Microservices

An introduction

Hany Ahmed
www.namozag.com

Introduction

Microservices Architecture

An architectural style

that structures an application as

a collection of loosely coupled,
collaborating services,

which implement business capabilities

Who is using it?

NETFLIX

1000+

Microservices

Architecture Evolution

1990

Monolithic

2000

SOA

2010

Microservices

Exist in a “dumb” messaging environment!

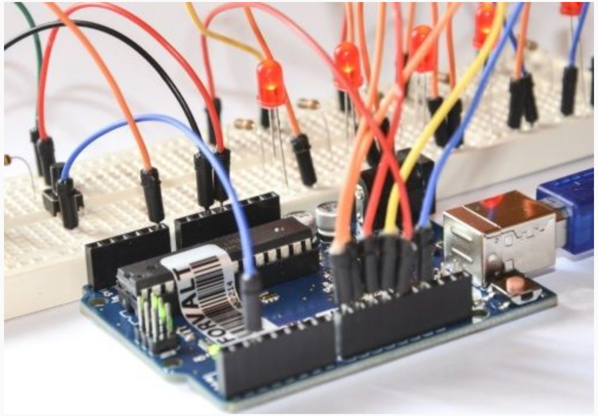
SOA vs Microservices

Dump services

Smart pipes

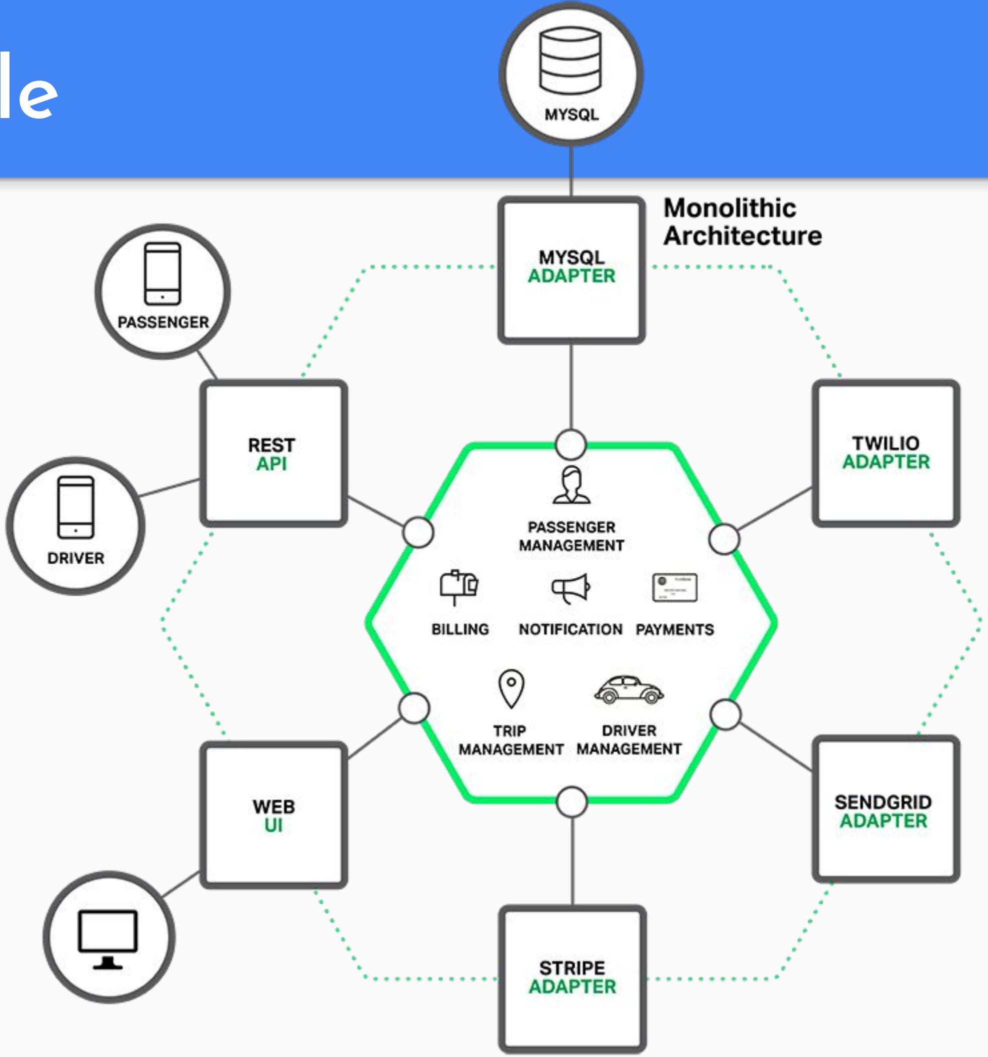
