YOOX NET-A-PORTER GROUP

YNAP @ UNIMORE

17 Maggio 2024

YNAP intro

Open questions

Our experience as developers

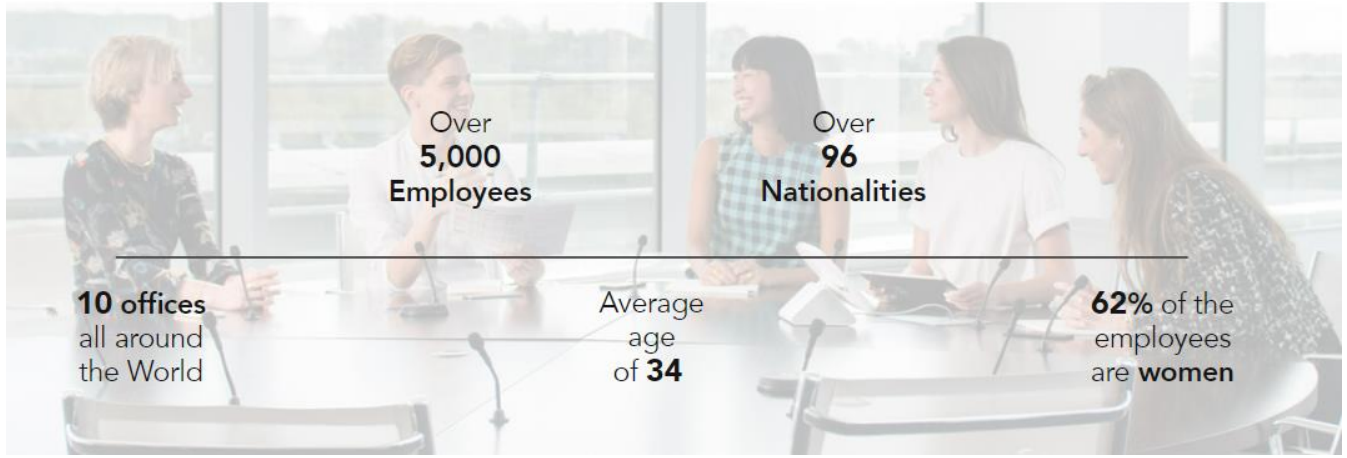
How technology and the world of work in IT are evolving

Microservices VS Monolithic Application

Q&A

YNAP INTRO

YOOX NET-A-PORTER GROUP is the world's leading online luxury and fashion retailer



WHAT WE DO

YOOX NET-A-PORTER GROUP is a unique ecosystem of 4 multi-brand online stores (NET-A-PORTER, MR PORTER, YOOX and THE OUTNET), and numerous ONLINE FLAGSHIP STORES, powered by one technology and operations platform serving more than 4 million of active customers in 170 countries around the world.

WHERE WE OPERATE

YOOX NET-A-PORTER GROUP has offices and operations in the United States, Europe, Middle East, Japan, China and Hong Kong and delivers to more than 180 countries around the world.

WHO WE ARE

Marco Petrucci
CTPO
+15y in YNAP

Francesco Pichierri
Head of Enterprise Architecture
+12y in YNAP

YNAP intro

Open questions

Our experience as developers

How technology and the world of work in IT are evolving

Microservices VS Monolithic Application

Q&A

OPEN QUESTIONS

- **INFO / KNOWLEDGE**
- **PASSION / CURIOSITY**
- **EXPECTATIONS**

YNAP intro

Open questions

Our experience as developers

How technology and the world of work in IT are evolving

Microservices VS Monolithic Application

Q&A

MARCO – MY MAIN EXPERIENCES IN IT

Small Web Agency

- Developed multiple skills
- End-to-end solution lifecycle
- Focus on productivity

Enterprise Company

- High specialization
- Investments in innovation
- Culture of quality

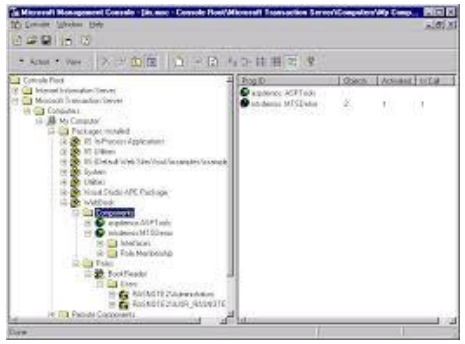
Managerial path

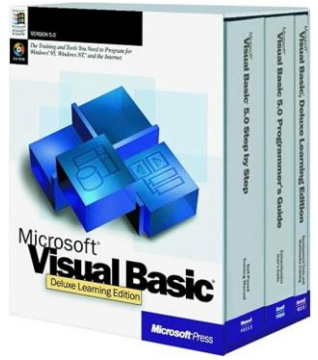
- Personal attitude
- Balance between passion and delegation
- Satisfaction from others successes

FRANCESCO – MY MAIN EXPERIENCES IN IT

Several Software House

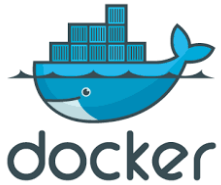
- Full-stack developer
  - Desktop application development
  - Client/Server application development
  - Web application development





Enterprise Company

- Distributed system
- Application containers
- Cloud
- Enterprise Architecture
- Software Development Methodologies



YNAP intro

Open questions

Our experience as developers

How technology and the world of work in IT are evolving

Microservices VS Monolithic Application

Q&A

MAIN TRENDS

- From prem to Cloud
- Data close to applications
- AI everywhere!

YNAP intro

Open questions

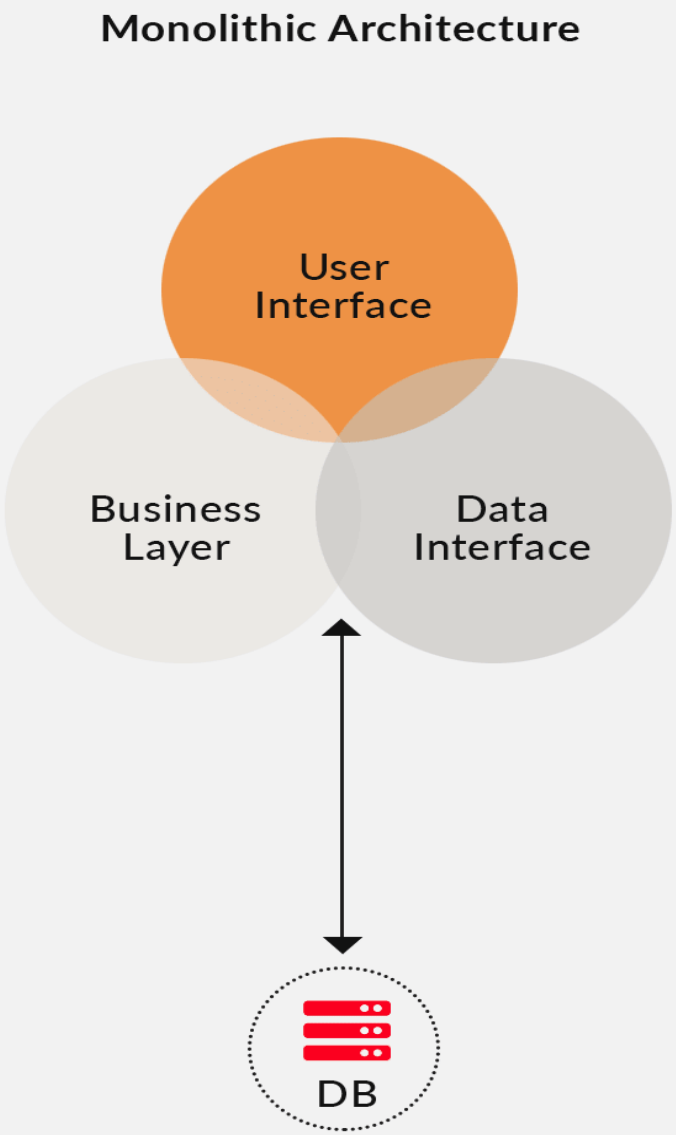
Our experience as developers

How technology and the world of work in IT are evolving

Microservices VS Monolithic Application

Q&A

WHAT EXACTLY IS A MONOLITHIC ARCHITECTURE?

