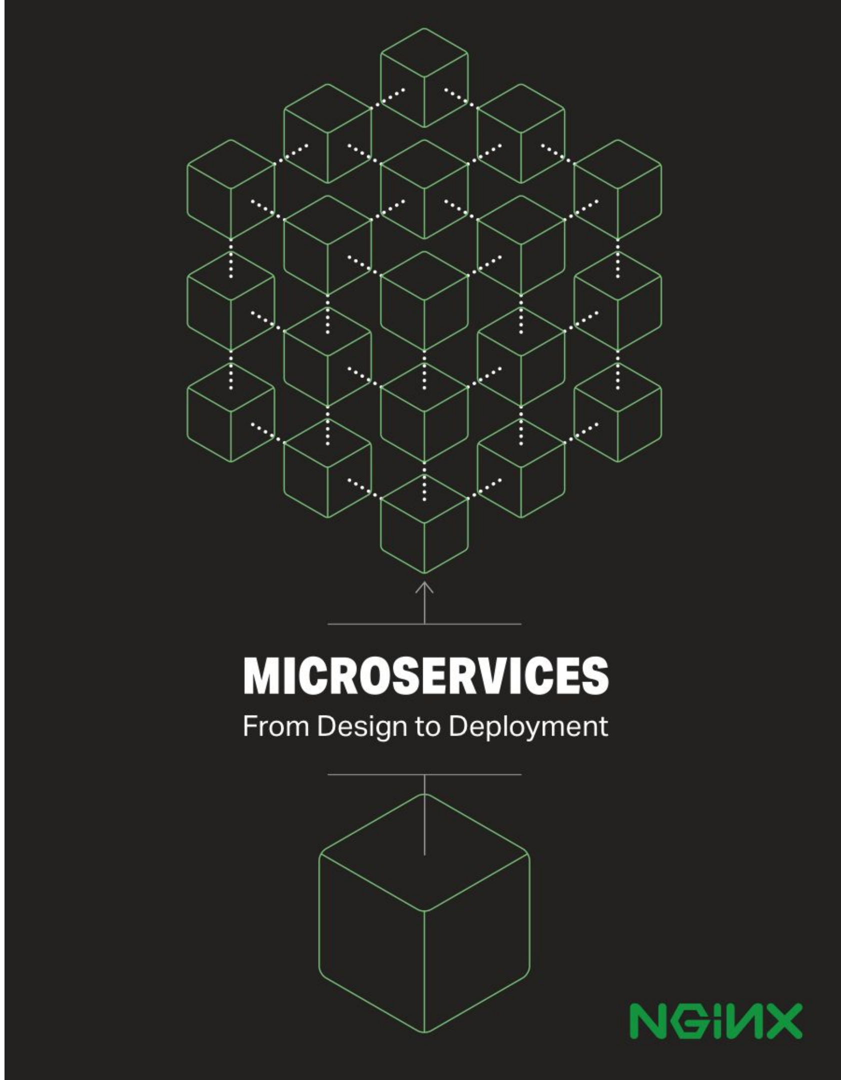
Chapter 2

Using an API Gateway

Designing and Deploying Microservices

by Chris Richardson



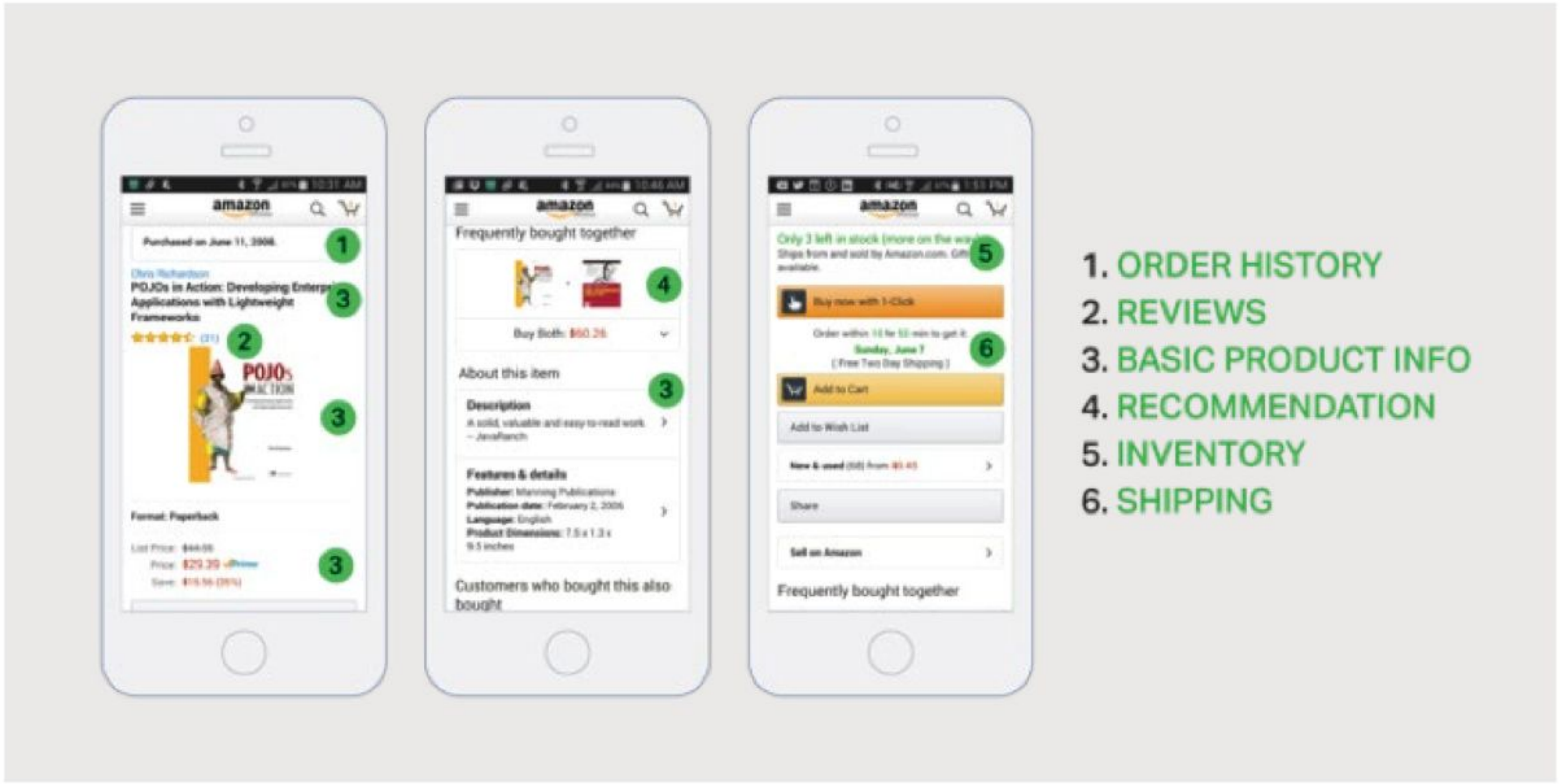


Figure 2-1. A sample shopping application.

