Технології створення Web-застосунків

Доц. Попівщий В.І. ЗНУ каф.ПЗАС 2020

Опис курсу

- *Курс має на **меті** сформувати у студентів цілісне уявлення про сучасні технології розробки додатків з мікросервісною архітектурою.*
- *Розглянуто стек технологій платформи .NET Core для мікросервісів, прийоми роботи з інструментальним засобом Microsoft Visual Studio 2019, що підтримує створення додатків з мікросервісною архітектурою.*
- *Розглянуто розгортання мікросервісів в хмарному сервісі Azure.*
- *Для формування необхідних навичок запропоновано шість навчальних завдань.*

ОСНОВНІ ДЖЕРЕЛА

1. Хорсдал К. Микросервисы на платформе .NET. – СПб. : Питер, 2018. 352 с.
2. Ньюмен С. Создание микросервисов. – СПб.: Питер, 2016. – 304 с.
3. Фаулер М. Шаблоны корпоративных приложений. – М.: Вильямс, 2016. – 544 с.
4. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. – СПб.: Питер, 2019. 544 с.
5. Cole Matt R. Hands-On Microservices with C#. – Packt Publishing, 2018. – 234 p.
6. Newman Sam Monolith to Microservices. – O'Reilly Media, 2020. – 256 p.
7. Gaurav Aroraa, Ed Price Hands-On Microservices with C# 8 and .NET Core 3 Third Edition. – Packt Publishing, 2020. – 451 p.

8. Morgan Bruce, Paulo A. Pereira **Microservices in Action**. – Manning Publications, 2019. – 366 p.

ТРЕНІНГИ

- Тренінг «Мікросервіси для Java розробників» https://itcluster.lviv.ua/paged/training-microservices-for-java-developers/
- Курси Microservices https://www.nobleprog.com.ua/kursy-microservices

Огляд відеокурсів

1. Linkedin - Learning Creating Your First Spring Boot Microservice 2019
2. **Lynda - Azure Microservices with .NET Core for Developers 2020**
3. Pluralsight - Building Microservices (2019)
4. Pluralsight – Microservices The Big Picture (2018)
5. ASP.NET Core 2.0 E-commerce Web Site Based on Microservices (2019)
6. Lynda - Microservices Design Patterns (2020)
7. Packt – A Beginner's Guide to a Microservices Architecture
8. O'Reilly - Building Microservice Systems with Docker and Kubernetes

Модуль 1. Мікросервісна архітектура.

- Мікросервіси — **архітектурний стиль** (MicroServices Architecture, MSA), за яким єдиний застосунок будується як сукупність **невеличких сервісів**, кожен з яких працює у своєму **власному процесі** і комунікує з рештою, використовуючи легковагові механізми, зазвичай HTTP.
- Ці сервіси будуються навколо **бізнес-потреб** і розгортаються незалежно з використанням зазвичай повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі по собі вони можуть бути написані **з використанням різних мов** і технологій **зберігання даних**.

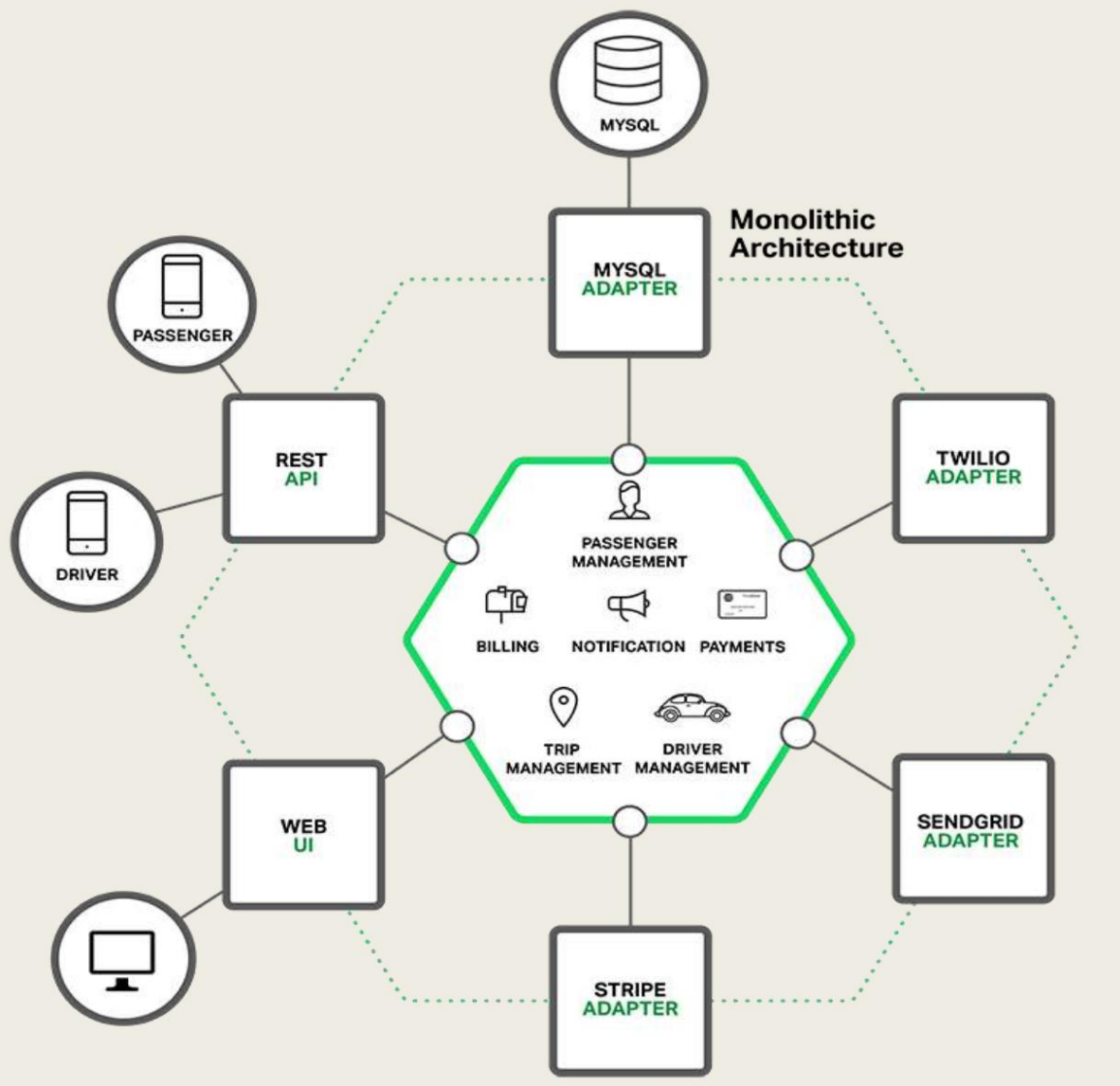
Причини використання

- Одна з причин використання мікросервісів полягає в тому, що компанії хочуть мати **можливість швидко щось змінювати**, щоб швидше реагувати на зміни бізнес-вимог, випереджати конкурентів.
- Мікросервіси допомагають розробникам доставляти зміни швидше, безпечніше і з більш високою якістю, тобто **зберігати швидкість розвитку продукту**, навіть коли той стає неосяжних розмірів.
- Адже не тісно зв’язані сервіси дають можливість проводити зміни з **більшою частотою ітерацій** мінімізуючи вплив змін на решту частин системи.

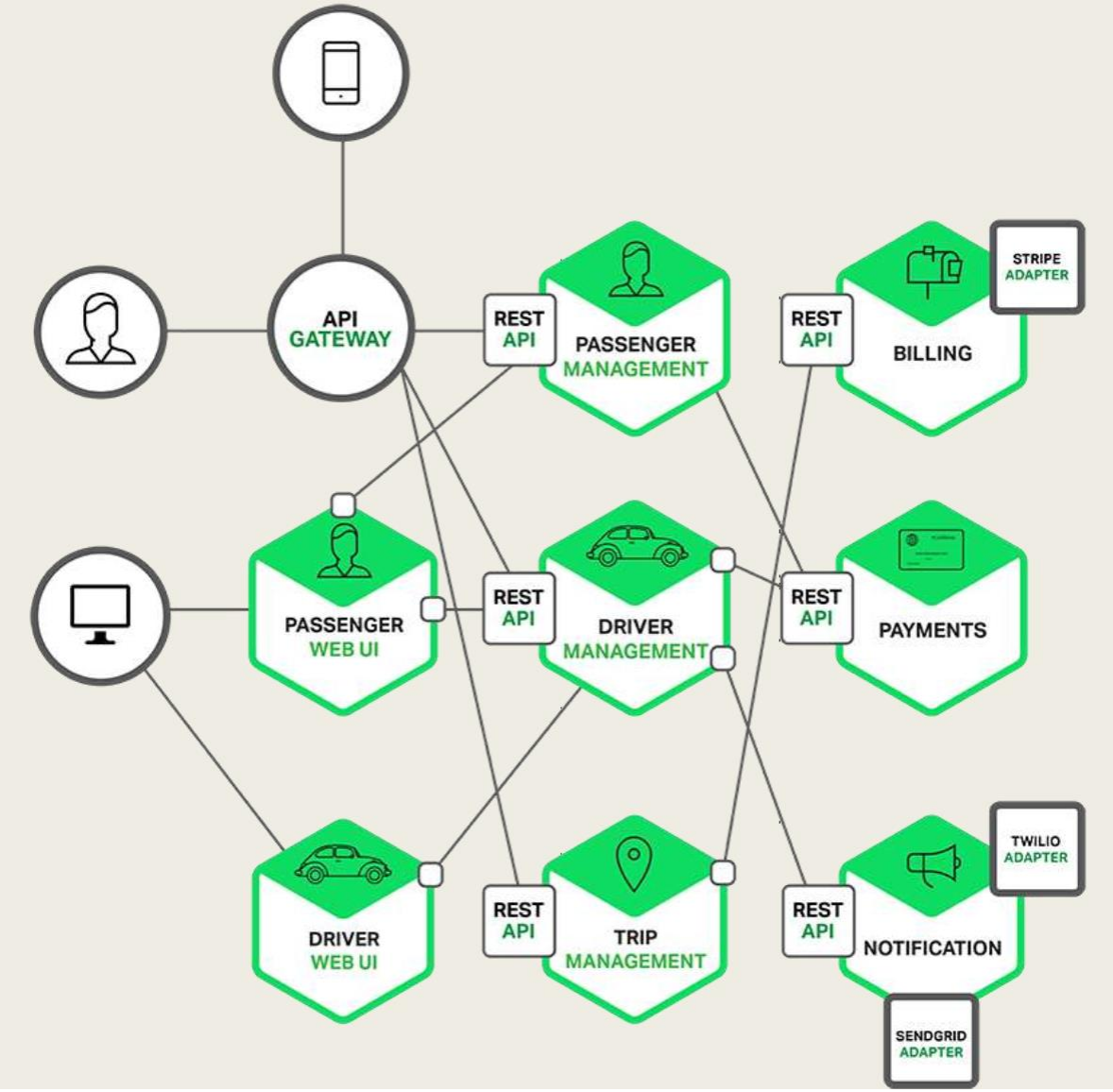
МОНОЛІТ

- **Моноліт** будуються як єдине ціле. Будь які зміни, навіть самі невеликі, потребують перебудови та розгортання всього додатку.
- З часом стає складніше зберігати хорошу модульну структуру, зміни логіки одного модуля мають тенденцію впливати на код інших модулів.
- Монолітні програми також може бути важко масштабувати, коли різні модулі мають конфліктні вимоги до ресурсів.

Архітектура Моноліту

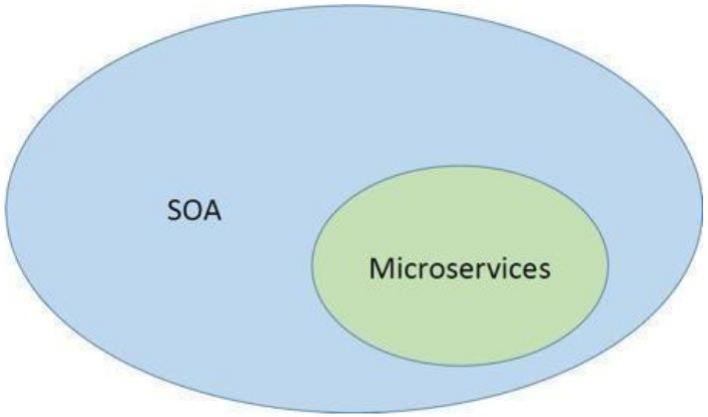


Архітектура Мікросервісу

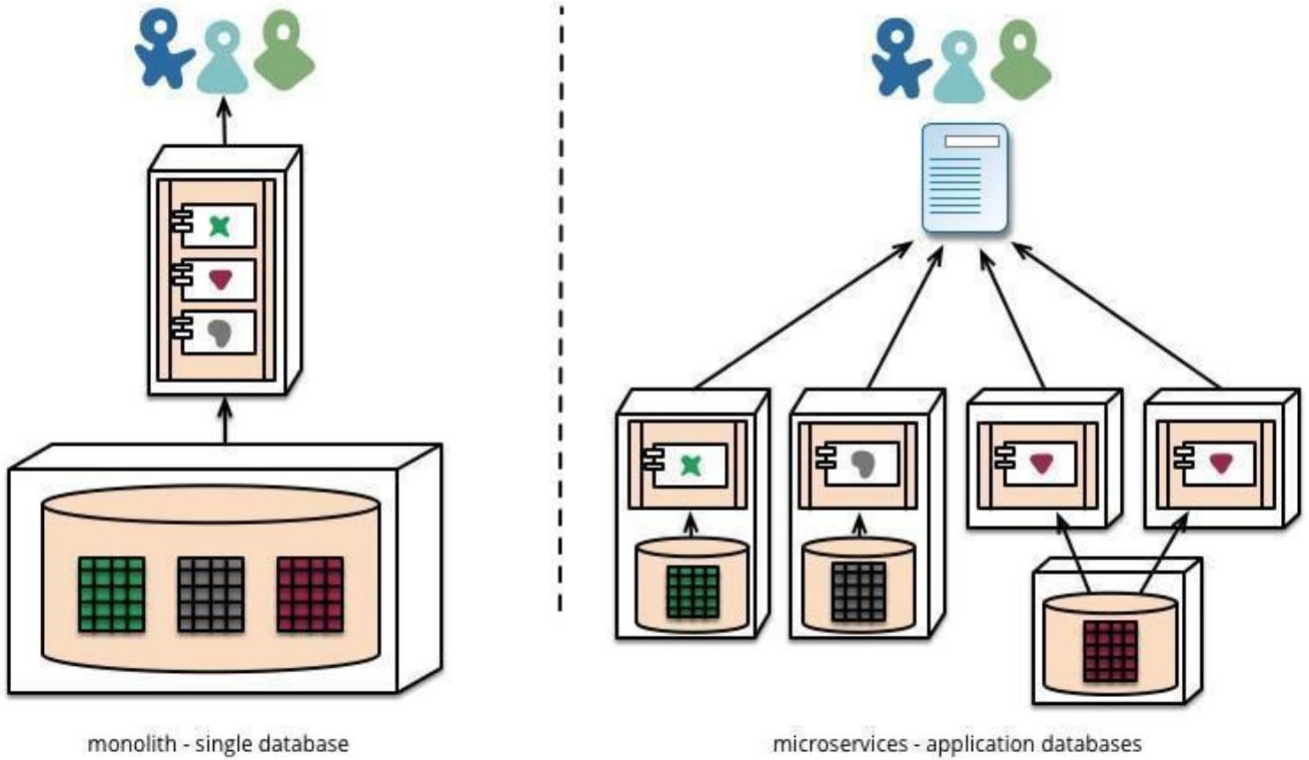


Взаємовідношення MSA та SOA

- Мікросервісна архітектура взагалі є підмножиною типу SOA
- Мікросервісну архітектуру слід вважати підходом до реалізації сервісо-орієнтованої архітектури



Різниця в організації даних



monolith - single database

microservices - application databases

ПЕРЕВАГИ МІКРОСЕРВІСІВ

- Високий рівень незалежності: незалежна розробка, незалежне розгортання
- Незалежне масштабування
- Невелика кодова база зменшує кількість конфліктів та дозволяє швидко залучувати нових розробників
- Простота заміни однієї реалізації сервісу іншою
- Простота додавання нового функціоналу в систему
- Ефективне використання ресурсів
- Еластичність: вихід з ладу одного сервісу зазвичай не призводить до виходу з ладу всієї системи
- Сервіси організовані відносно бізнес логіки яку вони виконують
- Кожен сервіс незалежно від інших може бути реалізований за допомогою будь-якої мови програмування, СУБД, та ін.
- Архітектурно побудовані за симетричним принципом (виробник-споживач)

НЕДОЛІКИ

- Значні накладні витрати на інфраструктуру, моніторинг і операційні дії
- Ускладнене налагодження
- Обмеження типу «одна команда — один сервіс» викликає бар'єри
- Незалежність сервісів призводить до дублювання коду
- Проблеми зі стабільністю мережевого зв'язку між сервісами
- Ускладнене тестування і розгортання
- Ускладнене забезпечення безпеки

Мікросервіси як подолання складності

- Багато відомих компаній вирішили проблему моноліту, прийнявши архітектуру мікросервісів, замість того, щоб будувати єдиний монструозний моноліт.
- Серед них маємо **Amazon**, **eBay**, **Walmart**, **Netflix**, **SoundCloud**, **Spotify**, **Twitter**, **Stripe**, **PayPal**, **Uber** та **Medium**.

