Building APIs with Microservices

Things I wish I'd known

James Gough

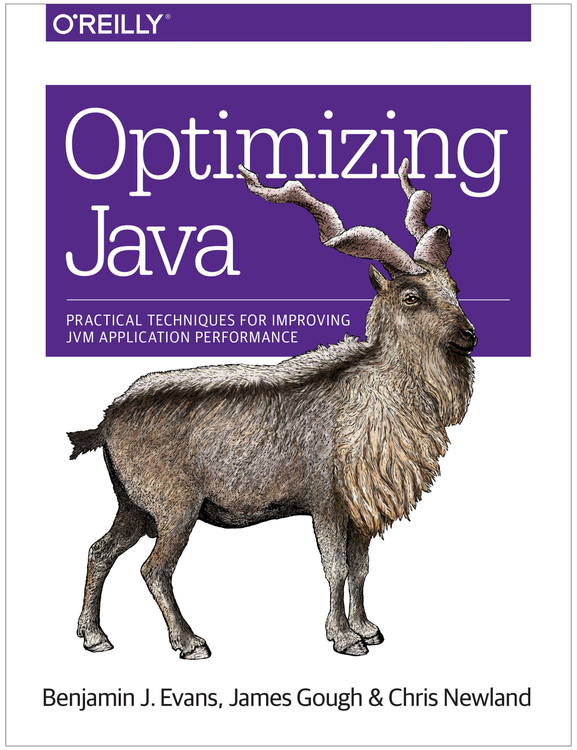
@Jim__Gough

Agenda

- What is an API?
- Problems faced when building APIs
- Designing and testing restful APIs
- API versioning concerns
- Developing APIs in a distributed team
- Microservices and Java
- Using Docker
- Scheduling containers
- Kubernetes
- Introducing Istio
- API Management

James (Jim) Gough

- Previous: Java developer and contributor
- Previous: Technical Trainer at Morgan Stanley and Bloomberg
- Current: Java Developer/Architect and API Technical Lead at Morgan Stanley
- Co-Author of Optimizing Java



London Java Community

What is an API?

“API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you’re using an API”

MuleSoft

What is an API?

“An application-programming interface (API) is a set of programming instructions and standards for accessing a Web-based software application or Web tool.”

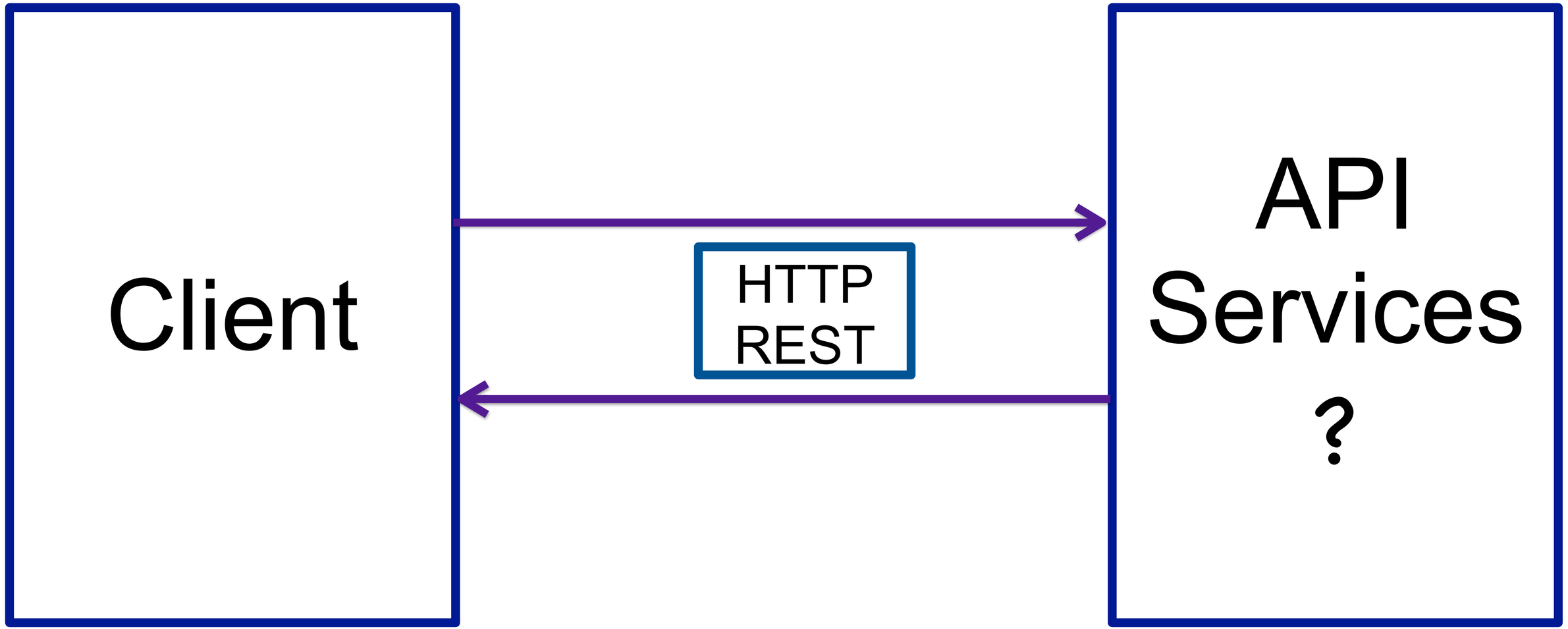
How Stuff Works

What is an API?

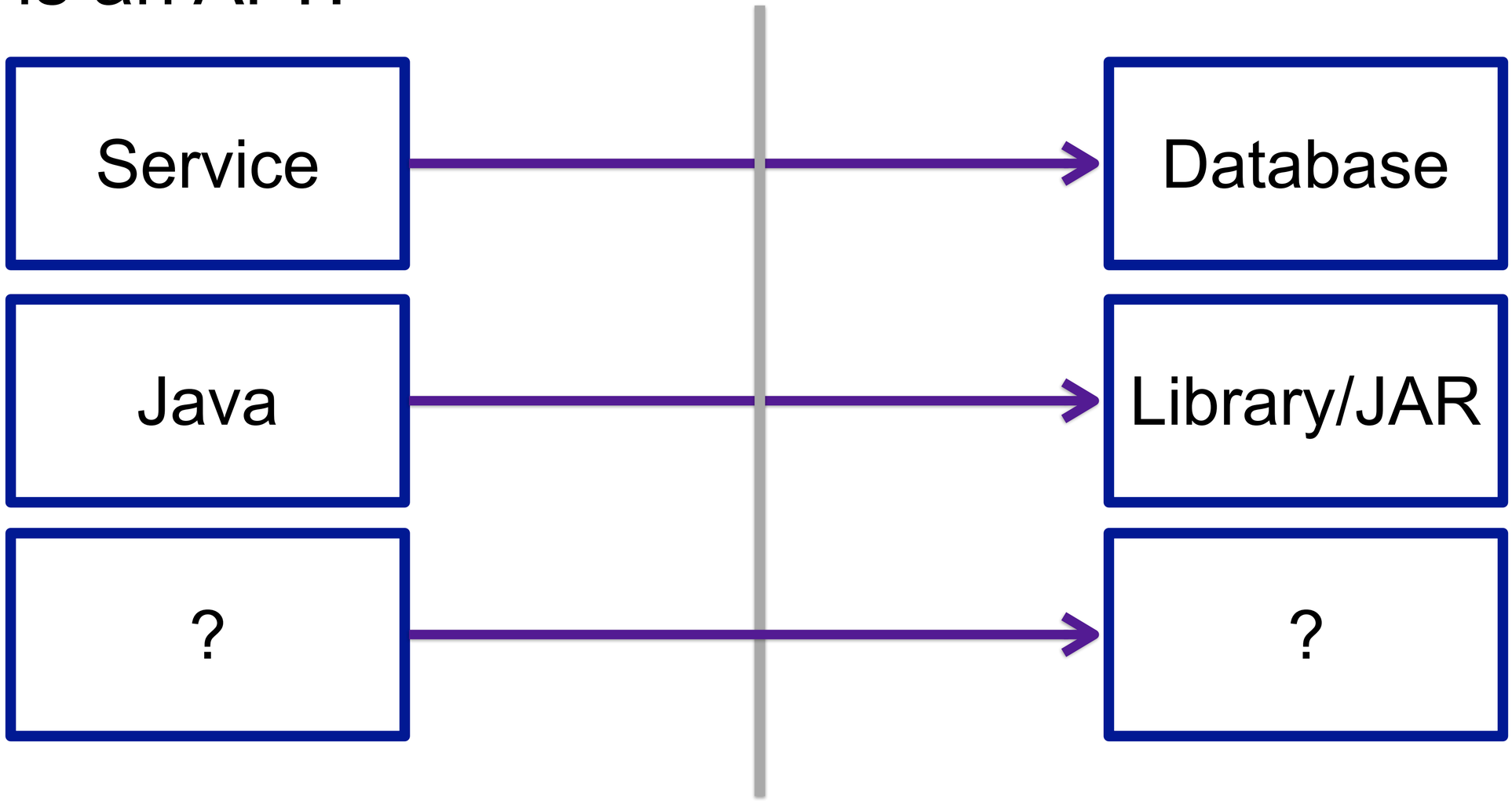
“Application programming interfaces hide complexity from developers, extend systems to partners, organise code, and make components reusable”

InfoWorld

What is an API?



What is an API?



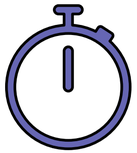
Problems Faced with all APIs

v 1.0
v 1.1

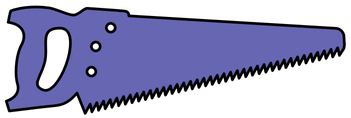
Versioning

Discovery

Security



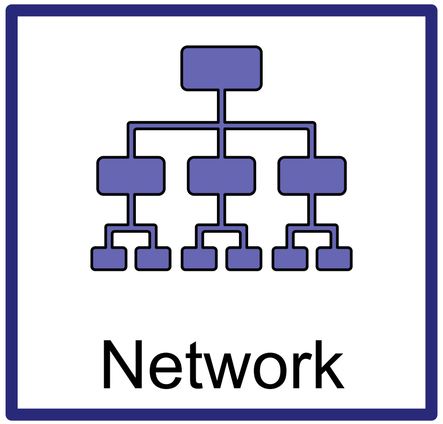
Performance

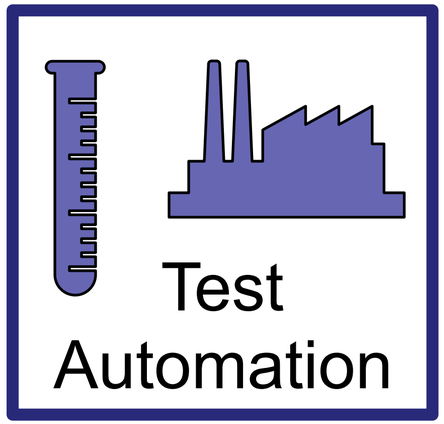


Quality

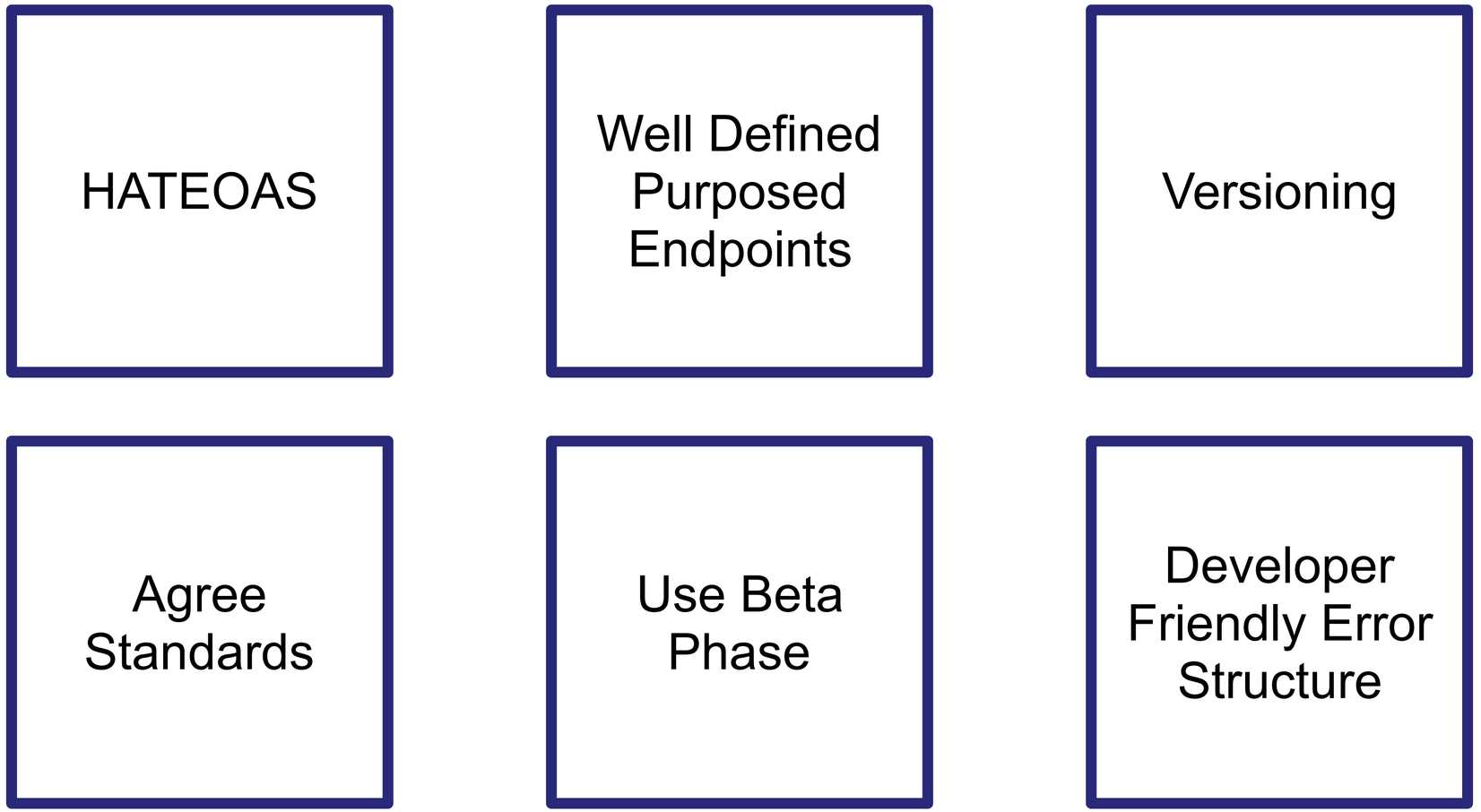
https://smartbear.com/SmartBear/media/ebooks/State-of-API-Report-2016.pdf