1. Number of items in the shopping cart
2. Order history
3. Customer reviews
4. Low inventory warning
5. Shipping options
6. Various recommendations, including other products this product is frequently bought with, other products bought by customers who bought this product, and other products viewed by customers who bought this product
7. Alternative purchasing options

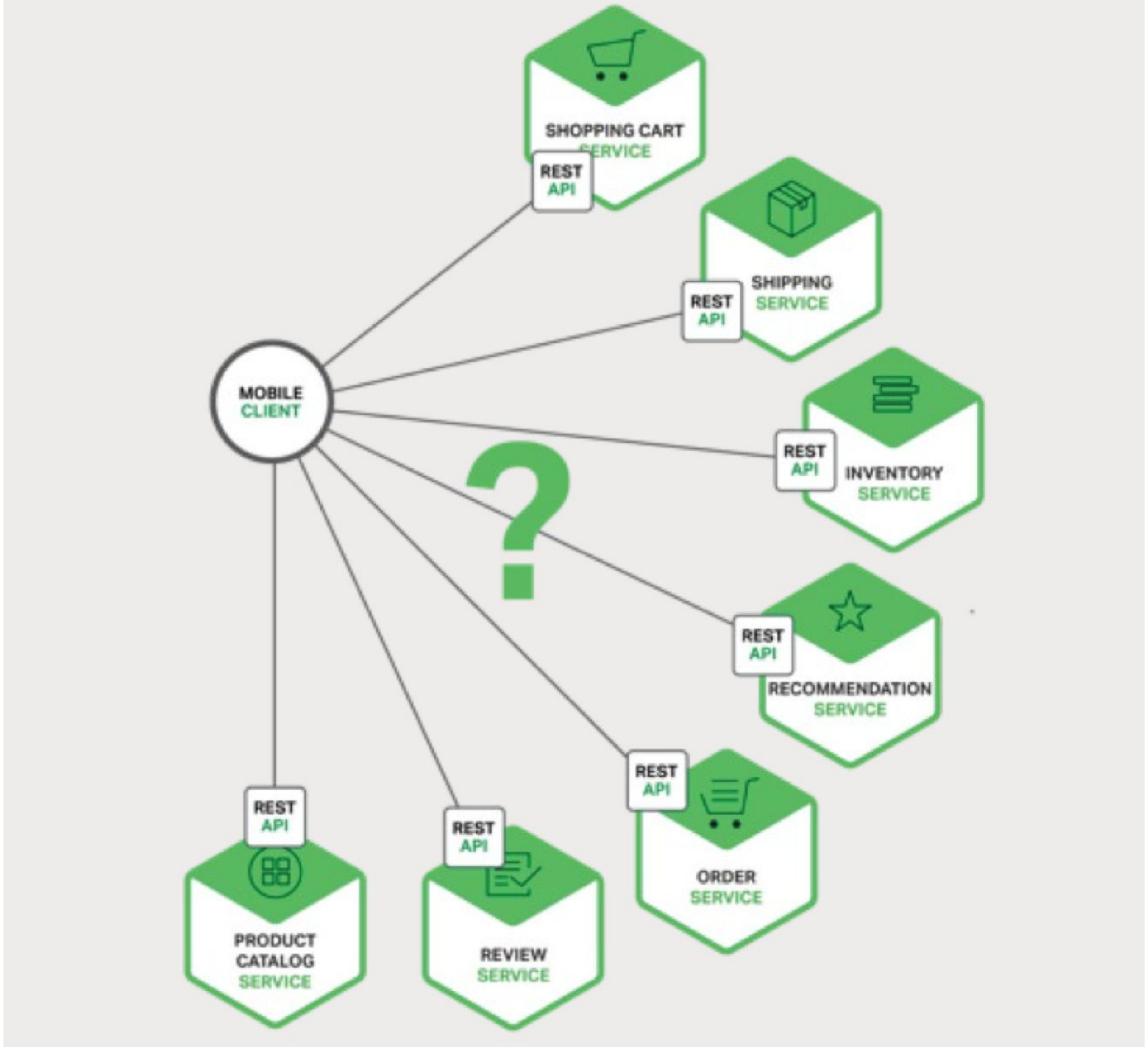
Monolithic application architecture

GET api.company.com/productdetails/*productId*

A load balancer routes the request to one of several identical application instances. The application then queries various database tables and return the response to the client

Microservices Architecture

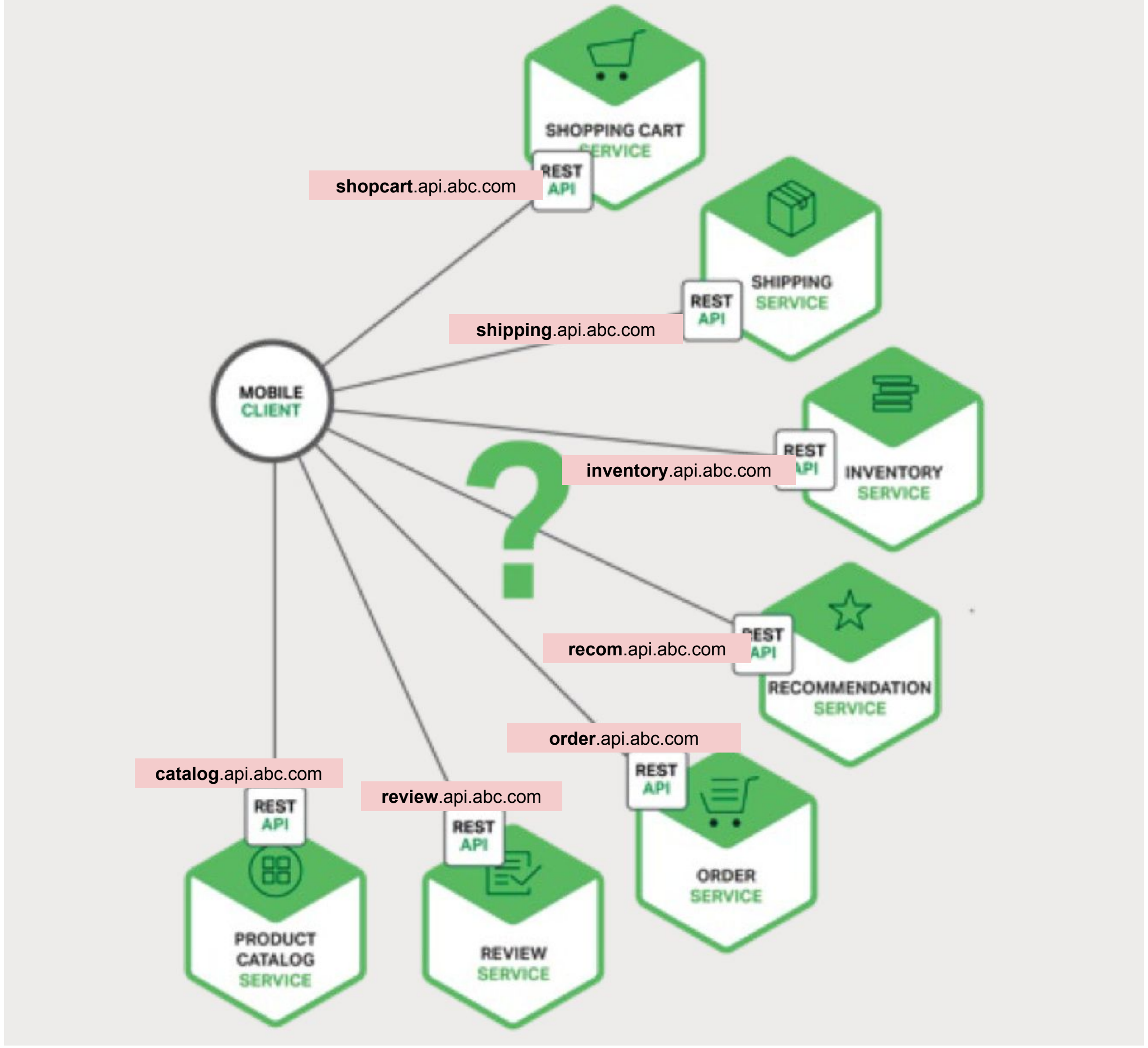
- Shopping Cart **Service** – Number of items in the shopping cart
- Order **Service** – Order history
- Catalog **Service** – Basic product information, such as product name, image, and price
- Review **Service** – Customer reviews
- Inventory **Service** – Low inventory warning
- Shipping **Service** – Shipping options, deadlines, and costs, drawn separately from the shipping provider’s API
- Recommendation **Service**(s) – Suggested items


