Microservices

INTRODUCING MICROSERVICES

What are microservices?

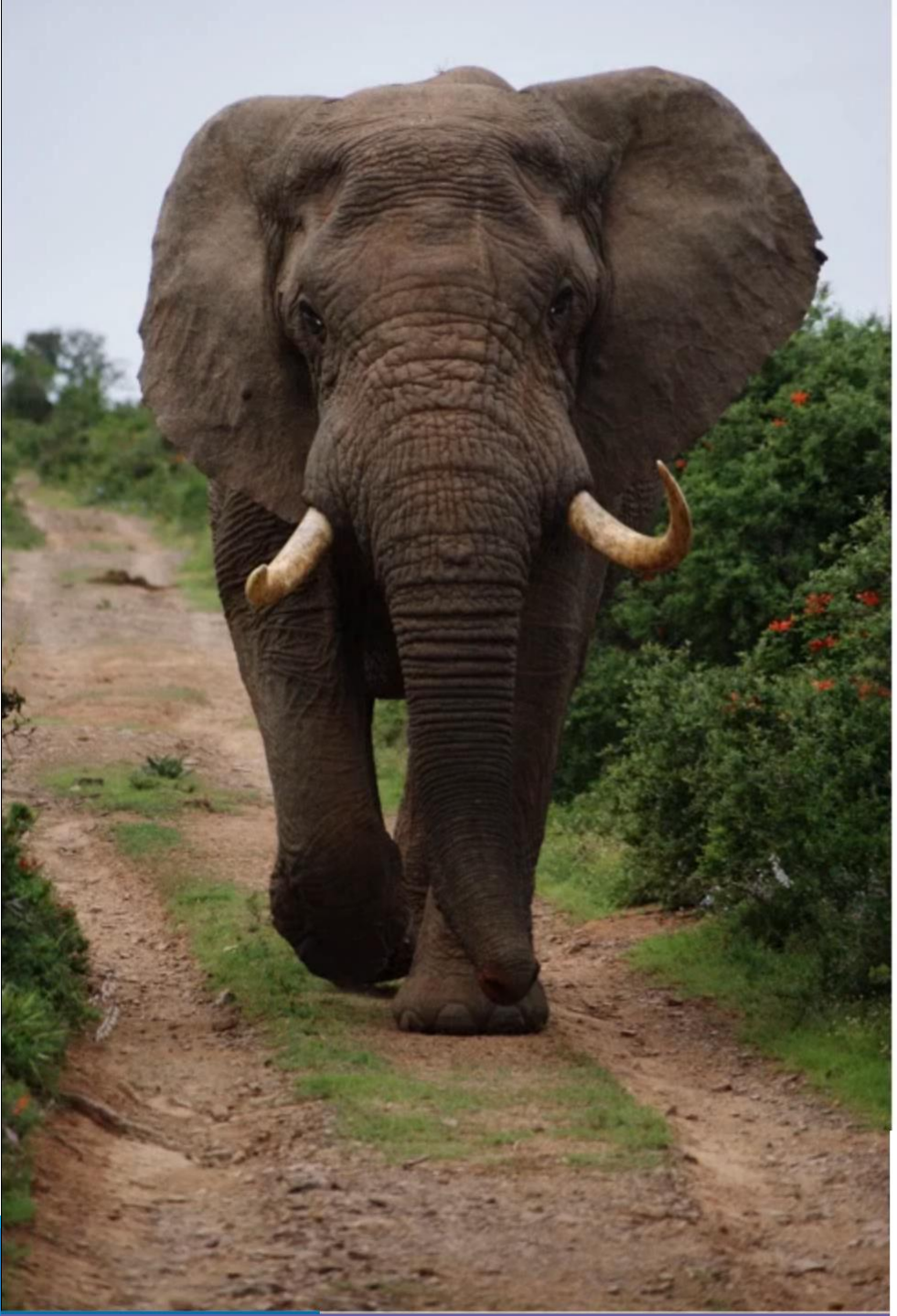
Microservices

autonomous, independently deployable services...

...collaborate to form an application

How small should they be?

Why do we need microservices?



"Monoliths"

Single codebase

Single process

Single host

Single database

Consistent technology

Monolith Benefits

Simplicity

One codebase

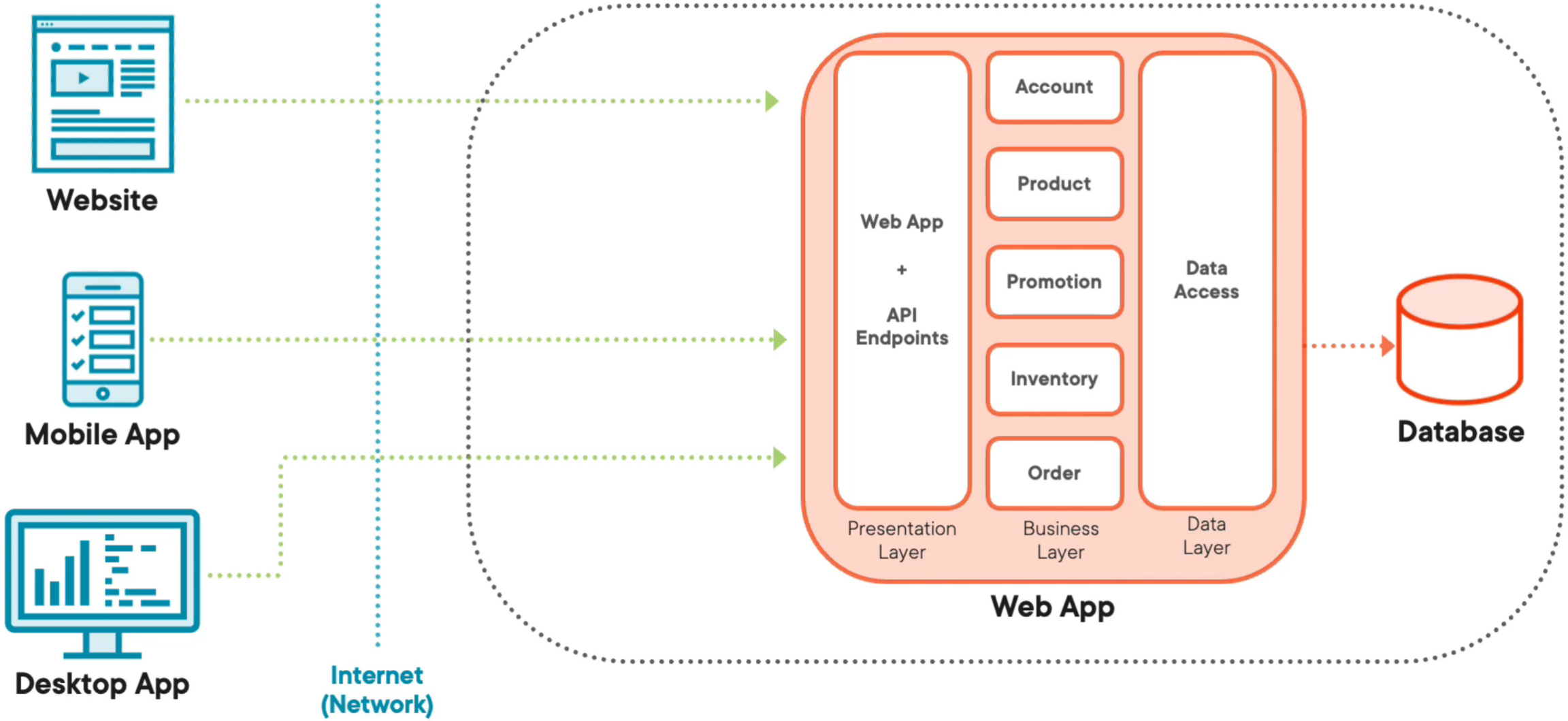
- Easy to find things

Deployment

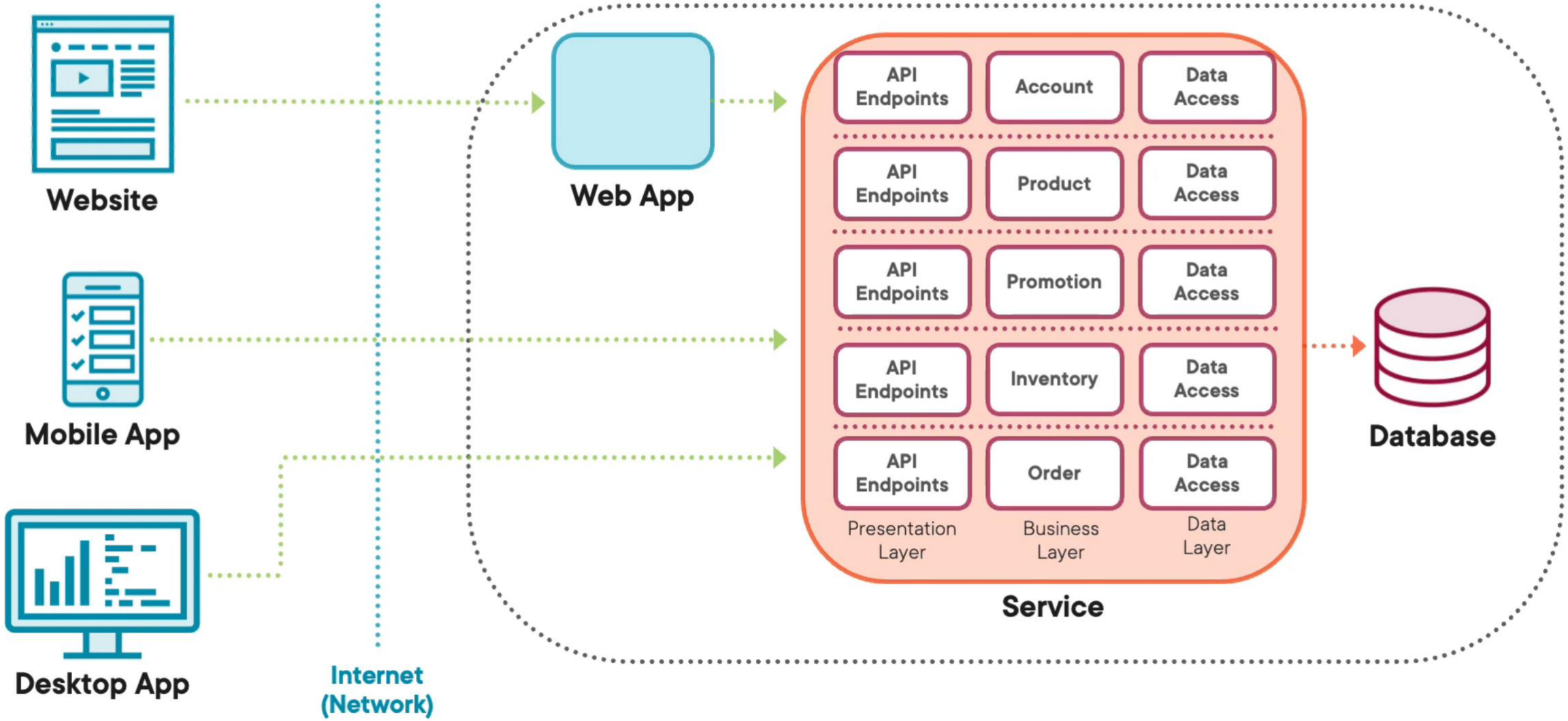
- One application to replace

Monoliths are not “wrong”

The Pure Monolith



Microservices Ready Monolith



Do you really need microservices?

The Problem of Scale

Monoliths can work well for small applications

The Problem of Scale

Monoliths can work well for small applications

Many developers

Many users

Big data

Difficult to maintain

Technical debt

Entangled modules

Monolith Problems

Difficult to deploy

- Risky
- Requires downtime

Difficult to scale

- Horizontal scaling often not possible
- Vertical scaling is expensive
- Whole application must be scaled

Wedded to legacy technology

- Reduces agility

What Is a Service?

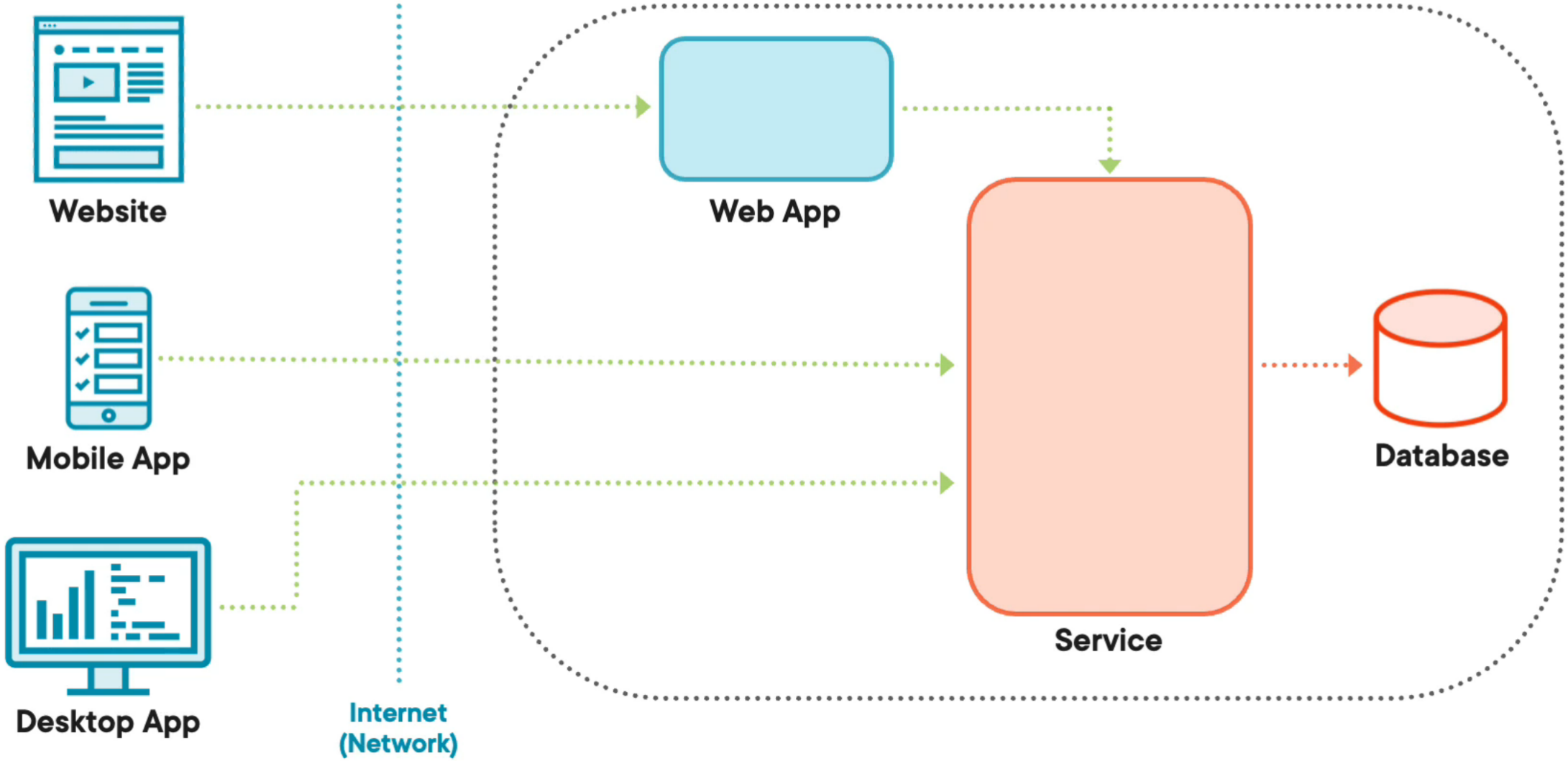
Website

Mobile App

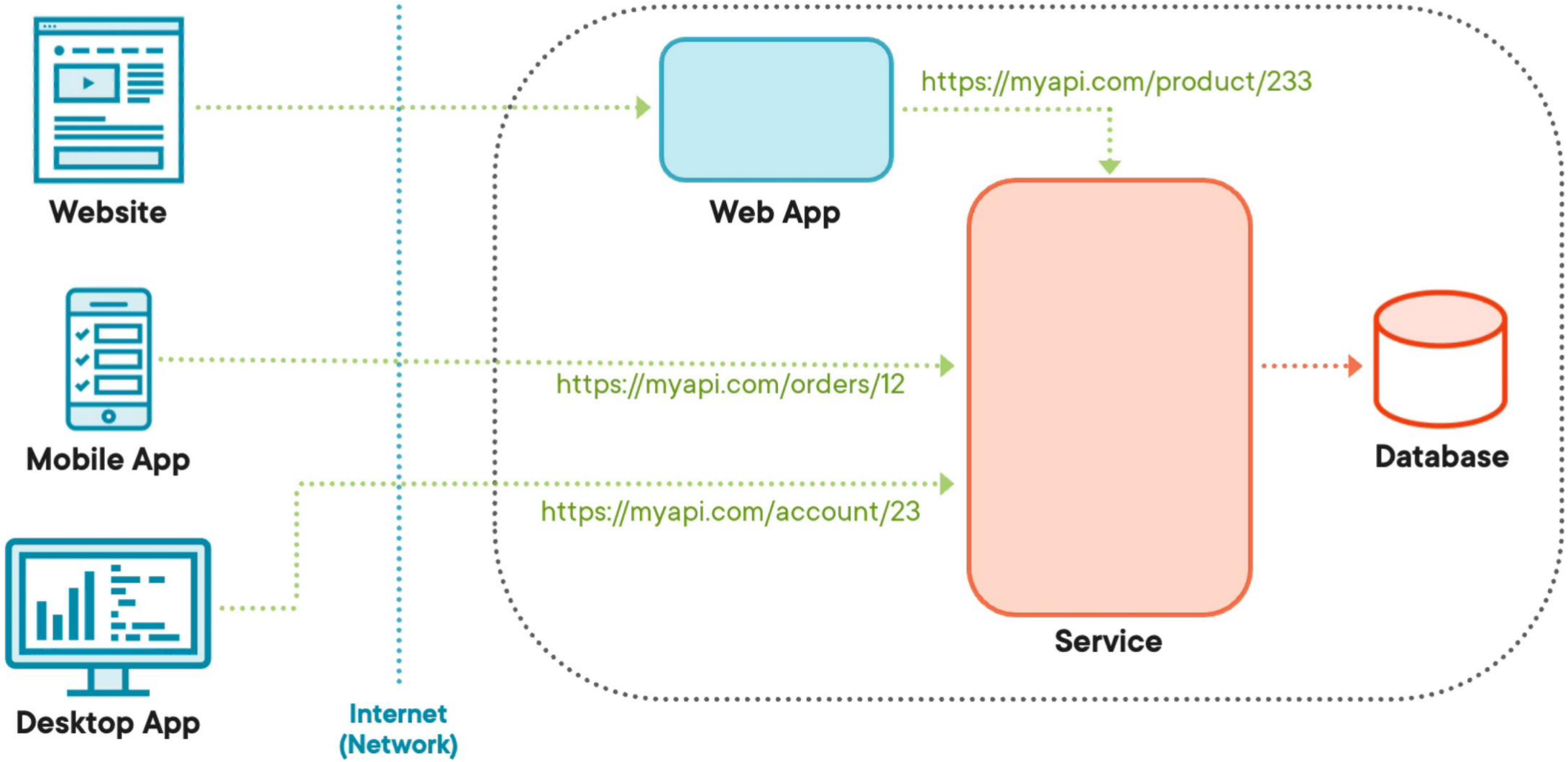
Desktop App

Internet
(Network)

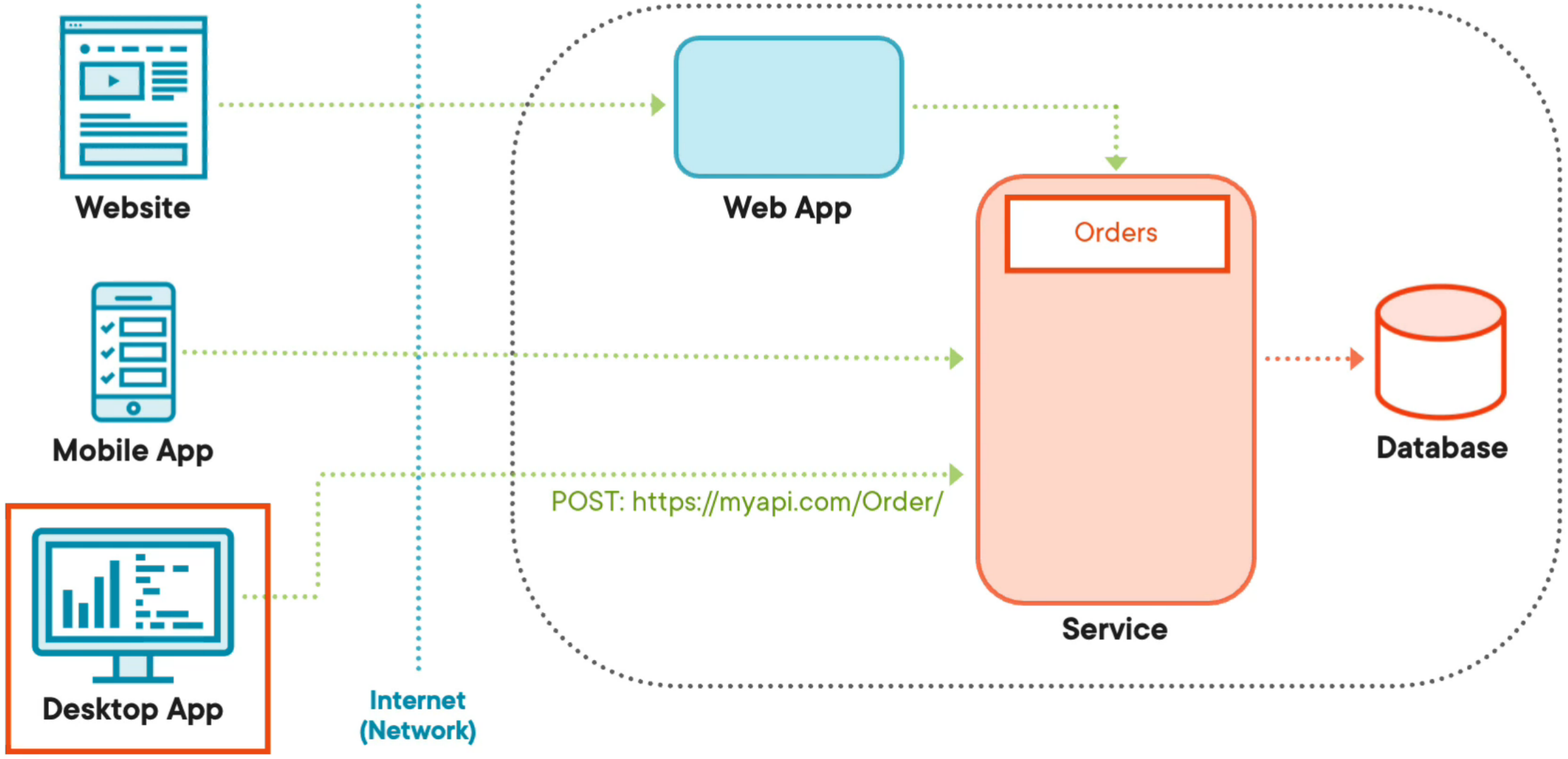
What Is a Service?



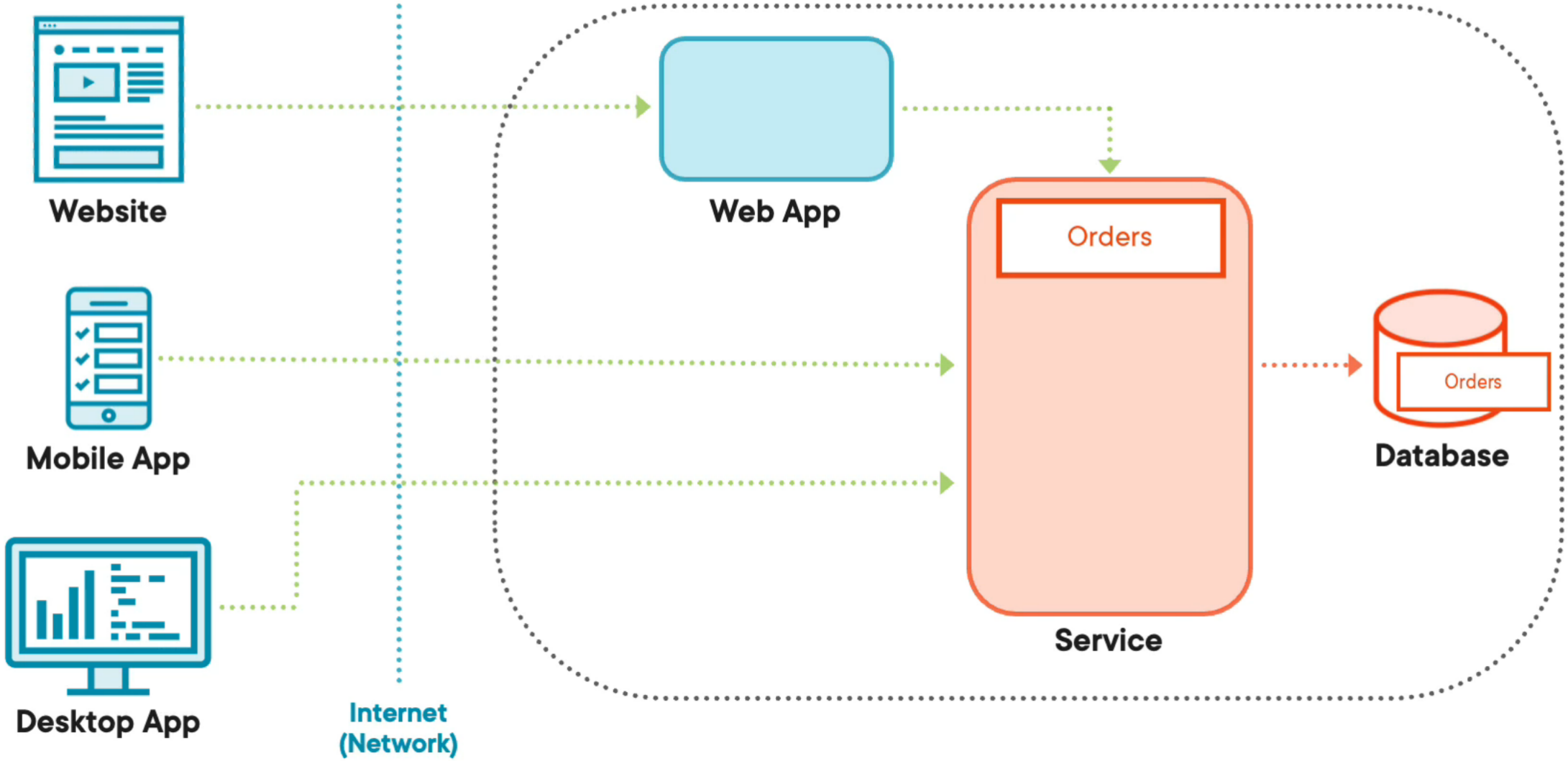
What Is a Service?



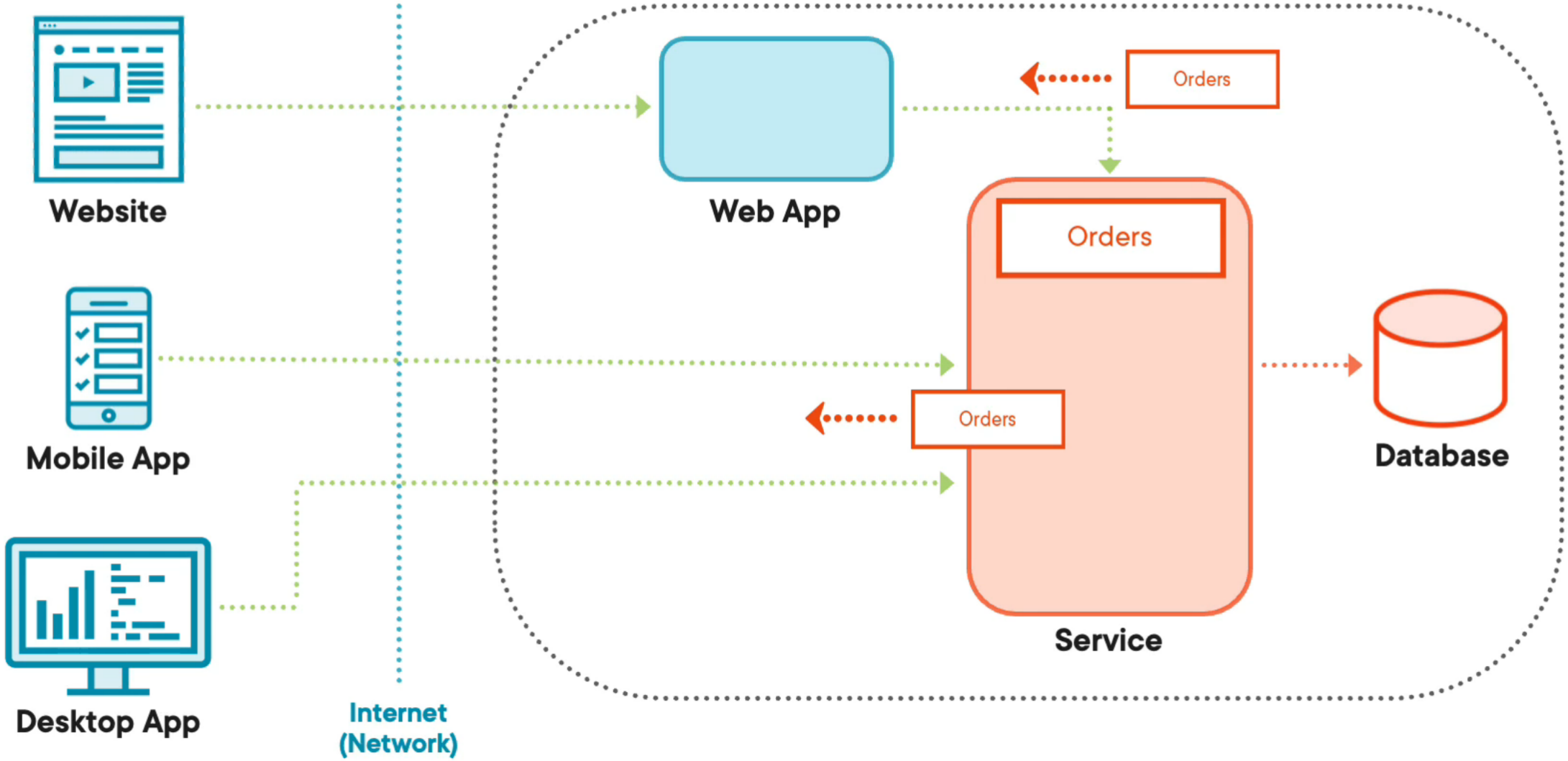
What Is a Service?



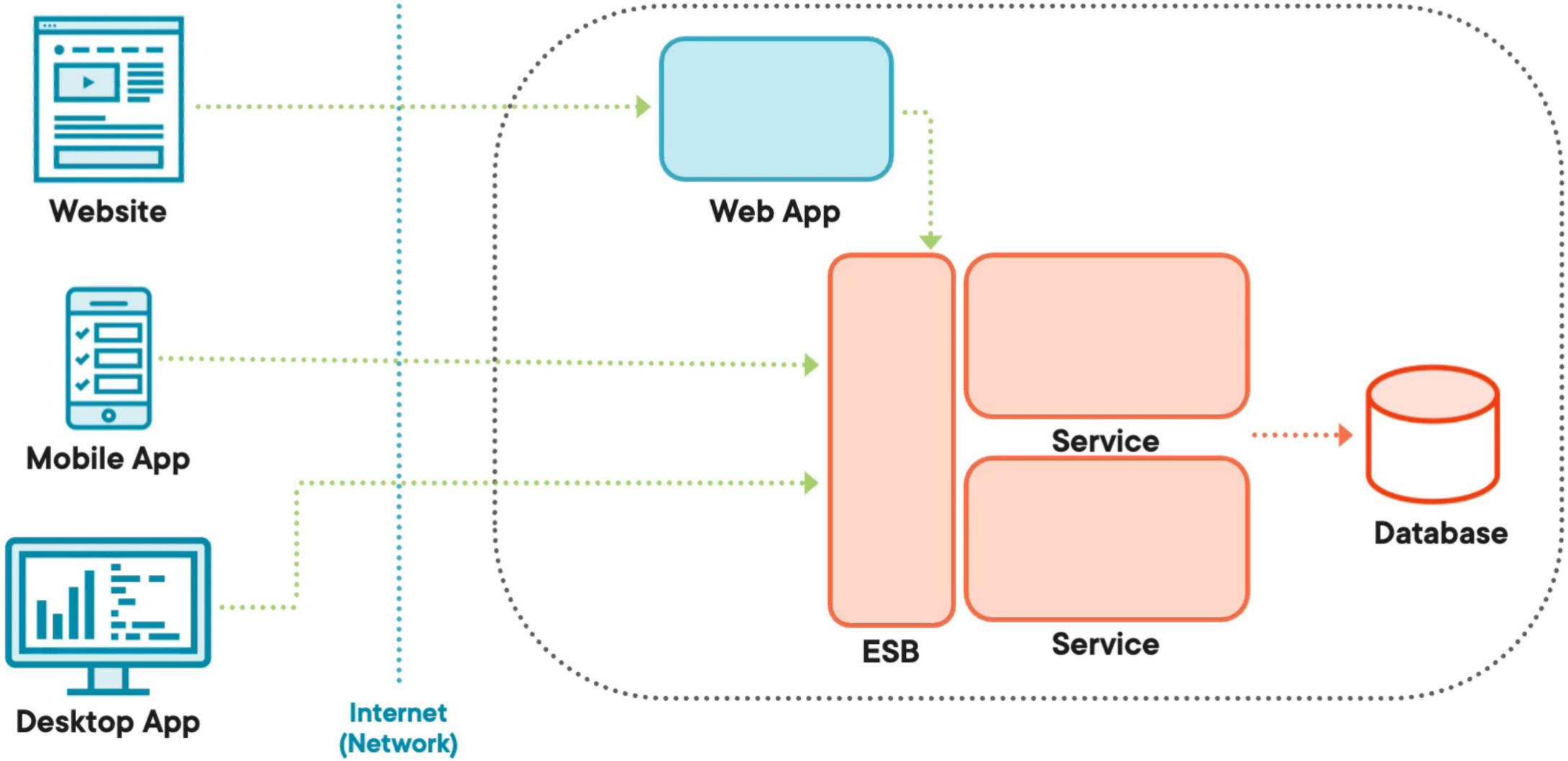
What Is a Service?



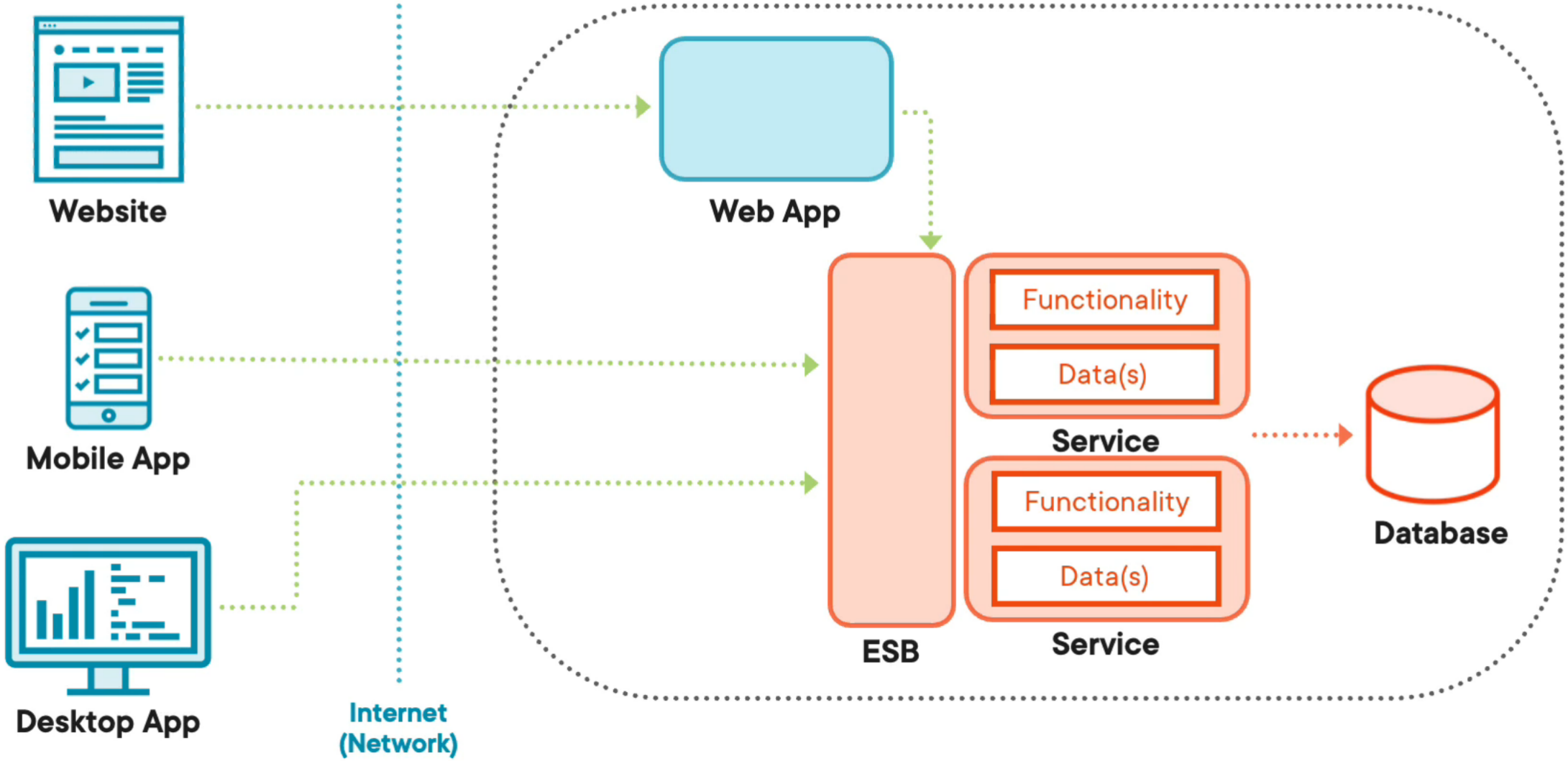
What Is a Service?



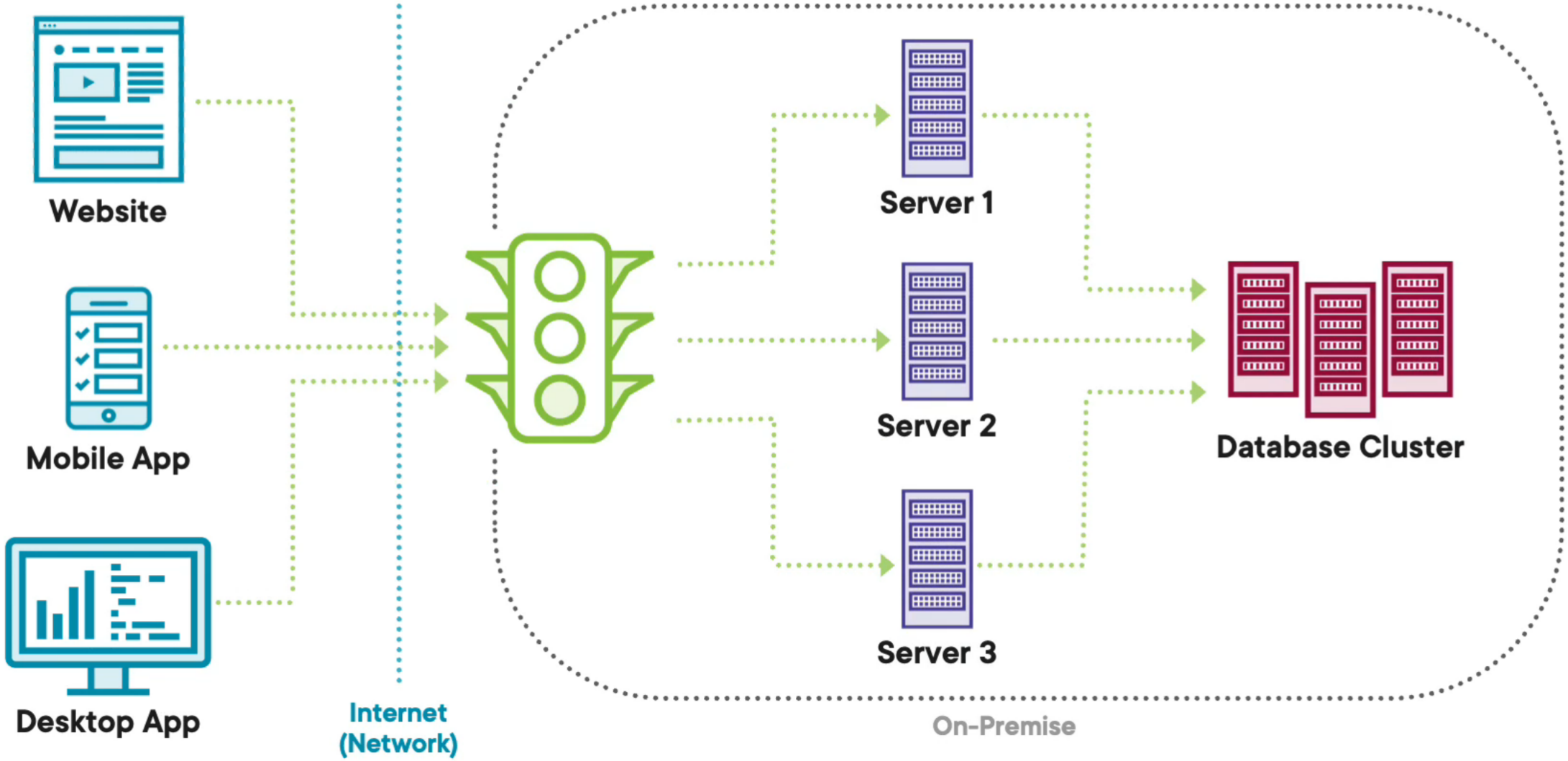
What Is a Service?



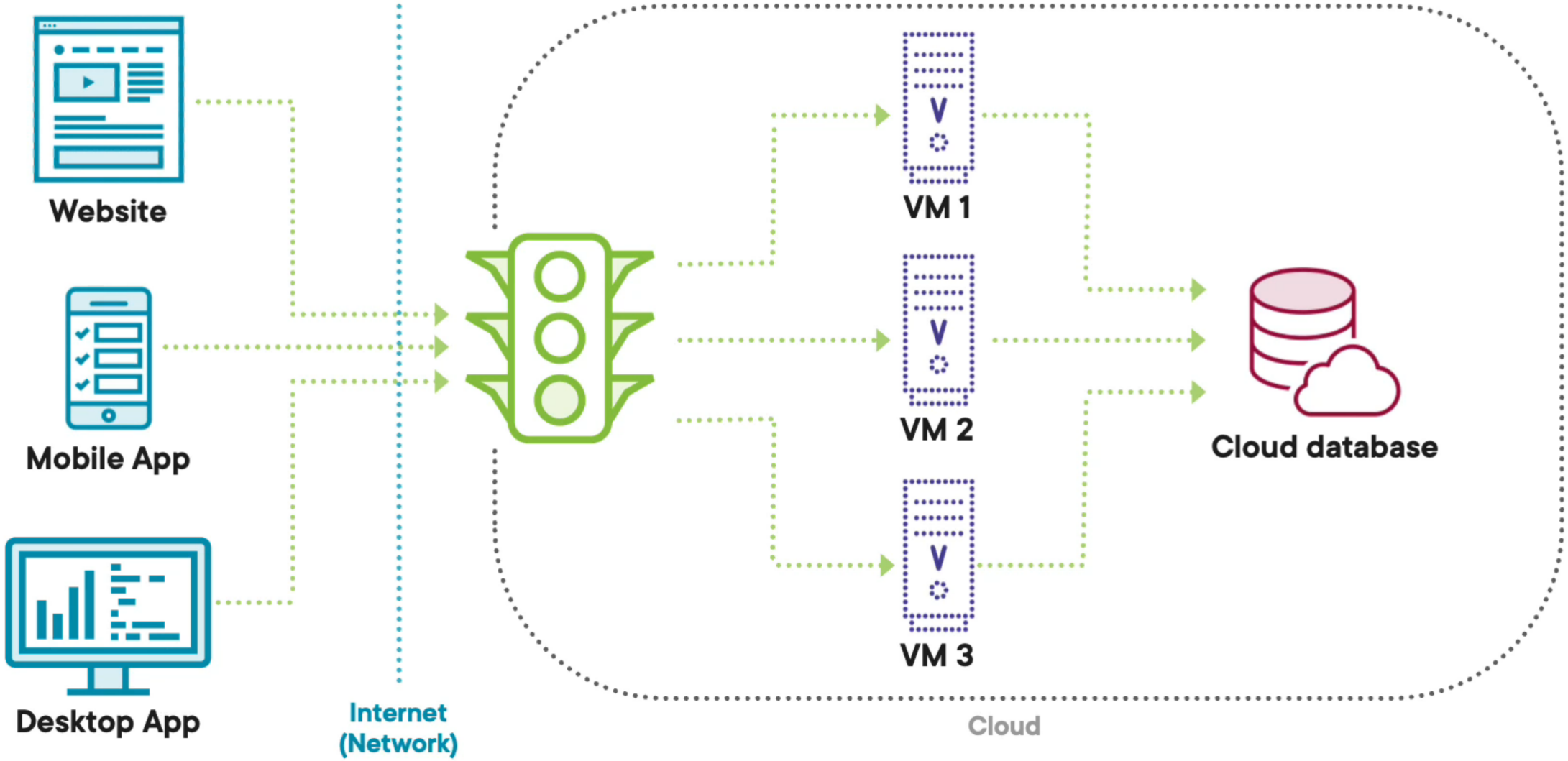
What Is a Service?



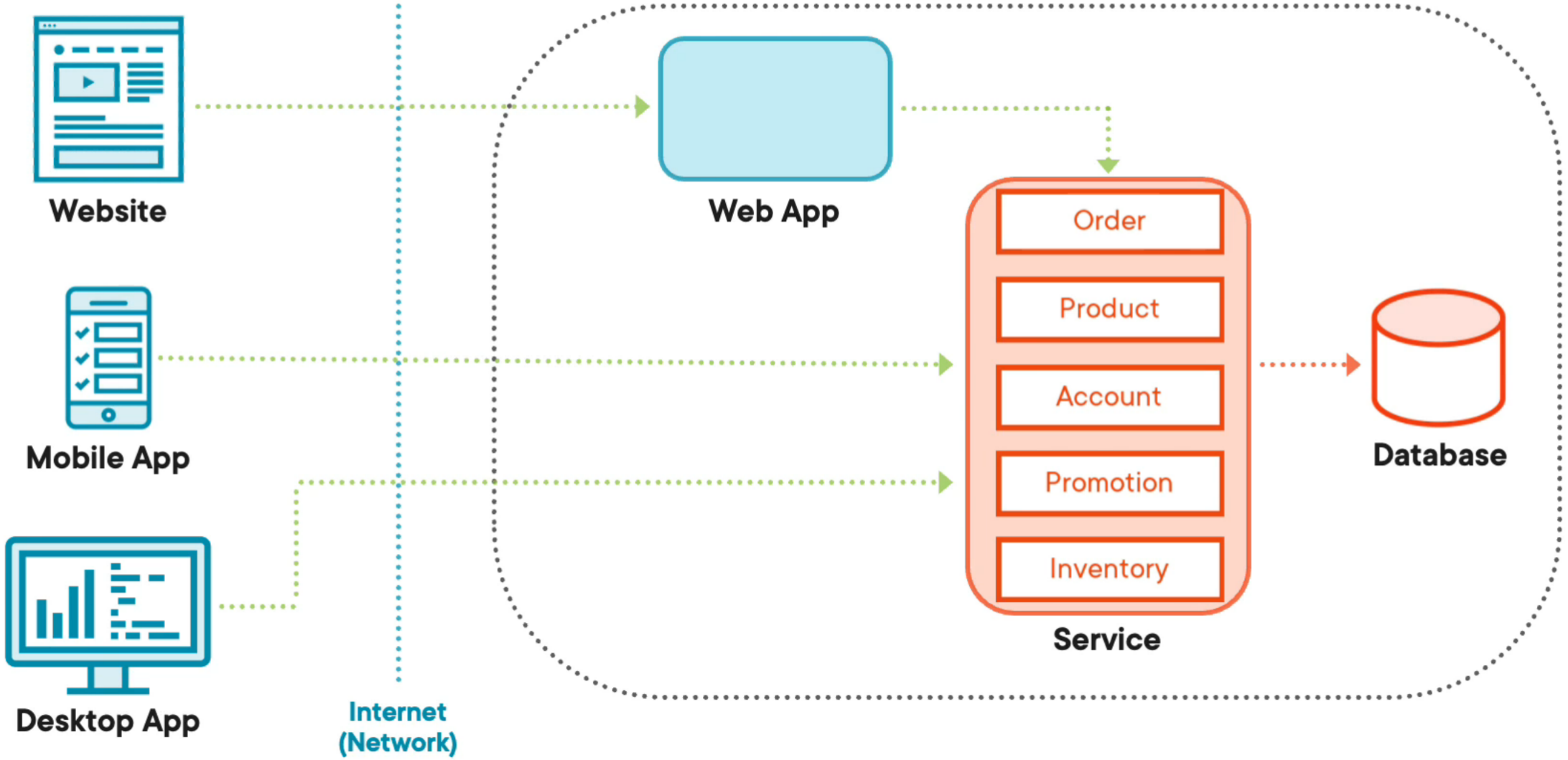
What Is a Service?



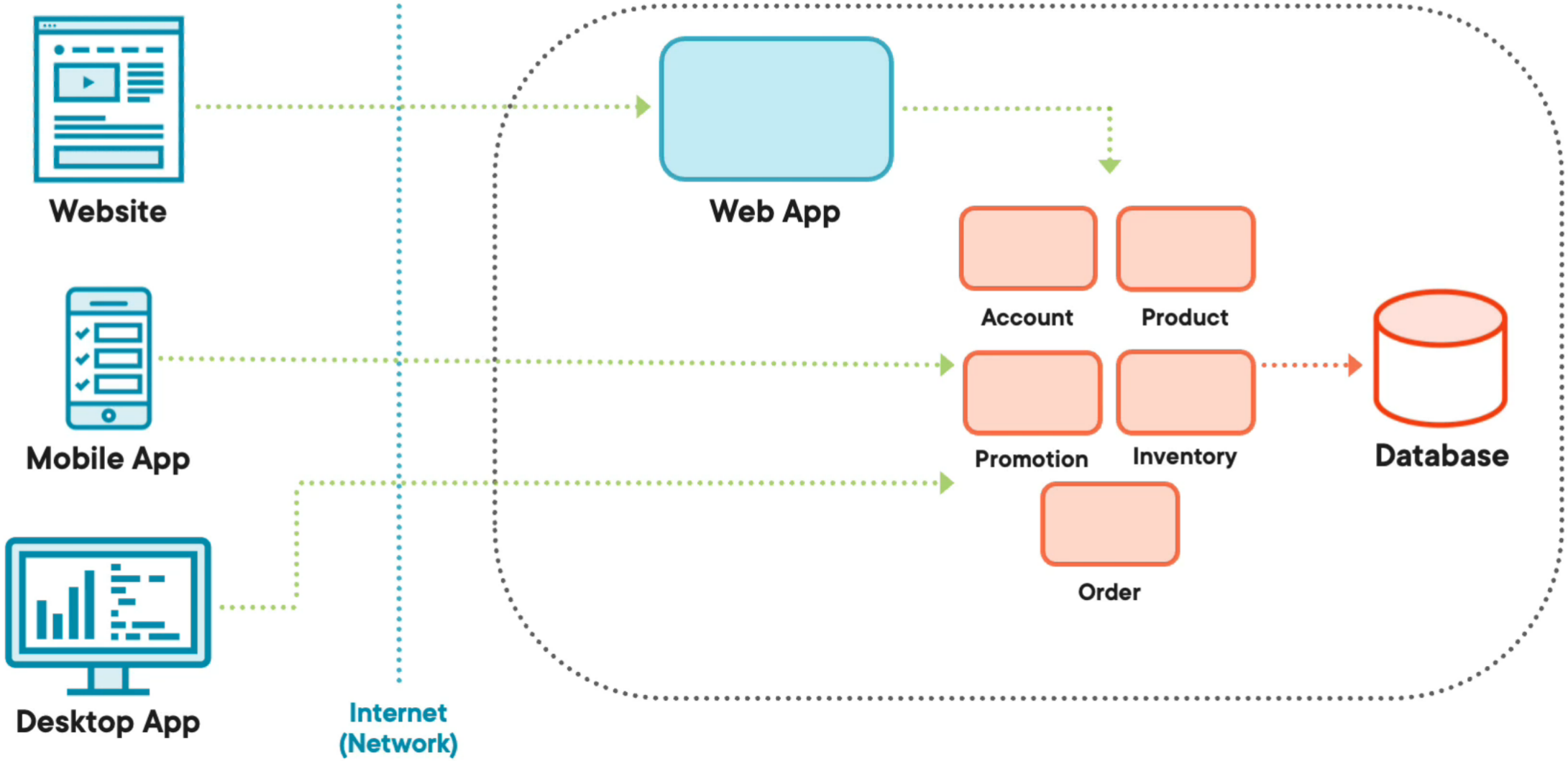
What Is a Service?