Decoupled API Challenges





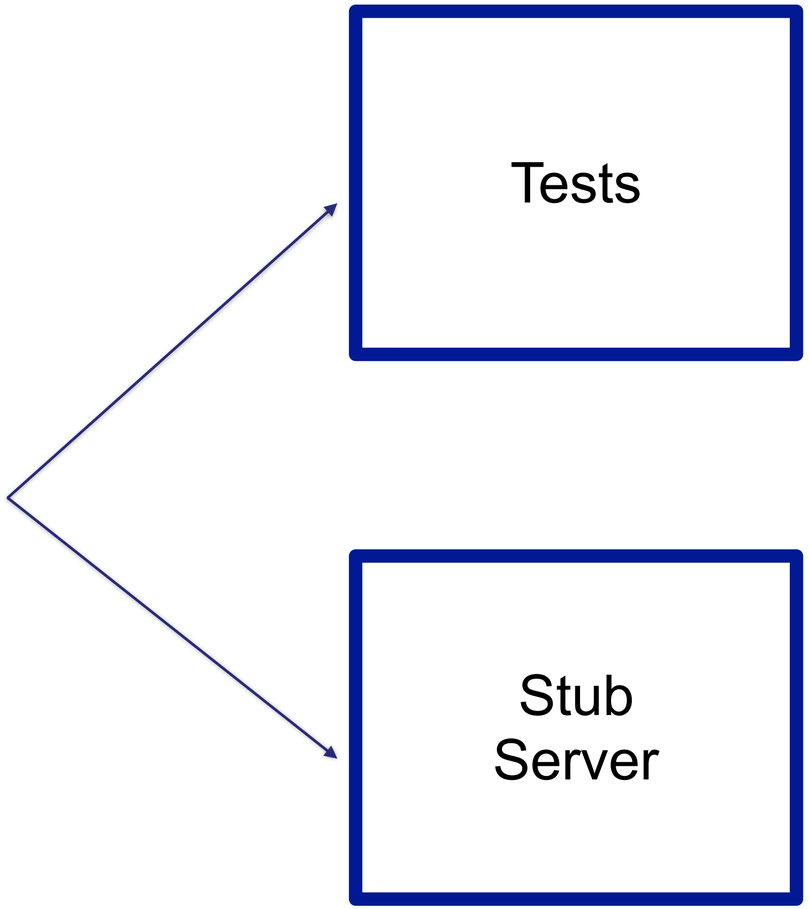
Designing Restful APIs



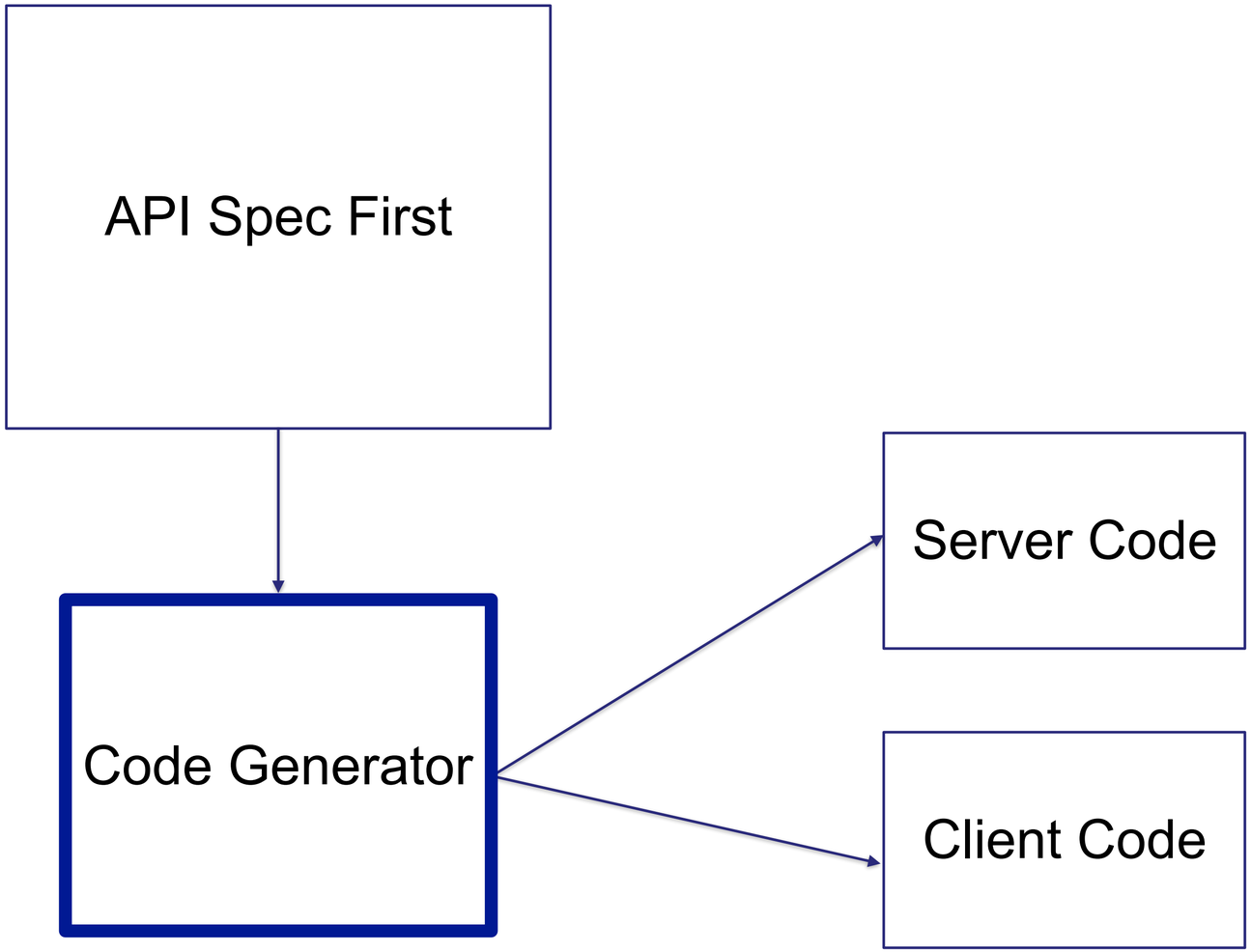
https://github.com/paypal/api-standards

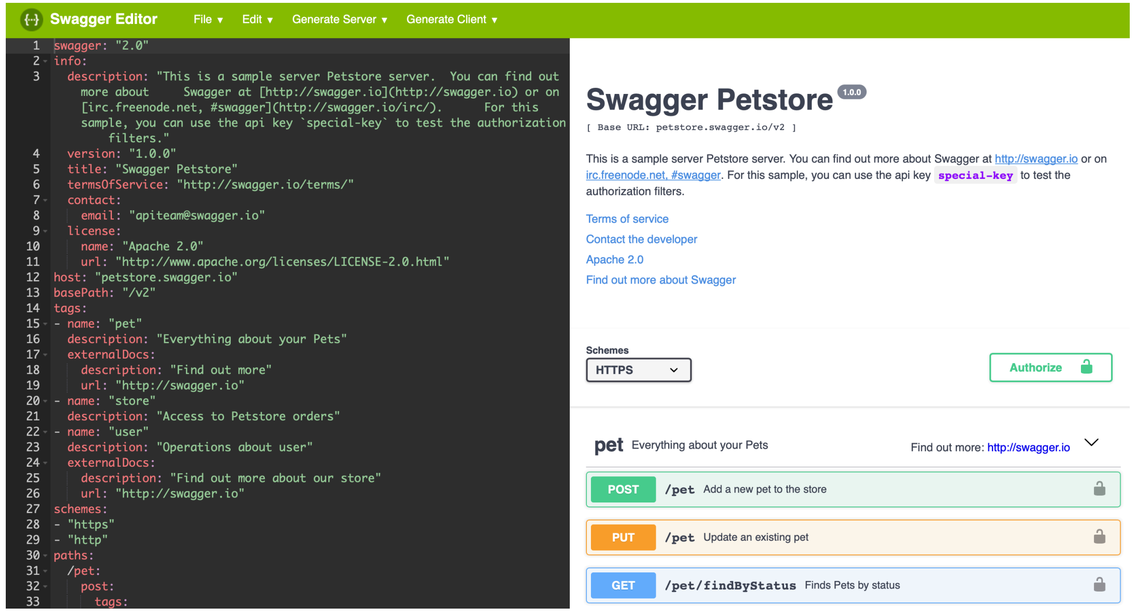
Contract Driven Development

```
org.springframework.cloud.contract.spec.Contract.make {
    request {
        method 'POST'
        body([
                "date": "2018-09-29"
             ])
        url '/day'
        headers {
          header('Content-Type', 'application/json')
        }
    }
    response {
        status 200
        body([
          "dayOfWeek": "SATURDAY"
        ])
    }
}
```



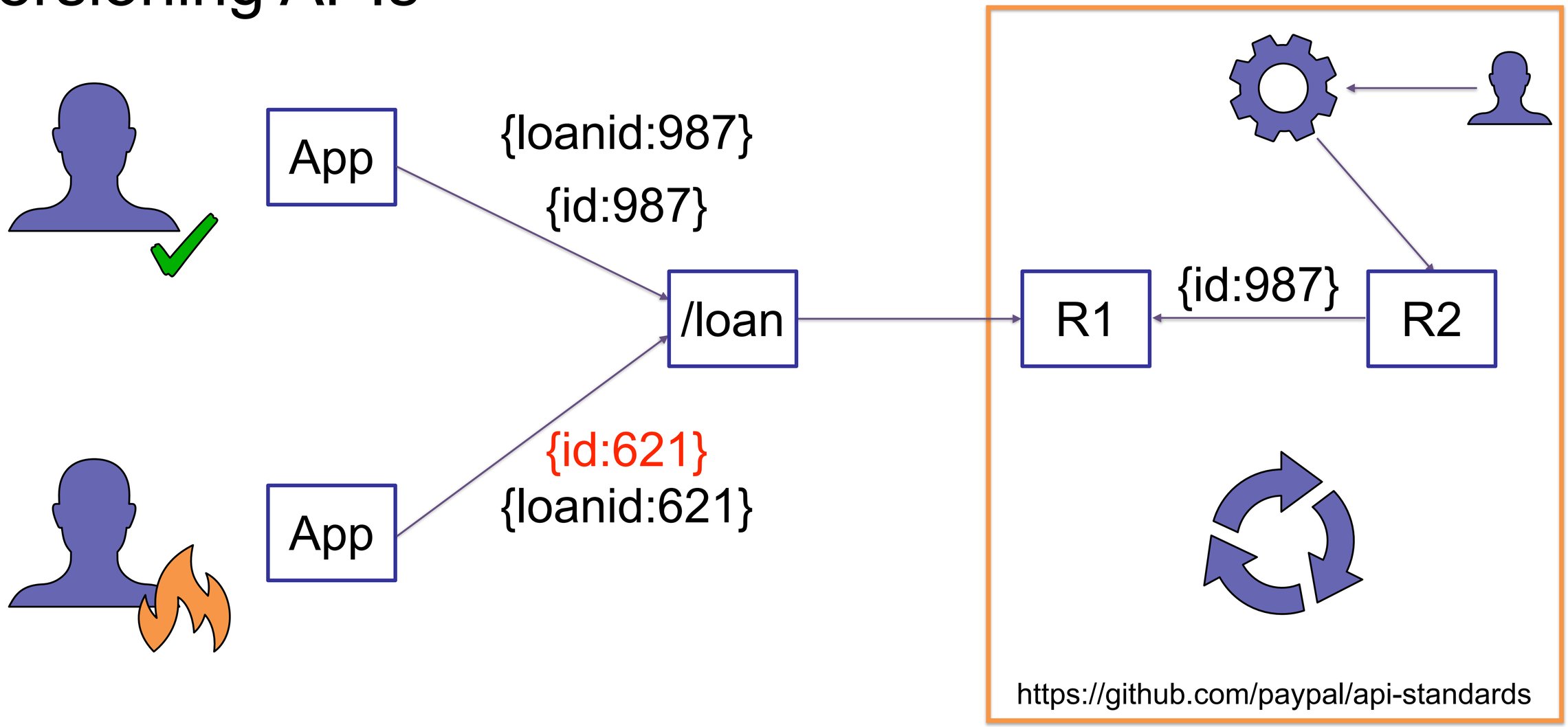
API Specification First



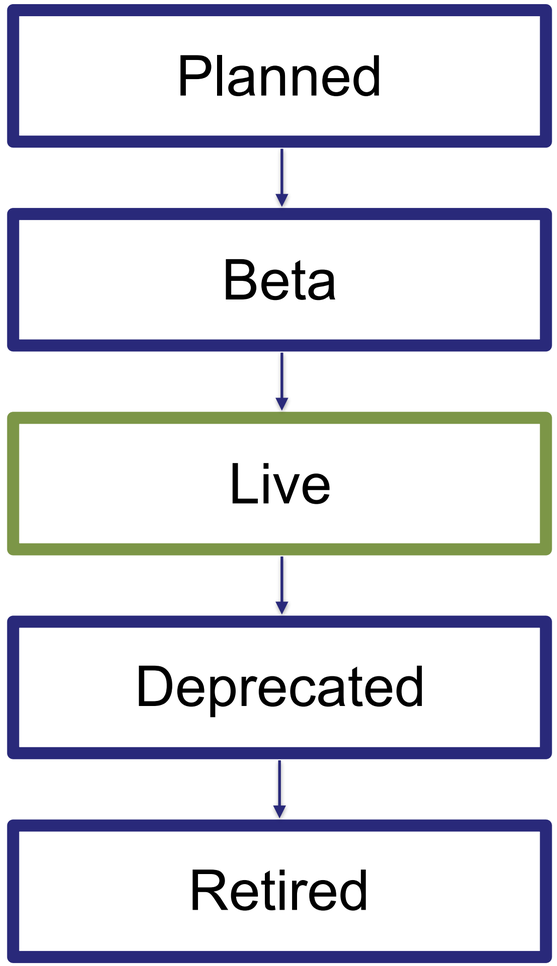


https://editor.swagger.io

Versioning APIs



API Lifecycle



Minor Release

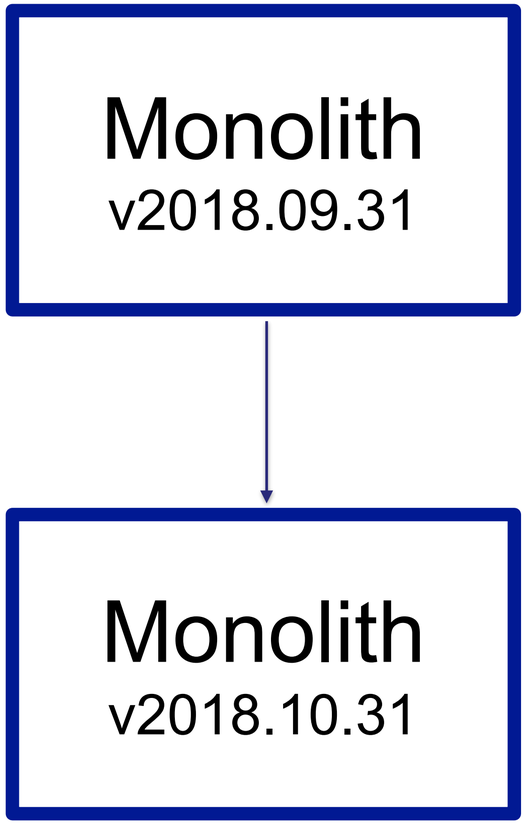
- Backwards compatible
- Customer does nothing
- Changes additive and optional
- Semantics don’t change

Major Release

- Can break compatibility
- Must have a migration plan
- Runs with deprecated API