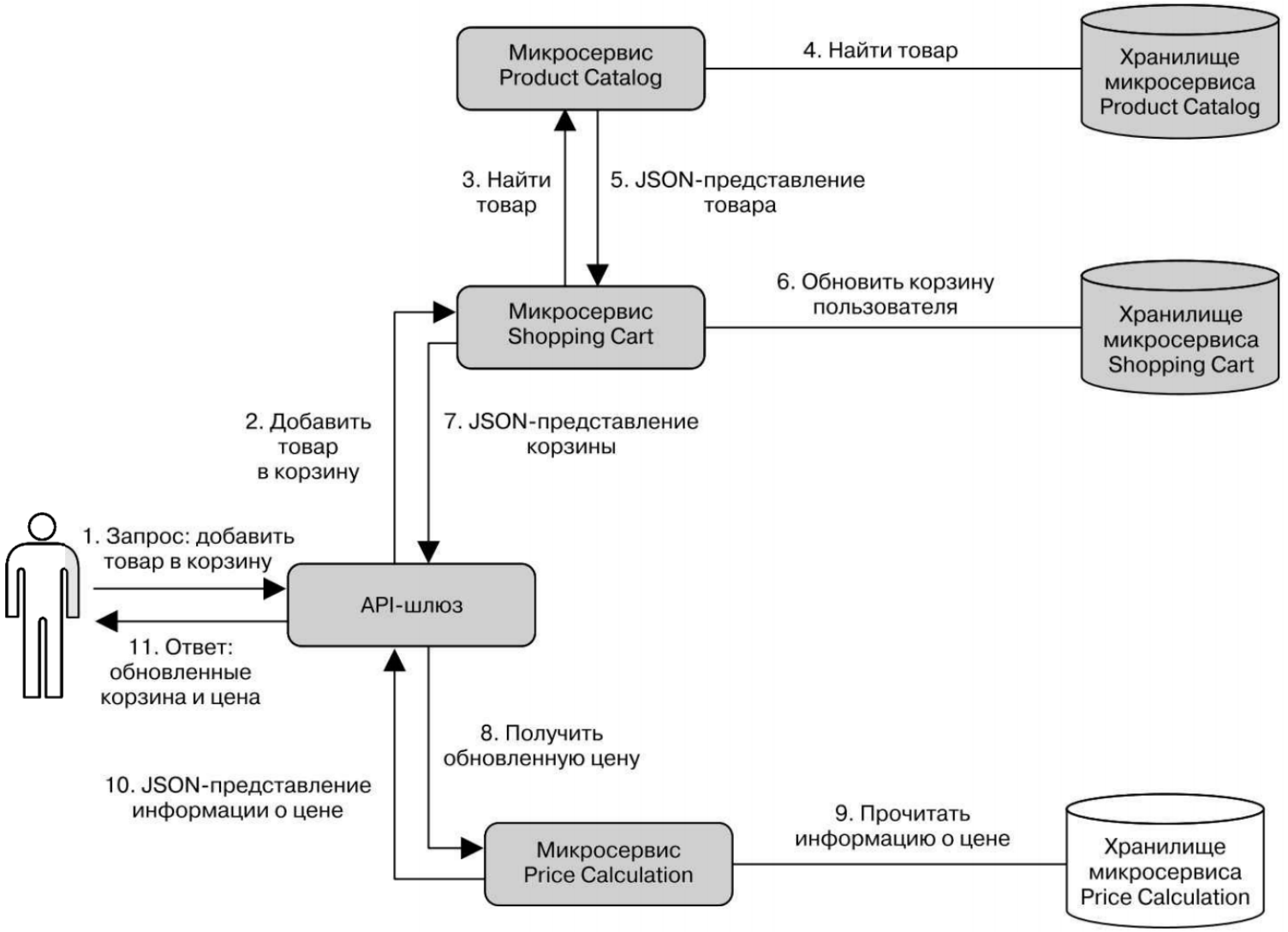
МІКРОСЕРВІСИ НА МАРШІ

- Згідно зі звітом VotE: DevOps, 2H 2019, компанії можуть швидше працювати з хмарою і бути більш ефективними завдяки використанню хмарних мікросервісів.
- **Майже половина підприємств (49%) вважають їх освоєння пріоритетним напрямком розвитку своїх IT-інфраструктур.**

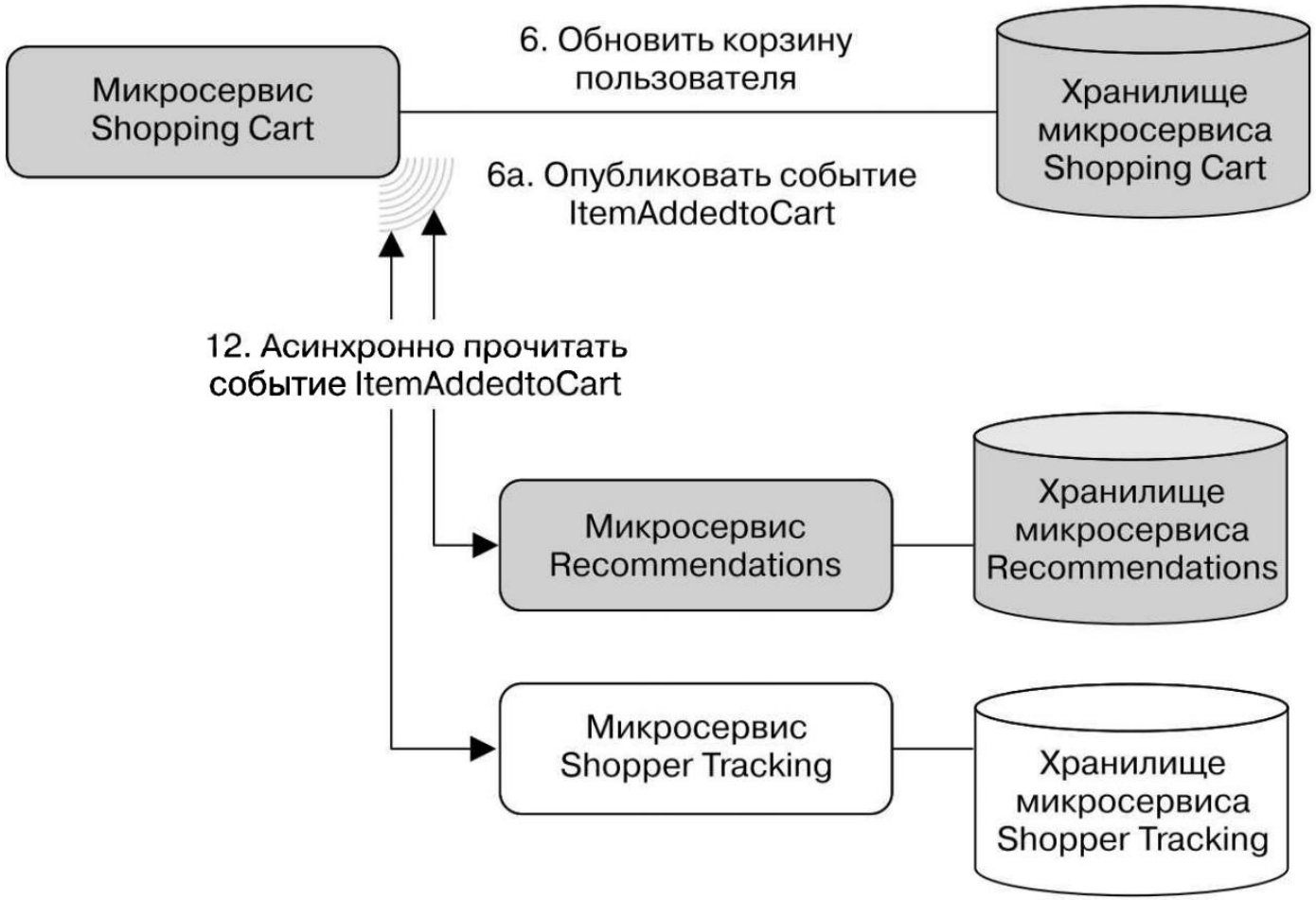
Обслуговування запитів користувача: приклад спільної роботи мікросервісов [1], с. 35

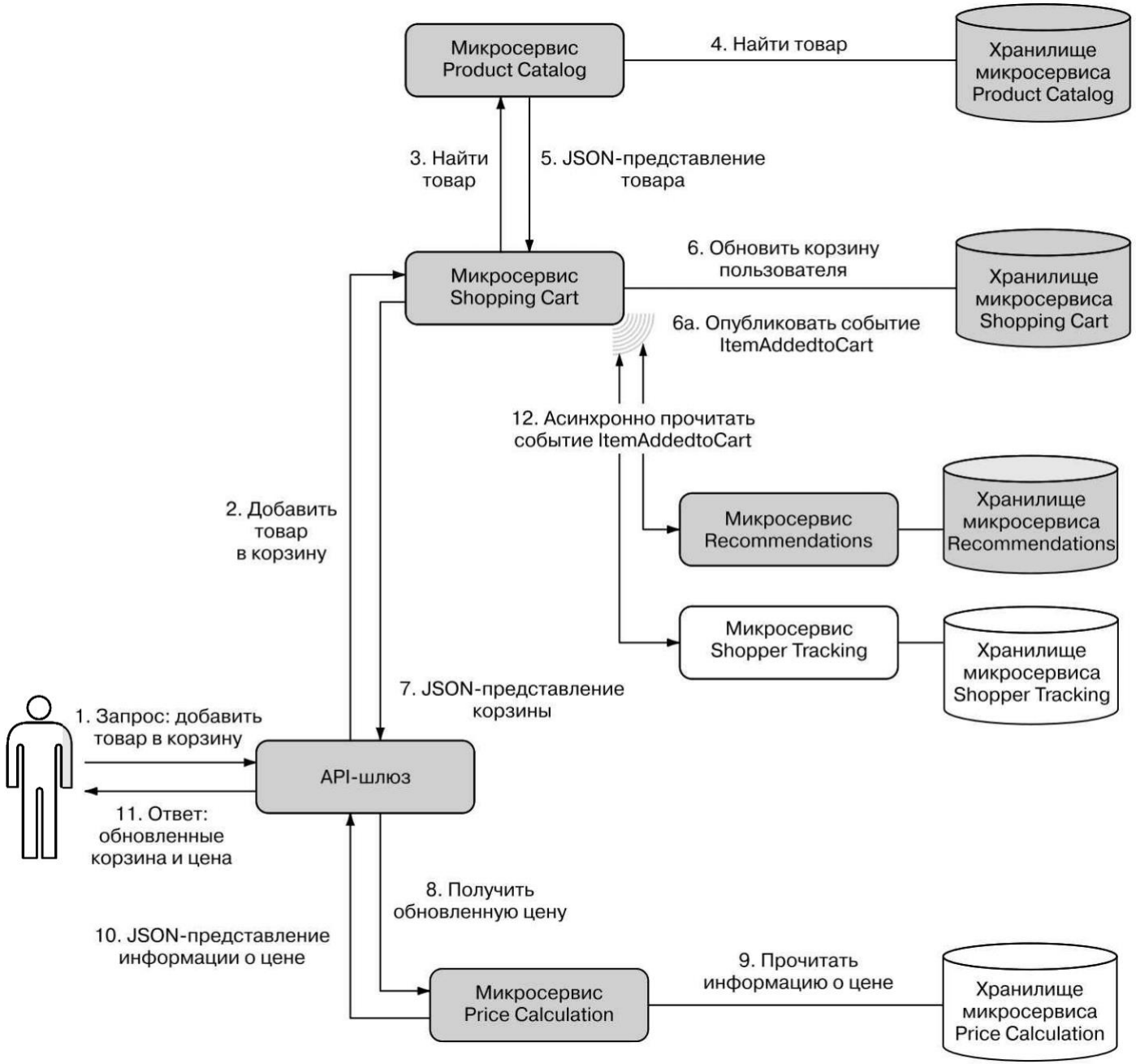
- Розглянемо приклад: відвідувач інтернет-магазину додає товар в корзину замовлень.

Основна обробка призначеного для користувача запиту



Мікросервіс Shopping Cart публікує події, а інші підписані на нього мікросервіси реагують на них




Микросервис Product Catalog
4. Найти товар
Хранилище микросервиса Product Catalog
3. Найти товар
5. JSON-представление товара
Микросервис Shopping Cart
6. Обновить корзину пользователя
Хранилище микросервиса Shopping Cart
6a. Опубликовать событие ItemAddedtoCart
12. Асинхронно прочитать событие ItemAddedtoCart
2. Добавить товар в корзину
Микросервис Recommendations
Хранилище микросервиса Recommendations
Микросервис Shopper Tracking
Хранилище микросервиса Shopper Tracking
7. JSON-представление корзины
1. Запрос: добавить товар в корзину
API-шлюз
11. Ответ: обновленные корзина и цена
8. Получить обновленную цену
10. JSON-представление информации о цене
Микросервис Price Calculation
9. Прочитать информацию о цене
Хранилище микросервиса Price Calculation

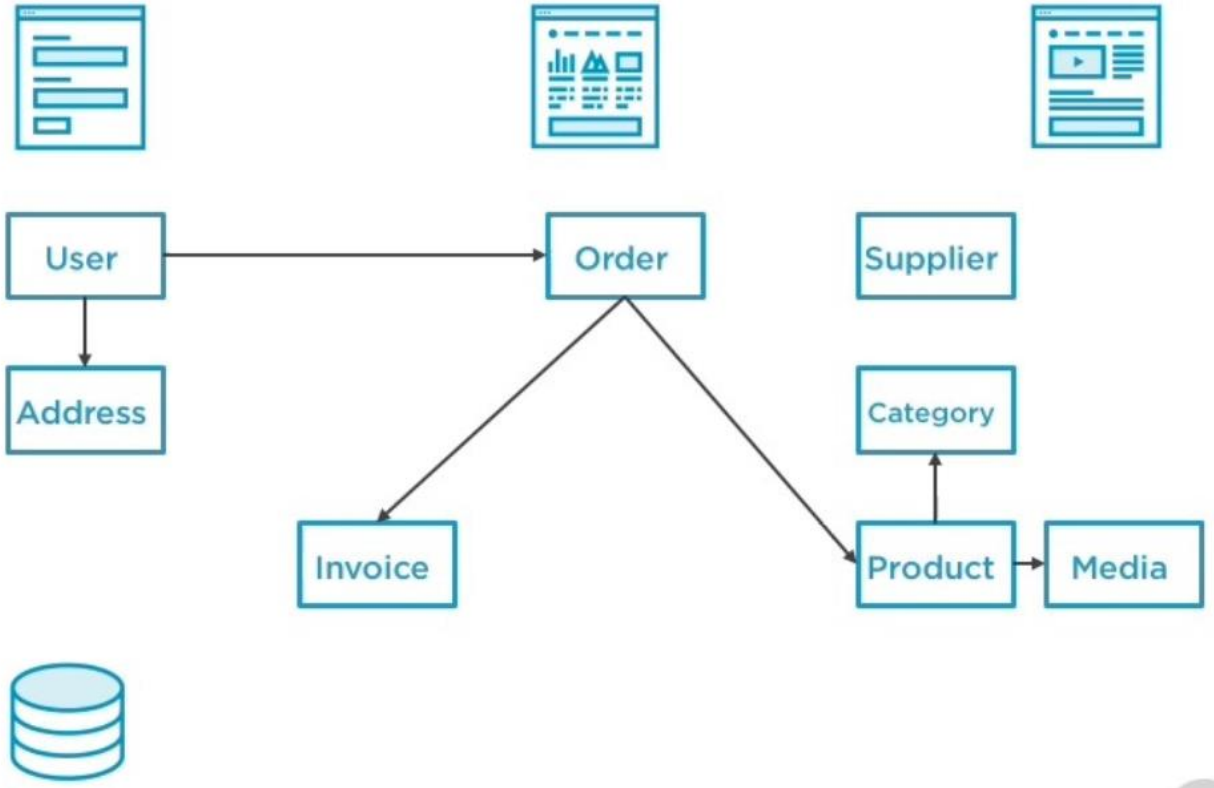
Ще приклад. Designing a Monolith (Відеокурси [4])

eCommerce webapp

Model

Single database

User interface



Deploying a Monolith

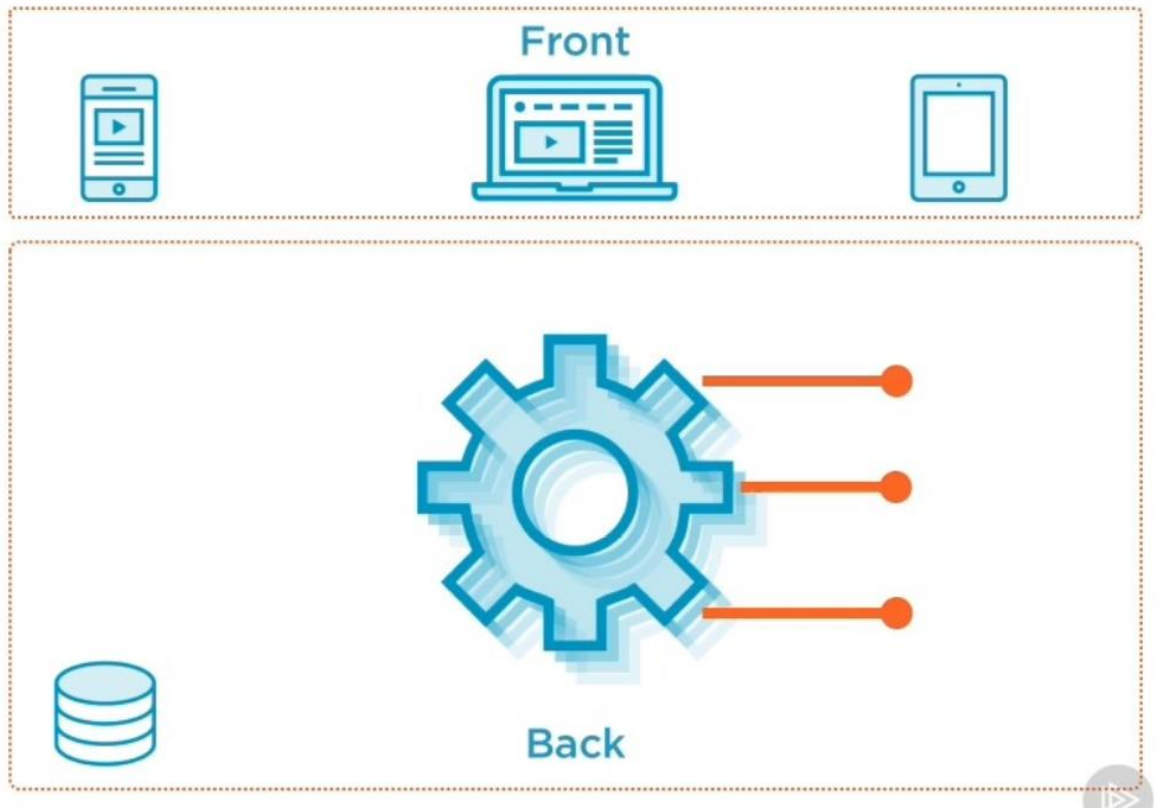
Single monolith

Single database

User interface

Expose APIs

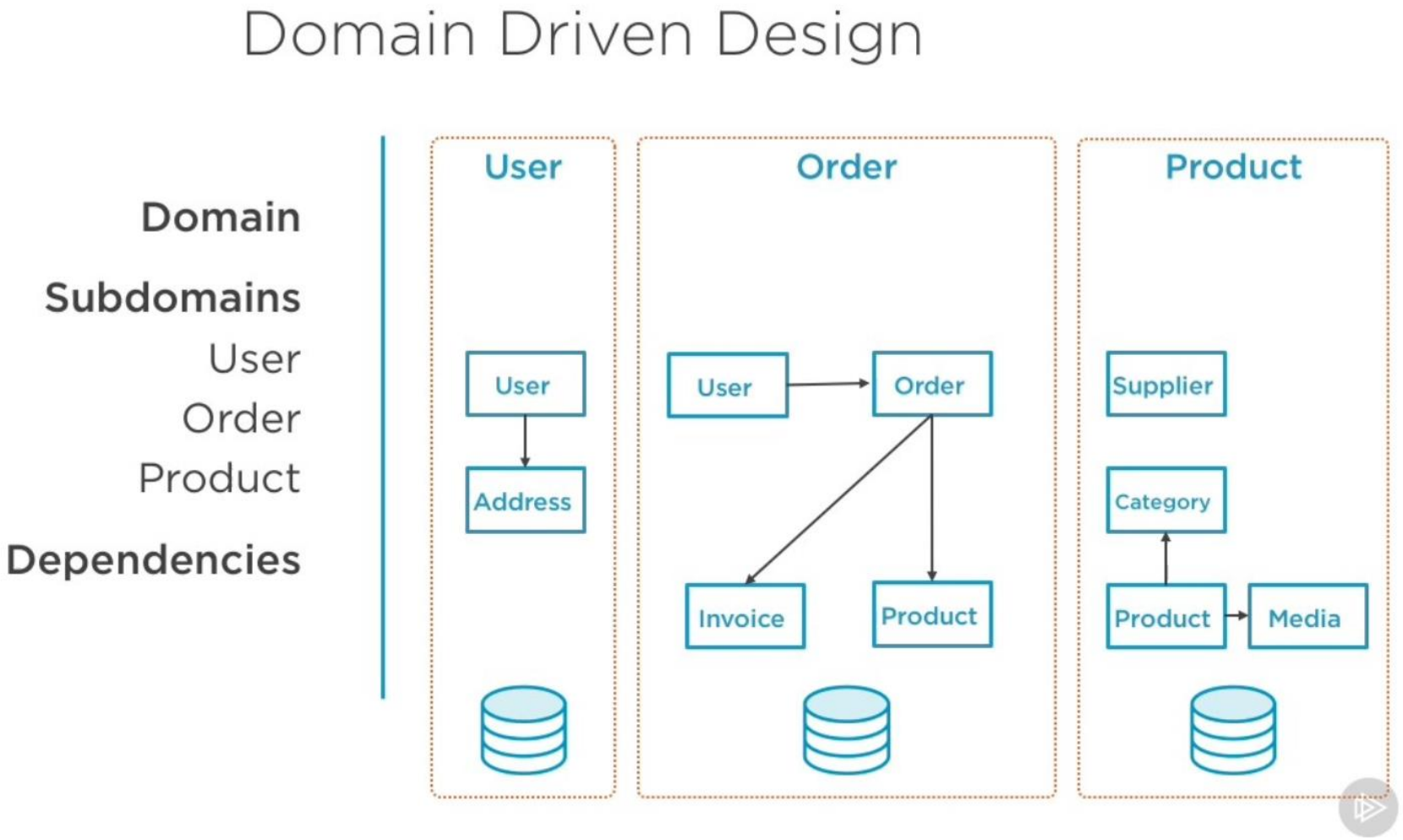
Multiple instances



Monolith

Pros	Cons
Simple to develop	New team members productivity
Simple to build	Growing teams
Simple to test	Code harder to understand
Simple to deploy	No emerging technologies
Simple to scale	Scale for bad reasons
	Overloaded container
	Huge database

Building Microservices



Building Microservices. Organization.