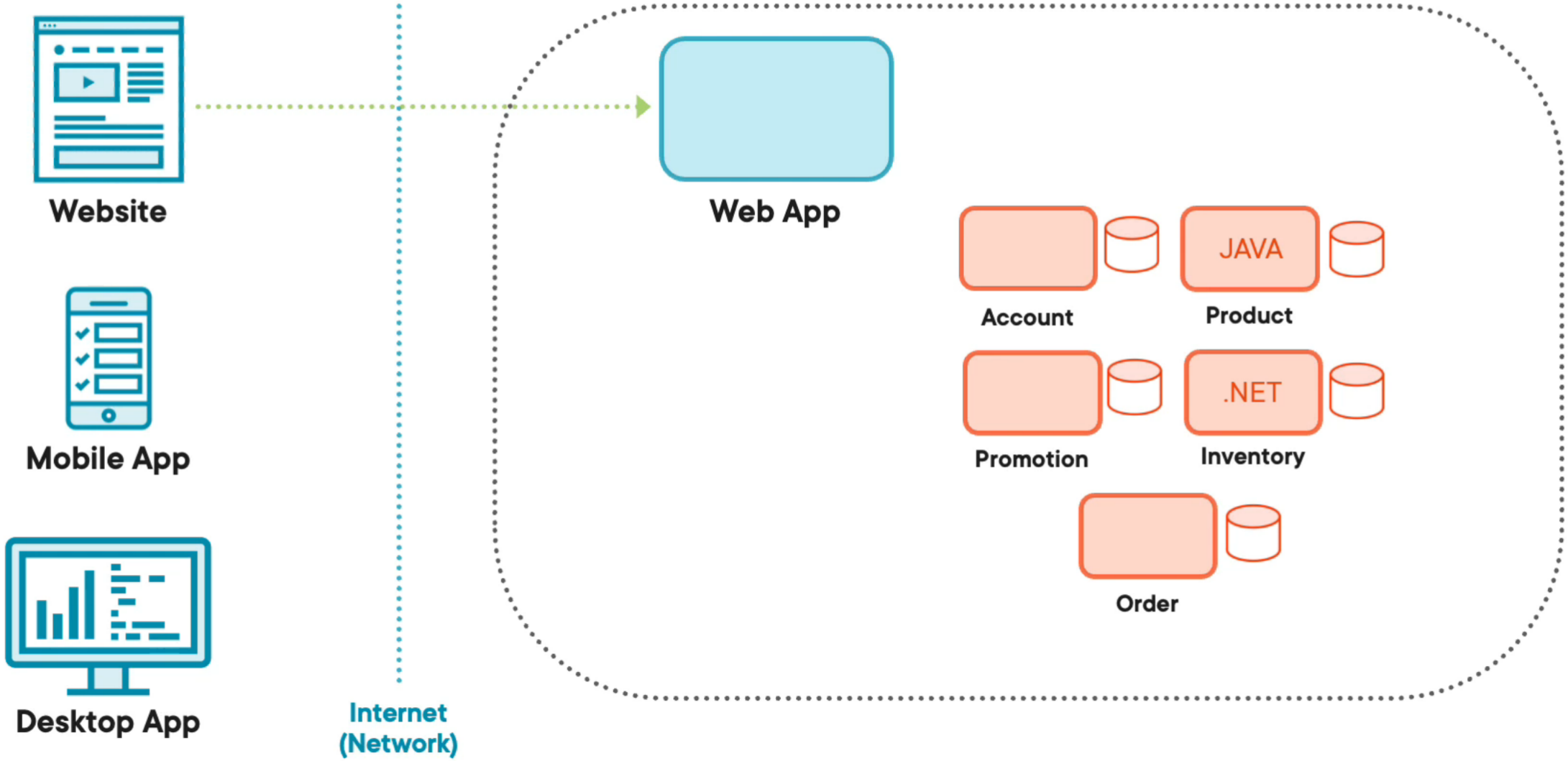
Microservices Architecture



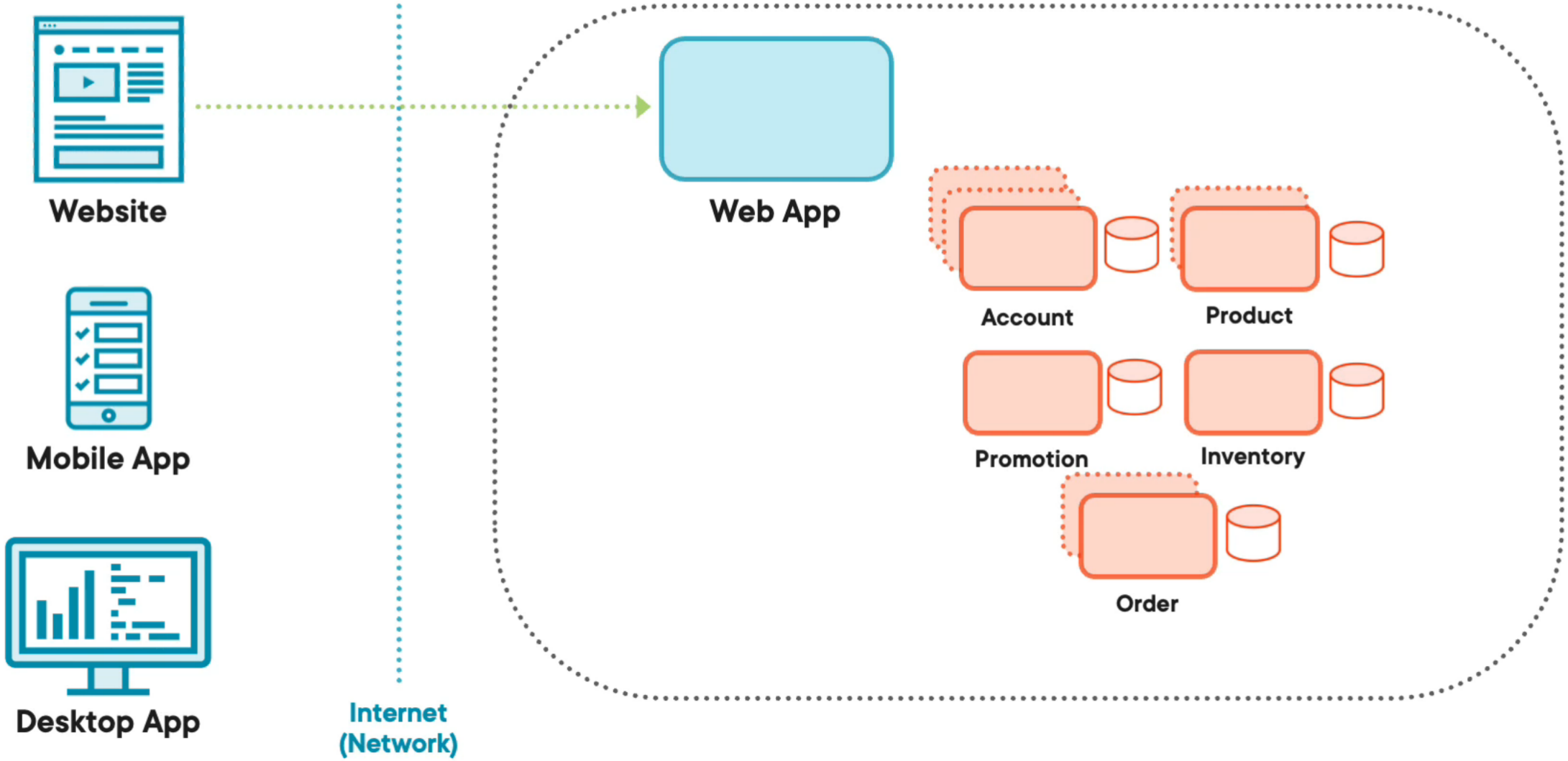
Microservices Architecture



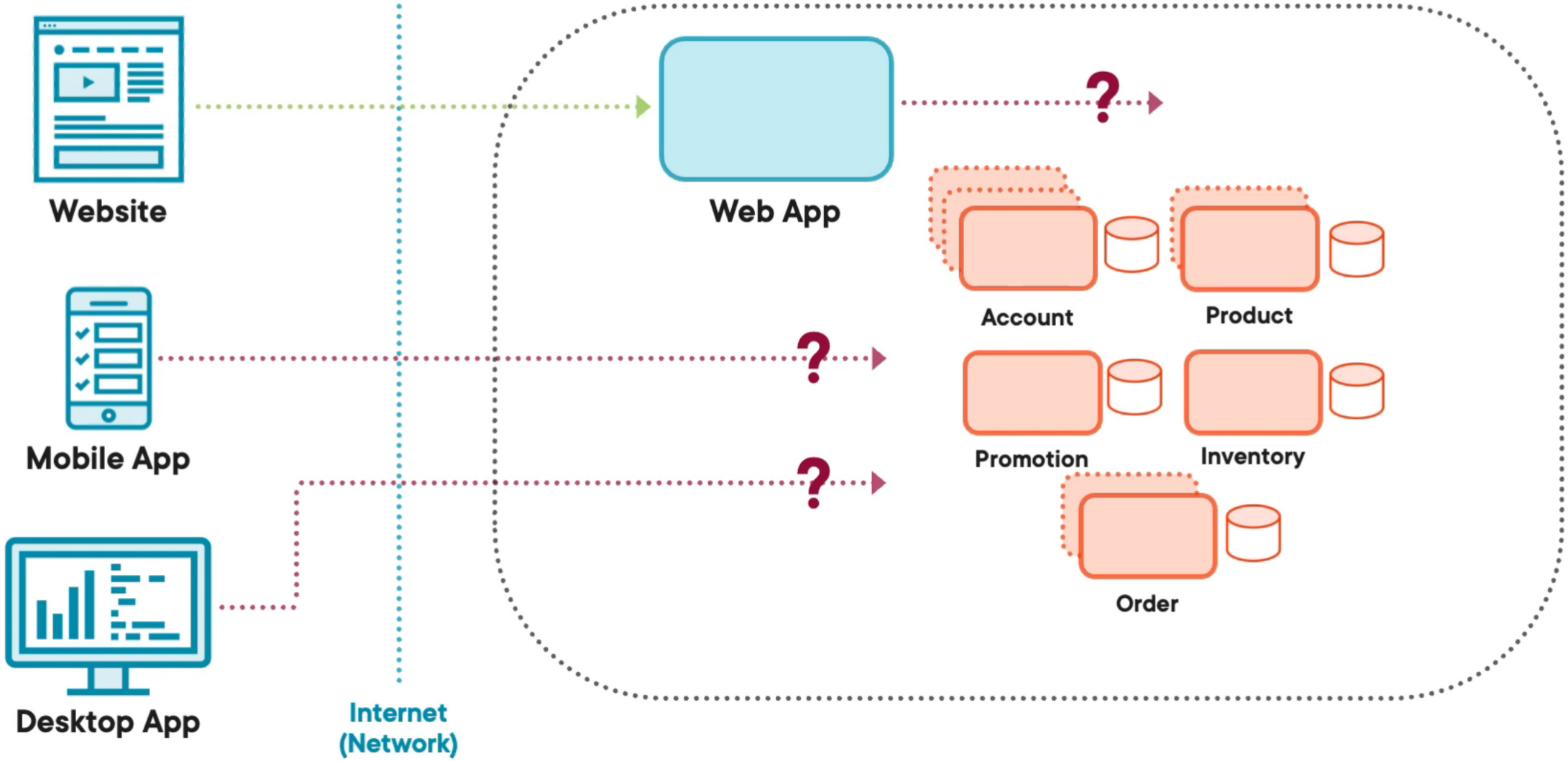
Microservices Architecture



Microservices Architecture

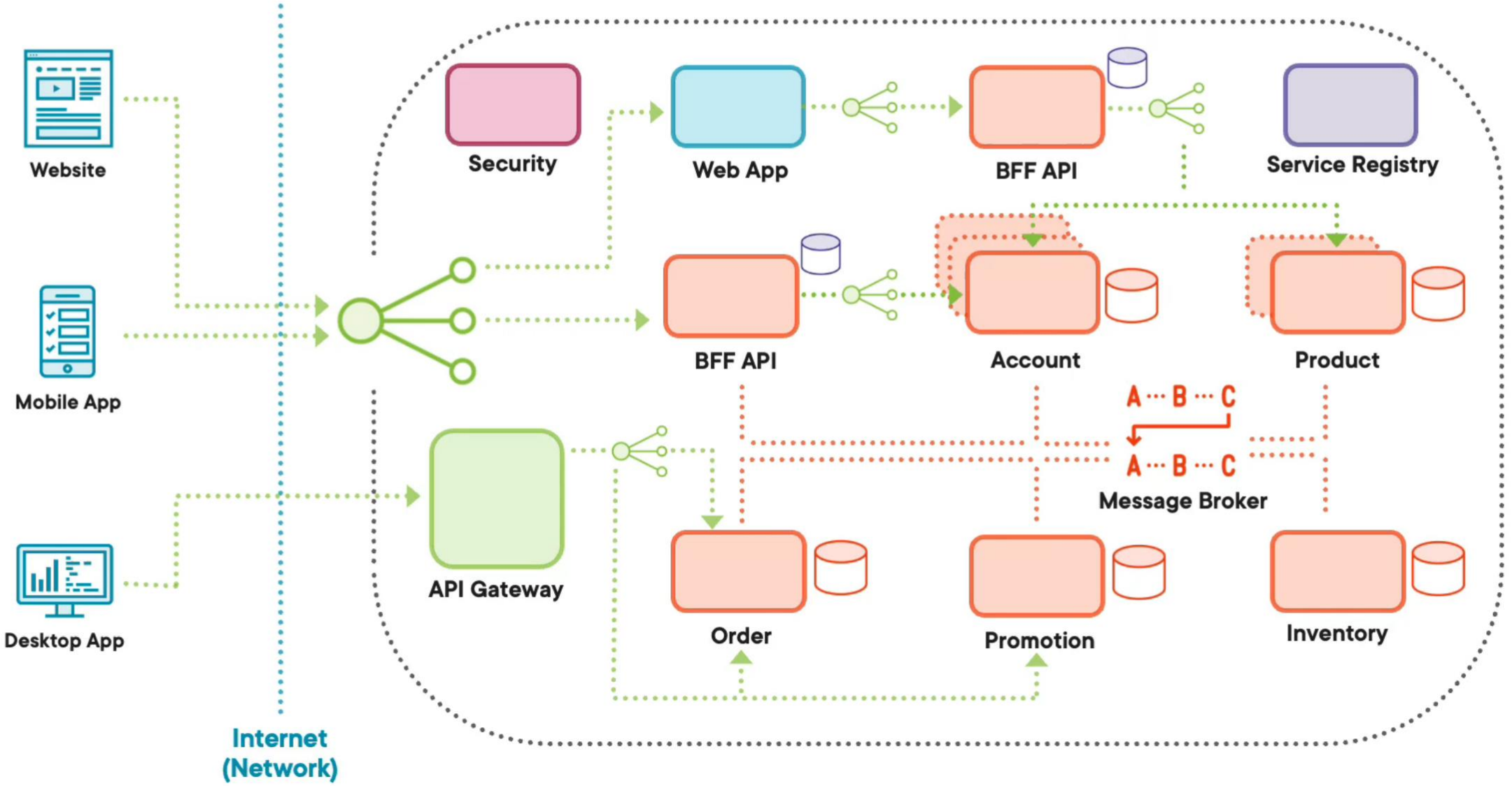


Microservices Architecture

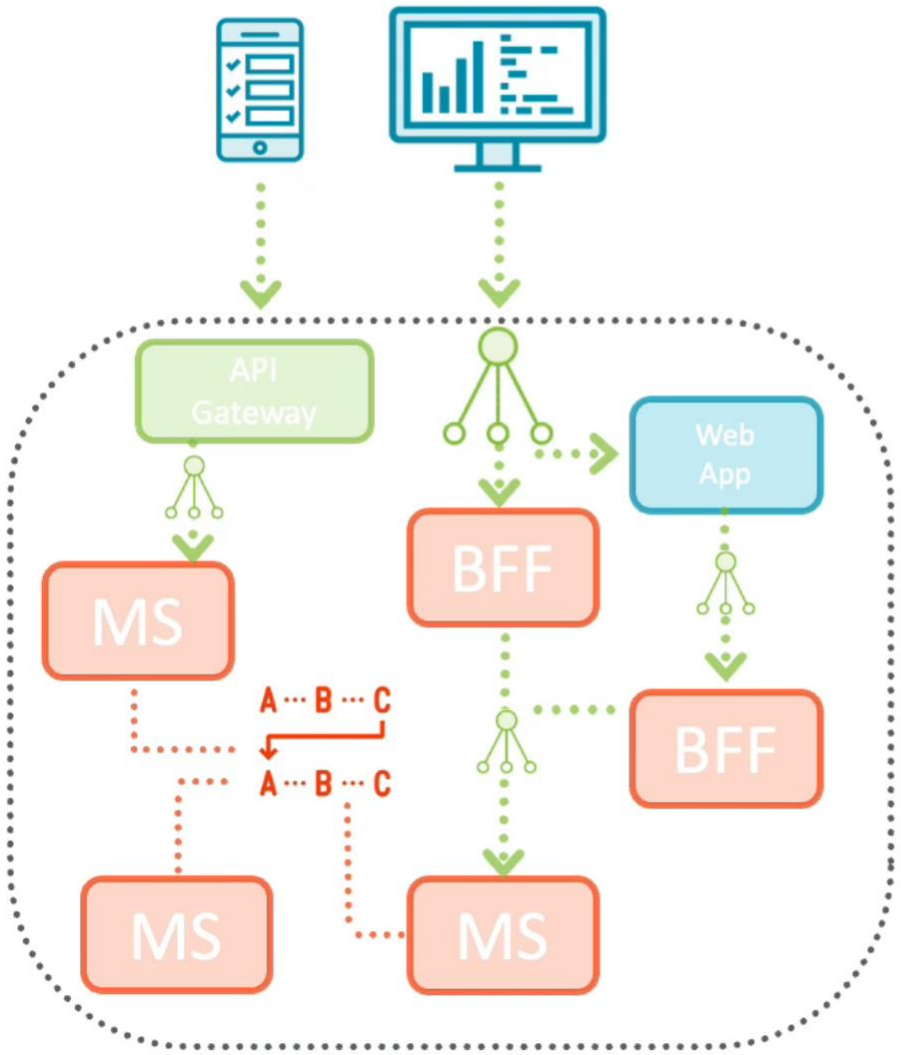


Microservices Architecture

- Authorization
- Authentication
- Worker Service
- API Based Service
- Eventual consistency
- Avoid Distributed Transactions
- Fire and forget calls
- Messages/Events
- Message Broker
- Asynchronous Communication
- Caches
- Temporal Coupling
- Request and Response (wait) calls
- Synchronous Communication
- API Gateway
- Service Registry
- Stateless Instances
- Load Balancer
- BFF API



Cheat-sheet: The Monolith



No restriction on size

Longer development and deployment

Inaccessible features

Fixed technology stack

High levels of coupling

Failure could affect the whole system

Inefficient scaling

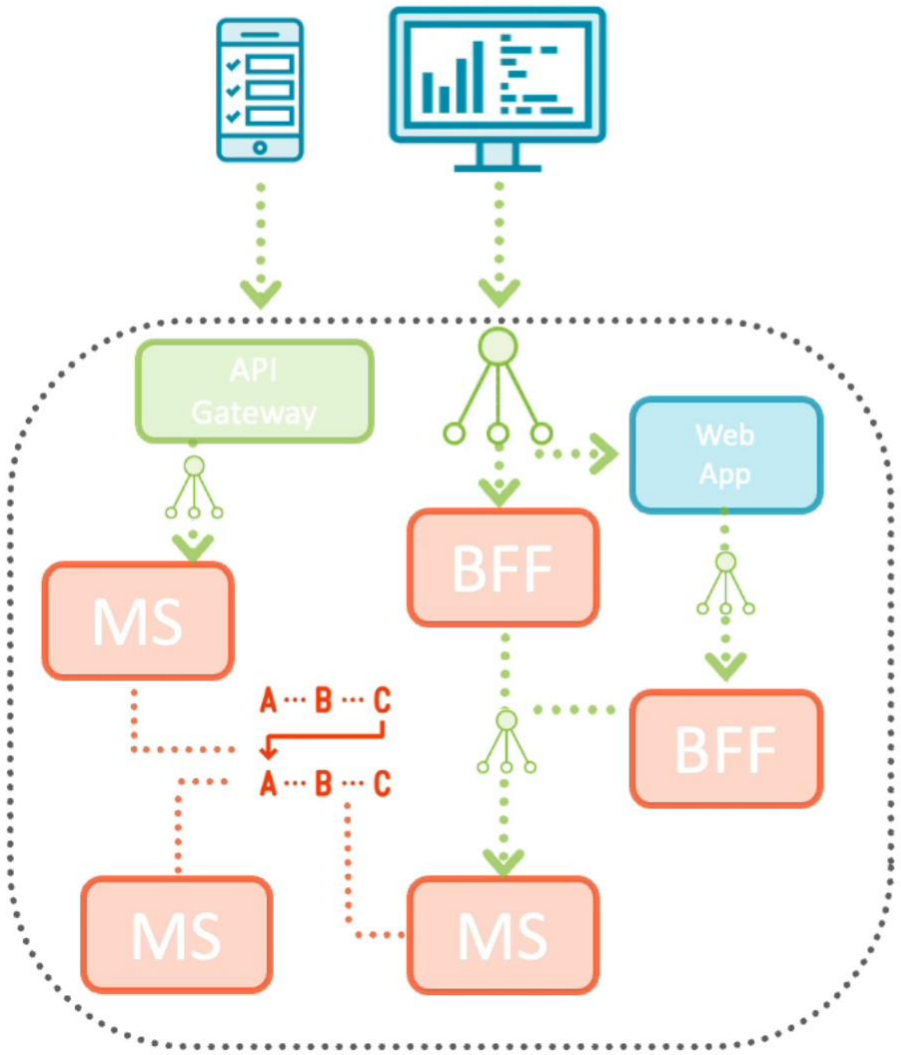
Minor change results in entire system deployment

Easy to replicate the environment

Quick solution for startups and prototyping

Monolith is an alternative

Cheat-sheet: Microservices Architecture



SOA with application-level scope for services

Smaller services with a cohesive focus

Independent data storage

Scalable, performant, and flexible applications

High availability and reliable application

Technology agnostic services

Independently changeable and deployable

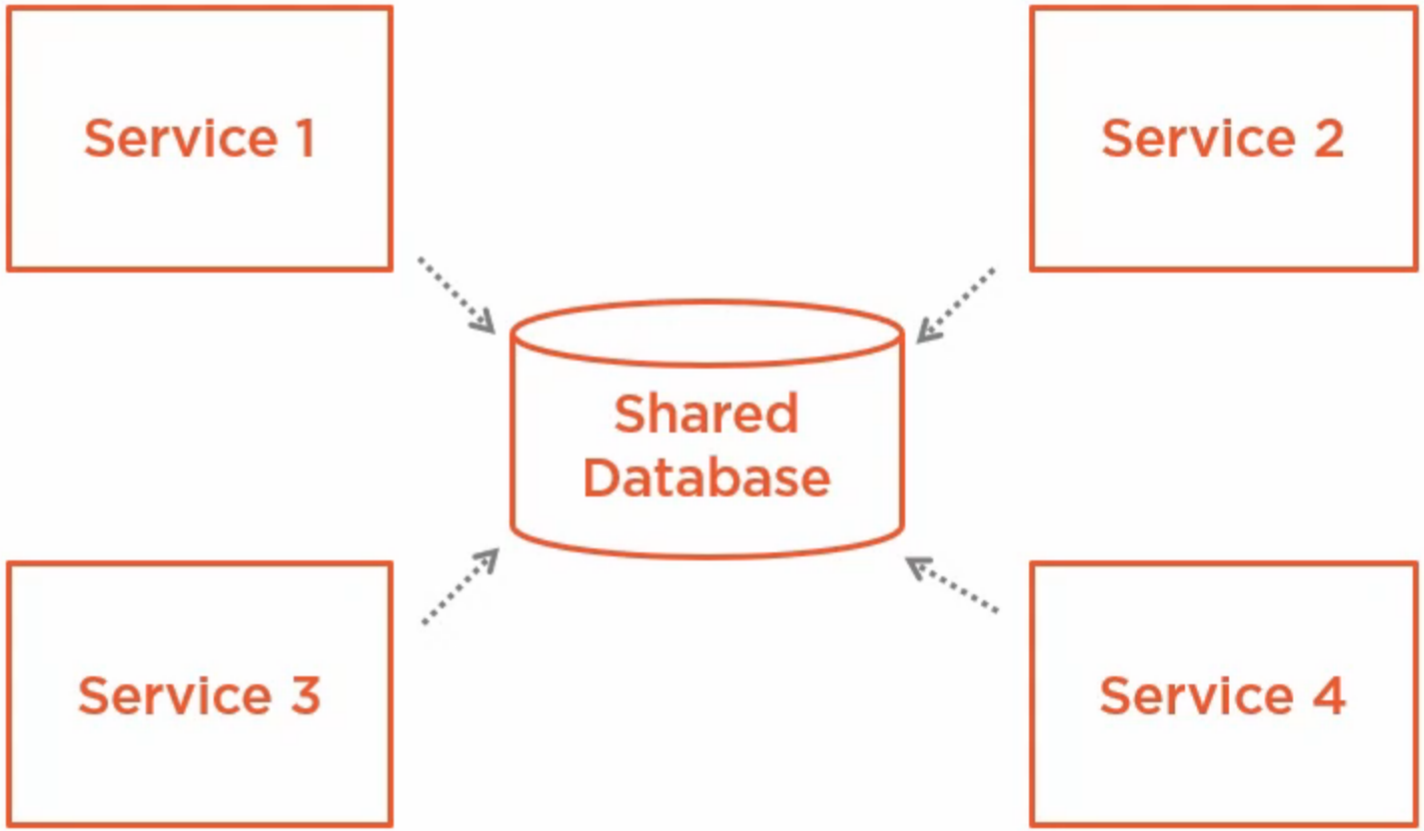
Network communication mechanism

Sync and async com to avoid distributed transactions

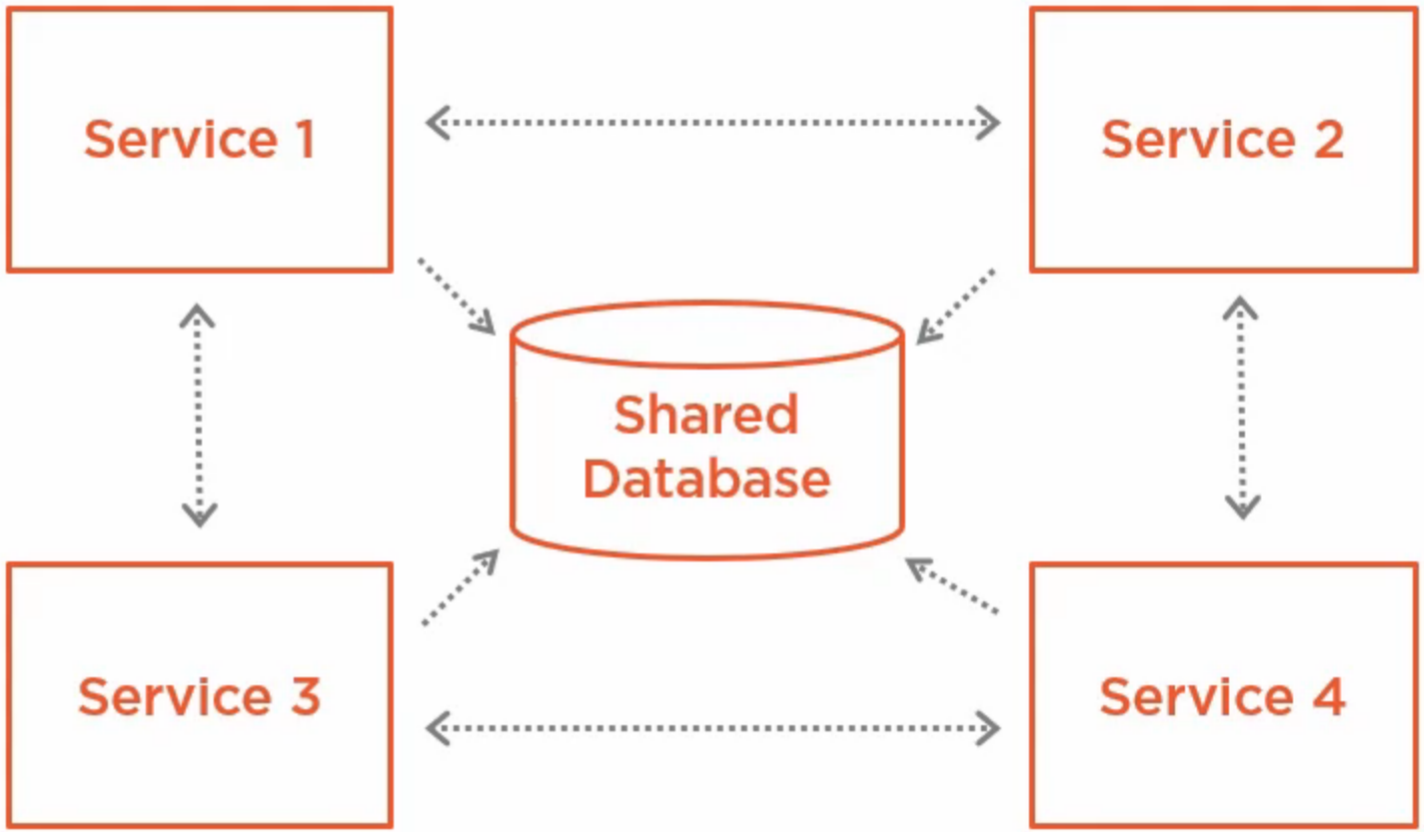
Centralized tooling for management

Distributed Monoliths

Distributed Monoliths

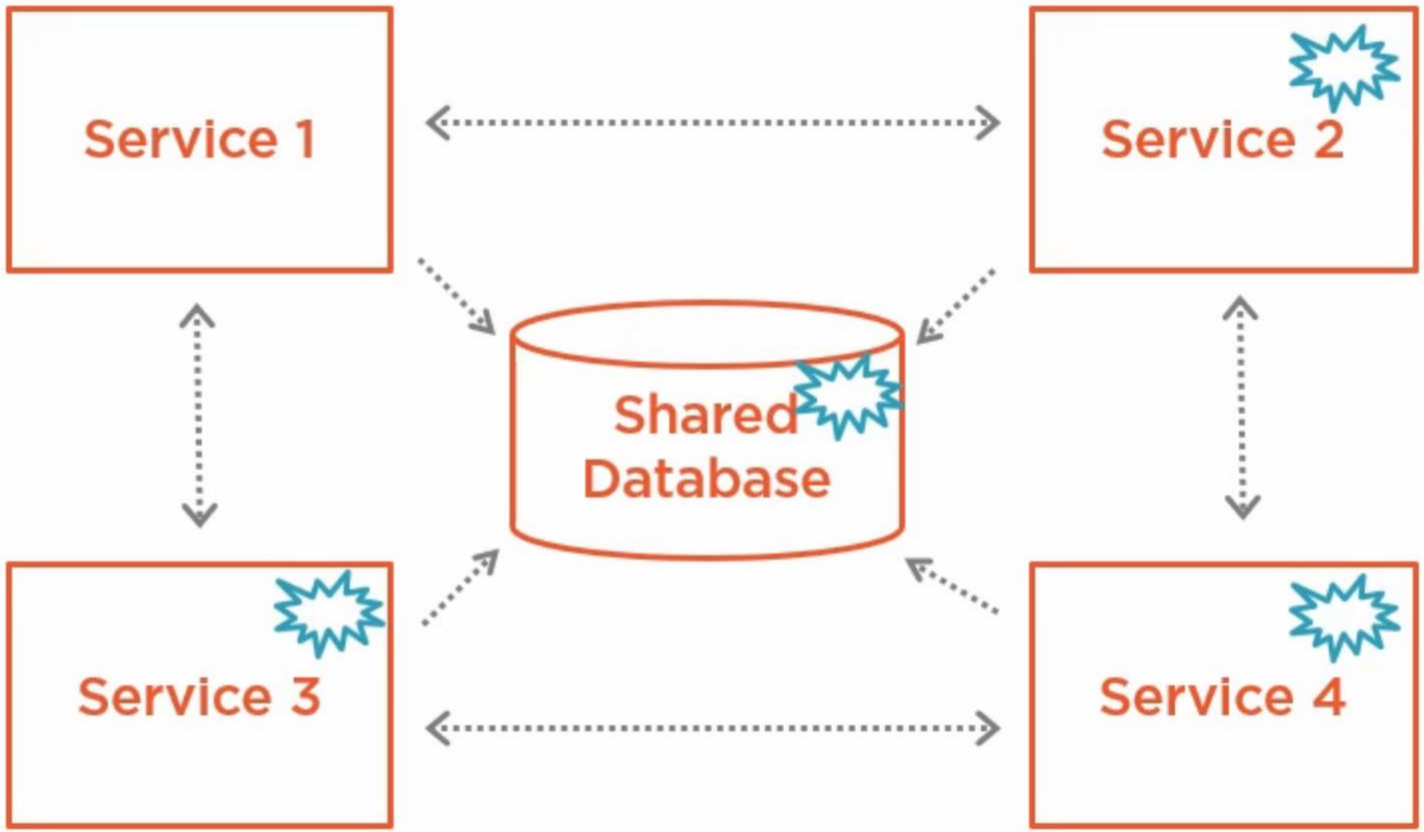


Distributed Monoliths



tight coupling

Distributed Monoliths



tight coupling

Everything must be deployed together

New features require changes everywhere

How can microservices help us?

Benefits of Microservices

Small Services

Can be owned by a team

Easier to understand

Can be rewritten

Benefits of Microservices

Small Services

Can be owned by a team

Easier to understand

Can be rewritten

Technology Choice

Adopt new technology

Use the right tool

Standardize where it makes sense

Benefits of Microservices

Small Services

Can be owned by a team

Easier to understand

Can be rewritten

Technology Choice

Adopt new technology

Use the right tool

Standardize where it makes sense

Individual Deployment

Lower risk

Minimize downtime

Frequent updates

Benefits of Microservices

Small Services

Can be owned by a team

Easier to understand

Can be rewritten

Technology Choice

Adopt new technology

Use the right tool

Standardize where it makes sense

Individual Deployment

Lower risk

Minimize downtime

Frequent updates

Scaling

Scale services individually

Cost-effective

Benefits of Microservices

Small Services

Can be owned by a team

Easier to understand

Can be rewritten

Technology Choice

Adopt new technology

Use the right tool

Standardize where it makes sense

Individual Deployment

Lower risk

Minimize downtime

Frequent updates

Scaling

Scale services individually

Cost-effective

Agility

Adapt rapidly

Easier reuse

What are the downsides of microservices?

Microservices solve some problems, but bring new challenges

Challenges of Microservices

Developer Productivity

How can we make it easy for developers to be productive working on the system?

Challenges of Microservices

Developer Productivity

How can we make it easy for developers to be productive working on the system?

Complex Interactions

Take care to avoid inefficient, chatty communications between microservices

Challenges of Microservices

Developer Productivity

How can we make it easy for developers to be productive working on the system?

Complex Interactions

Take care to avoid inefficient, chatty communications between microservices

Deployment

You will need to automate the process

Challenges of Microservices

Developer Productivity

How can we make it easy for developers to be productive working on the system?

Complex Interactions

Take care to avoid inefficient, chatty communications between microservices

Deployment

You will need to automate the process

Monitoring

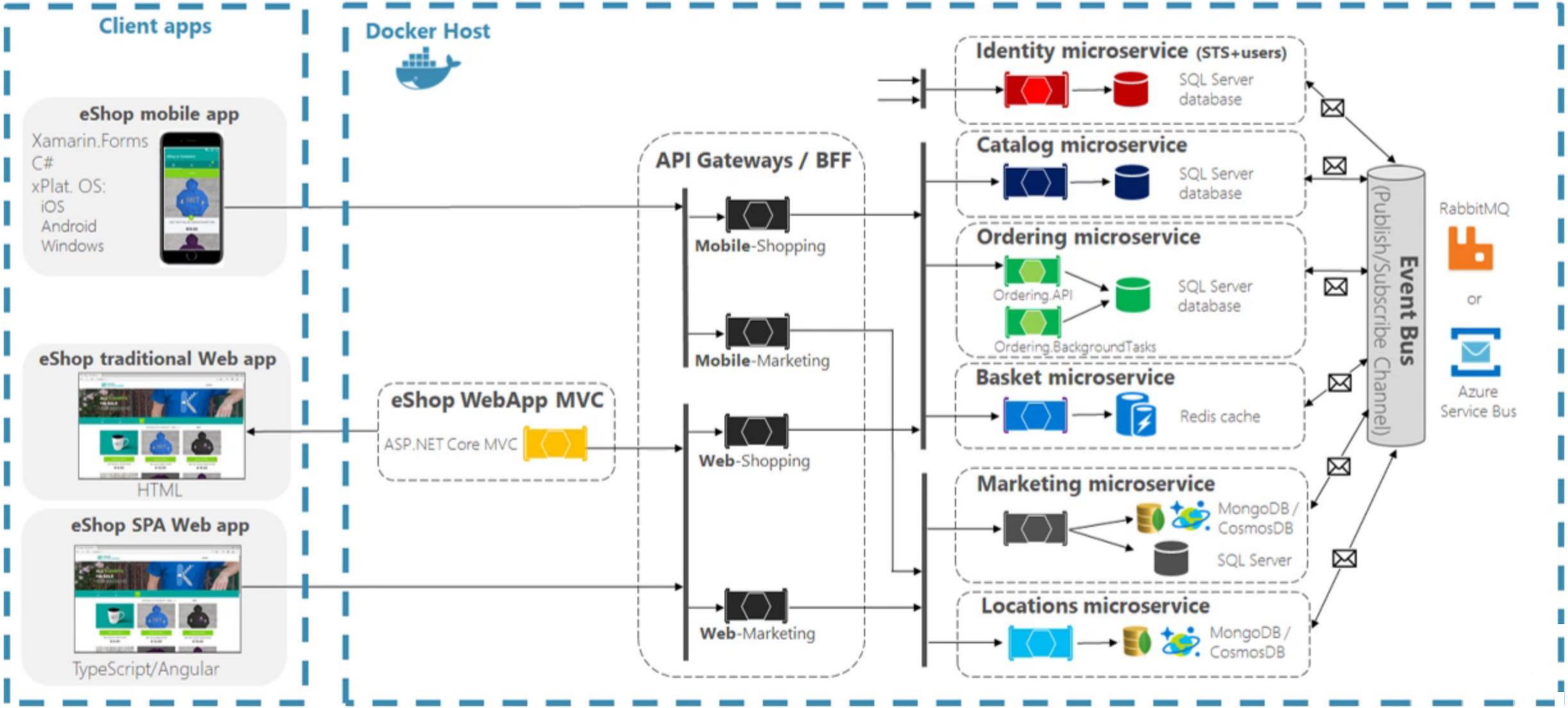
We need a centralized place to check logs and monitor for problems

You can succeed with microservices

eSHOP onCONTAINERS

E-commerce

eShopOnContainers Architecture



Microservices Give You Choice

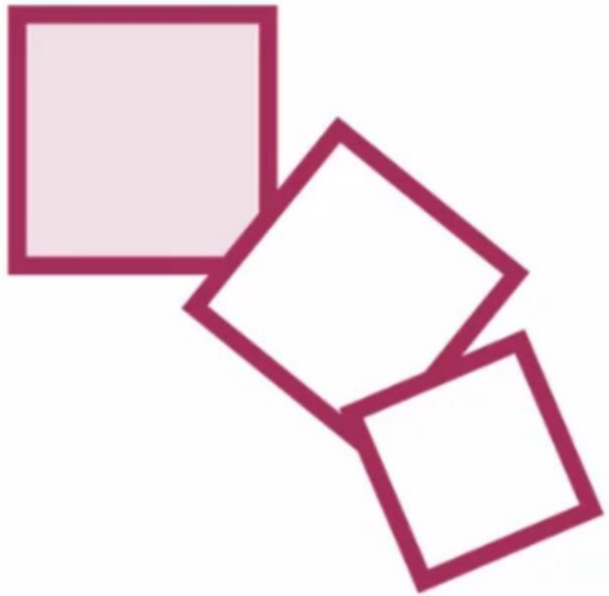
Microservices do not dictate technology choices

Use your favorite programming language and tools

Architecting Microservices

What if I already have an existing (monolithic) application?

Evolving towards Microservices



Augment a monolith

- Add new microservices

Decompose a monolith

- Extract microservices

You don’t need to start with microservices

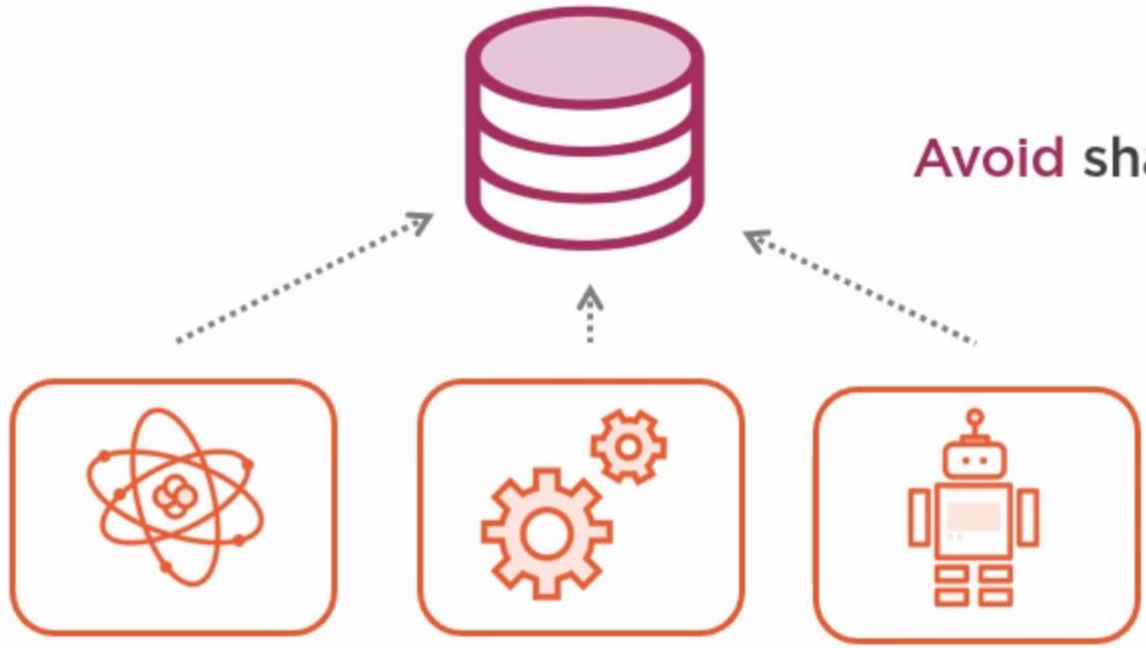
- It’s hard to get service boundaries right

Defining microservice responsibilities

- Public interface

Microservices own their own data

Microservice Data Ownership



Avoid sharing a data store

Microservice Data Ownership

Avoid sharing a data store

Data store per microservice

Limitations:
Database joins
Transactions
Eventual consistency

Mitigating Data Ownership Limitations

Define service boundaries well

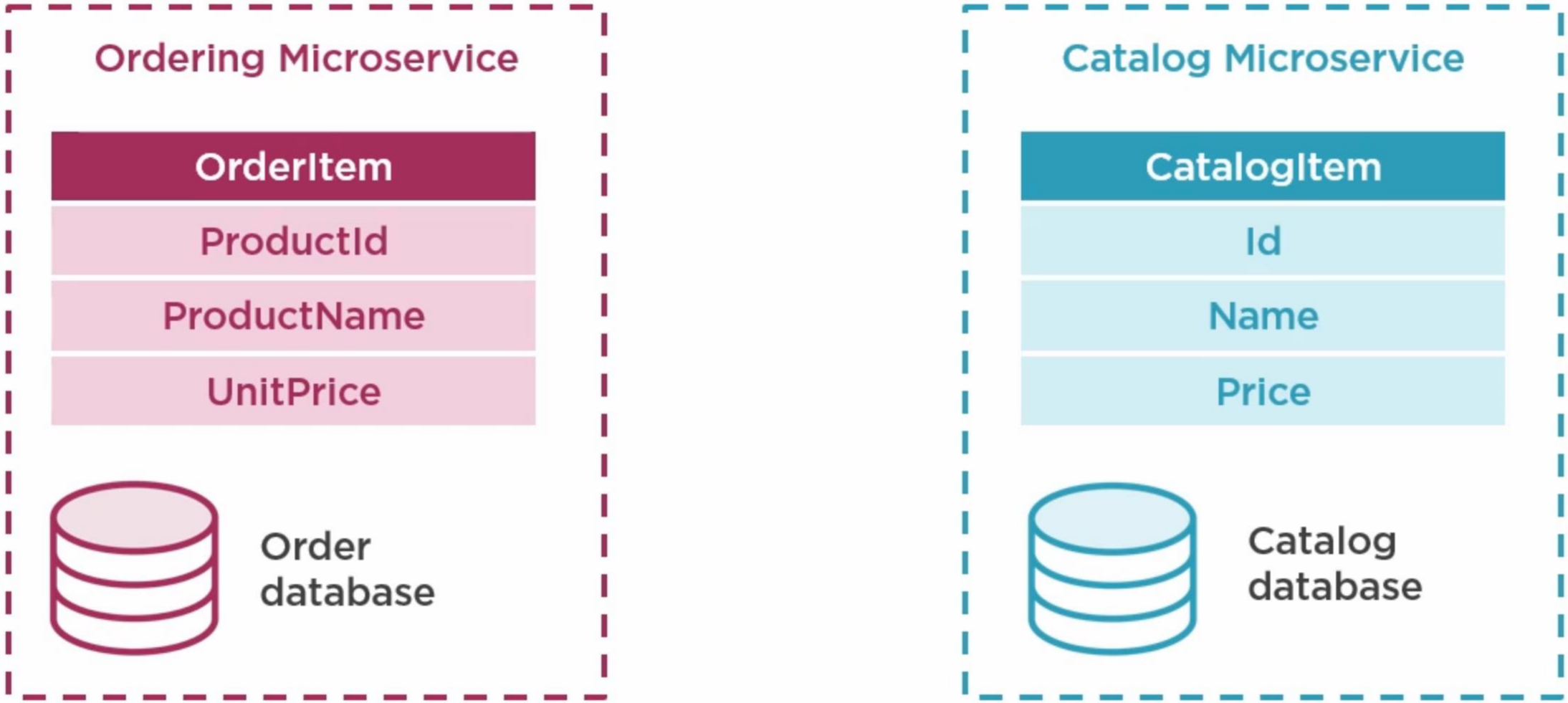
- Minimize the need to aggregate data

Caching

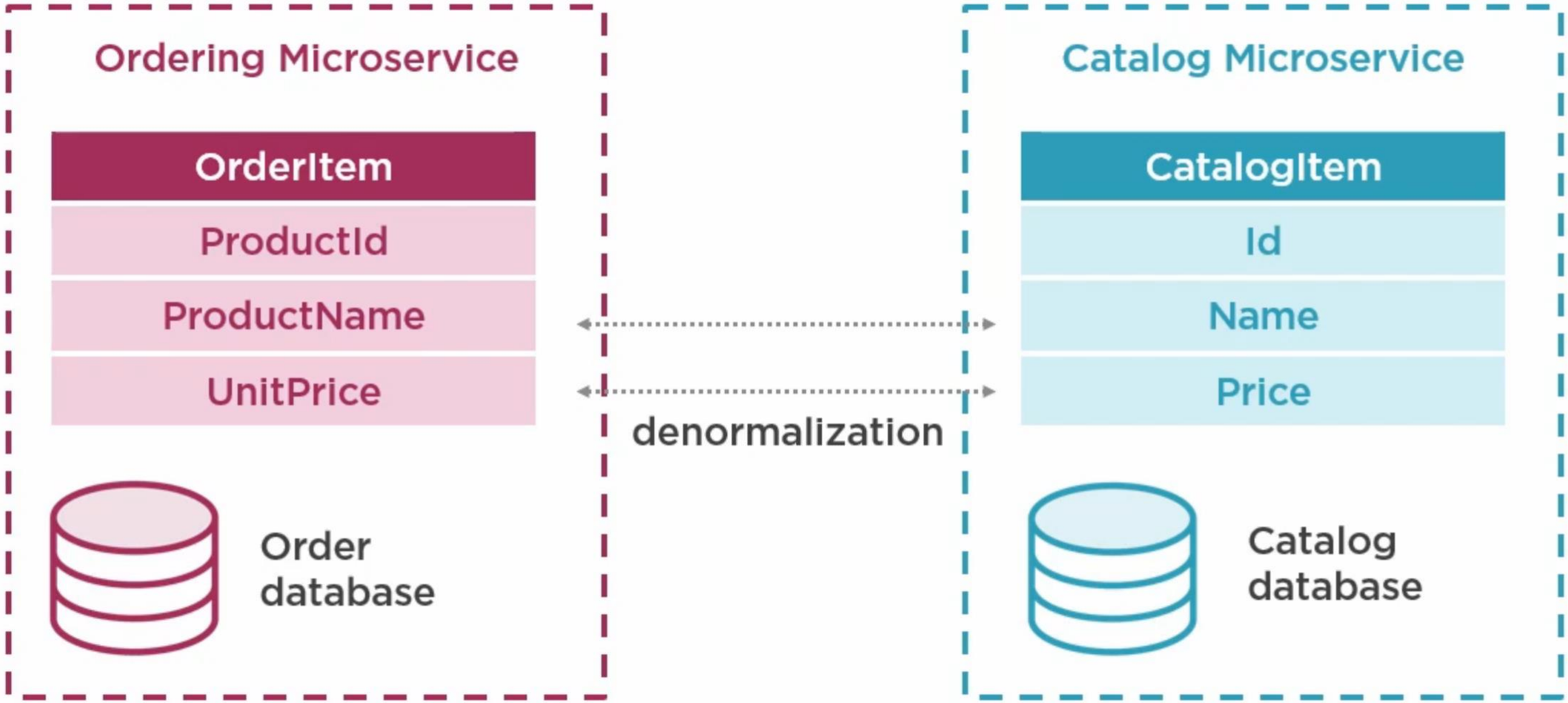
- Improved performance
- Improved availability