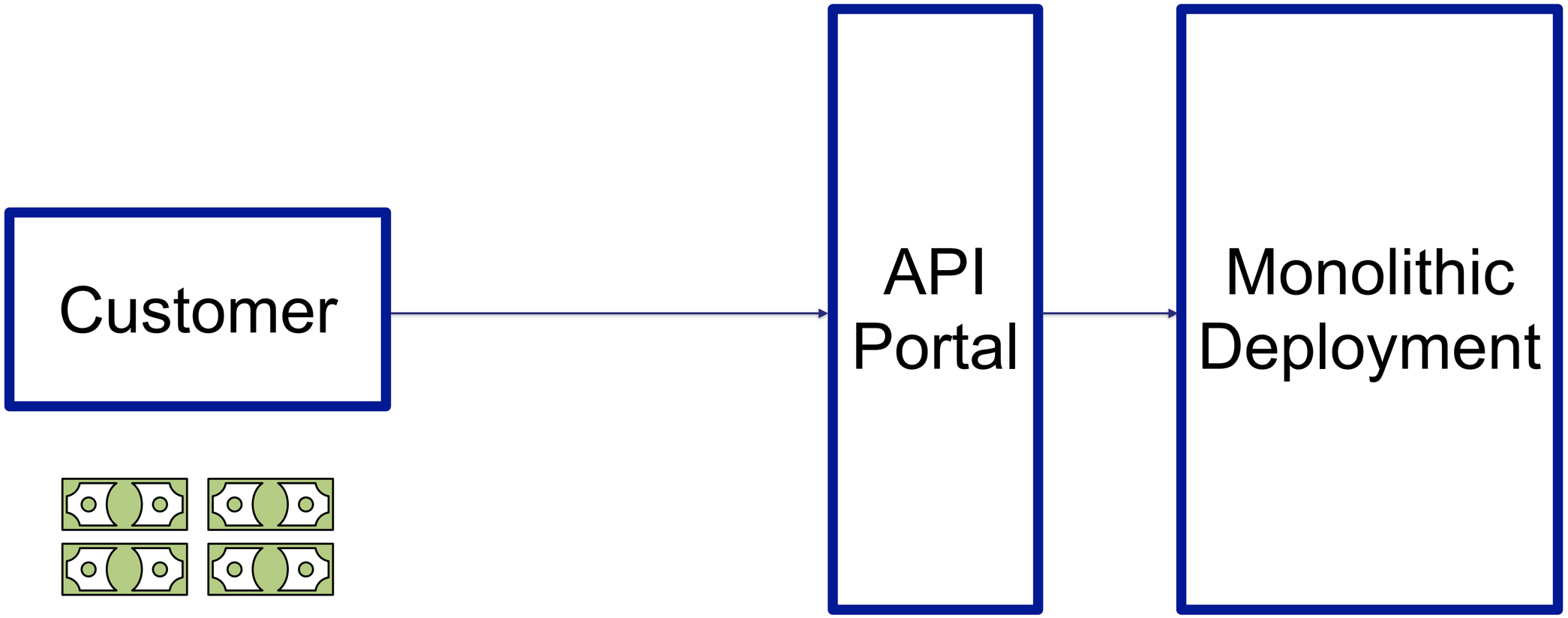
https://github.com/paypal/api-standards

Developing APIs in a Distributed Team

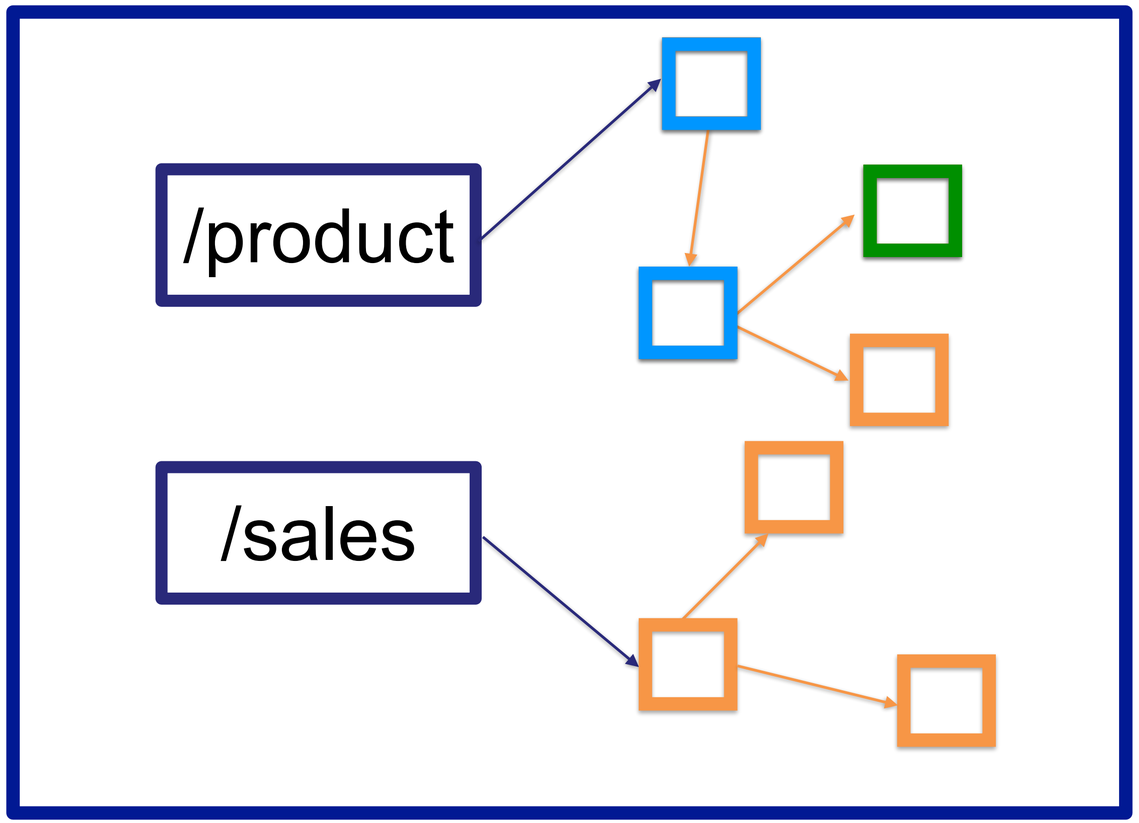


- ✓ Binary compatibility
- ✓ Build time consistency
- ✓ Easy to reason about
- ✓ Simple to deploy
- ✗ Does not scale.. at all

Delivering a Simple API

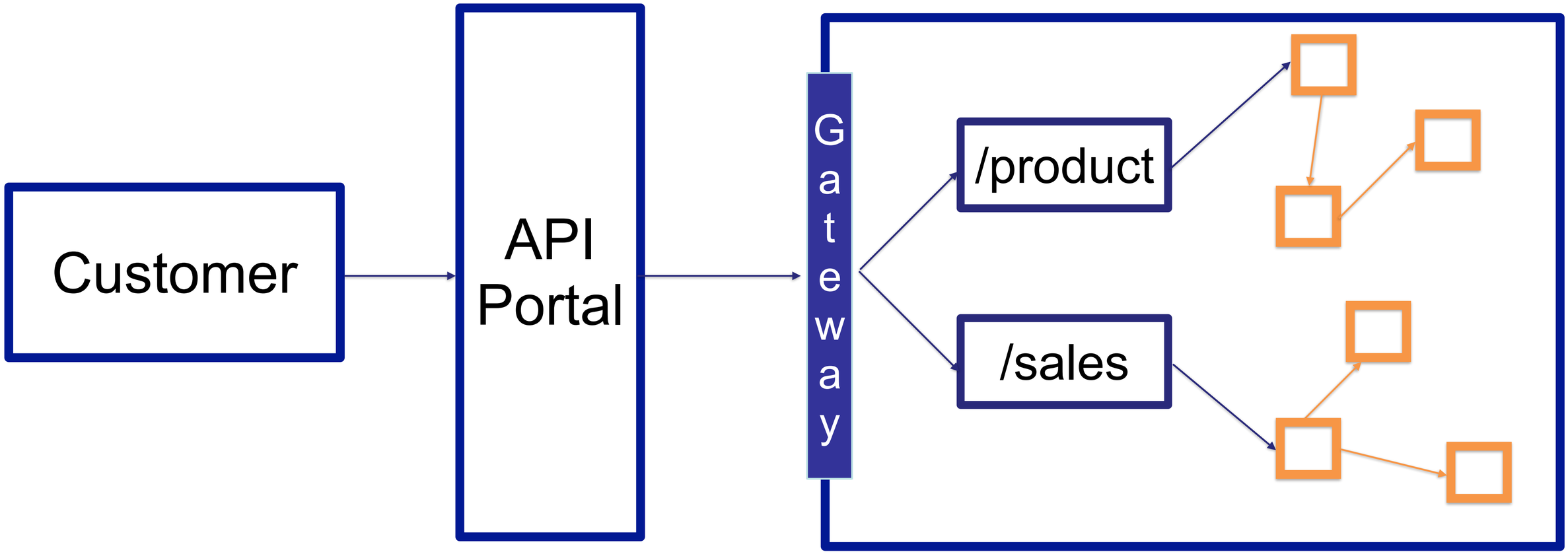


Developing APIs in a Distributed Team

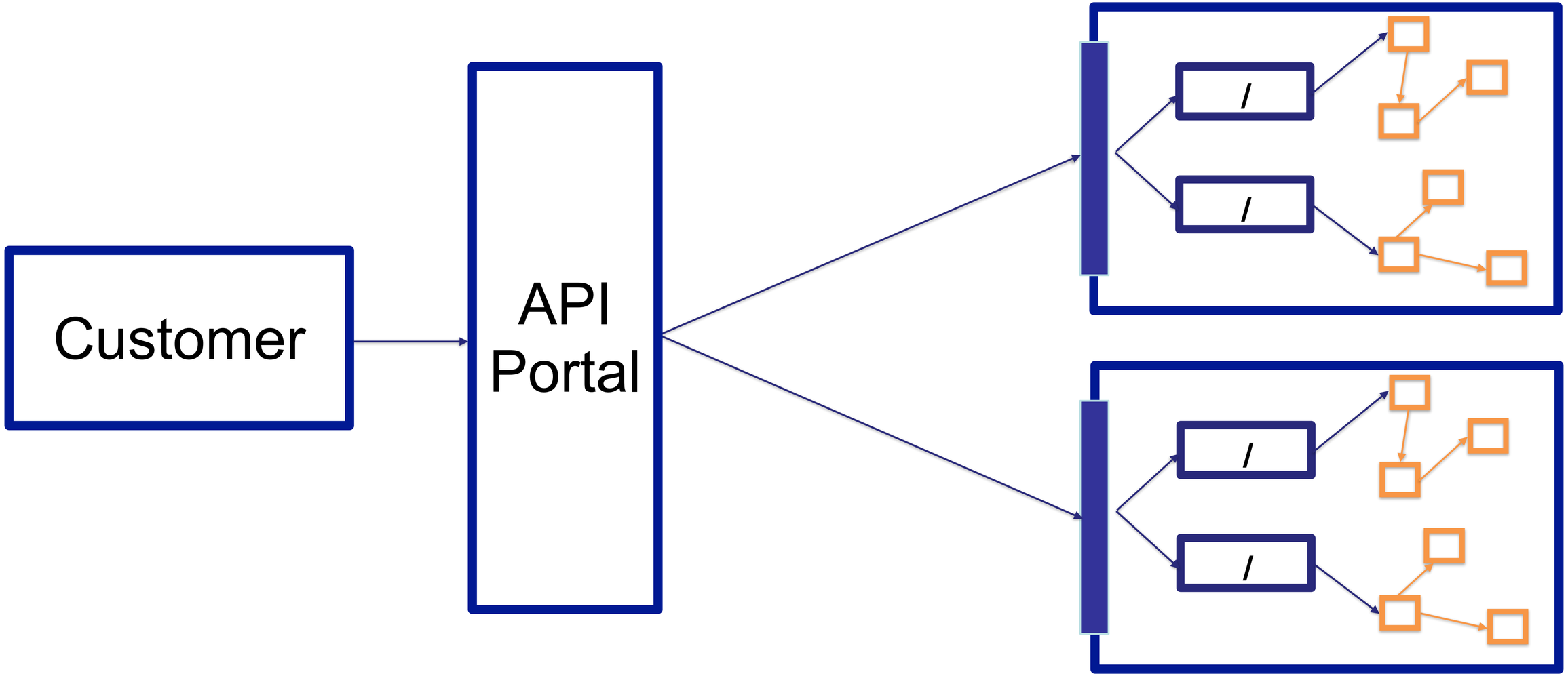


- All API deployments must move freely
- Full automated ephemeral testing available
- Replicated in a UAT environment for testing
- Expect many deployments per day
- Easy roll back
- Enhance test cases if issues found in Prod

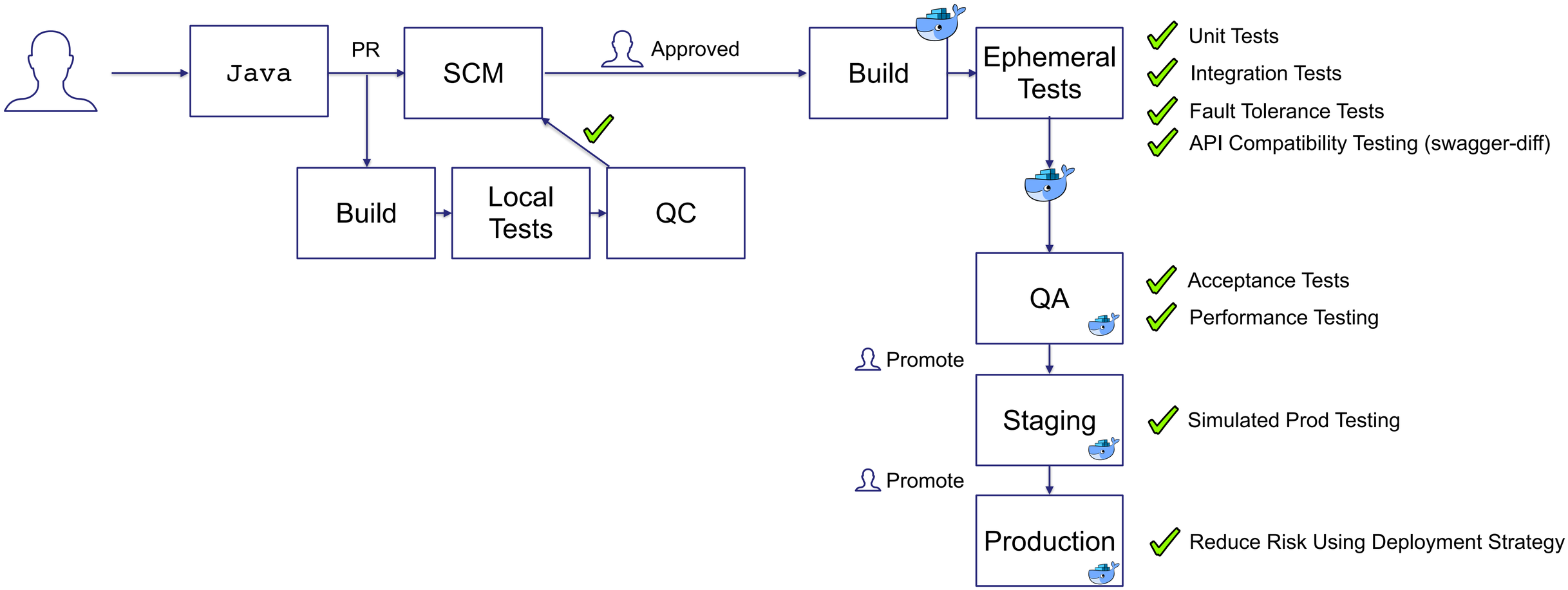
Delivering an API Program



Delivering an API Program



Continuous API Delivery



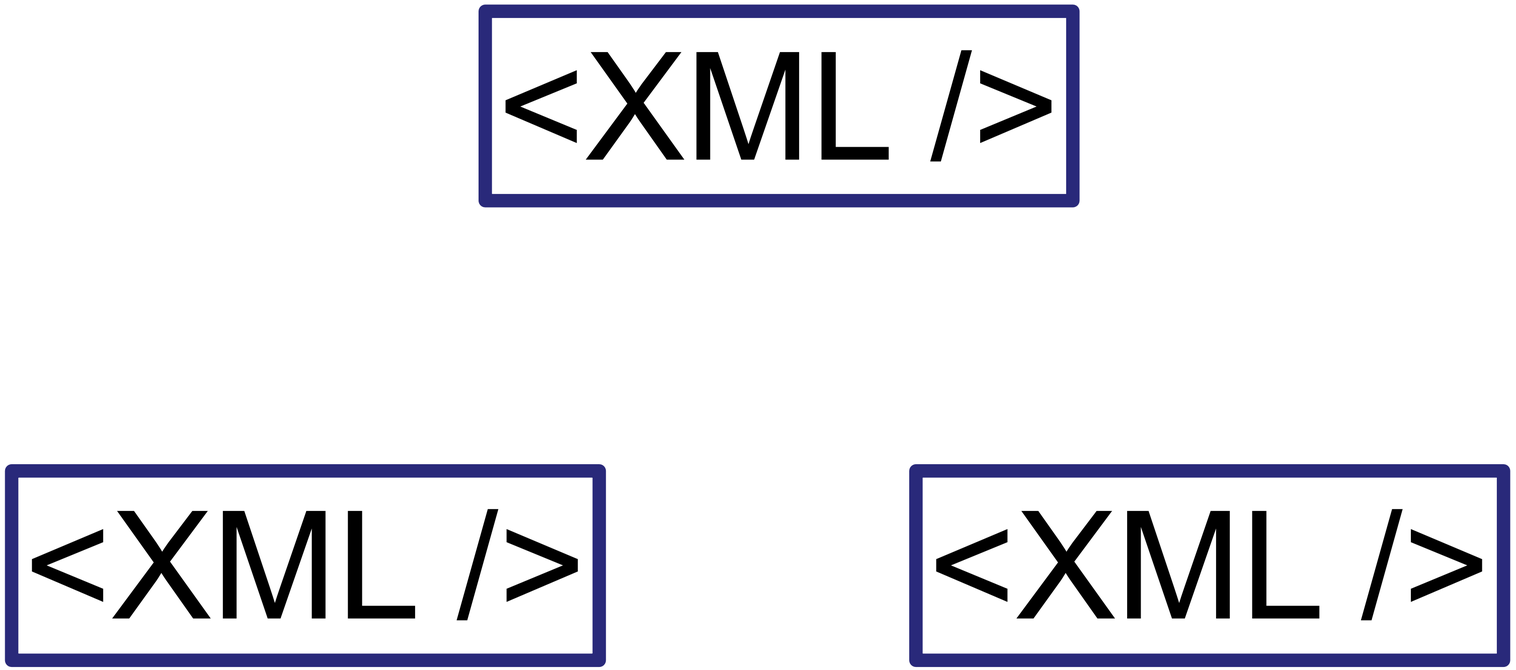
Daniel Bryant - Continuous Delivery in Java

Let’s Go! Full Microservices?