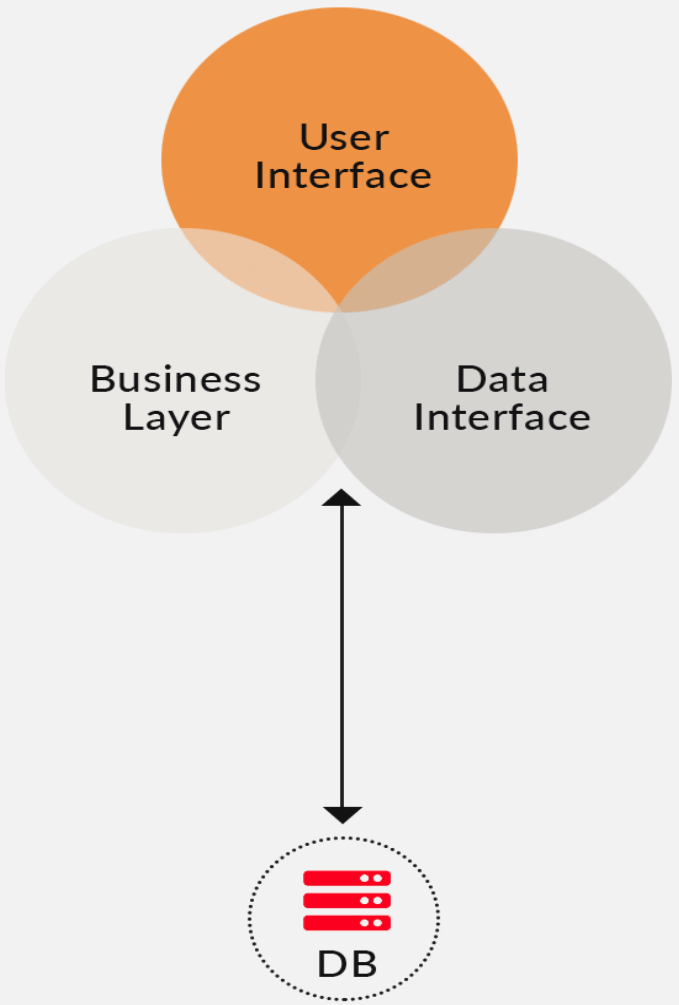


A monolithic architecture is a traditional software development model that uses one code base to perform multiple business functions. All the software components in a monolithic system are interdependent due to the data exchange mechanisms within the system.

This means that multiple services, such as APIs, databases, load balancers, and more, will function as one large application organized into different layers.

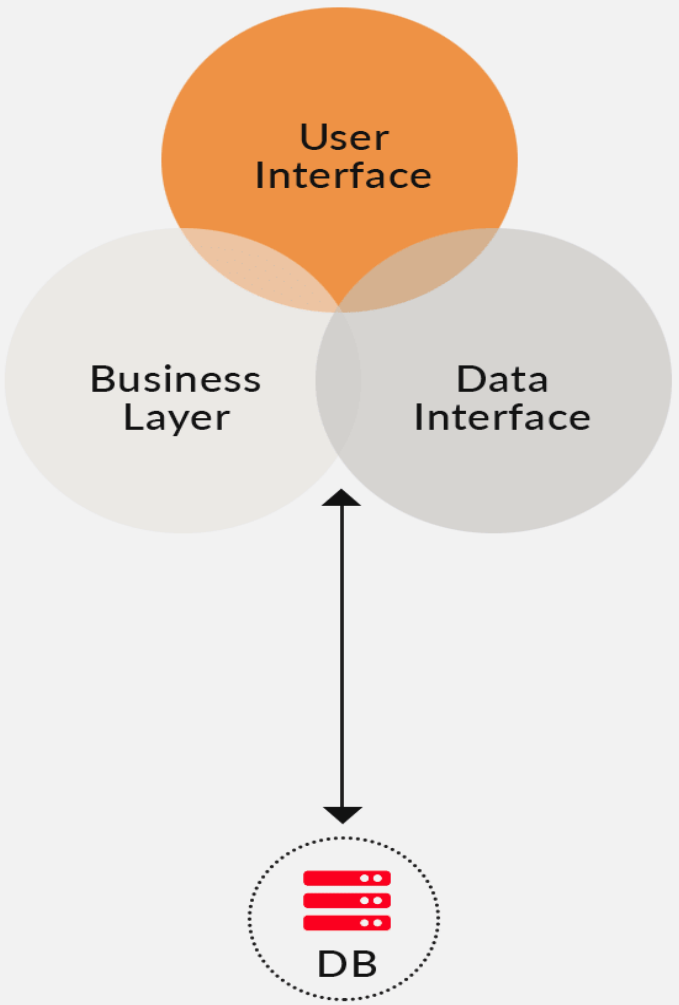
Monolithic applications typically consist of a client-side UI, a database, and a server-side application. Developers build all of these modules on a single code base.

ADVANTAGES OF A MONOLITHIC ARCHITECTURE



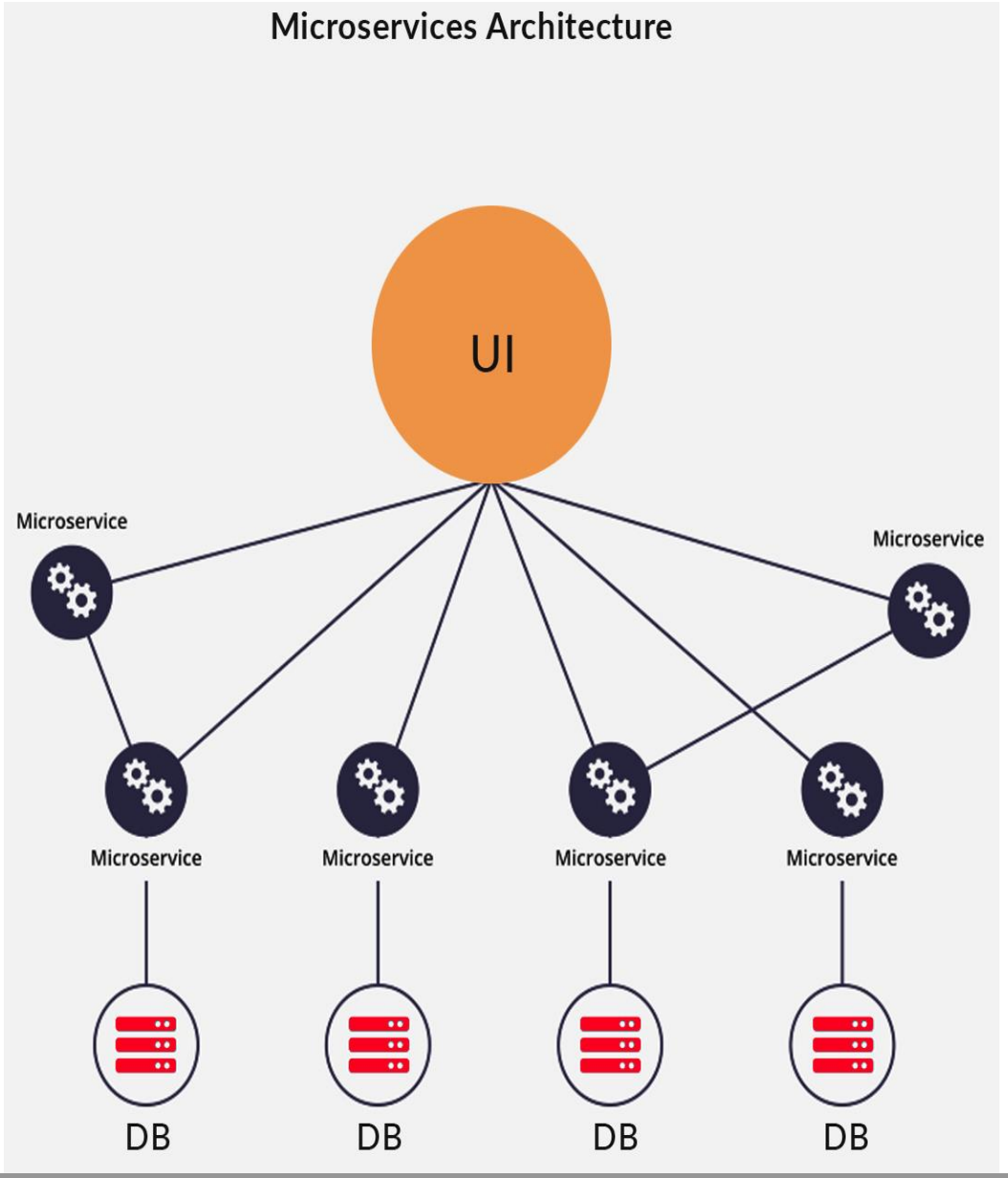
- **Easy deployment** – One executable file makes deployment easier
- **Development** – When an application is built with one code base, it is easier to develop
- **Performance** - In a centralized code base and repository, one API can often perform the same function that numerous APIs perform with microservices
- **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