SHOPPING CART SERVICE
REST API
SHIPPING SERVICE
REST API
MOBILE CLIENT
REST API
INVENTORY SERVICE
?
REST API
RECOMMENDATION SERVICE
REST API
ORDER SERVICE
REST API
PRODUCT CATALOG SERVICE
REST API
REVIEW SERVICE

https://**serviceName**.**api**.**company.name**

Direct Client-to-Microservice Communication

Each microservice would have a **public endpoint.**


SHOPPING CART SERVICE
REST API
shopcart.api.abc.com
SHIPPING SERVICE
REST API
shipping.api.abc.com
MOBILE CLIENT
INVENTORY SERVICE
REST API
inventory.api.abc.com
?
RECOMMENDATION SERVICE
REST API
recom.api.abc.com
order.api.abc.com
REST API
ORDER SERVICE
catalog.api.abc.com
REST API
PRODUCT CATALOG SERVICE
review.api.abc.com
REST API
REVIEW SERVICE

Direct Client-to-Microservice Communication

The First Problem is

the **mismatch** between the needs of the client and the **fine-grained APIs exposed** by each of the microservices.

The First Problem

1. The client in this example has to make **seven separate requests.**
   - For example, Amazon describes how **hundreds of services** are involved in rendering their product page.
2. Too inefficient over the public Internet

一個頁面要七個請求
一個頁面要二十個請求
一個頁面要一百個請求

The Second Problem is

the client directly calling the microservices is that some might use **protocols** that are not **web-friendly.**

The Second Problem

1. One service might use **Thrift binary RPC** while another service might use the **AMQP messaging protocol**.
2. An application should use protocols such as **HTTP** and **WebSocket** **outside of the firewall.**

Apache Thrift

The Third Problem is

it makes it **difficult to refactor** the microservices.

The Thrid Problem

1. Over time we might want to change how the system is partitioned into services. For example, we might **merge two services** or **split a service into two** or **more services**.
2. The clients communicate directly with the services, then performing this kind of **refactoring** can be extremely difficult.

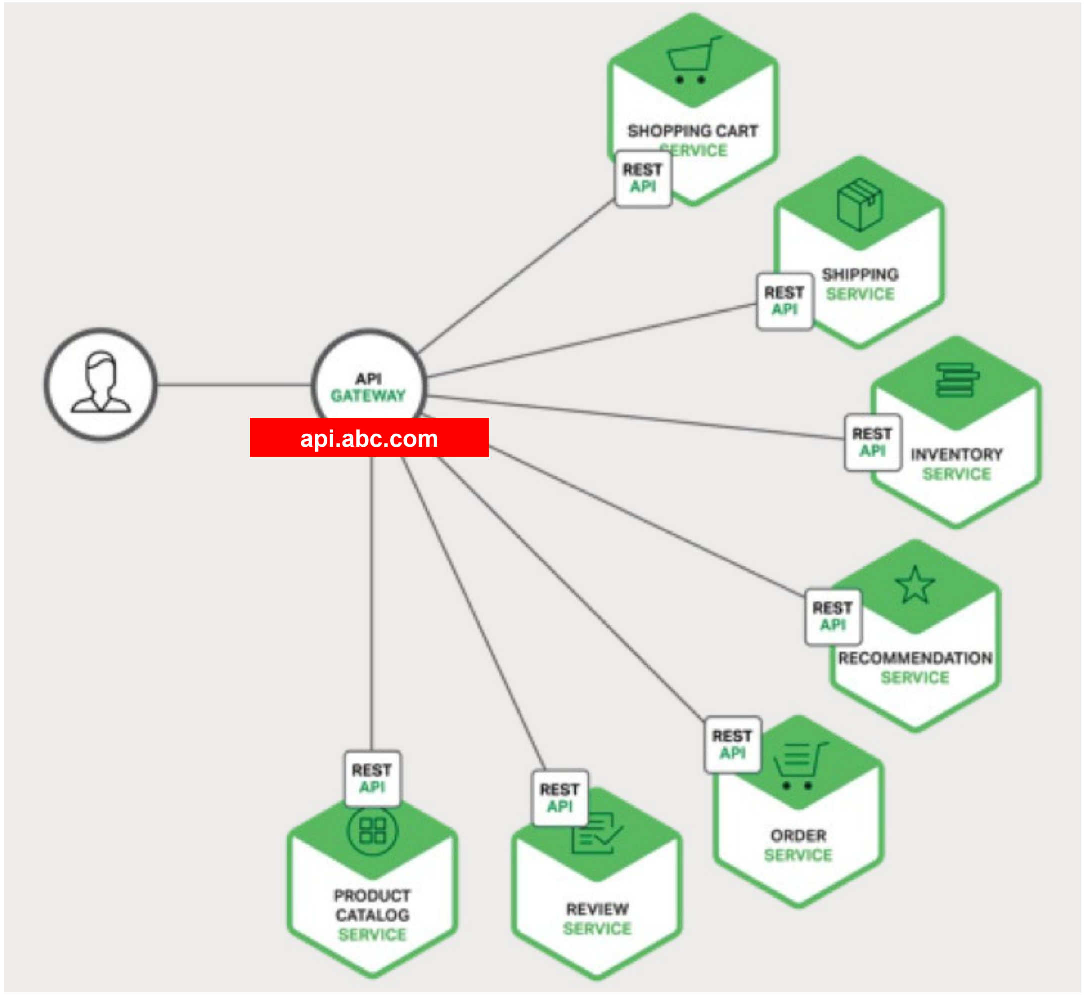
Direct Client-to-Microservice Communication

1. fine-grained APIs exposed
2. protocols are not web-friendly.
3. difficult to refactor