Team per subdomain

Right-sized teams

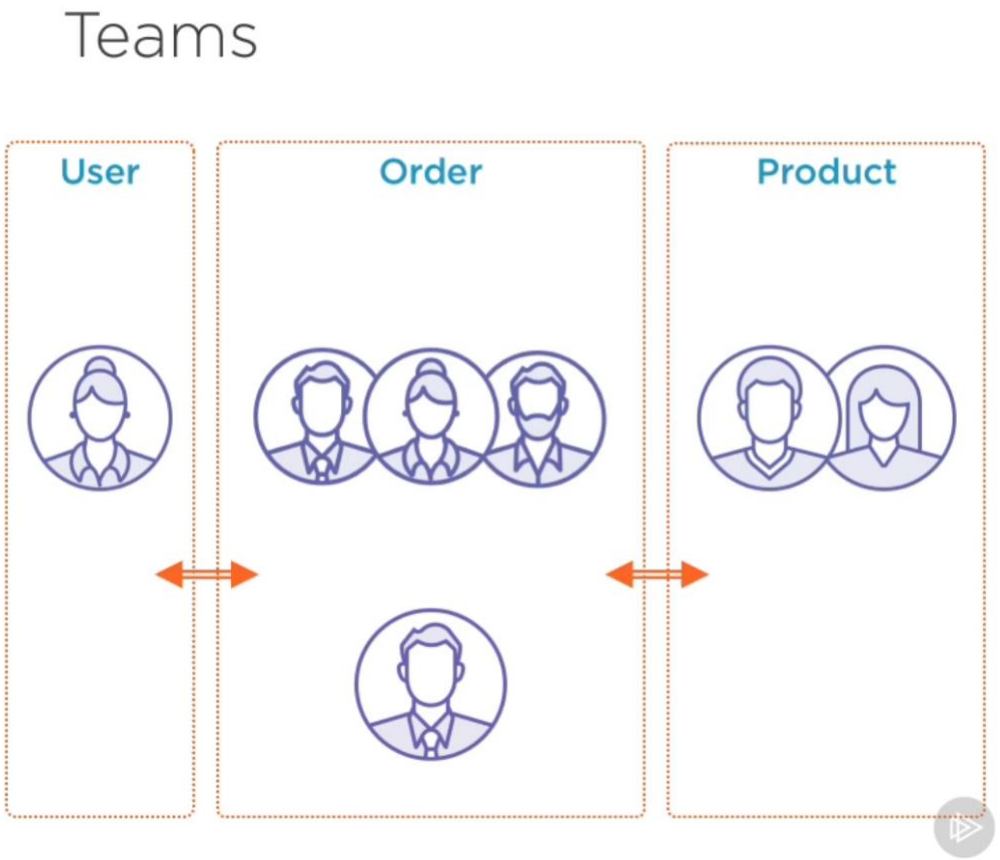
Independent

Responsible

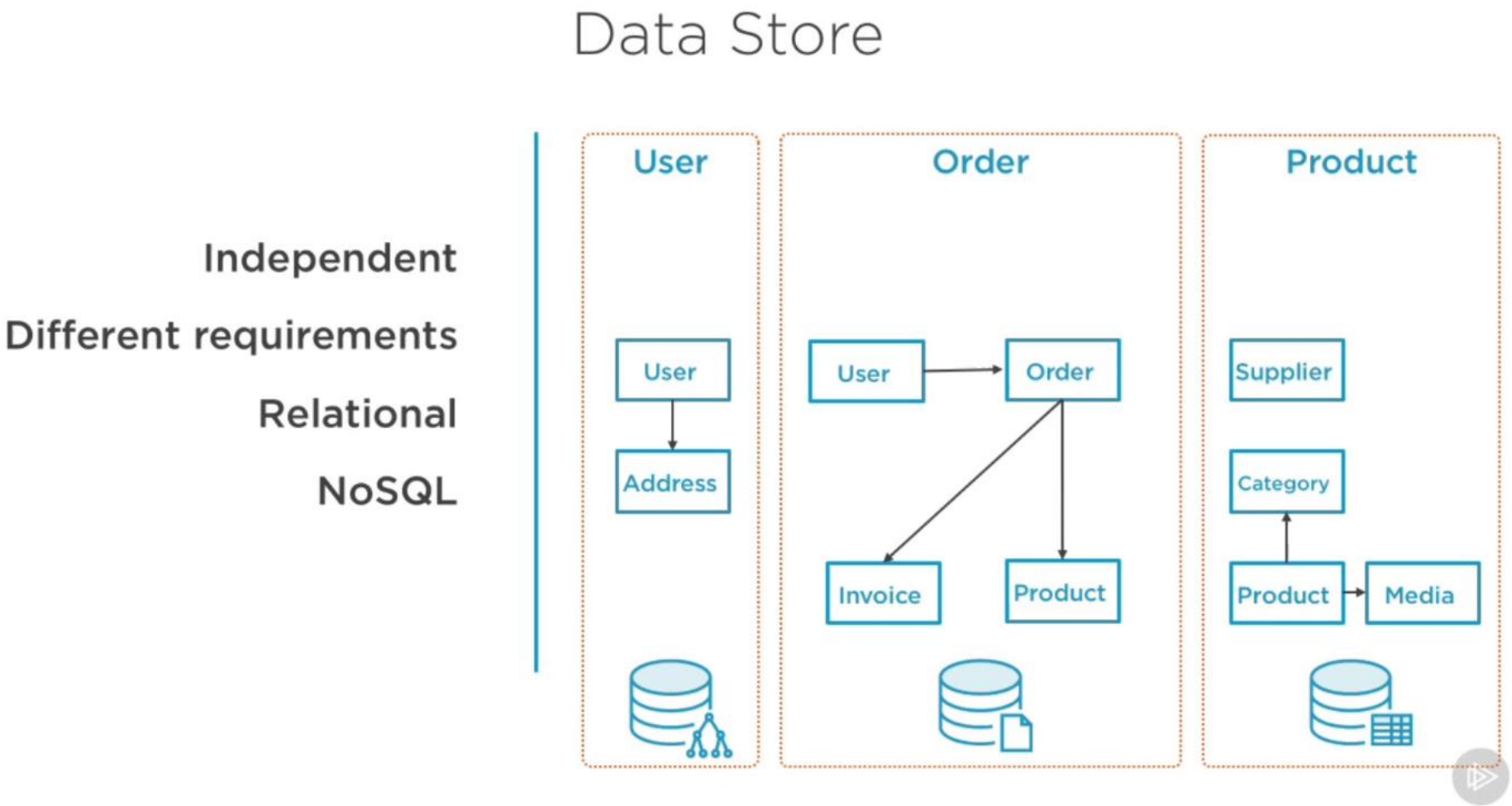
Agile and Devops

Communication

Management



Building Microservices.



Building Microservices.

Data Synchronization

No distributed transaction

Immediately consistent

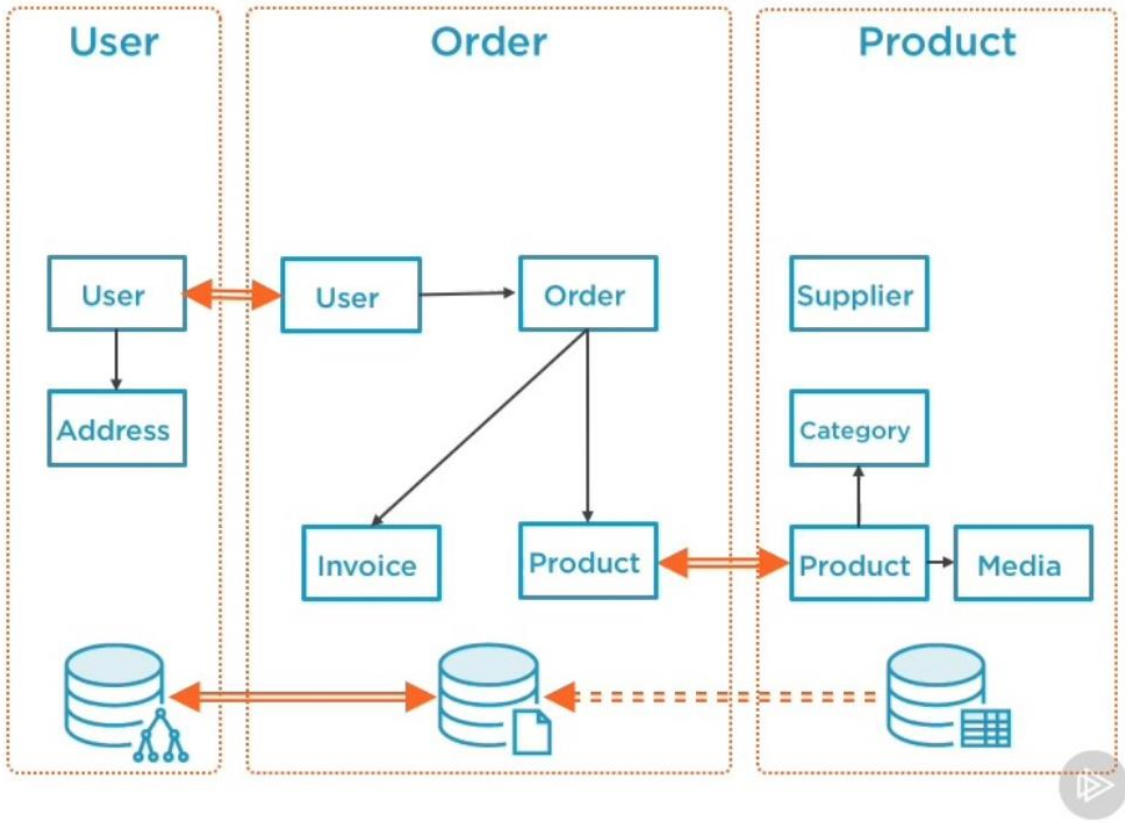
Eventual consistency

Capture data change

Event sourcing

Akka, Kafka, Rabbit MQ

Debezium



Building Microservices.