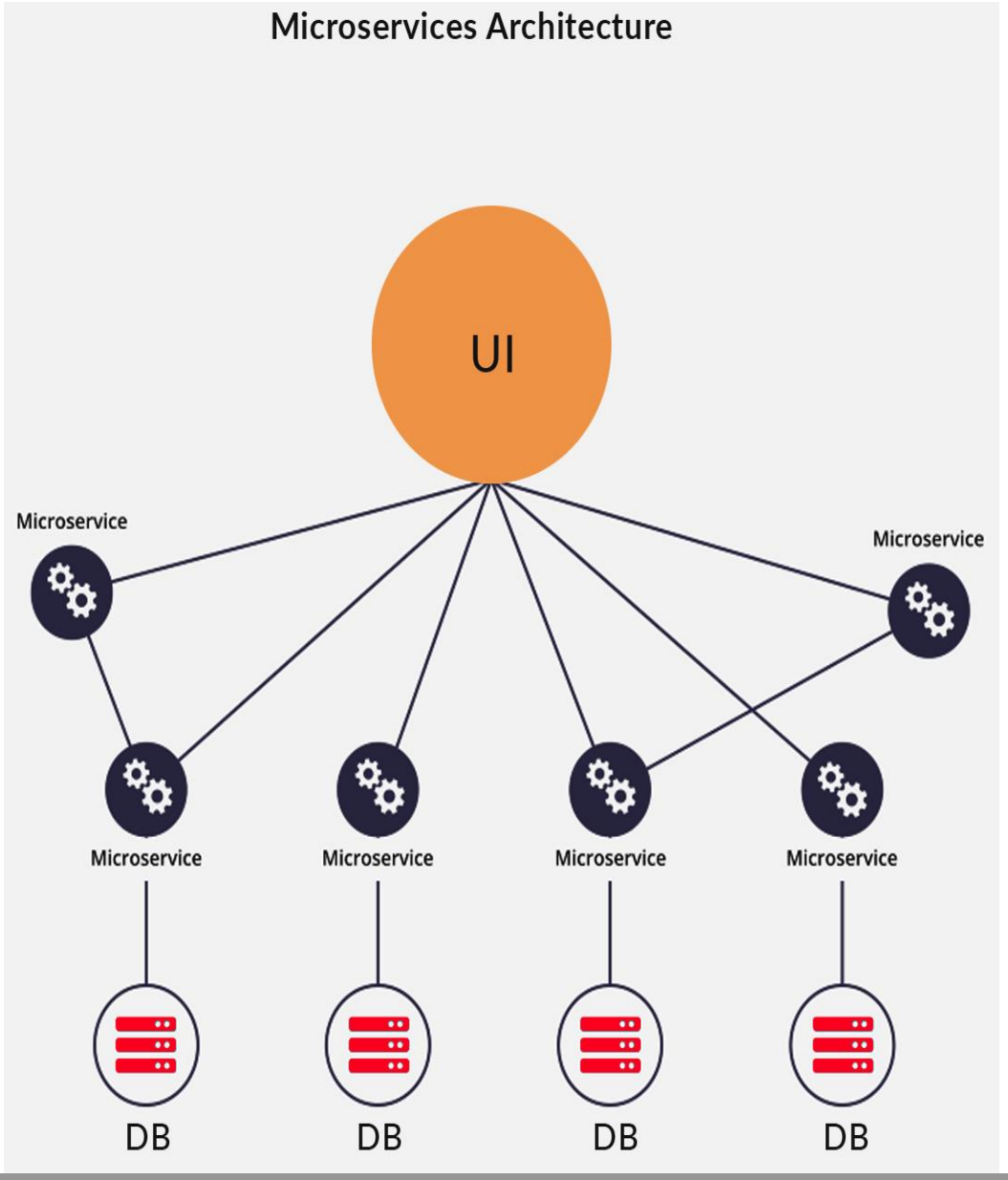
WHAT EXACTLY IS A MICROSERVICES ARCHITECTURE?



Microservices are an architectural approach that composes software into small independent components or services. Each service performs a single function and communicates with other services through a well-defined interface. Because they run independently, you can update, modify, deploy, or scale each service as required.

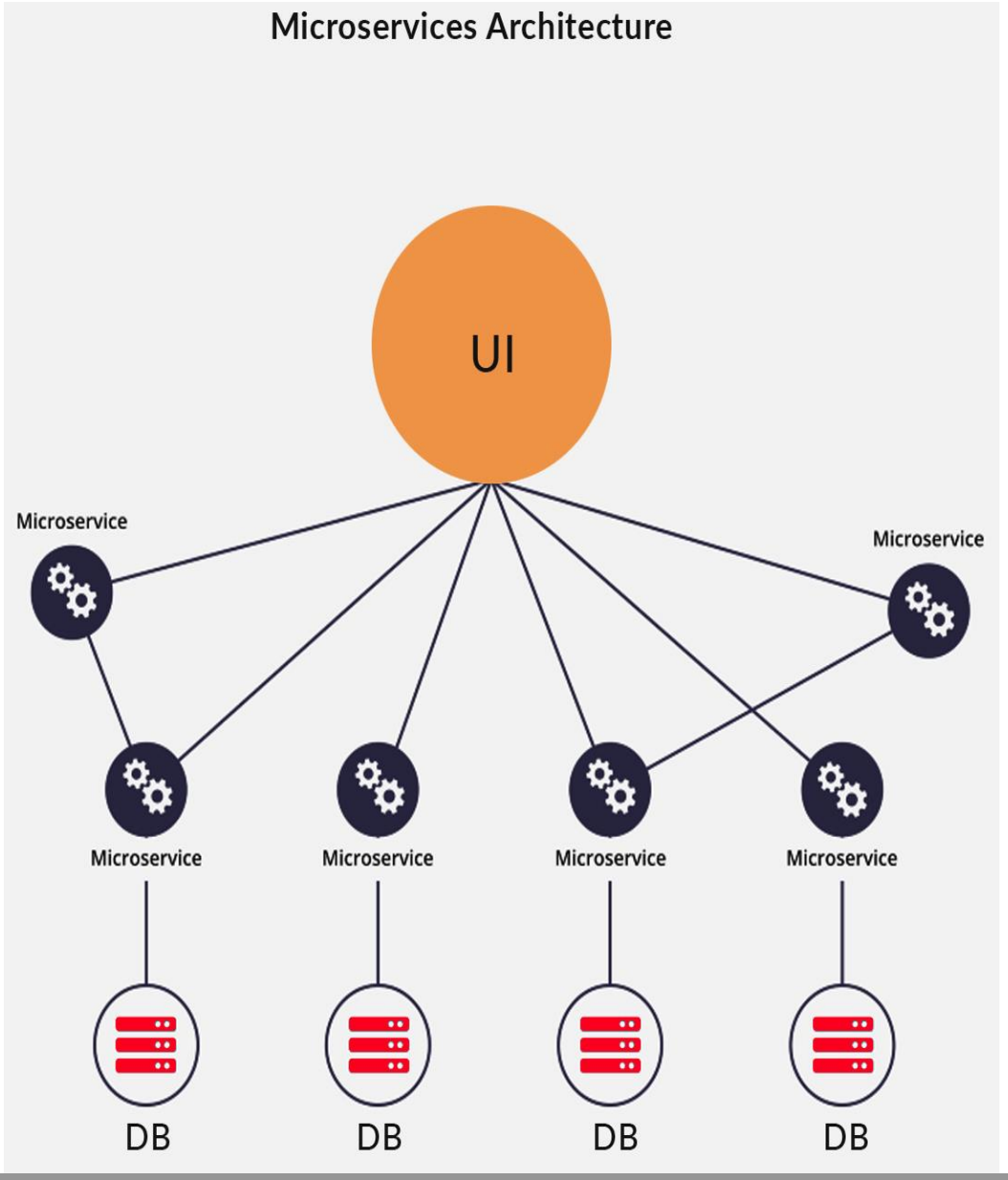
Microservices don't reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole.