Identify “seams” in the database

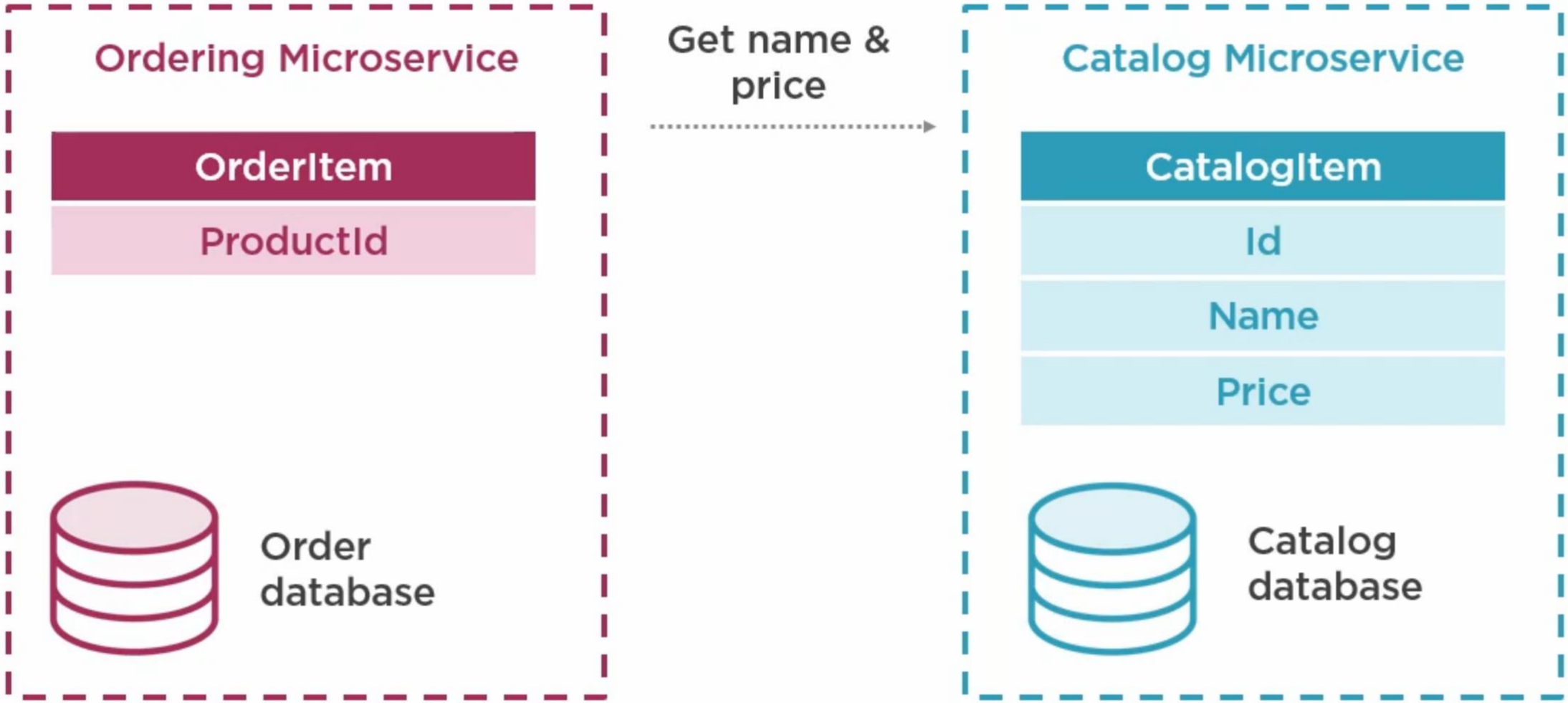
Duplicating Data



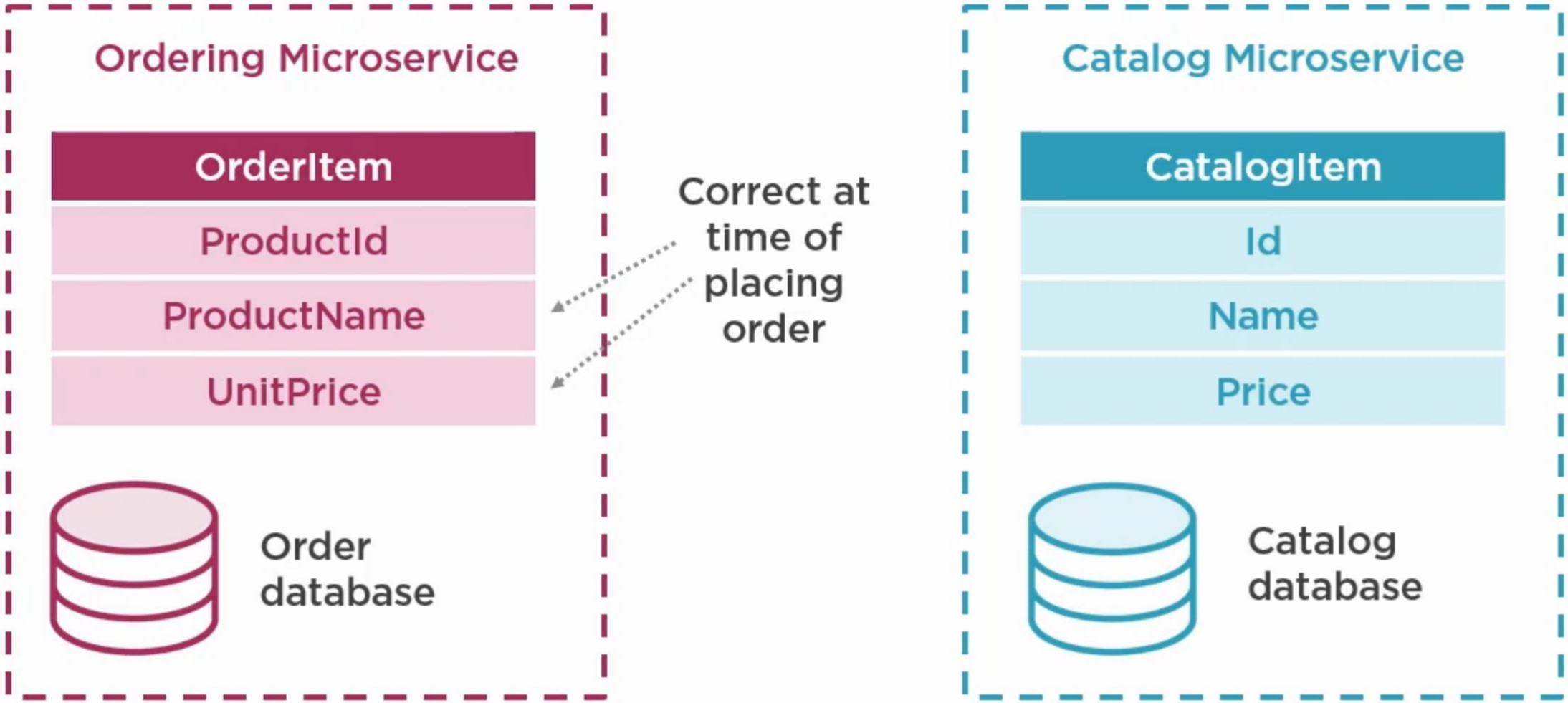
Duplicating Data



Duplicating Data

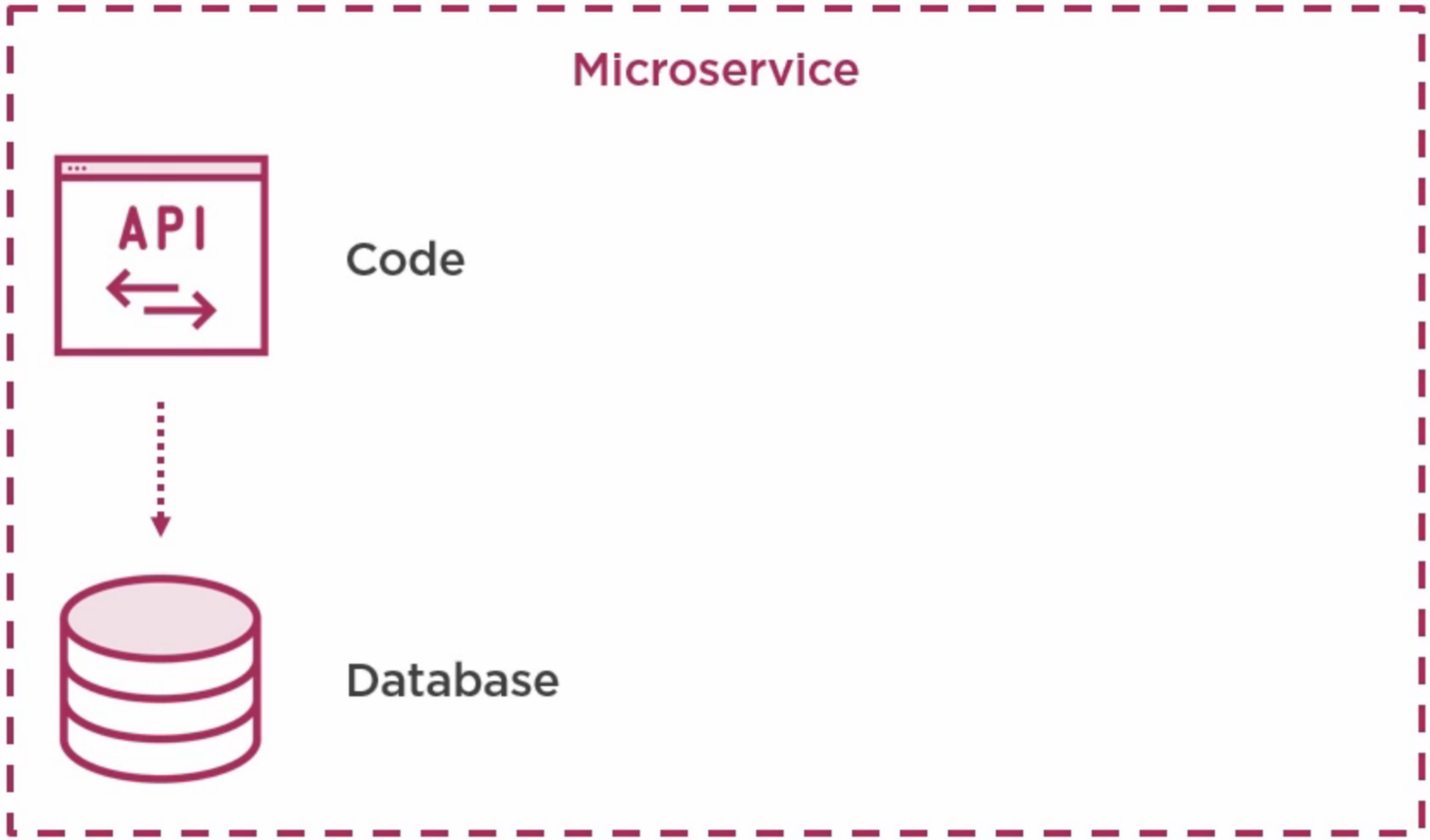


Duplicating Data

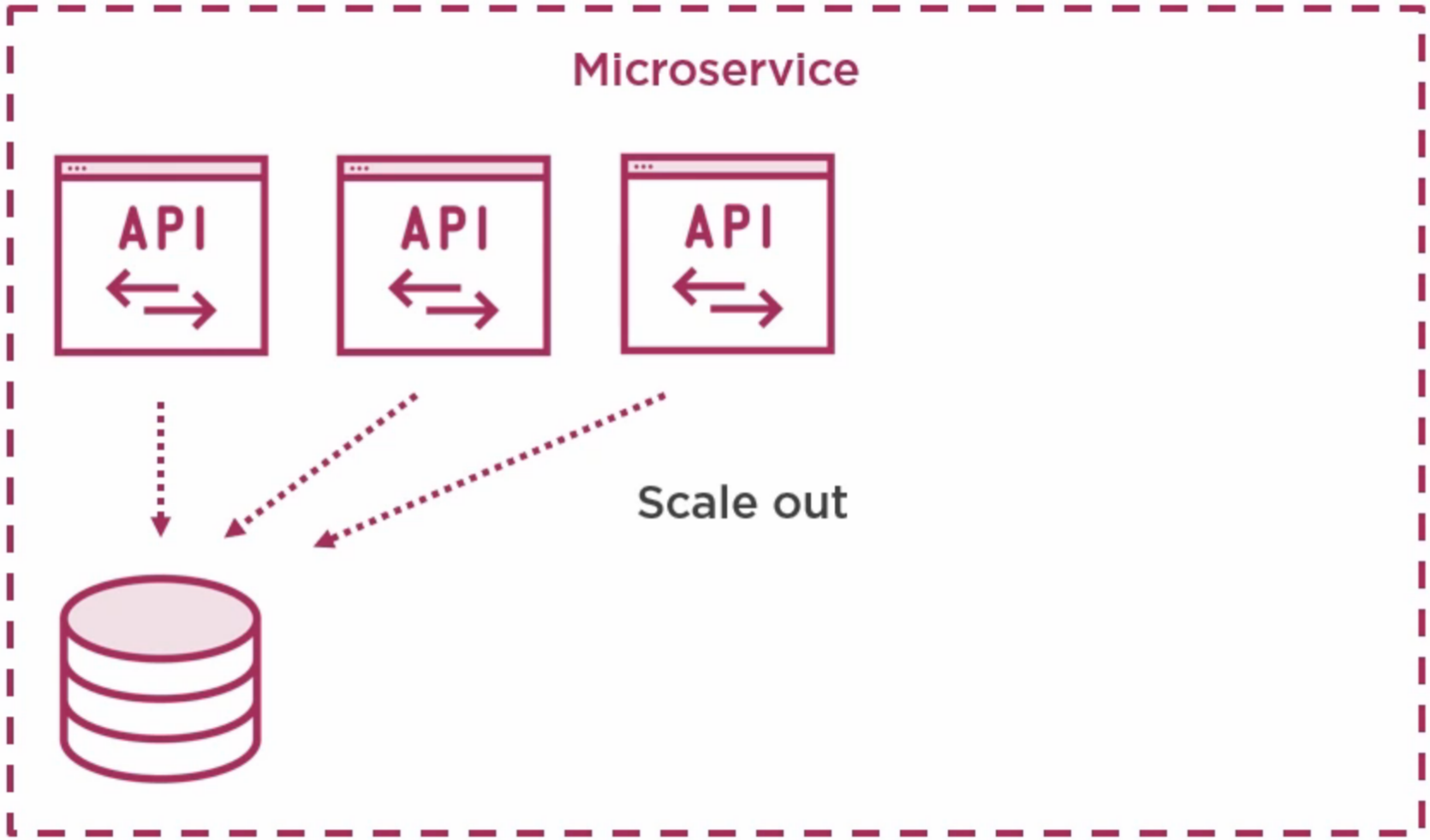


Microservices can consist of more than one process

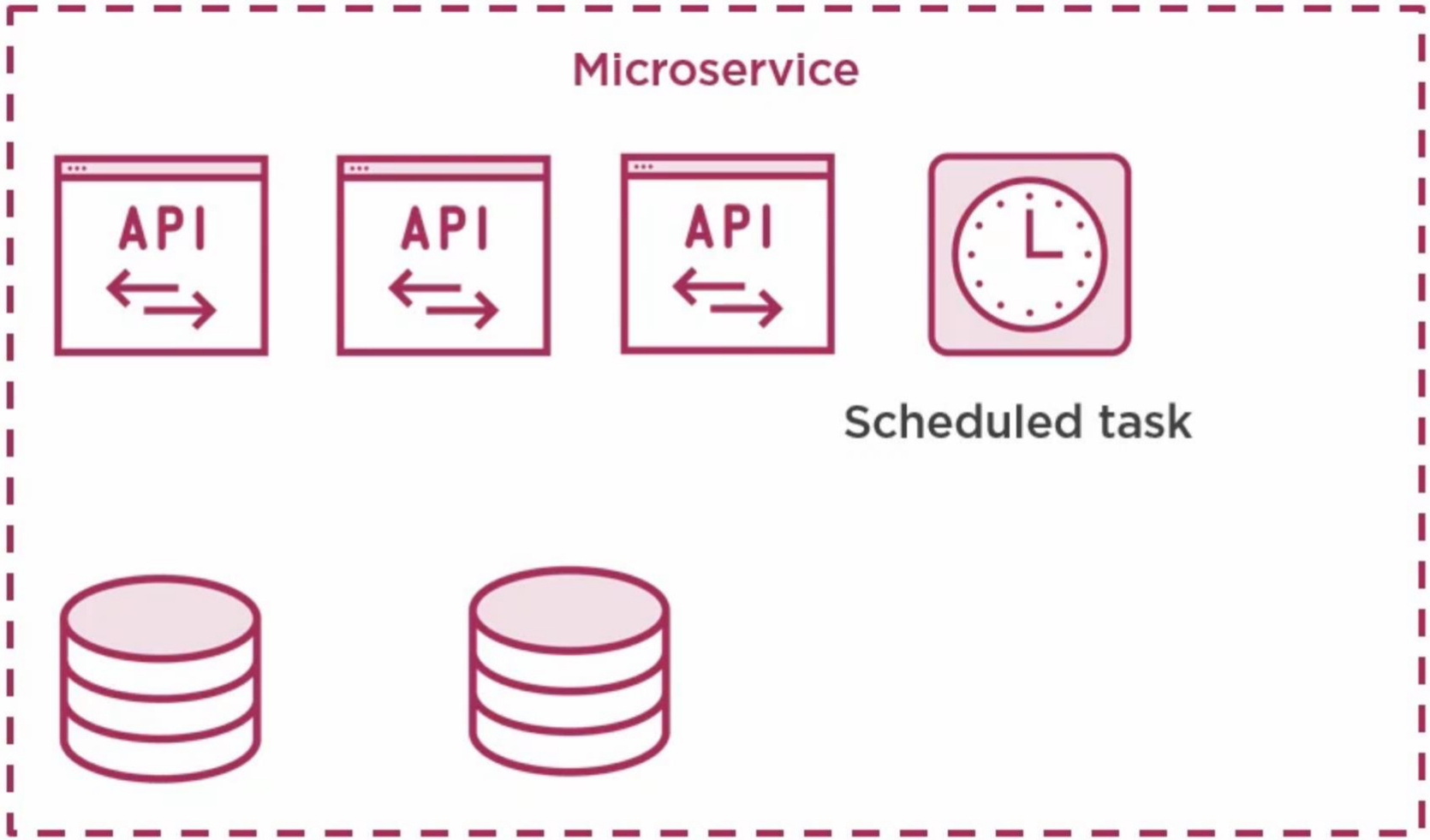
Microservice Components



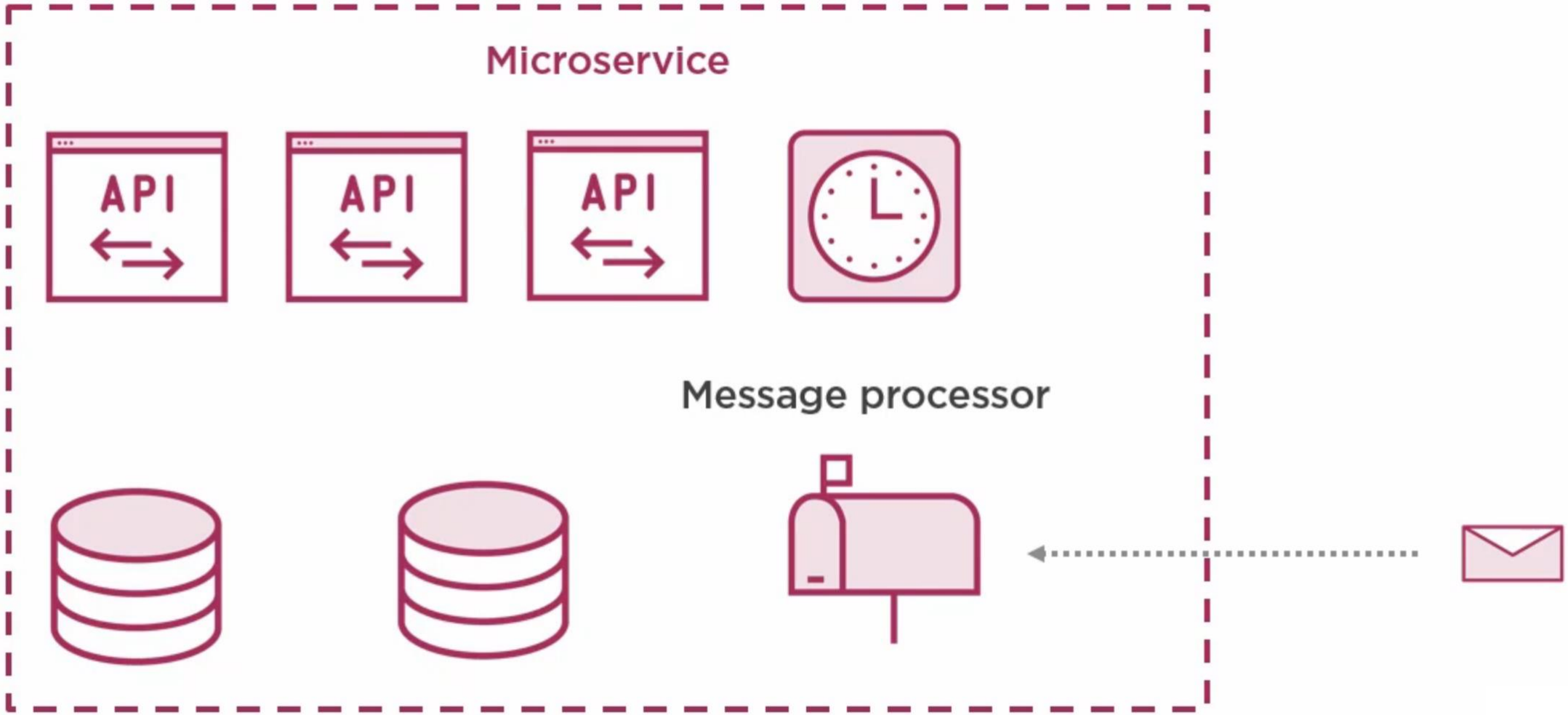
Microservice Components



Microservice Components



Microservice Components



Microservices should have a clearly defined public interface

Microservices can be upgraded without their clients needing to upgrade

Microservice Contracts

Make additive changes

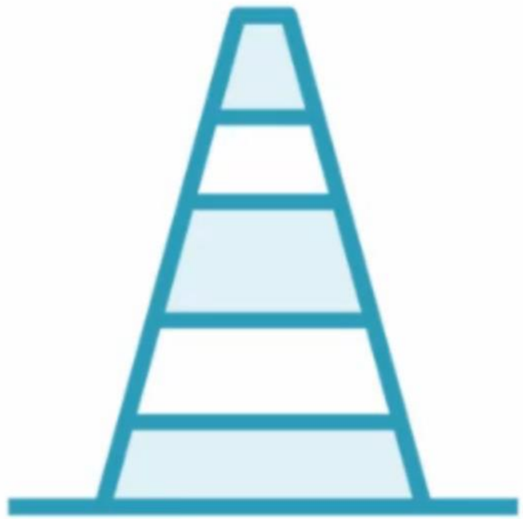
- New endpoints
- New properties on DTOs

Introduce version 2 API

- Version 1 clients must still be supported

Easily forgotten in development

Avoiding Upgrade Issues



Team ownership of microservices

- First, add a new capability
- Then, deploy updated microservice
- Later, update clients

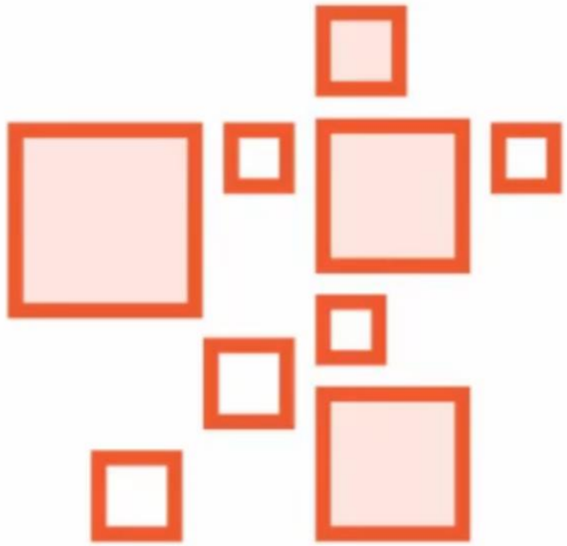
Create automated tests

- Ensure that older clients are supported
- Run as part of a CI build process

Beware of shared code

- Can result in tight coupling

Identifying Microservice Boundaries



Getting it wrong can be costly

- Poor performance
- Hard to change

Start from an existing system

- Identify loosely coupled components
- Identify database seams

Organize microservices around business capabilities

Domain Driven Design

Identify "bounded contexts"

- Break up large domains
- "Ubiquitous language"
- Microservices do not share models
- e.g. OrderItem and CatalogItem

Sketch your ideas on a whiteboard

- Run them through real-world use cases
- Identify potential problems

Microservice Boundary Pitfalls

Don't turn every noun into a microservice

- Anemic CRUD microservices
- Thin wrappers around databases
- Logic distributed elsewhere

Avoid circular dependencies

Avoid chatty communications

eShopOnContainers Service Boundaries

Catalog

Browsing for products

Large dataset

Support flexible queries

Non-sensitive data

Basket

Preparing for purchase

Short-lived data

Redis cache

Ordering

Processing orders

Mostly writes new data

Reliability is vital

Highly sensitive data

Identity

Authentication

Marketing

Email campaigns

Location

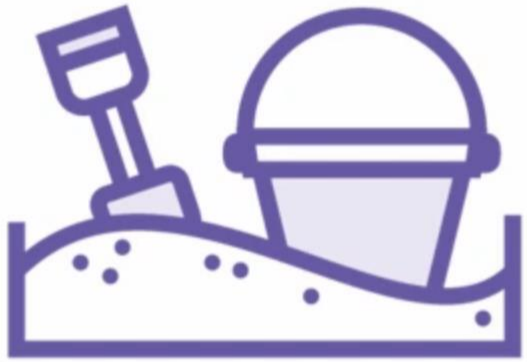
Customer locations

How would you break your application into microservices?

Microservice Hosting Environments

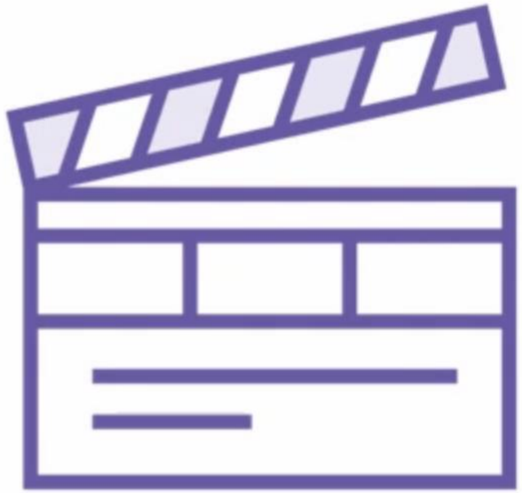
Development

Developers want to debug the application locally



Staging

Testers want to try out the application in a production-like environment



Production

We also need to host the application for end users

Microservices Hosting Options

Virtual Machines	Platform as a Service	Containers
VM per microservice?	Automatic scale-out	Portable: run anywhere
Operational challenges	DNS addresses	Easily run locally
Service discovery	Load balancing	Docker Compose
	Security	
	Monitoring	
	Serverless	

Microservices Security

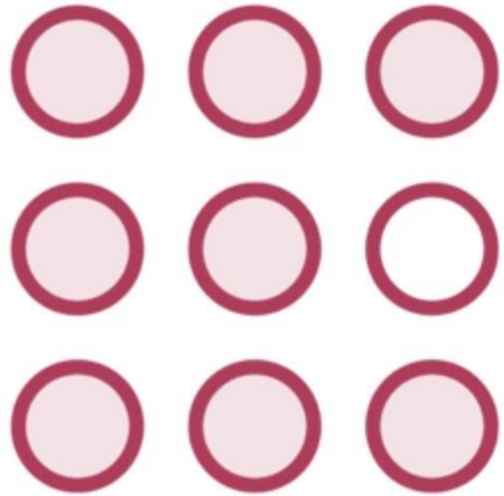
Microservices

International Data Corporation (IDC) predicts that by:

90%

Of all applications will feature microservices architectures that improve the ability to design, debug, update, and leverage third-party code.

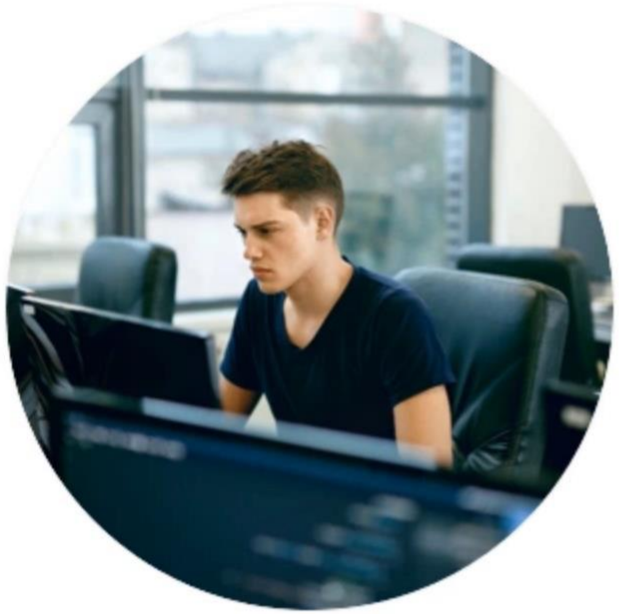
Flexibility a Microservices Architecture



Polyglot

Each service can implement its own technology stack


Microservices the promised land



How do you secure your Microservices without ?

Bugs in Microservices

Fix, test and deploy the offending microservice.

Fail Fast

Fail Early

Fail Often

Consequences of Security Breaches

Reputational and brand damage

Legal issues

Loss of trust

Bankruptcy

Financial loss

Negative headlines

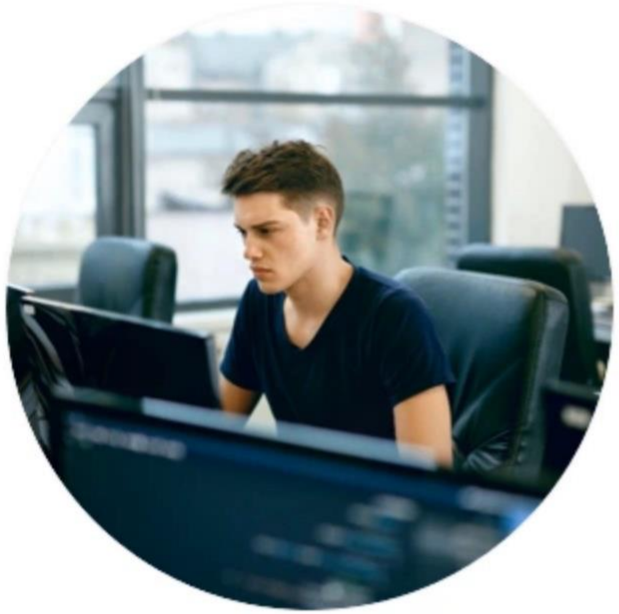
There are also tried and tested best practices and architectural patterns you can use to solve the security challenges within your Microservices architecture.

DevOps: Security is now everyone's responsibility.....

Your Security Implementation Should Not Be

Draconian

Excessively harsh, severe and lock everything down



How do you secure your Microservices without ?

Stifling team productivity.

Reduce the performance or time to market of the application.

Negating any of the benefits a microservices architecture.

Security Fundamentals and Prevention

- the various techniques and patterns you can use secure your microservices architecture.

Hackers Are Lazy

Detection

Identifying security vulnerabilities throughout the development lifestyle.

Monitoring and identifying security breaches.

Reacting to security breaches.

Engrain a Security Culture within Your Development Teams

Threat Modelling

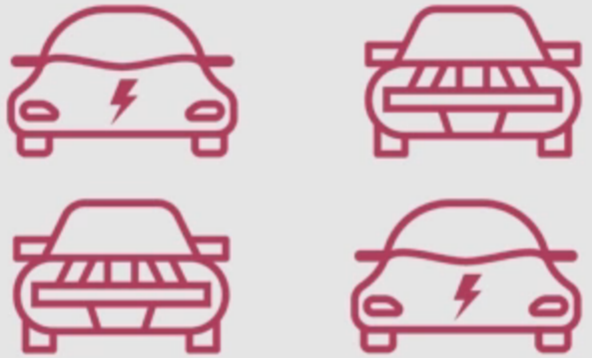
Prioritize security vulnerabilities

Defence in Depth

Is an information assurance concept in which multiple layers of security controls (defence) are placed throughout an information technology (IT) system.

Also known as a castle approach.

Monolith



Microservices

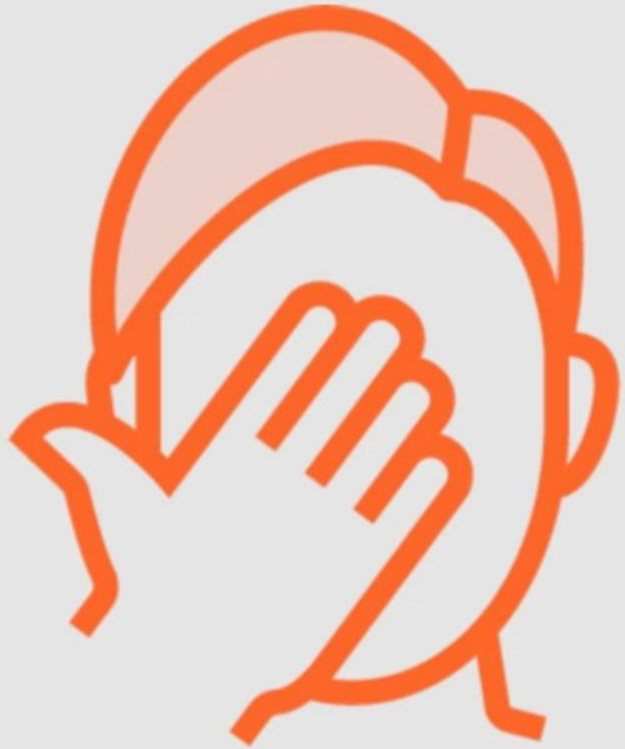
Tightly Coupled

Difficult to isolate services for independent scaling

Vertically scaled

Vulnerable to entanglement

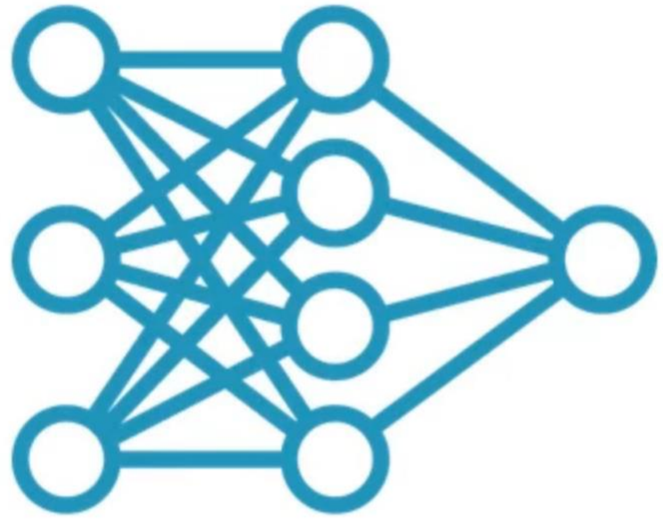
Monolith



Monolith

Microservices

Contrast Security Challenges



Monolith

Microservices

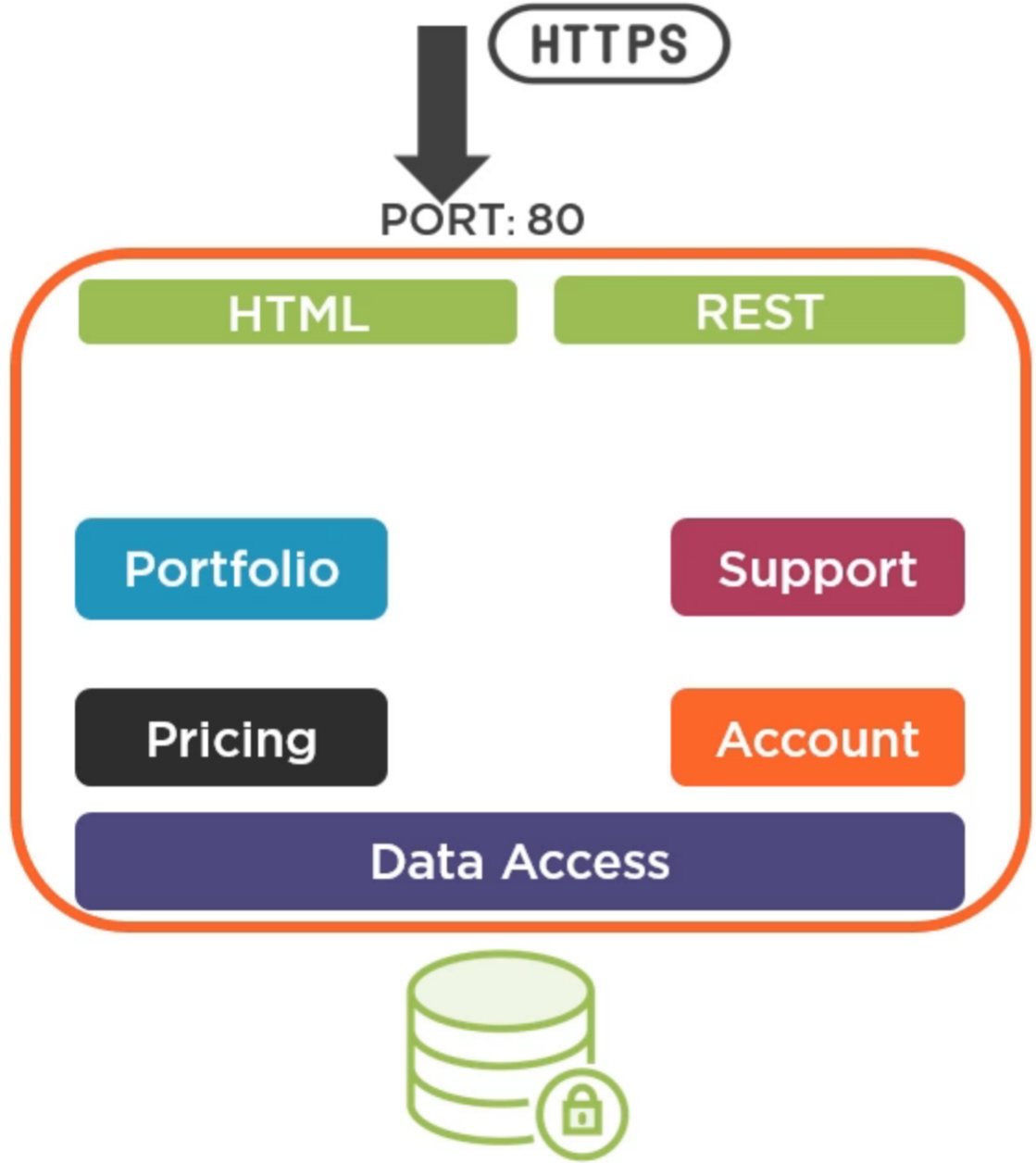
Monolith

Microservices

Monolith

Monolith

Smaller attack surface.

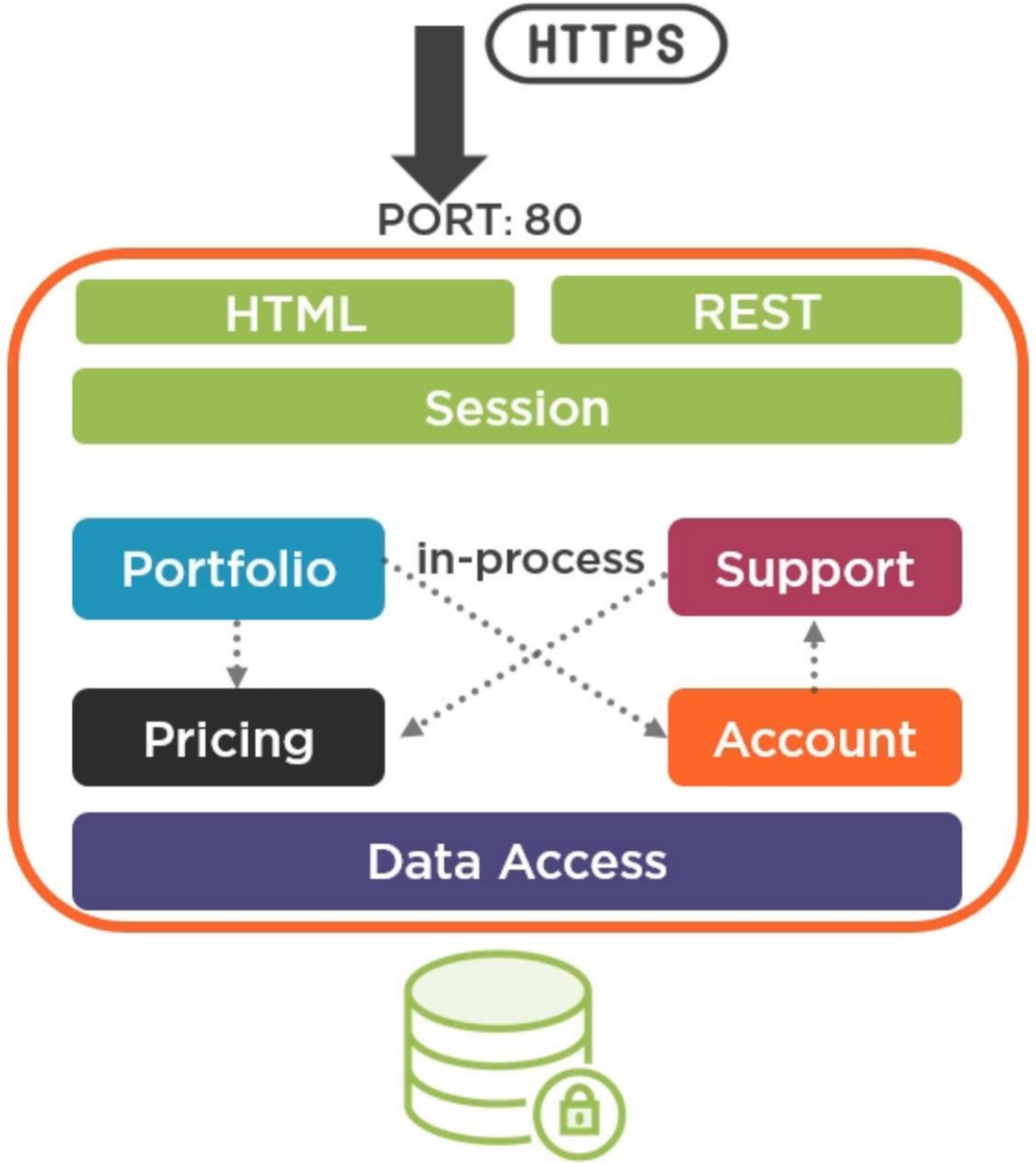


Monolith

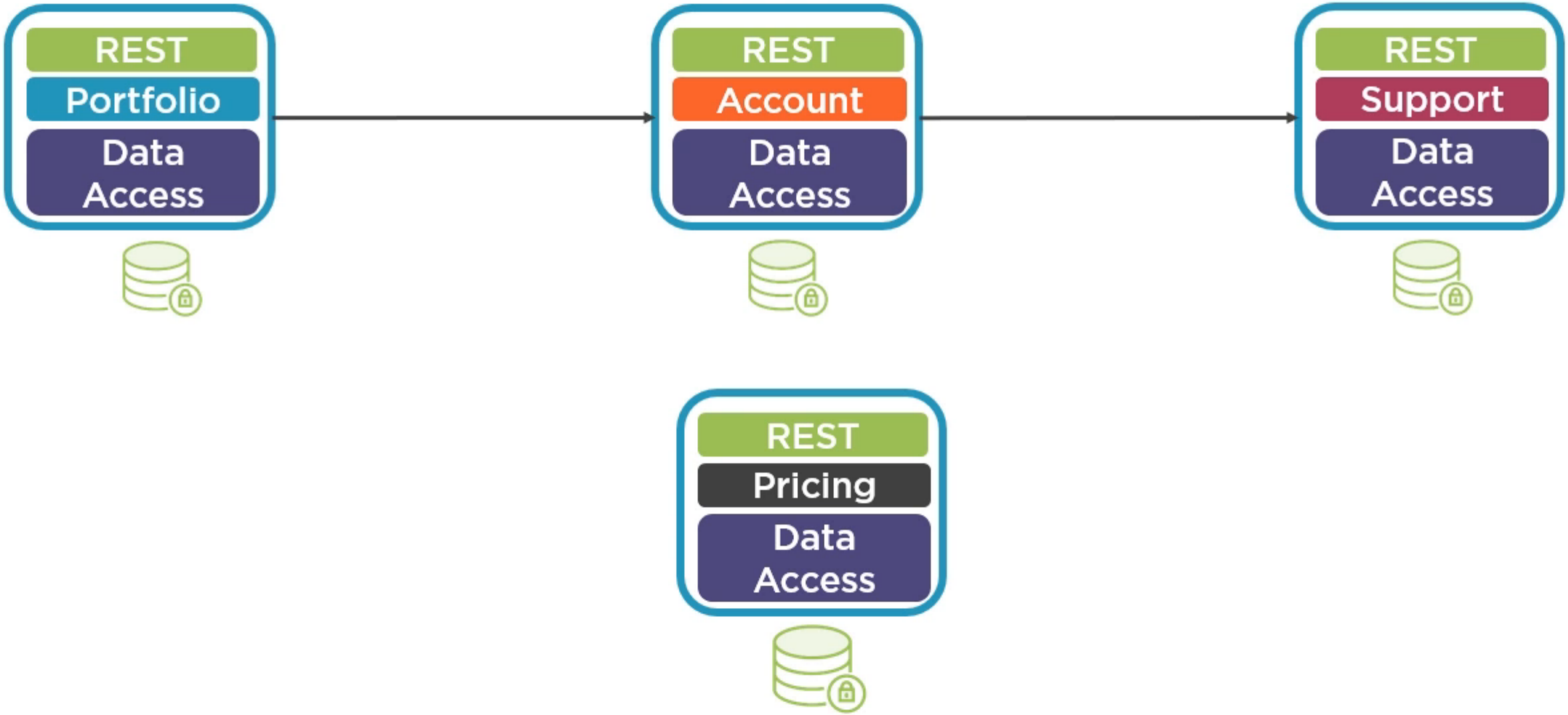
Smaller attack surface.

In-process communication between components is more secure.

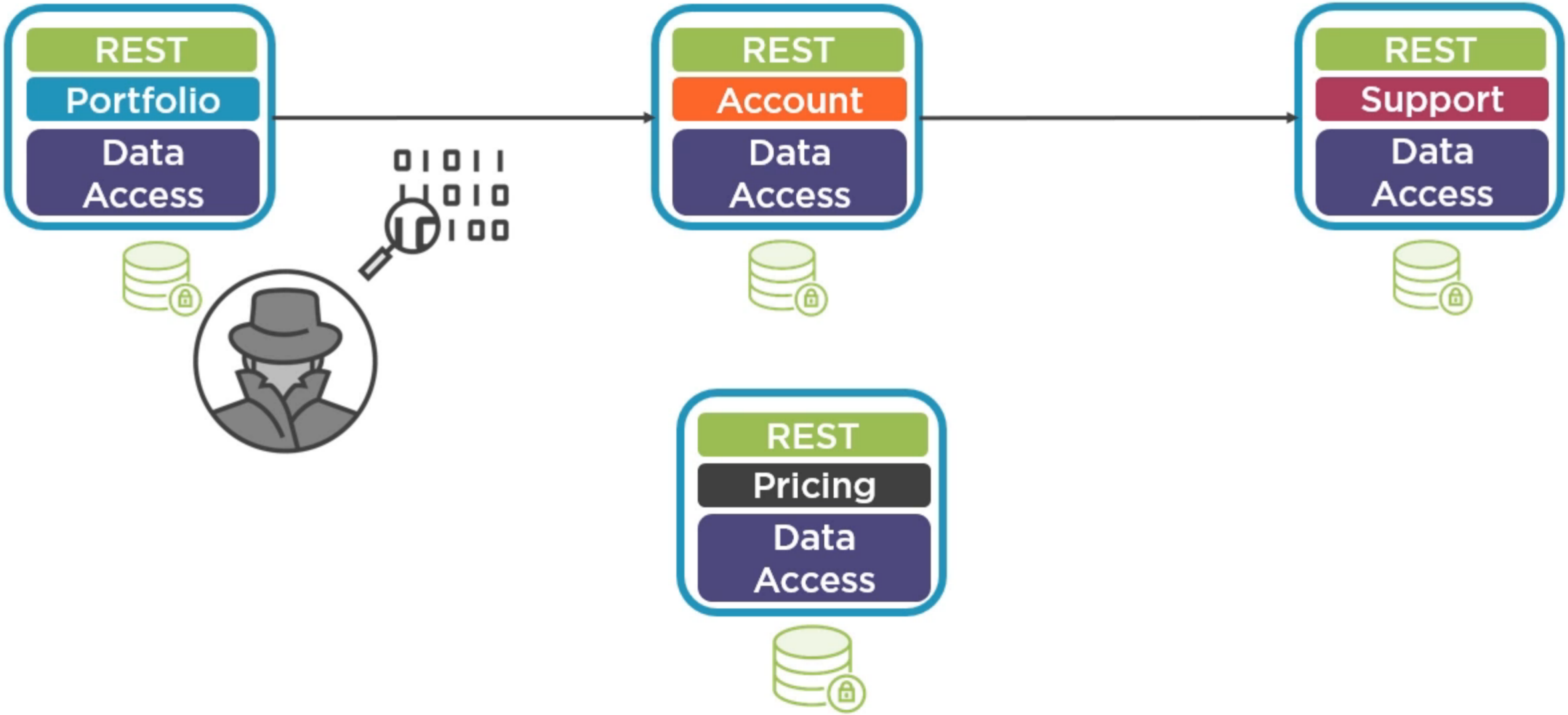
User context is stored centrally, easily retrievable and trusted.



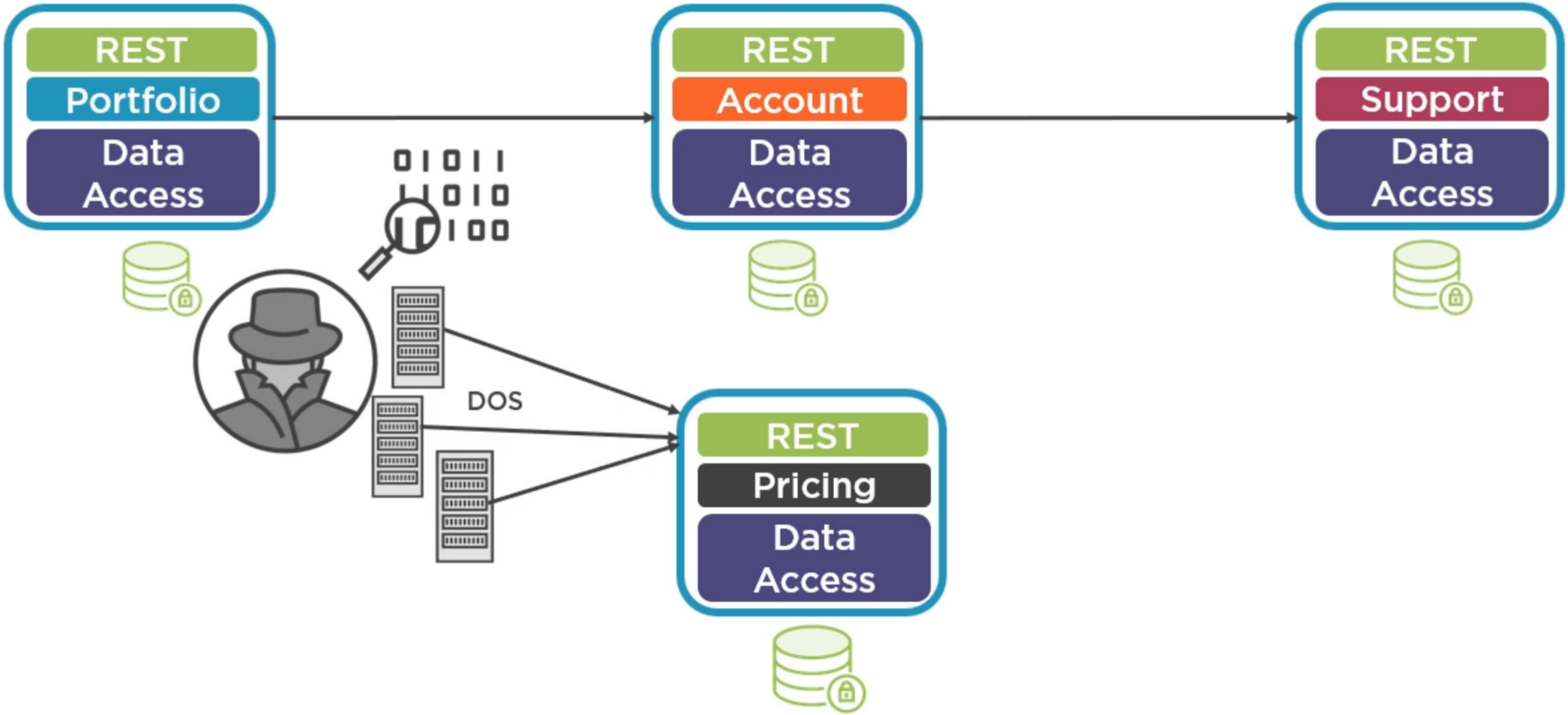
Microservices



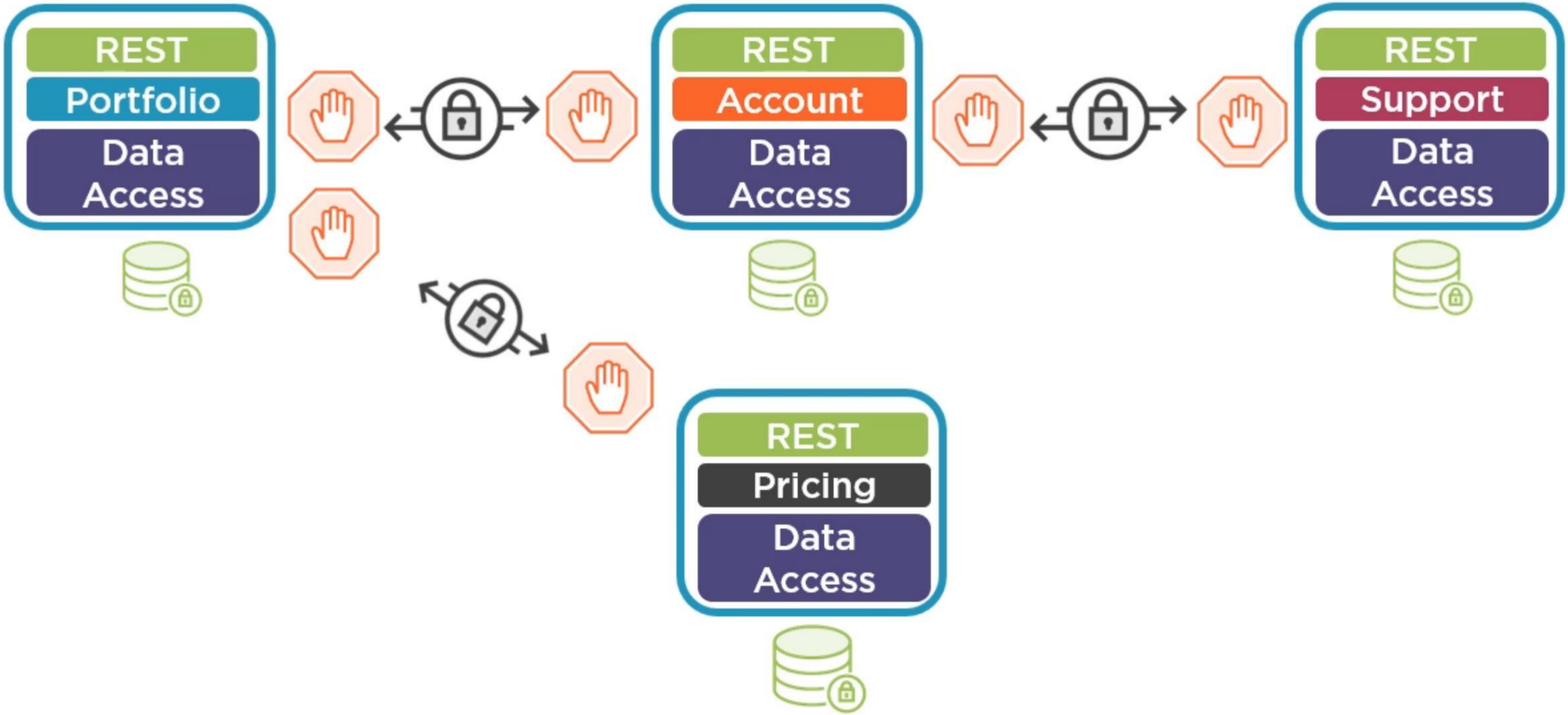
Microservices



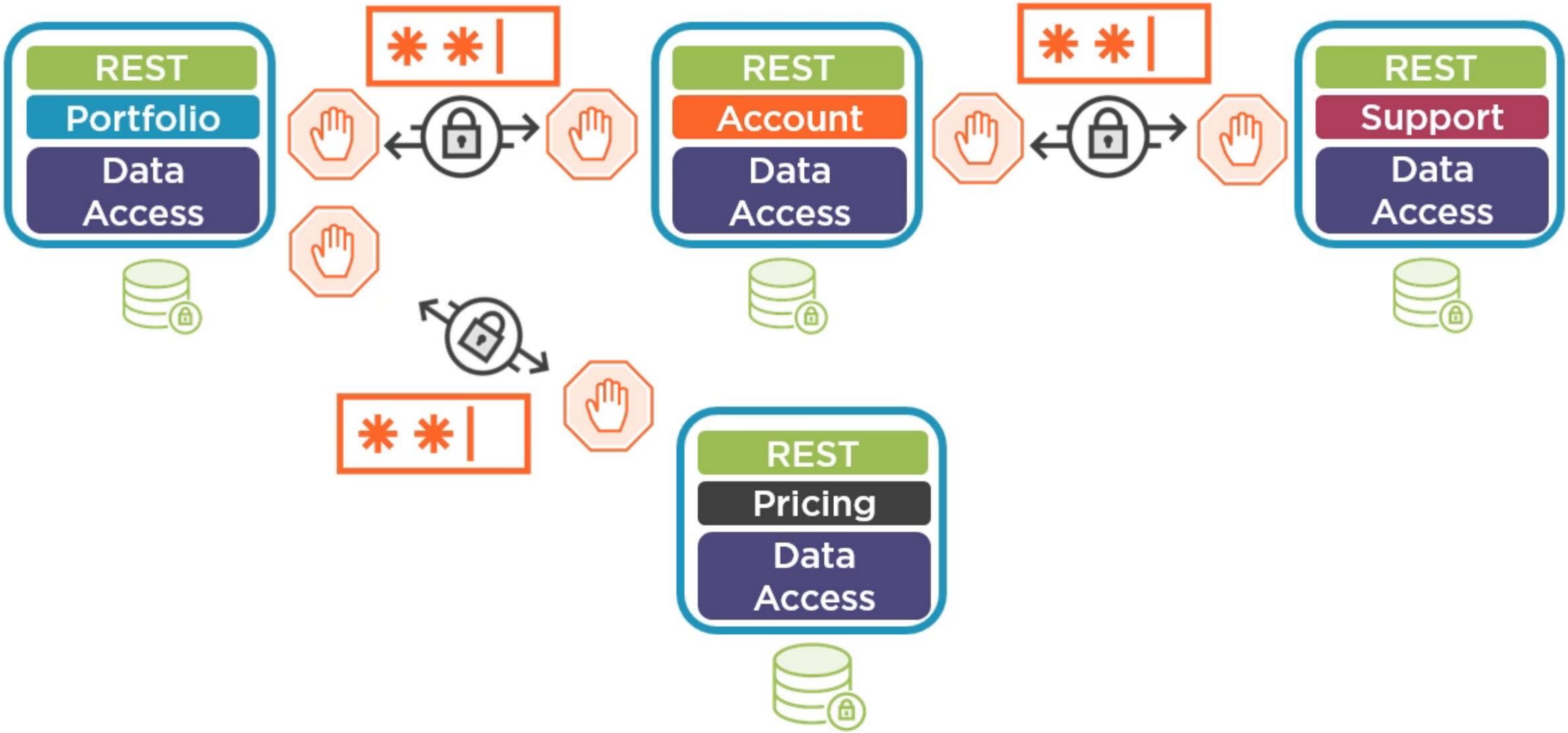
Microservices

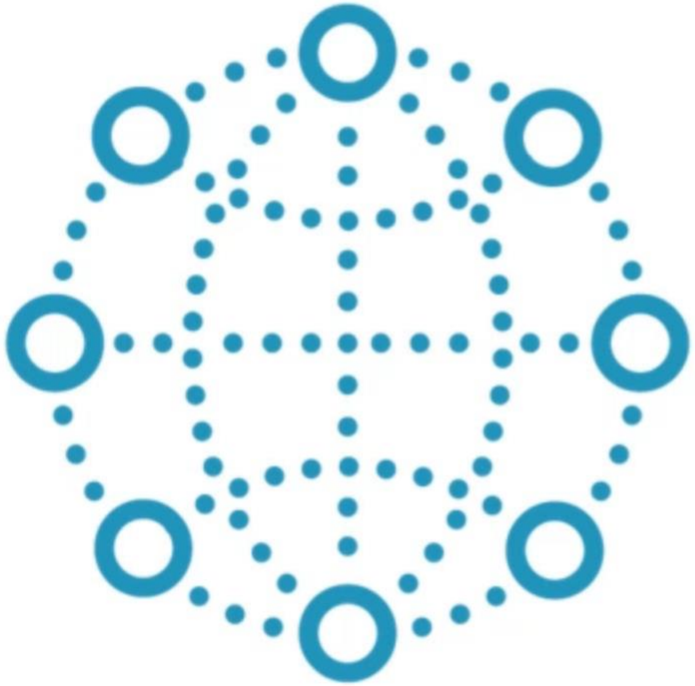


Microservices



Microservices





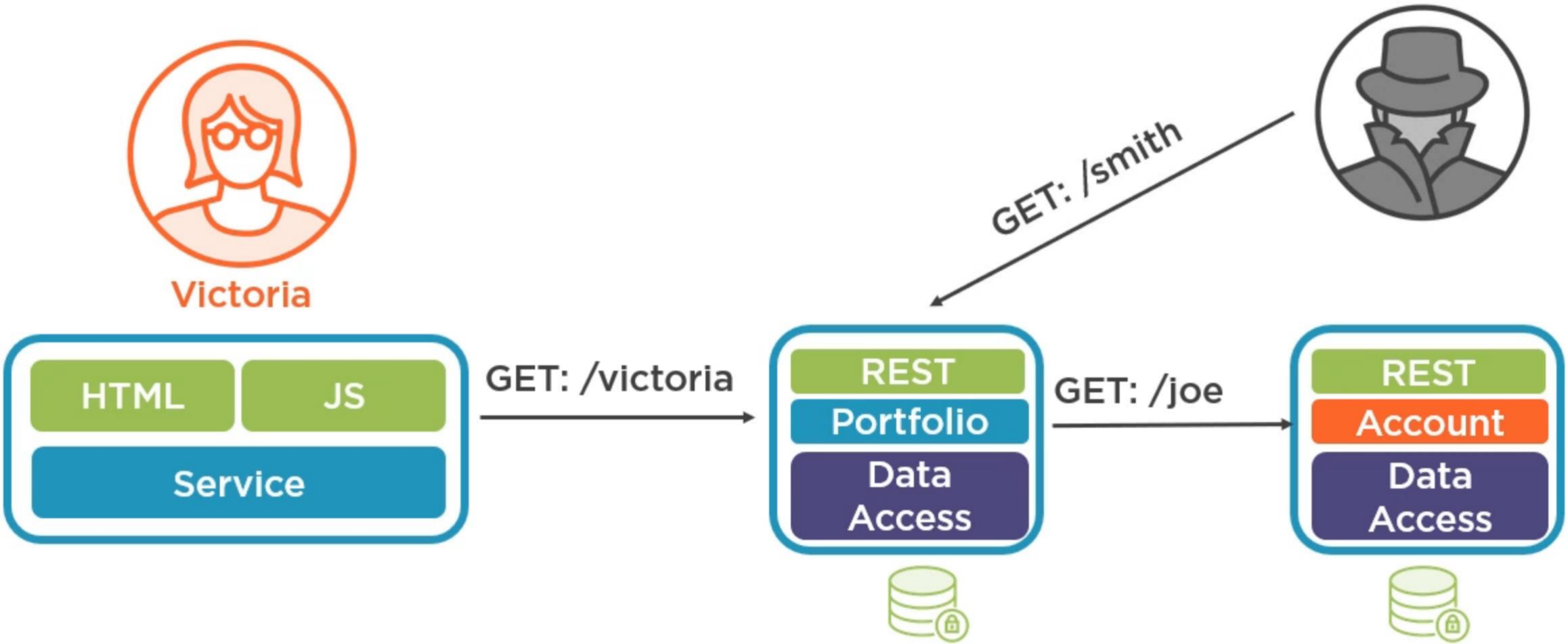
Microservices trade-offs

- Remote calls are slower vs in-process function calls.
- Communication between each service may require encryption and authentication checks.