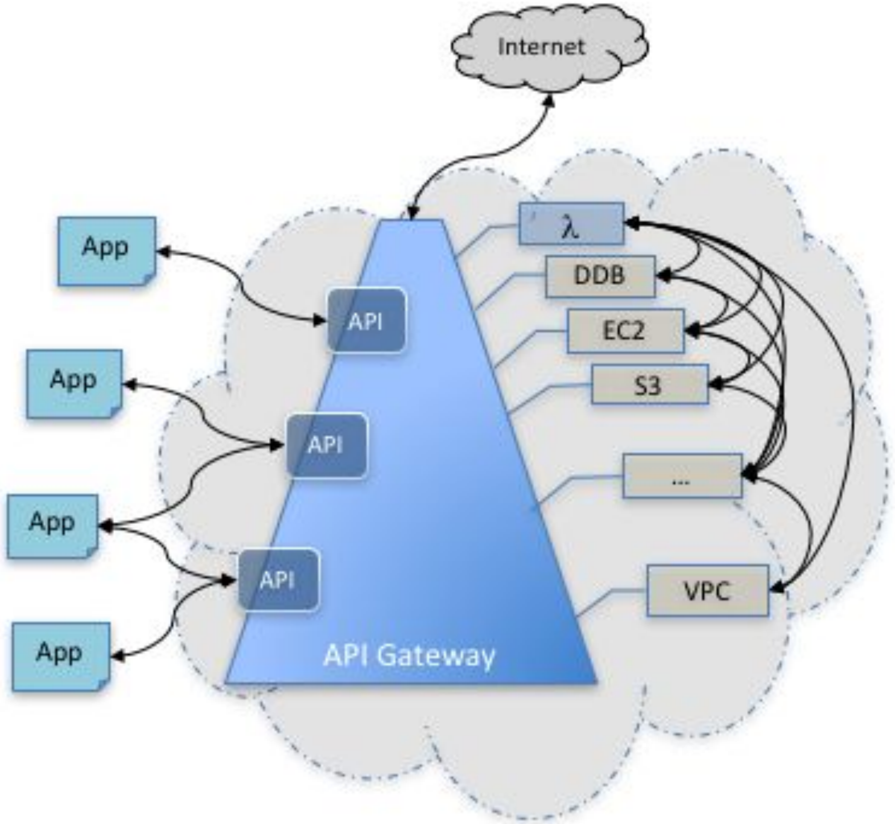
Using an API Gateway

What is an API Gateway?

- **Single entry point** into the system
- similar to **Facade pattern** from OOD.
- **other responsibilities** such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling

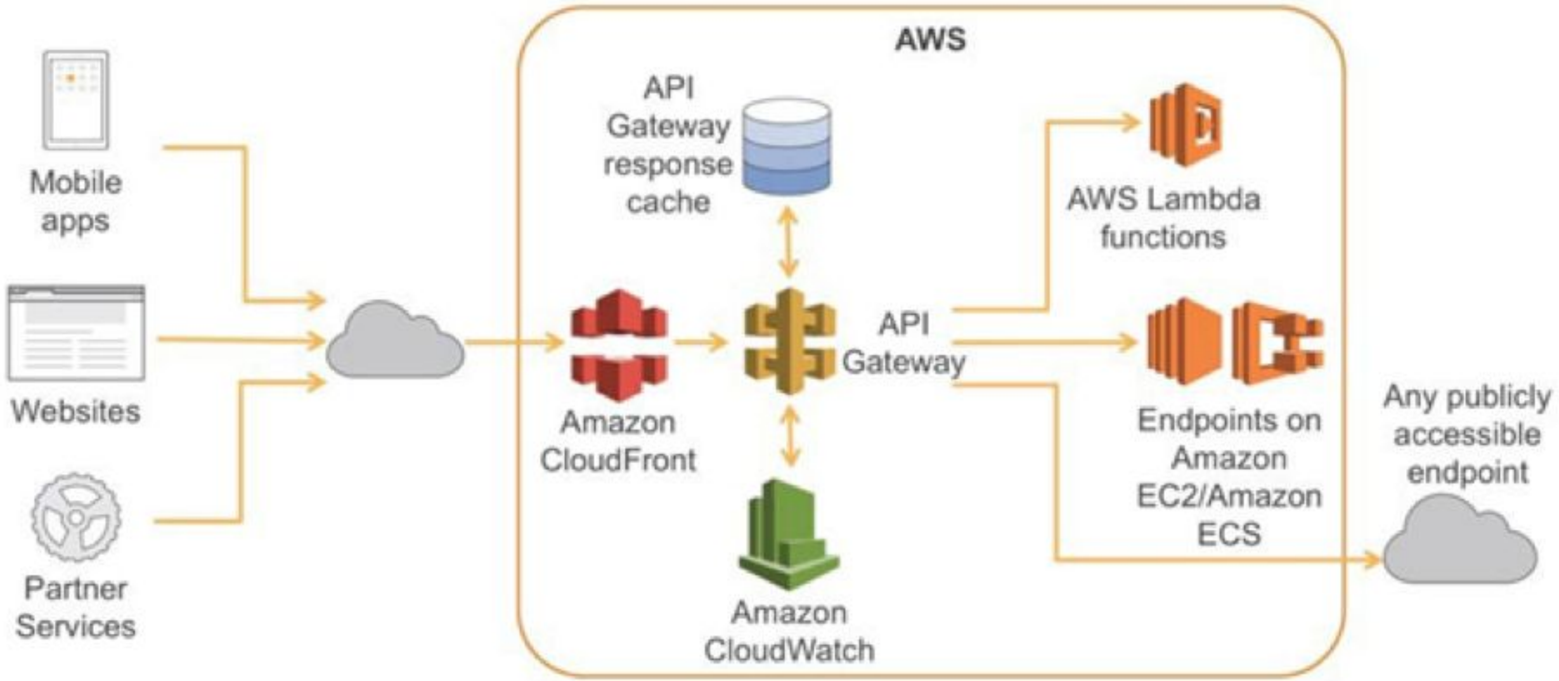

API GATEWAY
api.abc.com
REST API
SHOPPING CART SERVICE
SHIPPING SERVICE
INVENTORY SERVICE
RECOMMENDATION SERVICE
ORDER SERVICE
REVIEW SERVICE
PRODUCT CATALOG SERVICE