ADVANTAGES OF A MICROSERVICES ARCHITECTURE



- **Flexible scaling** - If a microservice reaches its load capacity, new instances of that service can rapidly be deployed to the cluster to help relieve pressure
- **Continuous deployment** – We now have frequent and faster release cycles
- **Independently deployable** – Since microservices are individual units they allow for fast and easy independent deployment of individual features
- **High reliability** – You can deploy changes for a specific service, without the threat of bringing down the entire application
- **Technology flexibility** – Microservice architectures allow teams the freedom to select the tools they desire

DISADVANTAGES OF A MICROSERVICES ARCHITECTURE



- **Development sprawl** – Microservices add more complexity compared to a monolith architecture, since there are more services in more places
- **Exponential infrastructure costs** – Each new microservice can have its own cost for test suite, deployment playbooks, hosting infrastructure, monitoring tools, and more
- **Lack of standardization** – Without a common platform, there can be a proliferation of languages, logging standards, and monitoring
- **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

REFACTORING A MONOLITH TO MICROSERVICES

Strangler Pattern

The Strangler Pattern is a software design and architectural pattern used in the context of legacy system modernization or application migration.

The Strangler Pattern aims to replace or evolve an existing system gradually, without causing any significant disruptions to the overall system functionality.

How the Strangler Pattern addresses legacy system modernization challenges

Legacy systems pose several challenges when it comes to modernization, including:

- **Risk and Complexity**: Legacy systems often have complex codebases and undocumented functionalities. Undertaking a full-scale rewrite can be risky, time-consuming, and costly. The Strangler Pattern helps mitigate these risks by allowing modernization to occur incrementally, one component at a time

- **Business Continuity**: Many legacy systems are critical to an organization's daily operations. Interrupting these systems for a major overhaul can lead to disruptions, downtime, and financial losses. The Strangler Pattern enables the coexistence of old and new components, ensuring that the system remains functional during the migration process

How the Strangler Pattern addresses legacy system modernization challenges

- **Inflexibility**: Legacy systems may not support modern technologies or meet current business requirements. The Strangler Pattern addresses this by allowing organizations to gradually introduce new technologies and features while retaining existing functionality

- **Resource Allocation**: A complete rewrite of a large legacy system requires significant resources, including time, money, and skilled personnel. The Strangler Pattern enables organizations to allocate resources based on priority, focusing on areas that require immediate attention

Identifying scenarios suitable for the Strangler Pattern

Some common situations where the Strangler Pattern can be applied include:

- **Monolithic Legacy Systems**: When dealing with large monolithic applications that have grown complex and difficult to maintain, using the Strangler Pattern allows the gradual replacement of components, making the modernization process more manageable

- **Business-Critical Applications**: If the legacy system is critical to the organization's day-to-day operations, a full replacement could cause significant disruptions. The Strangler Pattern enables continuous operation during the migration, minimizing downtime

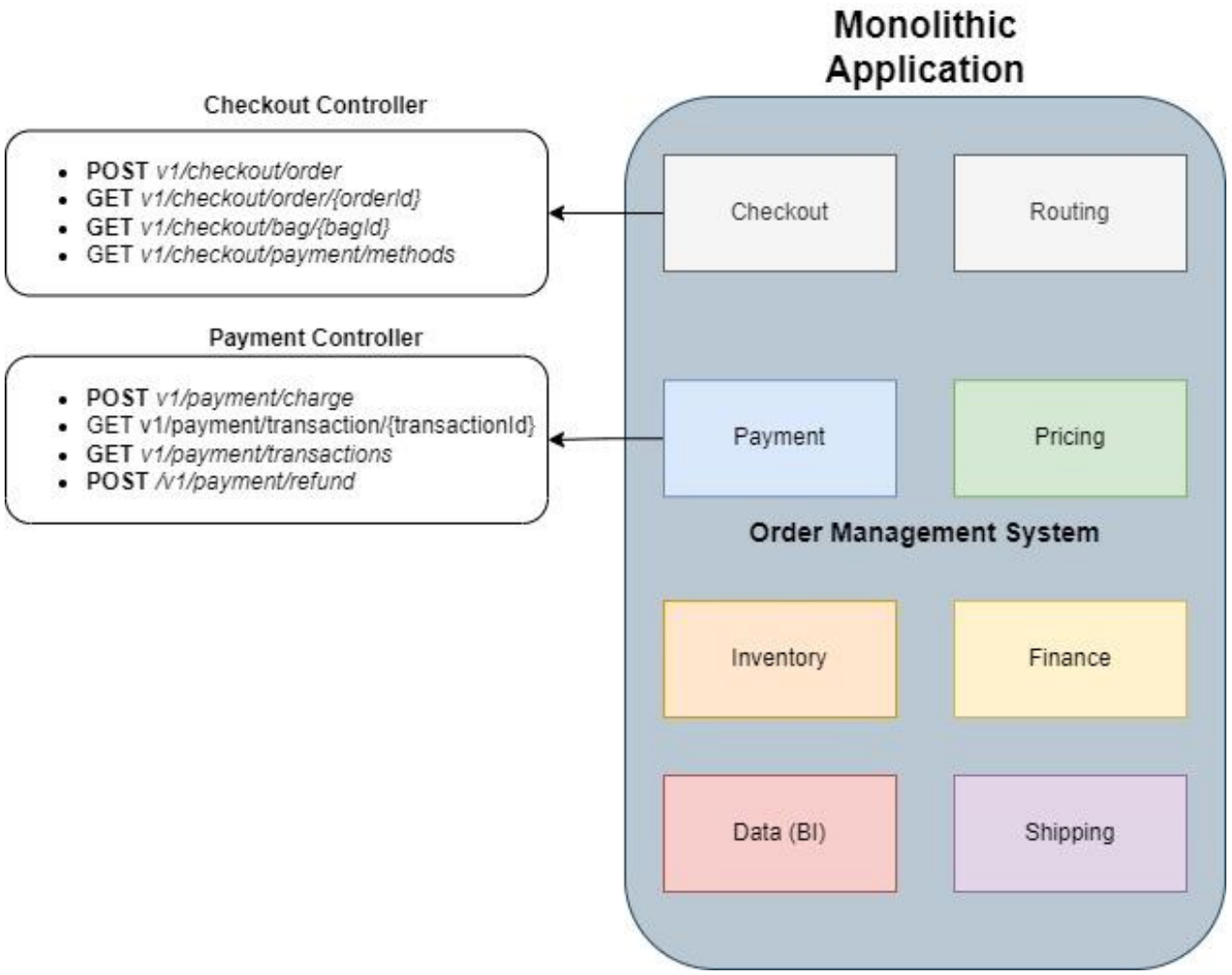
Identifying scenarios suitable for the Strangler Pattern