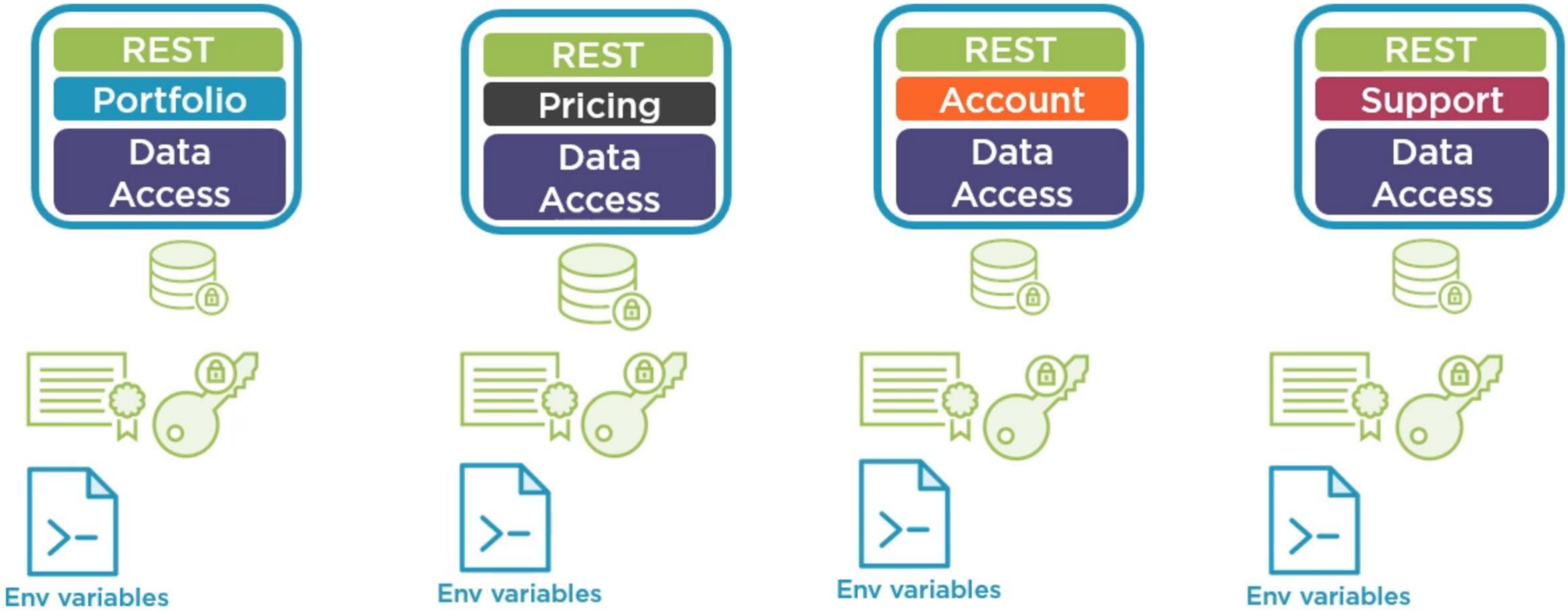
Confused Deputy

Victoria

Confused Deputy



Bootstrapping Secrets



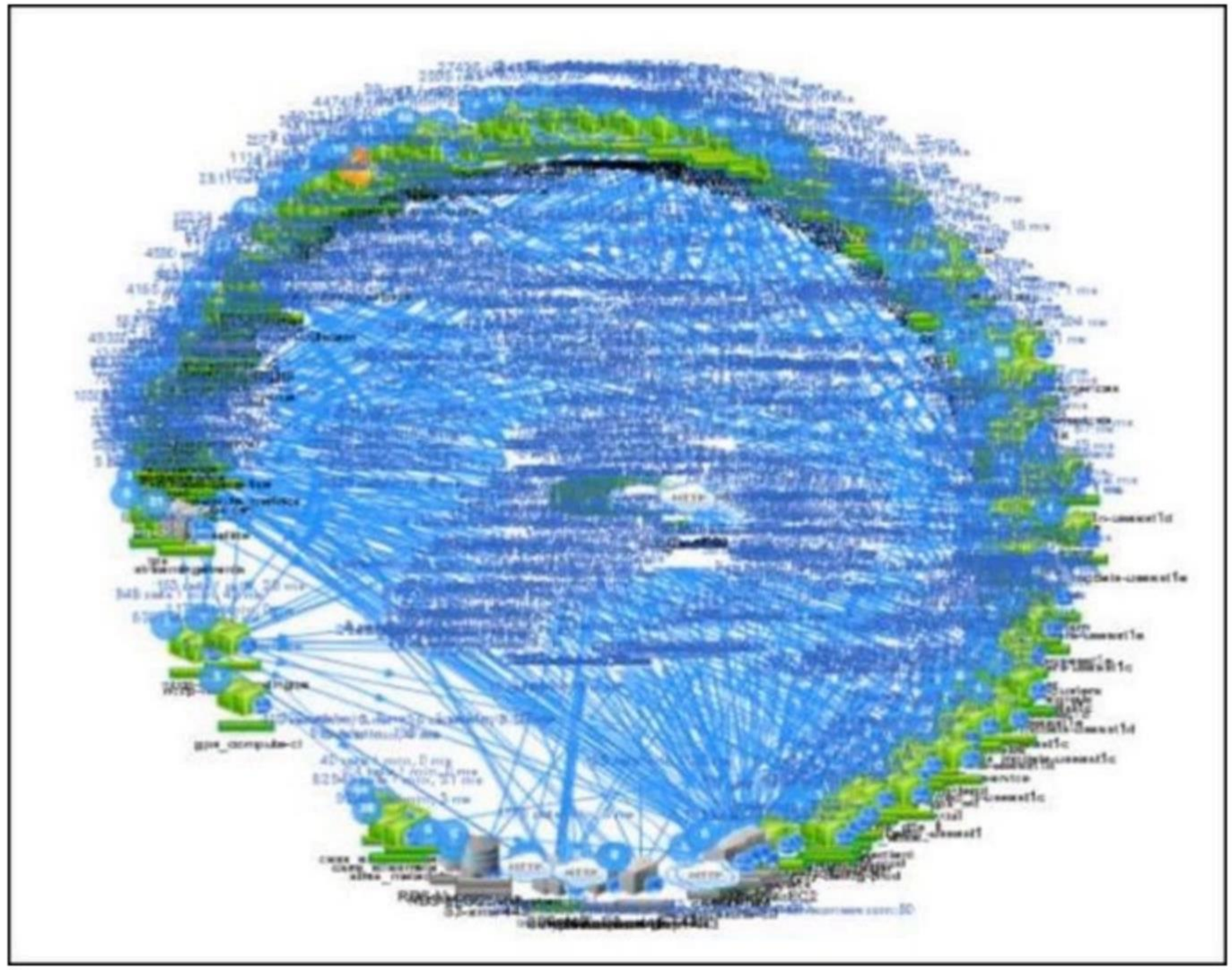
Secret Sprawl

Challenges with immutable servers

- Secrets and whitelists cannot be maintained on the servers file system.

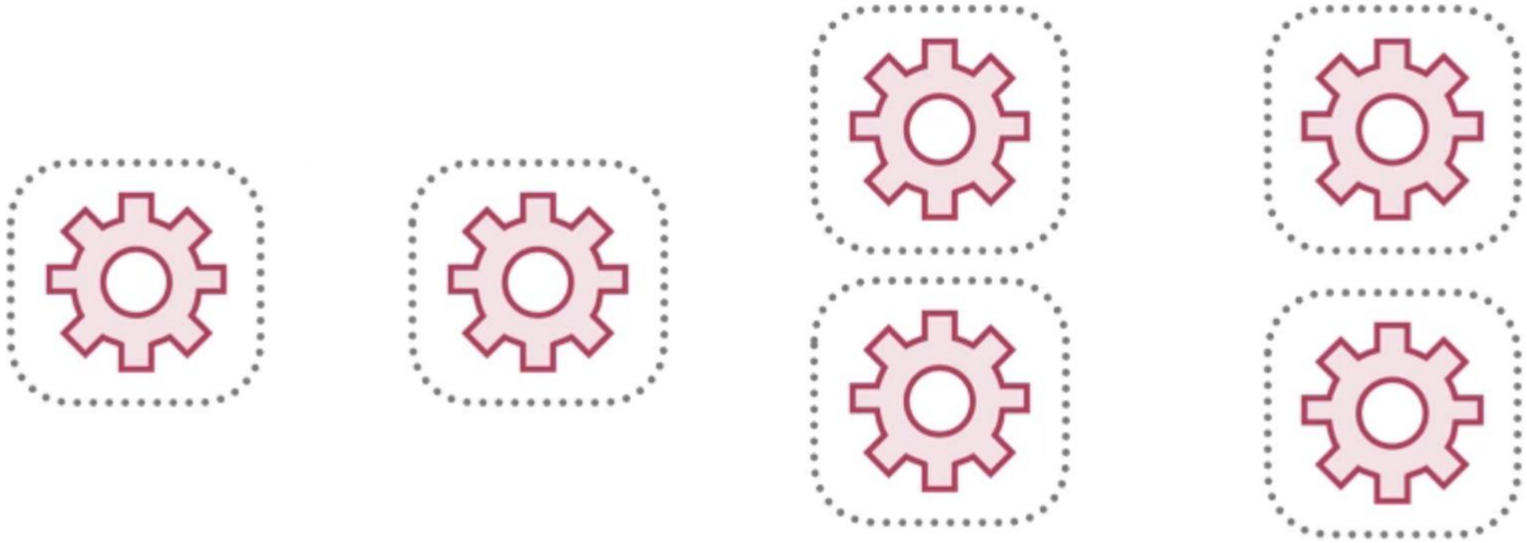
Security is not just authentication and authorization, it's also quality of service

Netflix Microservices Architecture

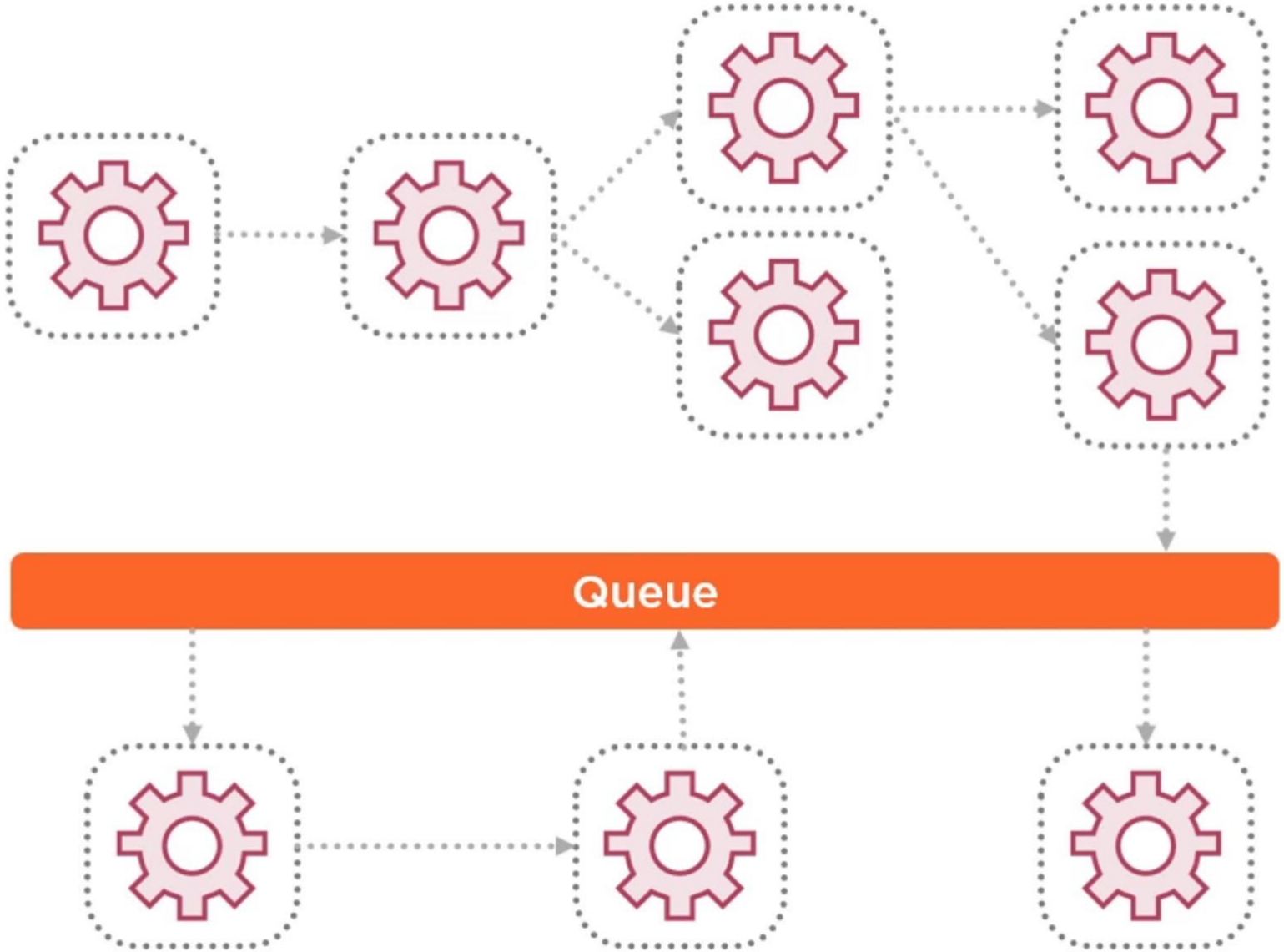


NETFLIX

Monitoring and Tracing

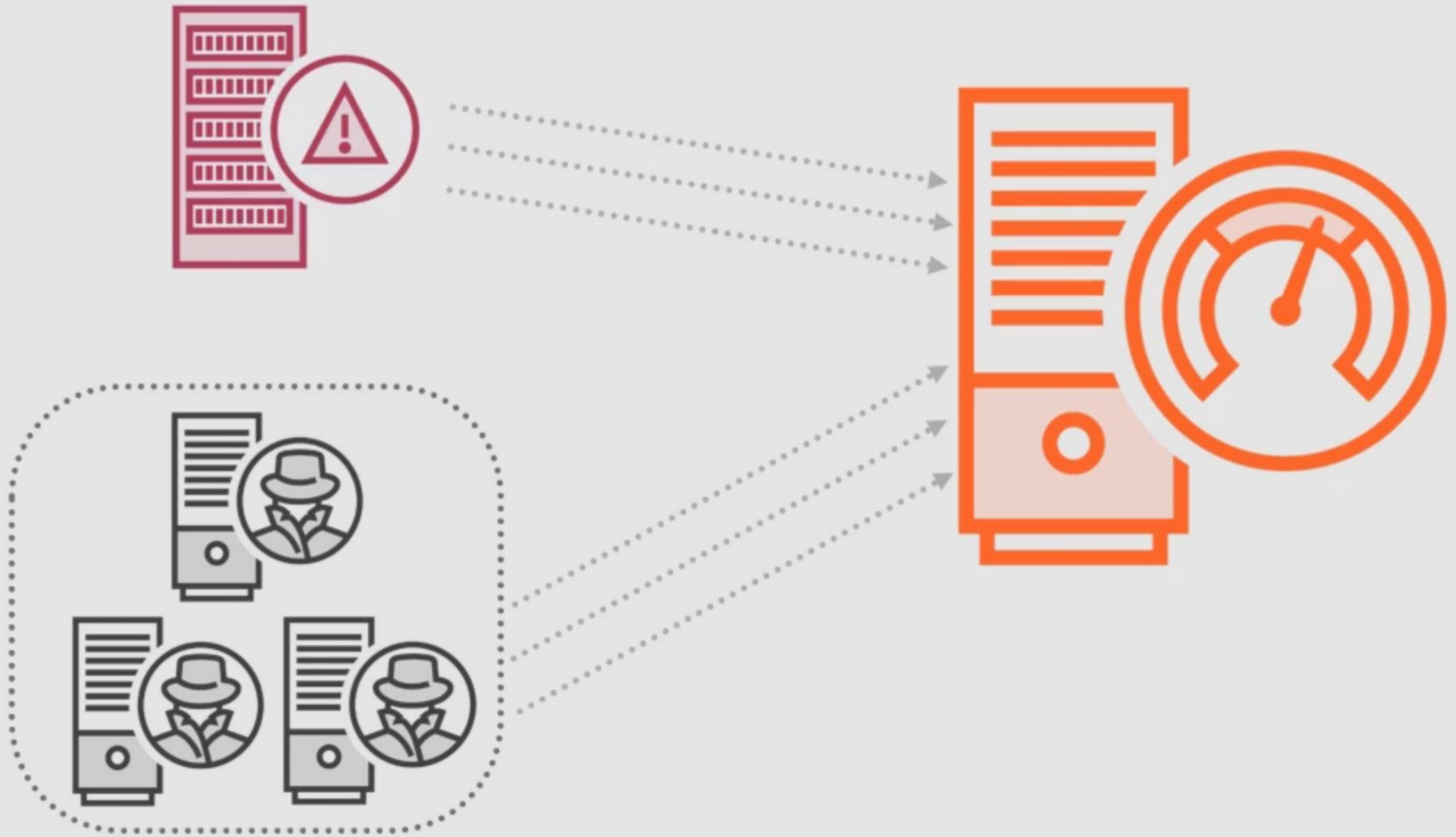


Monitoring and Tracing



Security is not just authentication and authorization, it's also quality of service

Denial of Service

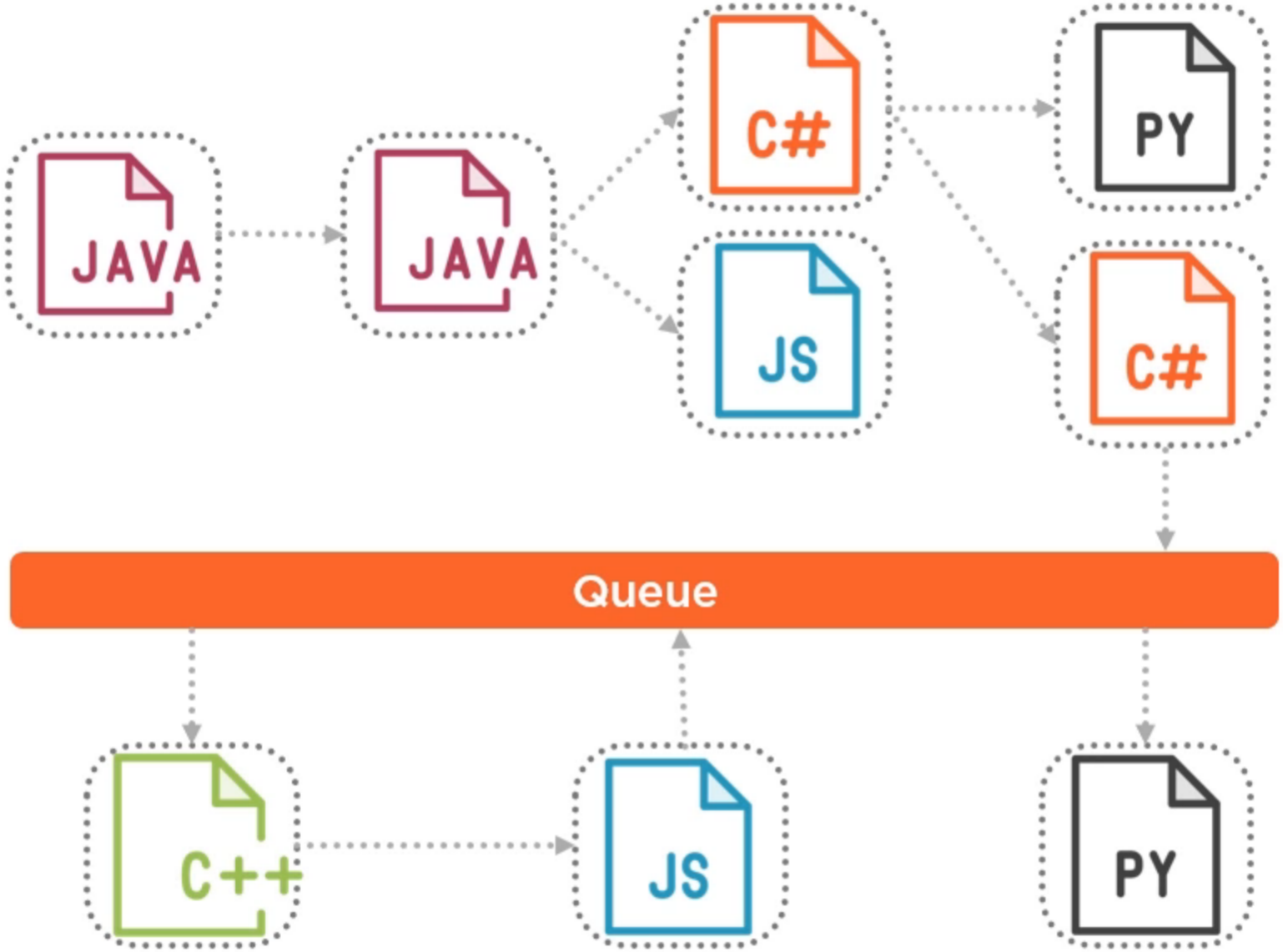


Challenges due to polyglot microservices architectures.

Requires security expertise for each technology.

Maintaining multiple sets of security best practices and guidelines for each technology.

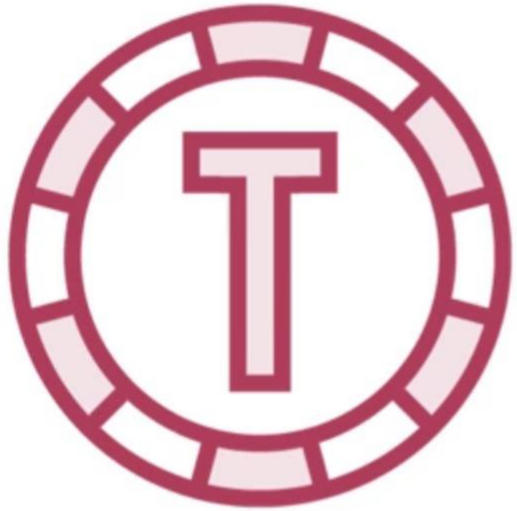
Keeping up with security patches.



Securing the Perimeter around Your Microservices

Basic authentication

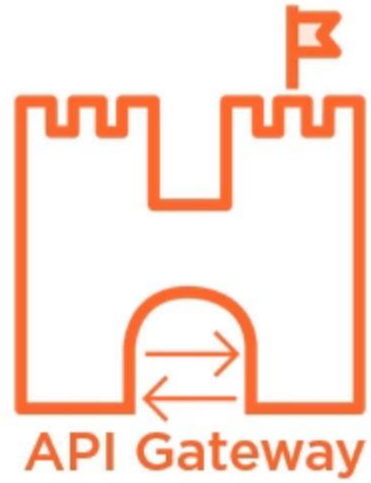
Certificates

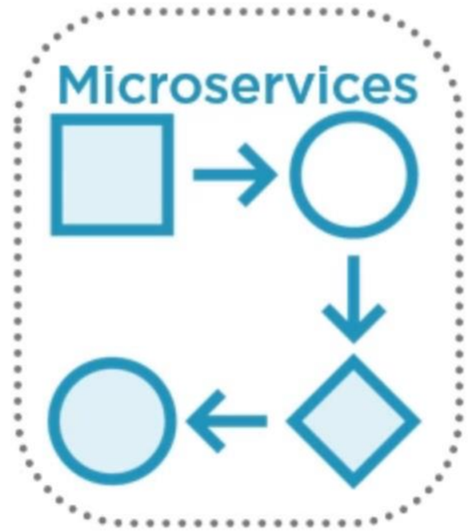


Tokens, JWT

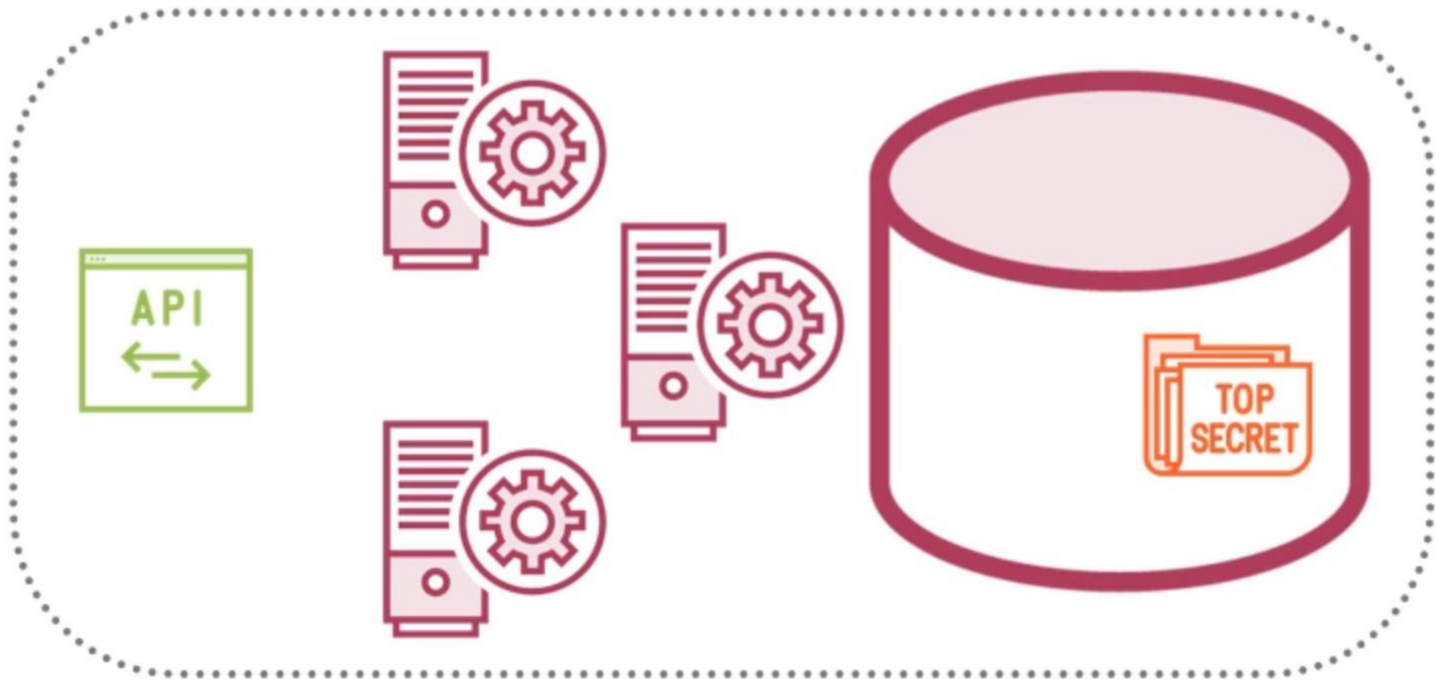
Oauth2, OpenID Connect

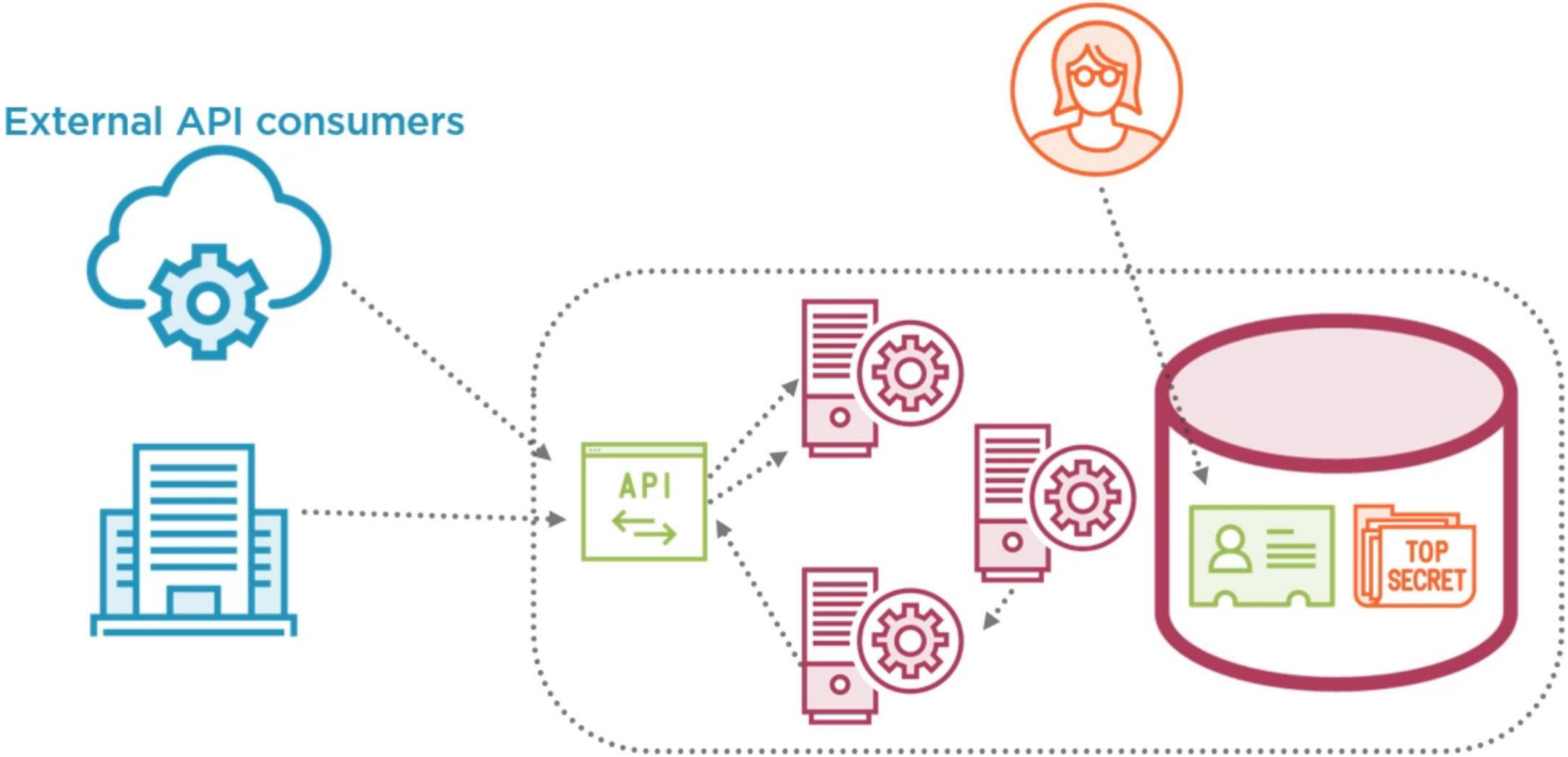

Identity provider


API Gateway


Microservices

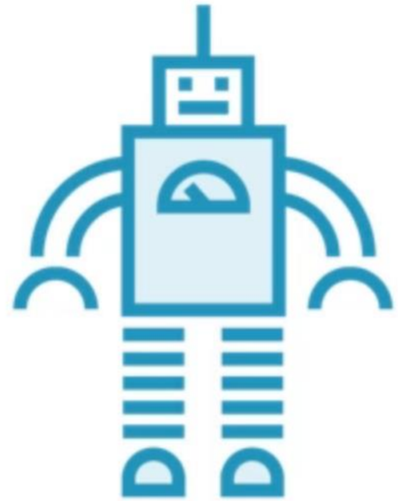
Challenges with Edge Security


API
TOP
SECRET


External API consumers
API
TOP
SECRET

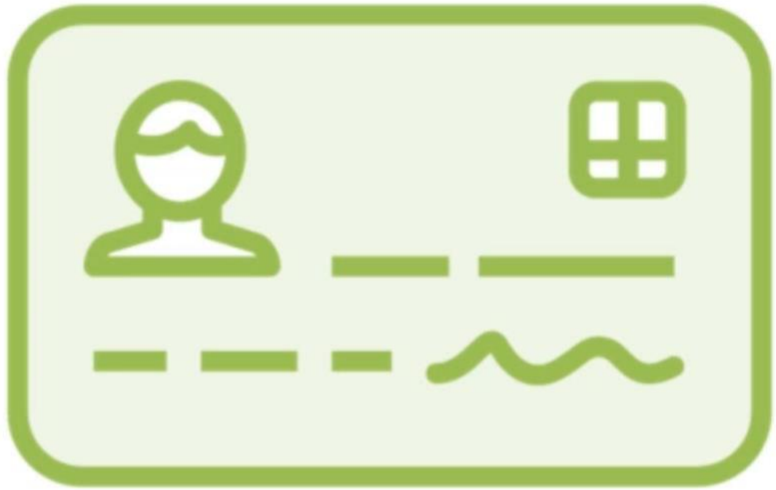
Type of API Consumer

Human (User)



Non-human (Service or system)

Authentication / Authorization



Authentication

The process of identifying who the consumer is.

Authorization

The process of identifying what the consumer can do.

Principal of Least Privilege

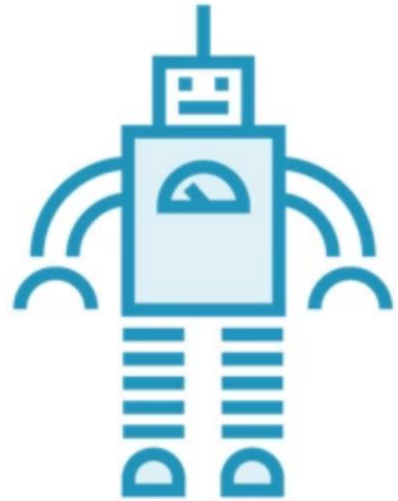
Every user or service consumer has the bare minimum privileges which are essential to perform their tasks.

Principal of Least Privilege

Limits the blast radius of any compromise to your system.

Type of API Consumer

Human (User)



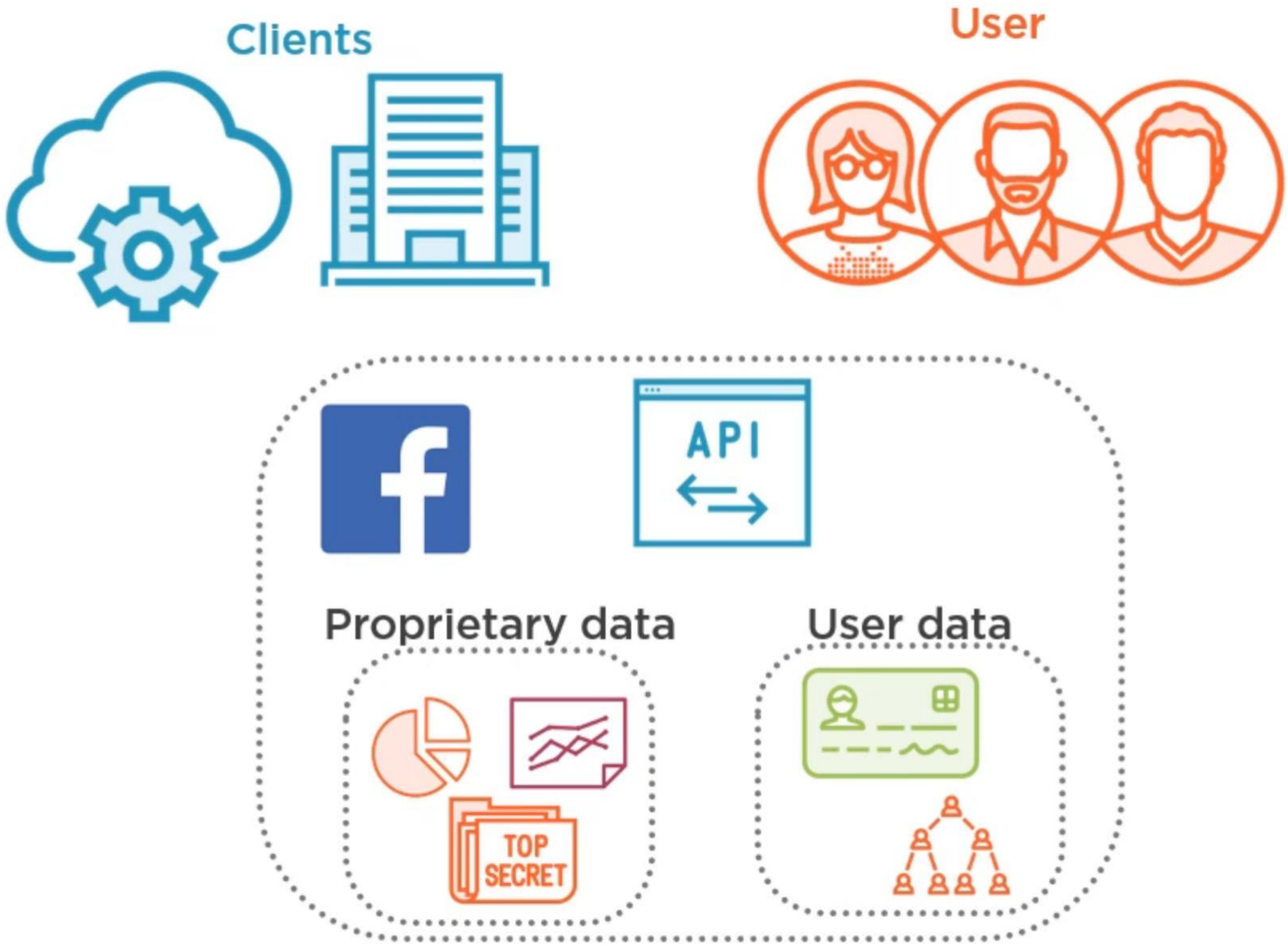
Non-human (Service or system)

Human Users

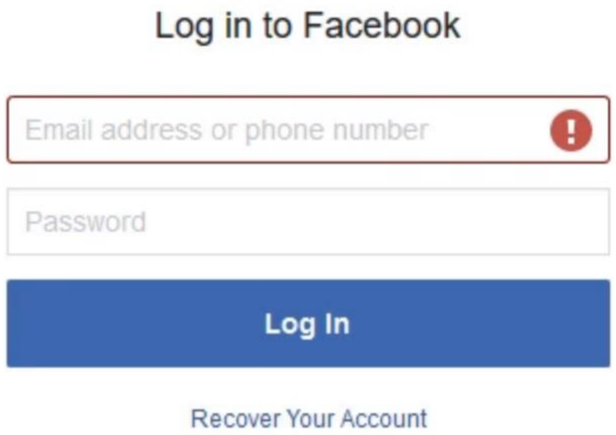
Human Users

Delegation Problem

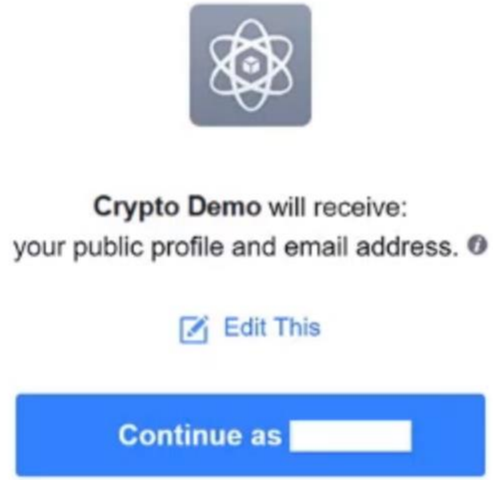
Can you trust that the client is in-fact acting on behalf of the user and has their consent to do so?


Clients
User
API
Proprietary data
User data
TOP
SECRET

Delegated Access

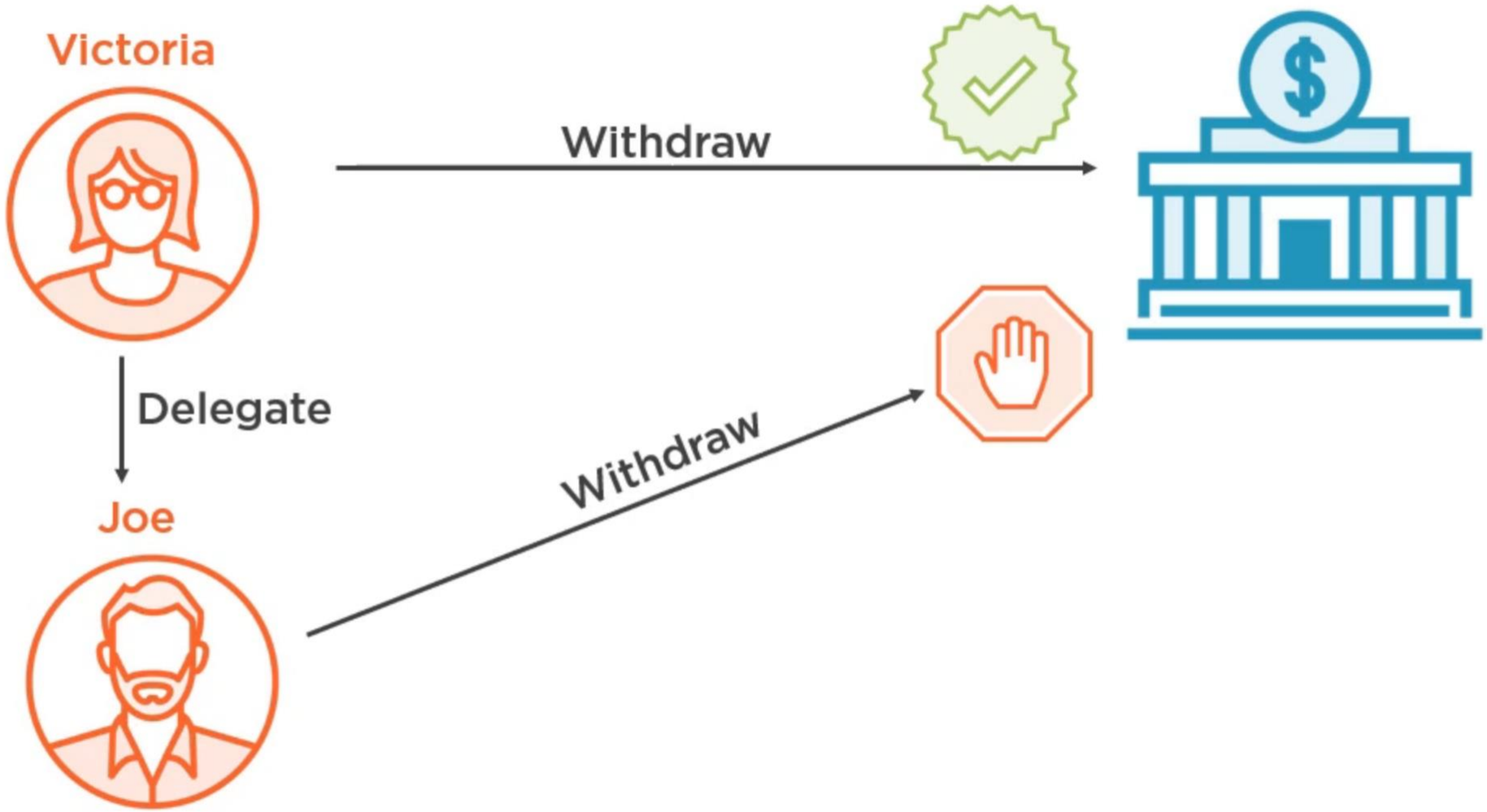


Authenticate user (resource owner)



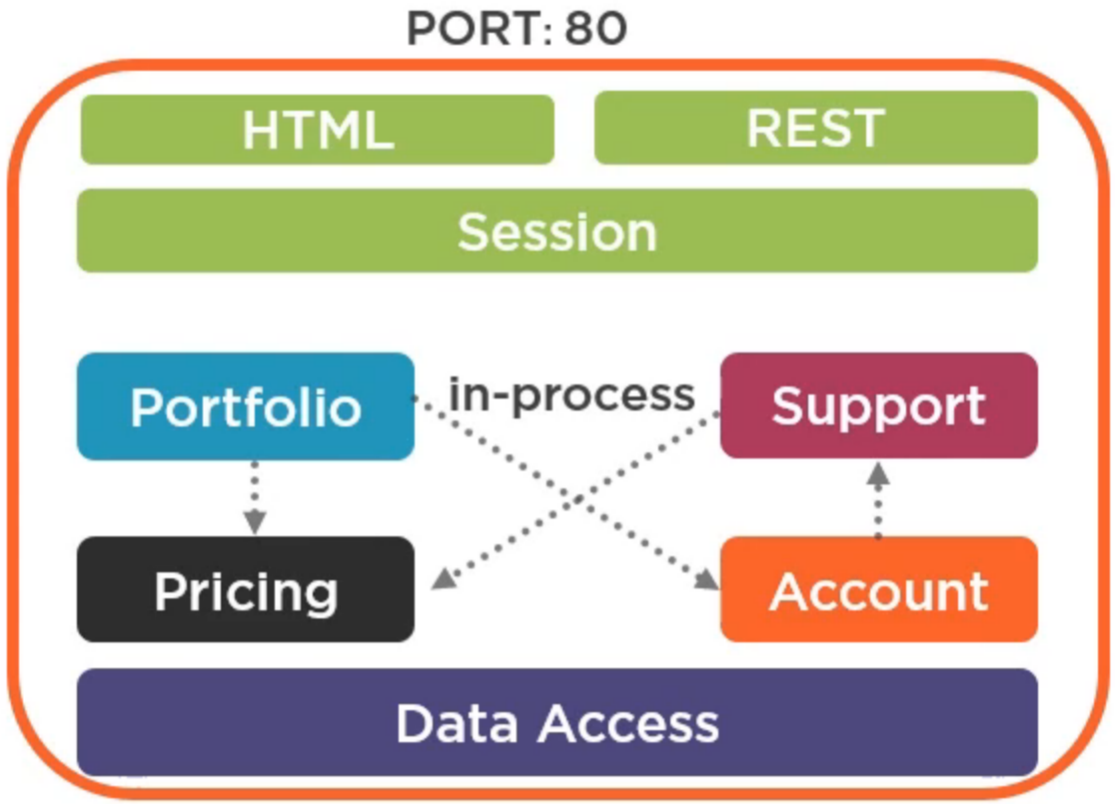
Get their consent to authorize client access to their resources

Delegated Authorization

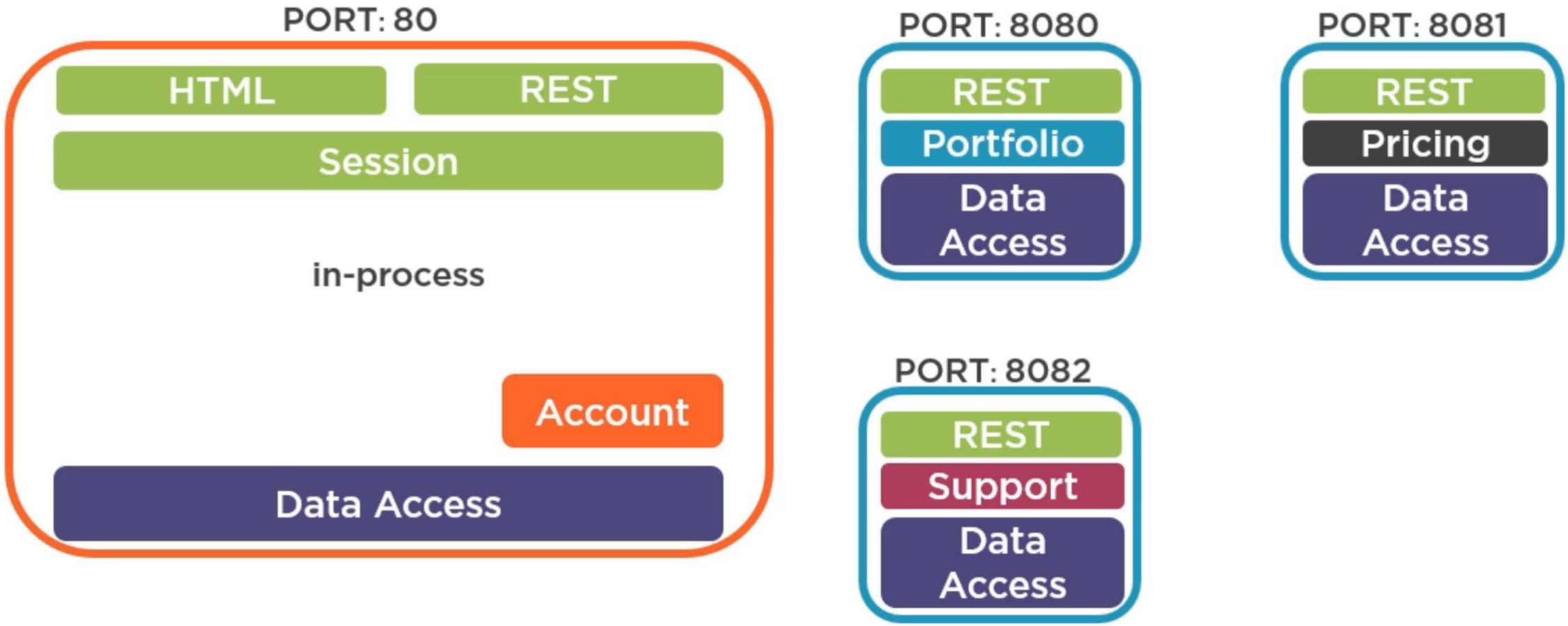


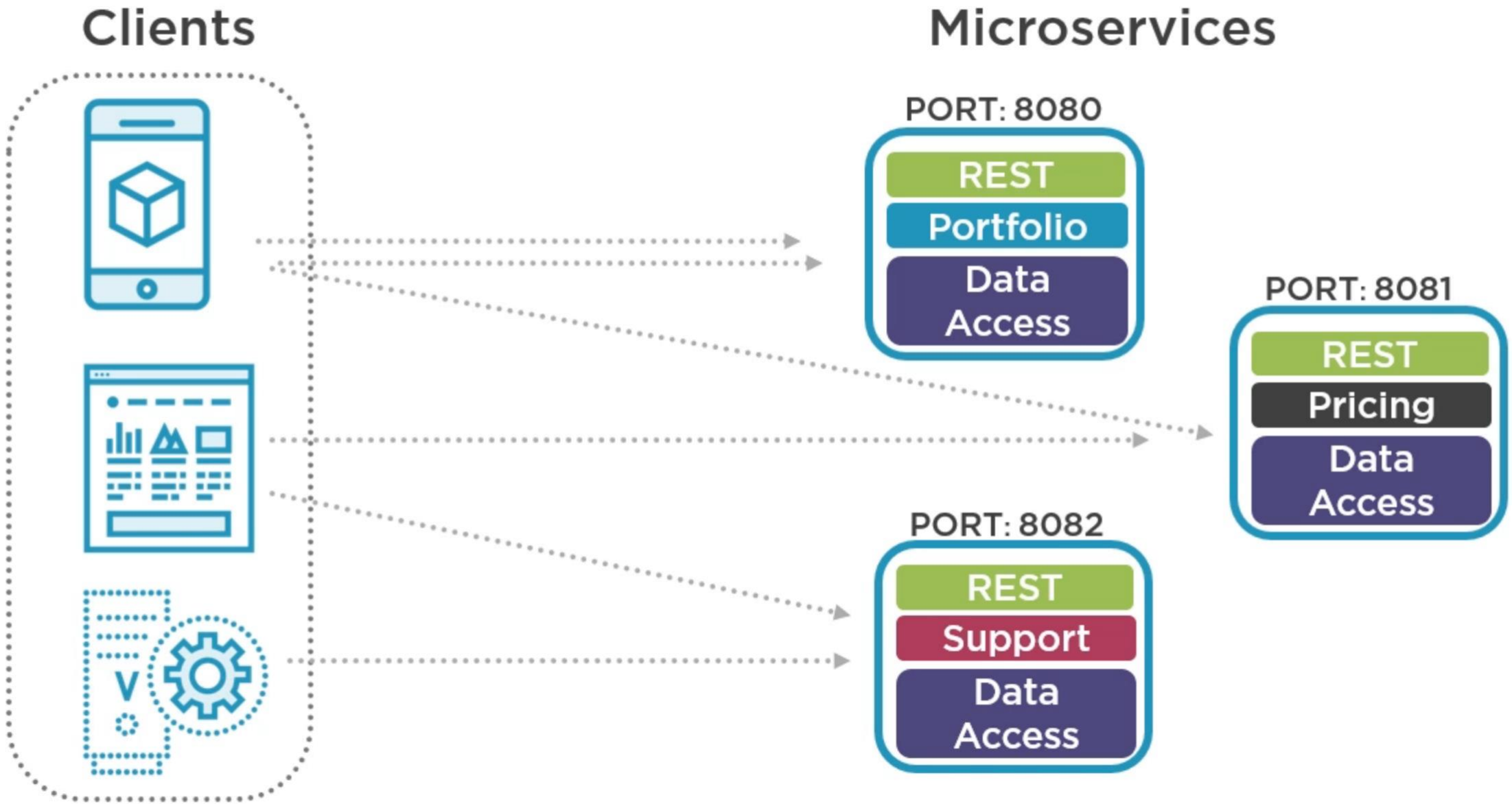
API Gateway: Exposing Your API Securely

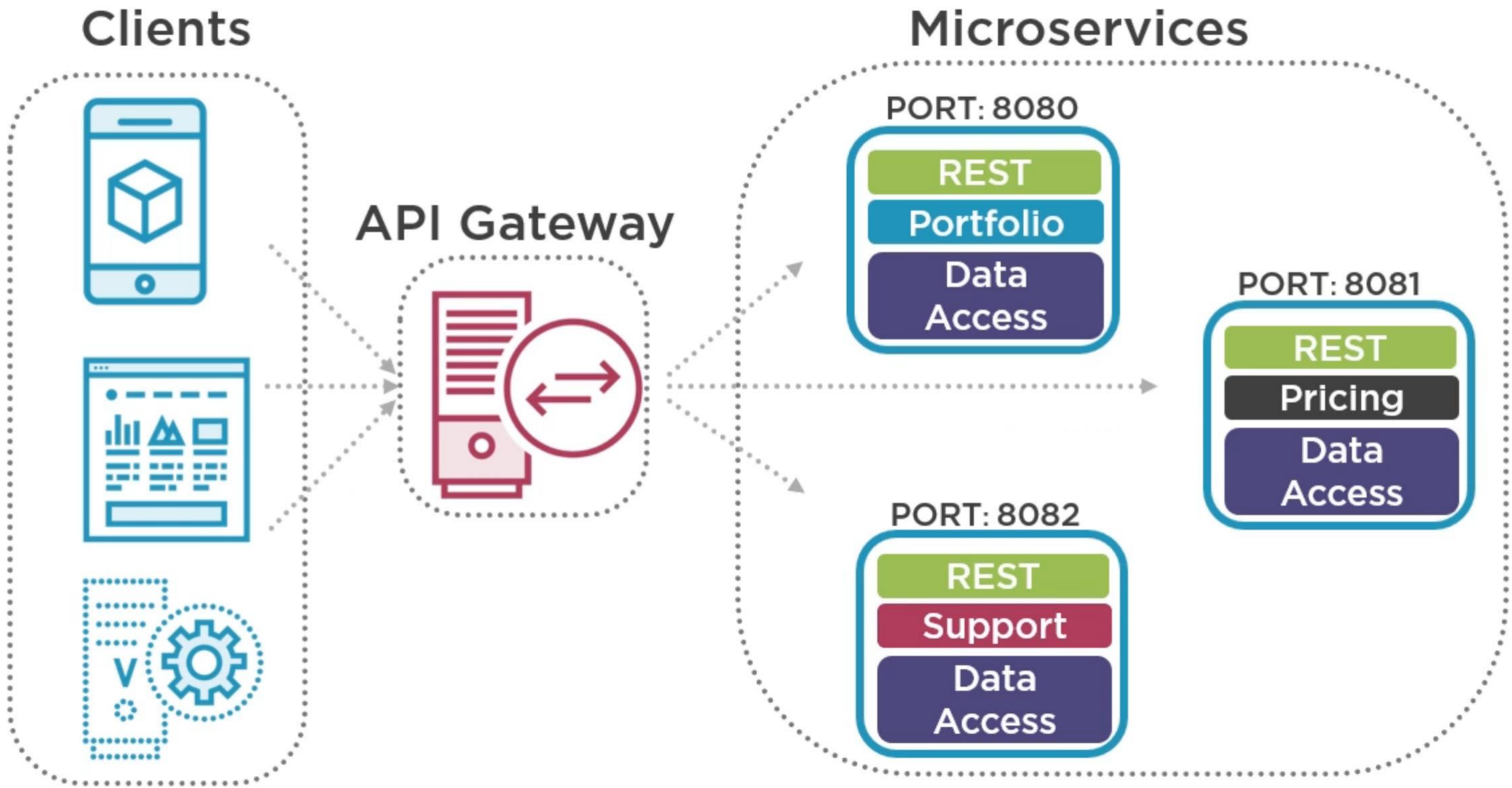
Transition to Microservices

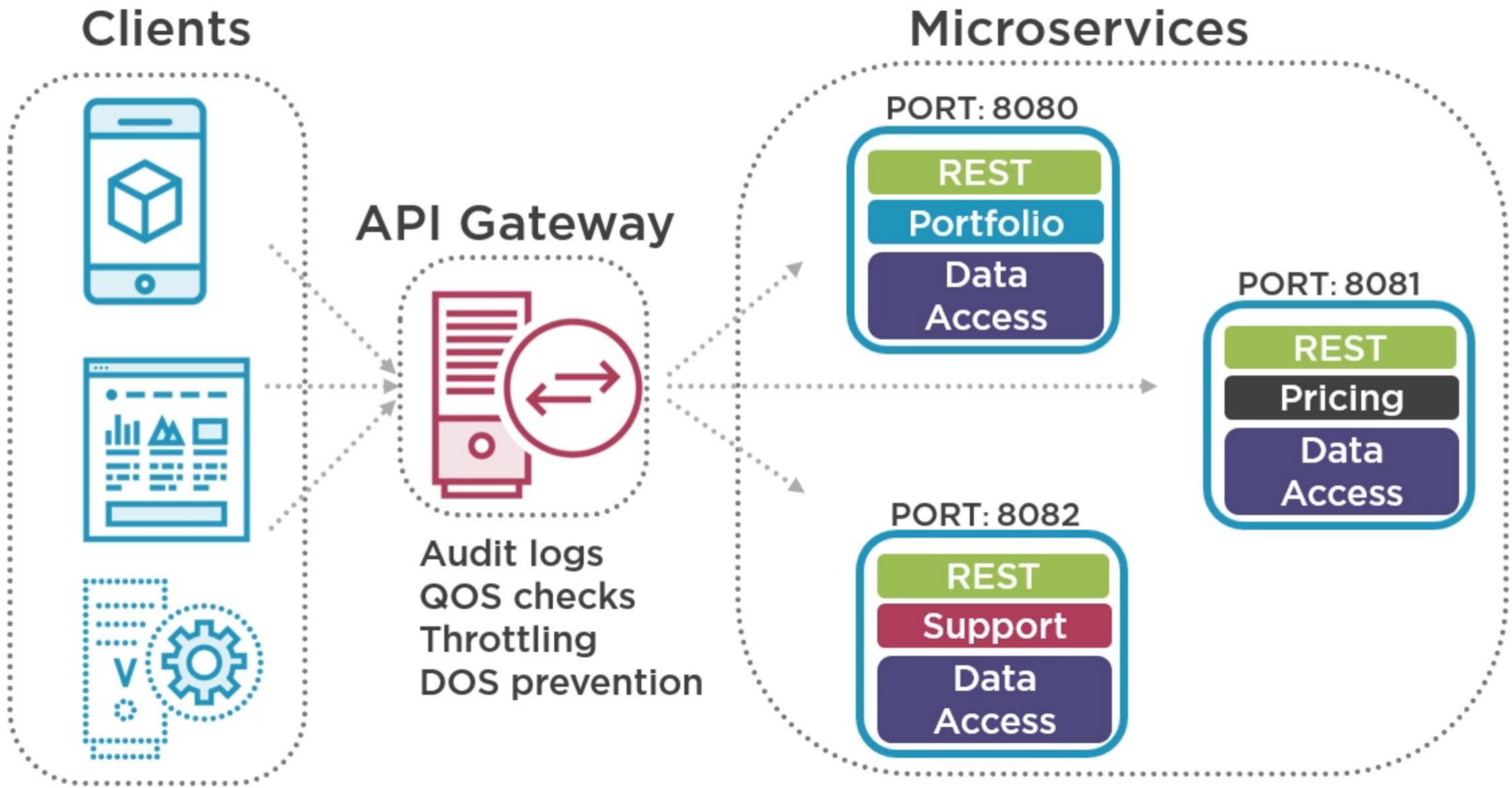


Transition to Microservices




Clients
Microservices
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access