Single Entry Point



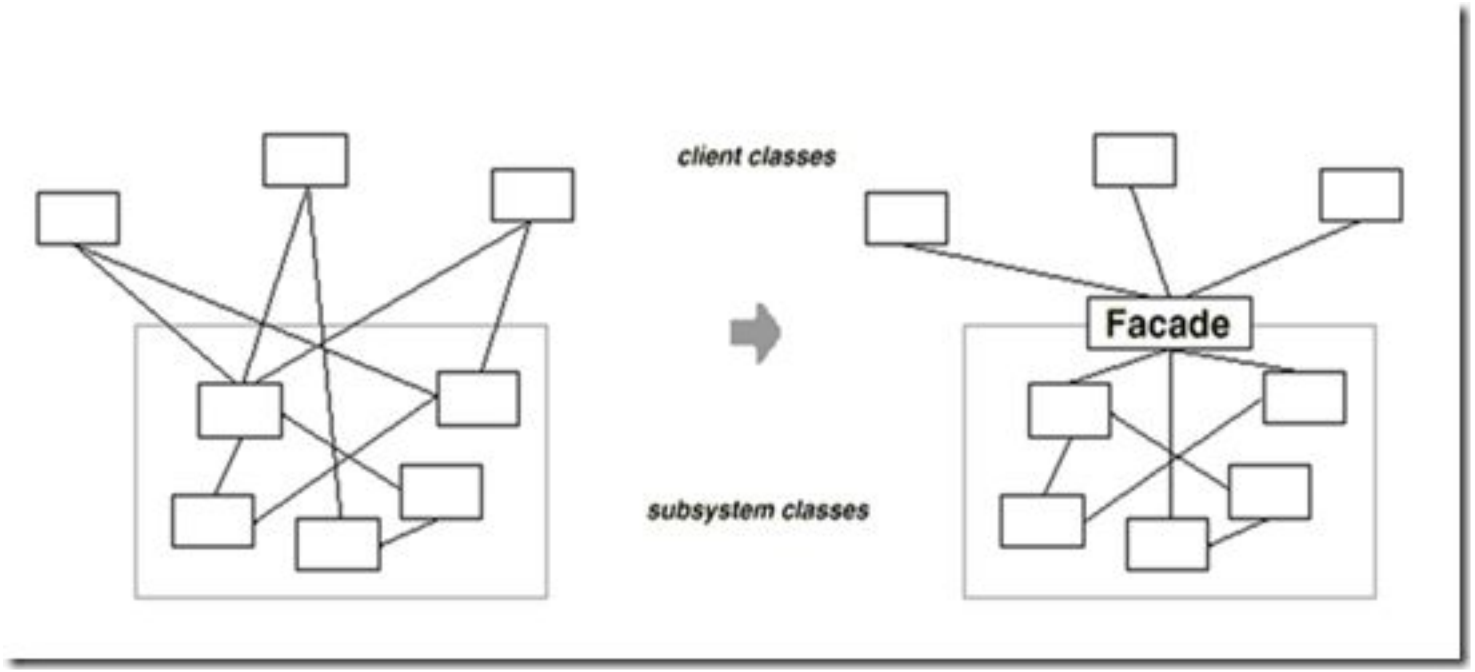
Amazon API Gateway

Overview API Gateway

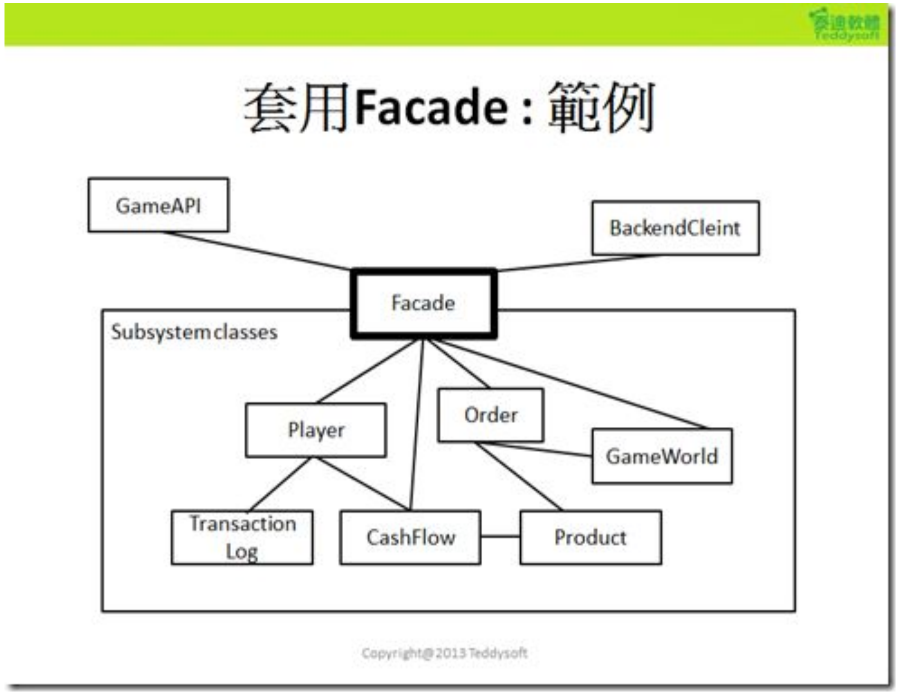


Microservices on AWS (AWS Whitepaper, PDF)

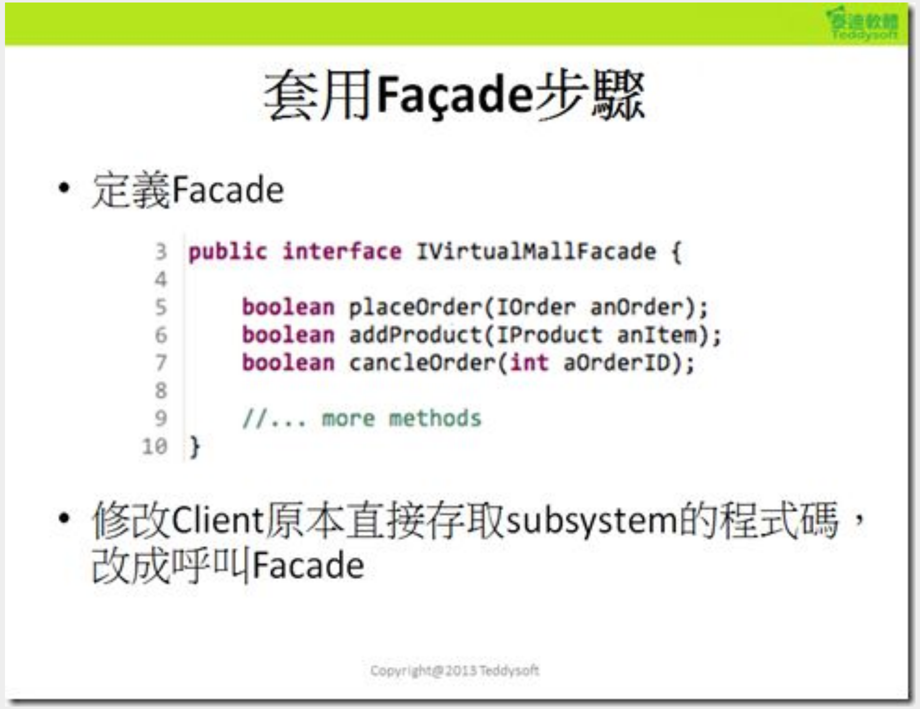
Facade Pattern



http://teddy-chen-tw.blogspot.com/2013/08/facade-pattern.html



套用Facade：範例



套用Façade步驟

- 定義Facade

```
public interface IVirtualMallFacade {

    boolean placeOrder(IOrder anOrder);
    boolean addProduct(IProduct anItem);
    boolean cancleOrder(int aOrderID);

    //... more methods
}
```