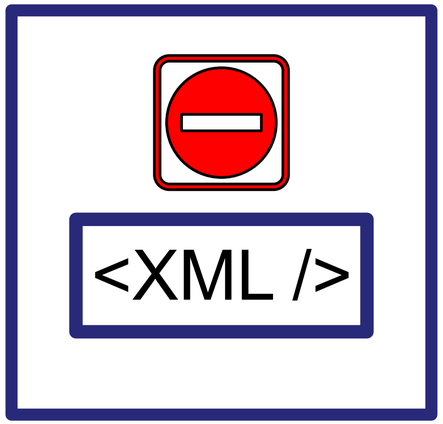
	Level 0 Traditional	Level 1 Basic	Level 2 Intermediate	Level 3 Advanced
Application	Monolithic	Service Oriented Integrations	Service Oriented Applications	API Centric
Database	One Size Fits All Enterprise DB	Enterprise DBs + No SQL	Polyglot, DBaaS	Matured Data Lake
Infrastructure	Physical Machines	Virtualization	Cloud	Containers
Monitoring	Infrastructure	App and Infra Monitoring	APMs	APM and Central Log Management
Process	Waterfall	Agile and CI	CI & CD	DevOps

Spring 5.0 Microservices - Second Edition by Rajesh R V

Building Microservices Using Java



Introducing Spring Boot



Stand Alone

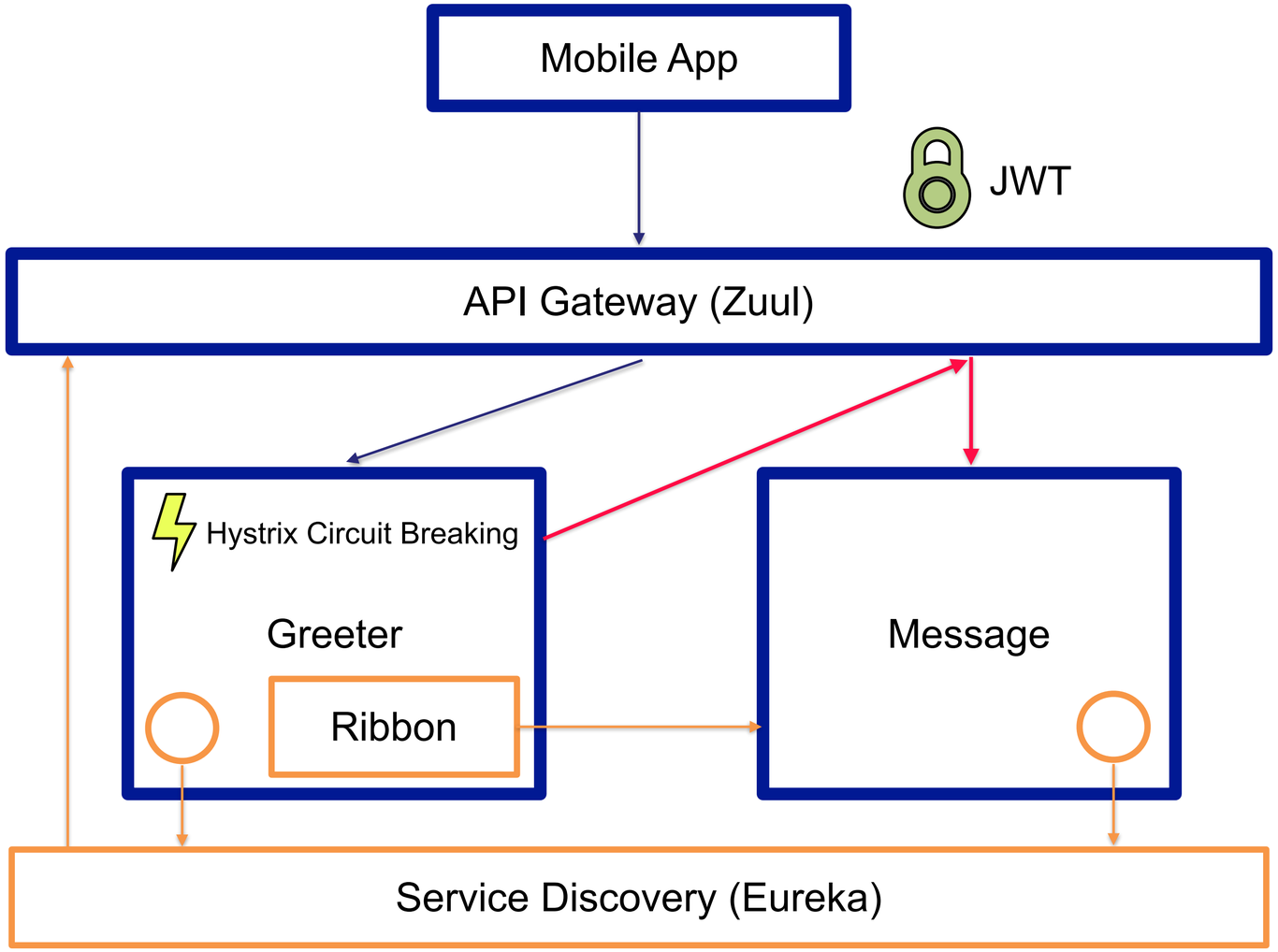
Demo Spring Boot

Java is All We Need

Spring Boot App

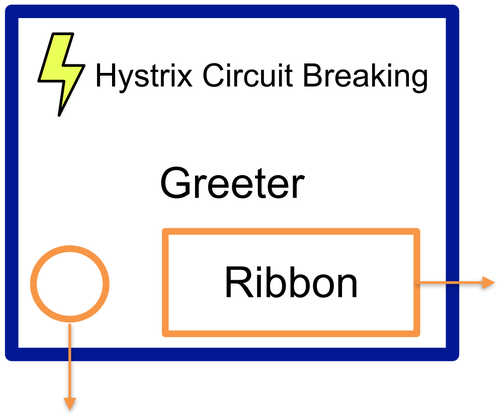
- Doesn’t work on my machine!
  - Different version of Java installed
  - Different file system supported
- Consistency between environments
  - Can’t be tested locally
- Coordinating Microservices is not just about Java
  - Use the right language for the job
- Coordinating Java is the wrong abstraction
  - Lots of scripts and Java specifics

The Netflix Stack

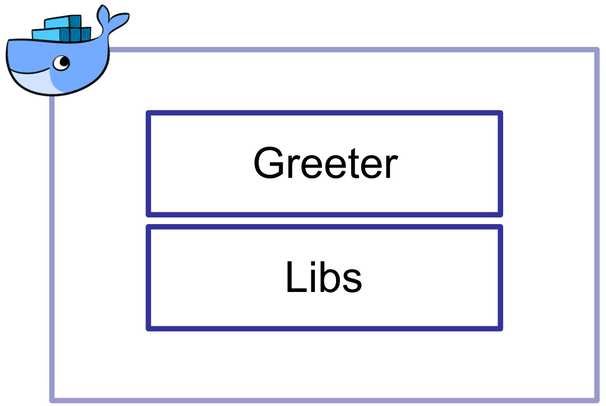


The Netflix Stack Today

- Provided a set of libraries to solve most microservices problems
- Are these services still micro?
- Doesn’t solve the problem of where to run services and deployment strategies
- Absolutely key in driving forward this complex technical space
- Environments can be challenging - a good place to start
- Can we do better? Is Java the wrong building block?



What is Docker?



REST API

Docker Engine

Docker Client | Docker Compose | Docker Swarm | Docker Registry

Generic

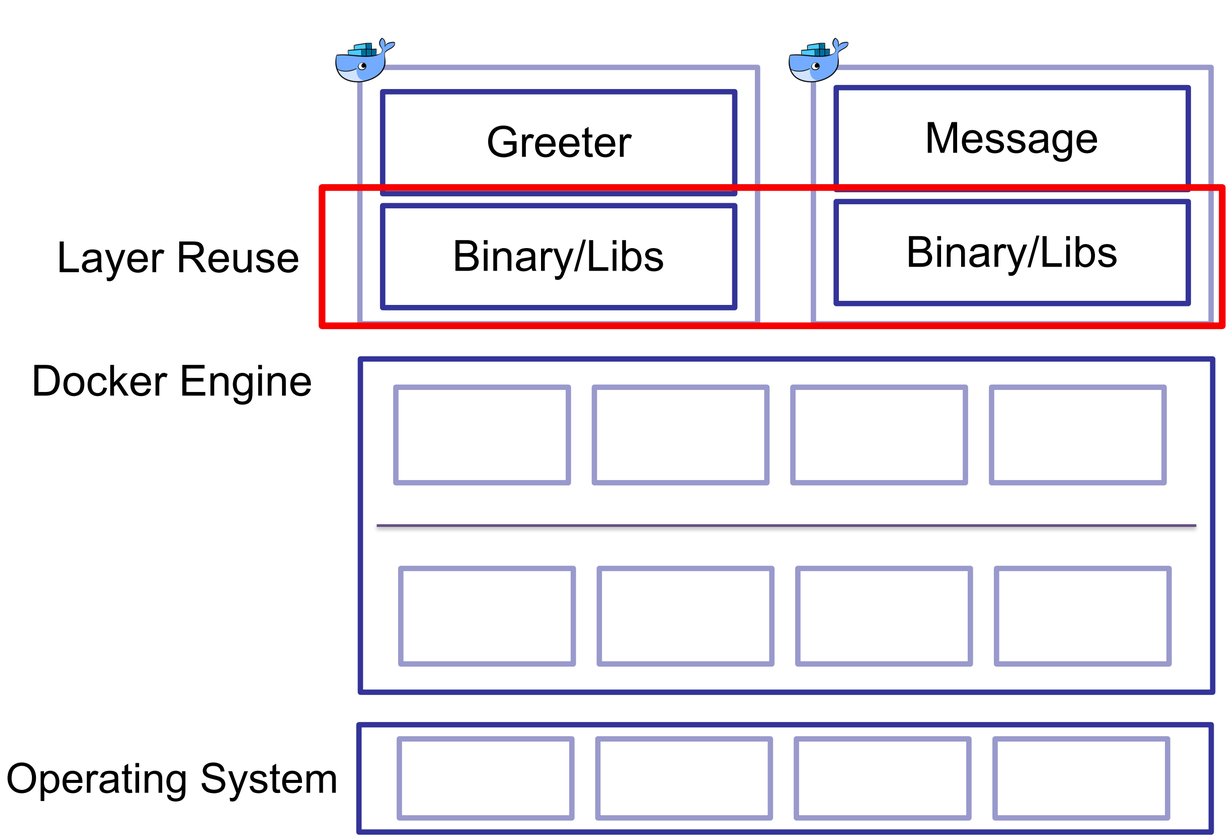
Platform Specific

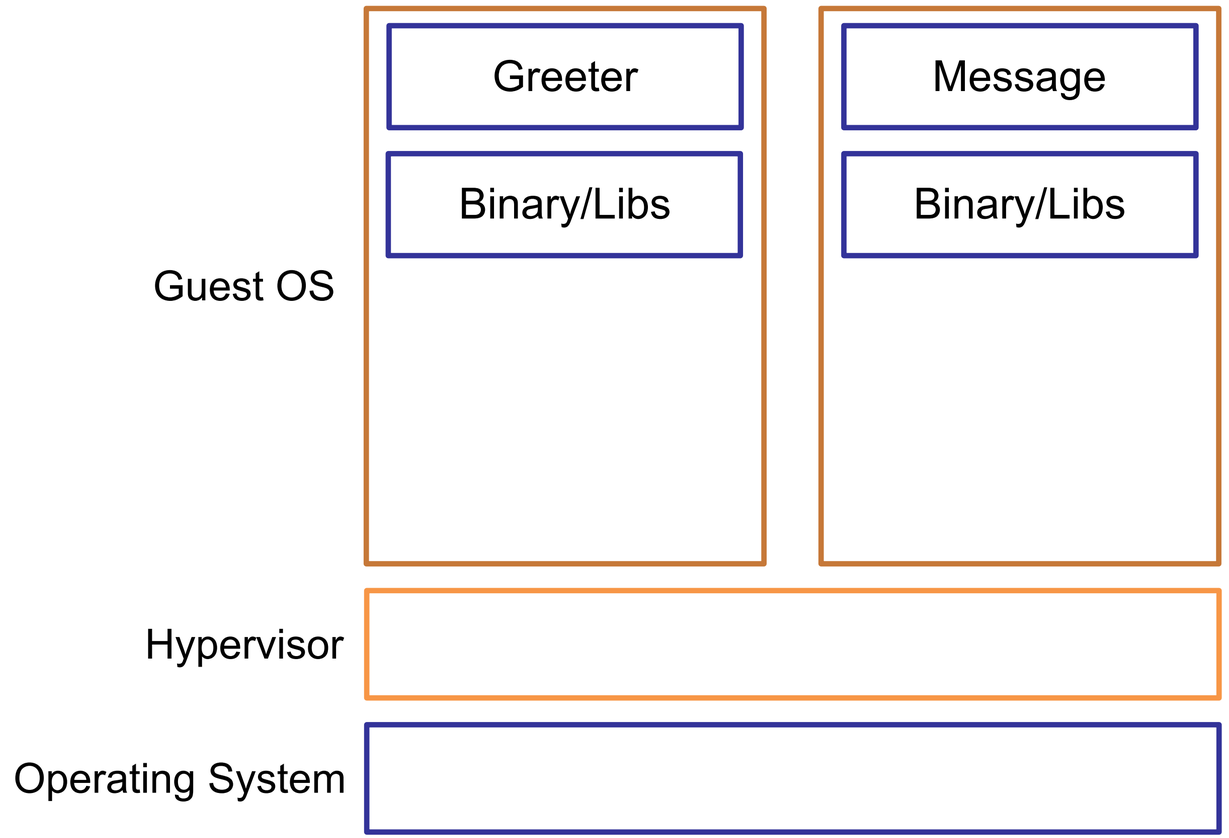
libcontainerd | lib network | graph | Plugins