- **Limited Resources**: When an organization lacks the resources (time, budget, or skilled personnel) required for a complete rewrite, adopting the Strangler Pattern allows for incremental modernization without overwhelming the team

- **Mitigating Risk**: The Strangler Pattern is ideal when the organization wants to mitigate risks associated with large-scale modernization projects, as each step is carefully validated before proceeding further

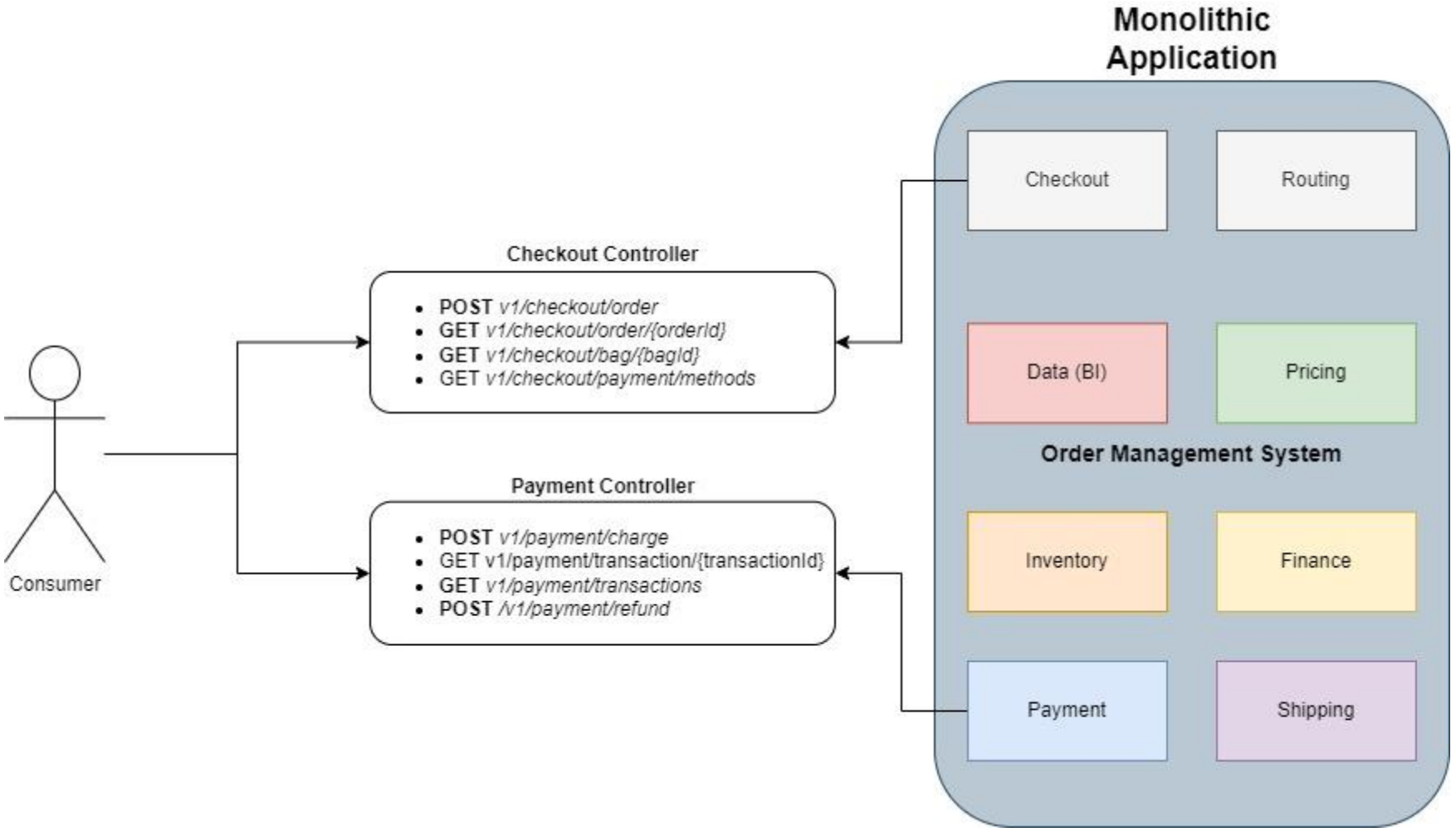
REFACTORING A MONOLITH TO MICROSERVICES

Strangler Pattern – Lets simulate a real world example



Imagine we are working with a monolithic application responsible for managing an entire ecommerce order system.

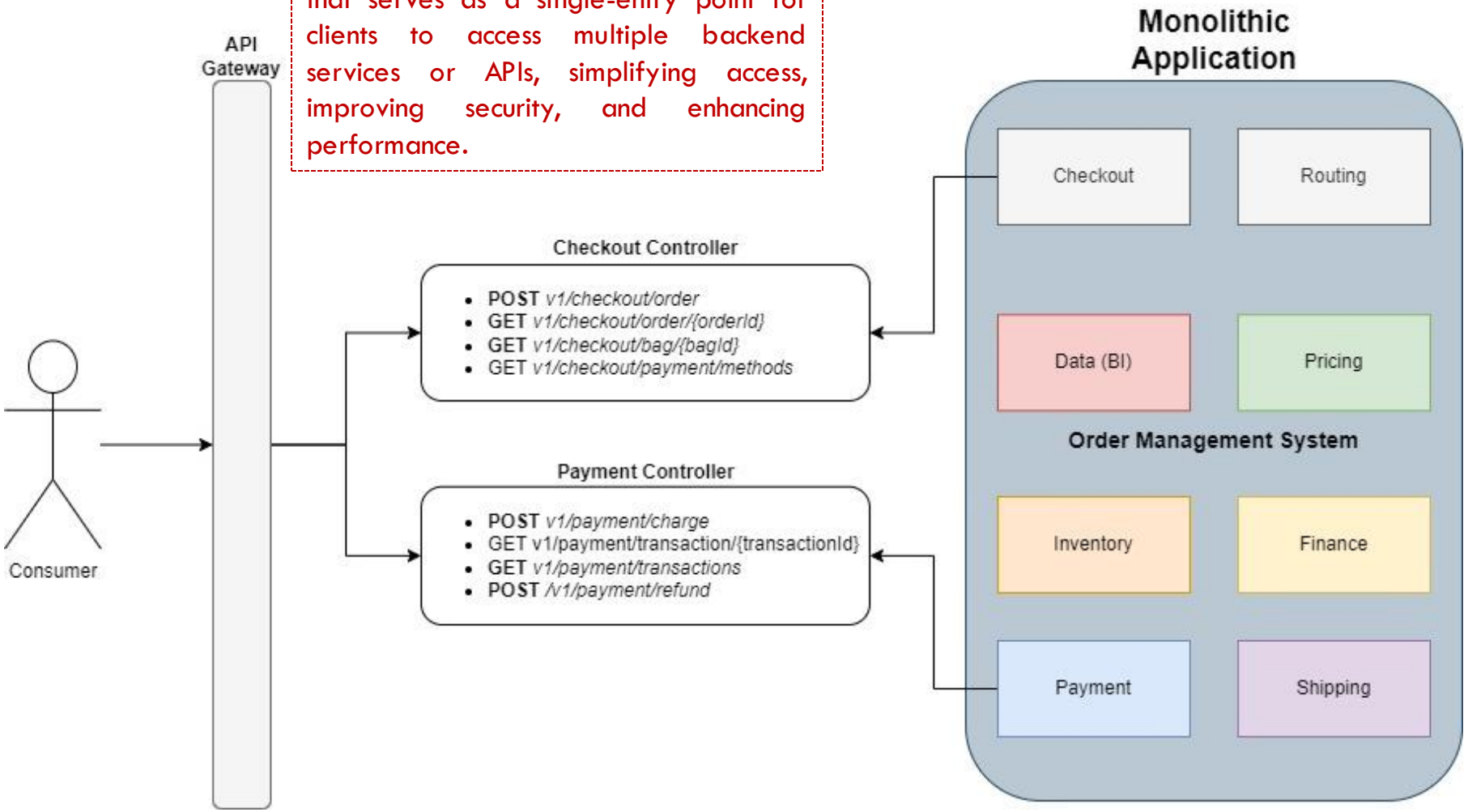
REFACTORING A MONOLITH TO MICROSERVICES



Let's consider the scenario where the aim is to extract the checkout and payment modules from this monolithic application and transform them into independent services.