- 修改Client原本直接存取subsystem的程式碼，改成呼叫Facade

http://teddy-chen-tw.blogspot.com/2013/08/facade-pattern.html

簡單說：就是個大門，而且只有一個

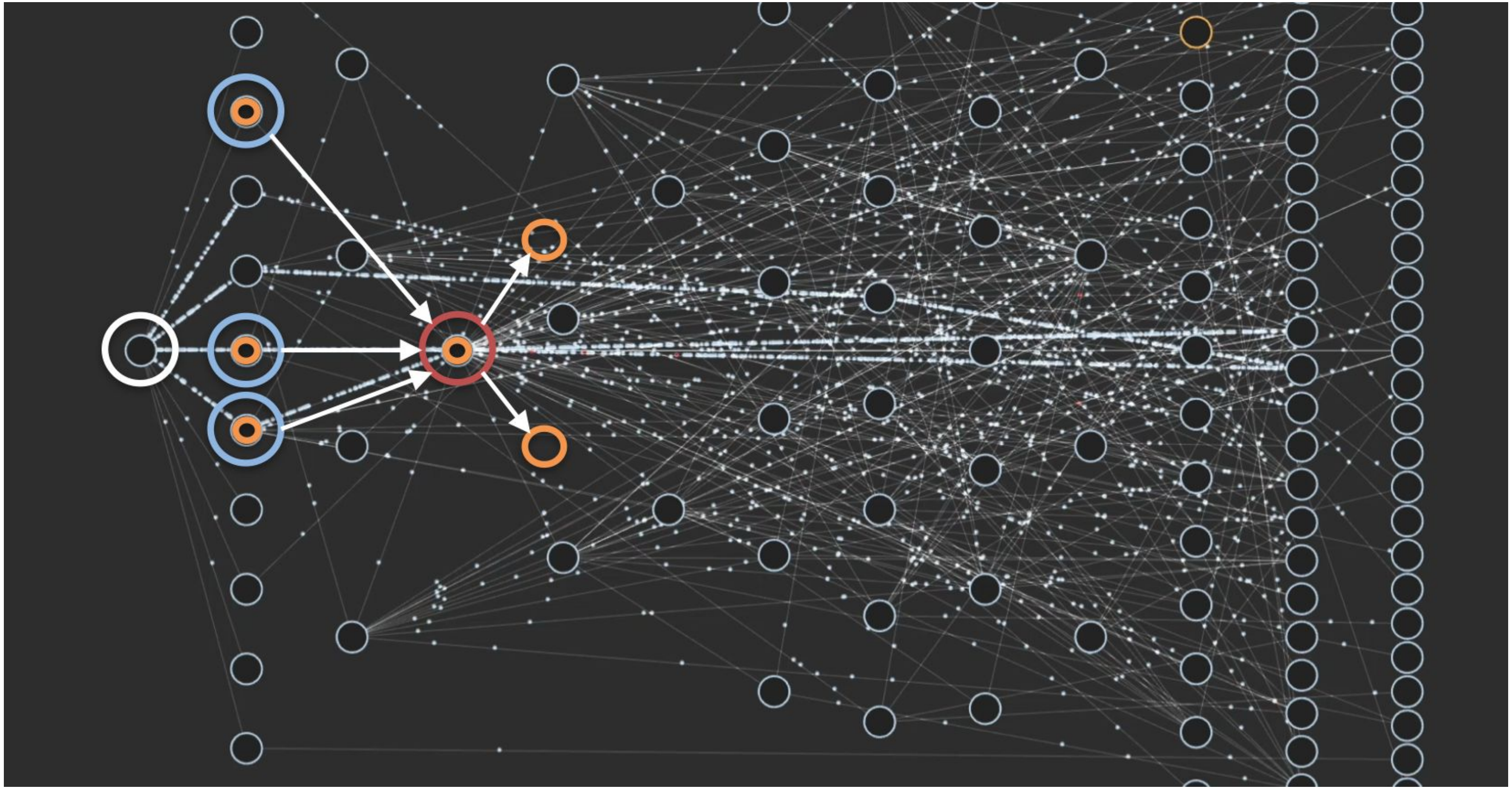
Authencation (鑰匙)

Monitoring (監控)

Cache (玄關)

Management (櫃檯)

...

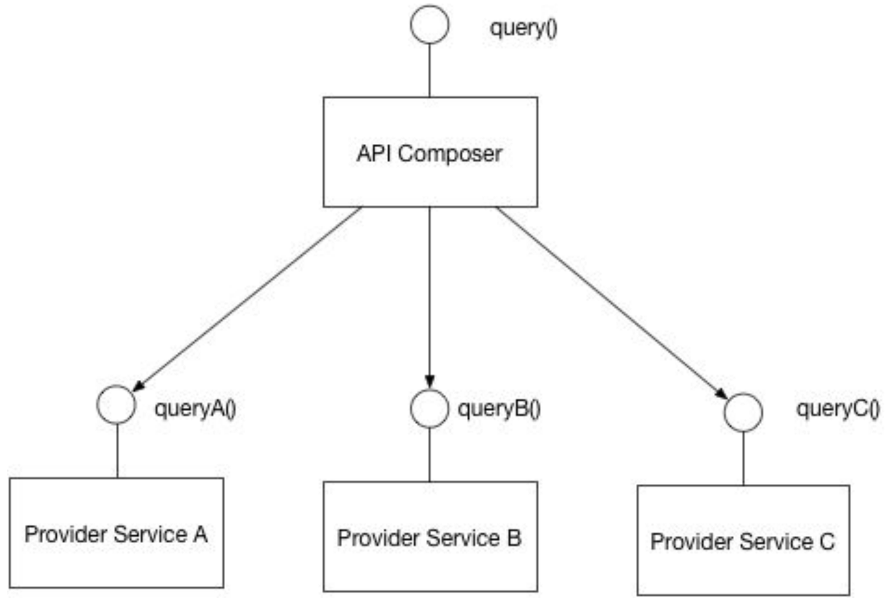


The API Gateway is responsible for

1. **request routing**: routes requests to the appropriate microservice.
2. **composition**: The API Gateway will often handle a request by invoking multiple microservices and **aggregating** the results
3. **protocol translation**: It can translate between web protocols such as HTTP and **WebSocket** and **web-unfriendly** protocols that are used internally.

API Composition

1. a mobile client to retrieve all of the product details with a **single request**.
2. The API Gateway handles the request by invoking the various services – **product information, recommendations, reviews**, etc – and **combining the results**



https://microservices.io/patterns/data/api-composition.html

Example: Nextflix API Gateway

1. The Netflix streaming service is available on **hundreds of different kinds of devices** including televisions, set-top boxes, smartphones, gaming systems, tablets, etc.
2. provide a **one-size-fits-all API** for their streaming service.
3. they use an API Gateway that provides an **API tailored** for each device by running device-specific adapter code. **An adapter typically handles each request by invoking**, on average, **six to seven backend services**.

Benefits and Drawbacks of an API Gateway

Benefits

- A major bene t of using an API Gateway is that it **encapsulates the internal structure of the application.**
- The API Gateway provides each kind of client with a specific API. This **reduces the number of round trips between the client** and application. It also simplifies the client code.

Drawbacks

- It is yet another **highly available component** that must be **developed, deployed, and managed.**
- There is also a risk that the API Gateway becomes a **development bottleneck**.

Notes

- It is important that the **process for updating** the API Gateway be as **lightweight** as possible. **(Deployment and Operational)**
- Despite these drawbacks, however, for most real-world applications it makes sense to use an API Gateway.

Implementing an API Gateway (賣產品)

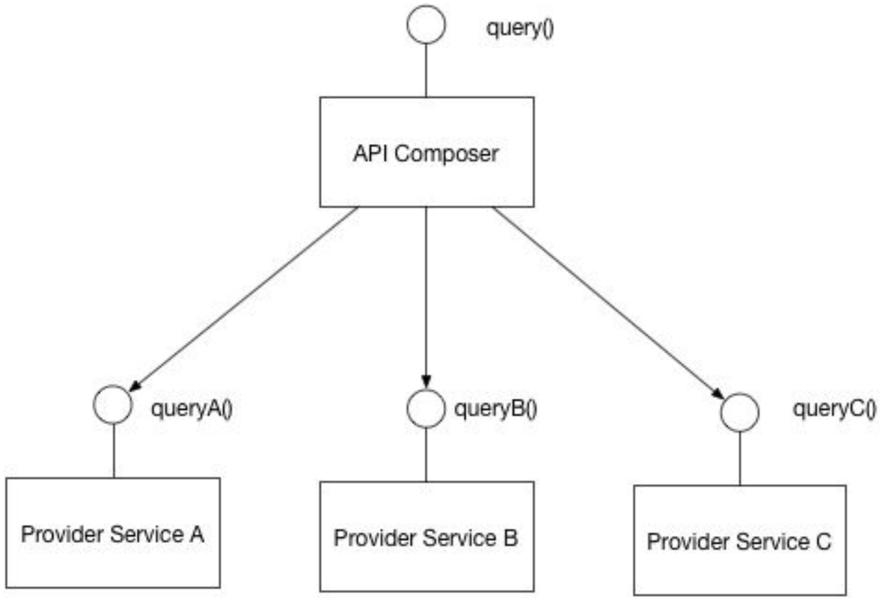
Performance and Scalability

- Only a handful of companies operate at the scale of **Netflix** and need to handle **billions of requests per day.**
- It makes sense, therefore, to build the API Gateway on a platform that supports **asynchronous, non-blocking I/O.**
- On the JVM you can use one of the NIO-based frameworks such **Netty, Vertx,** Spring Reactor, or JBoss Undertow. One popular non-JVM option is **Node.js**.
- **NGINX Plus** o ers a mature, scalable, high-performance web server and reverse proxy that is easily deployed, configured, and programmed.