Independent teams

Own set of components

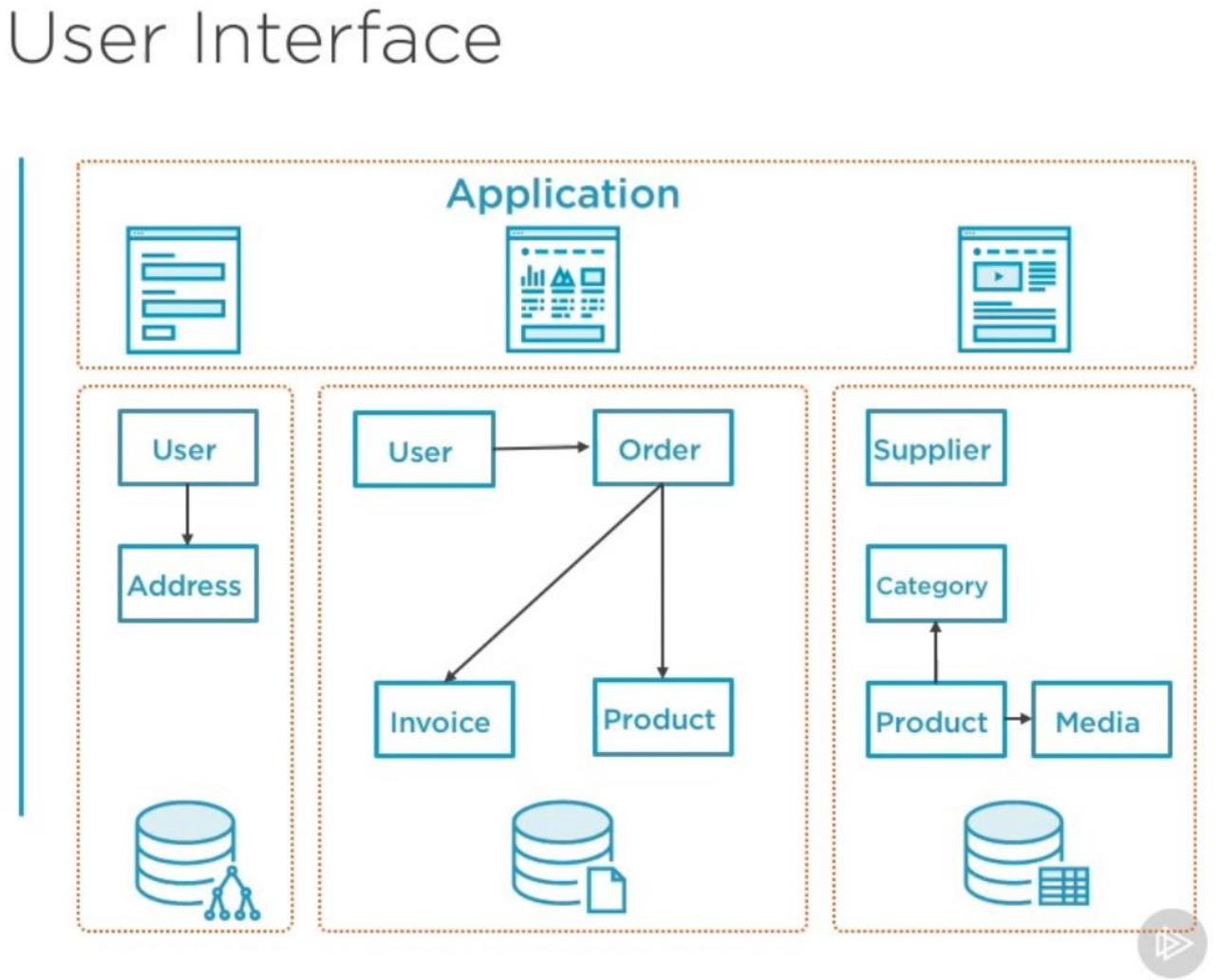
Unique UI

Single application

UI composition

Server side

Client side



Services

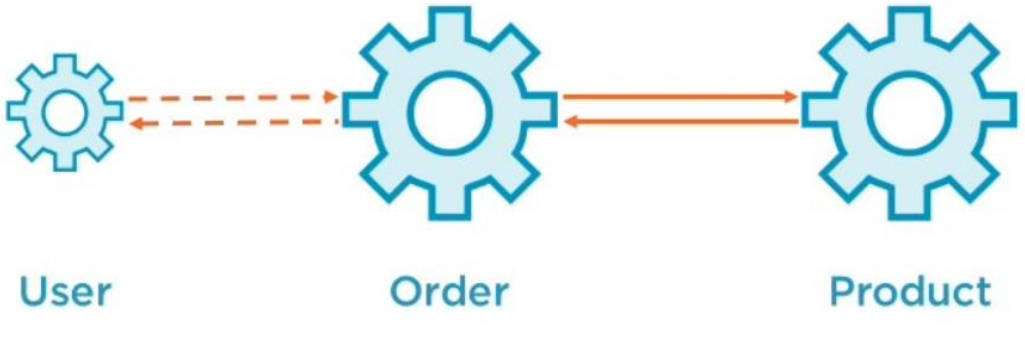
Remote Procedure Invocation

RPC

Request/reply

Synchronous

Asynchronous



Services

Messaging

Message or event

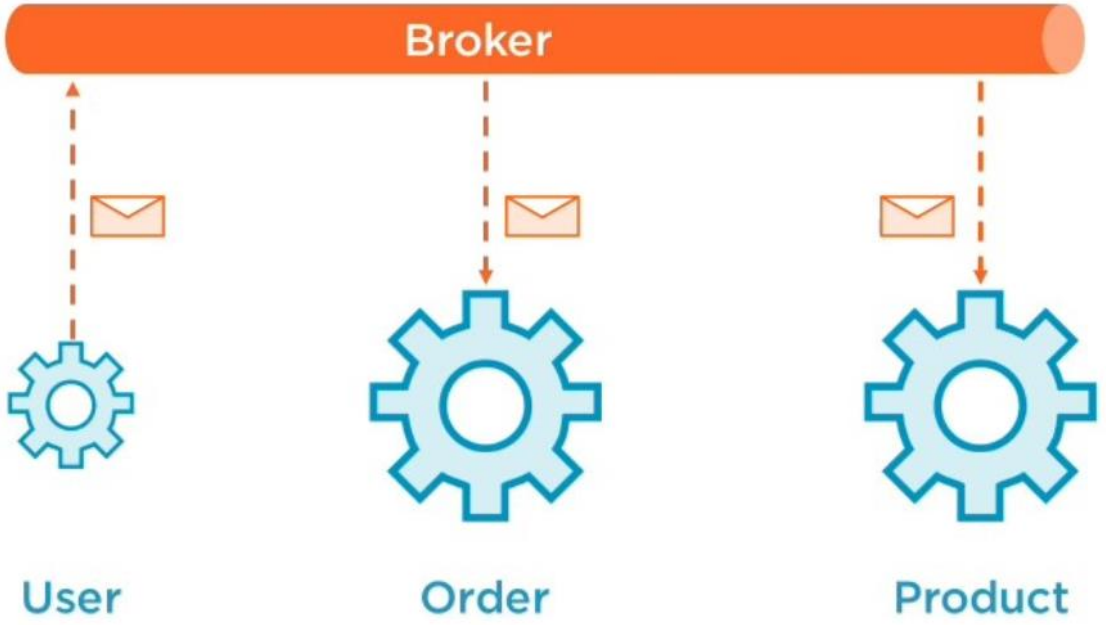
Broker or channel

Publish

Subscribe

Loosely couple

Kafka, RabbitMQ



Services

Protocol Format Exchange

Text

XML, JSON, YAML

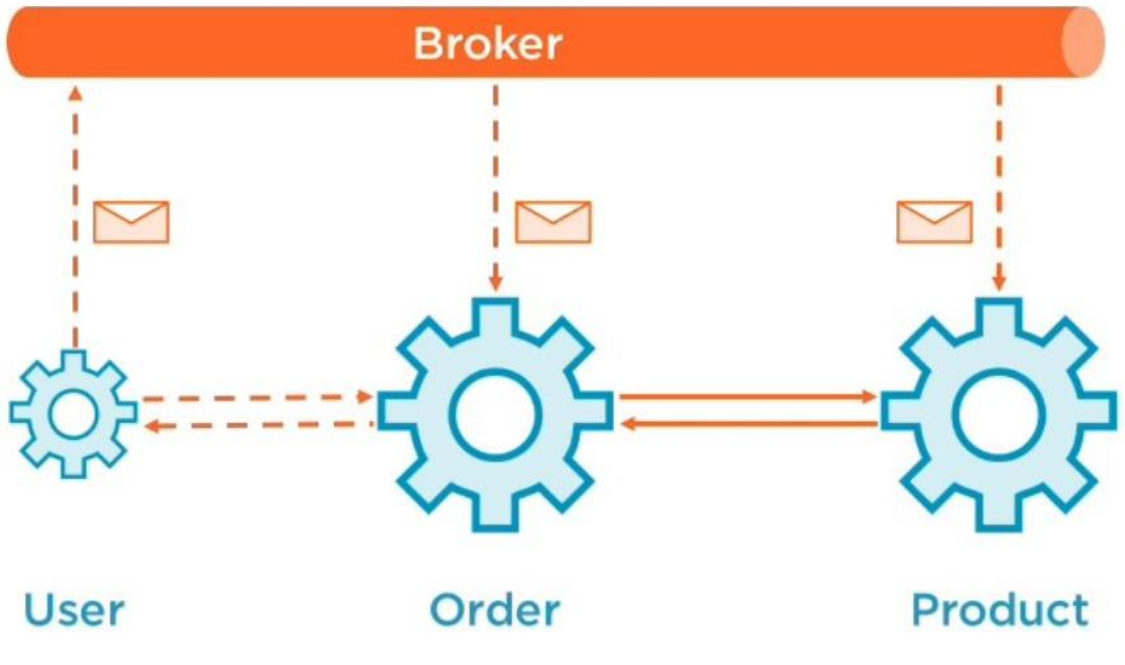
Human readable

Easy implement

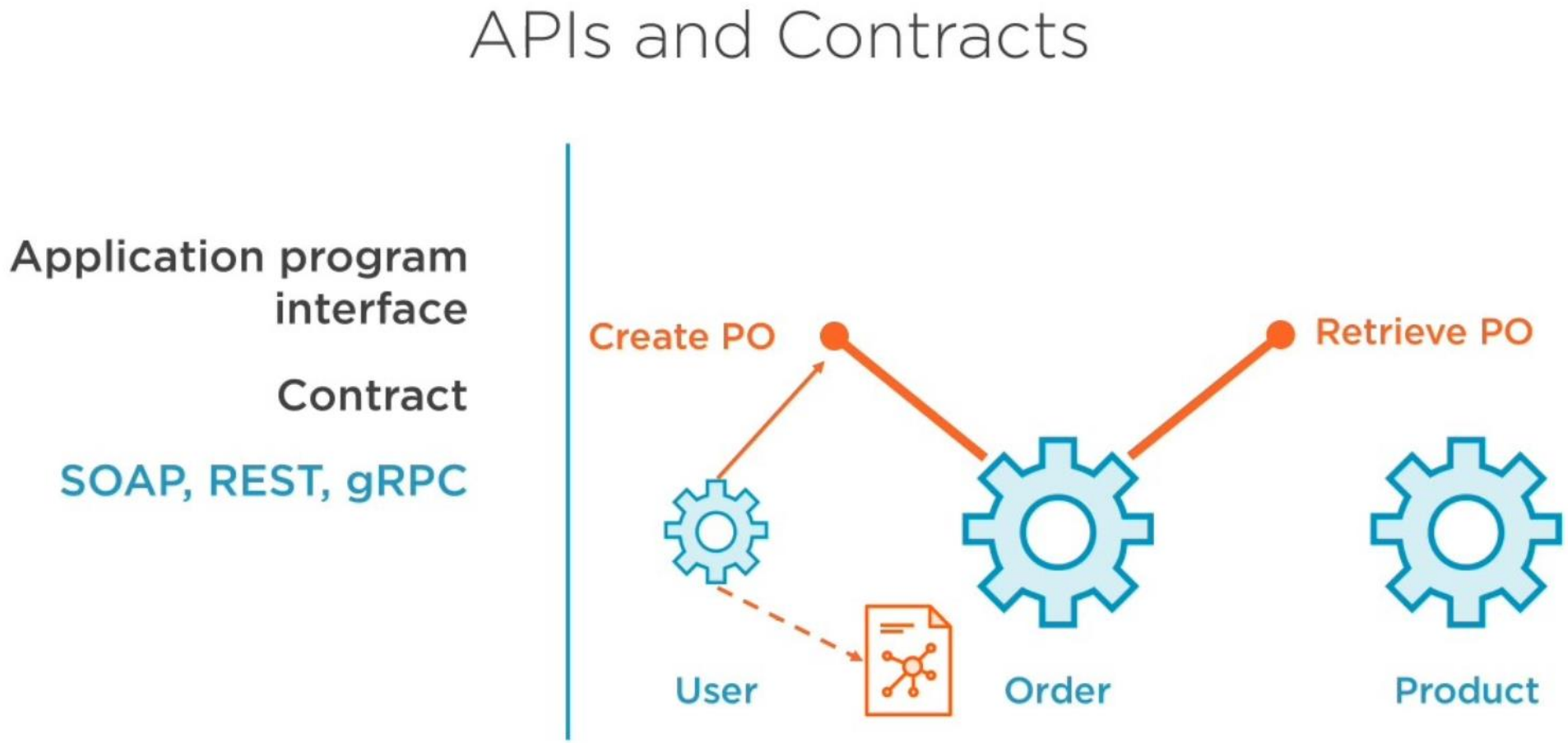
Binary

gRPC

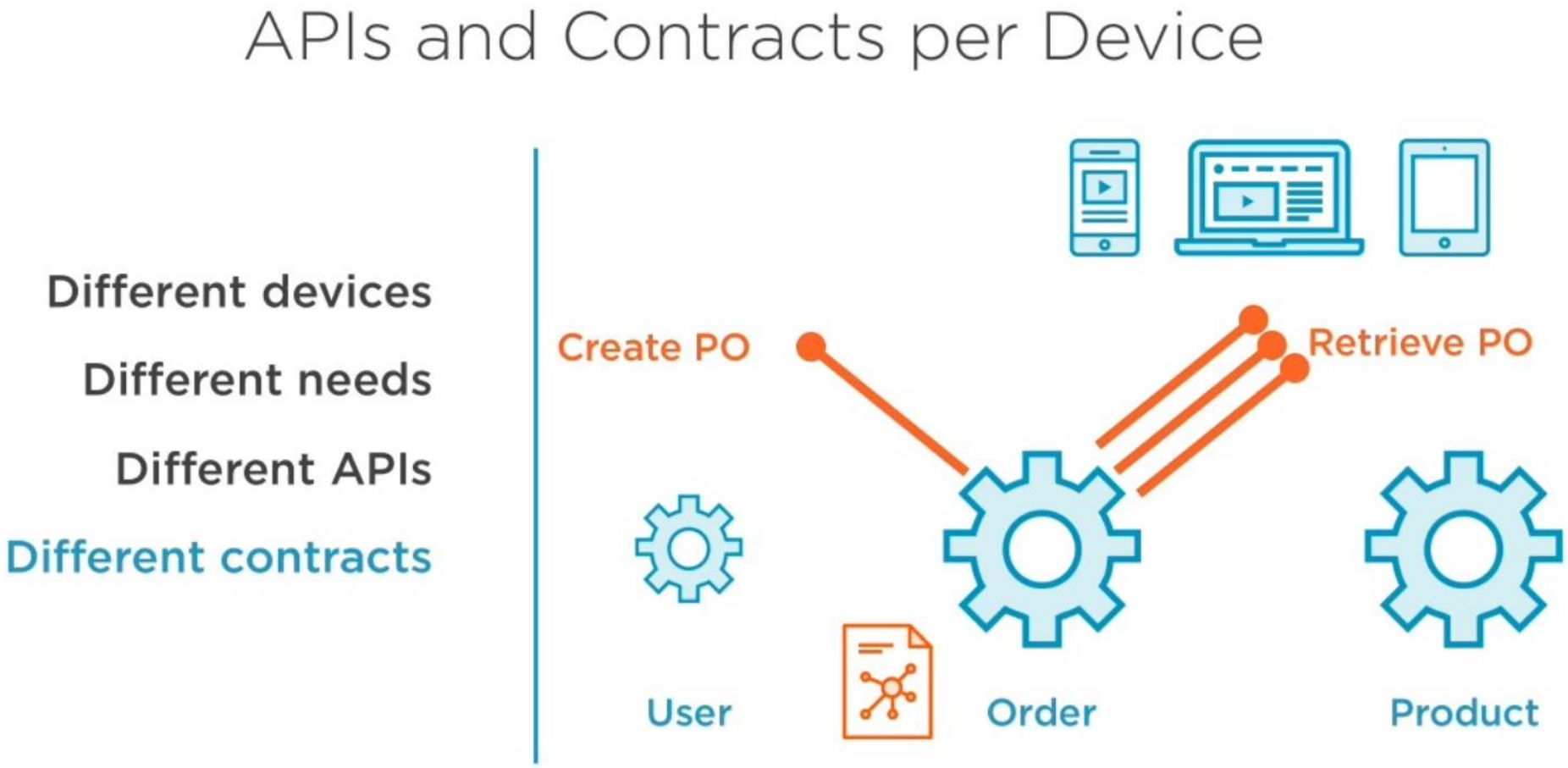
More compact



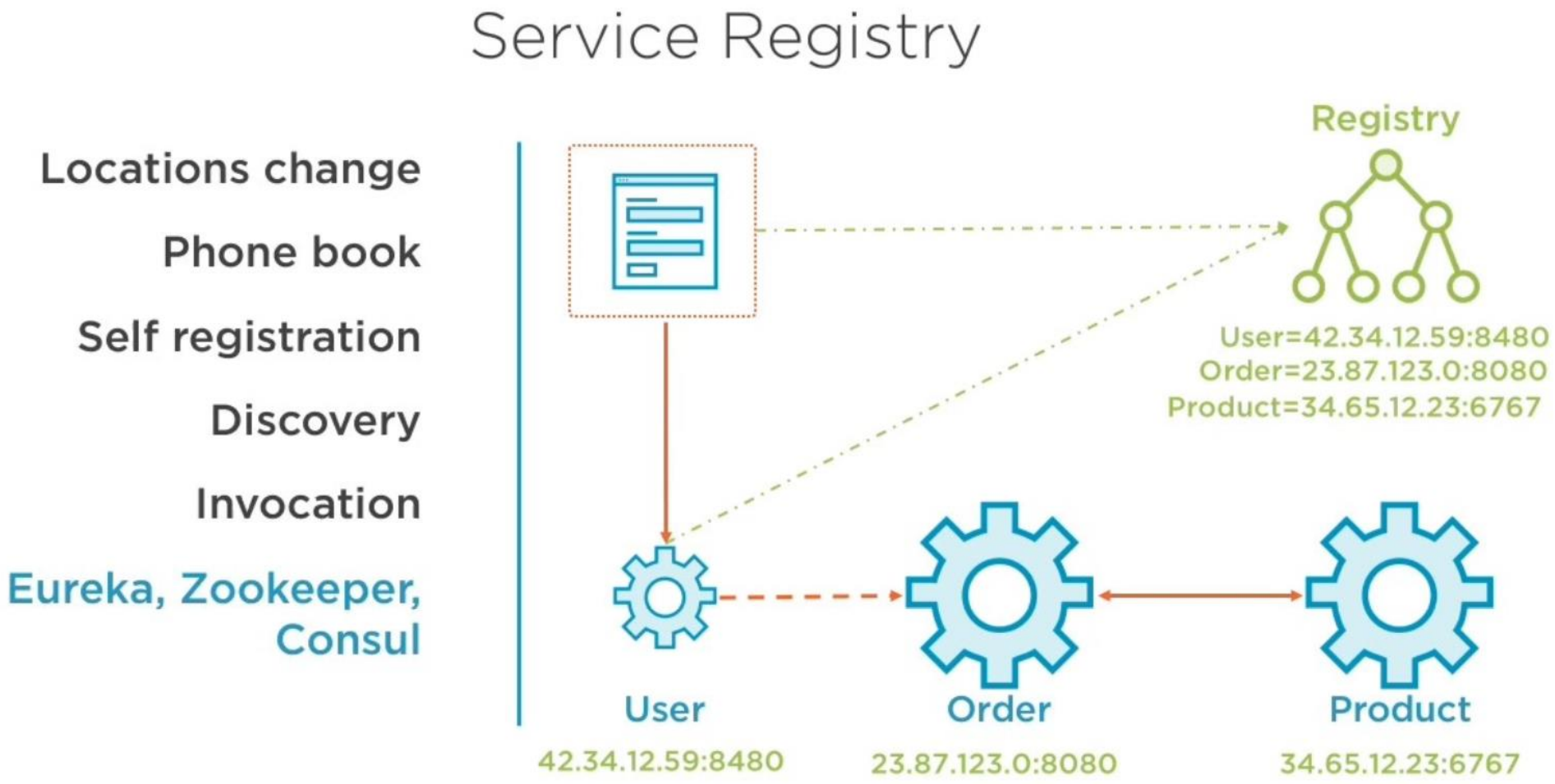
Services


APIs and Contracts
Application program interface
Contract
SOAP, REST, gRPC
Create PO
Retrieve PO
User
Order
Product

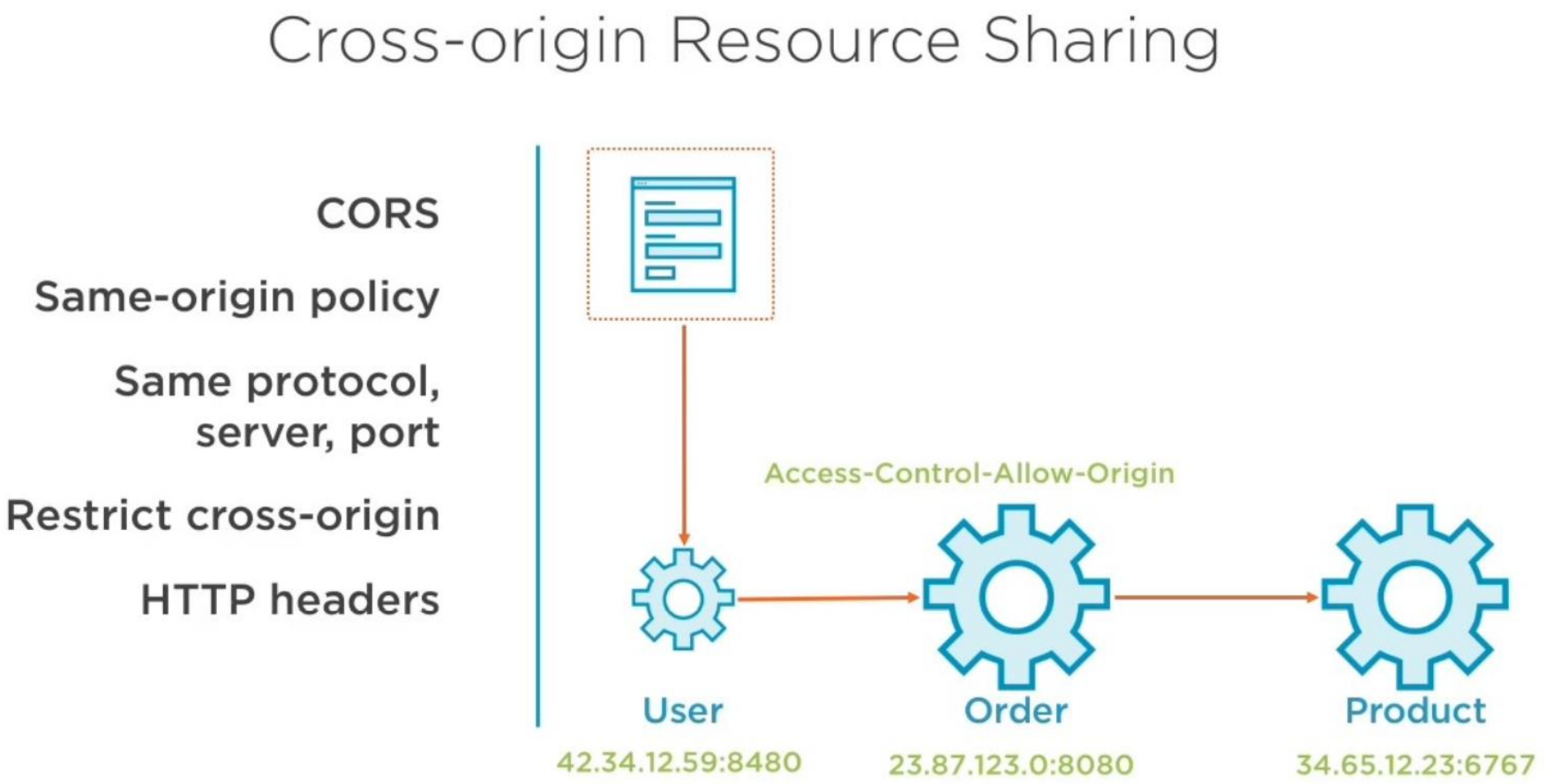
Services


APIs and Contracts per Device
Different devices
Different needs
Different APIs
Different contracts
Create PO
Retrieve PO
User
Order
Product