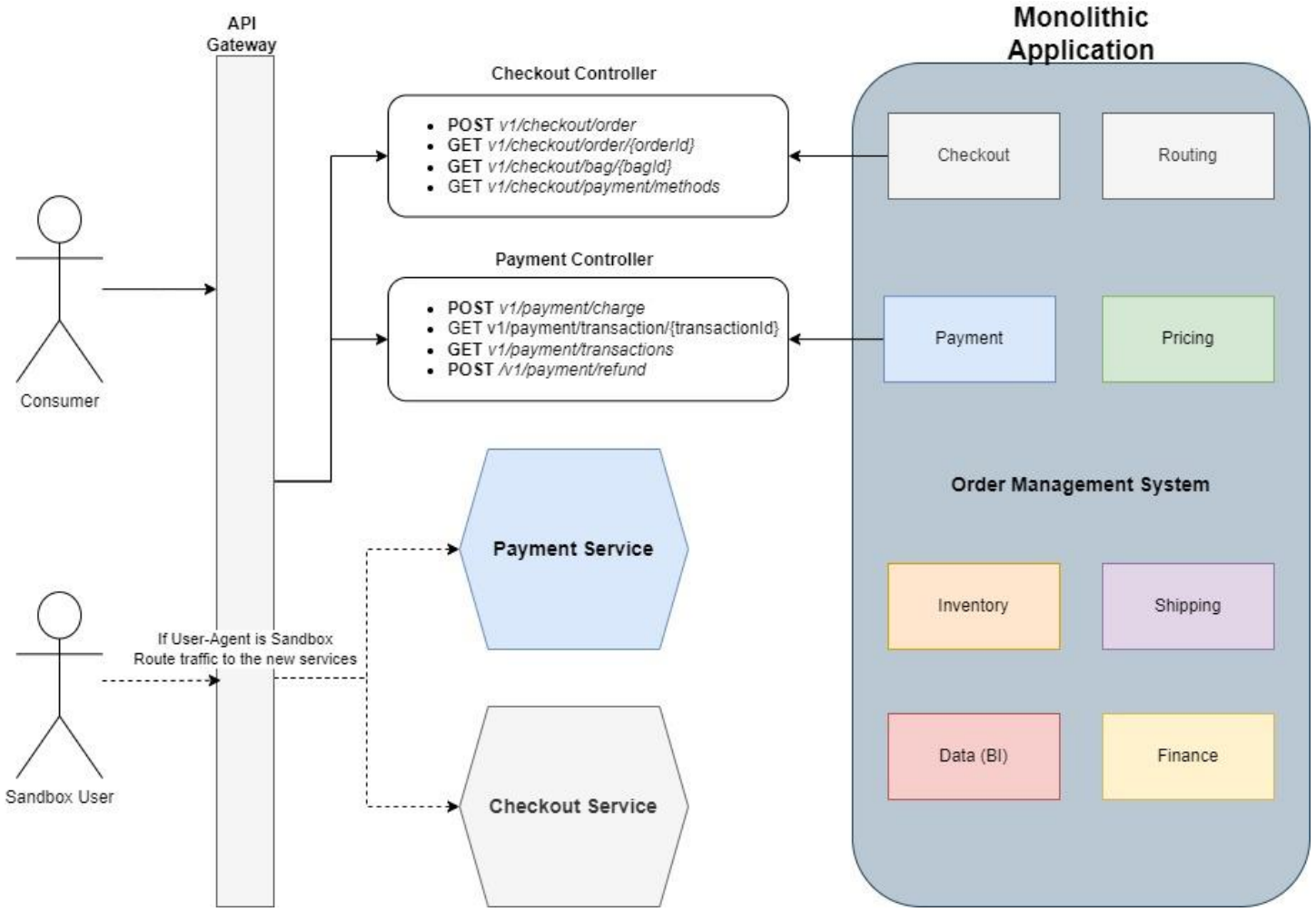
REFACTORING A MONOLITH TO MICROSERVICES

An API Gateway is a centralized server that serves as a single-entry point for clients to access multiple backend services or APIs, simplifying access, improving security, and enhancing performance.



Our monolith is currently providing all these services to one or more consumers. This means that we need to find a way to make this migration seamless: the API Gateway comes to the rescue!