Authencation before

Validation the request

using the traditional async callback approach quickly leads you to callback hell.



https://microservices.io/patterns/data/api-composition.html

Using a Reactive Programming Model

- **CompletableFuture** in Java 8
- **Promise** in JavaScript
- **Reactive Extensions** (also called Rx or ReactiveX), in Microsoft.NET Platform

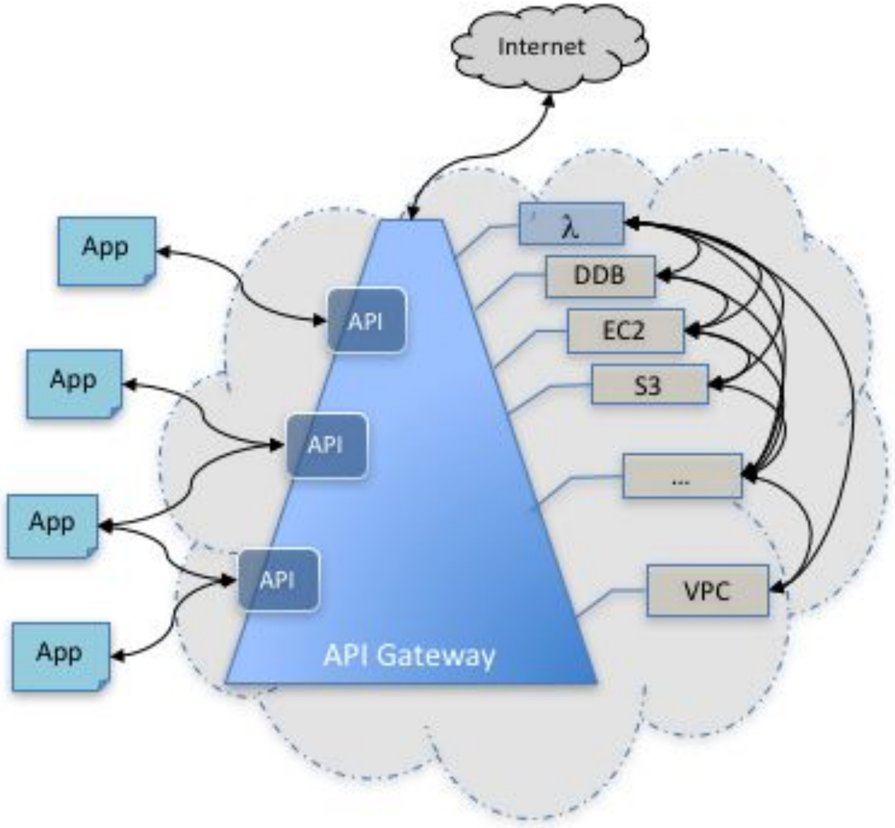
Service Invocation

- A microservices-based application is a **distributed system** and must use an **inter-process communication (IPC, Chapter 3) mechanism**.
    - One option is to use an **asynchronous, messaging-based mechanism.** Some implementations use a message broker such as **JMS** or **AMQP**. Others, such as **Zeromq**, are brokerless and the services communicate directly.
    - The other style of inter-process communication is a **synchronous mechanism** such as **HTTP** or **Thrift**.
- Consequently, the API Gateway will need to support a variety of **communication mechanisms.**

Service Discovery

- The API Gateway needs to know the location (IP address and port) of each microservice with which it communicates.
- in a modern, cloud-based microservices application, finding the needed locations is a **non-trivial problem.**
- determining the location of an application service is not so easy, because of **autoscaling** and **upgrades.**
- service discovery mechanism: either **server-side discovery** or **client-side discovery** Chapter 4

Single Entry Point



[Amazon API Gateway]

[Overview API Gateway]

Resource Discovery on AWS

- Security Groups
- IAM Roles
- Resource Tags
- AWS SDK / CLI

```
TAG="ops:status"
VALUE="retired"

# 找出標記 retire 的機器
aws ec2 describe-instances \
 --query 'Reservations[*].Instances[*].[InstanceId]' \
 --filters Name=tag:$TAG,Values=$VALUE\
 --output text |
 while IFS= read -r item
 do
   # 把 termination protection 關掉
   aws ec2 modify-instance-attribute \
     --instance-id $item \
     --no-disable-api-termination

   # terminate EC2 instance
   aws ec2 terminate-instances --instance-ids $item
 done
```

Ops as Code with AWS CLI

Handling Partial Failures

- This issue arises in all distributed systems whenever one service calls another service that is either **responding slowly** or is **unavailable**.
- For example, if the recommendation service is unresponsive in the product details scenario, the API Gateway should return the rest of the product details to the client since they are still useful to the user.
- The API Gateway could also return cached data if that is available.

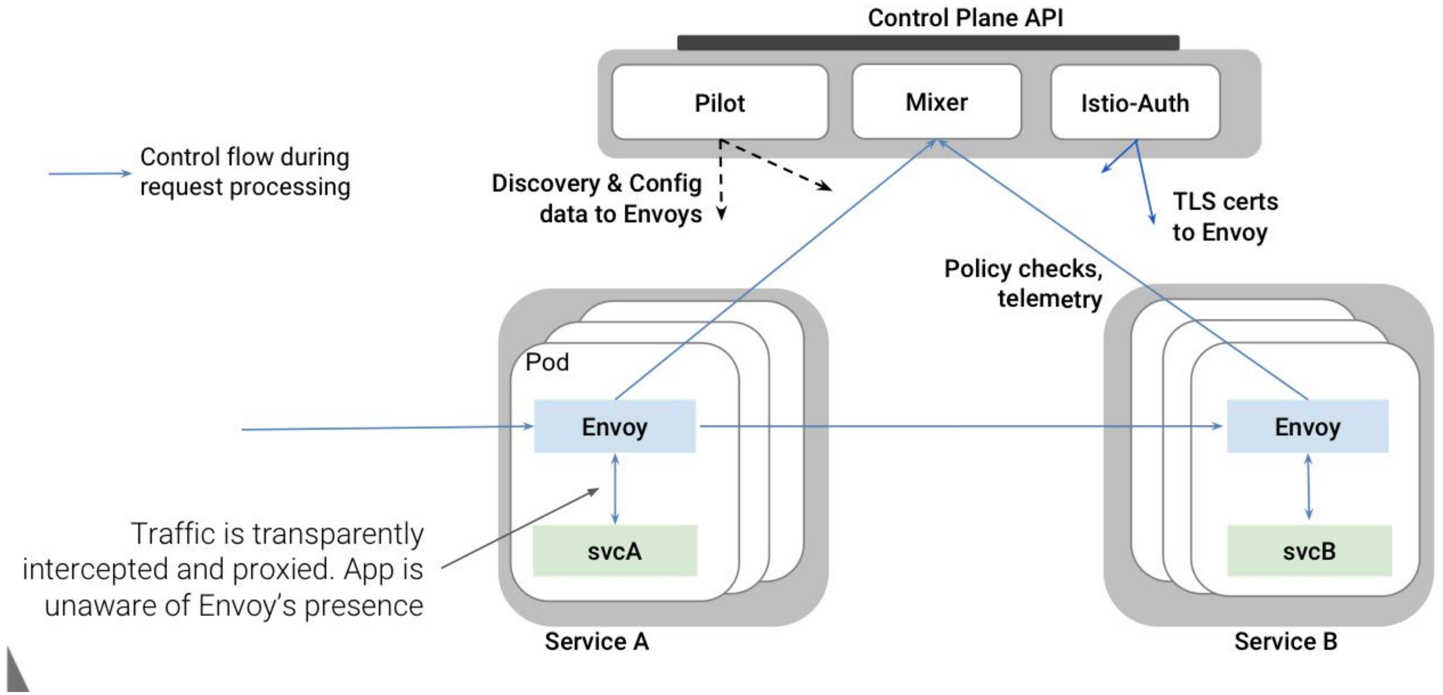
Netflix Hystrix (豪豬)