Distributed Services



Distributed Services



Distributed Services

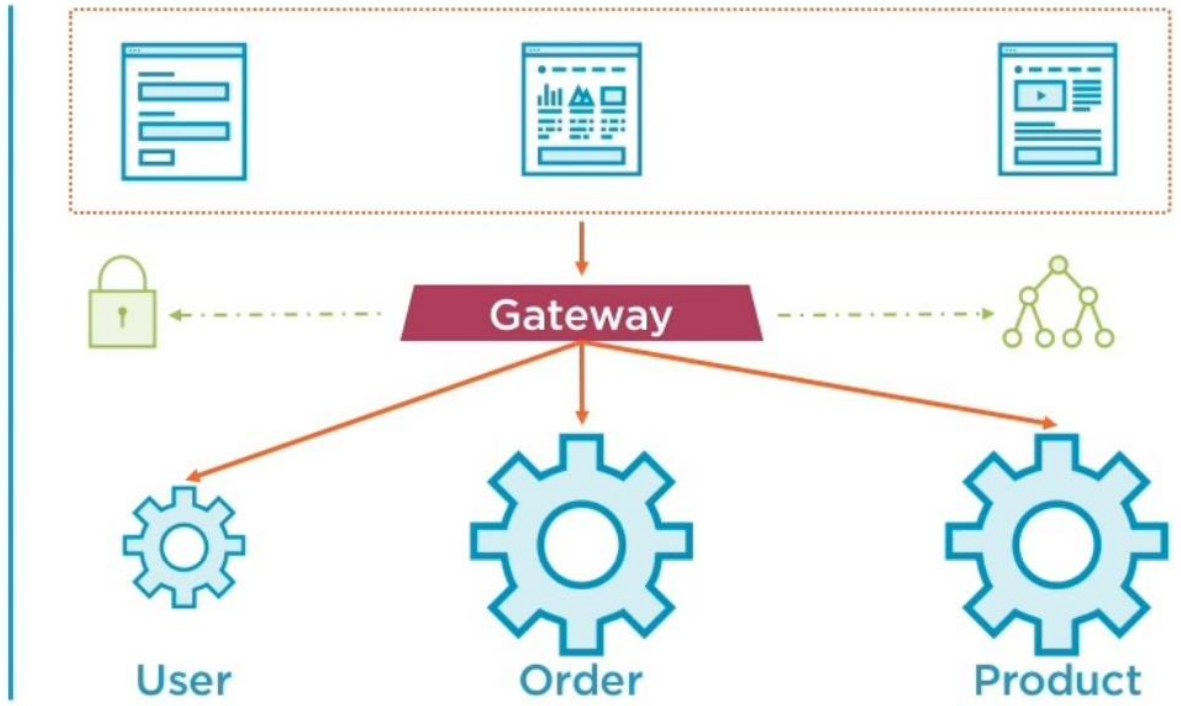
Gateway

Access individual services

Single entry point

Unified interface

Cross-cutting concerns



Security

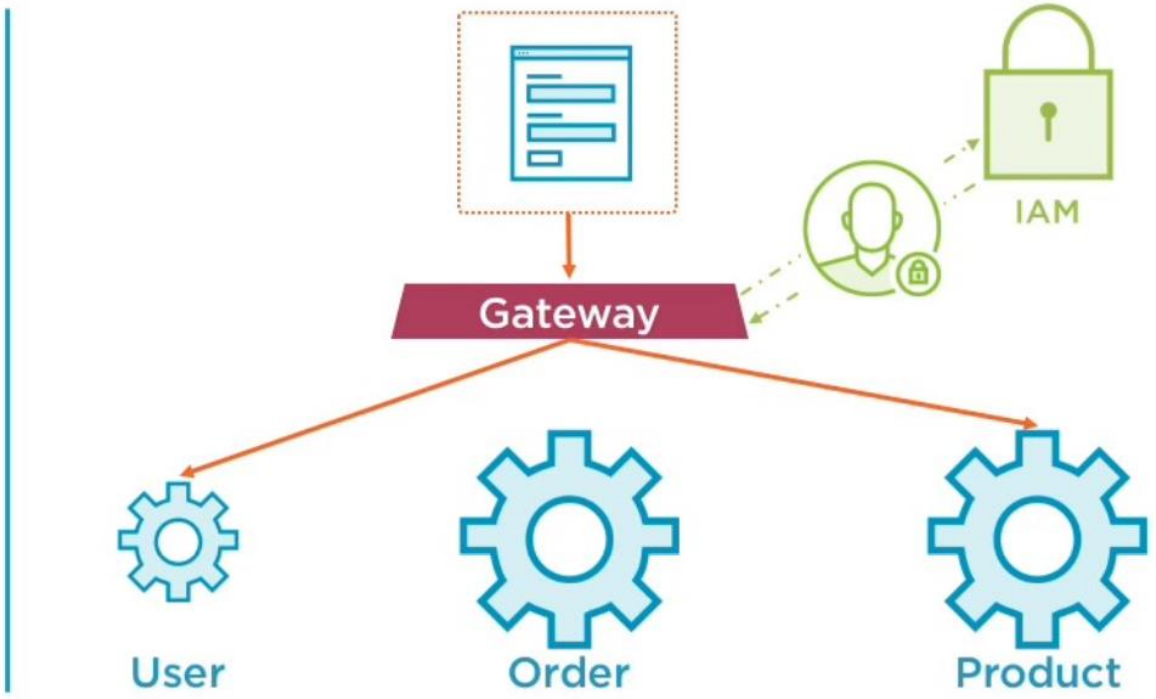
Authorization and Authentication

Identity and Access Management

Single Sign-on

Kerberos, OpenID, OAuth 2.0, SAML

Okta, Keycloak, Shiro



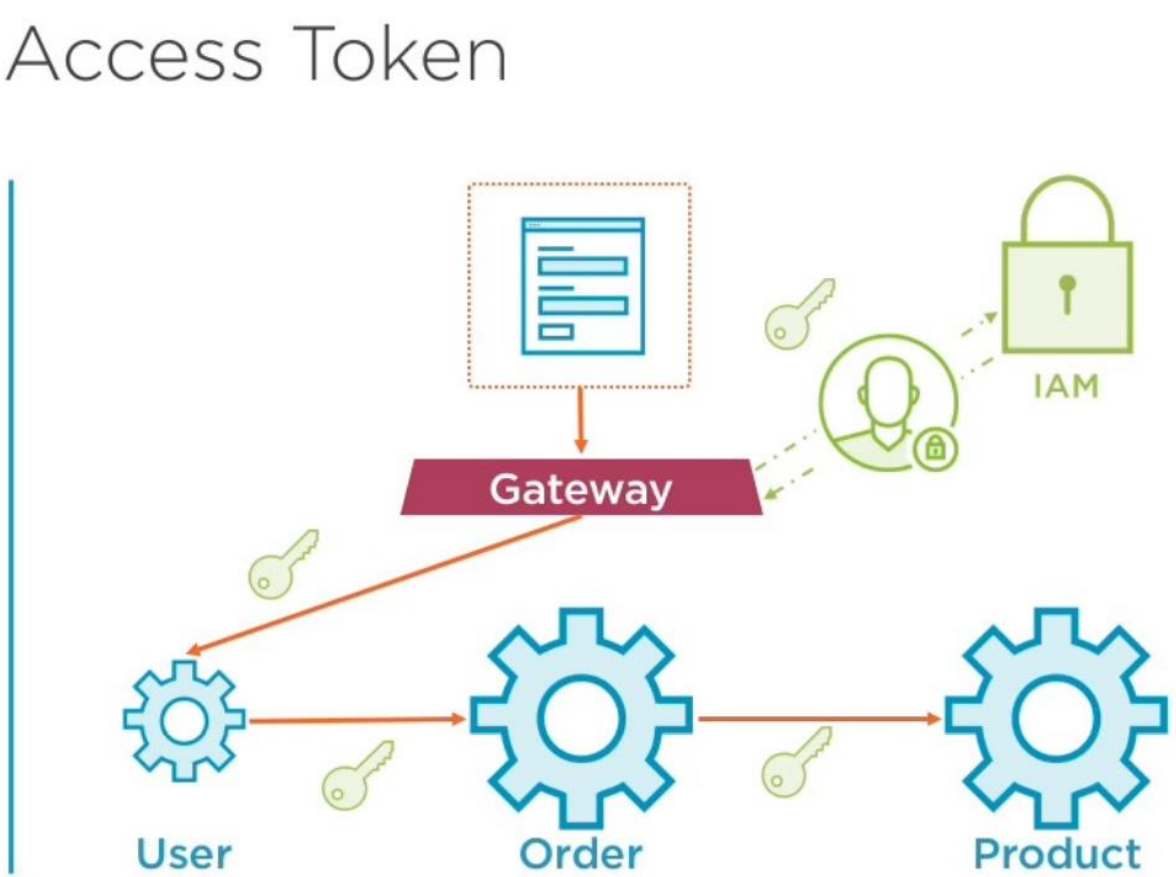
Security

Stores information about user

Exchanged between services

JSON Web Token

Cookie



Scalability and Availability

Scalability

Vertical

More CPU and RAM

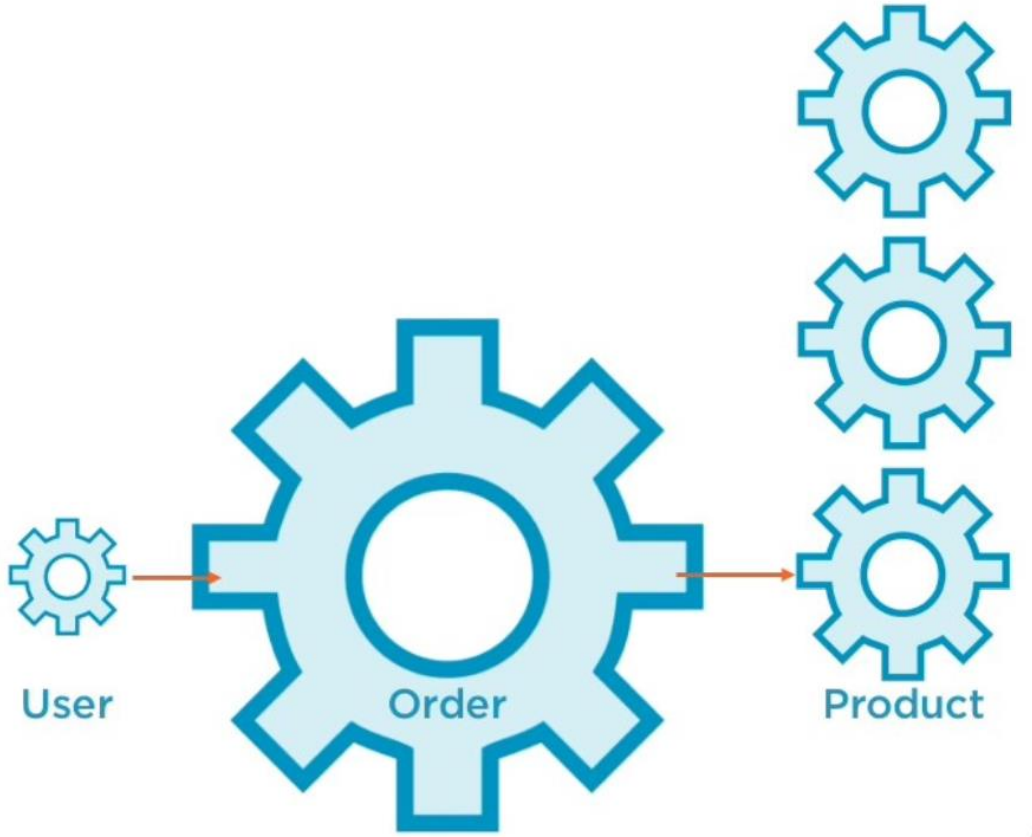
Horizontal

More machines

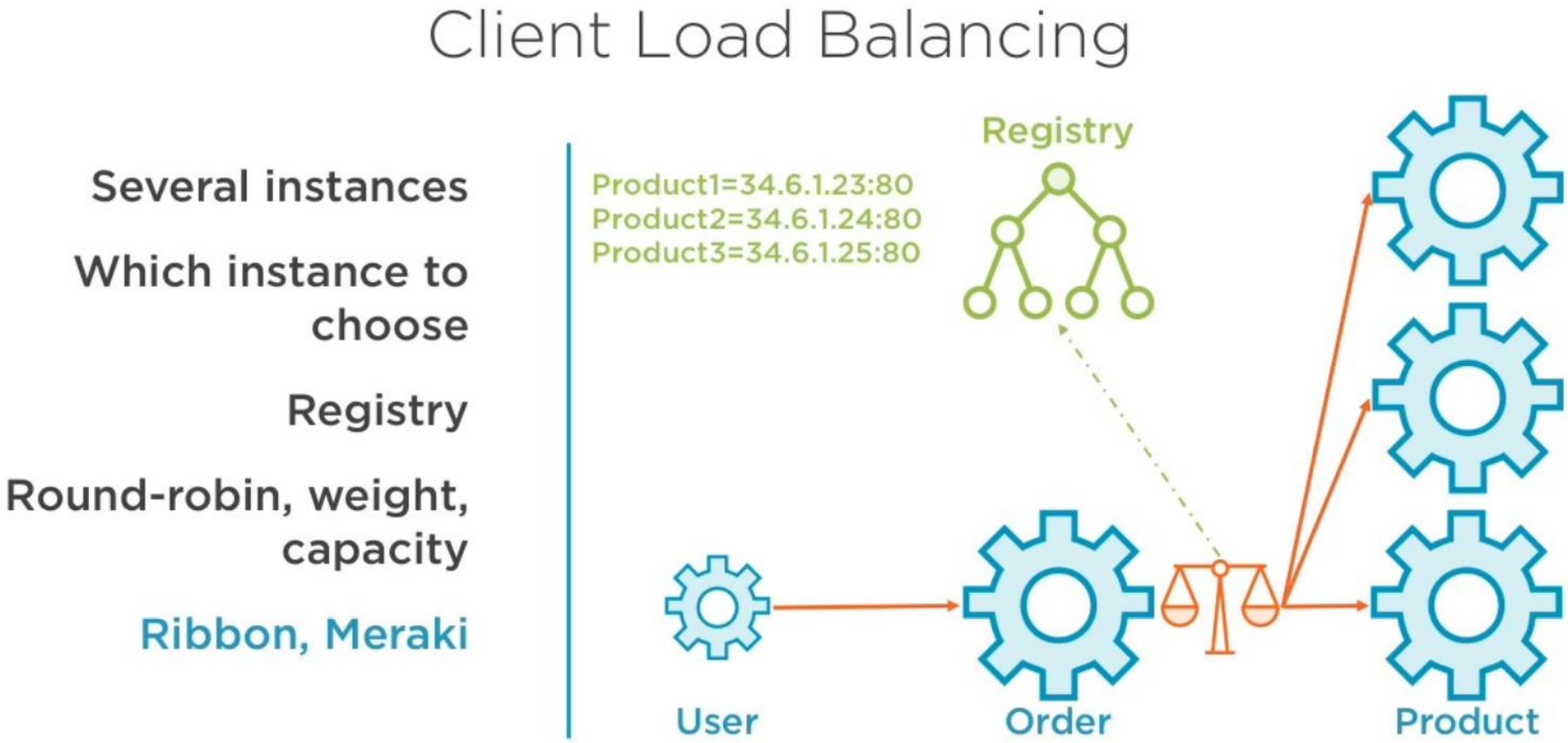
Service replication

Clustering

Scale up and down