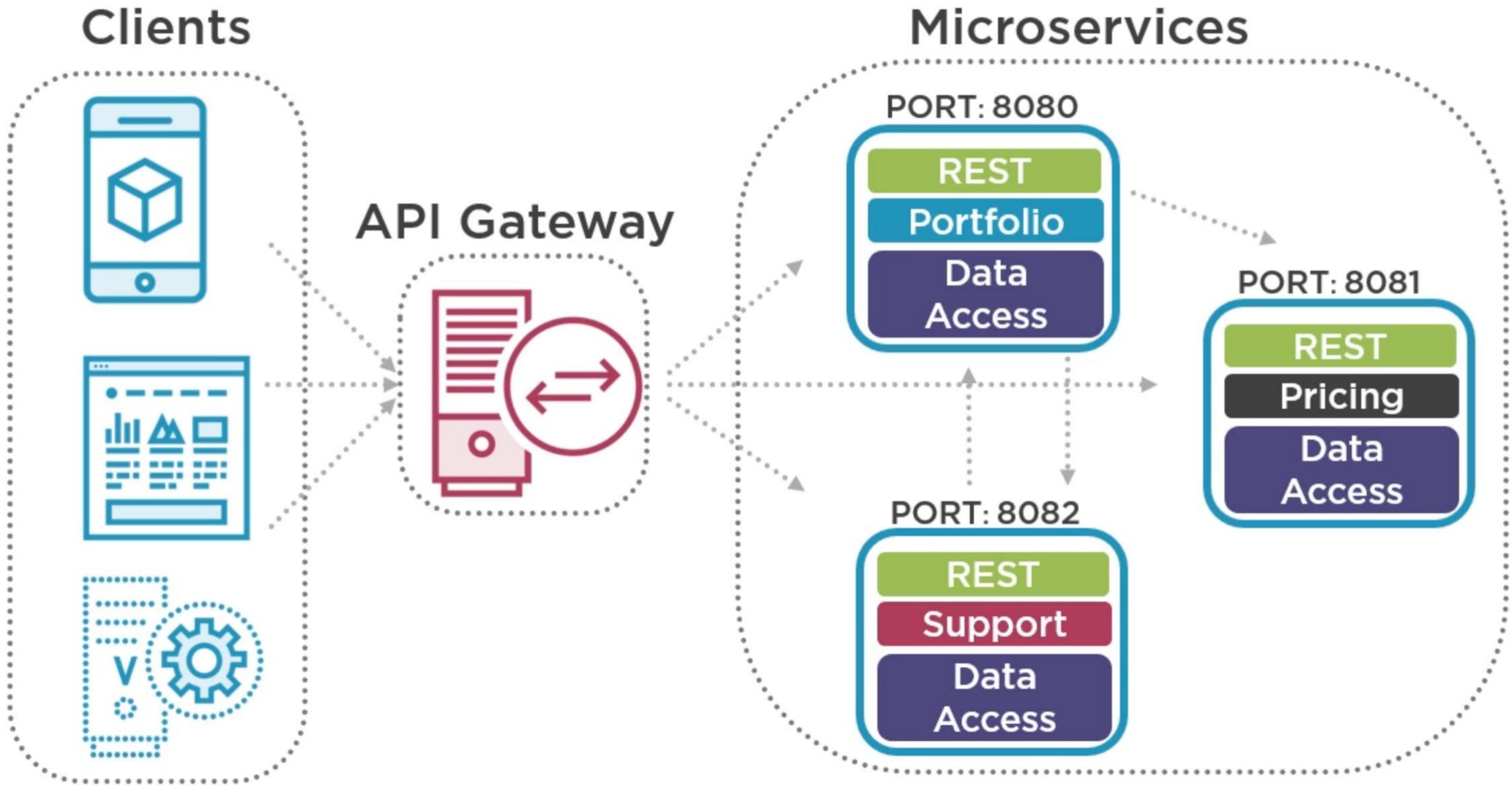

Clients
API Gateway
Microservices
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access


Clients
Microservices
API Gateway
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access
Audit logs
QOS checks
Throttling
DOS prevention

Development team can focus more on business requirements rather then non-functional security requirements.

Single Responsibility Principal

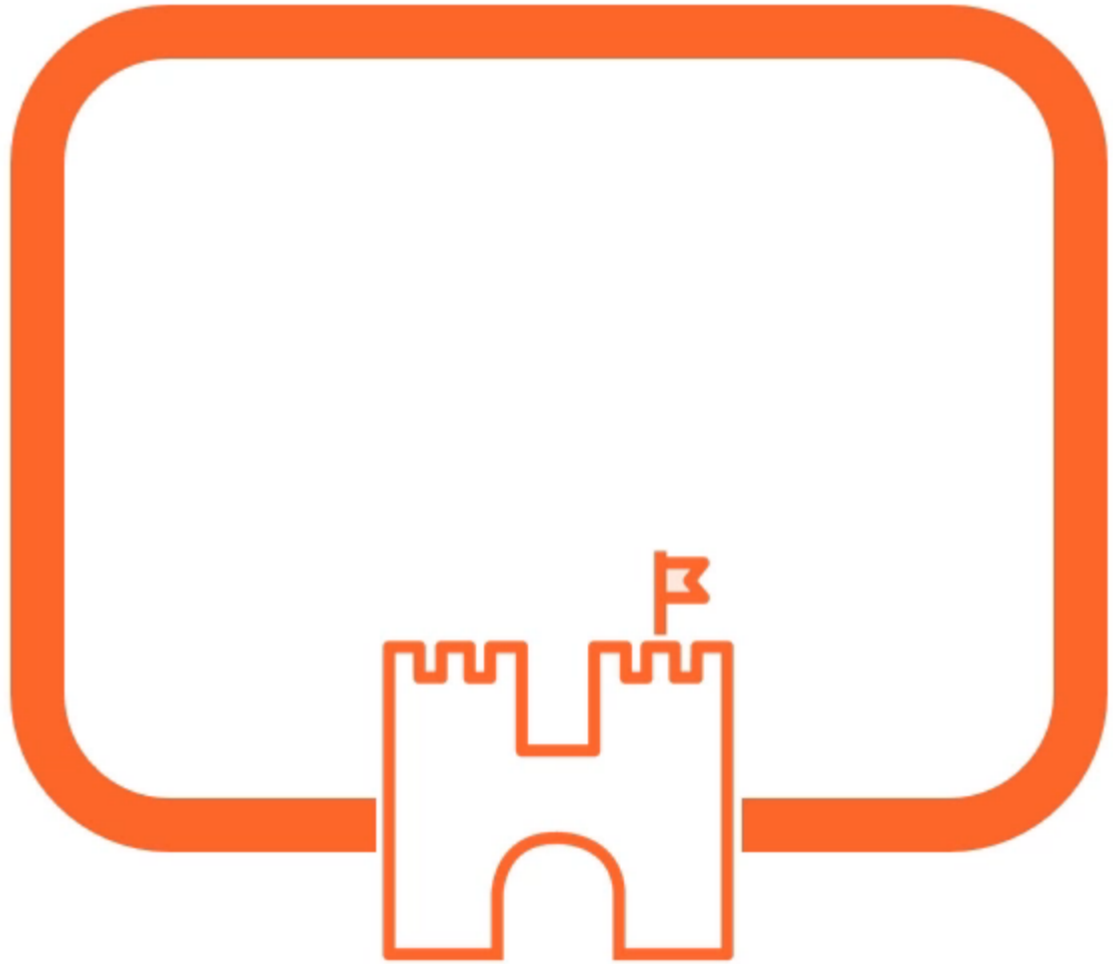
Each microservice should perform one function and do it well.

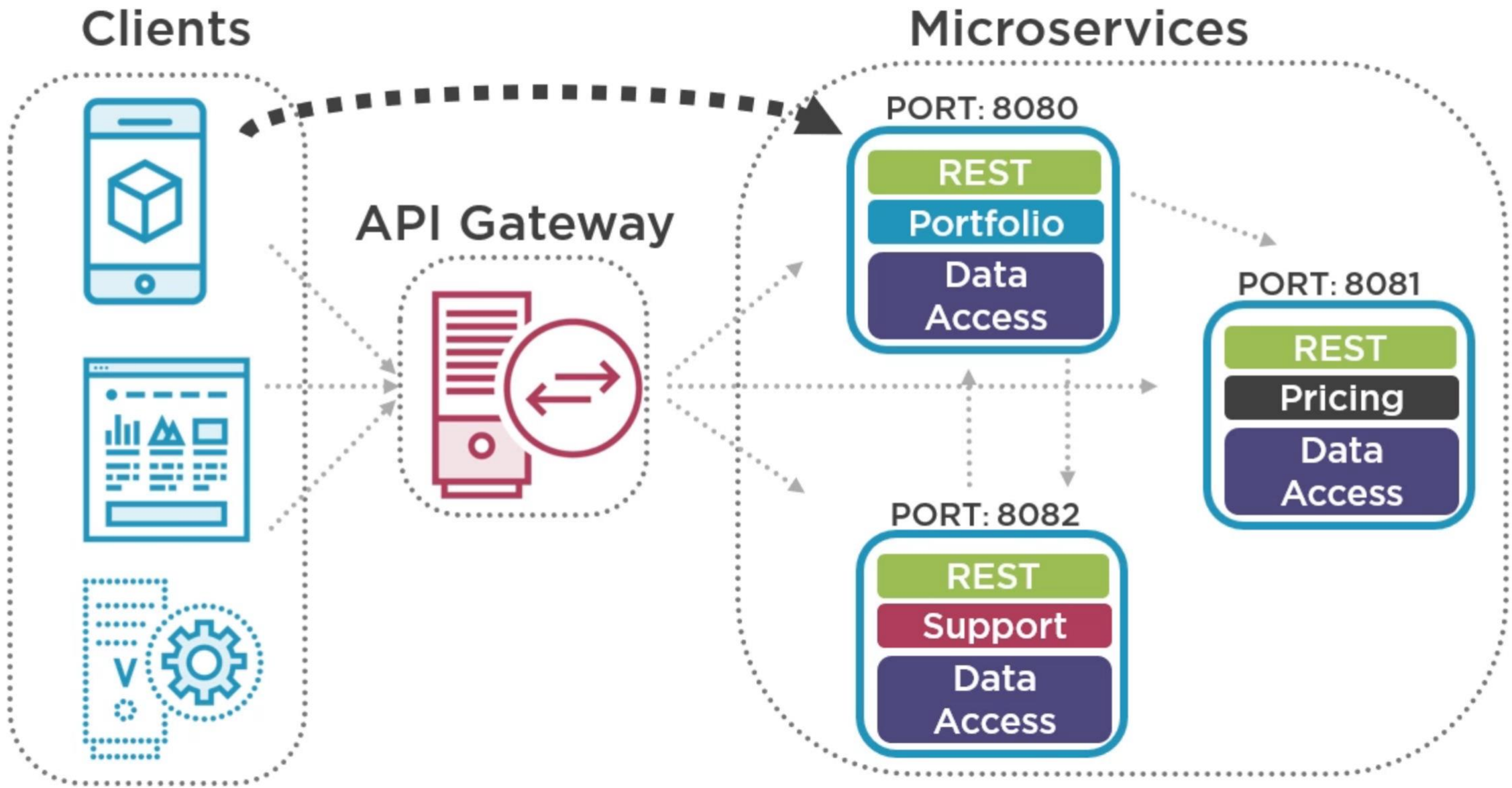

Clients
API Gateway
Microservices
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access

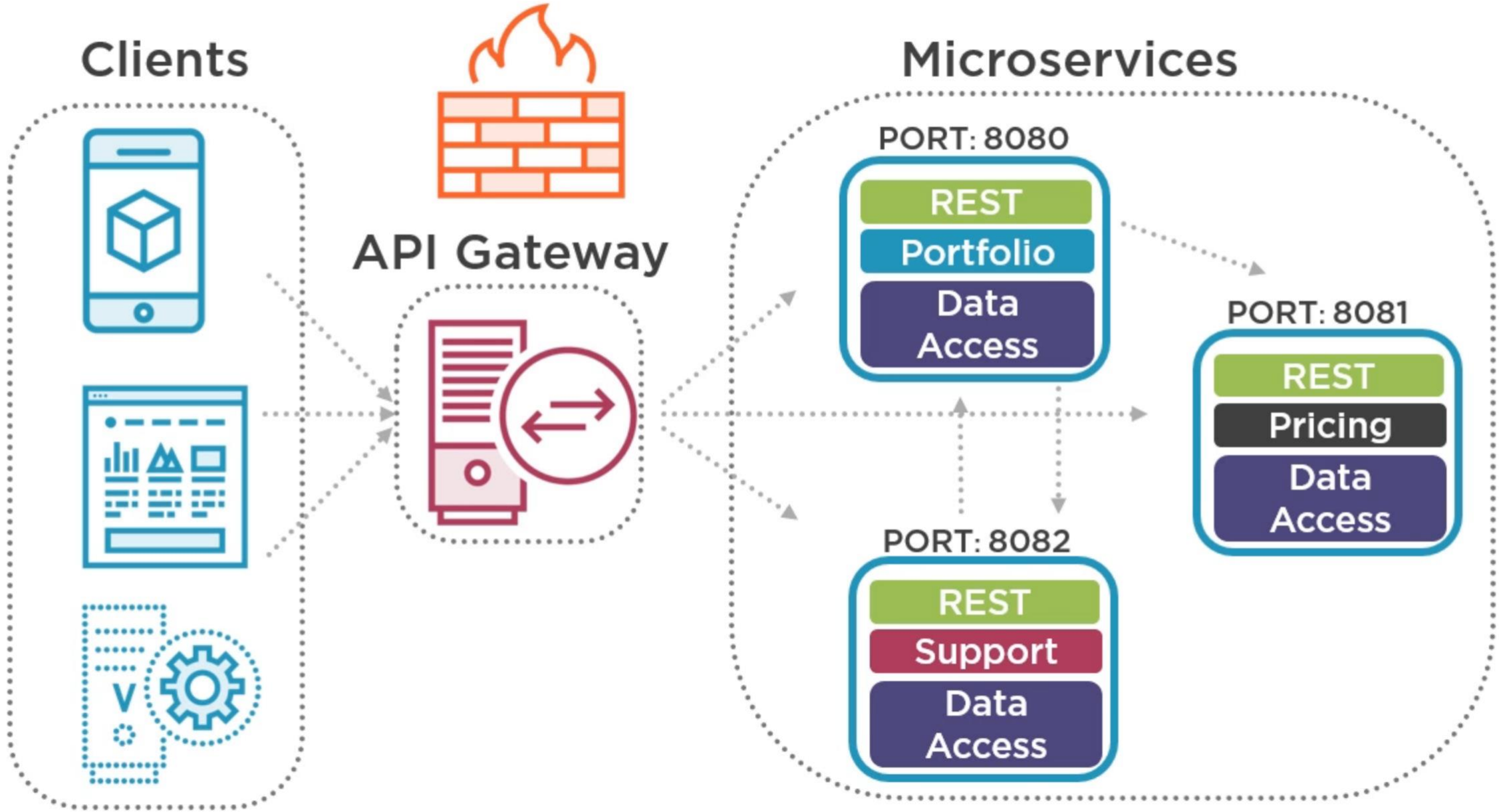
Benefits of an API Gateway

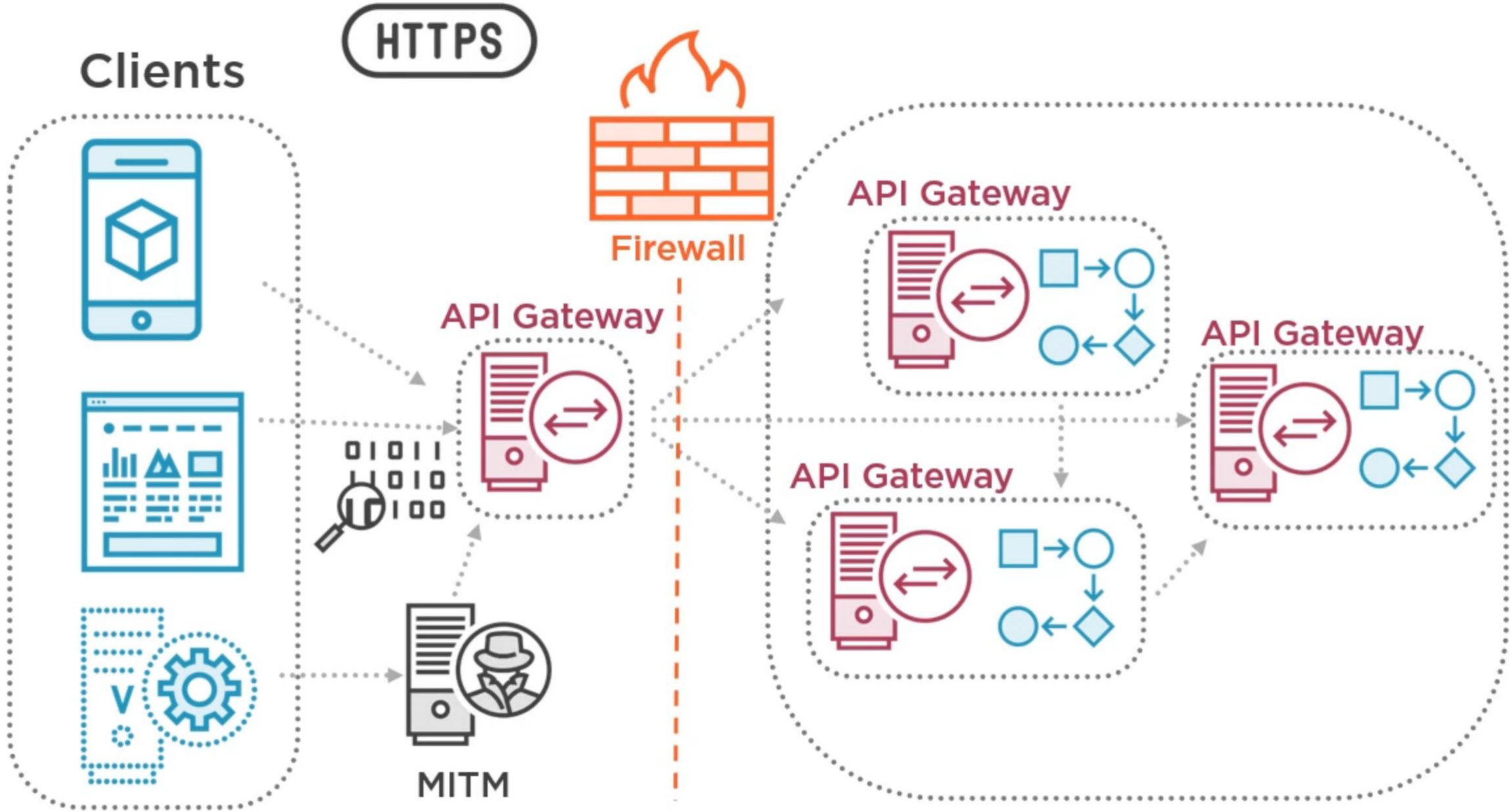
Avoids the need for shared libraries making our architecture polyglot.

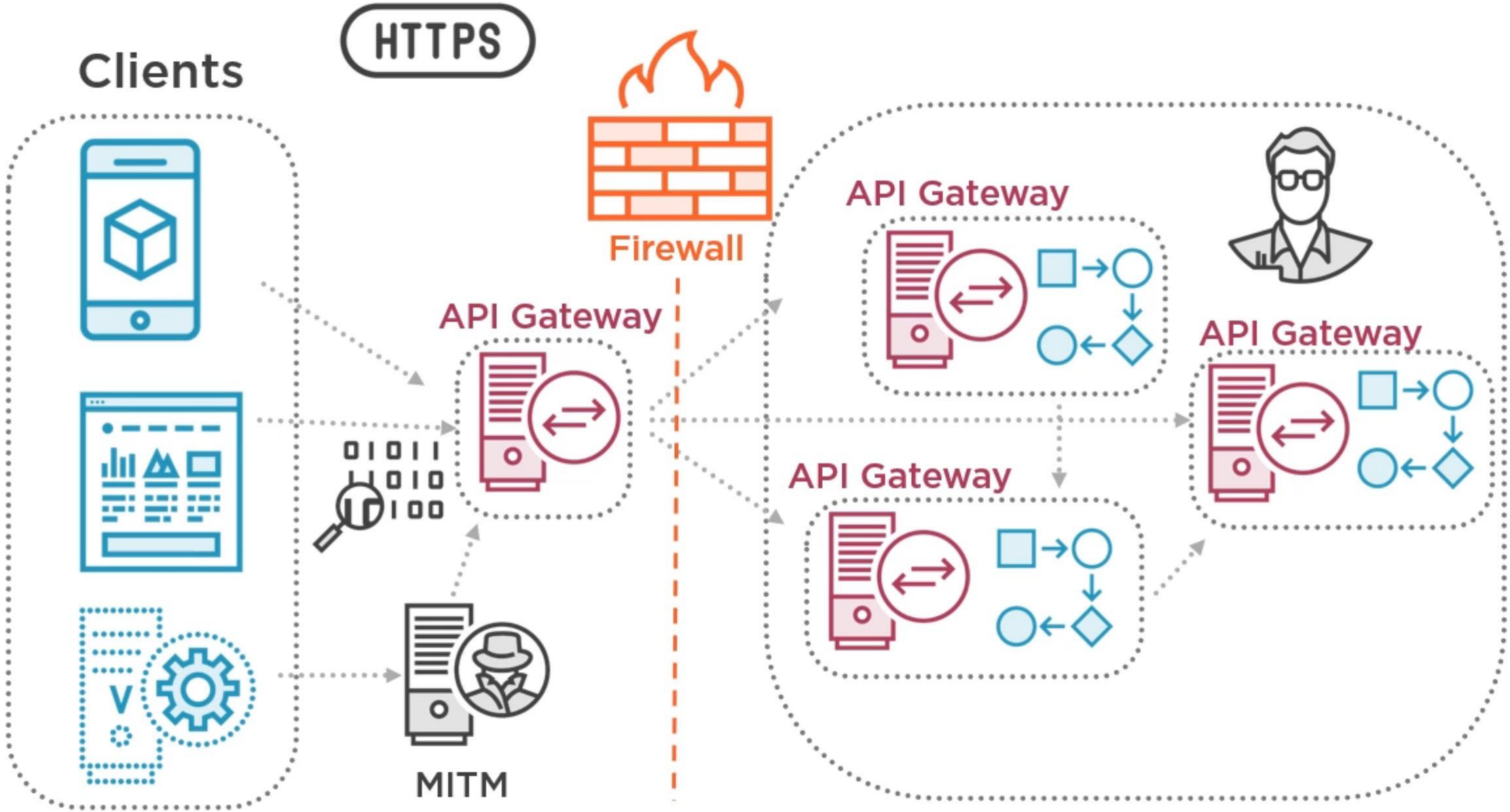
Defensive Structures




Clients
Microservices
API Gateway
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access


Clients
API Gateway
Microservices
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access


HTTPS
Clients
Firewall
API Gateway
API Gateway
API Gateway
API Gateway
MITM


HTTPS
Clients
Firewall
API Gateway
API Gateway
API Gateway
API Gateway
MITM

Basic- username / password

Static API key

Client id and client secret

OpenID connect

SAML

Fingerprinting

Certificates

Considering Basic Authentication

Basic Authentication

Base64 encoded

username:password

dXNlcm5hbWU6cGFzc3dvcmQK

Header name	Value
Authorization	Basic dXNlcm5hbWU6cGFzc3dvcmQK

Tweet Share 20.2K

Base64 encoder/decoder online

In this page you can encoder or decoder in Base64 a string and viceversa.

dXN1cm5hbWU6cGFzc3dvcmQK

encode -->
decode -->
<-- encode
<-- decode

username:password

This website uses cookies to ensure you get the best experience on our website

Why Basic Is Not Suitable for Microservices

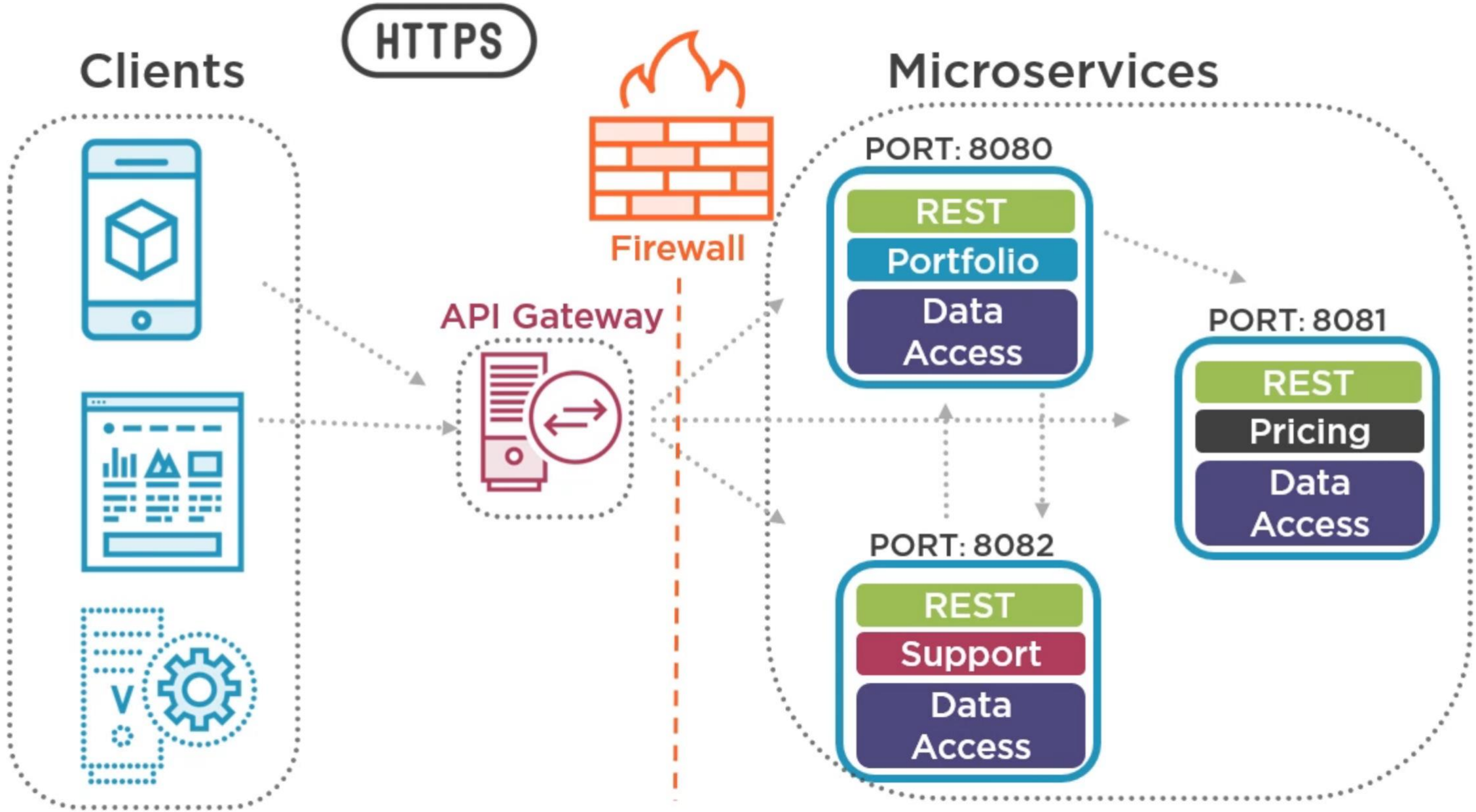
Authentication is more complex then just a username and password, requiring 2 Factor or Multifactor.

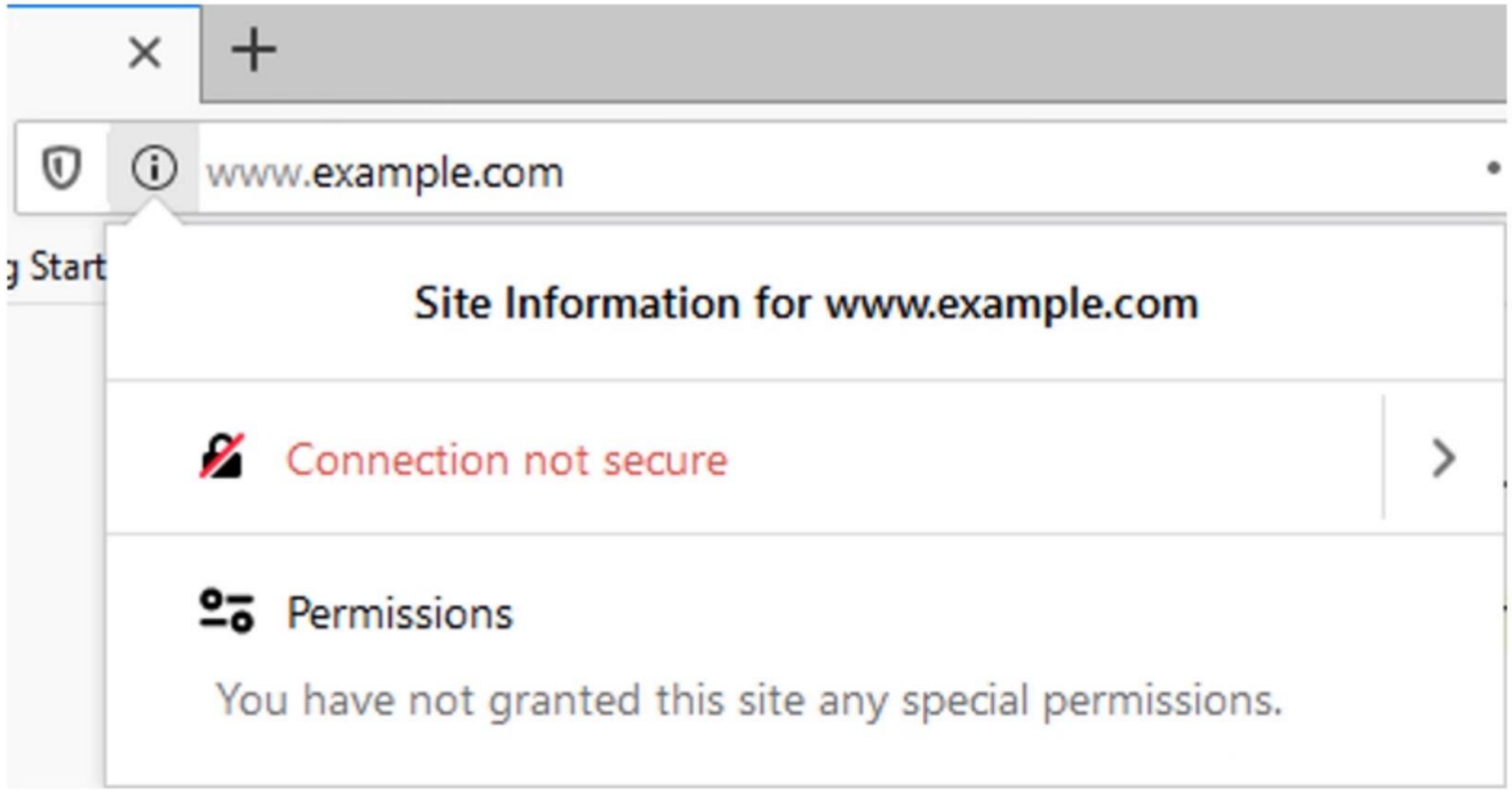
For persistent login the credentials would have to be stored on the client.

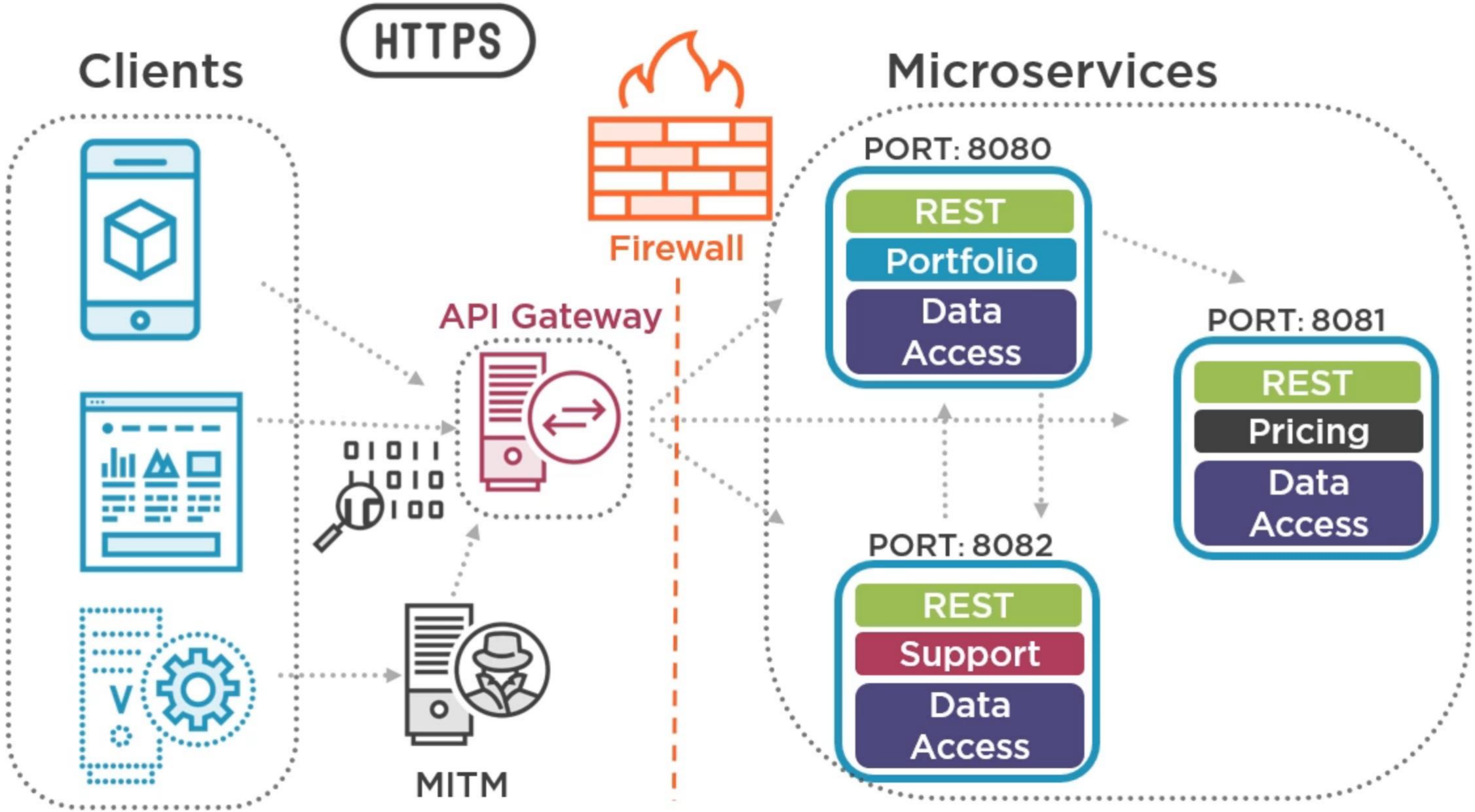
In a distributed system, credentials are exposed to multiple services.

Delegation can only be done by requesting the users credentials.

Authenticating Clients with Certificates

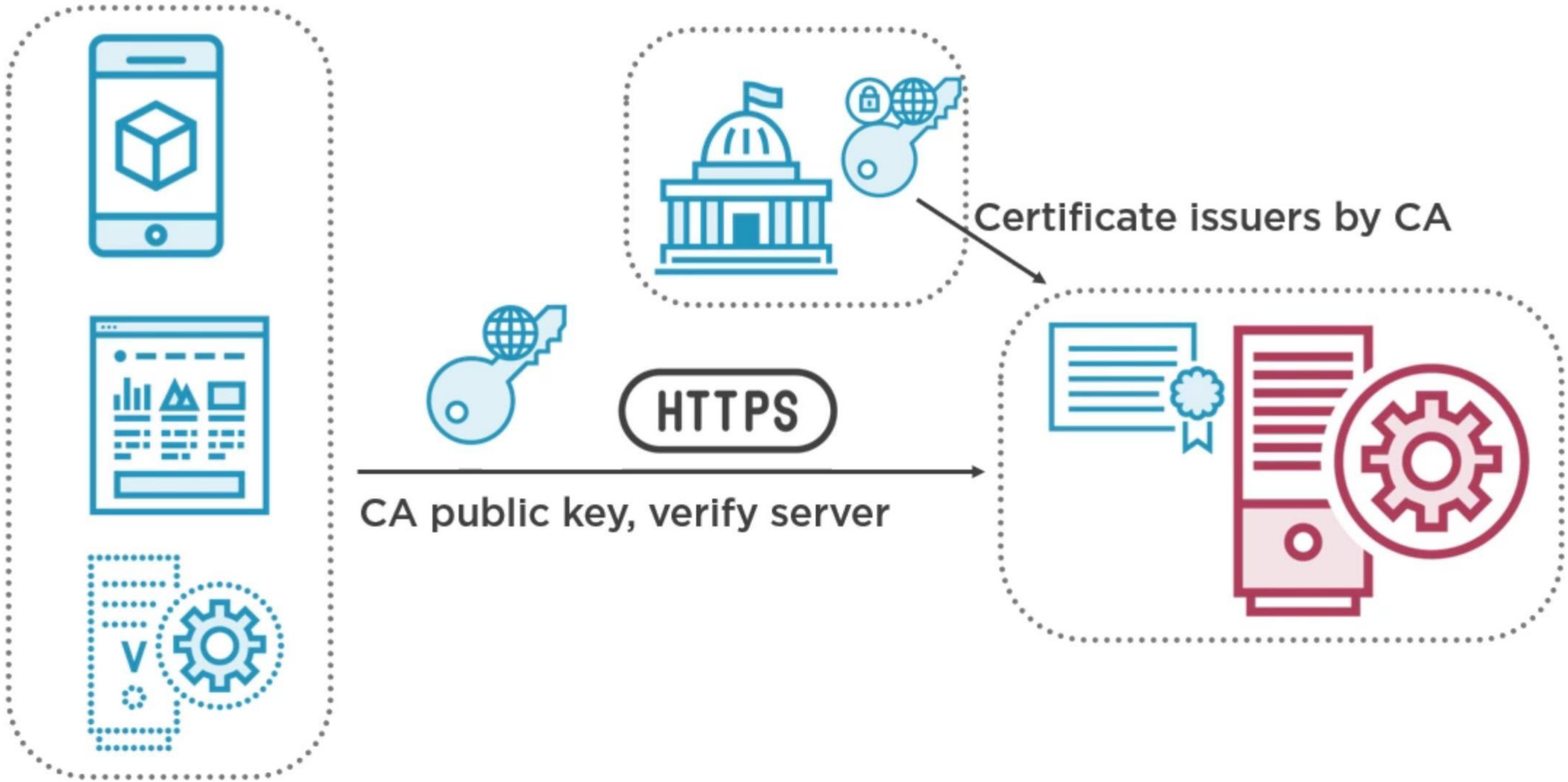

HTTPS
Clients
Microservices
Firewall
API Gateway
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access

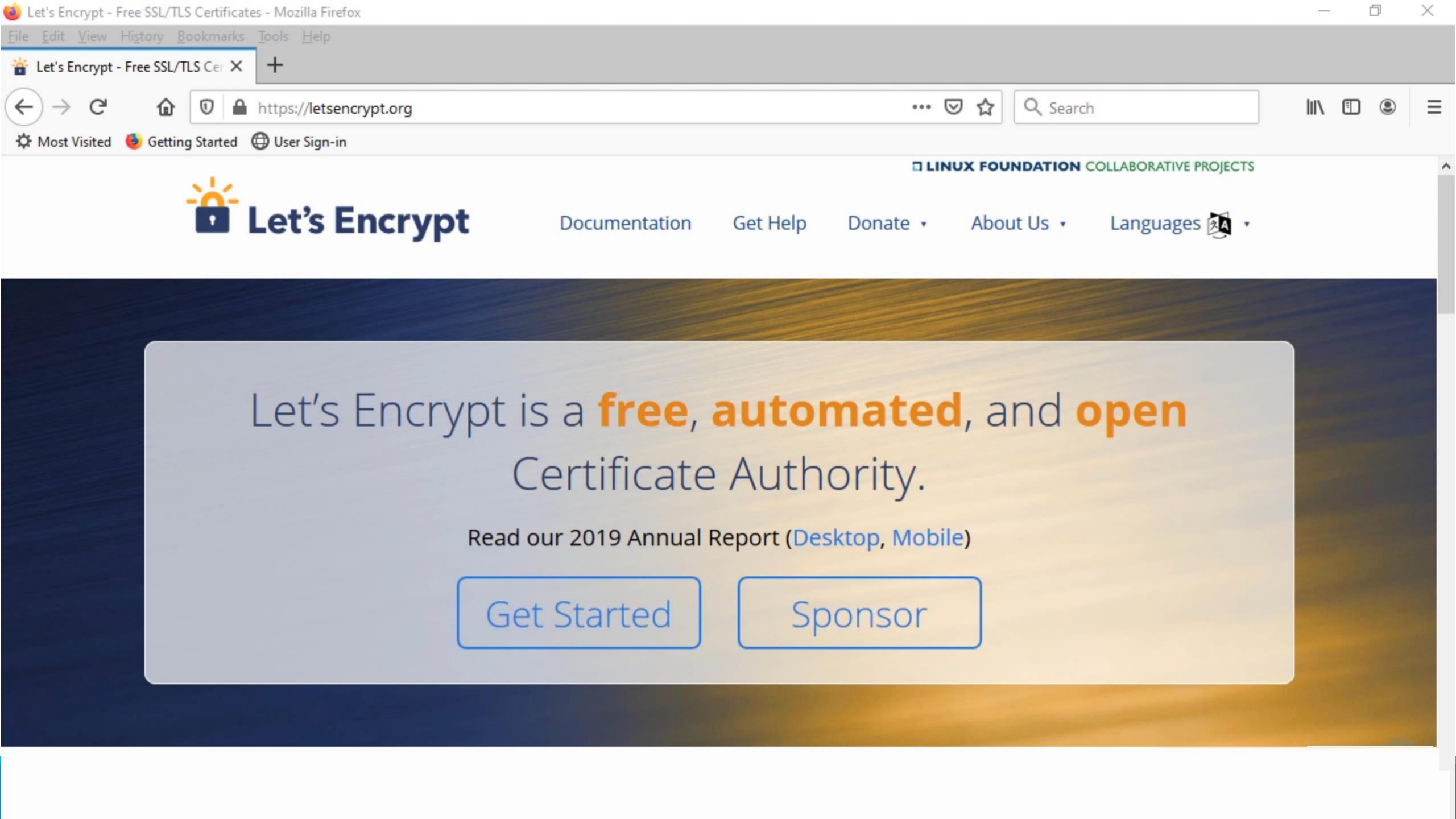

www.example.com
Site Information for www.example.com
Connection not secure
Permissions
You have not granted this site any special permissions.