Operating System

Namespaces | Control Groups | Layer Capabilities | Other OS Functions

Difference between a VM and Docker

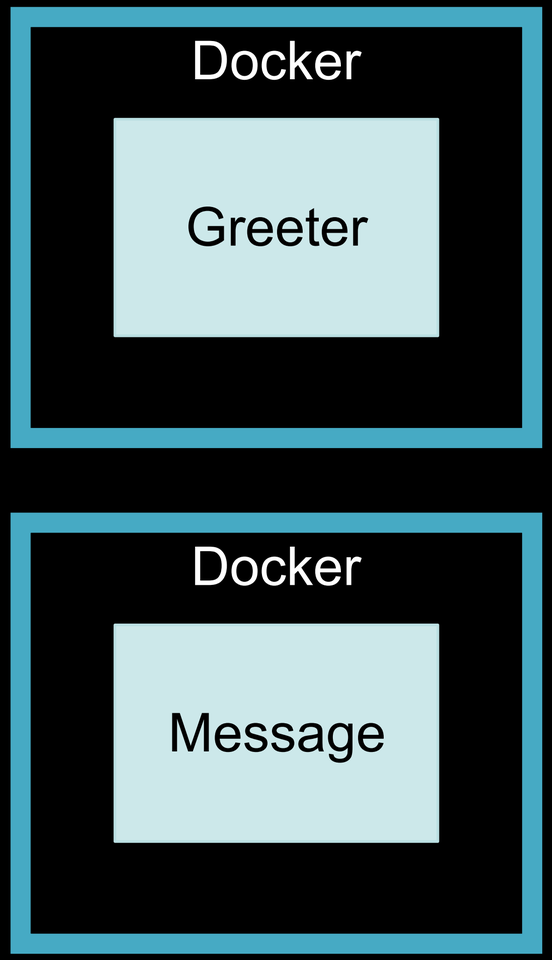




Advantages of using Containers

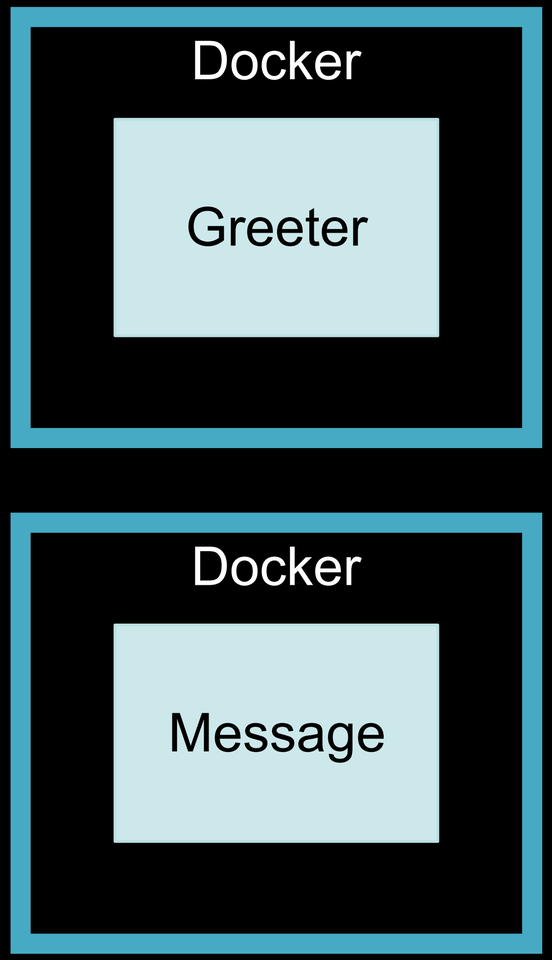
- Immutability
- Works across all environments and deployments
  - Easier to test
- Fundamental building block to simplify deployment
- Leave the specifics of the language behind

Demo Docker



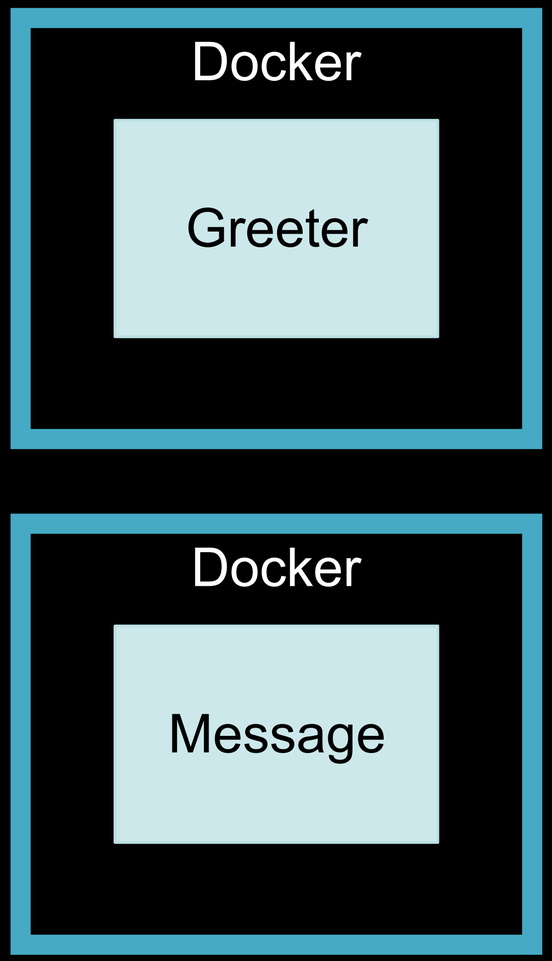

```
$ cat Dockerfile

FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD build/libs/greeter-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENV JAVA_OPTS=""
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -
Djava.security.egd=file:/dev/./urandom -jar
```

Demo Docker



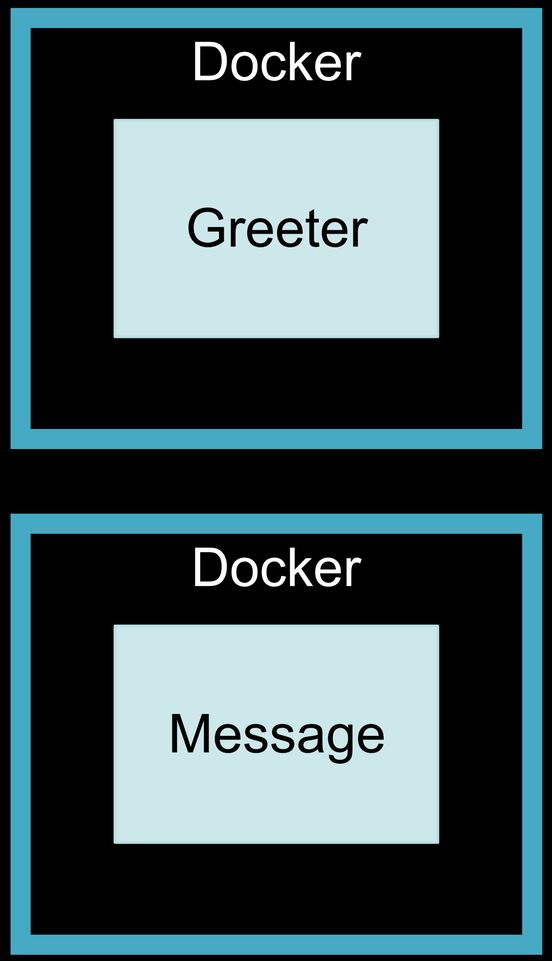

```
$ docker build -t greeter:conf .

Sending build context to Docker daemon  24.29MB
Step 1/6 : FROM openjdk:8-jdk-alpine
 ---> 54ae553cb104
Step 2/6 : VOLUME /tmp
 ---> Using cache
 ---> be3f9b66169a
Step 3/6 : ADD build/libs/greeter-0.0.1-SNAPSHOT.jar app.jar
 ---> Using cache
 ---> 9ebc7b30c927
Step 4/6 : EXPOSE 8080
 ---> Using cache
 ---> 13e6d1568396
Step 5/6 : ENV JAVA_OPTS=""
 ---> Using cache
 ---> 9659c9657c0b
Step 6/6 : ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -
Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
 ---> Using cache
 ---> 7622a23bac14
Successfully built 7622a23bac14
Successfully tagged greeter:conf
```

Demo Docker



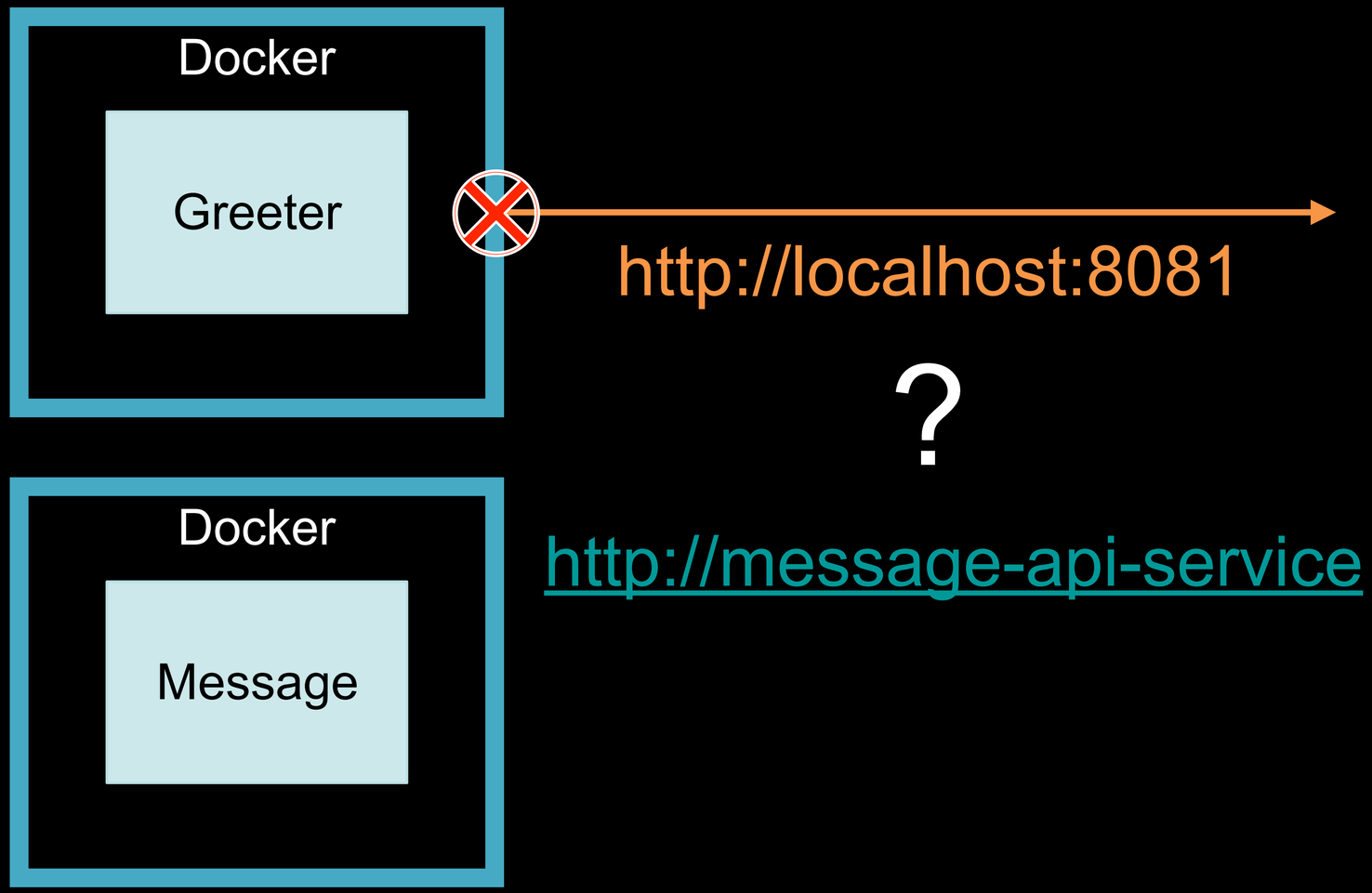

```
$ docker run -p 8080:8080 -t
greeter:conf
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.6.RELEASE)

2018-10-27 12:24:50.059  INFO 1 --- [           main]
com.oreilly.greeter.GreeterApplication   : Starting
GreeterApplication on 5db7d230d95e with PID 1 (/app.jar
started by root in /)

$ docker run -p 8081:8081 -t
message:conf
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
```

Demo Docker

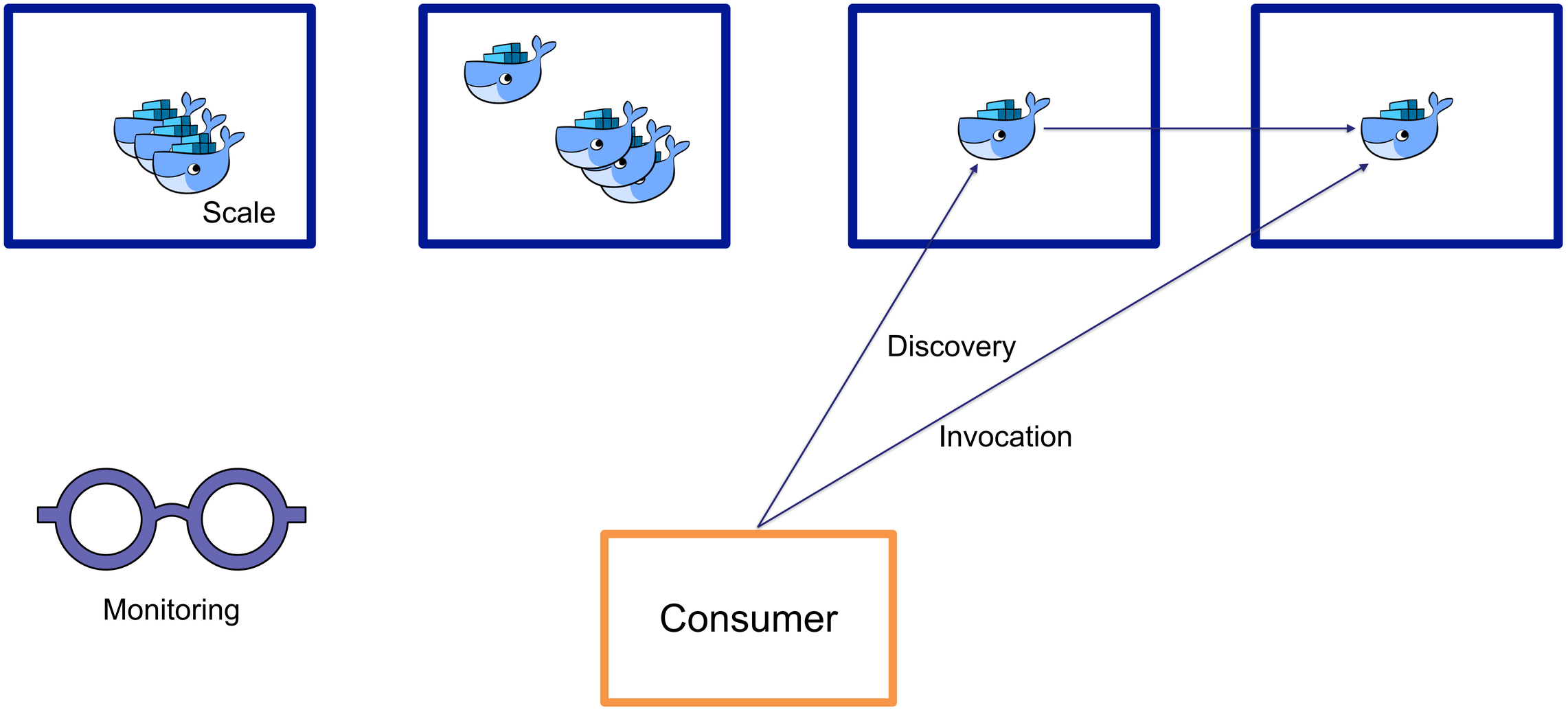



```
$ docker ps

CONTAINER ID        IMAGE                  COMMAND
CREATED              STATUS                PORTS
NAMES
224ab1a98b3f        message:conf           "sh -c 'java
$JAVA_O…"   About a minute ago   Up About a minute
0.0.0.0:8081->8081/tcp   naughty_leakey

912c04bda714        greeter:conf           "sh -c 'java
$JAVA_O…"   2 minutes ago        Up 2 minutes
0.0.0.0:8080->8080/tcp   boring_hermann

$ curl http://localhost:8080/simple
{"message":"Hello Conference”}

$ curl http://localhost:8080/custom
{"timestamp":"2018-10-27T12:29:45.679+0000","status":
500,"error":"Internal Server Error"
```