Scalability and Availability



Scalability and Availability

Availability

Be operational

Single Point of Failure

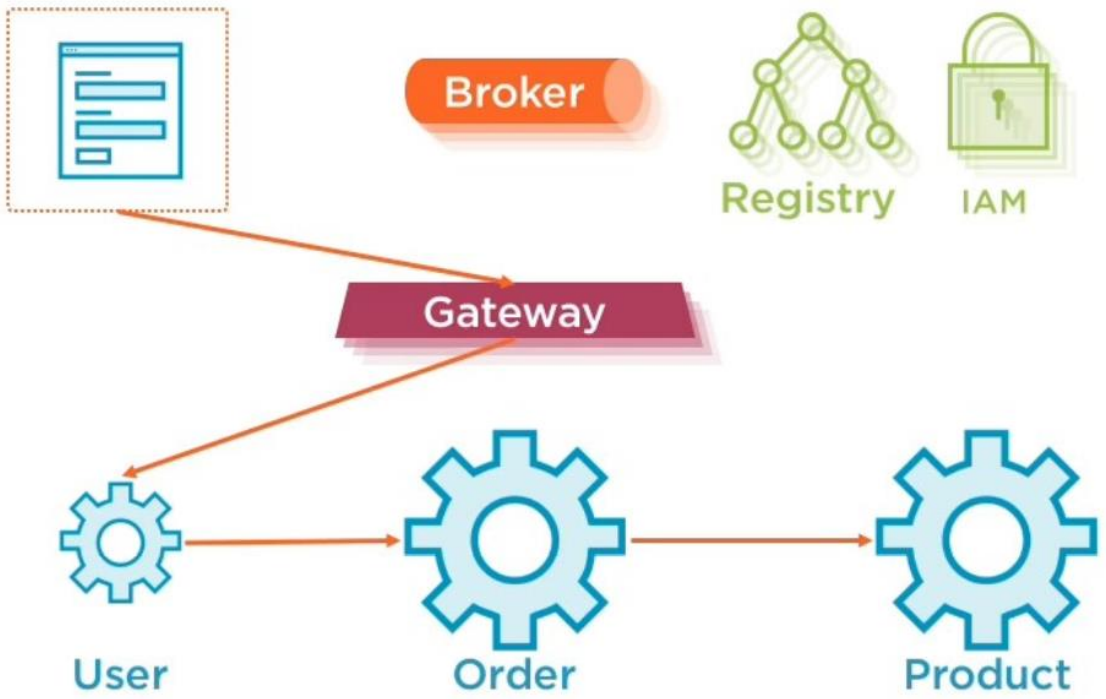
Gateway

Broker

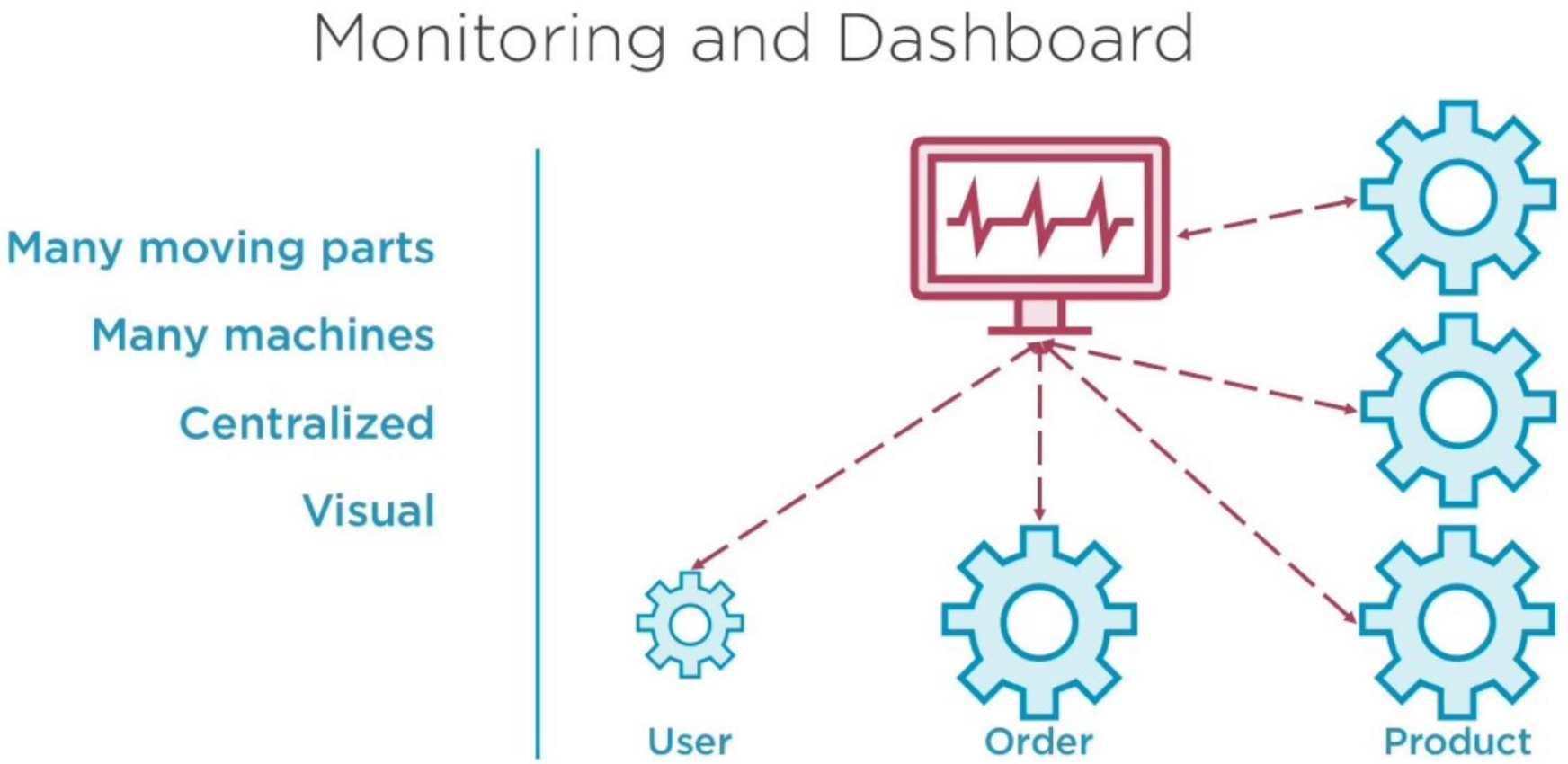
Registry

IAM

Multiply instances



Monitoring


Monitoring and Dashboard
Many moving parts
Many machines
Centralized
Visual
User
Order
Product

Monitoring

Service running

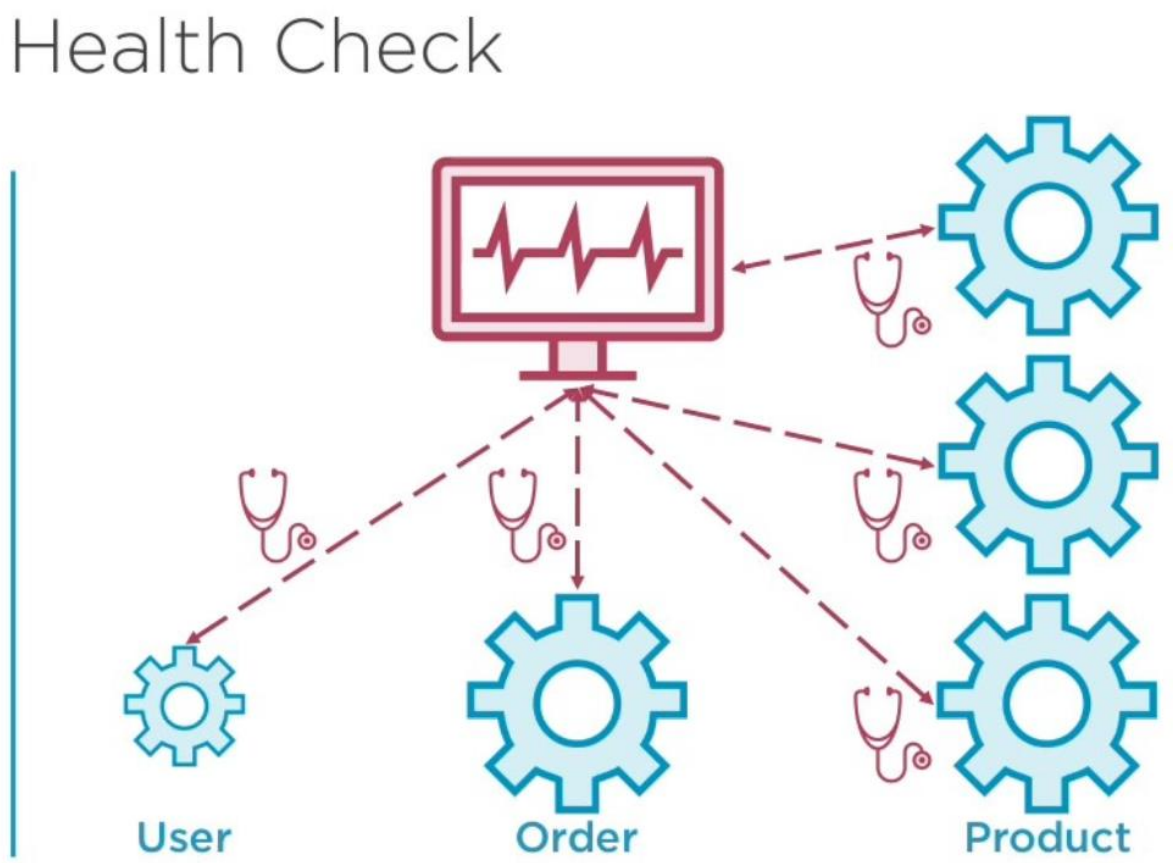
Incapable handling requests

Health check API

Database status

Host status

...



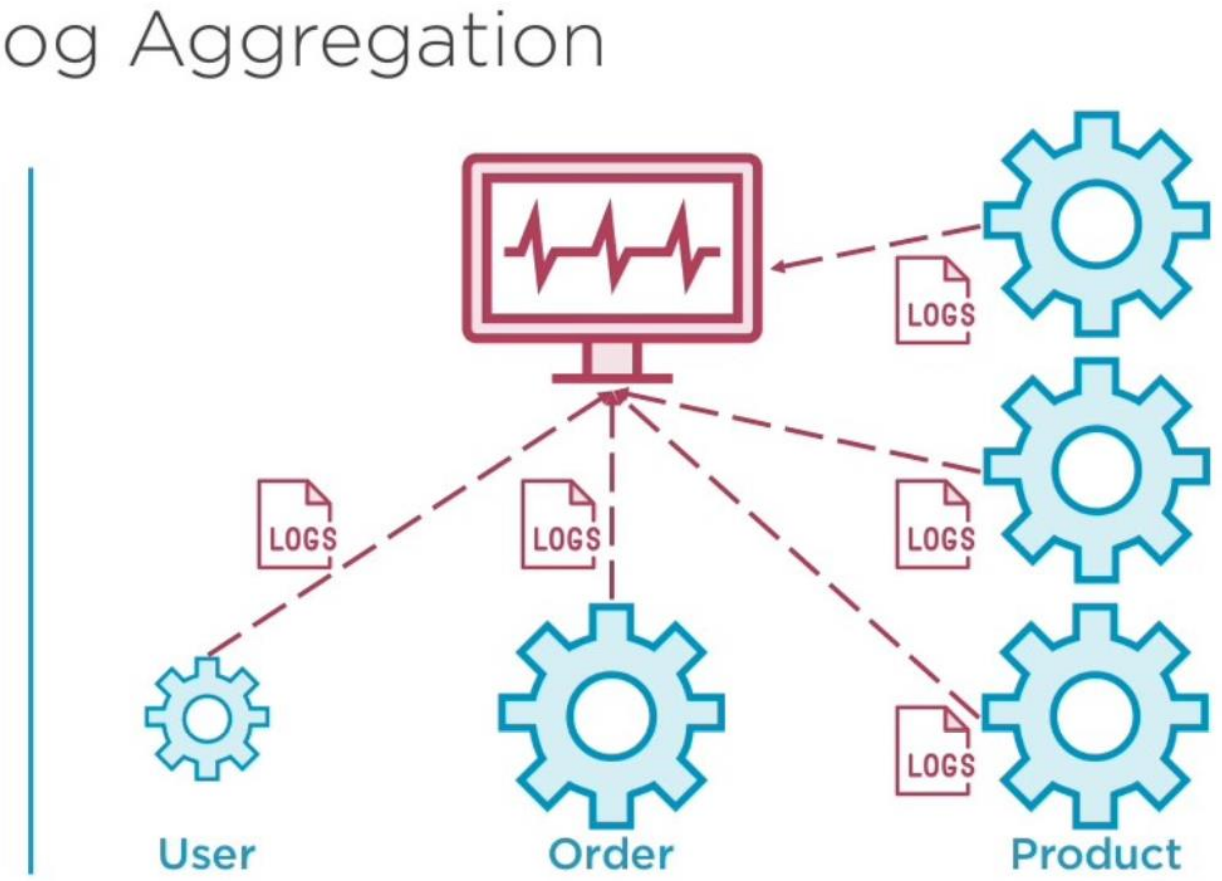
Monitoring

Understand behavior

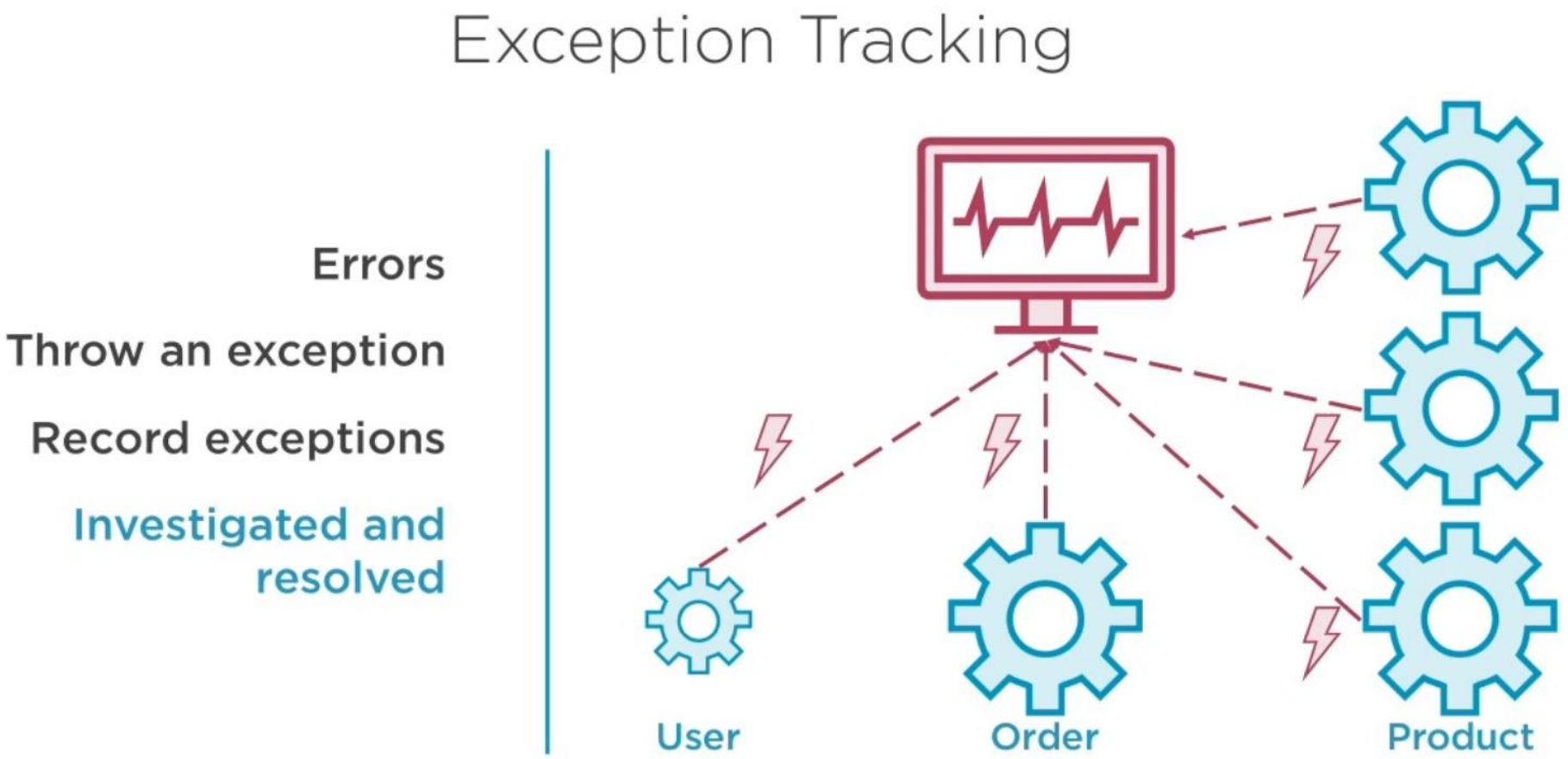
Write logs

Read each log file

Aggregate logs



Monitoring


Exception Tracking
Errors
Throw an exception
Record exceptions
Investigated and resolved
User
Order
Product