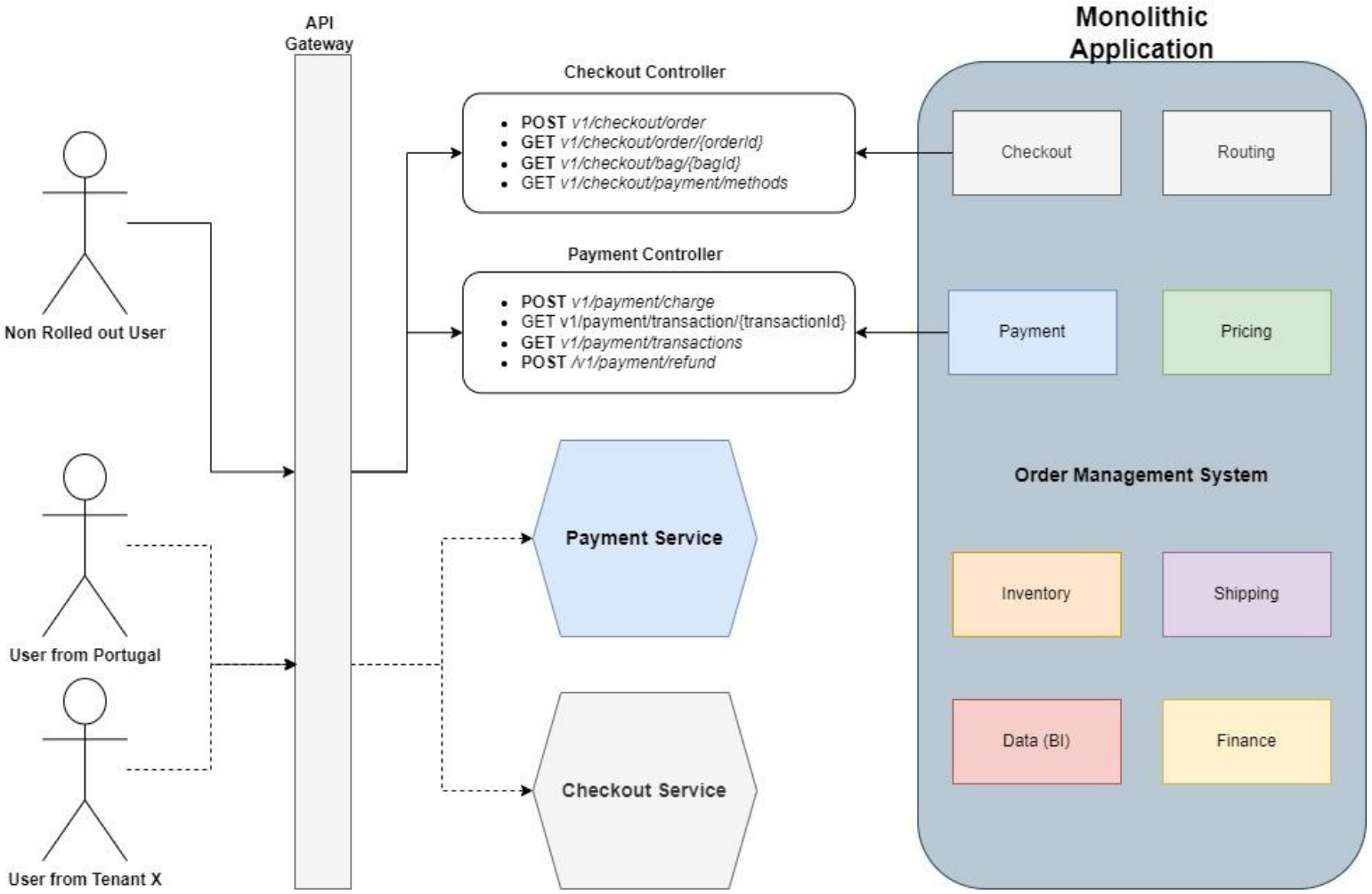
REFACTORING A MONOLITH TO MICROSERVICES



We can create a Sandbox user and configure our API Gateway to handle requests from this user agent. When the HTTP request's user agent matches "Sandbox-User," the API Gateway will forward the request to the Payment Service and Checkout Service, bypassing the legacy endpoints.

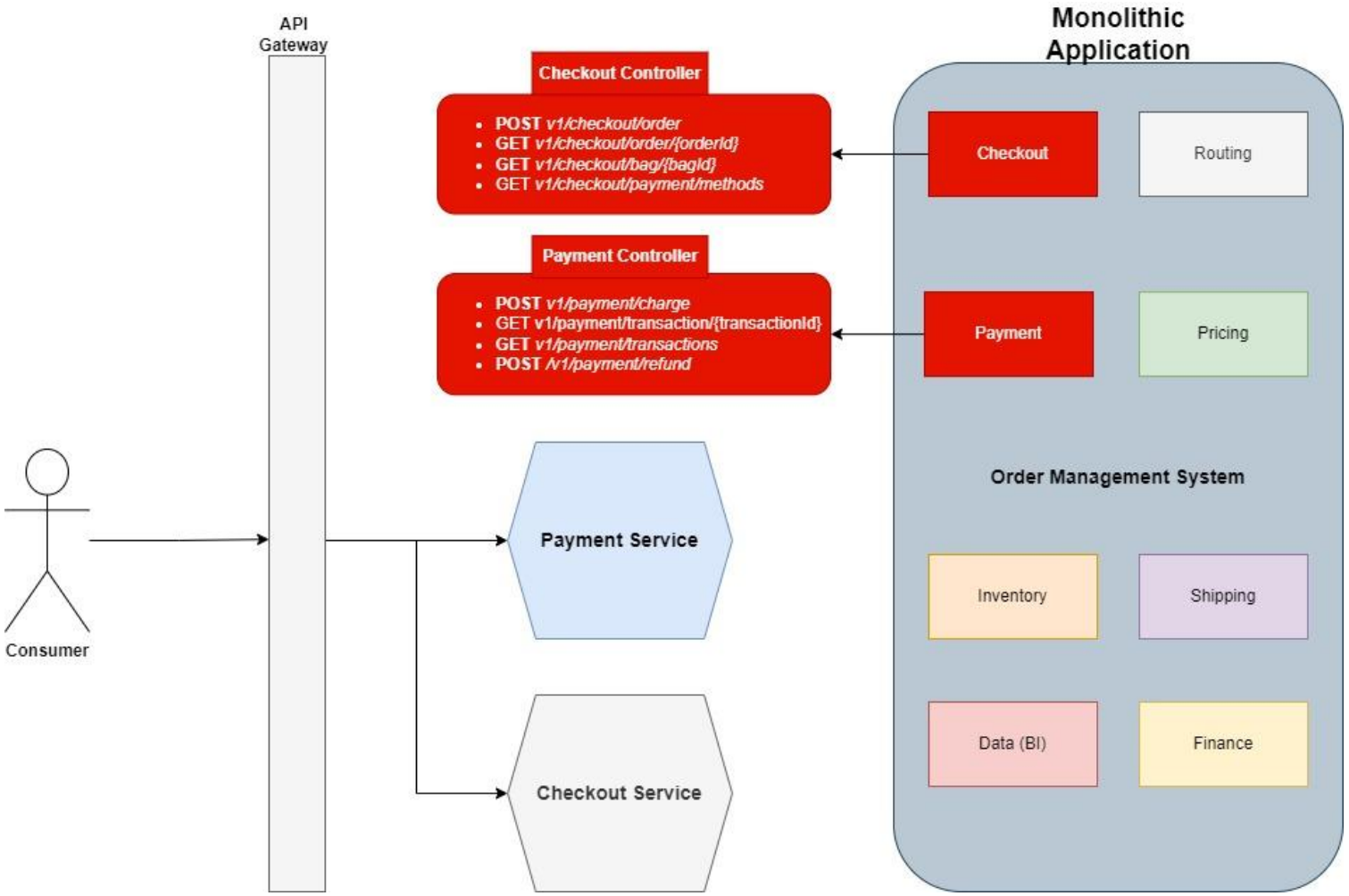
Now our system is completely testable. We can perform all the checkout and payment operations through the new architecture using the Sandbox user.

REFACTORING A MONOLITH TO MICROSERVICES



After ensuring that the checkout and payment services are ready for the production environment, we can begin routing real users through the API Gateway. This rollout process can be implemented using various strategies, but the key is to phase the rollout gradually, starting with lower throughput and gradually increasing it until the legacy endpoints are fully discontinued.

REFACTORING A MONOLITH TO MICROSERVICES



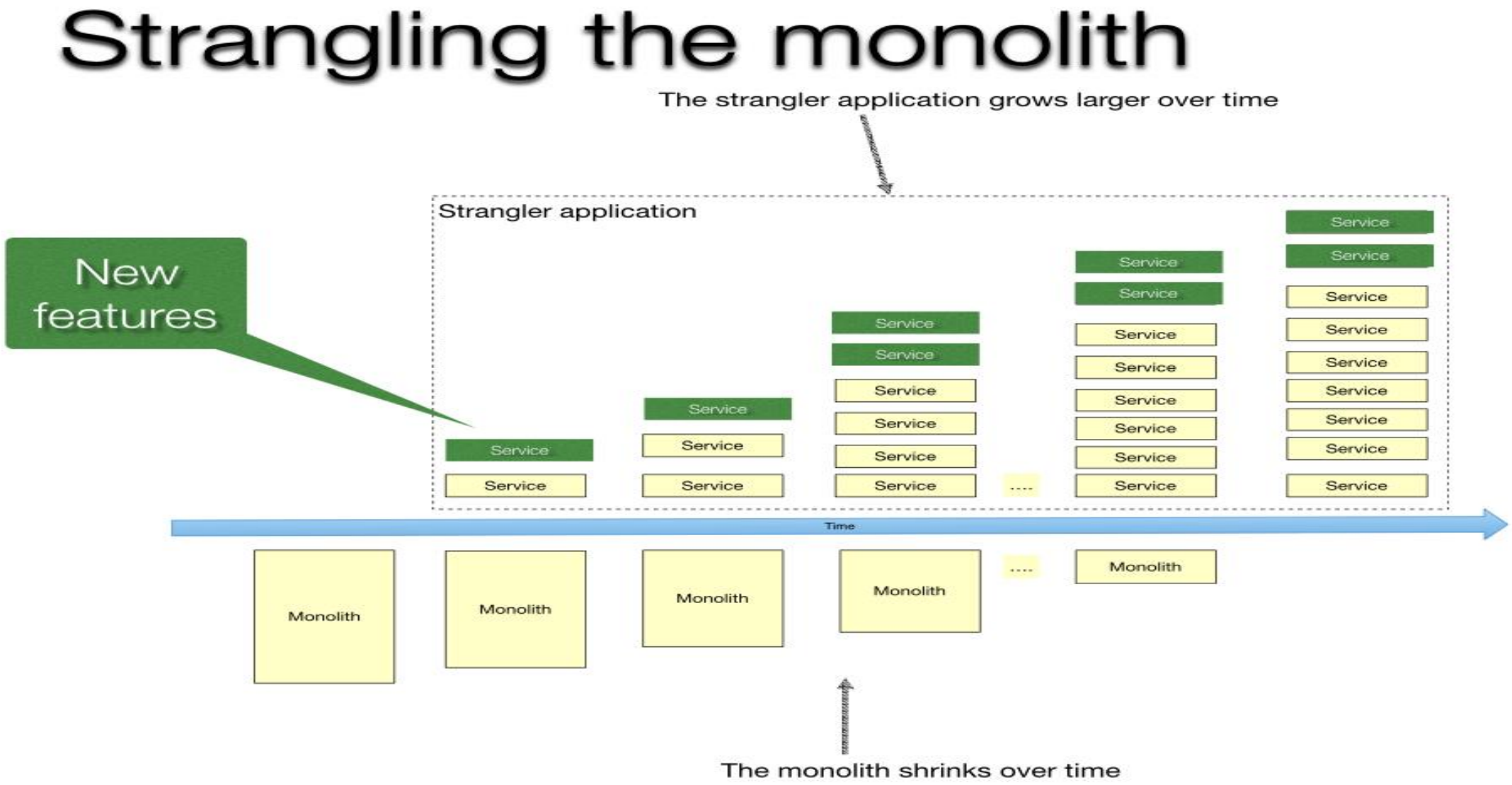
Finally, when all of your users are successfully utilizing the new services, you can decommission obsolete modules and endpoints.