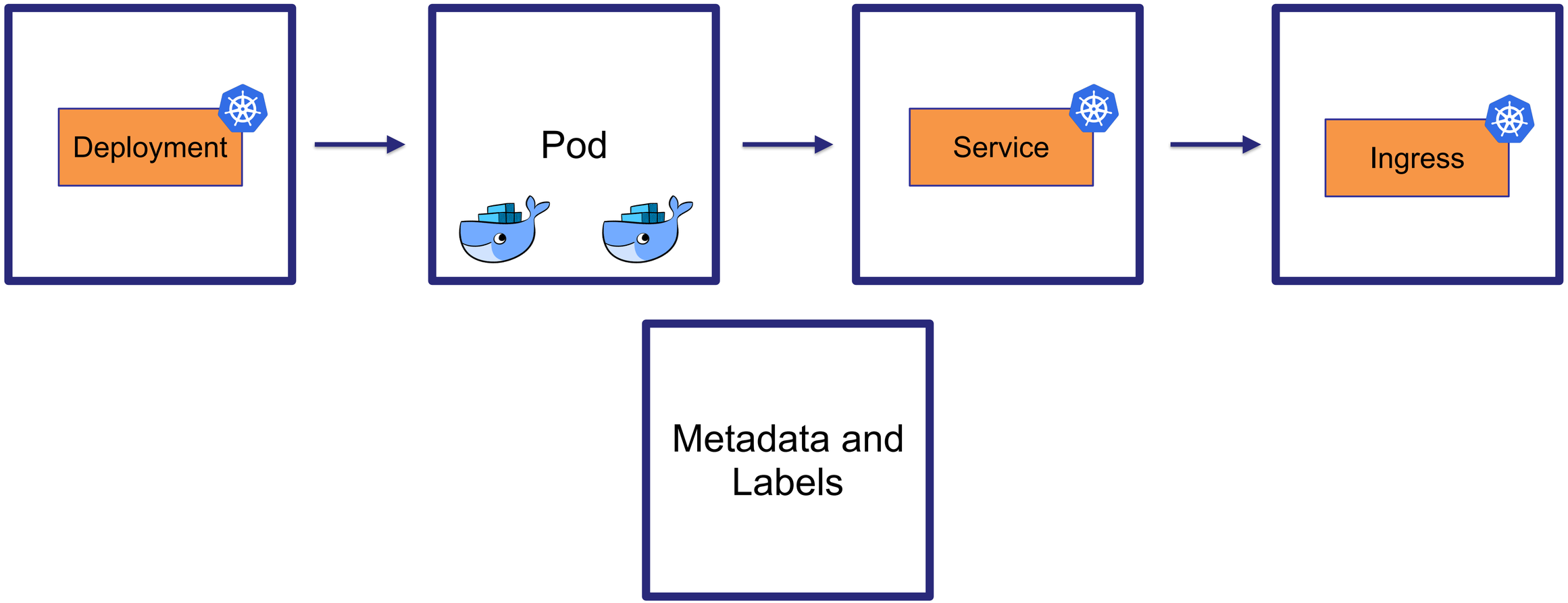
Demo Docker


Docker
Greeter
http://localhost:8081
?
Docker
Message
http://message-api-service

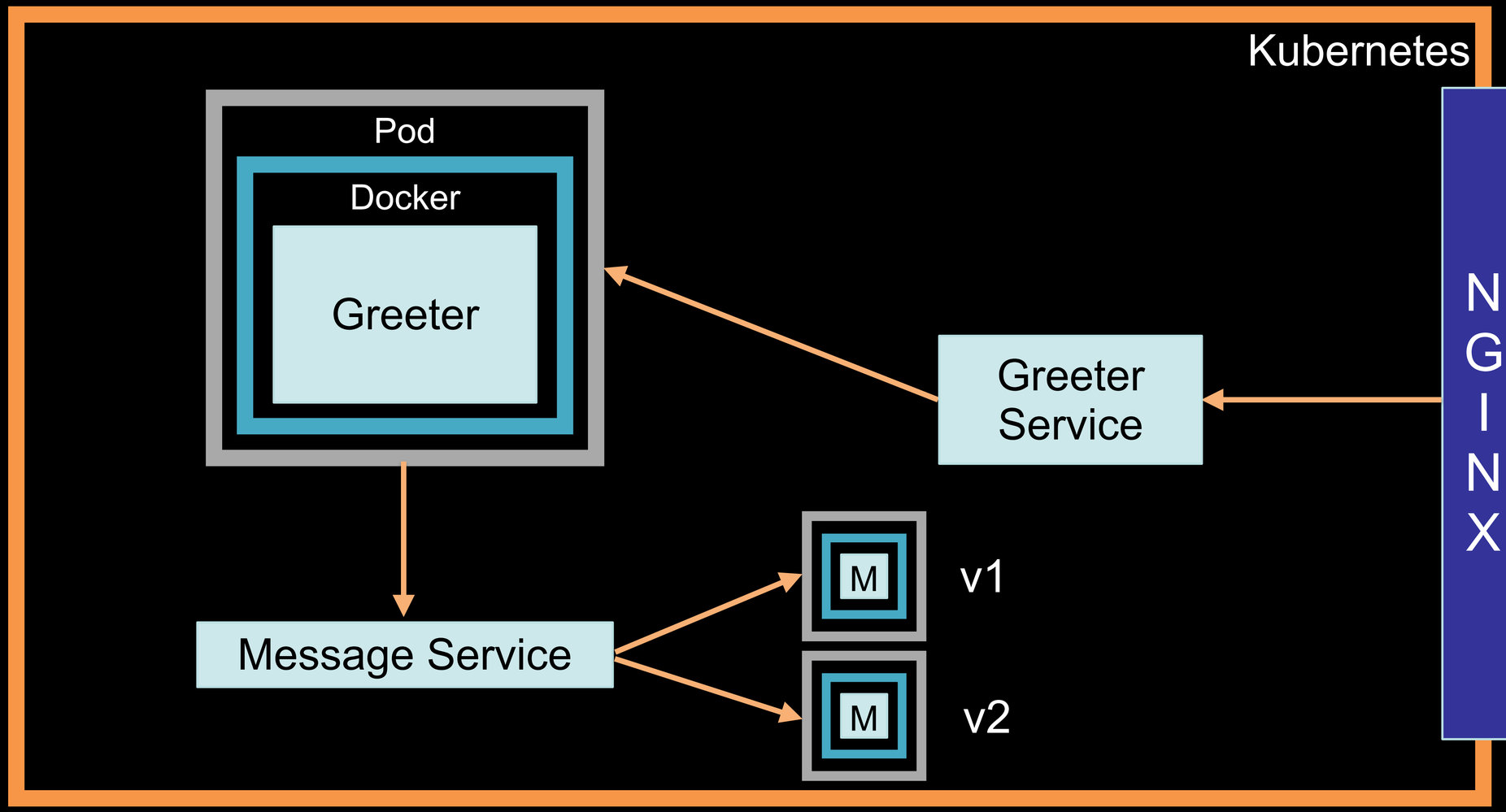
Scheduling and Running Containers

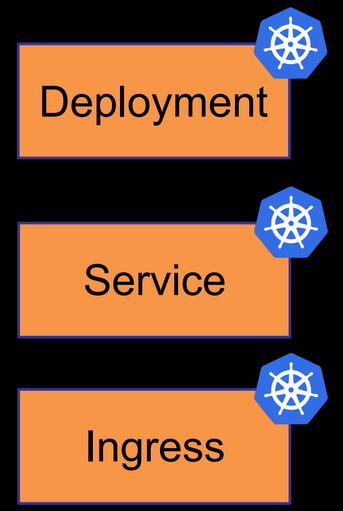


Components of Kubernetes

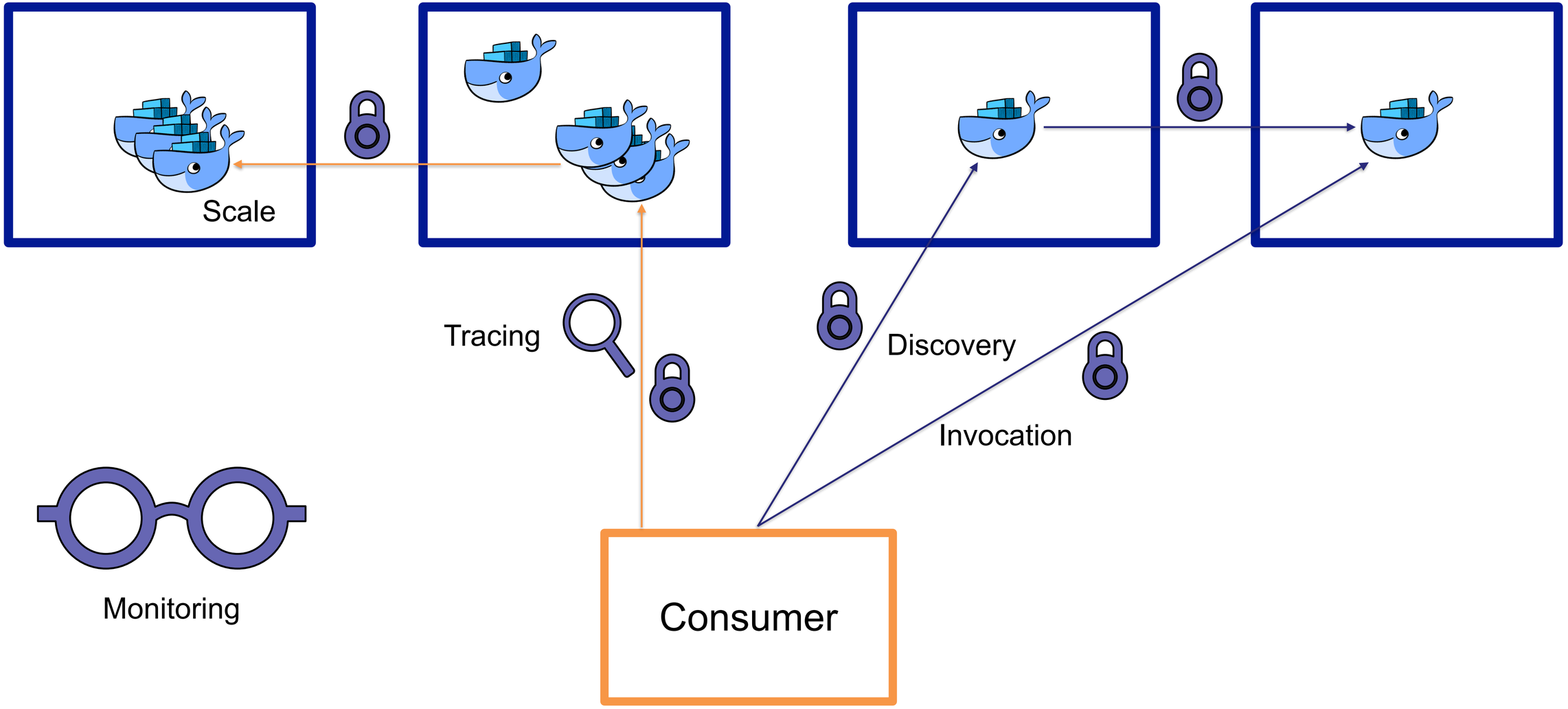


Demo Kubernetes

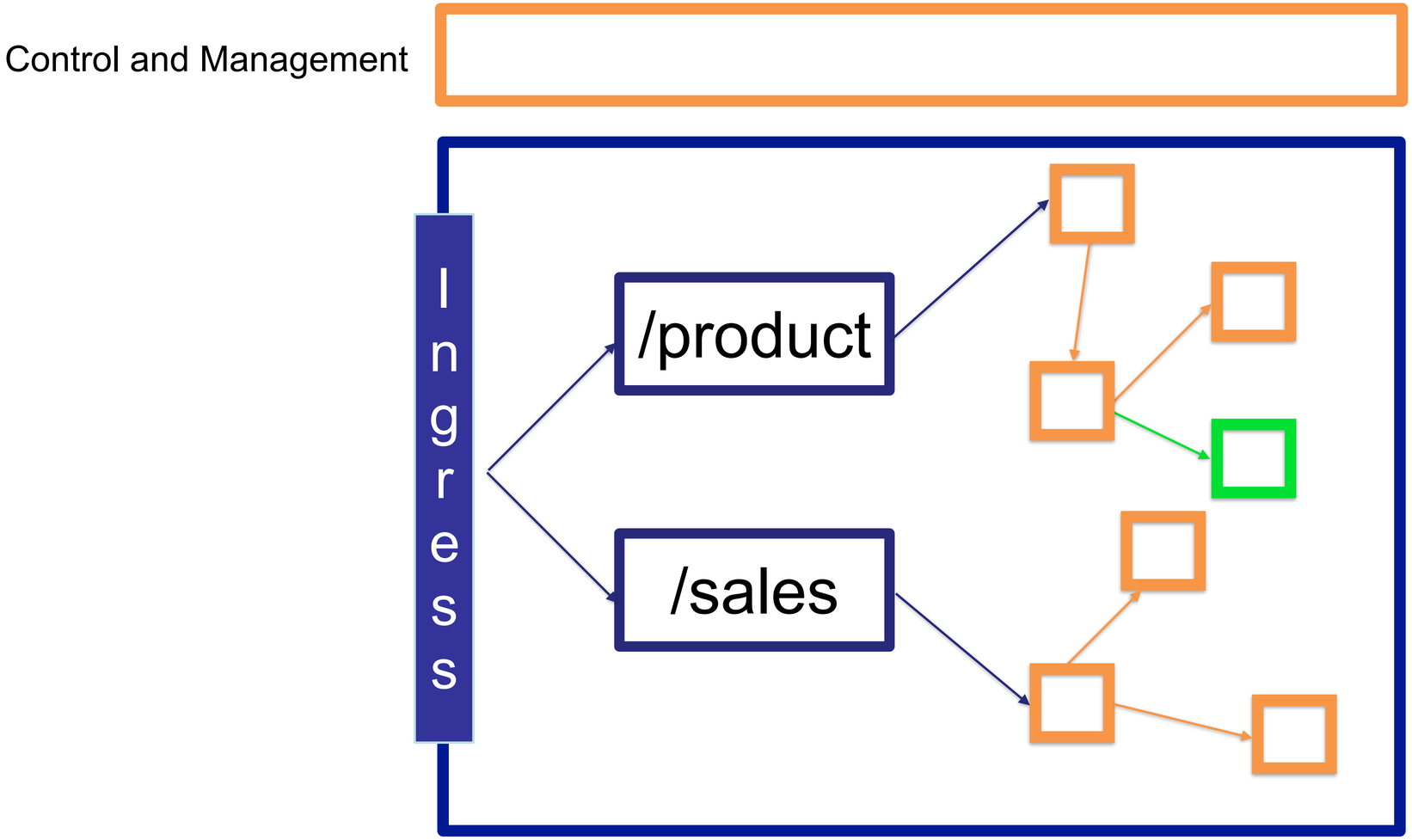




Scheduling and Running Containers Revisited



Introducing Service Mesh

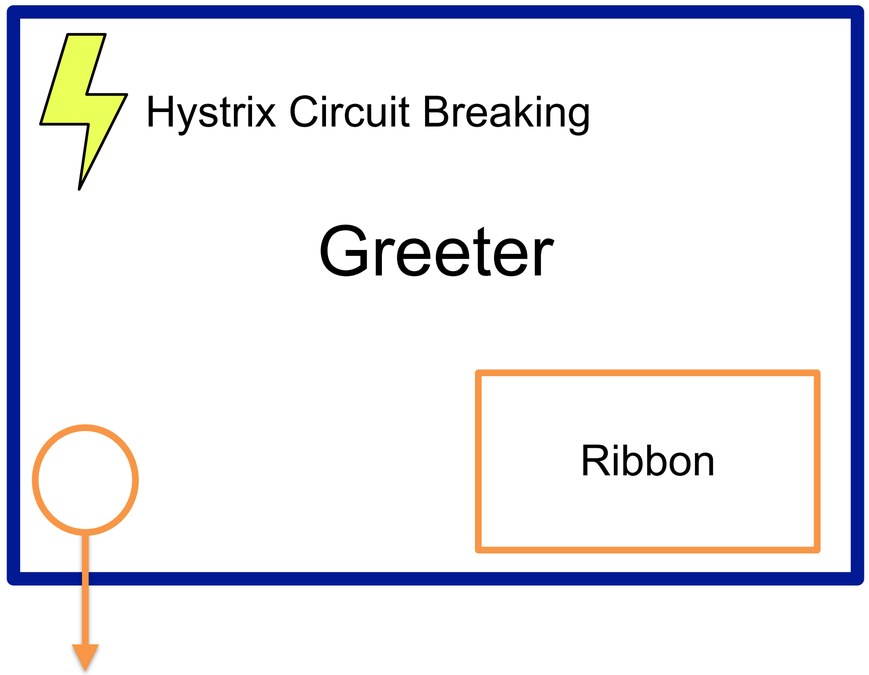


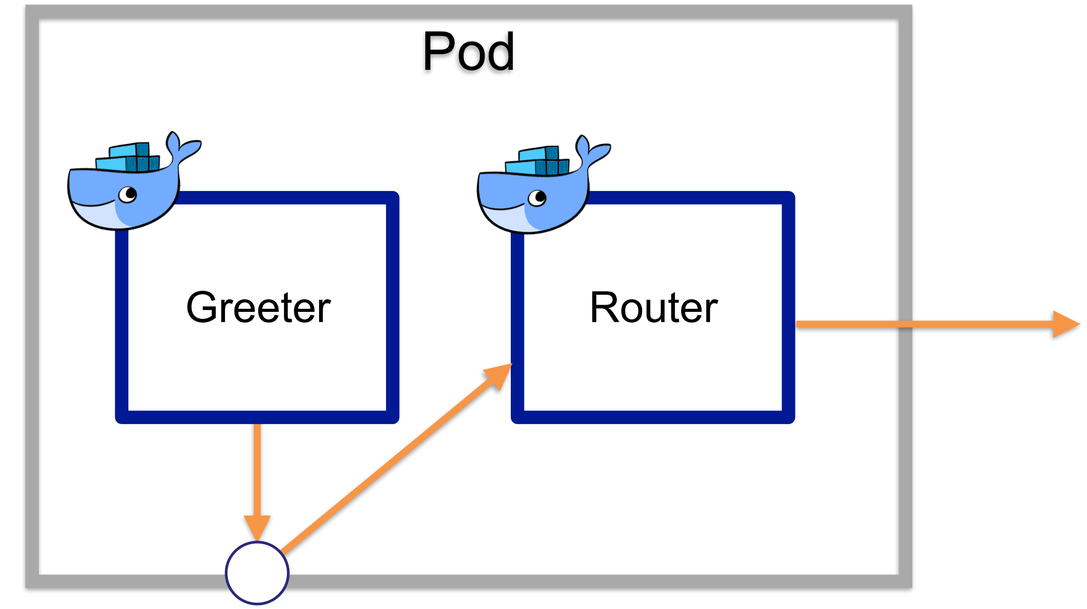
Security

Tracing

Dealing with Failures

Decoupled APIs Revisited

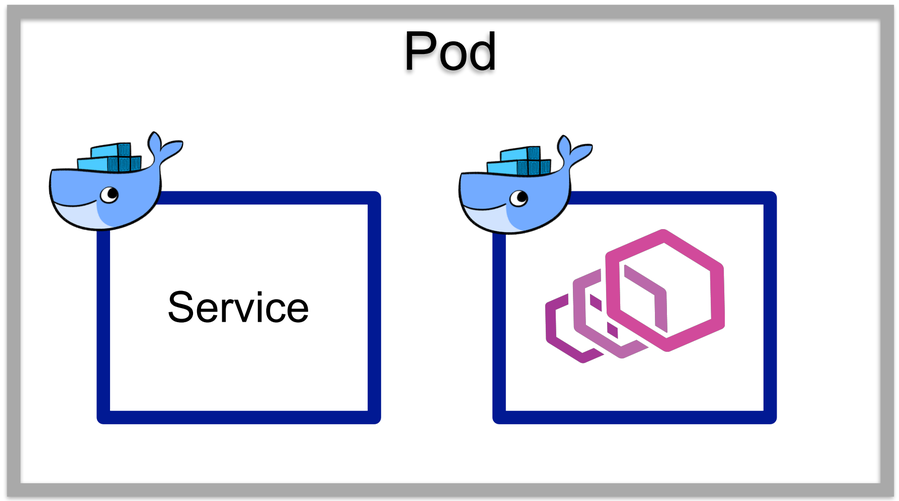




http://message-service-api

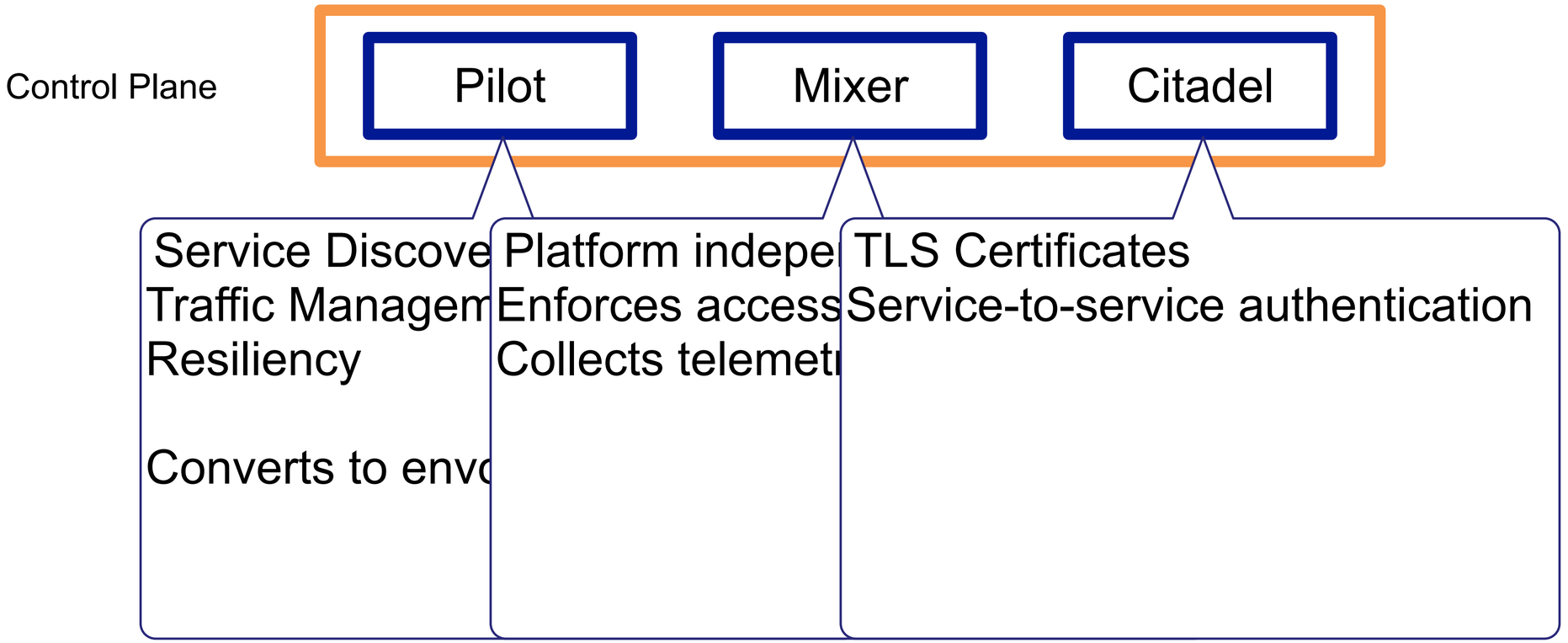
Istio - Building on the Envoy Sidecar

- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics

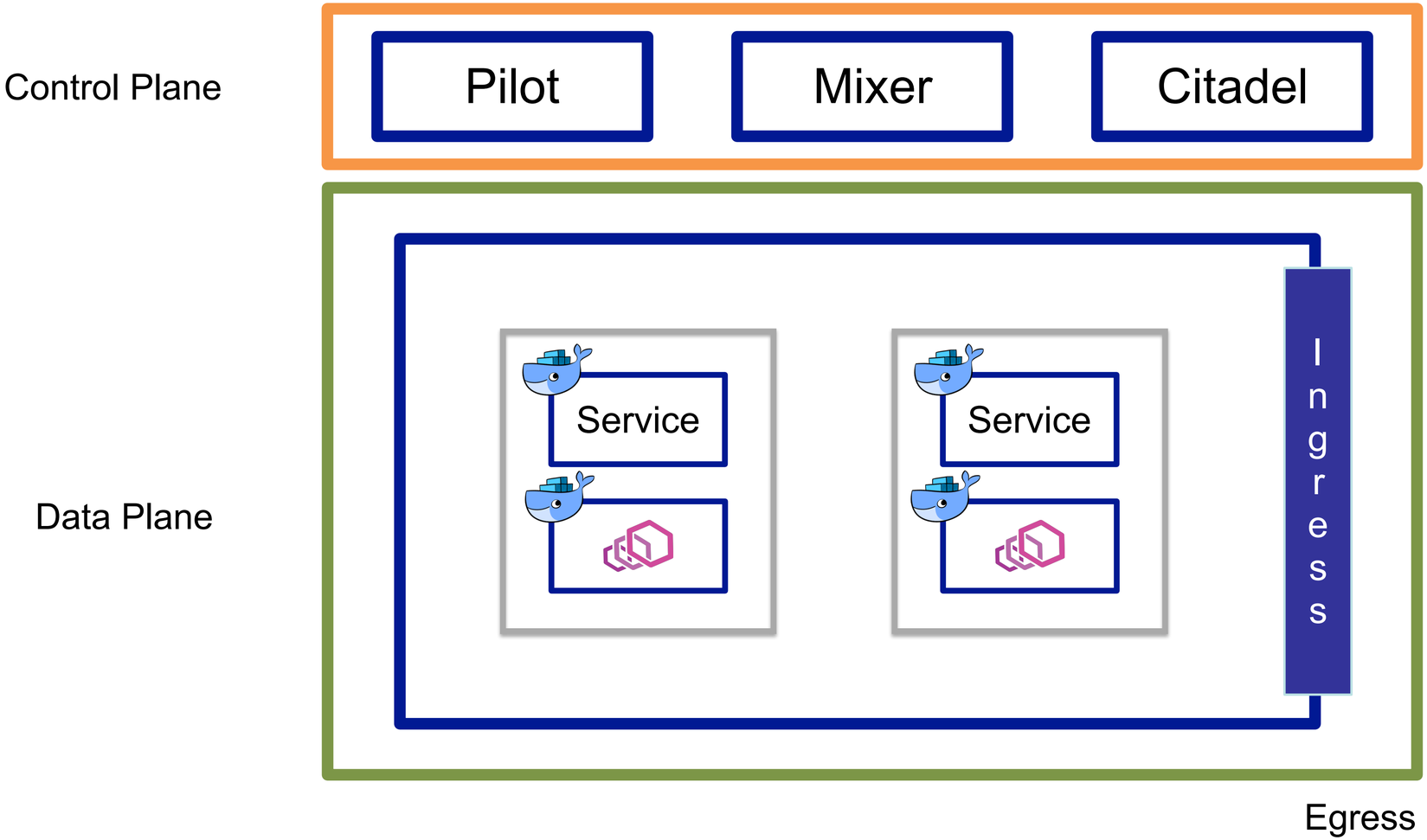


https://istio.io/docs/concepts/what-is-istio/

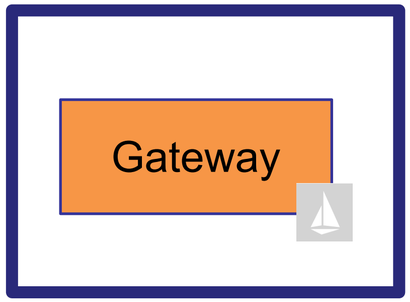
Istio Main Components



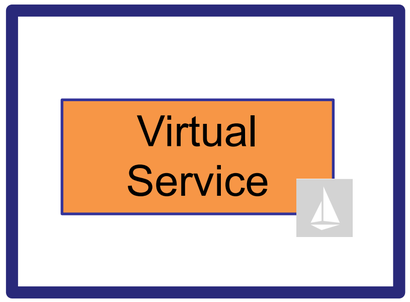
Istio Main Components



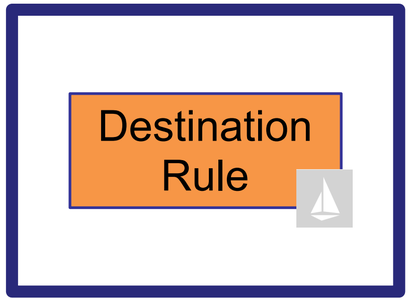
Istio Networking Configuration



Configure a load balancer for incoming HTTP Traffic
Attach against Ingress

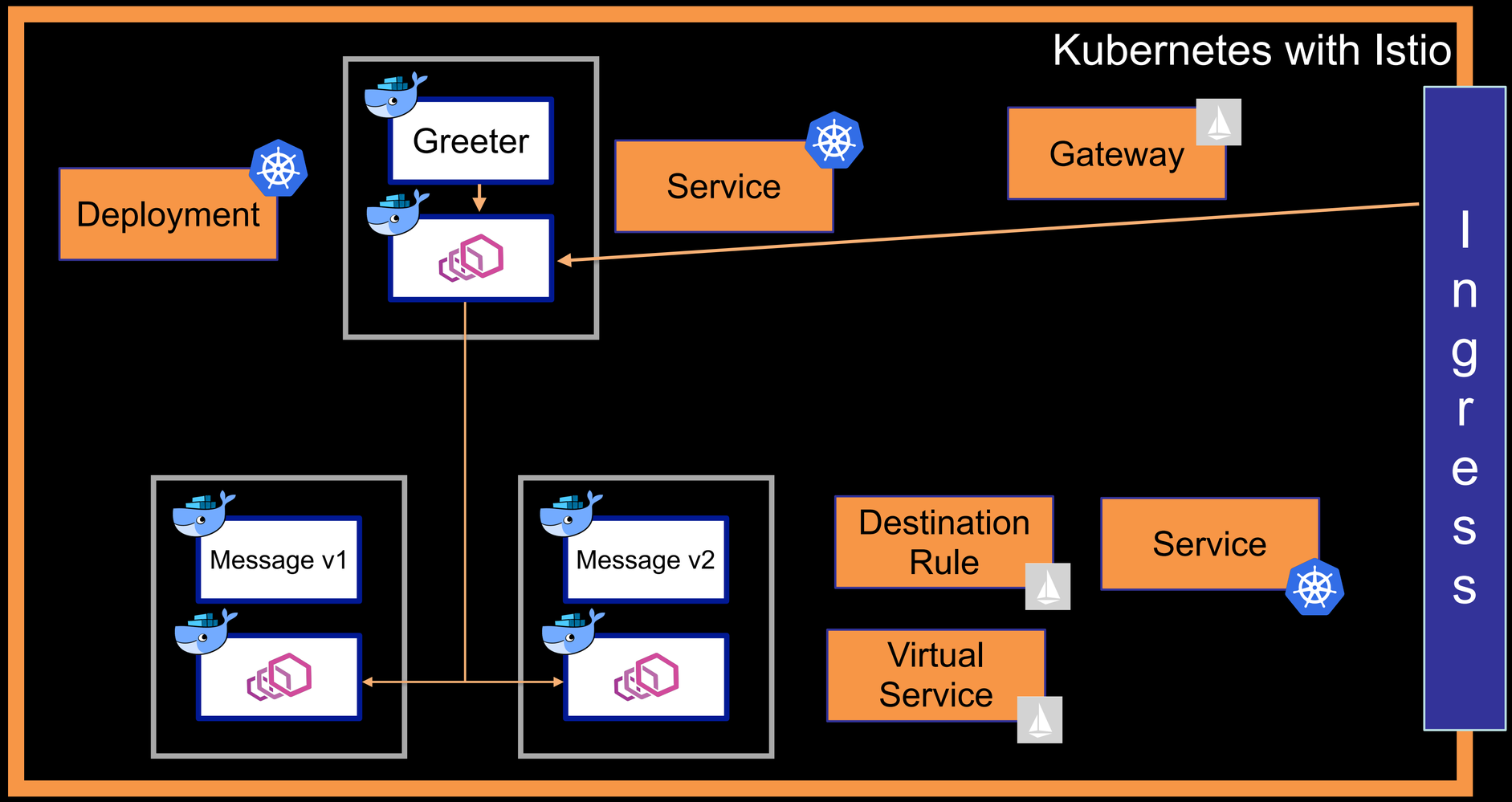


Configure how requests are routed within the Istio service mesh

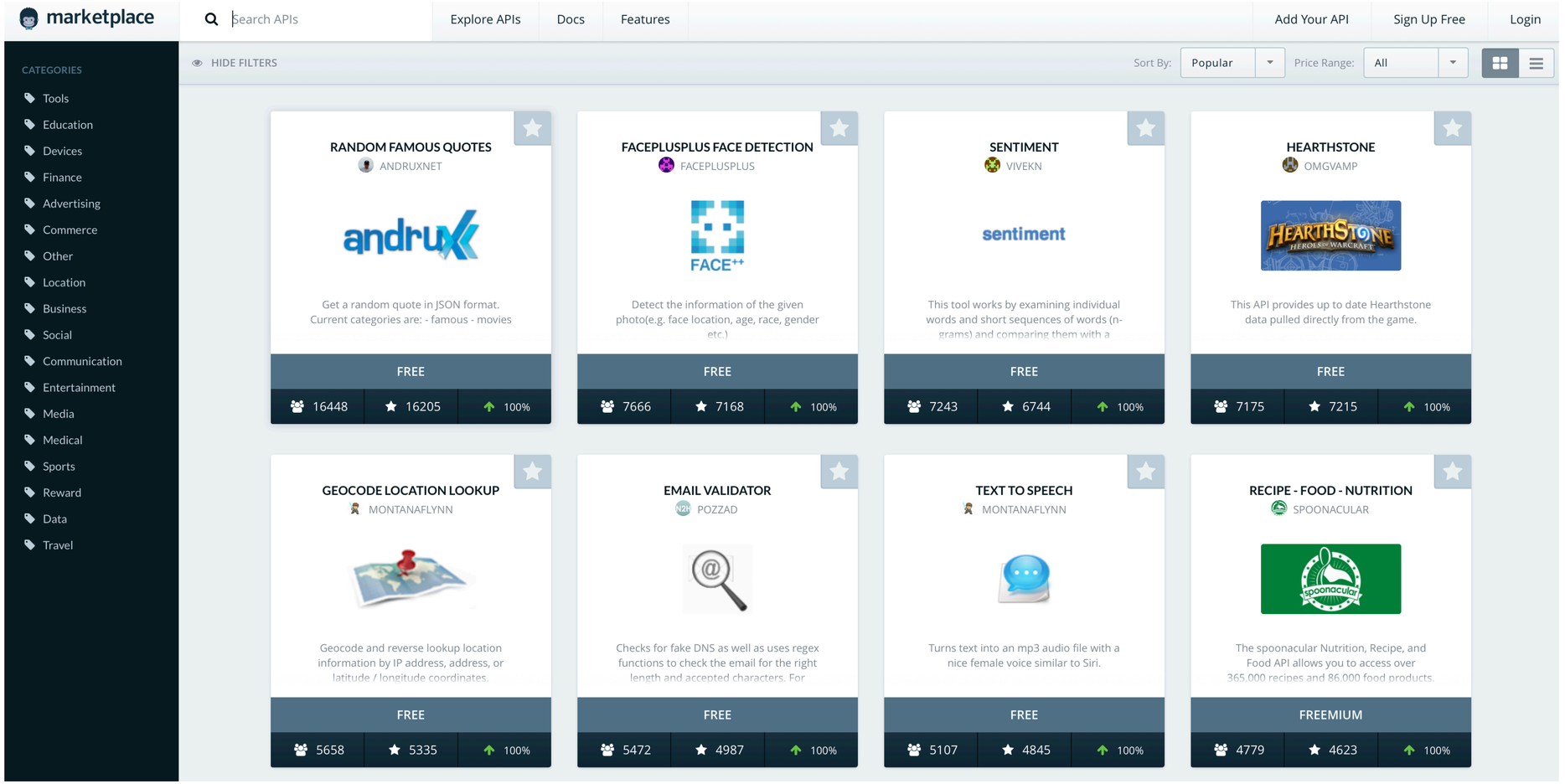


Configure traffic after VirtualService routing, e.g Least Connected

Demo Istio on Google Cloud

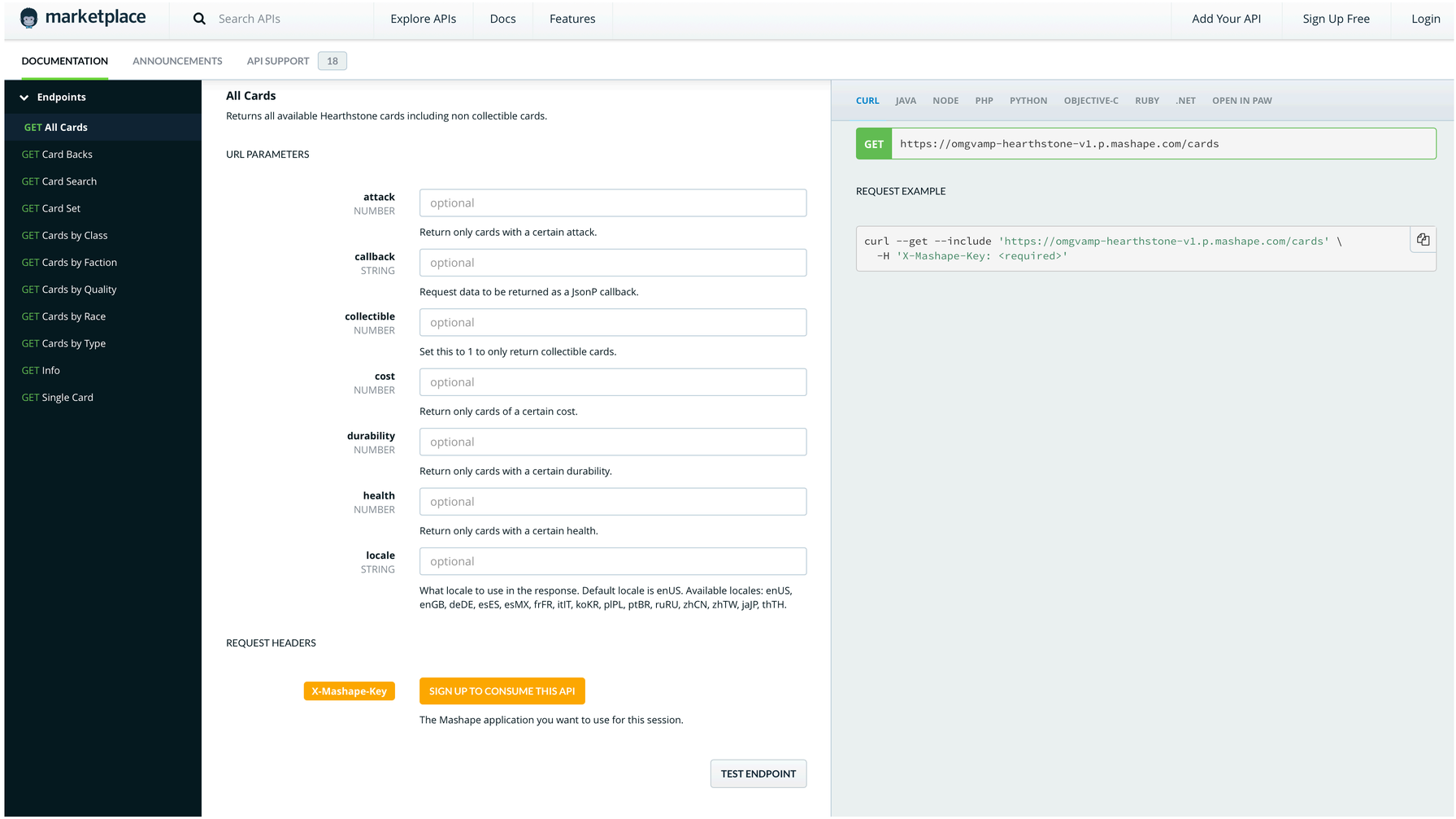


API Management - Customer View



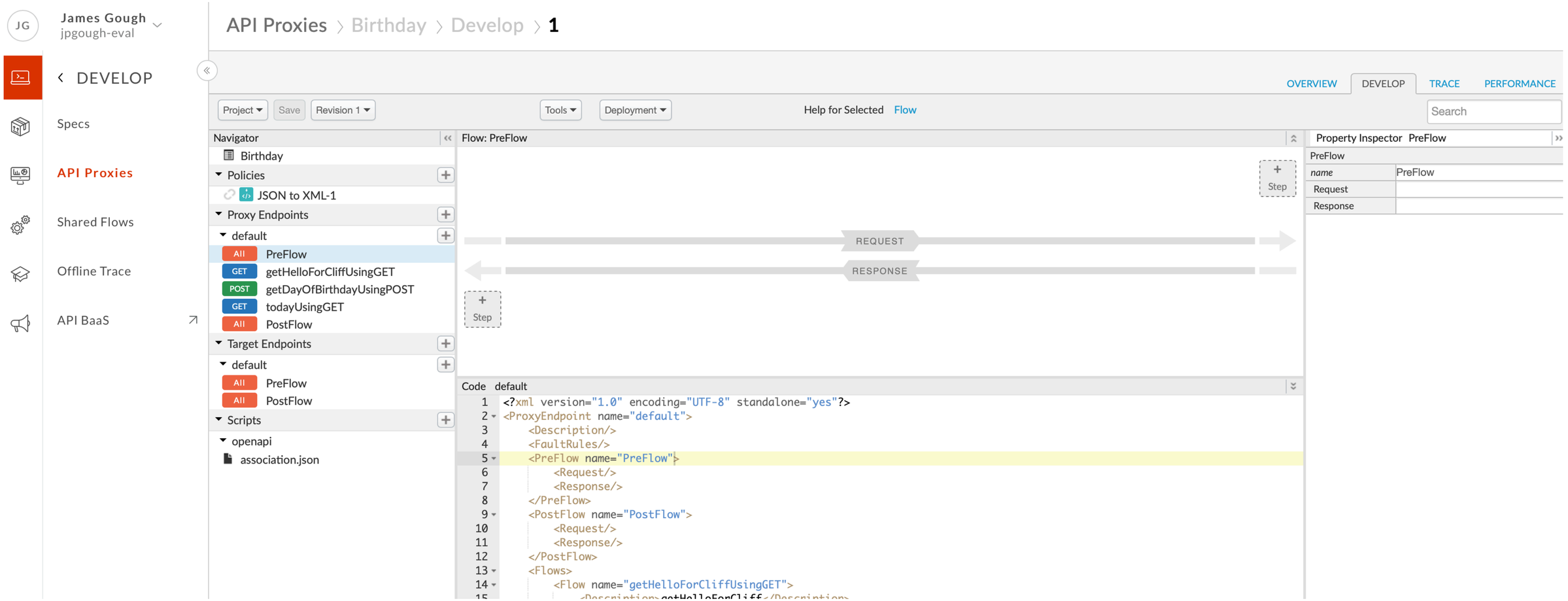