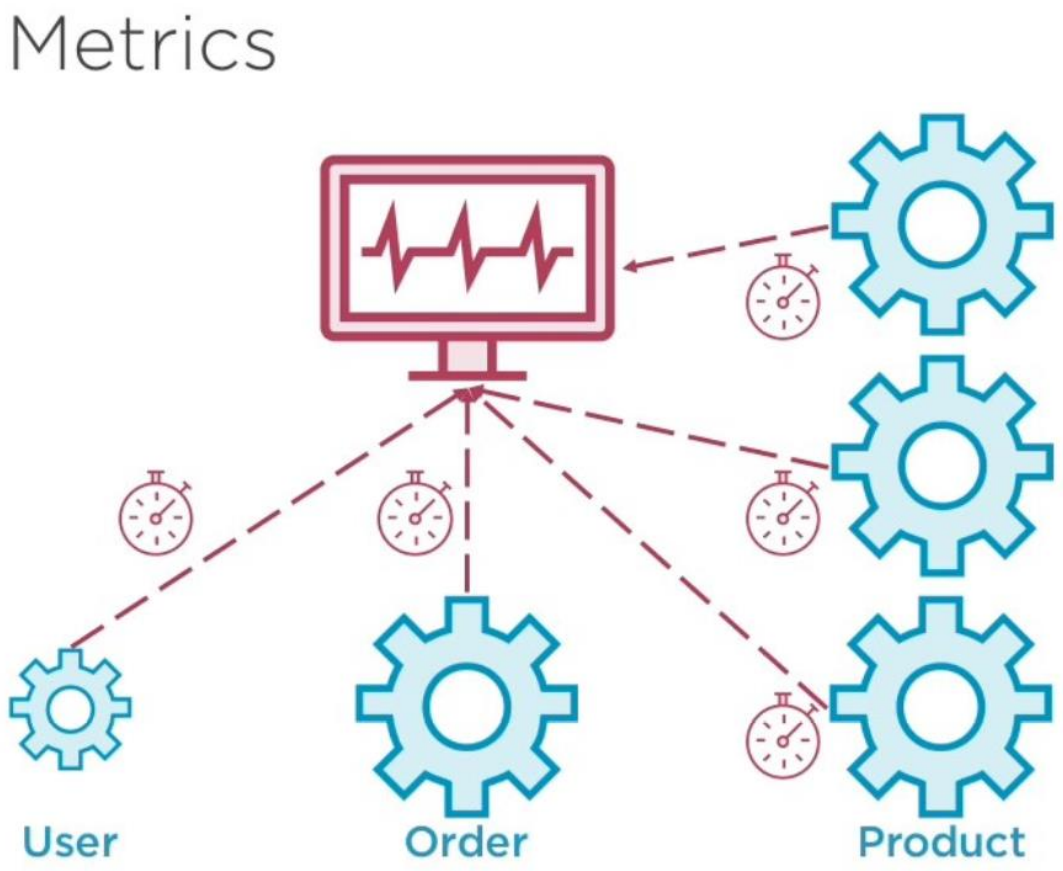
Monitoring

System slowing down

Performance issues

Gather statistics

Aggregate metrics



Monitoring

Behavior of users

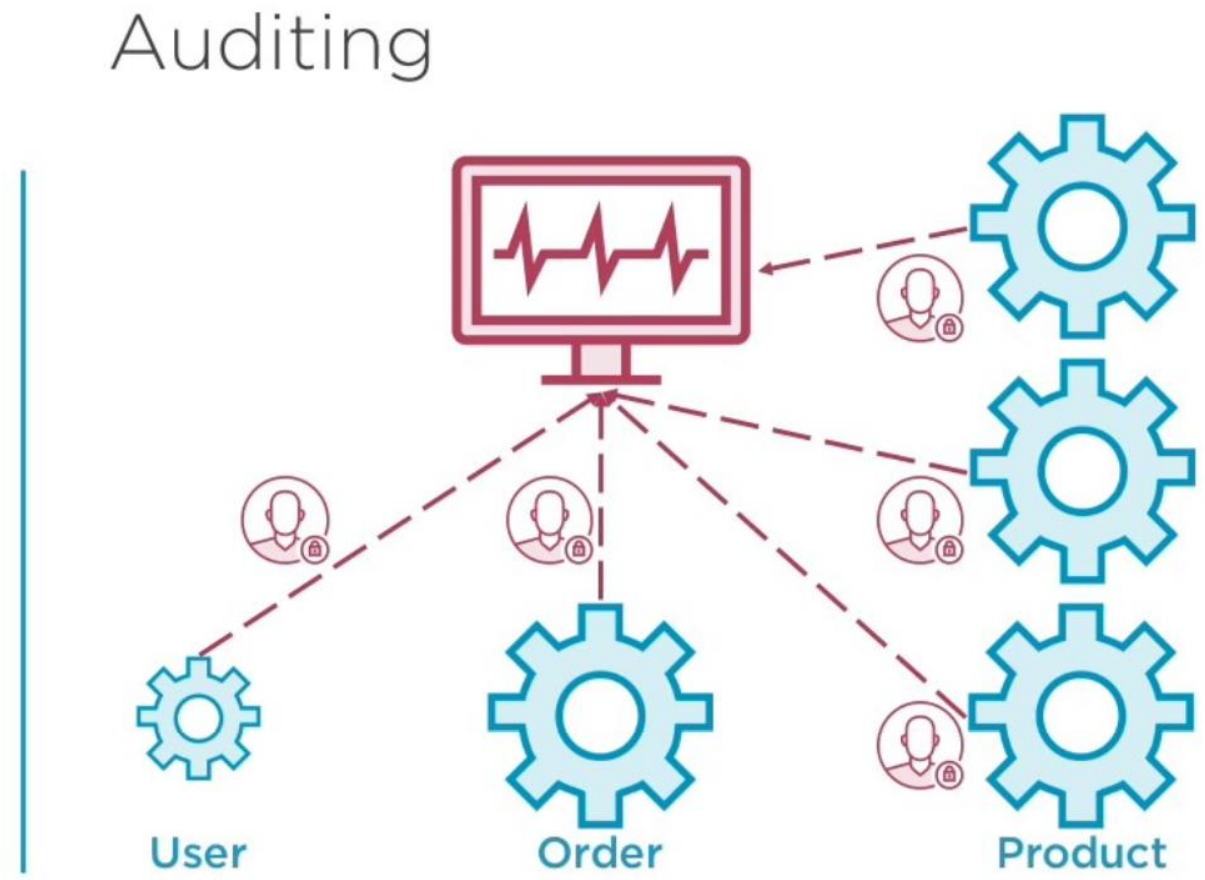
Login

Logout

Visited pages

Browsed products

Record user activity



Monitoring

Third-party access

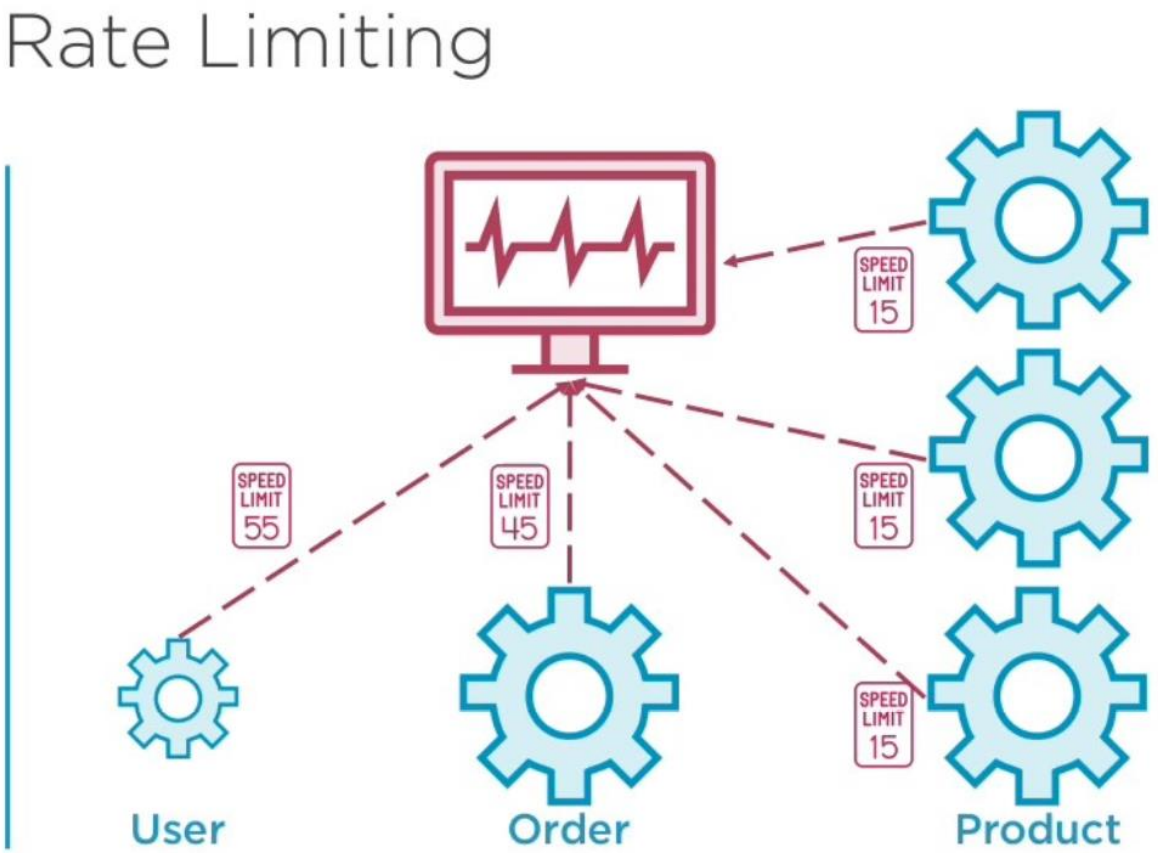
Control API usage

Defend DoS attacks

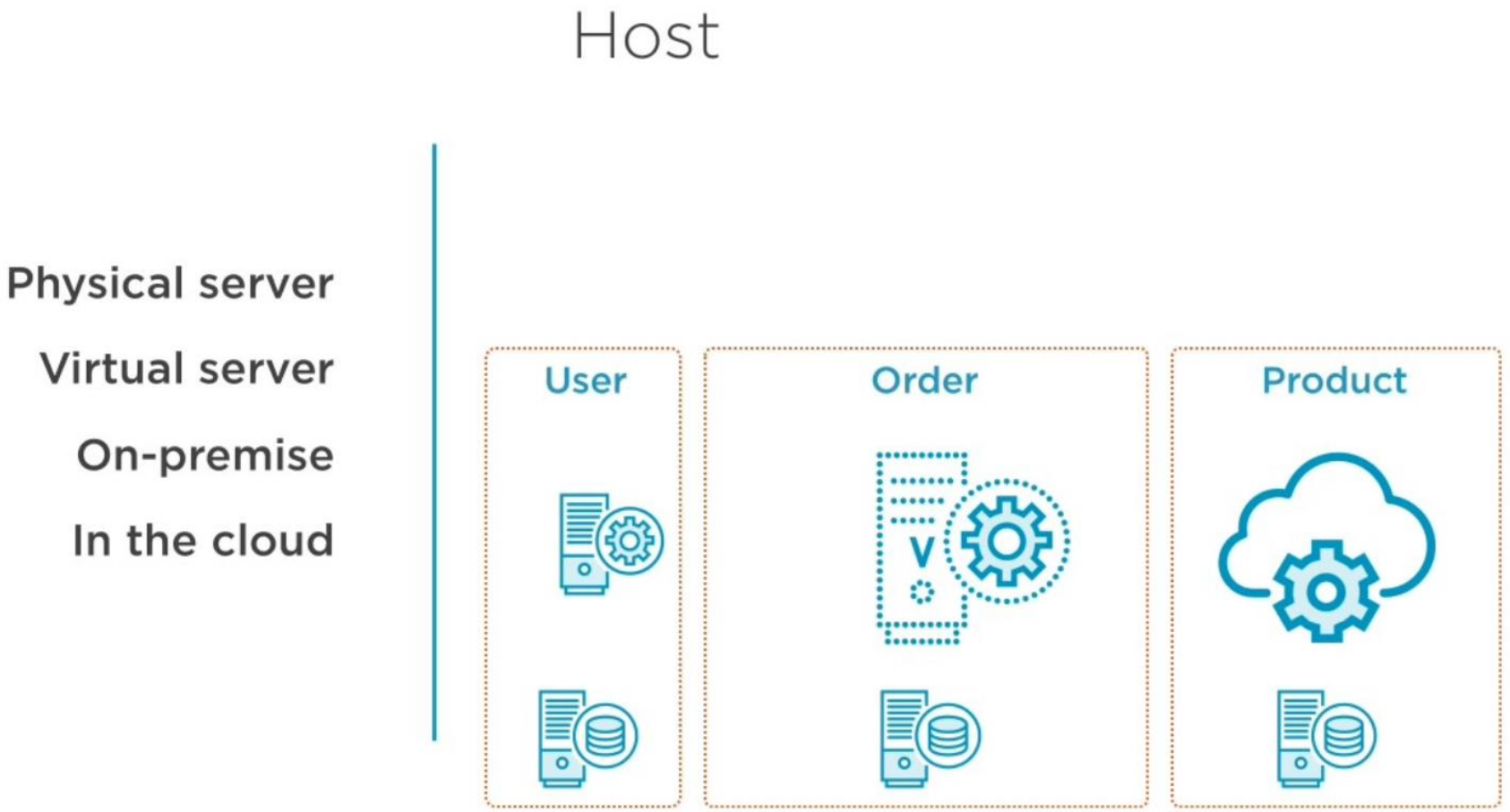
Limit traffic

In a period of time

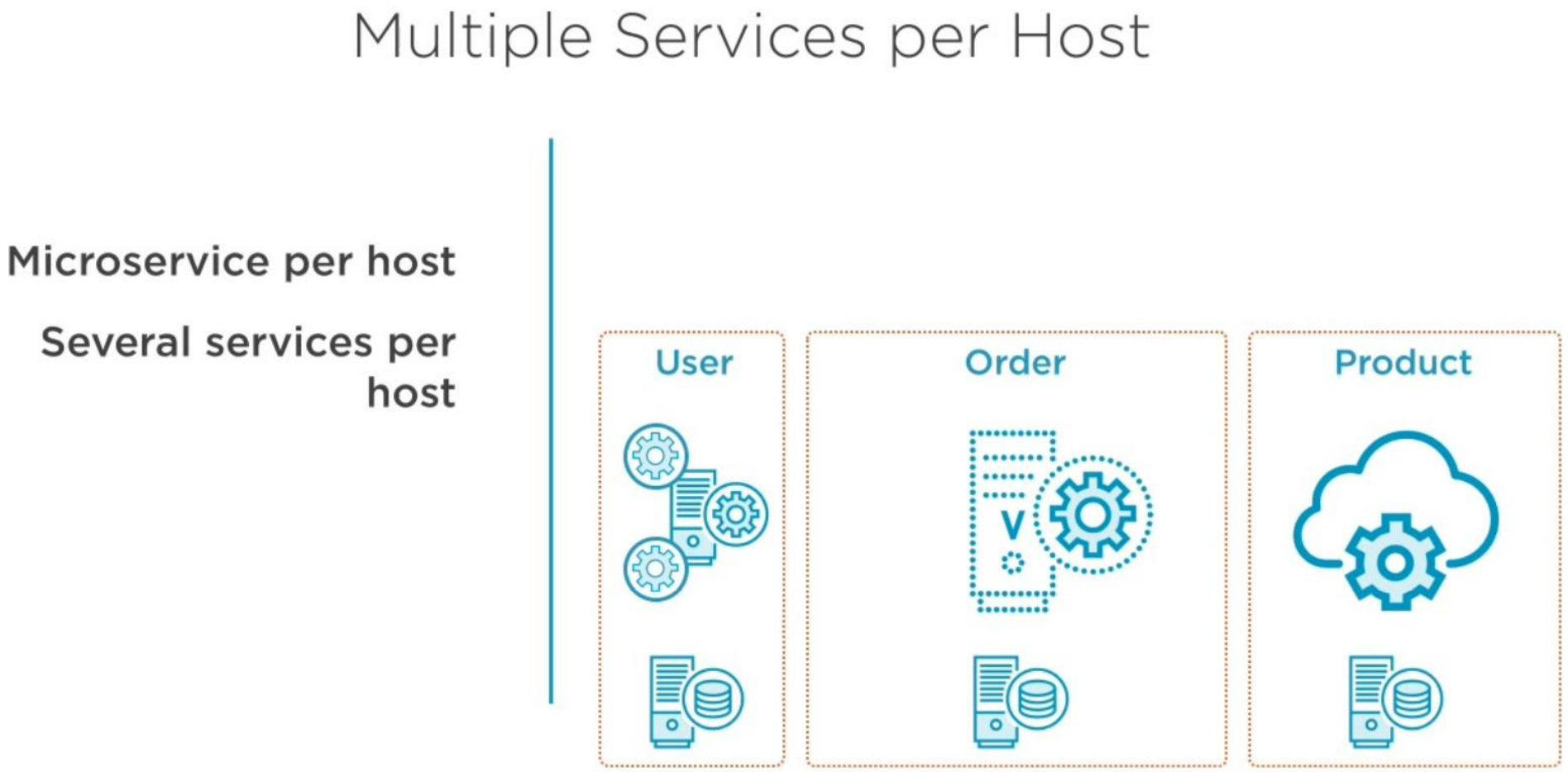
Monetize our APIs



Deployment


Host
Physical server
Virtual server
On-premise
In the cloud
User
Order
Product

Deployment

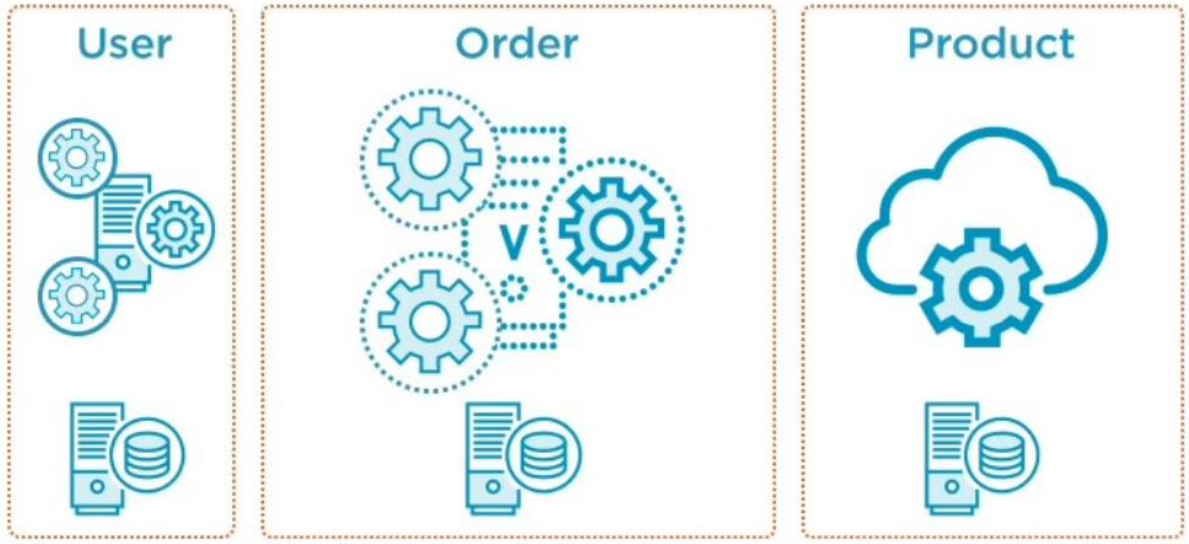

Multiple Services per Host
Microservice per host
Several services per host
User
Order
Product