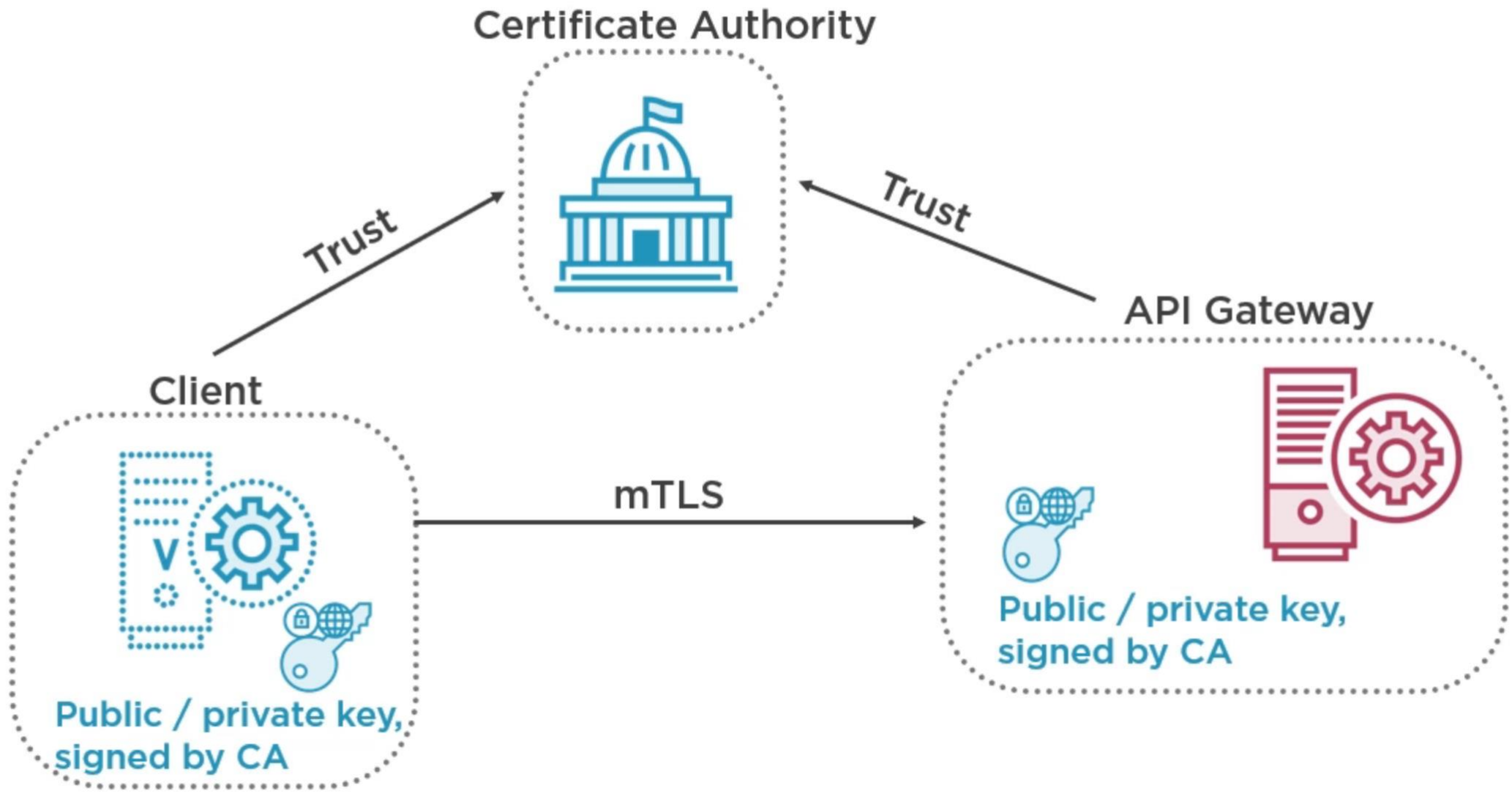

HTTPS
Clients
Microservices
Firewall
API Gateway
MITM
PORT: 8080
REST
Portfolio
Data Access
PORT: 8081
REST
Pricing
Data Access
PORT: 8082
REST
Support
Data Access

At minimum TLS 1.2 or higher

Transport Layer Security (TLS)




Let's Encrypt - Free SSL/TLS Certificates - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Let's Encrypt - Free SSL/TLS Ce
https://letsencrypt.org
Search
Most Visited
Getting Started
User Sign-in
LINUX FOUNDATION COLLABORATIVE PROJECTS
Let's Encrypt
Documentation
Get Help
Donate
About Us
Languages
Let's Encrypt is a free, automated, and open Certificate Authority.
Read our 2019 Annual Report (Desktop, Mobile)
Get Started
Sponsor


Certificate Authority
Trust
Trust
API Gateway
Client
mTLS
Public / private key,
signed by CA
Public / private key,
signed by CA

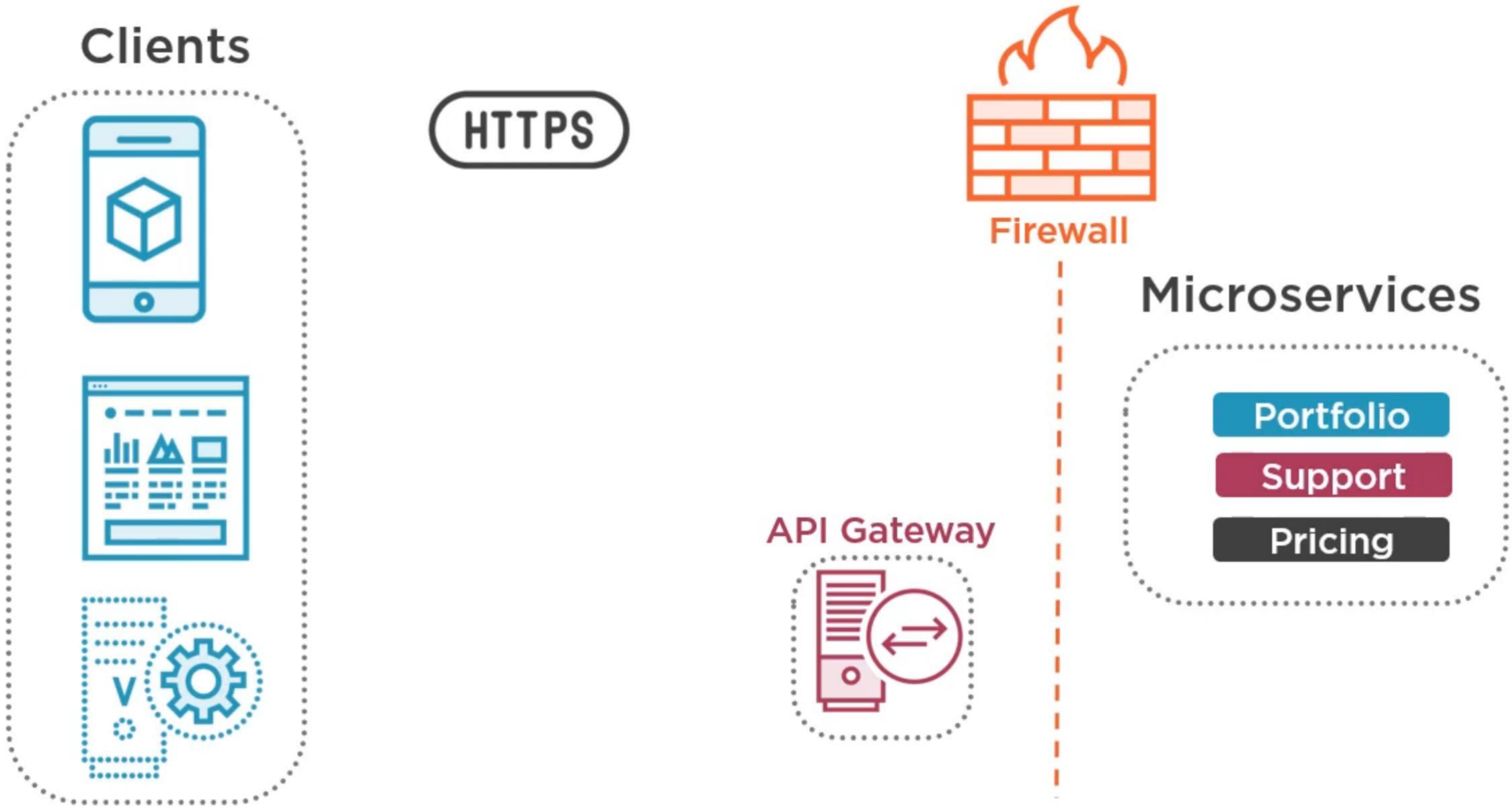
Challenges with mTLS

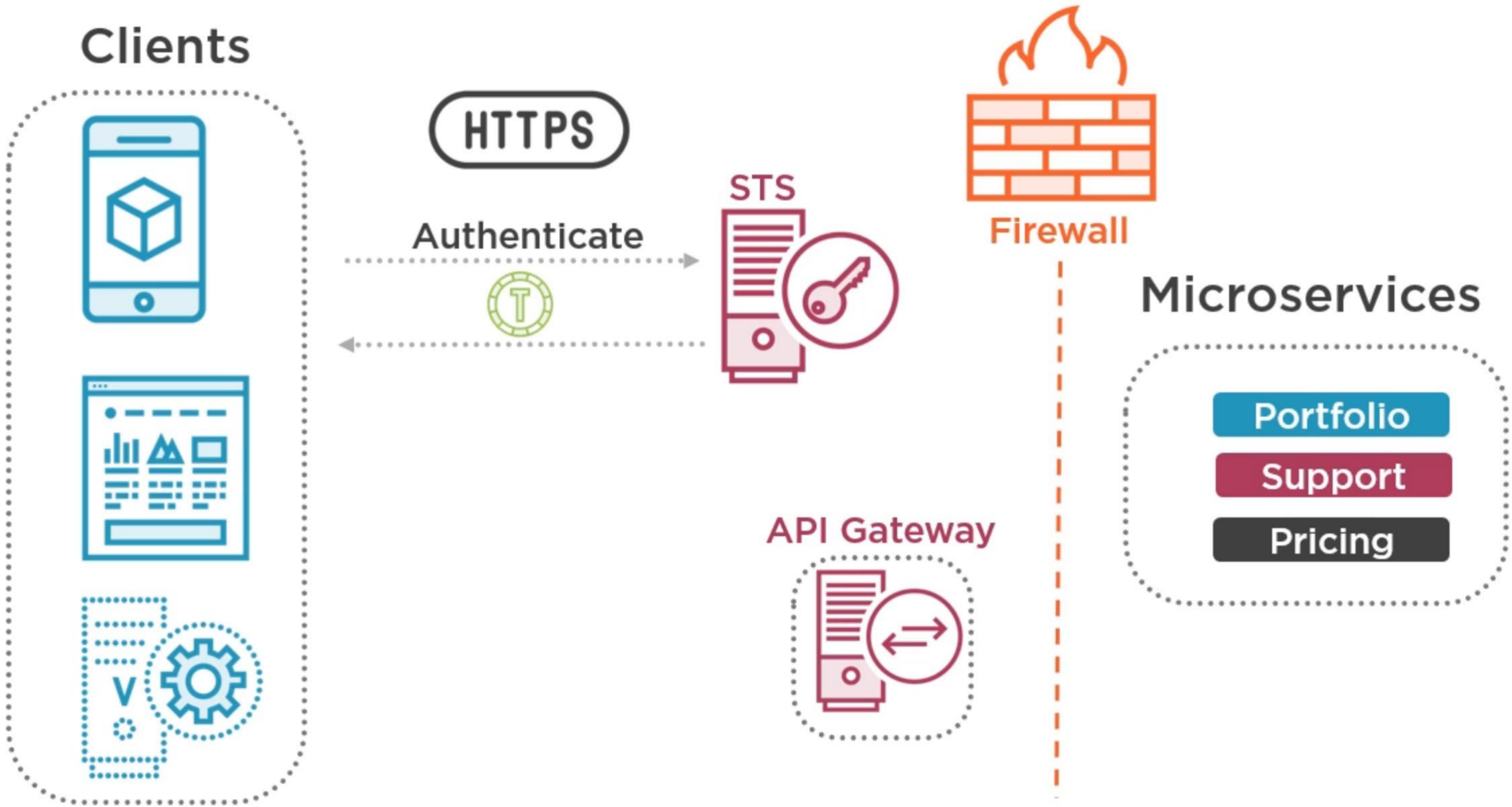
Does not solve the delegated access or the confused deputy problem.

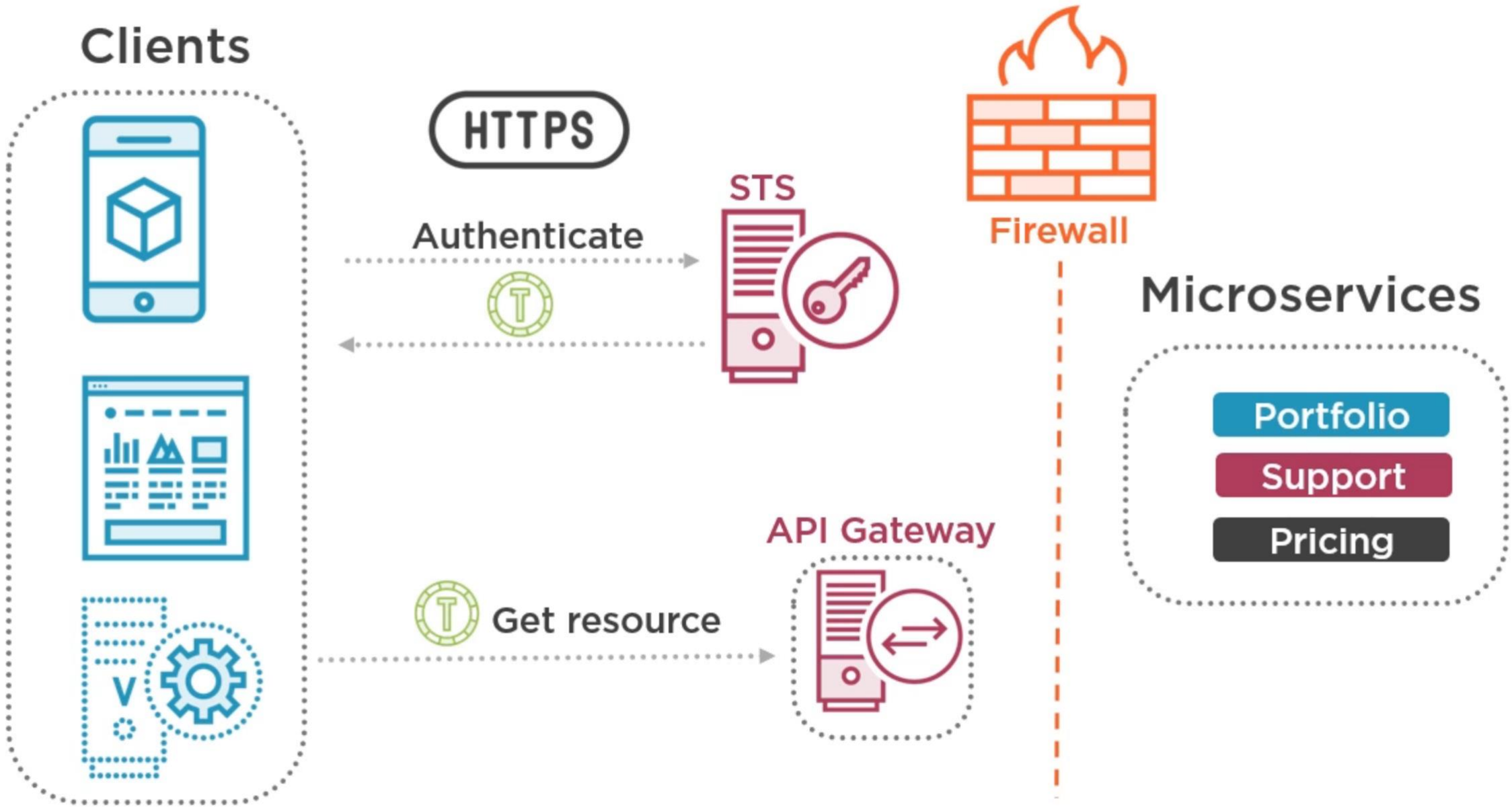
You have to maintain a Public Key Infrastructure and your Certificate Authority.

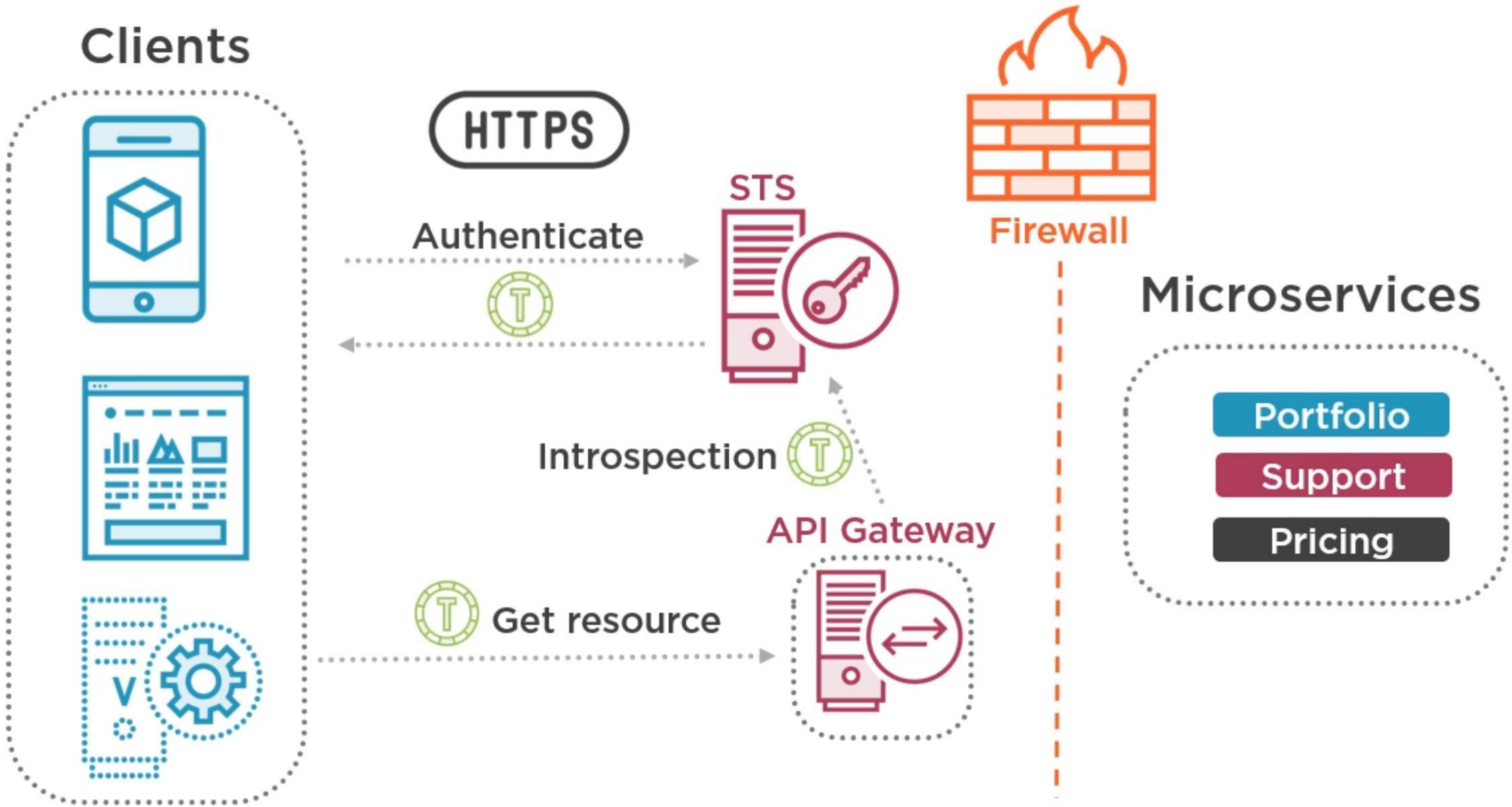
Key provisioning, bootstrapping, and rotation, and invalidating keys can be challenging as the number of clients and services increases.

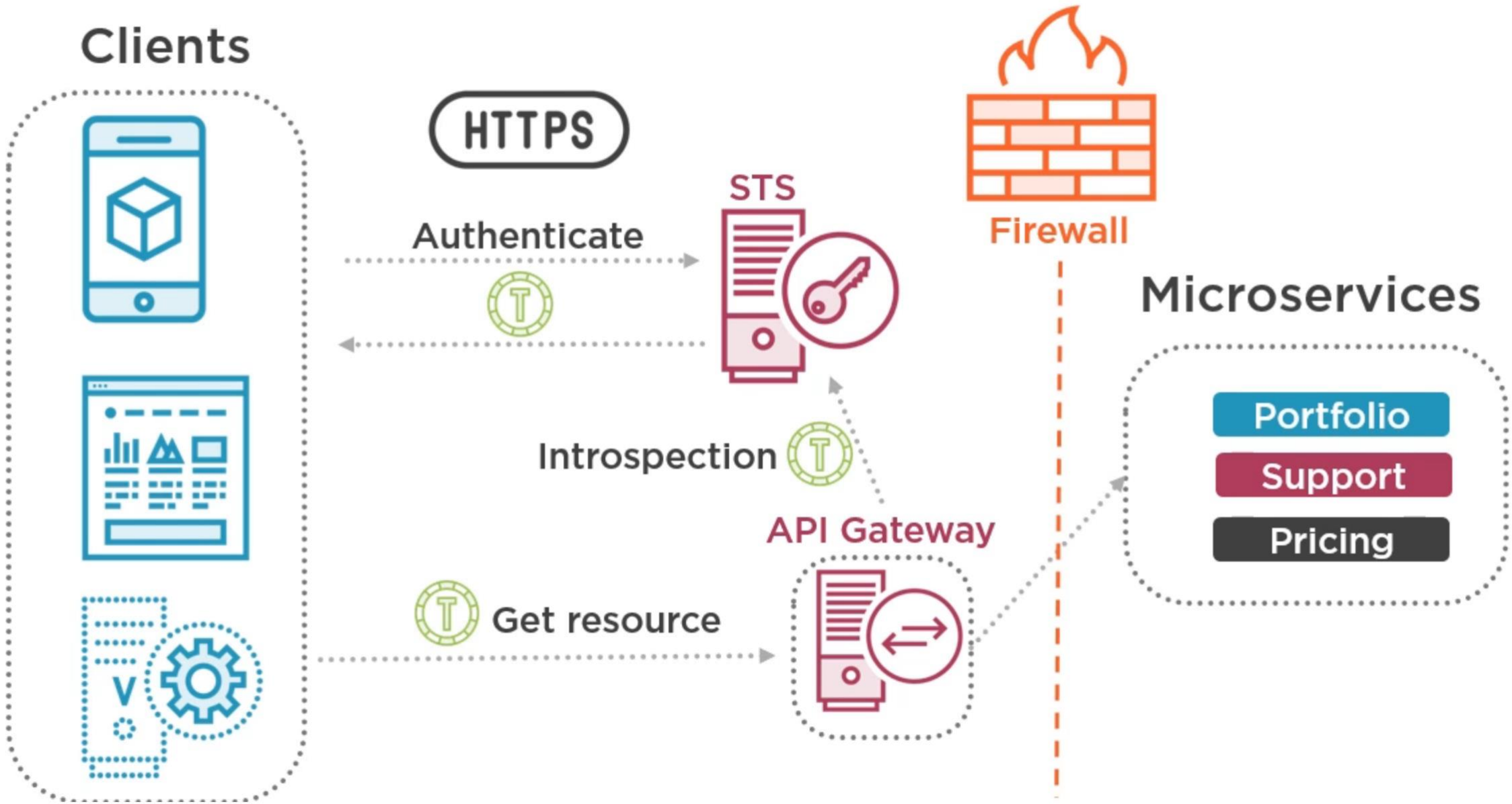
Introducing Tokens


Clients
HTTPS
Firewall
Microservices
Portfolio
Support
Pricing
API Gateway

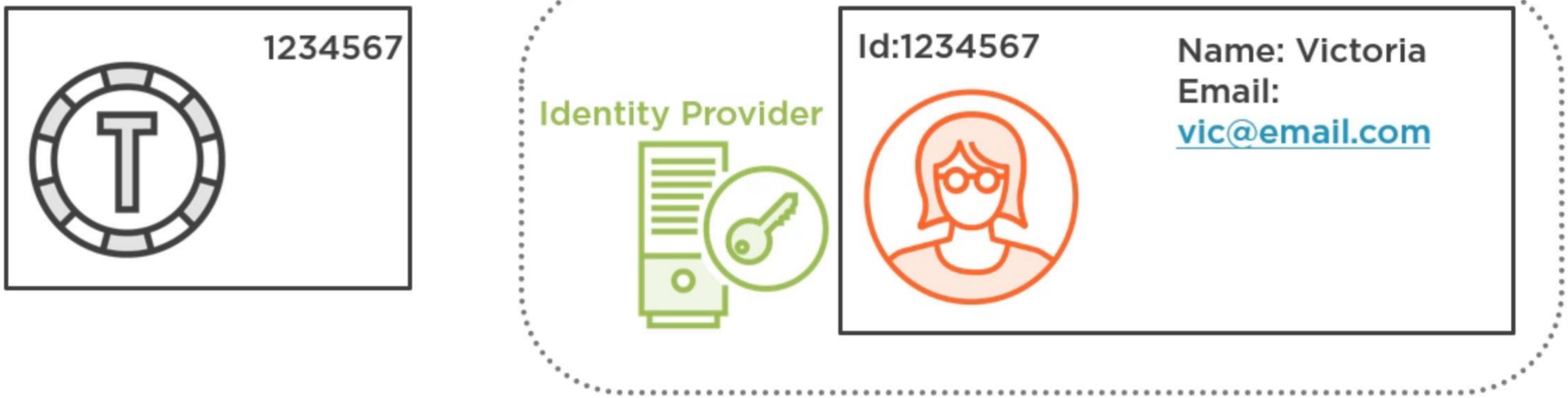

Clients
HTTPS
Authenticate
STS
Firewall
Microservices
Portfolio
Support
Pricing
API Gateway


Clients
HTTPS
STS
Authenticate
Firewall
Microservices
Portfolio
Support
Pricing
API Gateway
Get resource

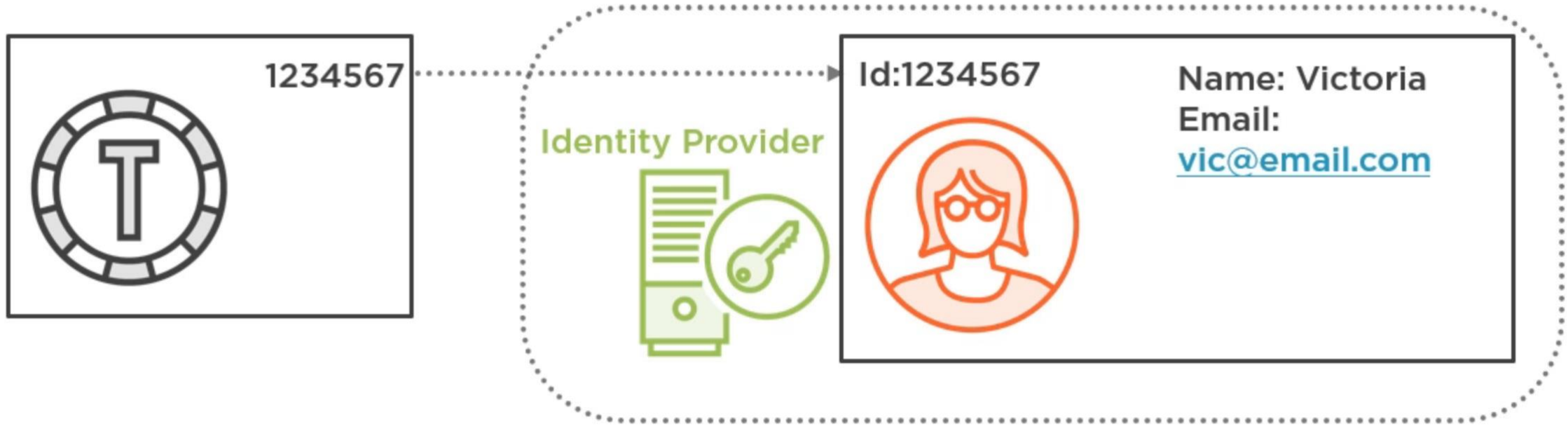

Clients
HTTPS
STS
Authenticate
Firewall
Microservices
Portfolio
Support
Pricing
Introspection
API Gateway
Get resource


Clients
HTTPS
STS
Authenticate
Firewall
Microservices
Portfolio
Support
Pricing
Introspection
API Gateway
Get resource

Opaque Tokens – by Reference

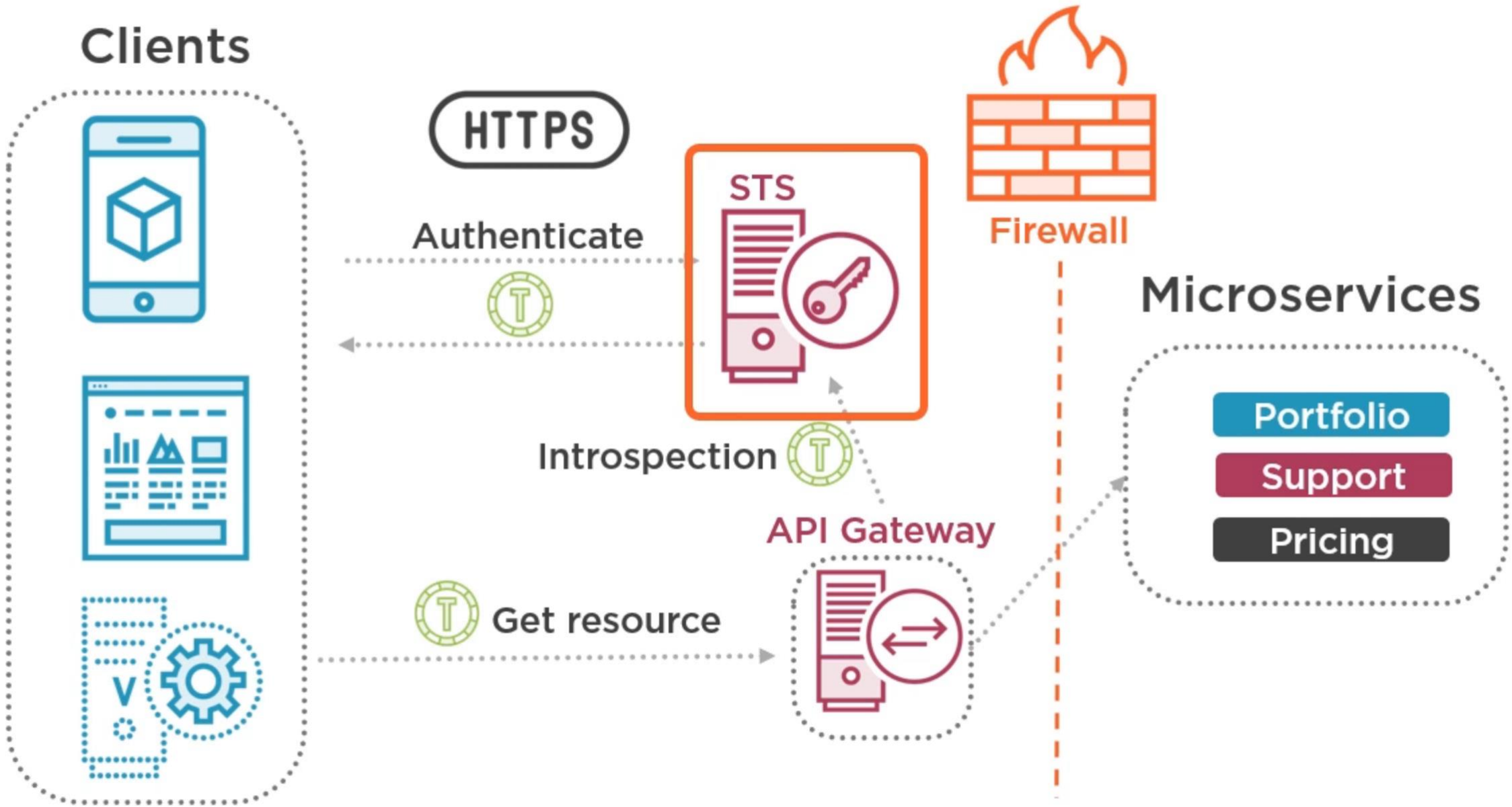


Opaque Tokens – by Reference



Bearer Token

A security token with the property that any party in possession of the token (a "bearer") can use the token in the same way that any other party with possession can.


Clients
HTTPS
Authenticate
STS
Firewall
Microservices
Portfolio
Support
Pricing
Introspection
API Gateway
Get resource

Benefits of Tokens

Microservices are never exposed to the users or clients password.

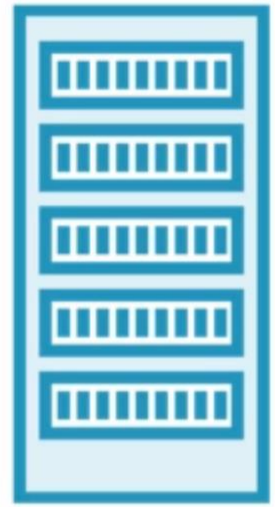
"Single responsibility Principal" your microservices development teams don't need to focus on non-functional security requirements.

Tokens can be used to provide delegated authorization.

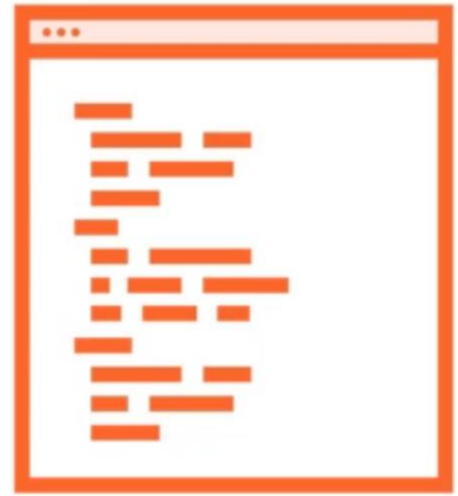
Identity Provider

Subject – Entity Requesting Access

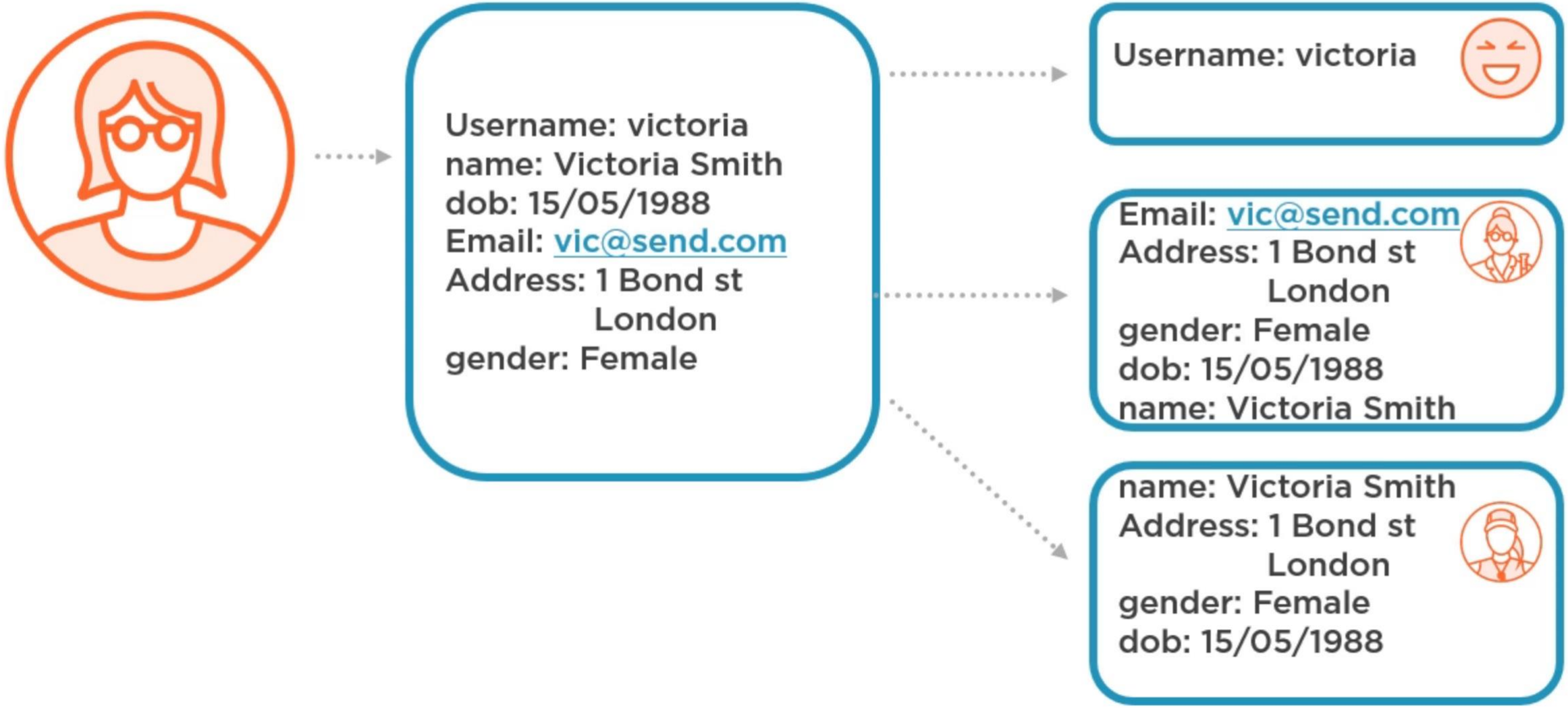
Human
user like Victoria



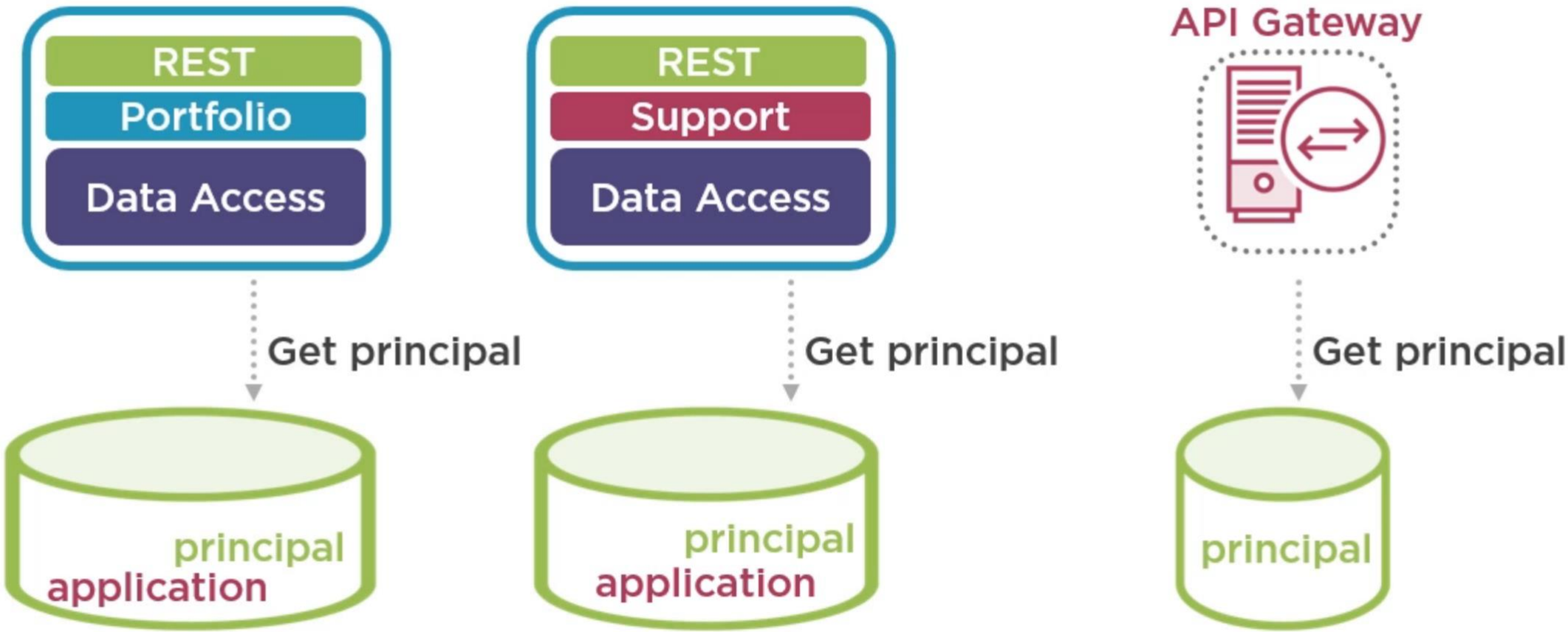
Machine



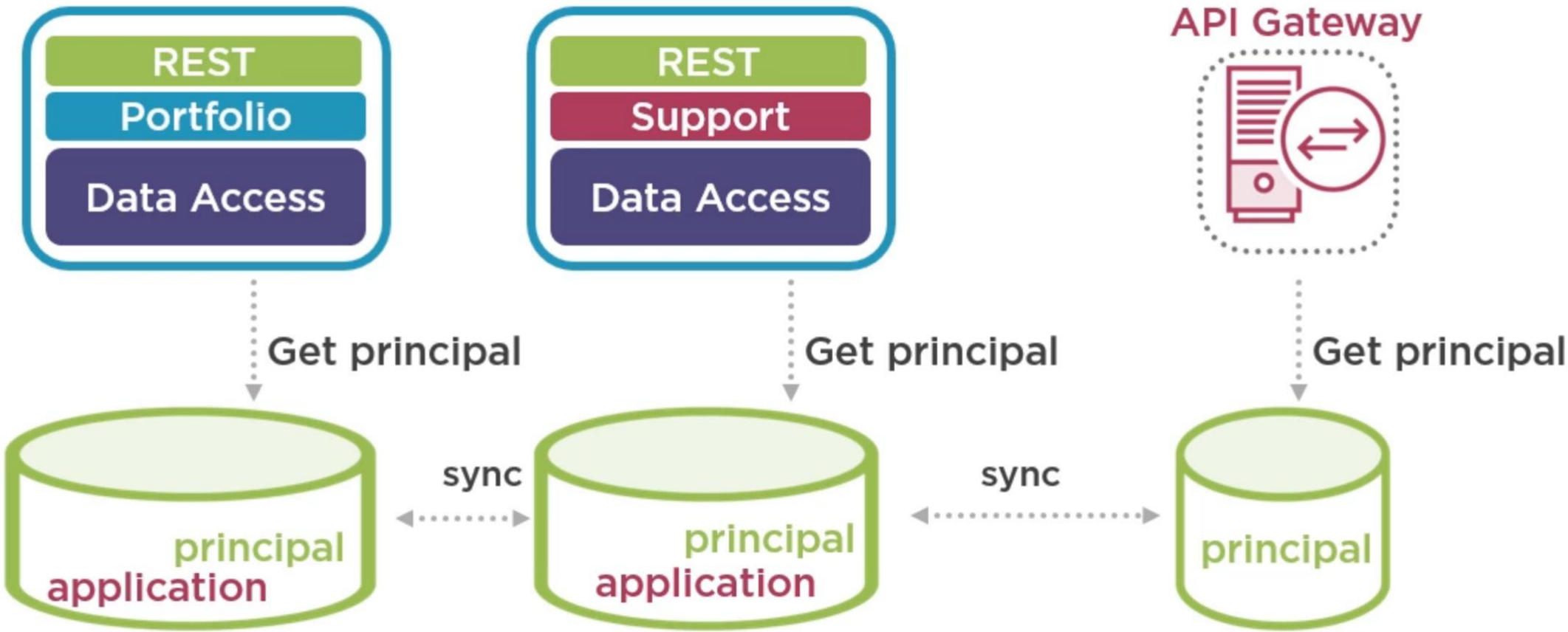
Software


Subject
Principals
Username: victoria
name: Victoria Smith
dob: 15/05/1988
Email: vic@send.com
Address: 1 Bond st
London
gender: Female
Username: victoria
Email: vic@send.com
Address: 1 Bond st
London
gender: Female
dob: 15/05/1988
name: Victoria Smith
name: Victoria Smith
Address: 1 Bond st
London
gender: Female
dob: 15/05/1988

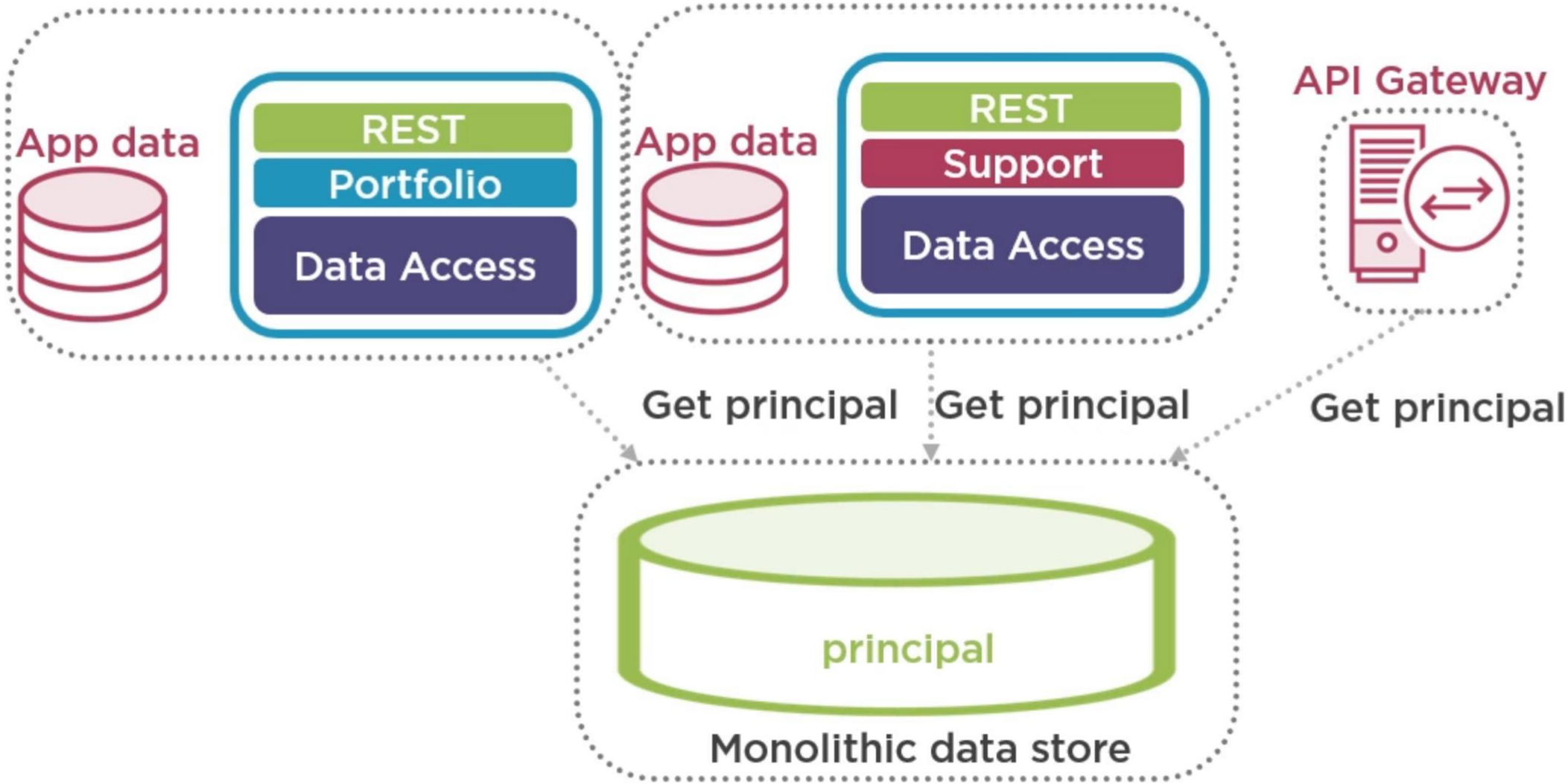
Coupling Application and Identity data



Coupling Application and Identity data



Microservices (Except for the Data)



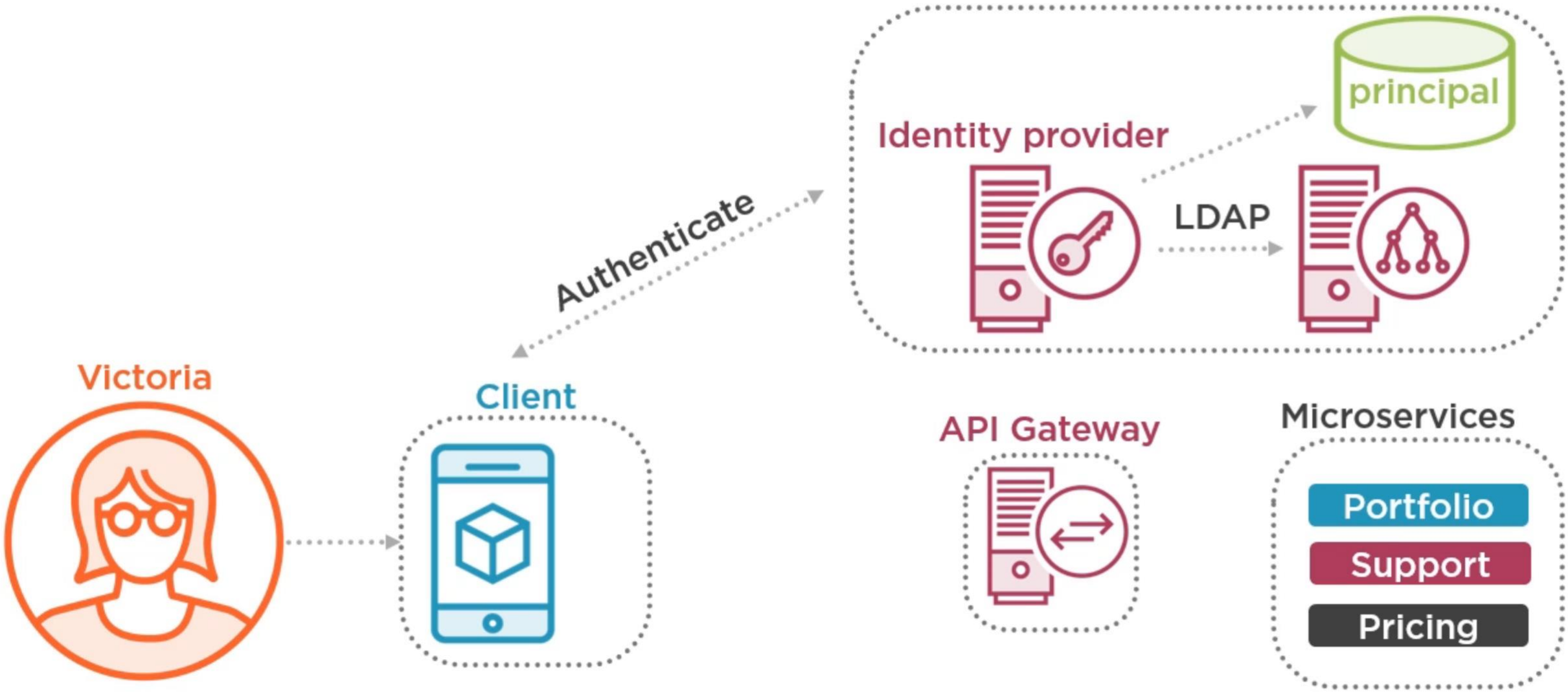
Identity Provider

A service that manages the creation, maintenance of identity information for principals.

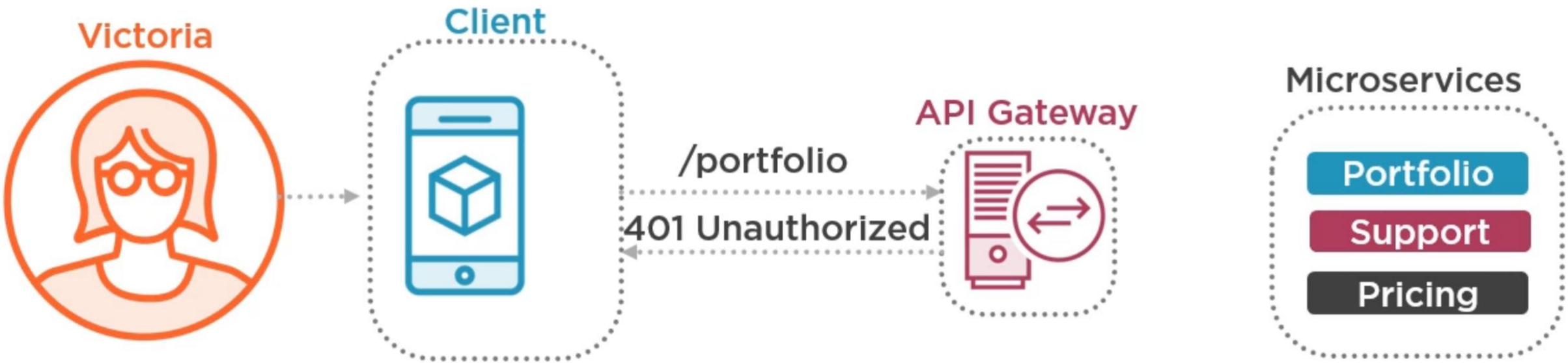
Authentication as a service (AaaS)

Manage and provider single sign in for all users in your applications, systems, and networks centrally.

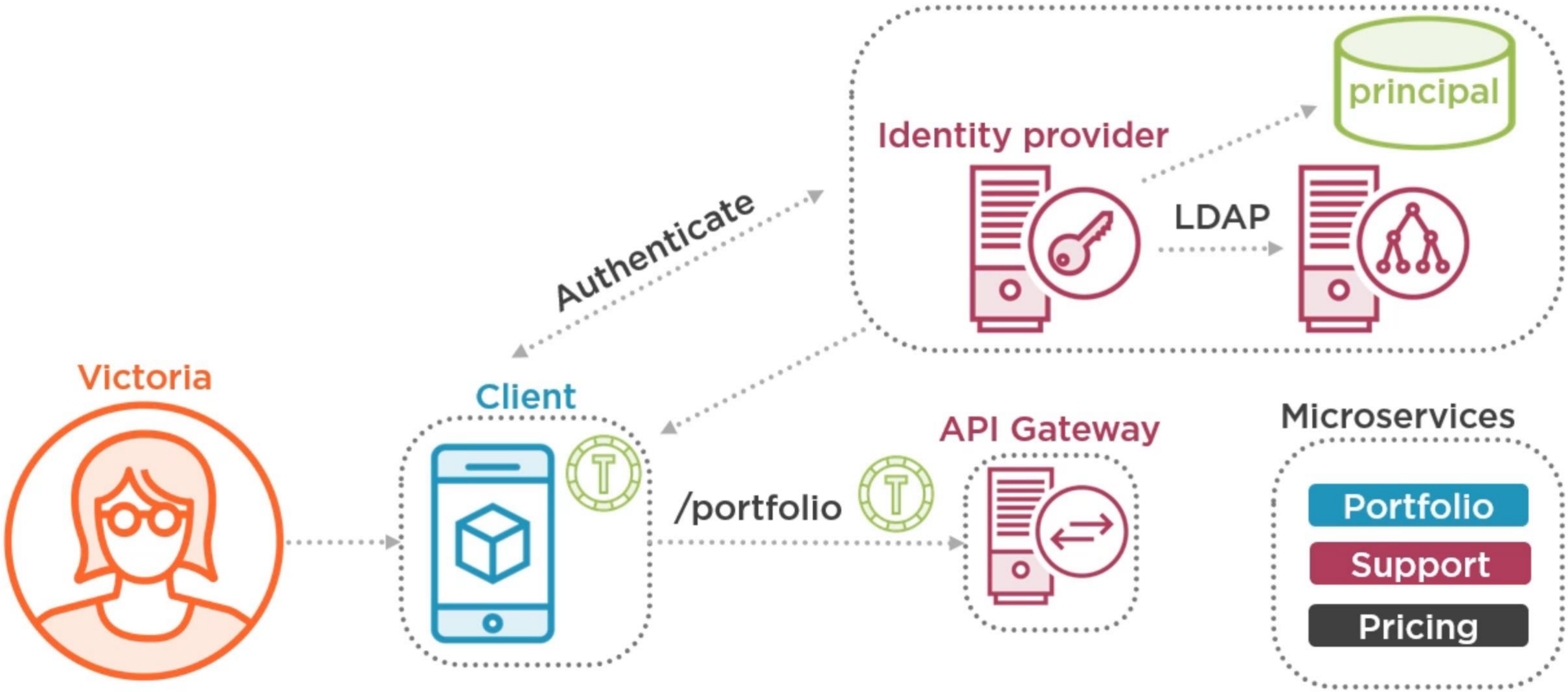
Relaying Party



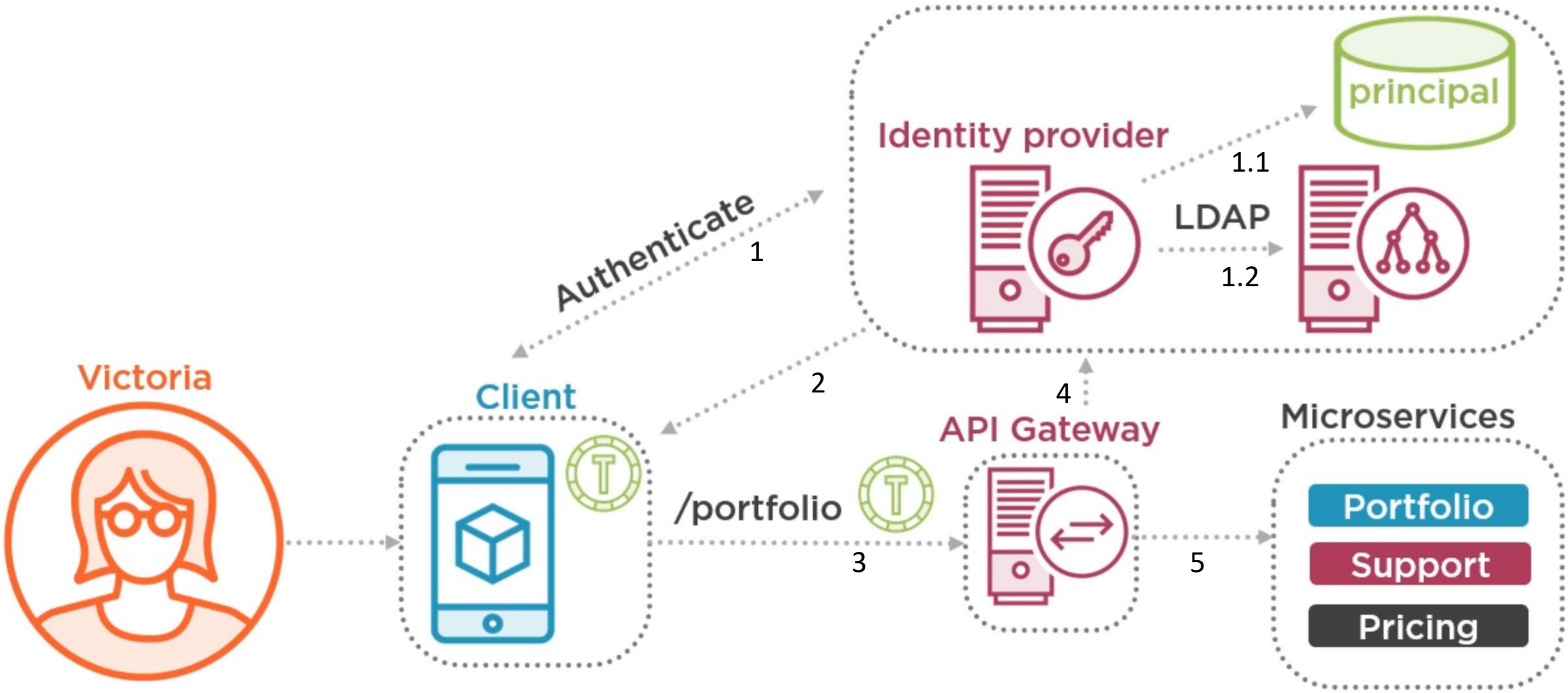
Identity Provider



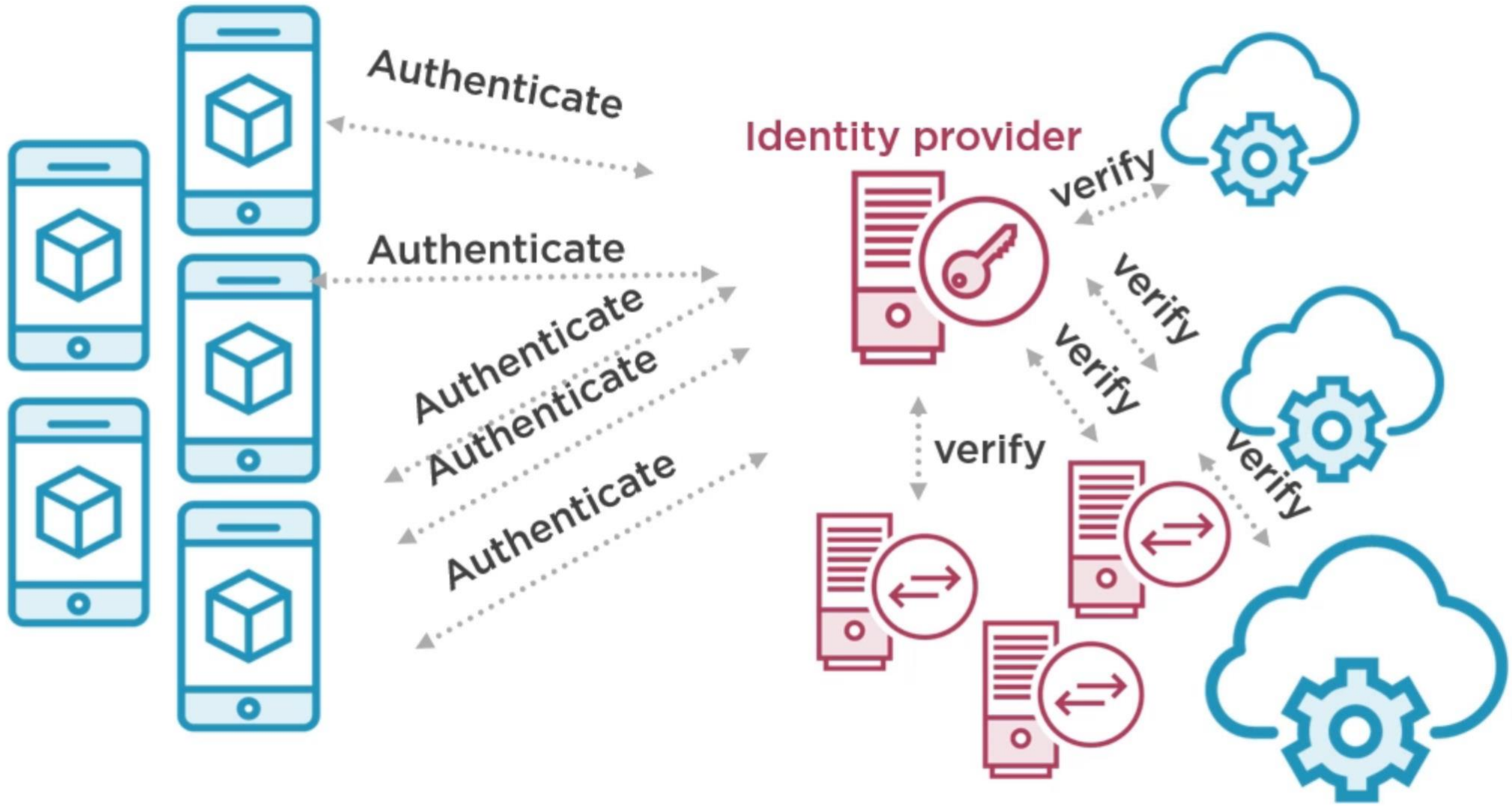
Relaying Party



Relaying Party



Identity Provider Can Become a Bottleneck



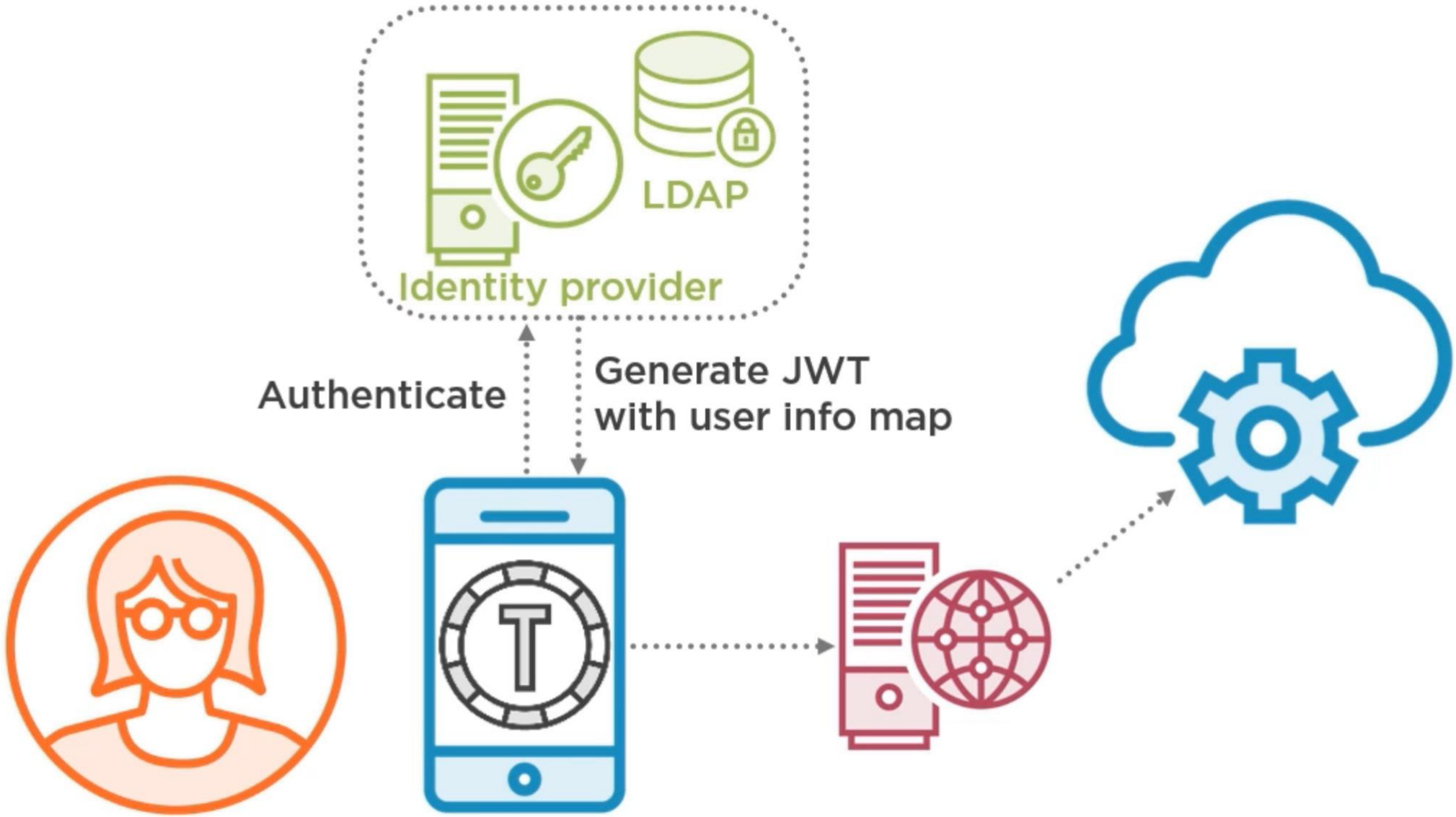
By-value Tokens

By-value Token

Claims-based identity

Claims define what the subject is and is not.