https://market.mashape.com

API Management - Customer View



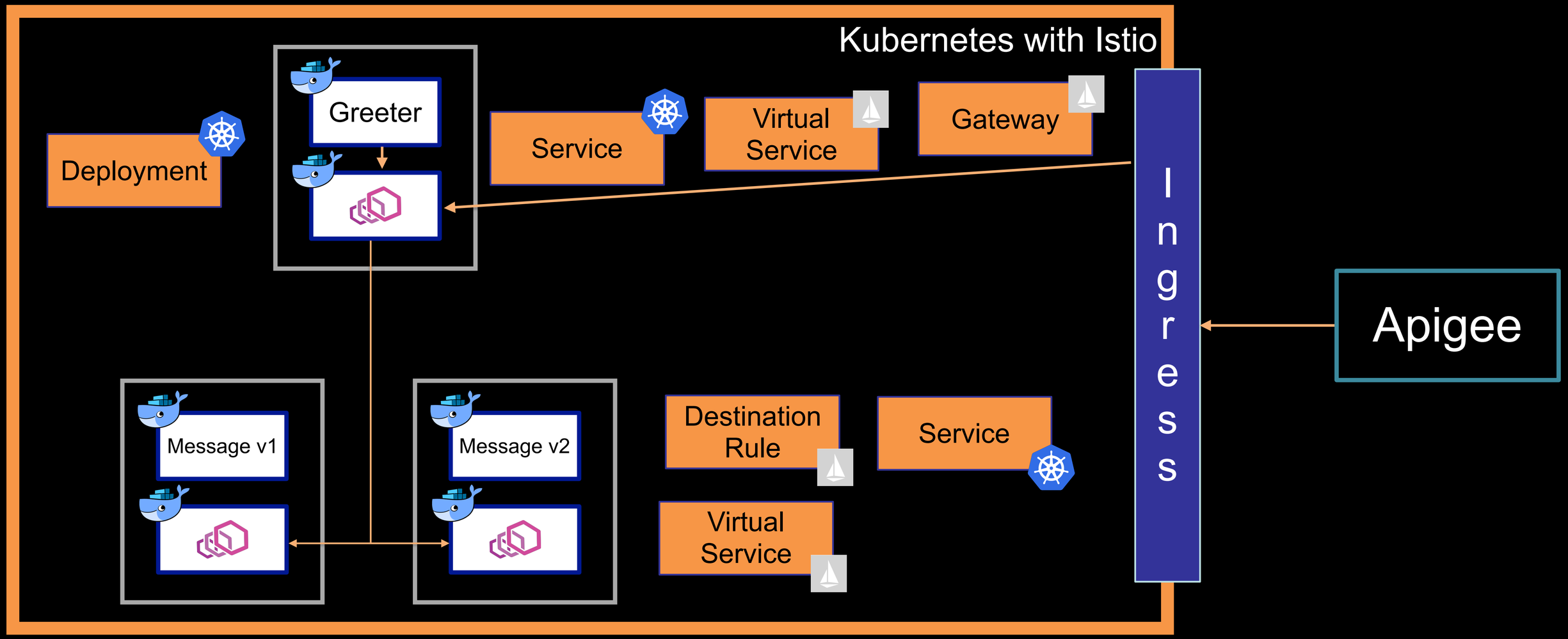
https://market.mashape.com

API Management - Development View

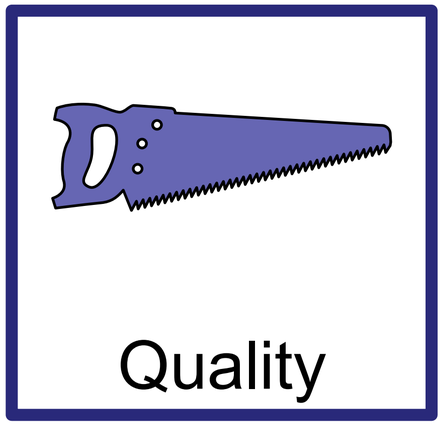


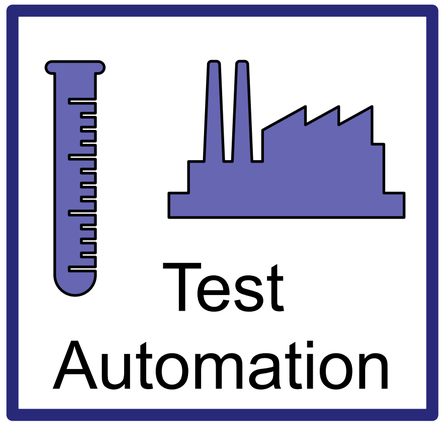
https://apigee.com/

Demo Istio with Apigee



Summary





@Jim__Gough

	Level 0 Traditional	Level 1 Basic	Level 2 Intermediate	Level 3 Advanced
Application	Monolithic	Service Oriented Integrations	Service Oriented Applications	API Centric
Database	One Size Fits All Enterprise DB	Enterprise DBs + No SQL	Polyglot, DBaaS	Matured Data Lake
Infrastructure	Physical Machines	Virtualization	Cloud	Containers
Monitoring	Infrastructure	App and Infra Monitoring	APMs	APM and Central Log Management
Process	Waterfall	Agile and CI	CI & CD	DevOps