Deployment

Containers

Packaging microservice

With dependencies



Deployment

Containers

Packaging microservice

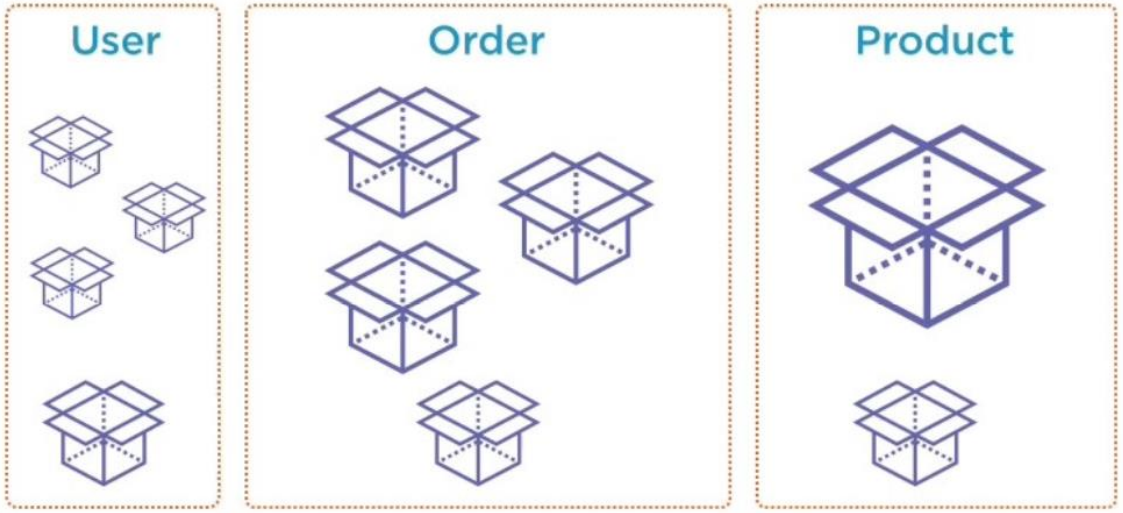
With dependencies

Container image

Easy to move from environment

Scale up and down

Docker, rkt



Deployment

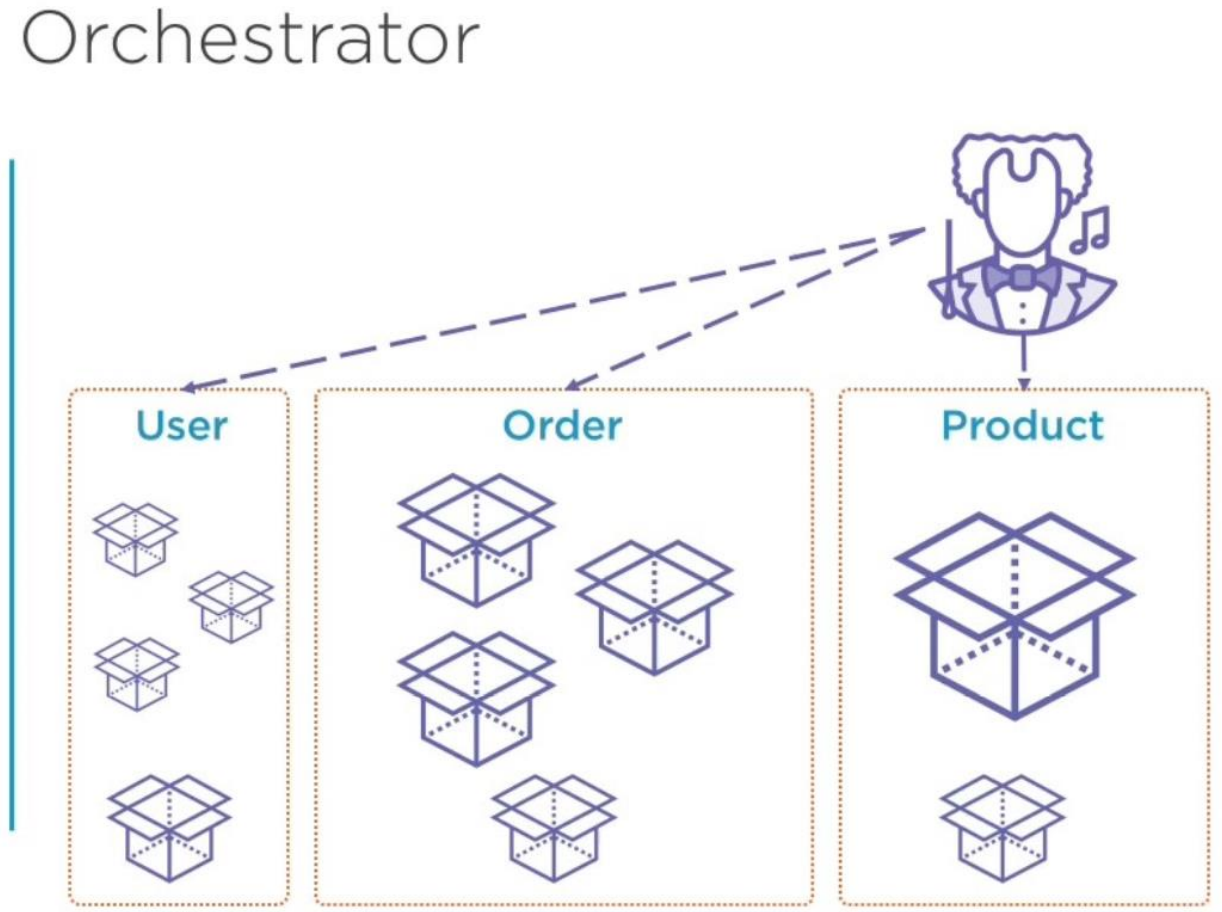
Multiple containers

Multiple machines

Start at the right time

Failed containers

Kubernetes, Mesos, Docker Swarm



Deployment

Continuous Delivery

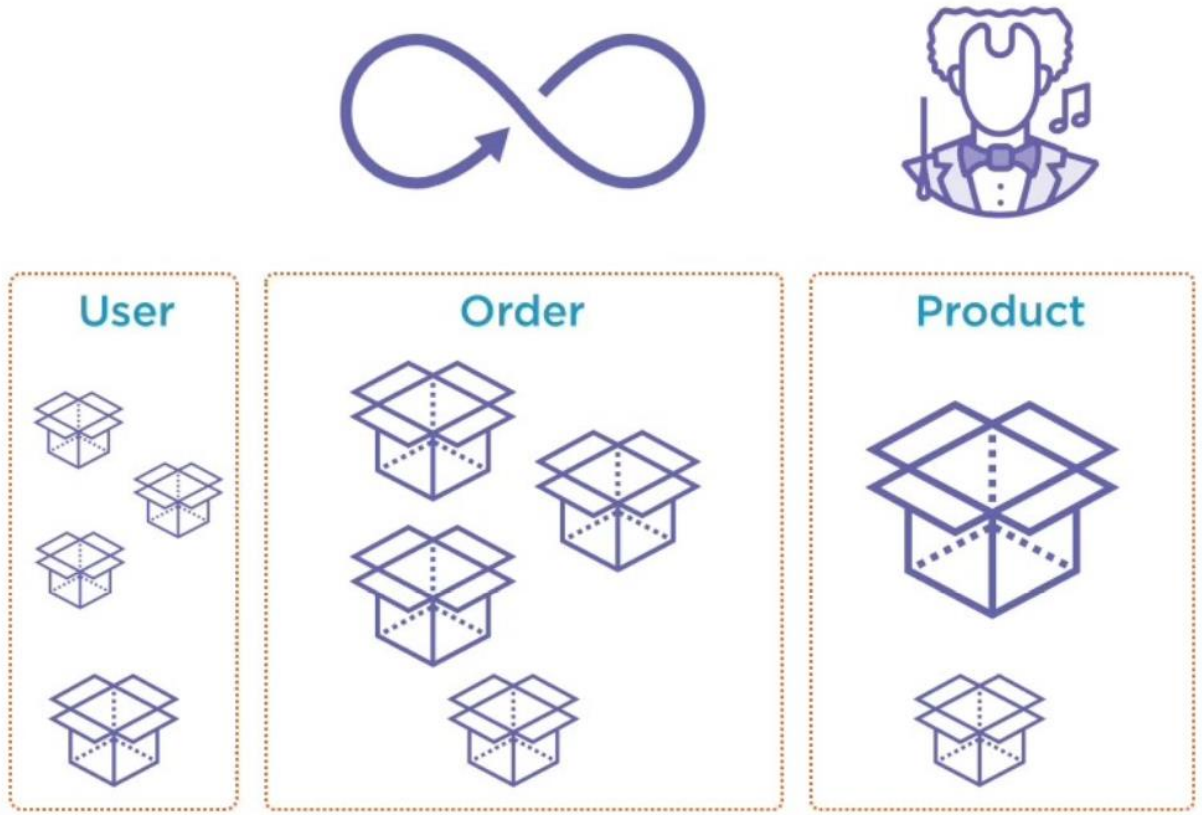
Automate deployment

Cost-effective

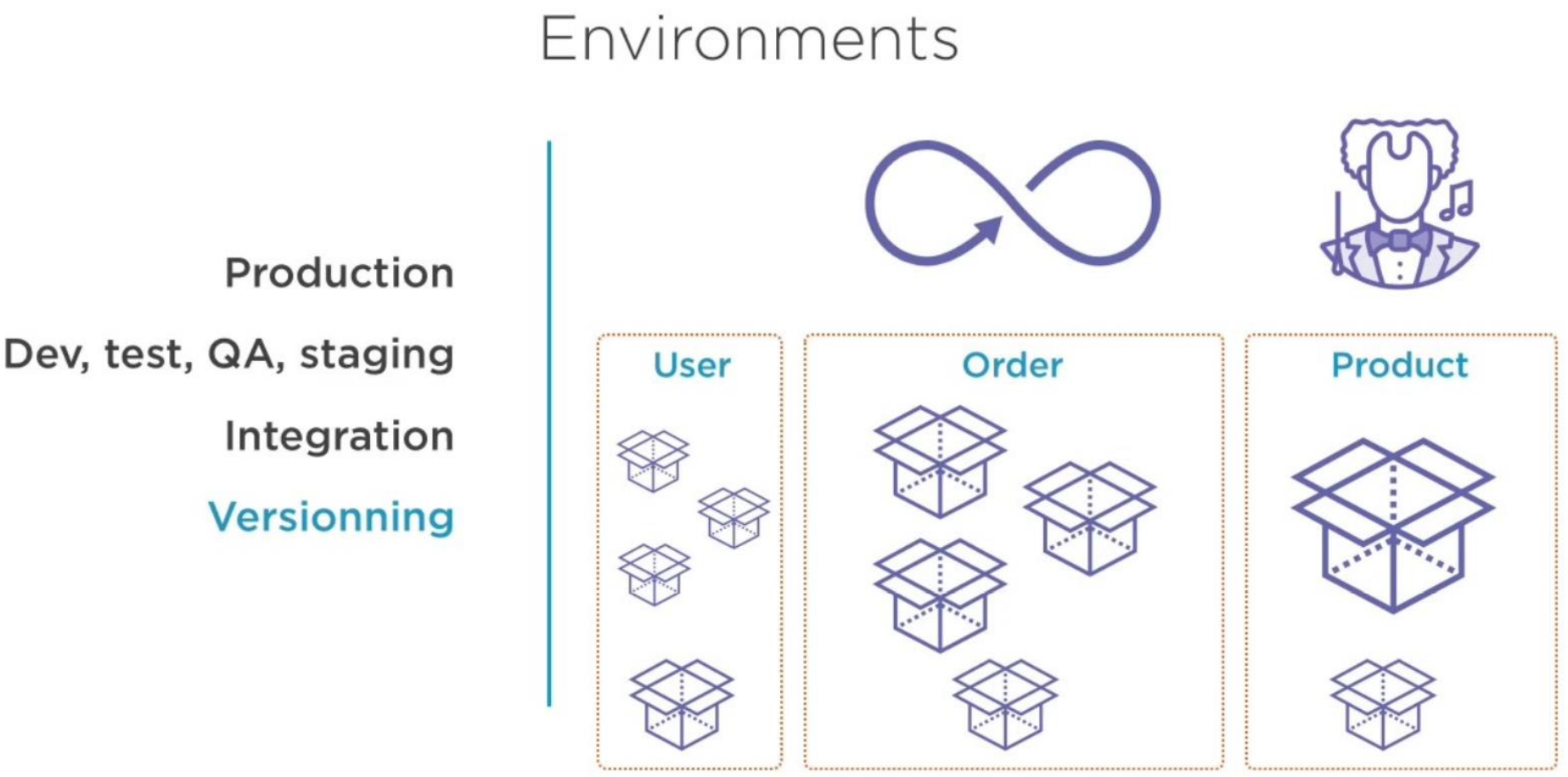
Quick

Reliable

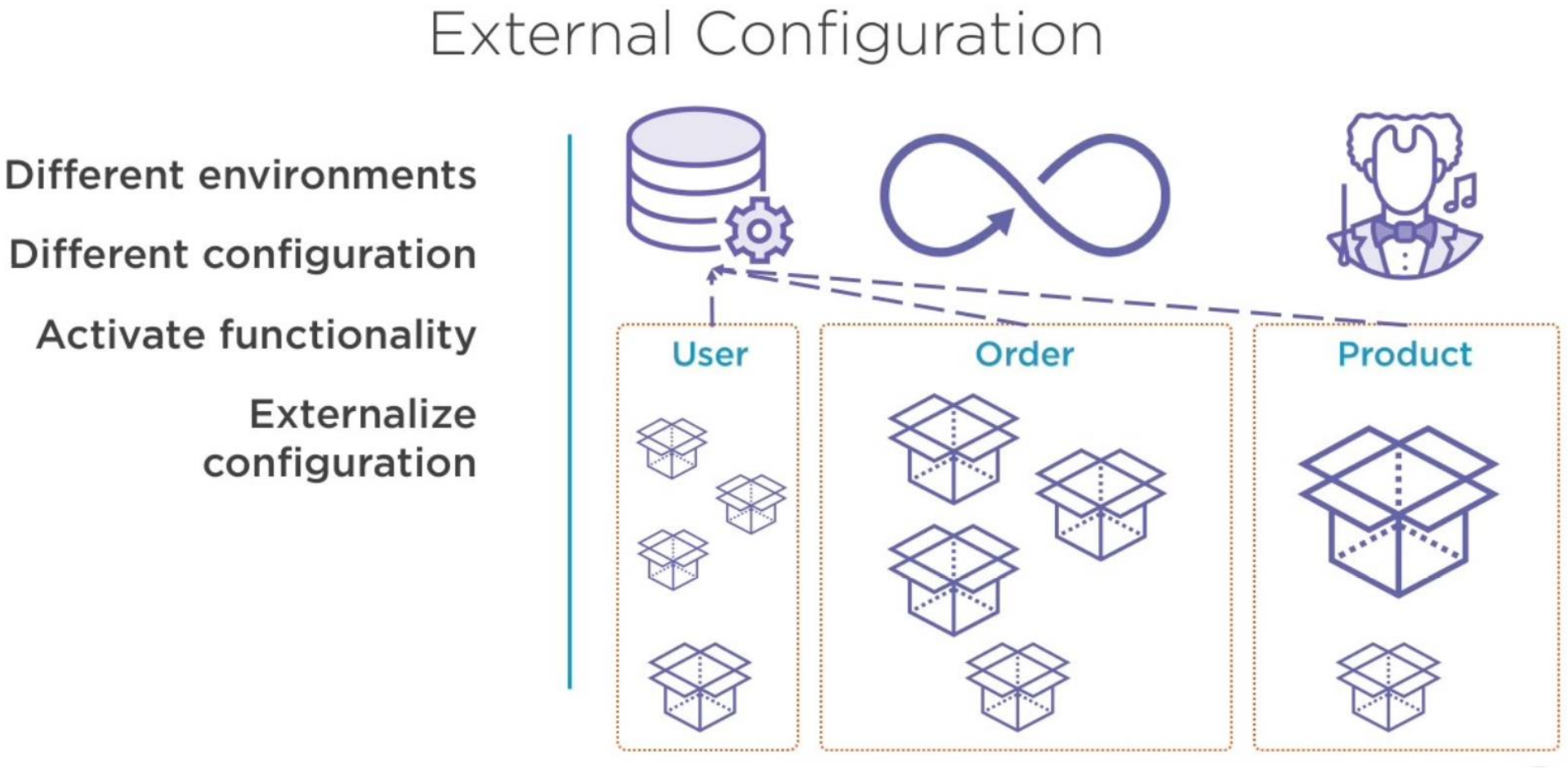
Build, test, deploy



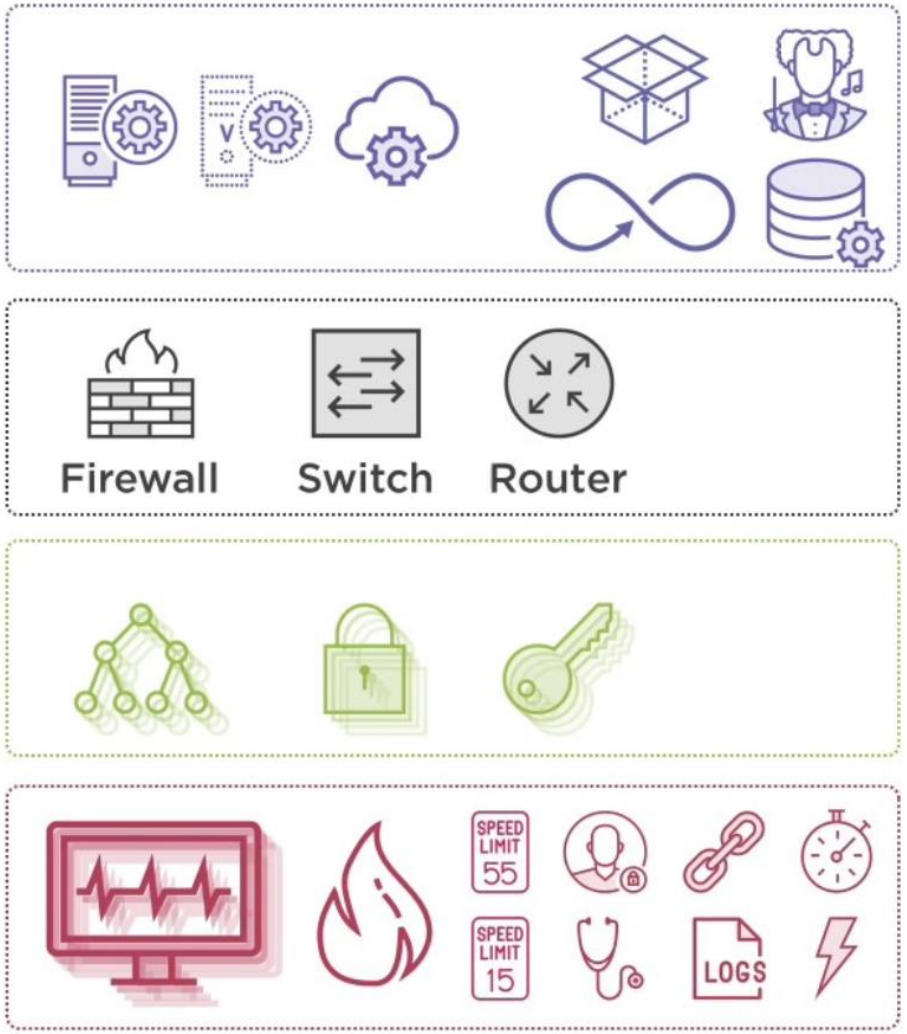
Deployment

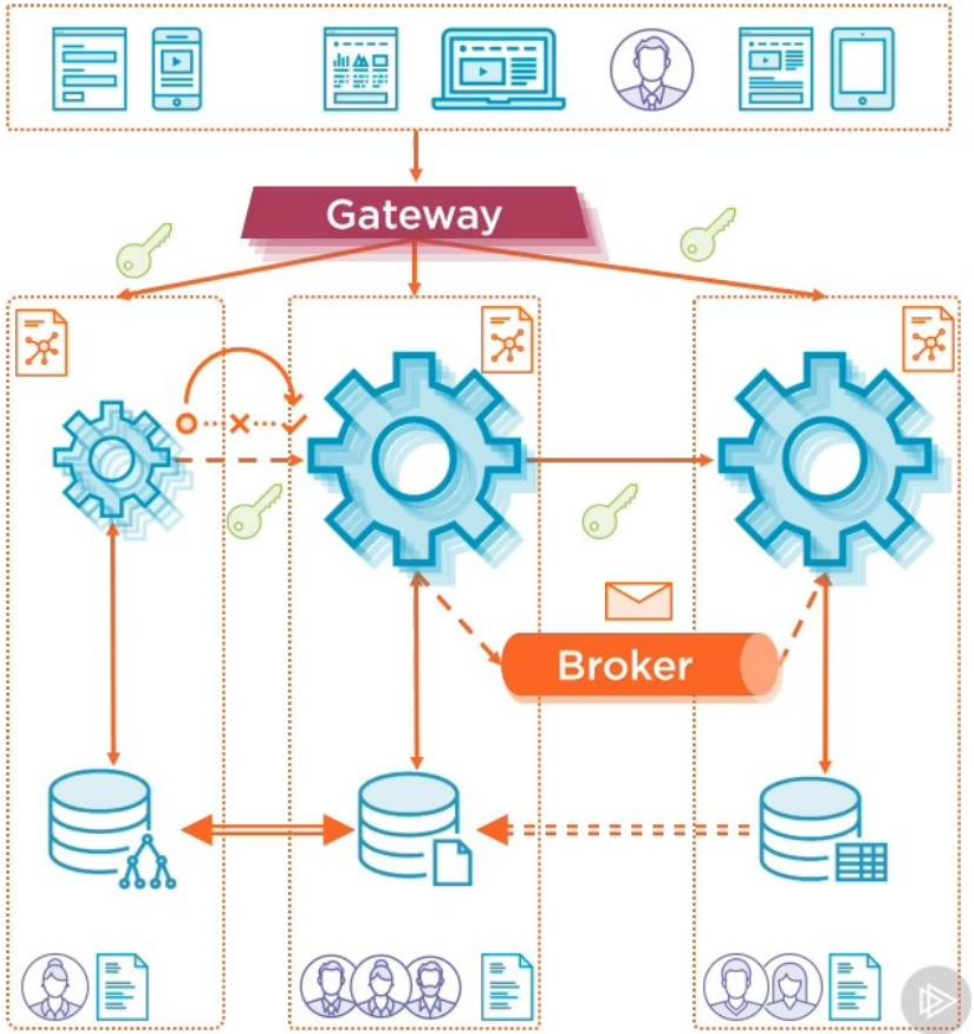

Environments
Production
Dev, test, QA, staging
Integration
Versionning
User
Order
Product

Deployment


External Configuration
Different environments
Different configuration
Activate functionality
Externalize configuration
User
Order
Product

Revisiting the Microservices Elements





Огляд сучасних платформ для створення мікросервісів.

- Створення **Spring Boot** мікросервісів
  - Spring Data JPA Repository Interfaces
  - Expose RESTful APIs with Spring Data REST
  - Expose RESTful APIs with Spring MVC
- Створення мікросервісів за допомогою **.NET Core**
  - Філософія мікросервісів та фреймворка .NET схожа, завдяки чому він чудово підходить для їх побудови.
  - З появою ASP.NET Core розробники отримали можливість будувати крос-платформні мікросервіси за допомогою .NET на Windows, Linux або Mac, а також з легкістю розгортати їх у хмарі.