- Hystrix is a **latency** and fault tolerance **library** designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.
- implement **circuit breaker pattern**
- If the error rate for a service exceeds a specified threshold, Hystrix trips the circuit breaker and **all requests will fail immediately** for a specified period of time. (service 的 error rate 超過指定的臨界值，Hystrix 跳開斷路器，在一段時間之 內立即中短所有的請求。)
- JVM base.

補充：Service Mesh

- 一種基礎架構 (infrastructure layer) 的服務，負責處理的是 Service 跟 Service 之間通訊的安全、可靠、速度。
- 現代網路的基礎協議是 TCP/IP，Microservice 的通訊就是 Service Mesh

Implementation: Envoy

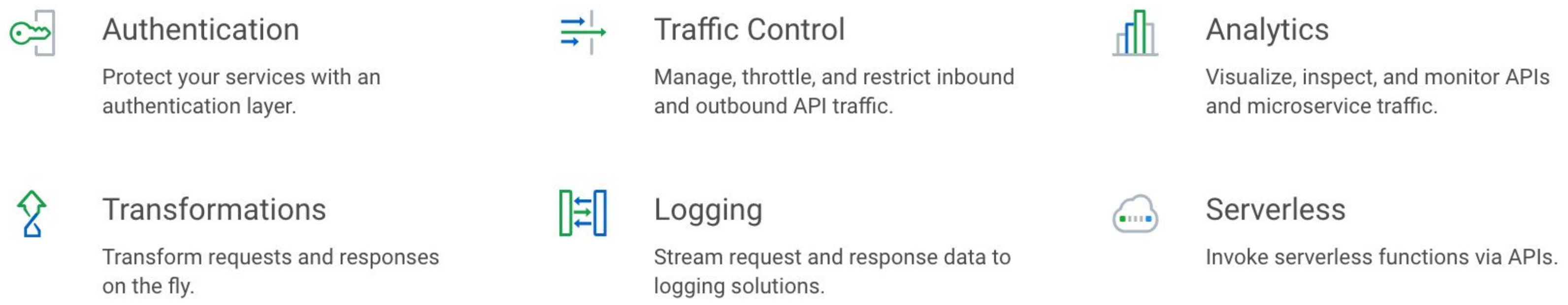

Control Plane API
Pilot
Mixer
Istio-Auth
Control flow during request processing
Discovery & Config data to Envoys
TLS certs to Envoy
Policy checks, telemetry
Pod
Envoy
svcA
Envoy
svcB
Traffic is transparently intercepted and proxied. App is unaware of Envoy's presence
Service A
Service B

Summary

1. makes sense to implement an API Gateway which acts as a **single entry point** into a system
2. responsible for **request routing, composition, and protocol translation**
3. provides each of the application’s clients with a **custom API.**
4. mask failures in the backend services by returning cached or default data

API Gateway Features

https://konghq.com/kong-community-edition/



Authentication

Protect your services with an authentication layer.

Traffic Control

Manage, throttle, and restrict inbound and outbound API traffic.

Analytics

Visualize, inspect, and monitor APIs and microservice traffic.

Transformations

Transform requests and responses on the fly.

Logging

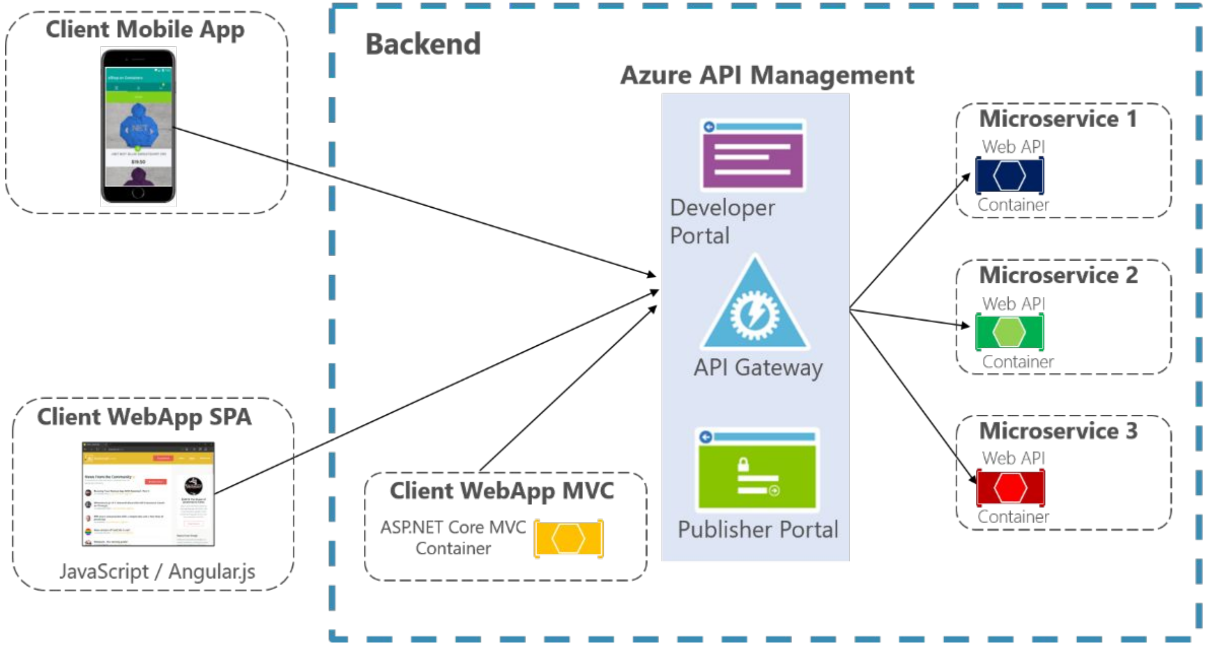
Stream request and response data to logging solutions.

Serverless

Invoke serverless functions via APIs.

API Gateway with Azure API Management

Architecture



https://docs.microsoft.com/zh-tw/dotnet/standard/microservices-architecture/architect-microservice-container-a
ions/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

Reference

- Microservices.io
- Production-Ready Microservices (Free, 120+)
- Building Microservices
- Microservice Patterns (Manning) - MEAP
- Microservices on AWS (AWS Whitepaper, PDF)
- AWS re:Invent 2017: Building Microservice on AWS
- AWS re:Invent 2016: From Monolithic to Microservices: Evolving Architecture Patterns