Strangler Pattern – In conclusion

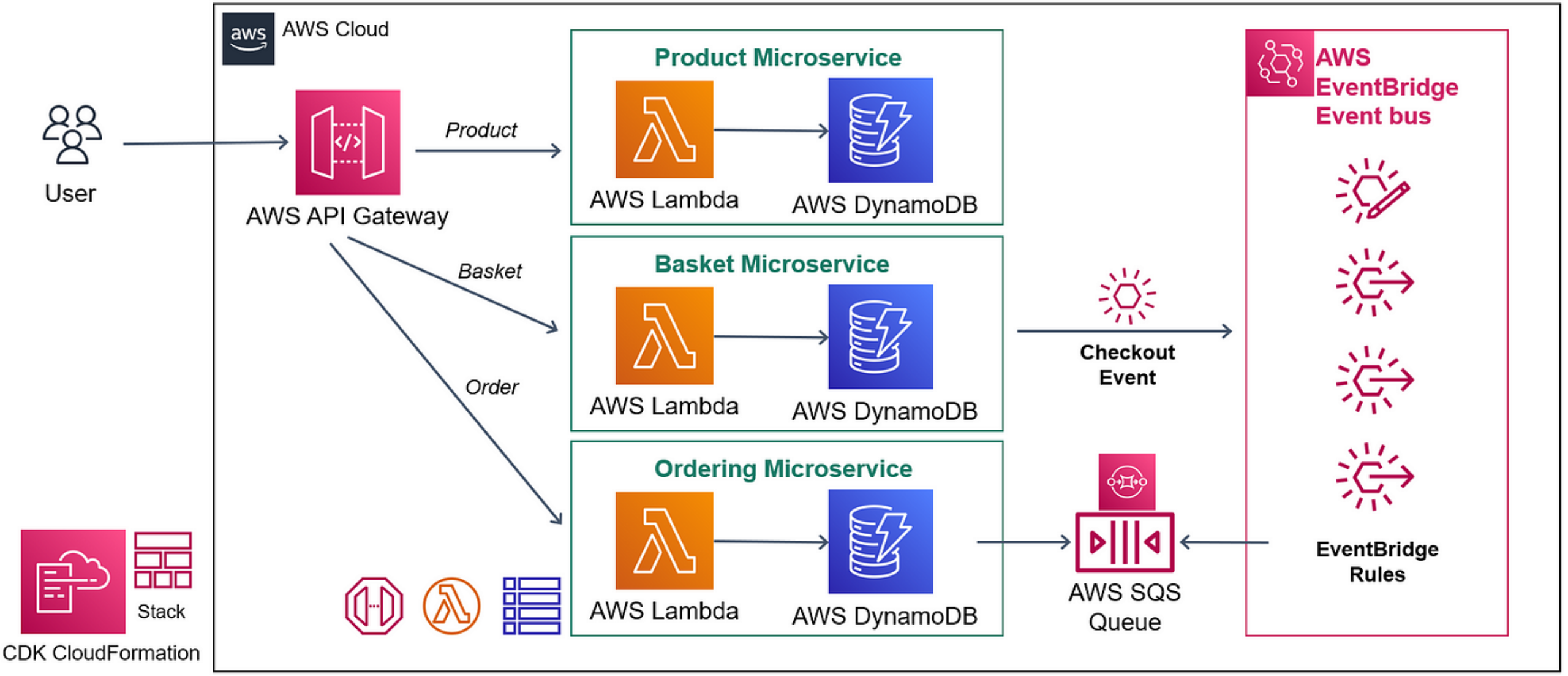
The Strangler Pattern offers a powerful and pragmatic approach to modernizing legacy systems. By gradually evolving the architecture, organizations can reduce risk, maintain business continuity, and optimize resource utilization. The introduction of an API Gateway further enhances the migration process, providing a seamless way to connect consumers with the new microservices while coexisting with the legacy system.

Embracing the Strangler Pattern empowers organizations to modernize their systems incrementally, avoiding the pitfalls of big-bang rewrites.

REFACTORING A MONOLITH TO MICROSERVICES



MICROSERVICES EXAMPLE – SERVERLESS E-COMMERCE APP

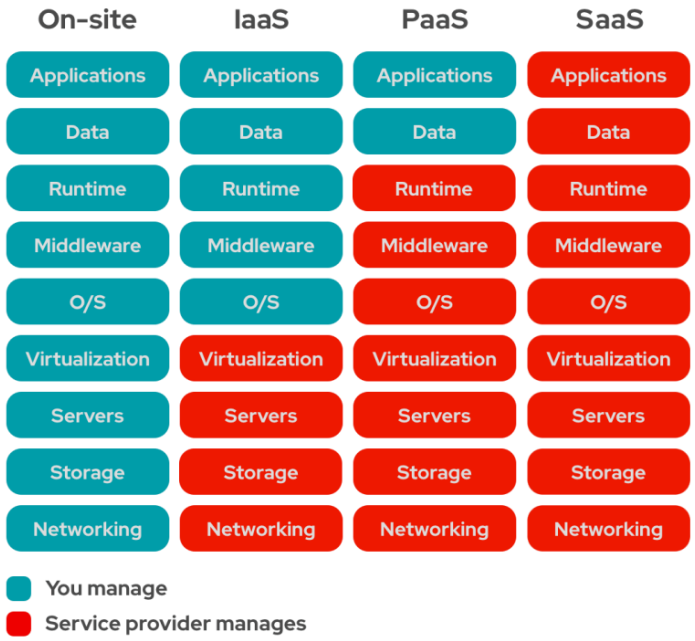


CROSS-CUTTING CONCERN

- **On-site Computing VS Cloud Computing**: Which runtime environment to choose?

- **Cloud Strategy**: Cloud First; Cloud Native; Lift and Shift; …

- ***as-a-Service***: Which level of governance to choose
  - Infrastructure-as-a-Service (IaaS)
  - Platform-as-a-Service (PaaS)
  - Software-as-a-Service (SaaS)



- **Hybrid Architecture**: Security; Integration; Latency; …

Q&A

YOOX
NET-A-PORTER
GROUP

YOOX
NET-A-PORTER
GROUP

Marco Petrucci: marco.pretrucci@ynap.com
Francesco Pichierri: francesco.pichierri@ynap.com