By-value Token


LDAP
Identity provider
Authenticate
Generate JWT
with user info map

A signed token without the expiration date is worse than a password.

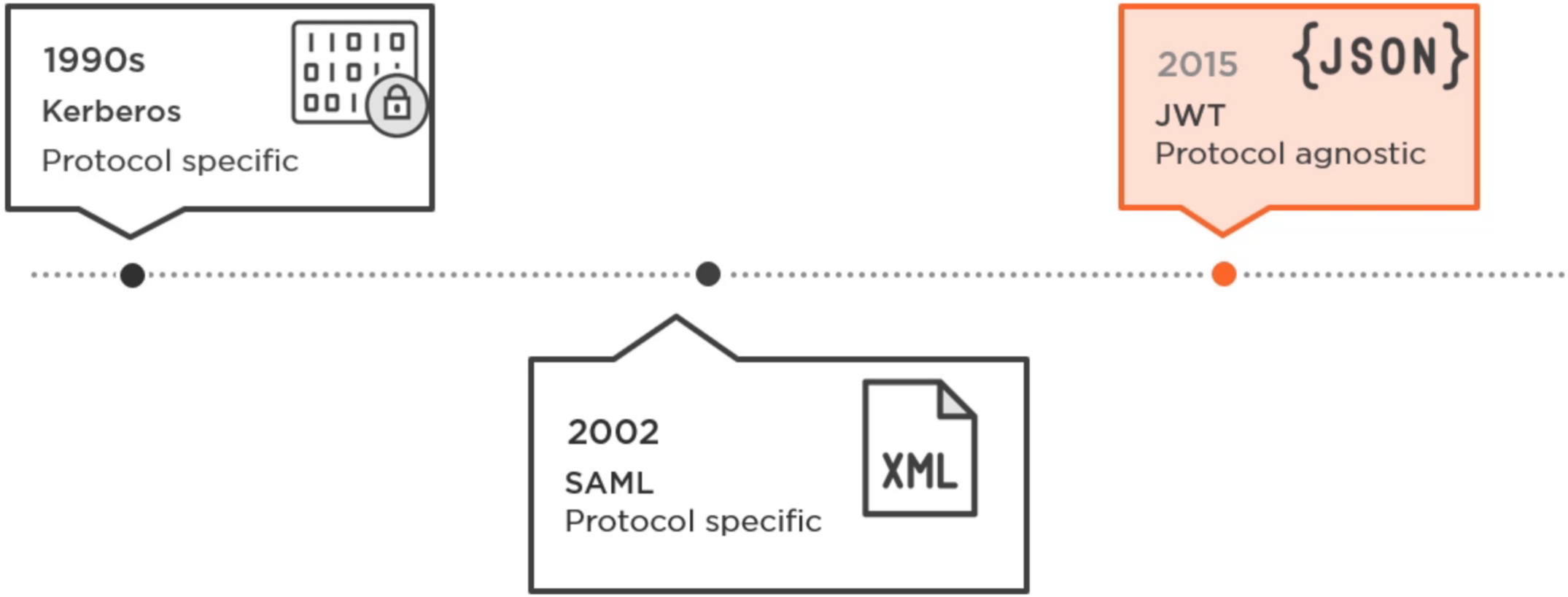
What format should your by-value token be?

Answer

A standard token format.

Different Token Formats Introduce Complexity

Evolution of Security Tokens



JSON Web Token (JWT)

JWT

{JSON}

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e
yJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6I
lZpY3RvcmEgU21pdGgiLCJyb2xlIjoiYWRta
W4iLCJpYXQiOjE1NTg2MTI4MTAsImV4cCI
6MTU1ODYxNjU0OCwiZW1haWwiOiJ2aWN
0b3JpYUBleGFtcGxlLmNvbSIsInBpY3R1cm
UiOiJodHRwOi8vZXhhbXBsZS5jb20vdmljc
21pdGgvbWUuanBnIiwianRpIjoiYzRkMmRi
ZmMtZWRiOS00MWZiLWE1N2QtNjk1ZWIx
OGE3MjAwIn0.z0WXlynyxUVAfphjA9ckw7
2o5yhMqJuekijZbxCvYM0

Header

```
{
 "typ": "JWT",
 "alg": "HS256"
}
```

Body

```
{
 "sub": "1234567890",
 "name": "Victoria Smith",
 “scopes": ["admin“],
 "iat": 1558612810,
 "exp": 1558616410,
 "email": "victoria@example.com",
 "picture": "http://example.com/vicsmith/me.jpg"
}
```

Signature

z0WXlynyxUVAfphjA9ckw72o5yhMqJuekijZbxCvYM0

JWT Optional Standard Claims

Claim	Description
jti	Unique token identifier
iss	Issuer – who issued the token
sub	Subject – Who the claims represent.
aud	Audience – The recipient the token was meant for.
exp	Expiration – Until when the token is valid
iat	Issued at – The time the token was issued.

The signature is used to verify that the sender of the JWT

Javascript Object Signing and Encryption (JOSE)

Delegated Authorization with OAuth2

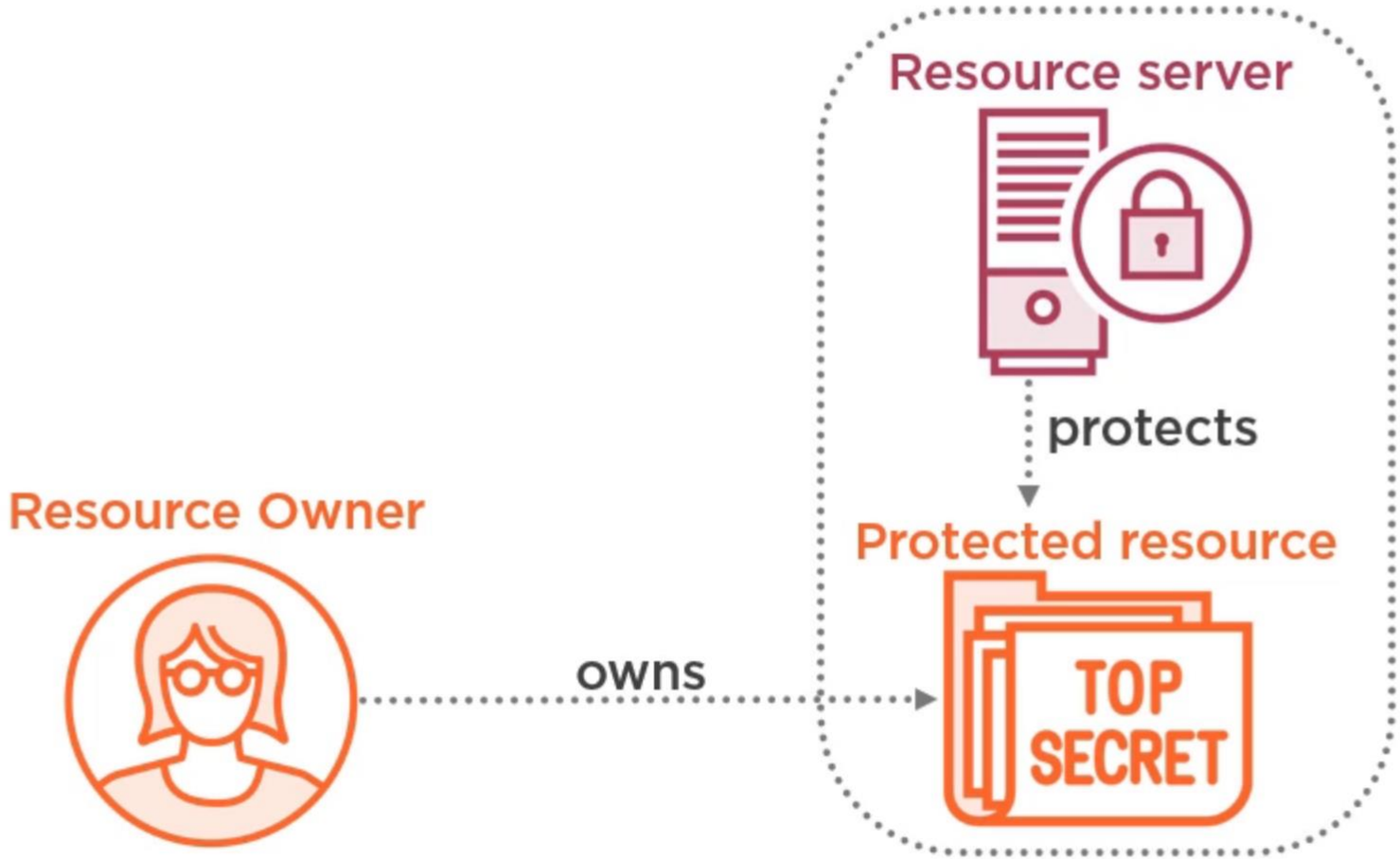
Open Authorization (OAuth2)

OAuth2 is an open standard used to allow entities to grant limited access to their resources, without sharing their credentials.

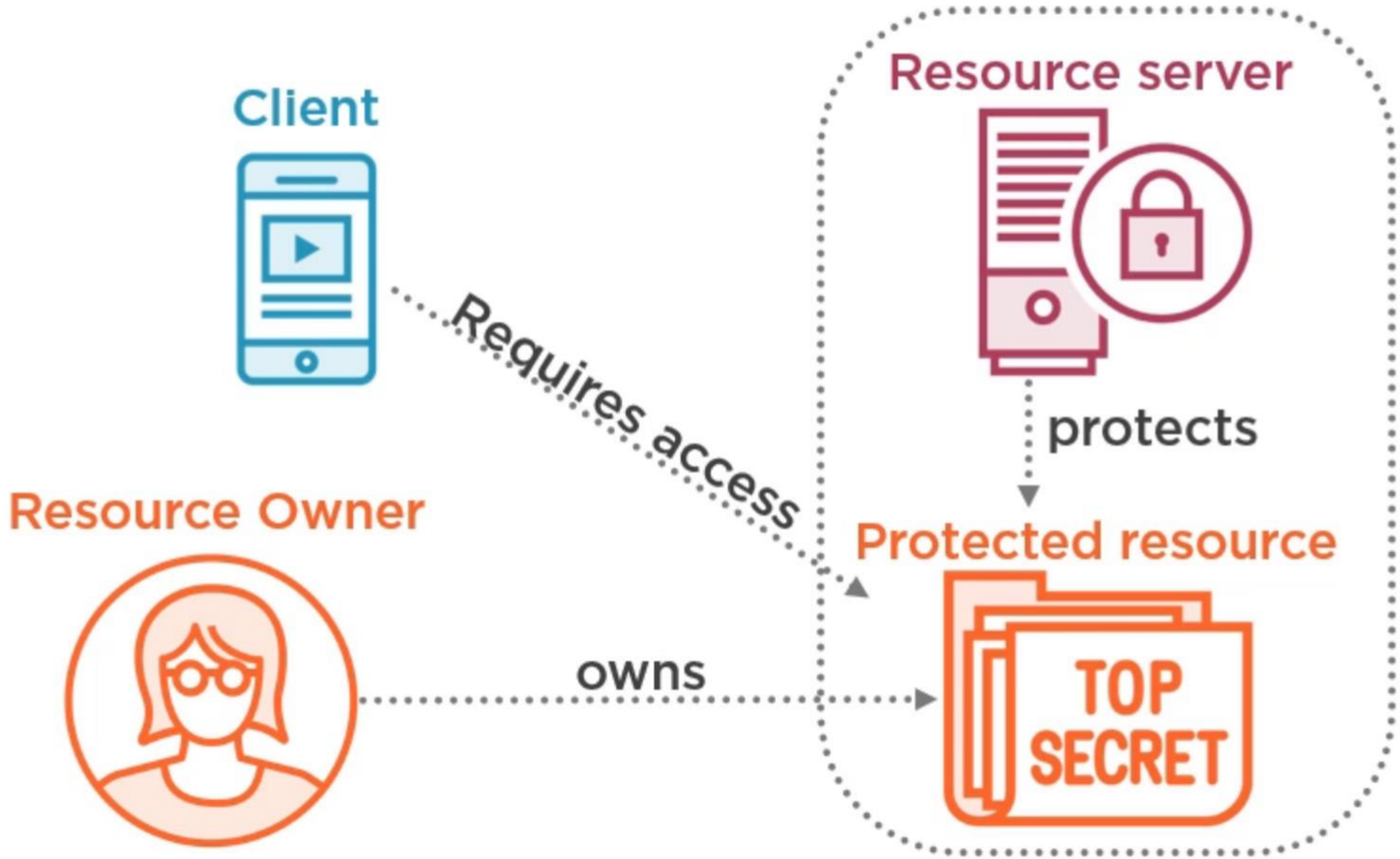
Actors in OAuth

Actors in OAuth

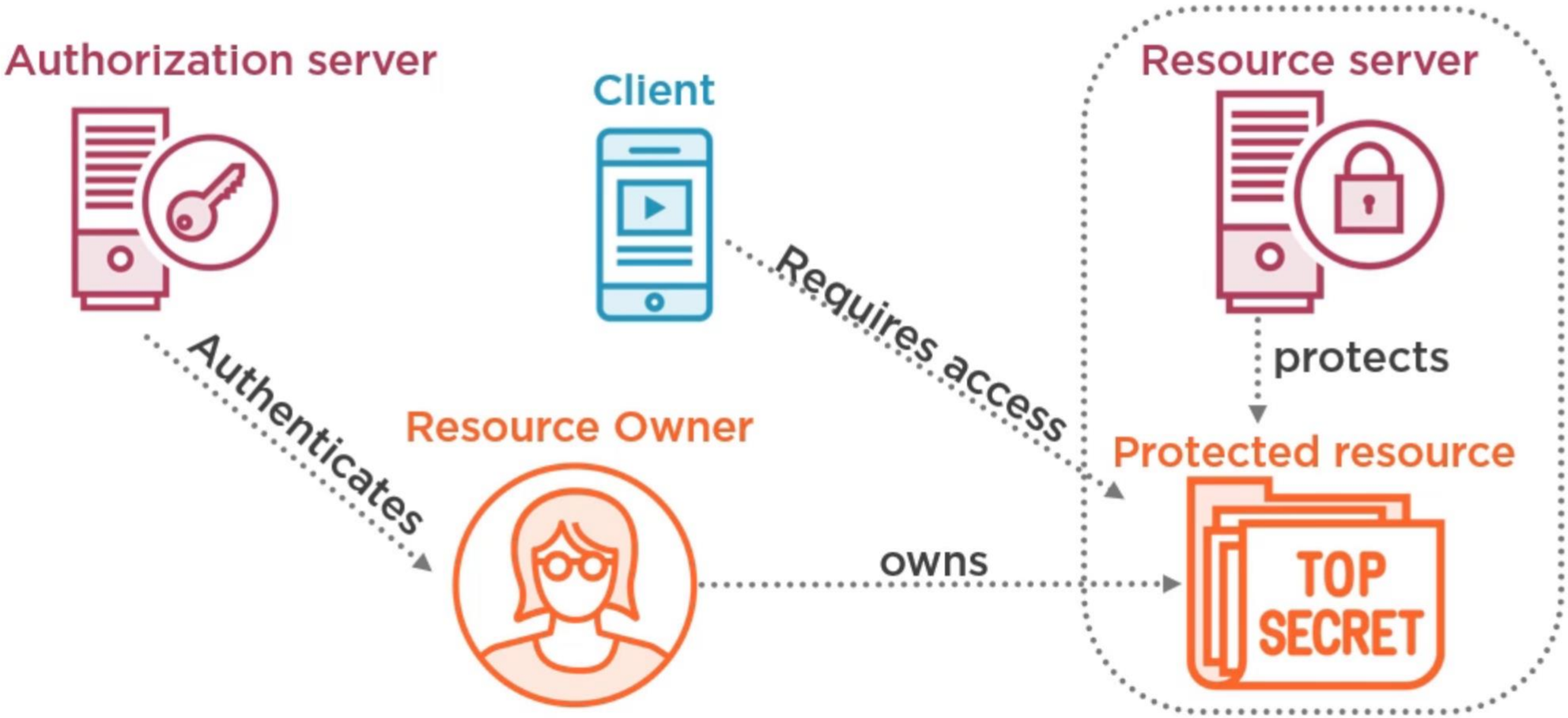
Actors in OAuth



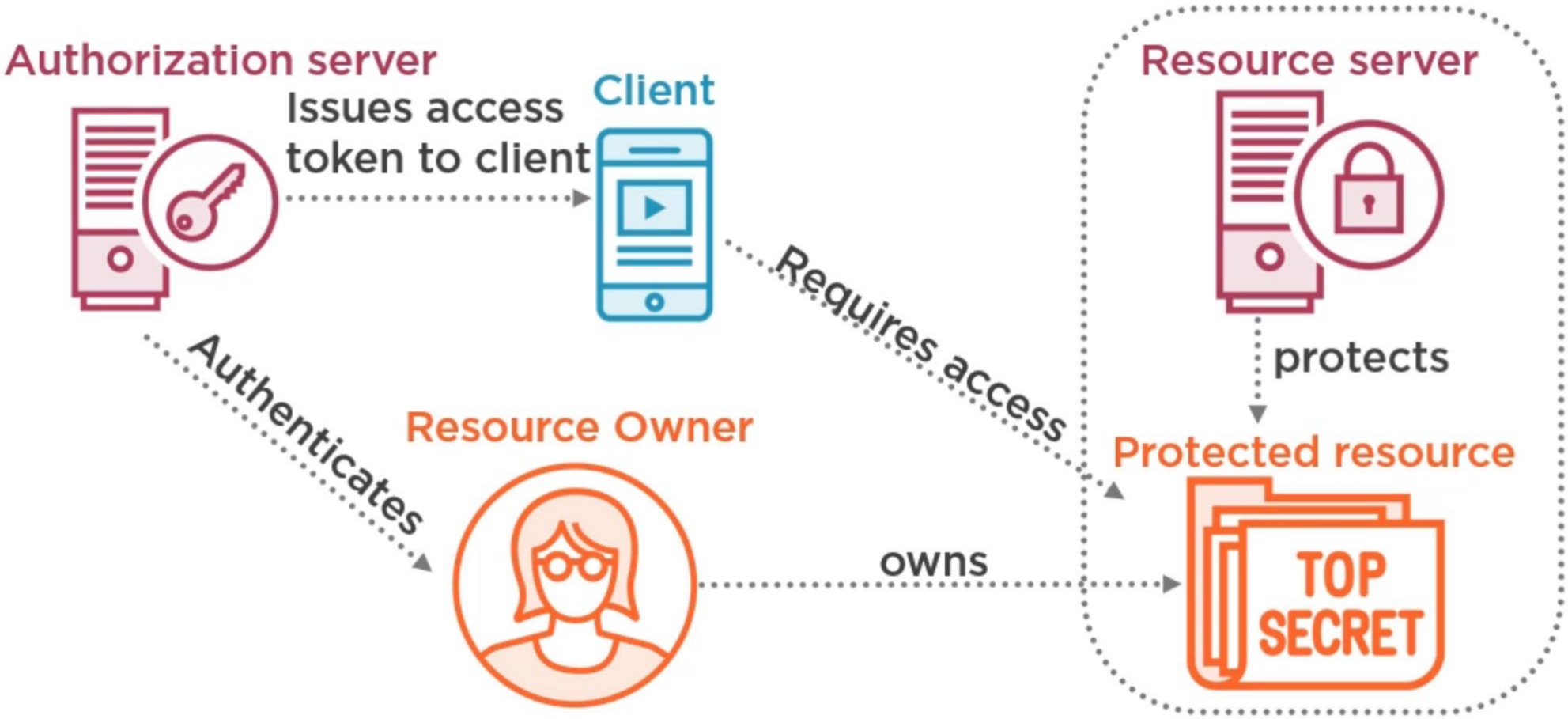
Actors in OAuth



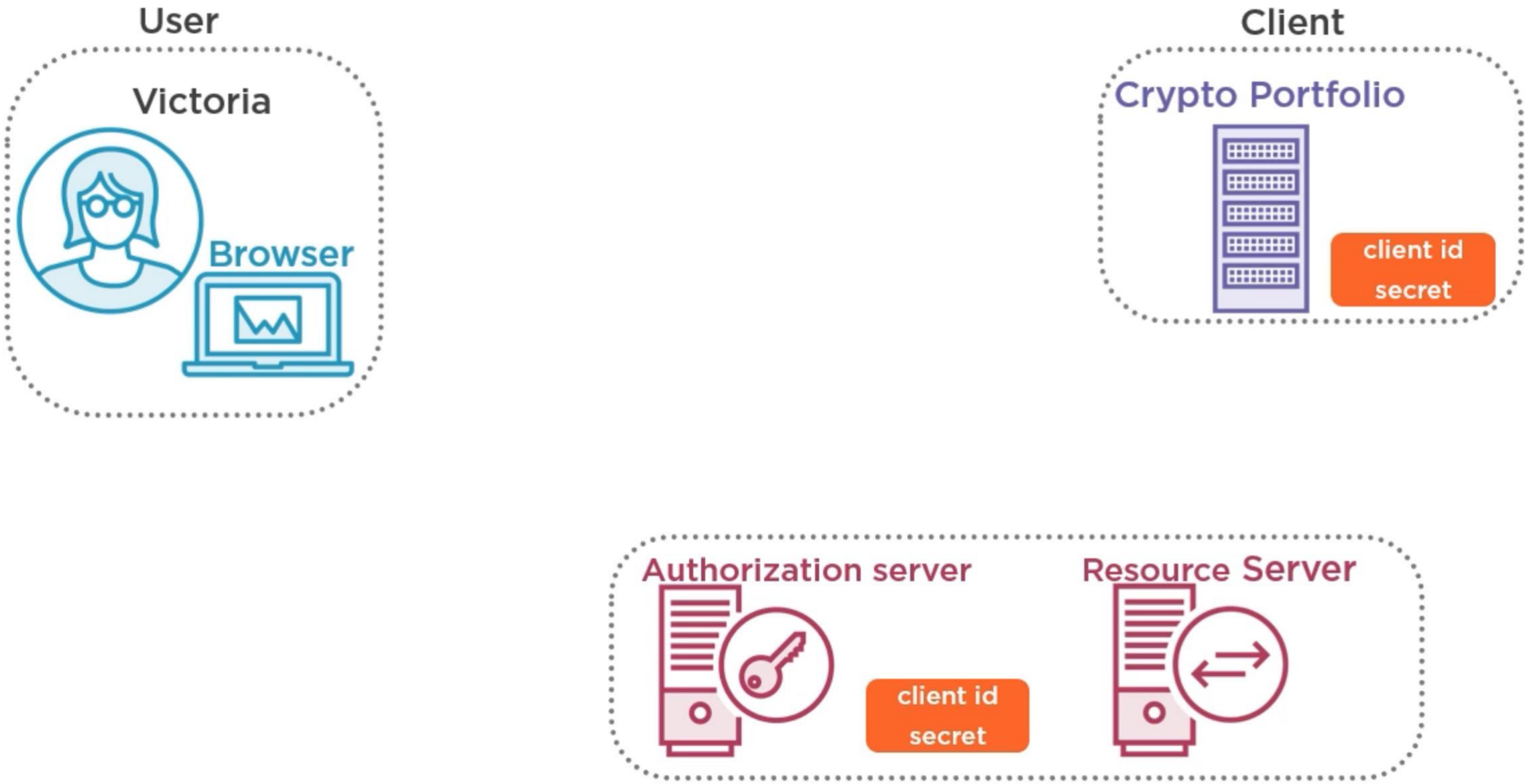
Actors in OAuth



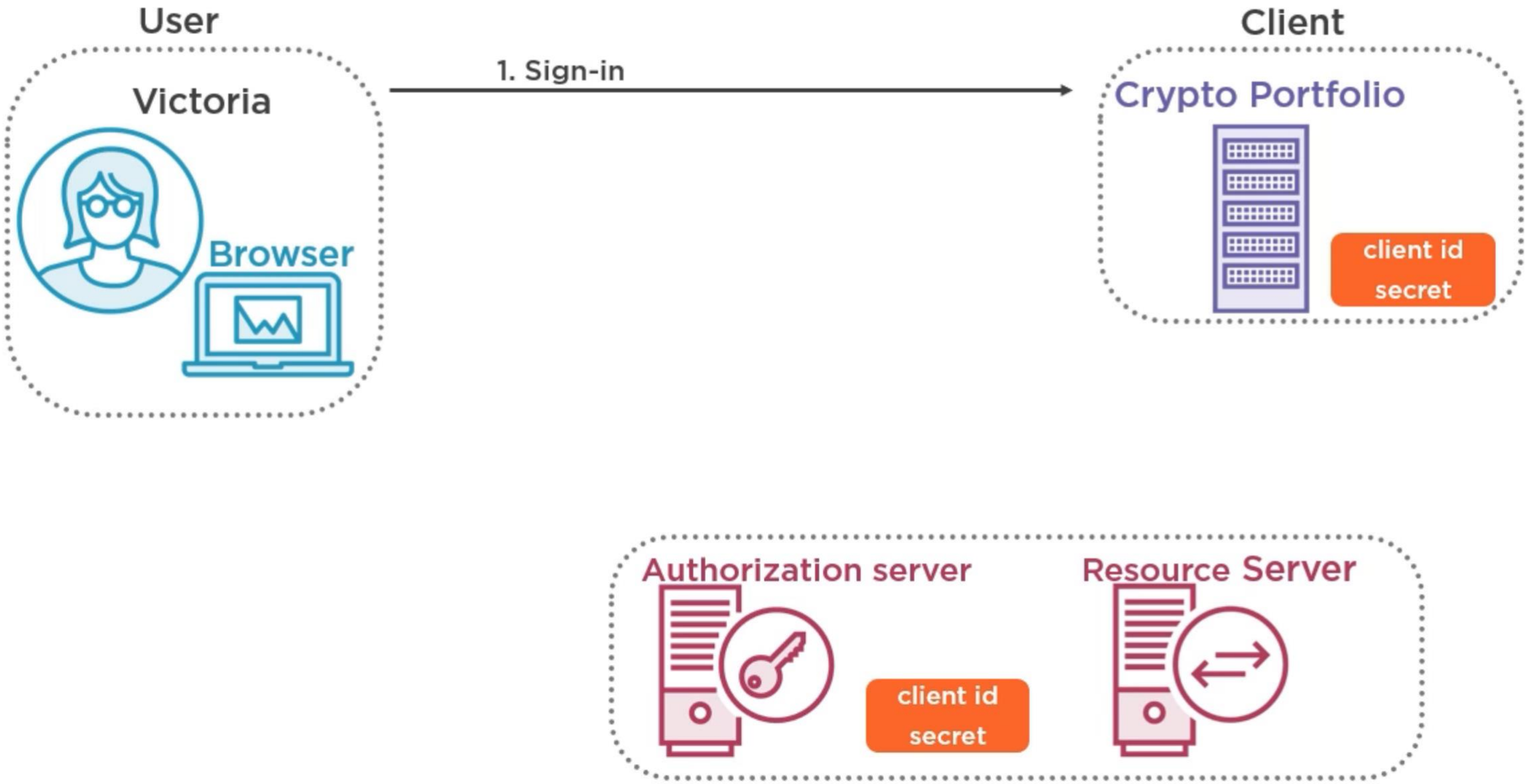
Actors in OAuth



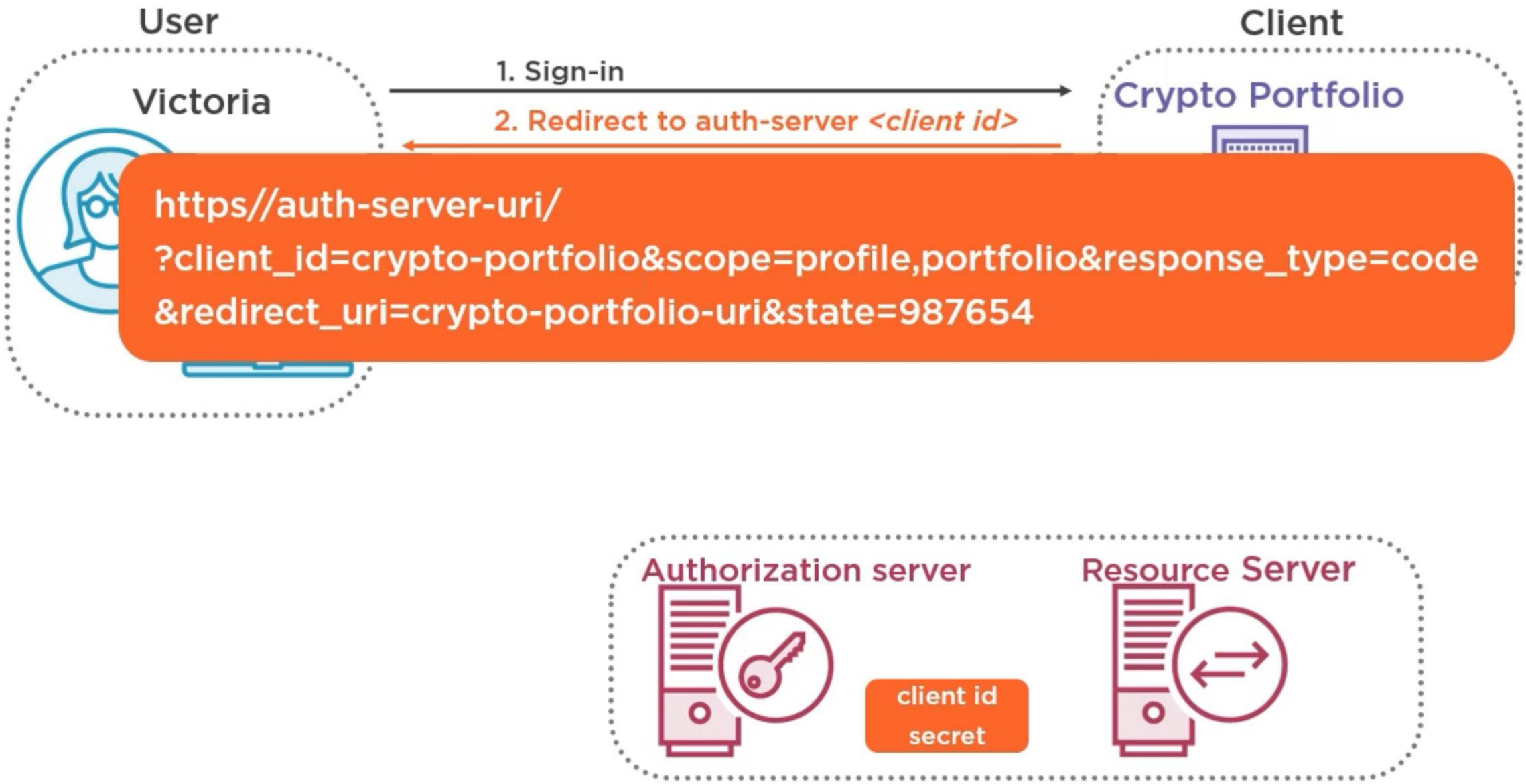
OAuth2 Authorization Code Grant



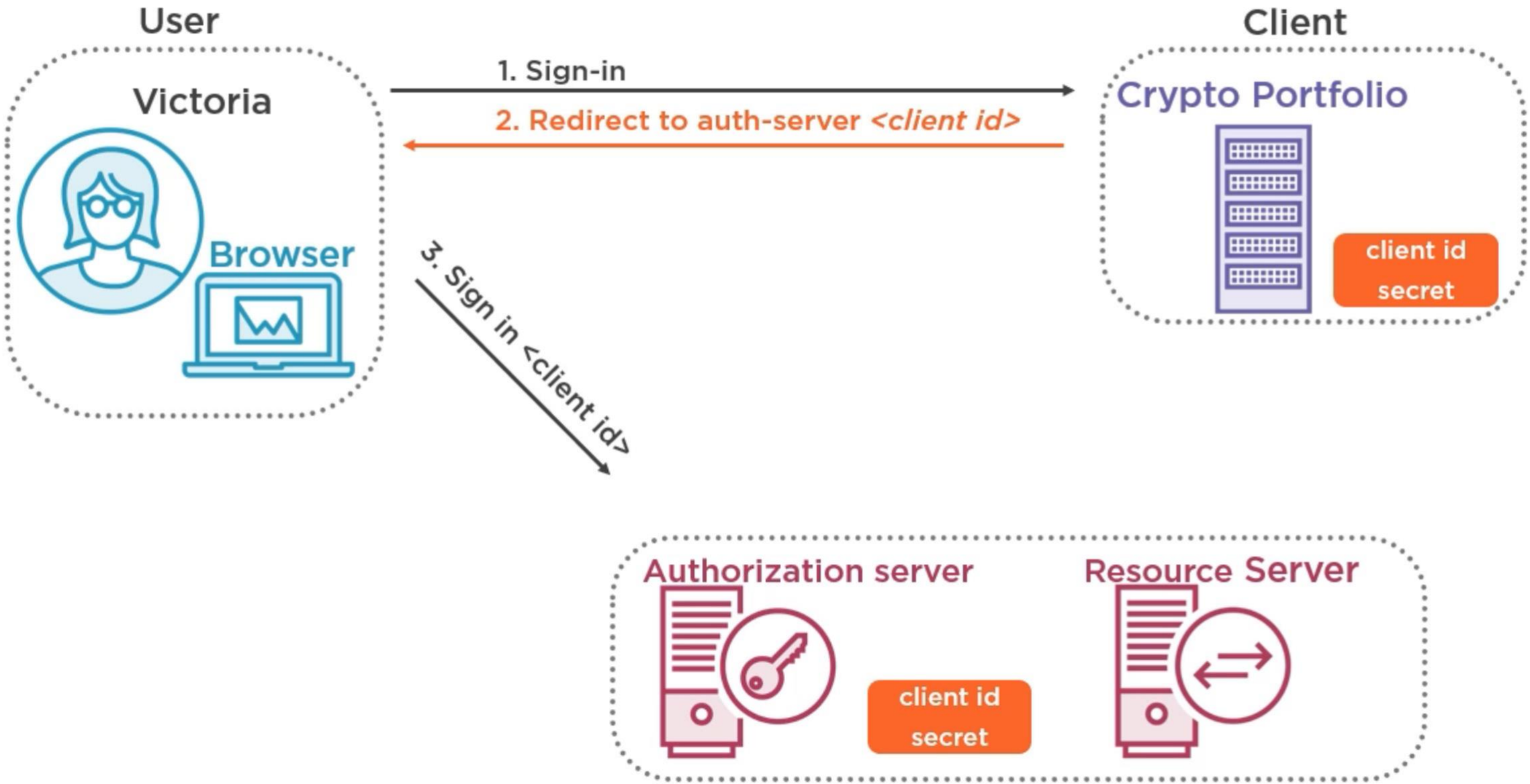
OAuth2 Authorization Code Grant



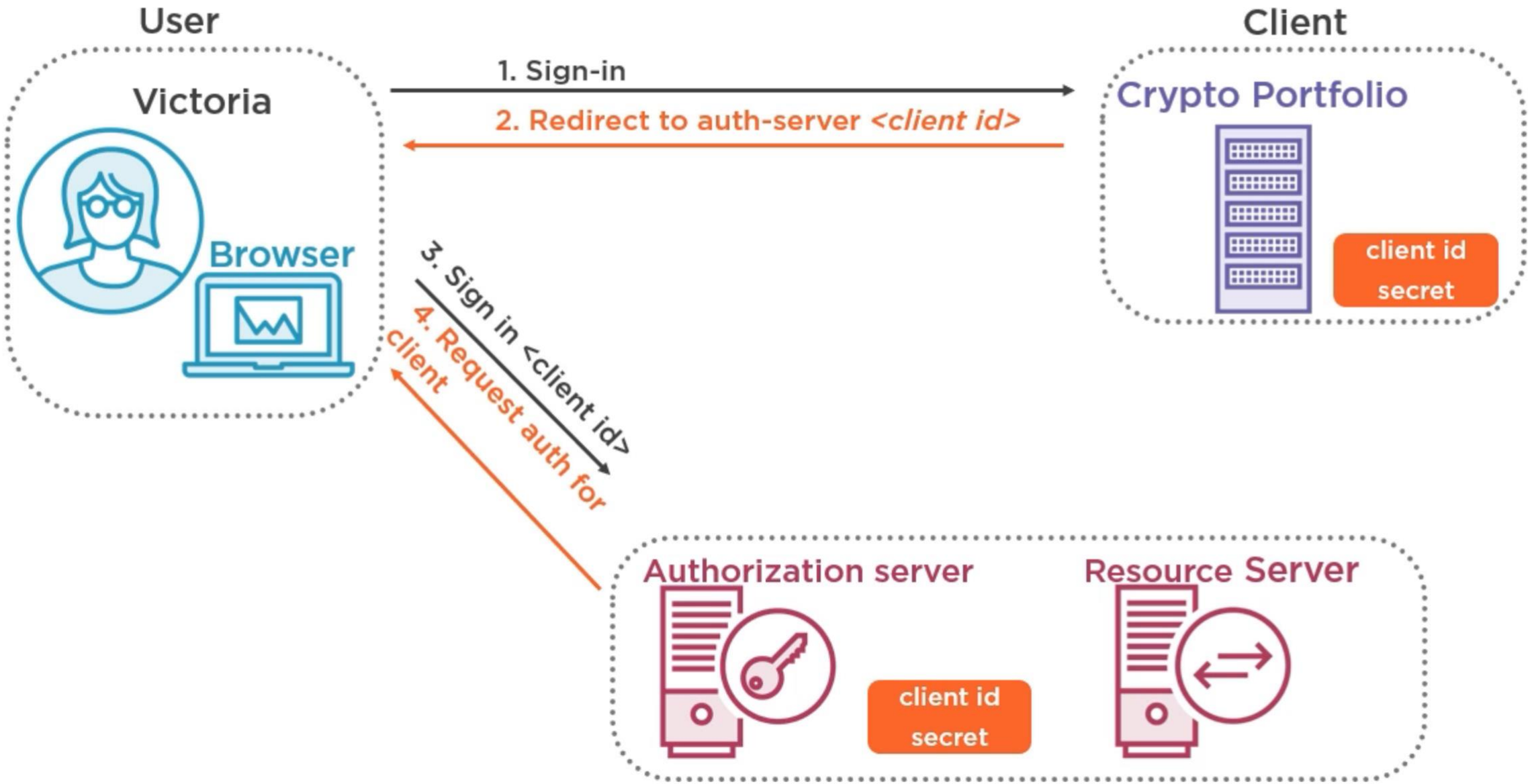
OAuth2 Authorization Code Grant



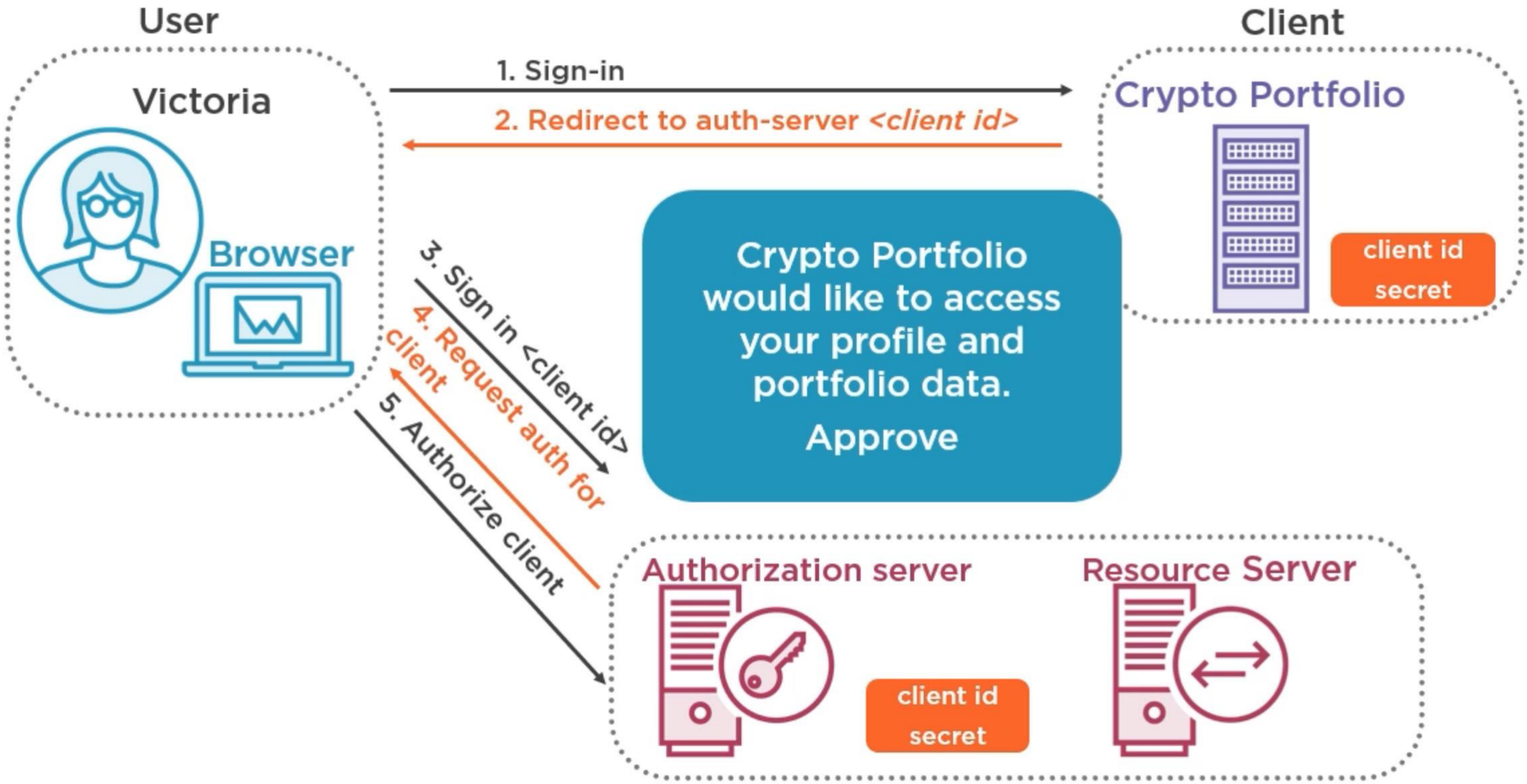
OAuth2 Authorization Code Grant



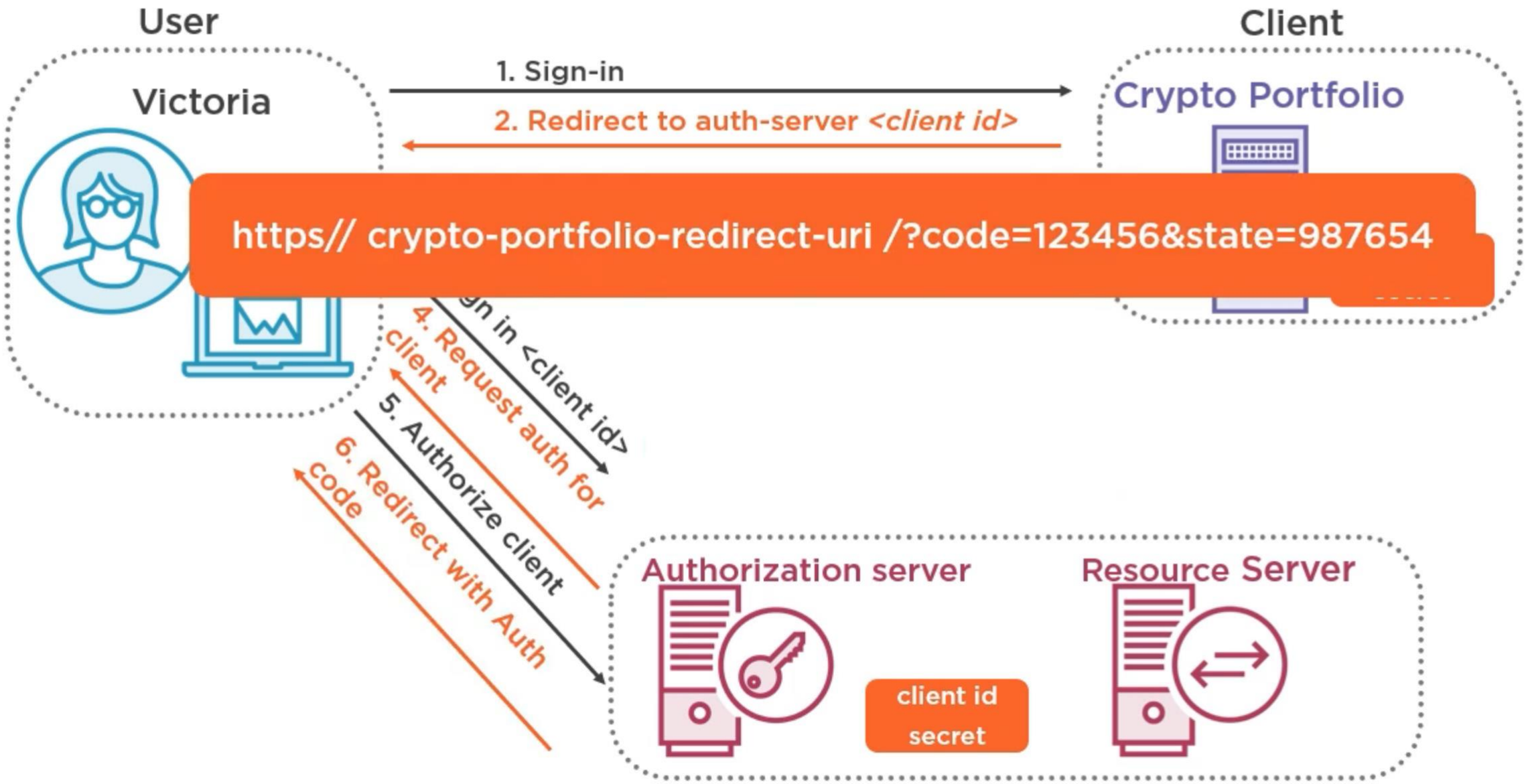
OAuth2 Authorization Code Grant



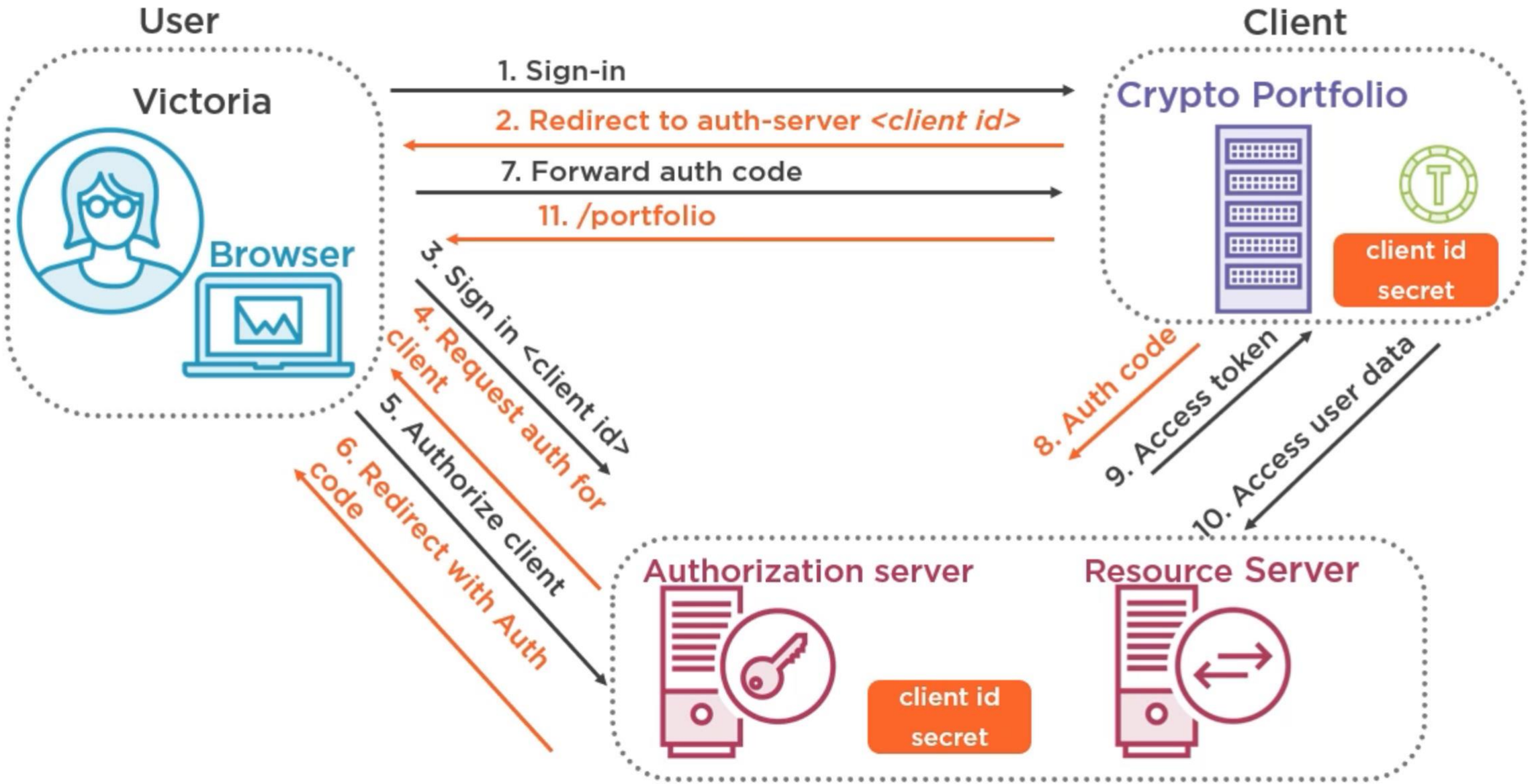
OAuth2 Authorization Code Grant



OAuth2 Authorization Code Grant



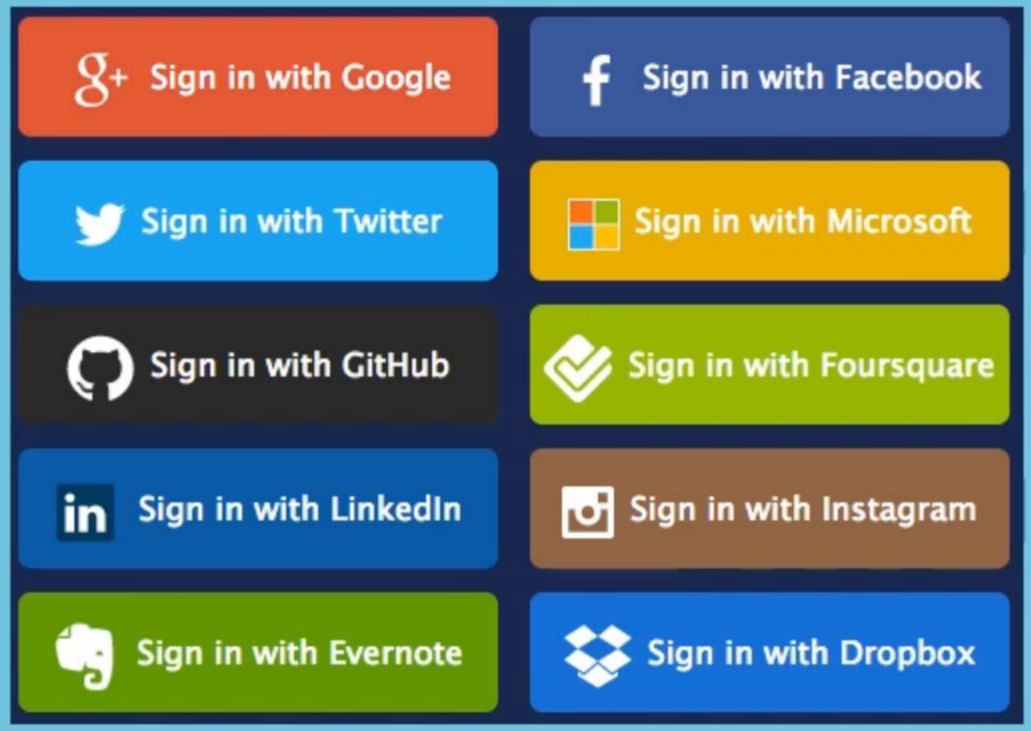
OAuth2 Authorization Code Grant



Other Grant Types

Client credentials

Sign-in with OAuth2



"OAuth is Not Authentication"

Oauth is actually for
"Delegated Authorization"
not “Authorization”

Options for API Gateway and Authorization Server

API Gateway Options

Spring Cloud Gateway

Identity Provider / Authorization Server

Keycloak

Cloud Foundry
UAA

Gluu

Opensource.

