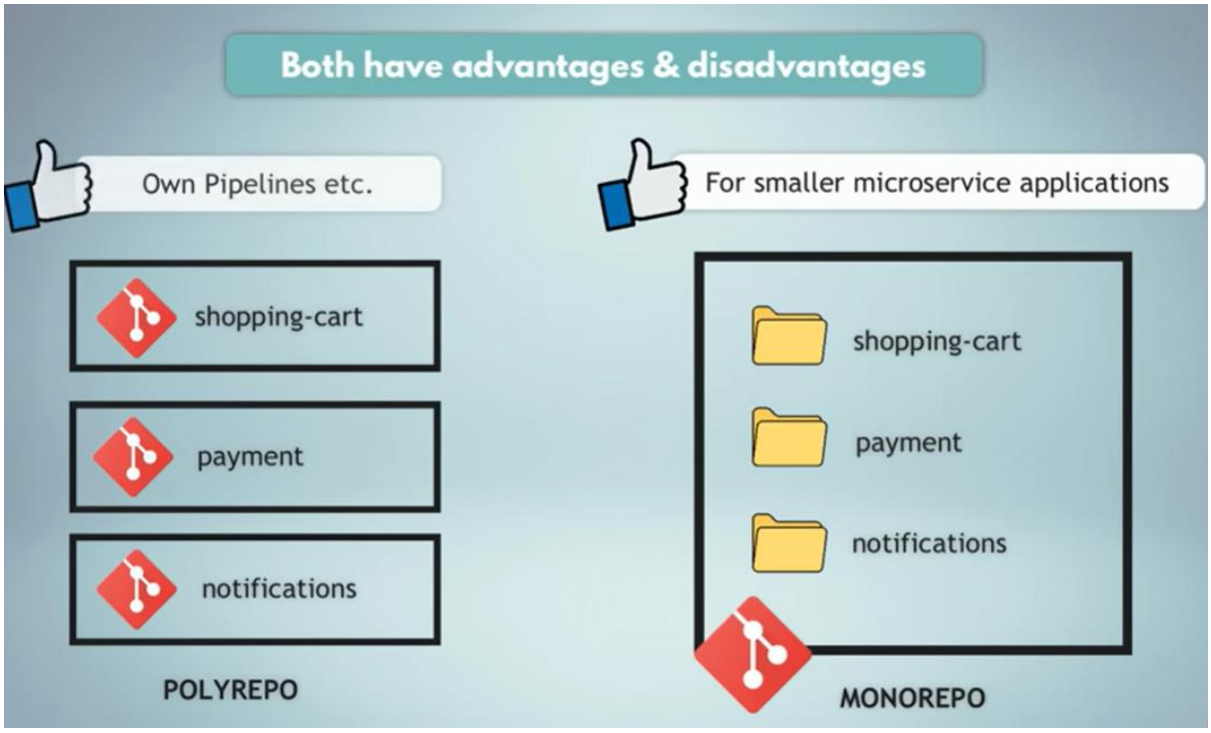
- Helps to keep an overview
- Create shared secrets, CI/CD variables, Runners



POLYREPO


Polyrepo
CI/CD for Polyrepo
Own pipeline for each repository
Pipeline
Pipeline
Pipeline
my-online-shop
shopping-cart
payment
notifications
POLYREPO


Polyrepo
Downsides of Polyrepo
Cross-cutting changes is more difficult
Changes spreaded across projects must be submitted as separate Merge requests instead of having a single, atomic MR
Switching between projects tedious
Searching, testing and debugging is more difficult
shopping-cart
payment
notifications
POLYREPO


Both have advantages & disadvantages
Own Pipelines etc.
shopping-cart
payment
notifications
POLYREPO
For smaller microservice applications
shopping-cart
payment
notifications
MONOREPO