3M+ downloads.

Built on top of NGINX.

Offers 60+ plugins.

Sub millisecond latency.

Platform agnostic, scalable.

Adopted by many global enterprises.

Keycloak

- Java based "Authentication and Authorization" Server
- Since 2013
- Apache opensource License
- Maintained by Red Hat and a large opensource community
- Red Hat SSO – Commercial offering

Features of Keycloak

Single sign-on

Supports all the key protocols: SAML 2.0, Oauth2, and OIDC

Integrates with Directory services making it easily compatible with your existing Identity stores

Out of the box support for: 2FA, email verification, user registration, social login

Identity as a Service (IDaaS)

https://www.okta.com

https://auth0.com/

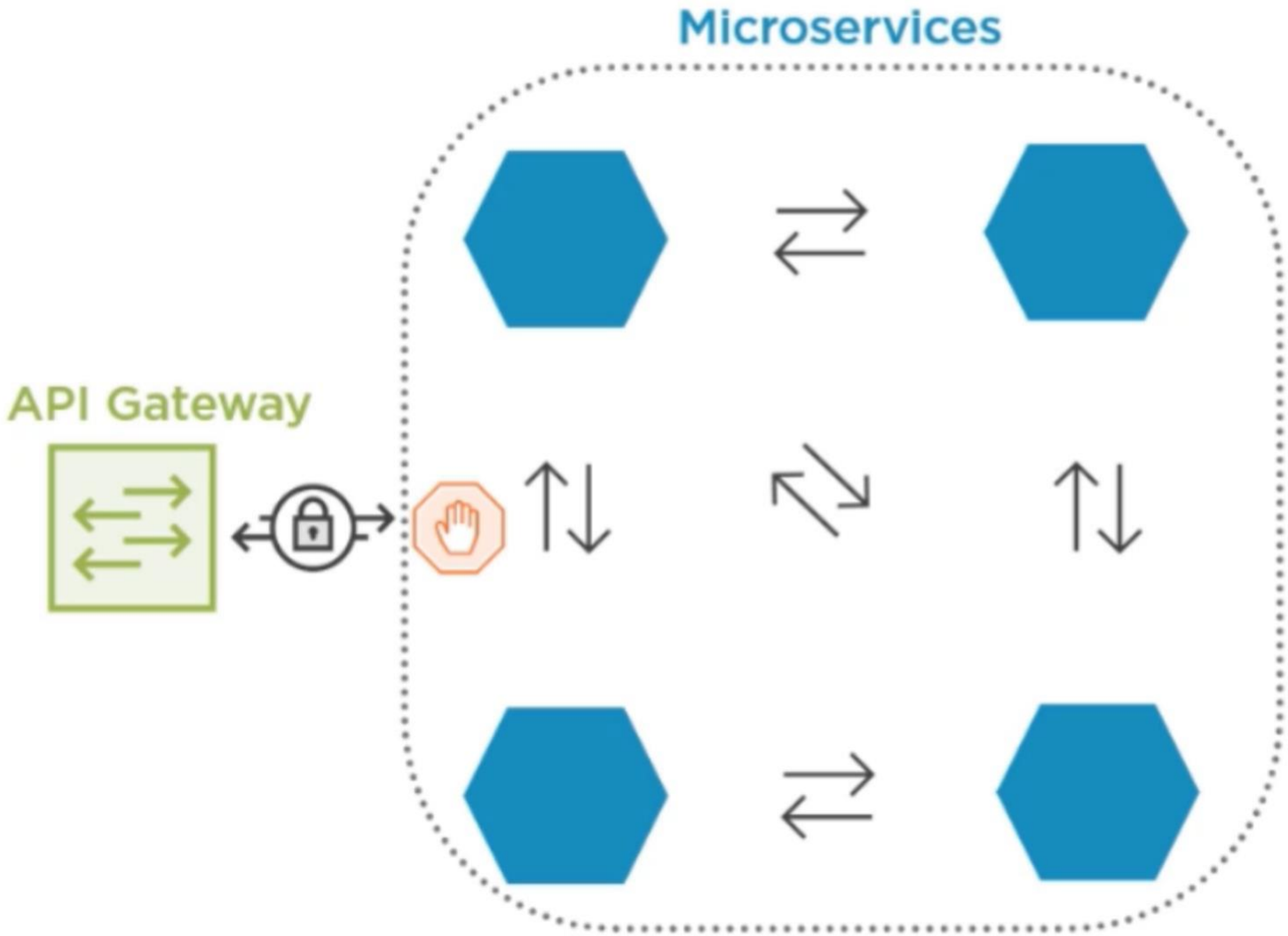
Securing the Communication between Your Microservices

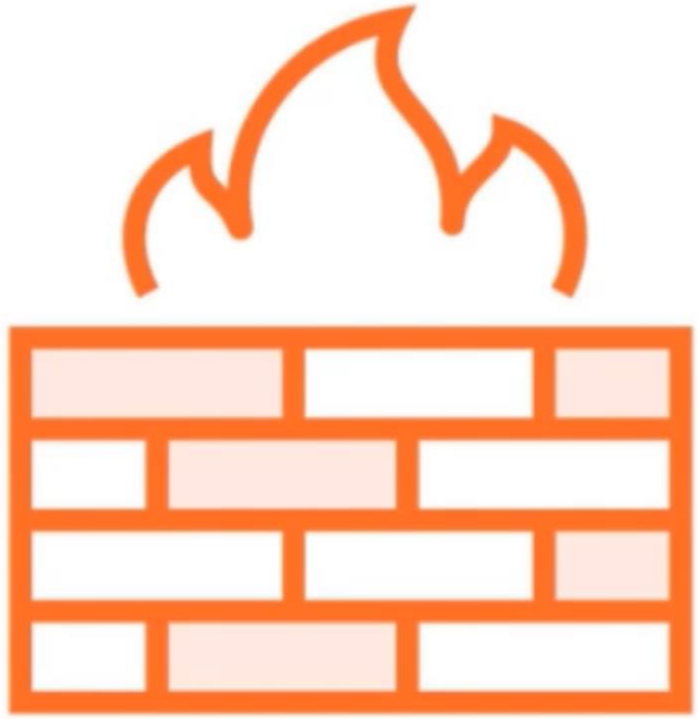
Organizations report that 38% of IT security incidents occur as a direct result of their employees' actions, and 75% originate from their extended enterprise (employees, customers, suppliers)

Ex-employees are responsible for 13% of cybersecurity incidents

Clearswift Insider Threat Index 2018

Trust the Network





In 2018, Fugue found that infrastructure misconfigurations such as:

- overlooked network settings, firewall rules, storage access policies.

are the leading cause of data breaches in the cloud, not software vulnerabilities or targeted attacks.

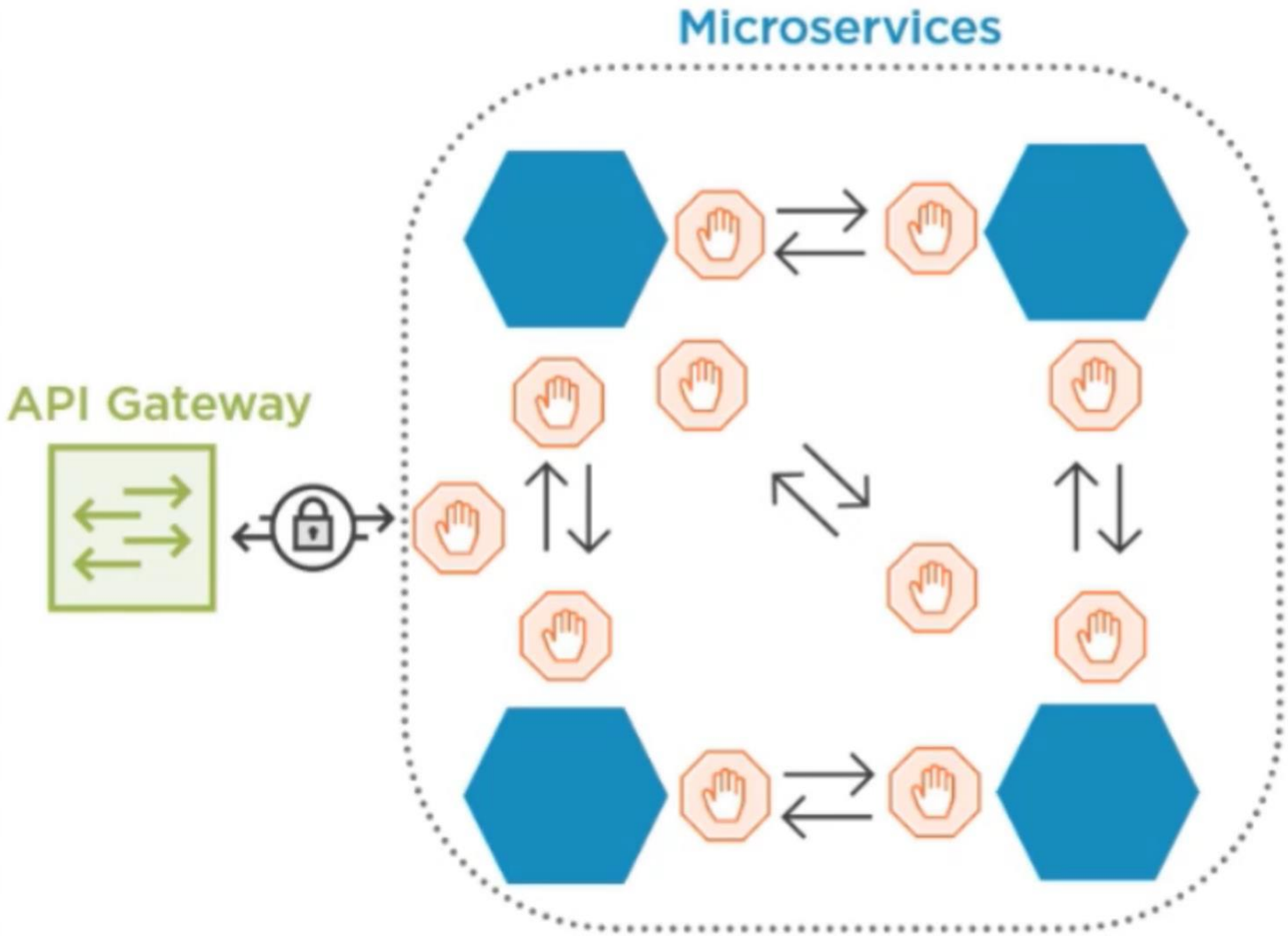
In 2013 Target was fined 18.5 million, as 41 million of the company's customer payment card accounts were compromised.

In 2013 Target was fined 18.5 million, as 41 million of the company's customer payment card accounts were compromised.

The attackers gained access to Targets corporate network by compromising a third-part vendor with a phishing attack.

Zero Trust

Trust no one, verify everything



Key Considerations

Integrity - Maintaining and assuring the accuracy and completeness of data in transit.

Confidentiality – Prevent data in transit being accessed by unauthorized parties.

Authentication – Verifying each party is who they claim to be.

Non-repudiation – Sender owns the request, no way to deny.

Delegated access – Verifying the client is acting in good faith on behalf of the user.

Security Protects Against

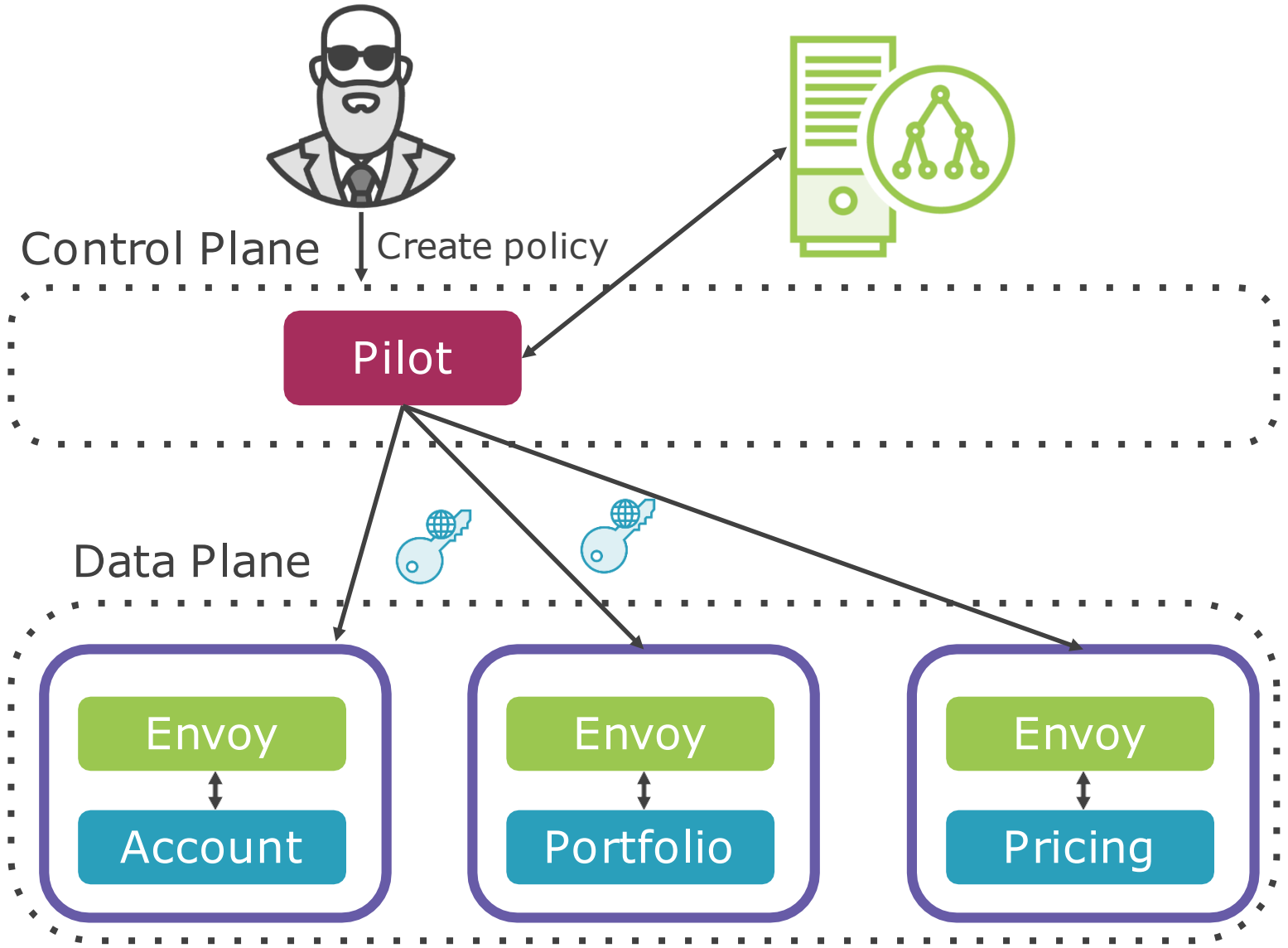
Hackers

Human error
miss-configuration

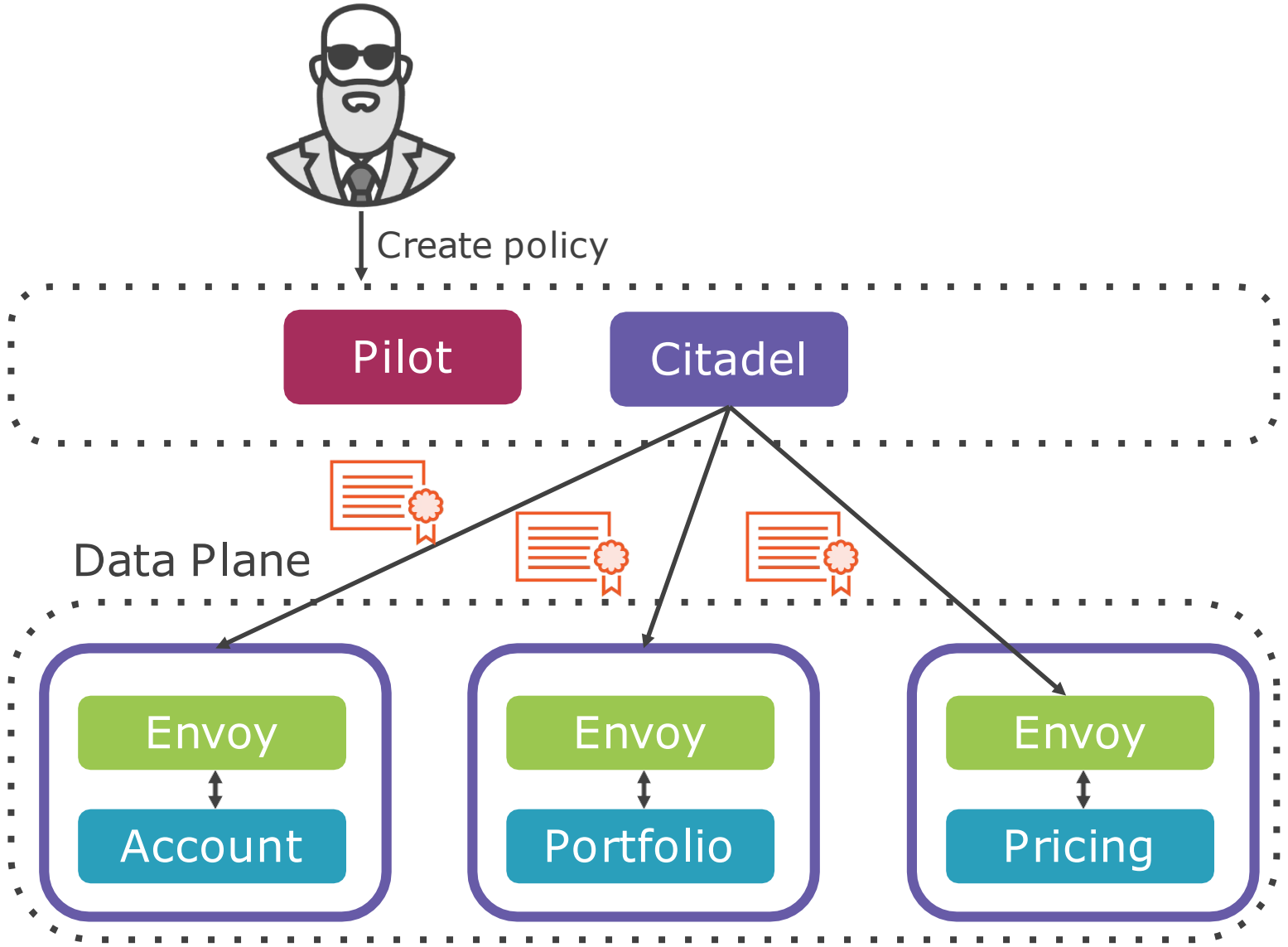
Fear of change

Service Mesh with Istio

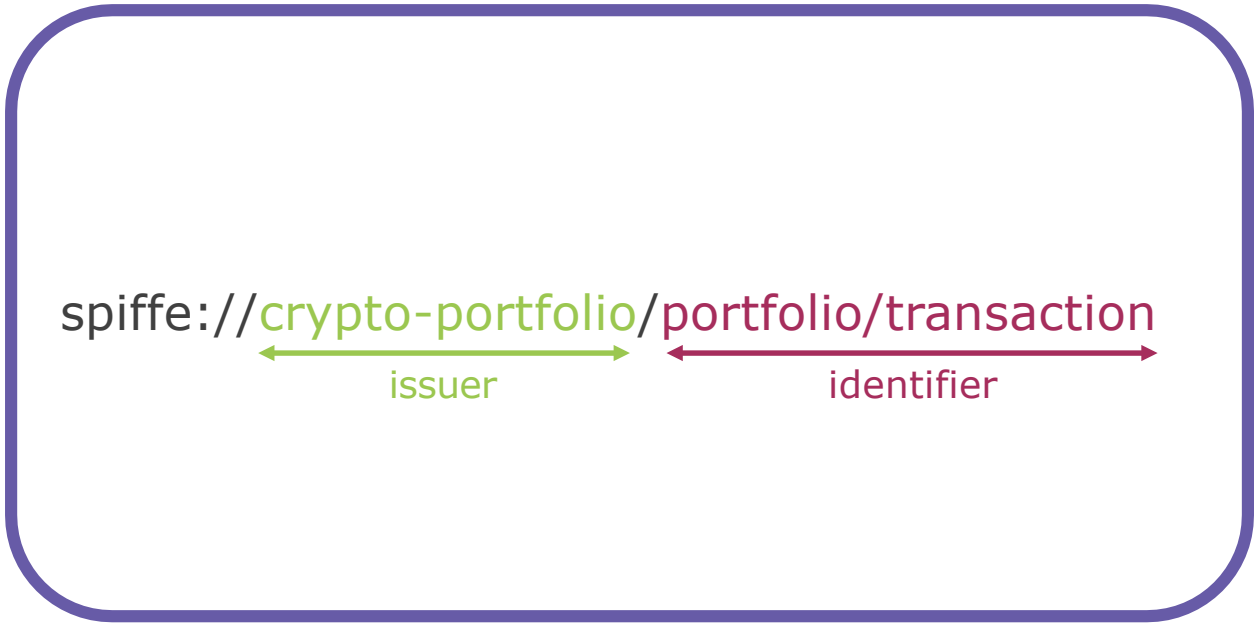
Authorization Server



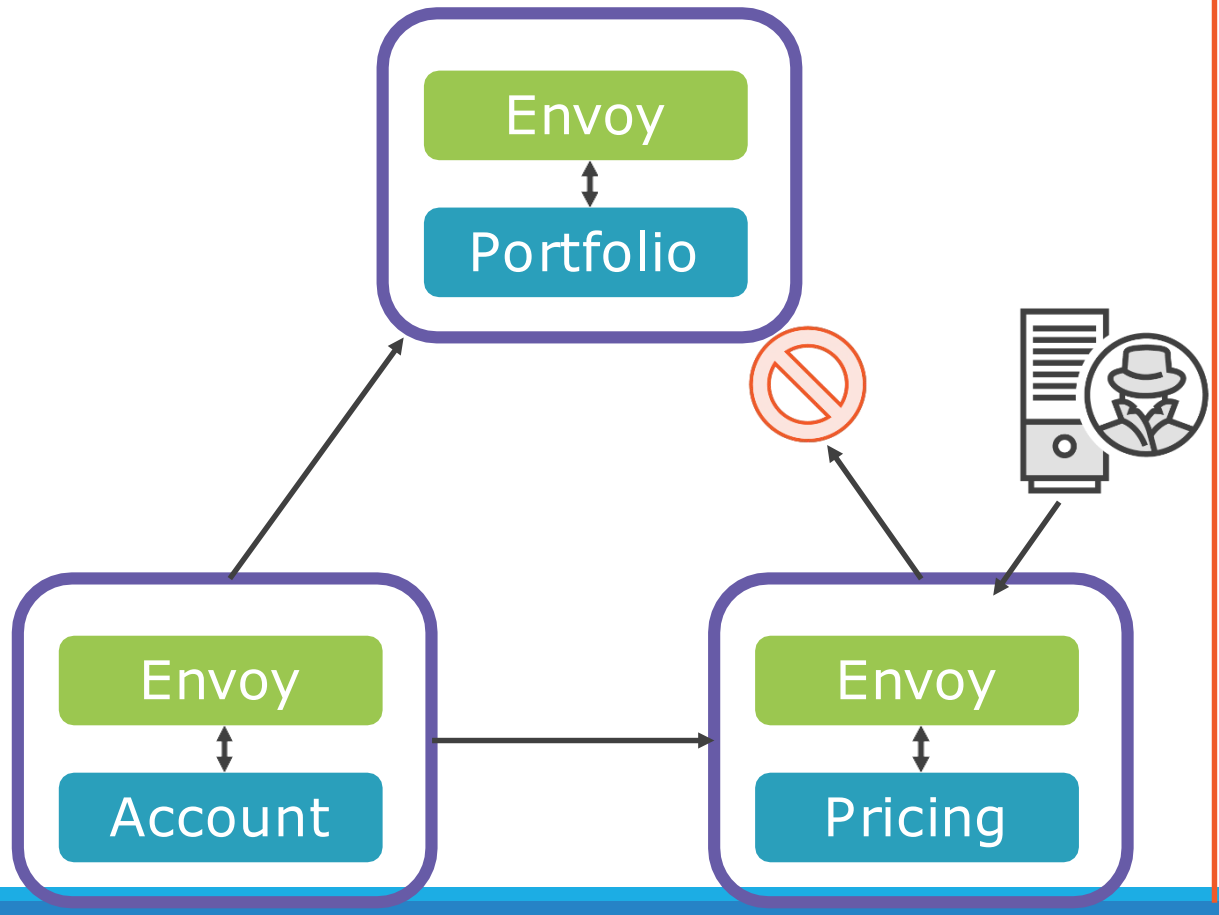
Control Plane



X.509 Certificate



Service-Service Authorization

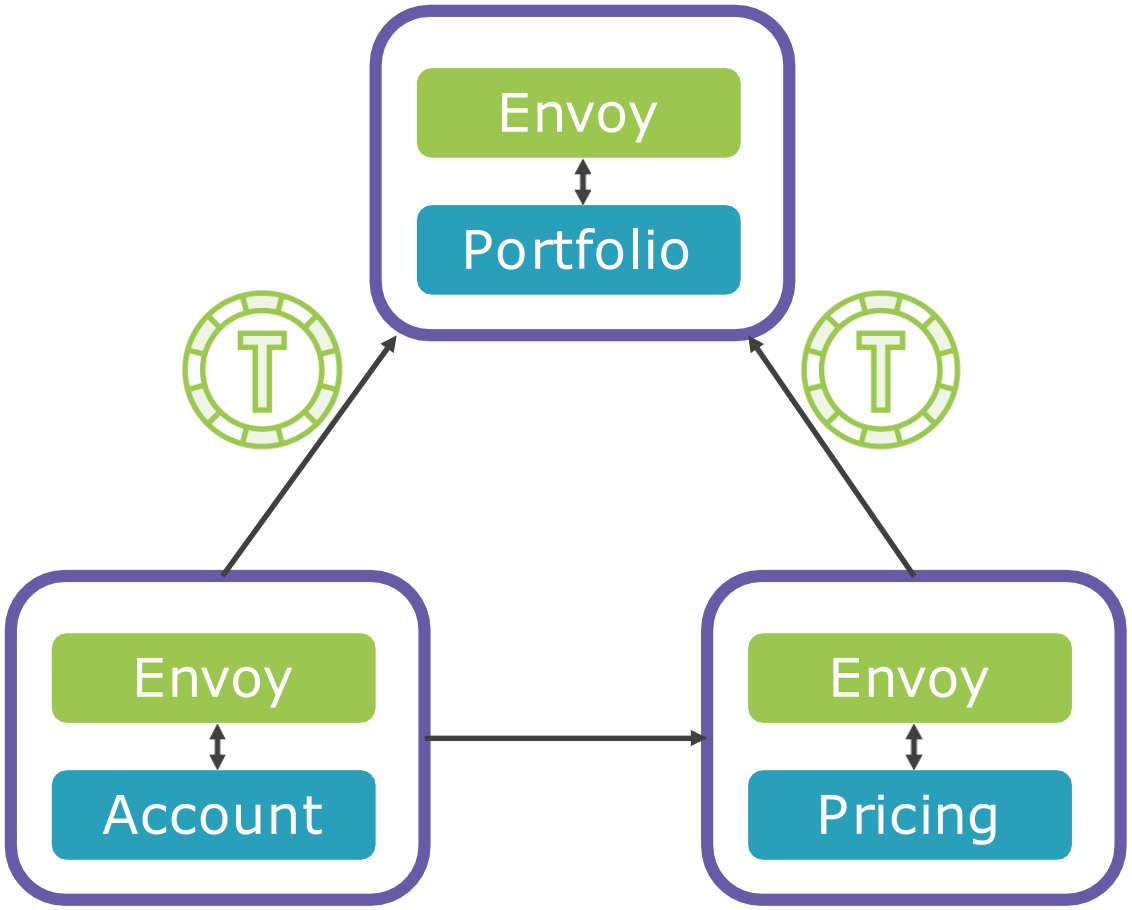


Can configure which services are allowed to communicate with each other.

Supports operational authorization.

Hence limits the impact of a single service being compromised.

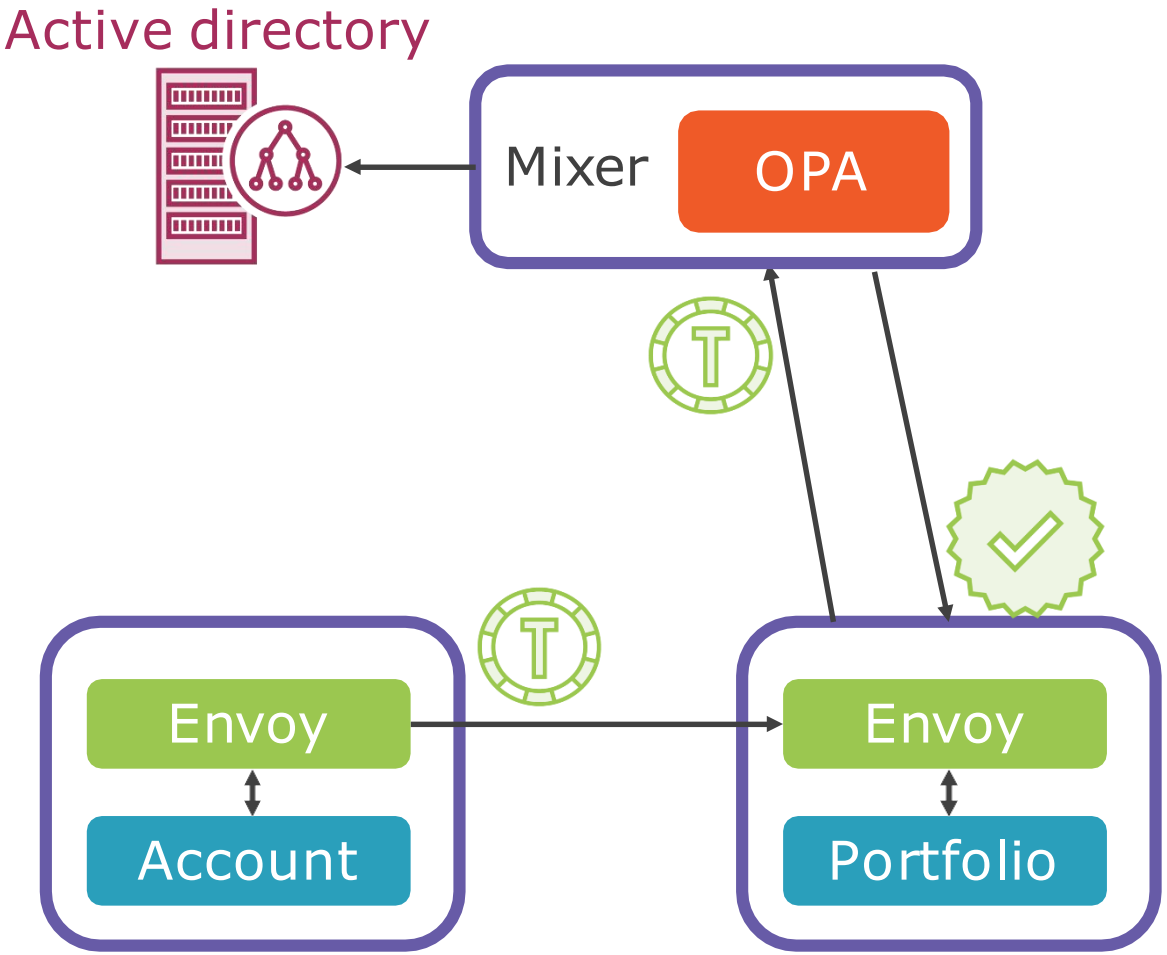
JWT Support



Can perform token verification.

Claims or Scope based access control.

Authorization as a Service

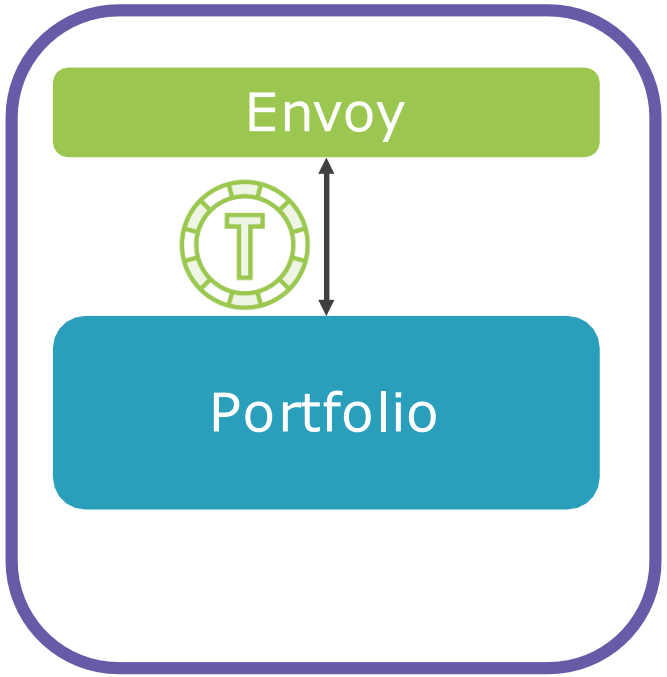


Supports authorization as a service.

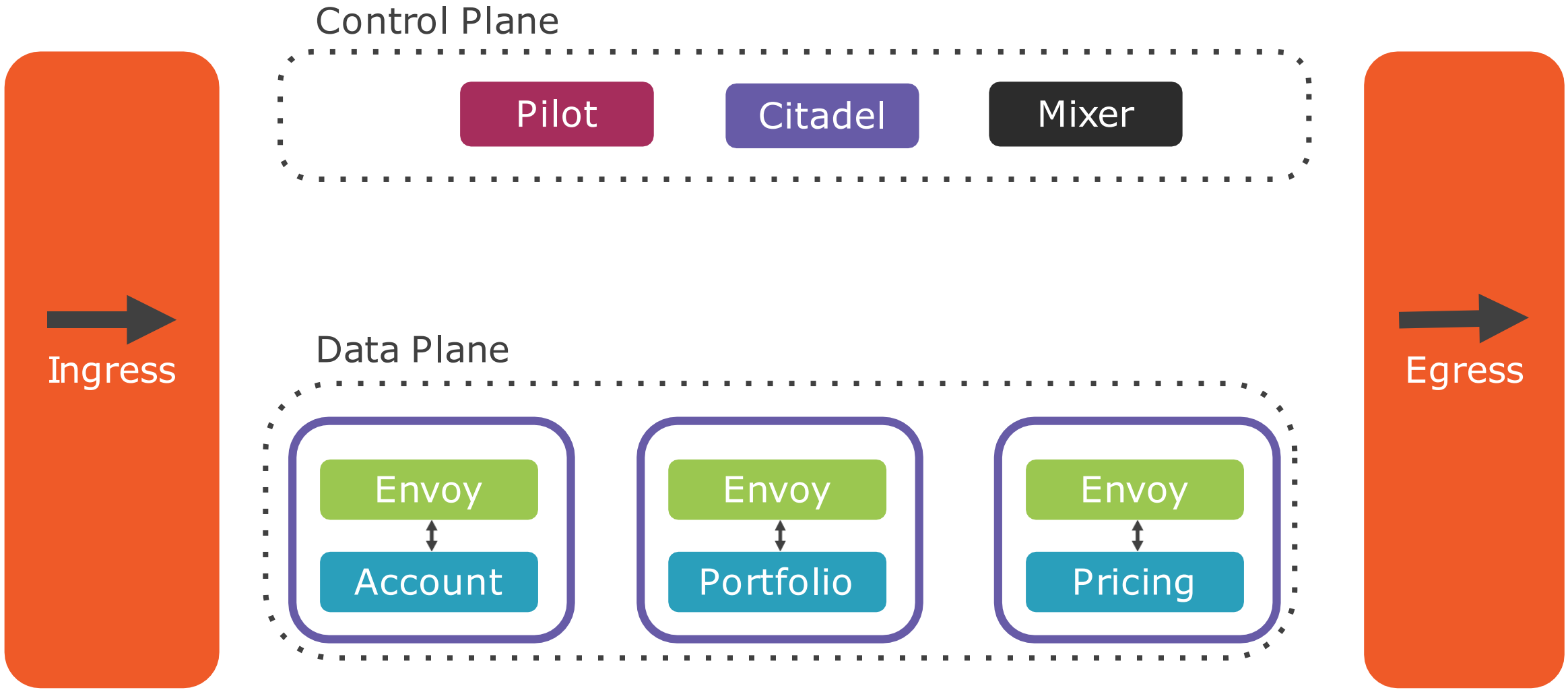
Can introduce additional latency to the request, however this can be limited with caching.

Mixer also is responsible for collecting metrics from your sidecars.

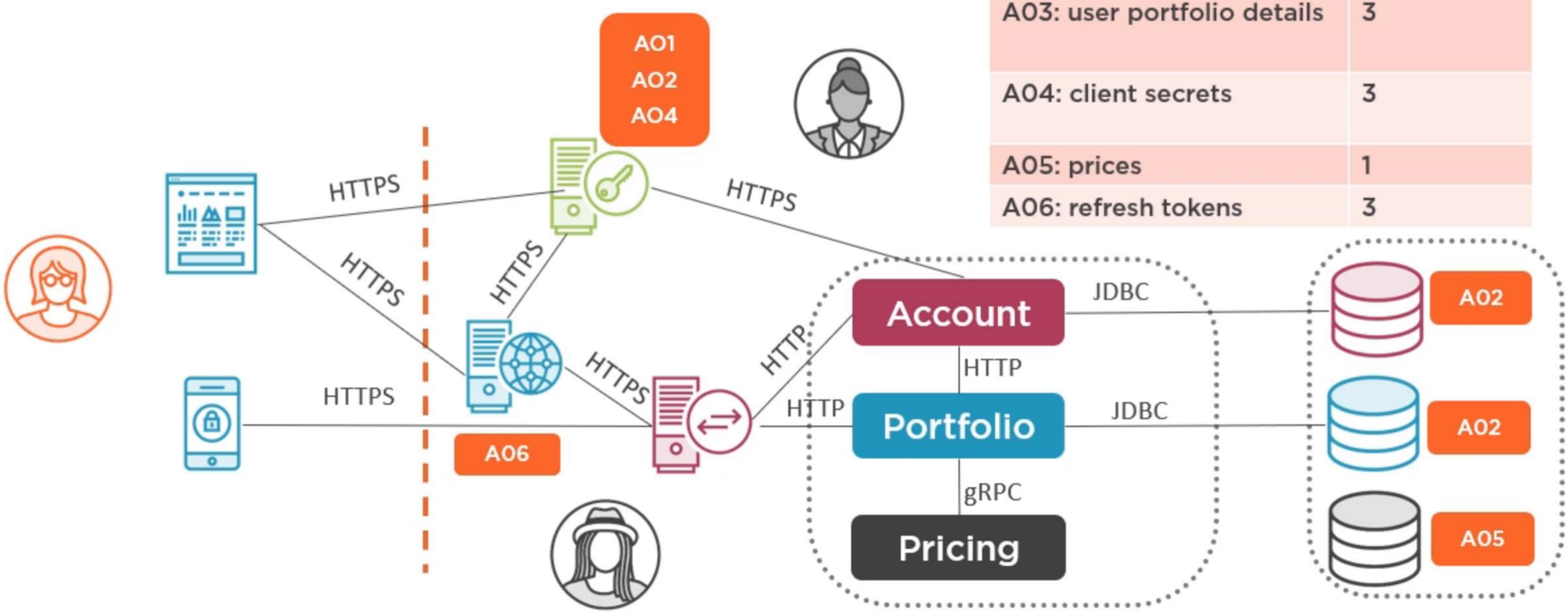
Authorization at the Microservice

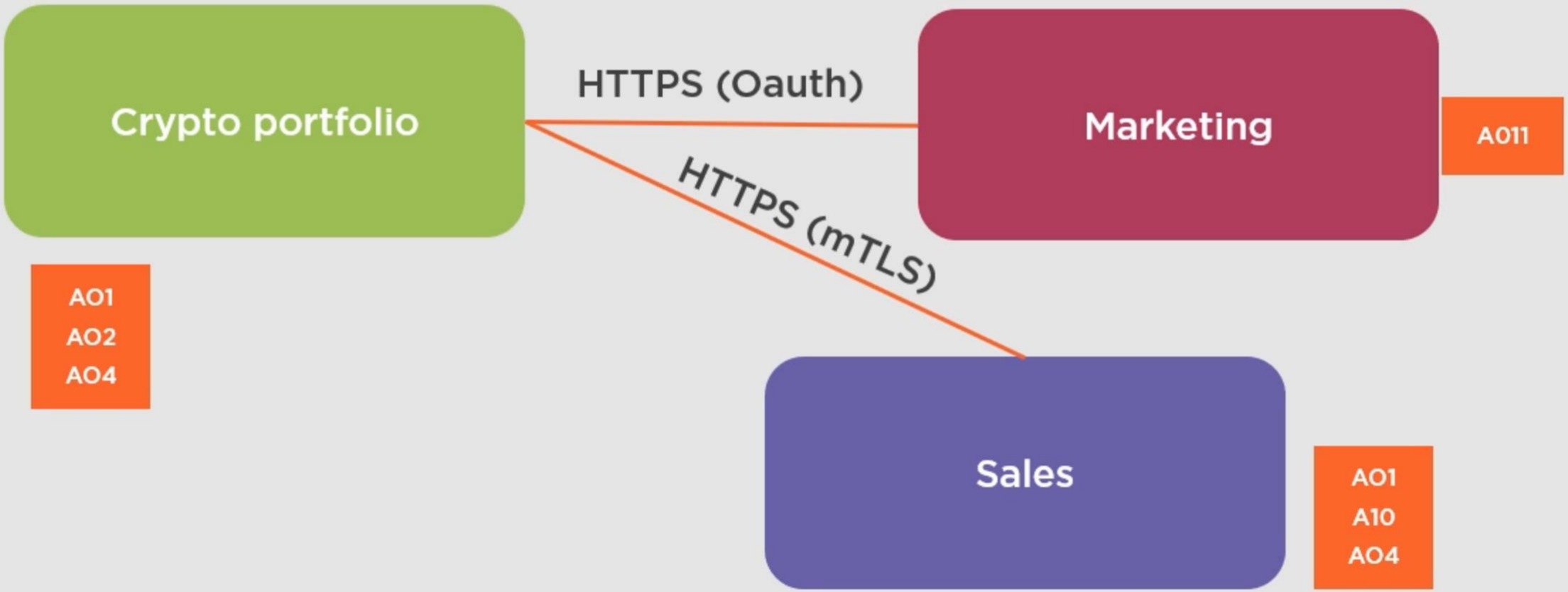


After successful authentication at the sidecar, the token is propagated to the microservice.


Control Plane
Pilot
Citadel
Mixer
Ingress
Egress
Data Plane
Envoy
Account
Envoy
Portfolio
Envoy
Pricing

Data	Sensitivity
A01: user credentials	3
A02: user account details	3
A03: user portfolio details	3
A04: client secrets	3
A05: prices	1
A06: refresh tokens	3




Crypto portfolio
HTTPS (Oauth)
HTTPS (mTLS)
Marketing
A011
AO1
AO2
AO4
Sales
AO1
A10
AO4

STRIDE

Spoofing

Tampering

Repudiation

Information disclosure

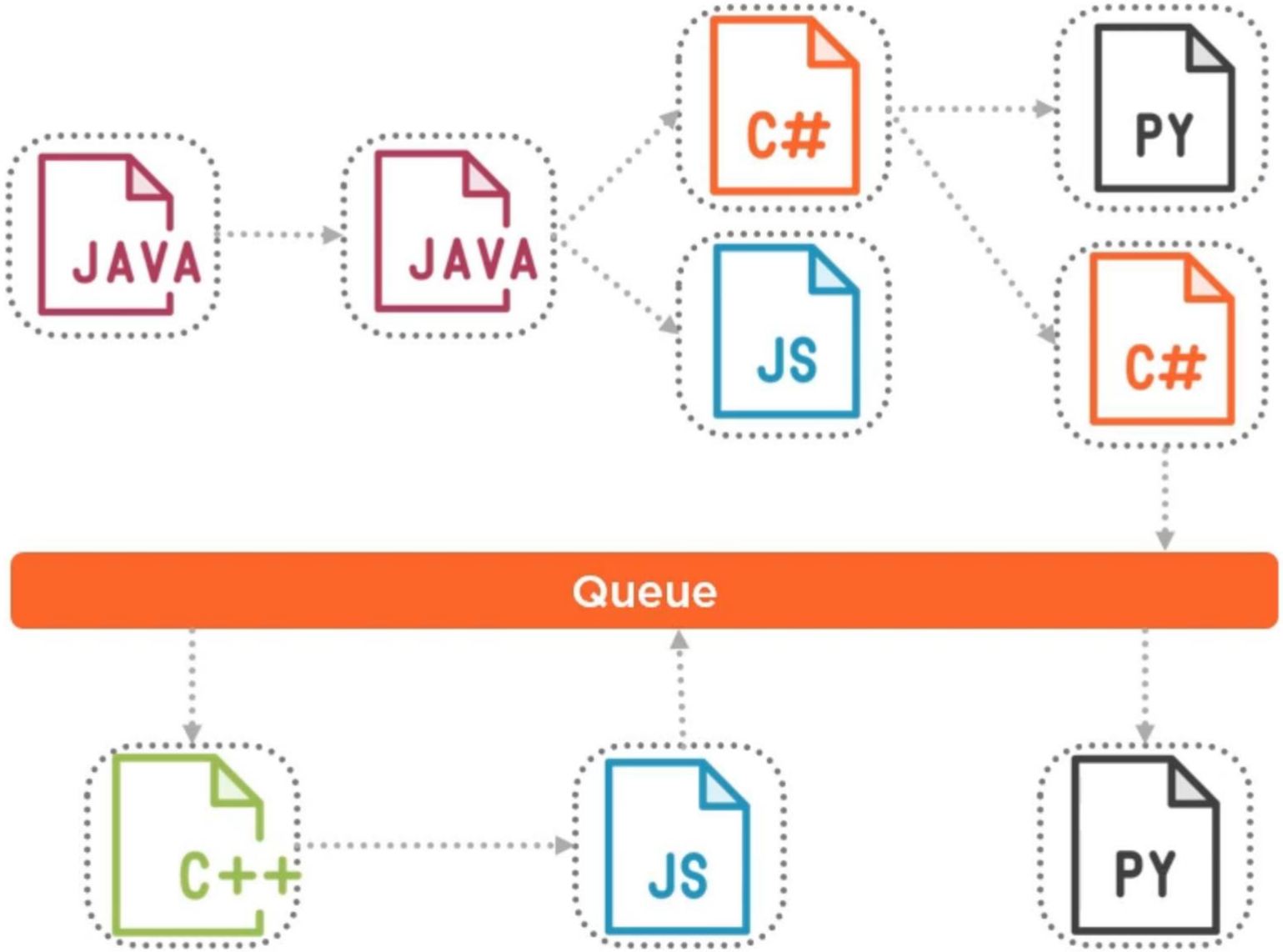
Denial of service

Elevation of privilege

Assessing the Risk

Likelihood \ Impact	High	Medium	Low	Negligible
High	Critical	High	Medium	Negligible
Medium	High	Medium	Low	Negligible
Low	Medium	Low	Low	Negligible
Negligible	Negligible	Negligible	Negligible	Negligible

Polyglot Microservices



Don't Re-invent the Wheel

Use industry recognised standards and protocols.

Use well known security frameworks.

- Don't disable any default security configuration unless you understand the impact.
- The adoption of frameworks has resulted in many common security vulnerabilities dropping out of the OWASP top 10.

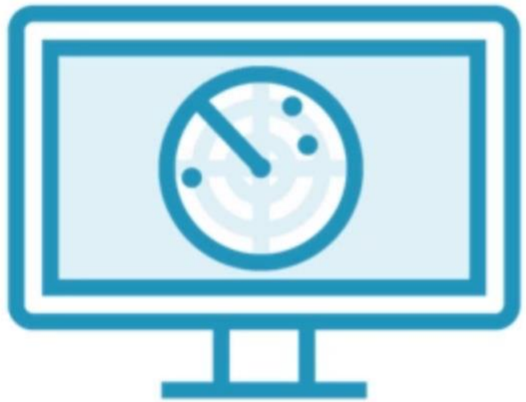
Implement a robust test suite for security.

Configure static code analysers to perform security checks.

- Integrate it with your CI pipeline.
- Fail your build if a critical vulnerability is detected.

Conduct training and code reviews.

The more simpler your architecture the more easier it is to understand and hence easier it is to secure.



Tools like SonarQube, can scan your code, libraries and dependencies for security vulnerabilities.

Developers will think twice before introducing new technology or libraries.

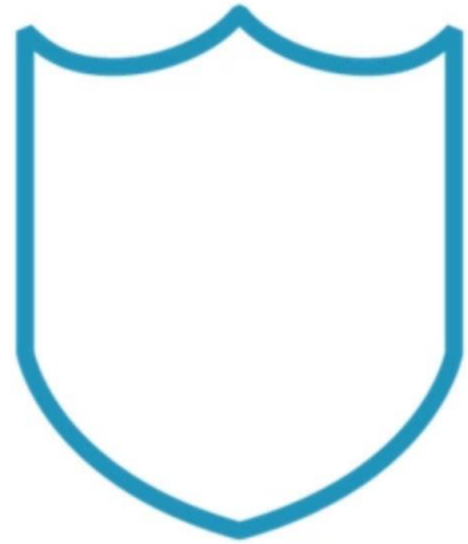
Patch your operating system and infrastructure.

Secret Management

Short lived

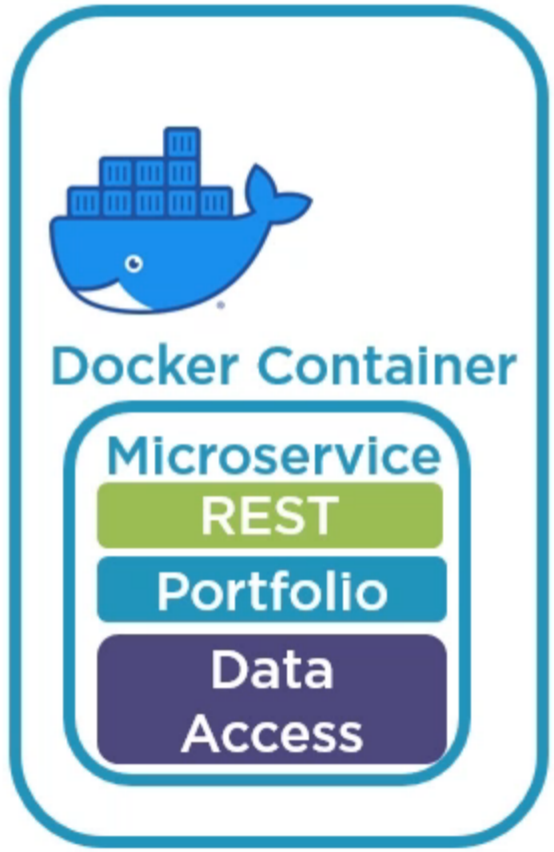
Rotated frequently

Encrypted



Least privilege

Immutable Server



The kernel is shared between the host and the containers.

- Risk of kernel panic and DOS attacks on the host.
- Risk of container breakout.
- Do not start containers with root user, use a low privileged user.
- Set container file systems to read only.
- Set memory limits on your containers.

Use minimal containers.

- Smaller memory footprint.
- Reduced attack surface.
- Less patching required.

Ensure you're using official images.

Perform static analysis on your images for security vulnerabilities.

Clair

Automatic container vulnerability and security scanning for appc and Docker

Documentation GitHub Project

Clair 2.0.1 Documentation

2.0.1 (latest)

Clair is an open source project for the static analysis of vulnerabilities in appc and docker containers.

Vulnerability data is continuously imported from a known set of sources and correlated with the indexed contents of container images in order to produce lists of vulnerabilities that threaten a container. When vulnerability data changes upstream, the previous state and new state of the vulnerability along with the images they affect can be sent via webhook to a configured endpoint. All major components can be customized programmatically at compile-time without forking the project.

Using Clair

Getting Started

Using Clair

Production users

Container Secret Management

Don't store any secrets on your images.

Options include:

- Environment variables:
  - Anything running in the container has access to them.
  - Inspect command or environment dump can reveal them.

Externalized properties files.

Centralized secret management tool like Vault.

Docker Content Trust (DCT)

Can also be used to sign and verify docker images.