Smart services

Dump pipes

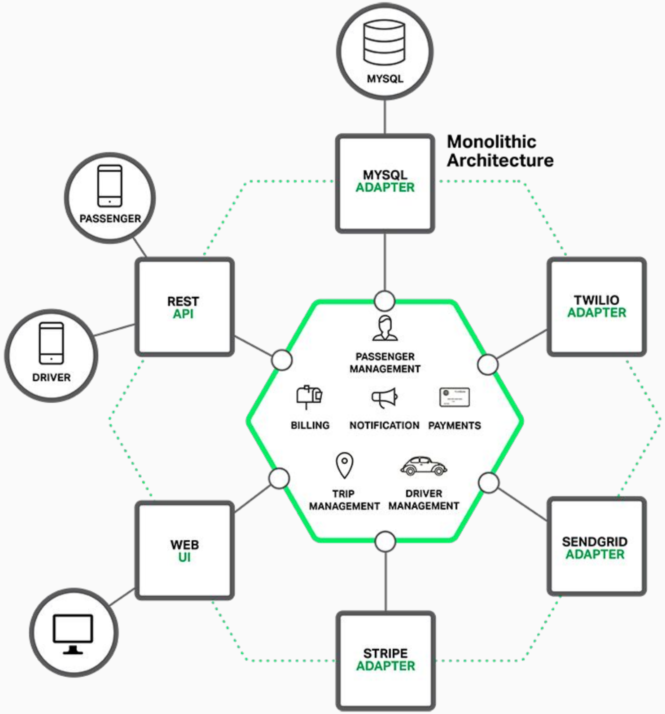


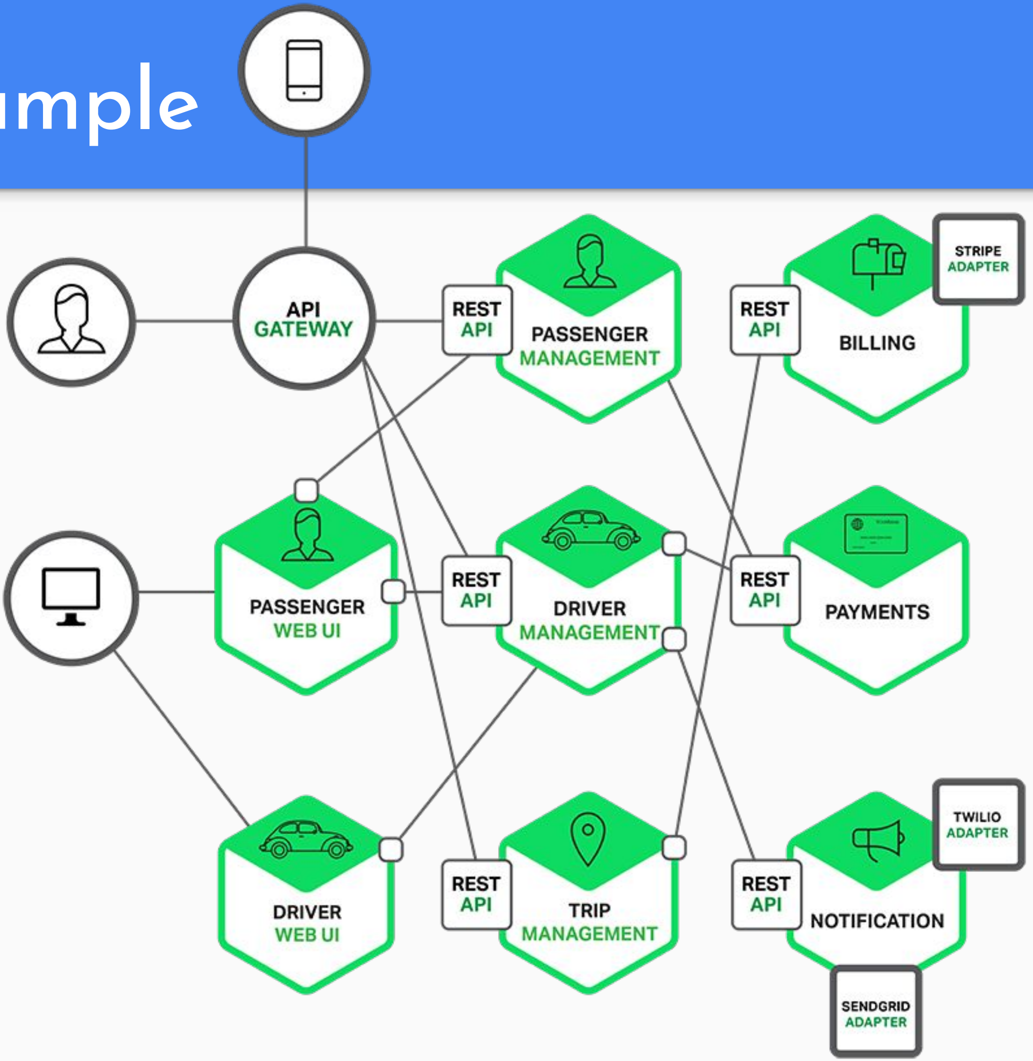
```
ps aux | grep java
```

Monolithic Example



Microservices Example



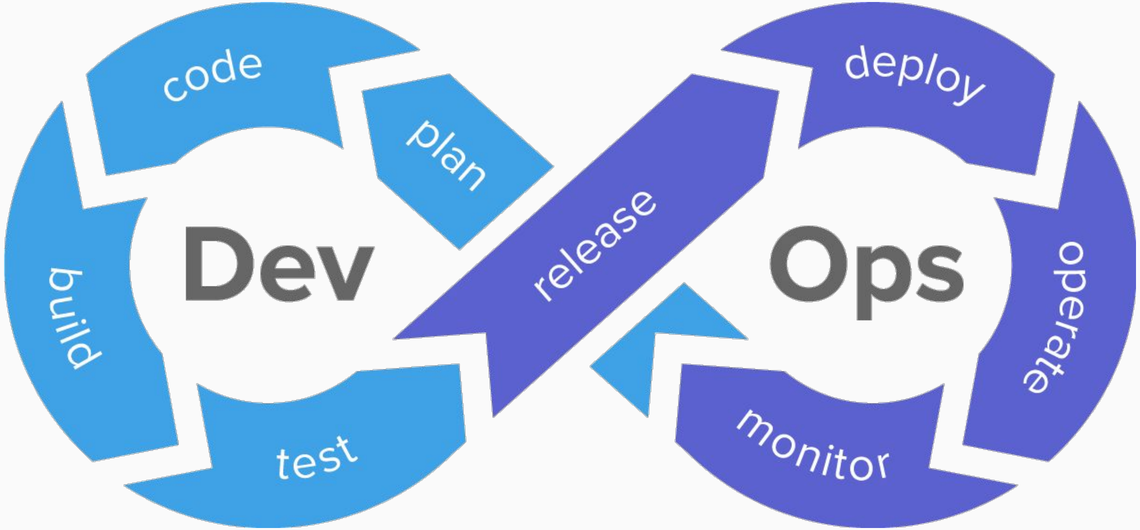


DevOps

practices

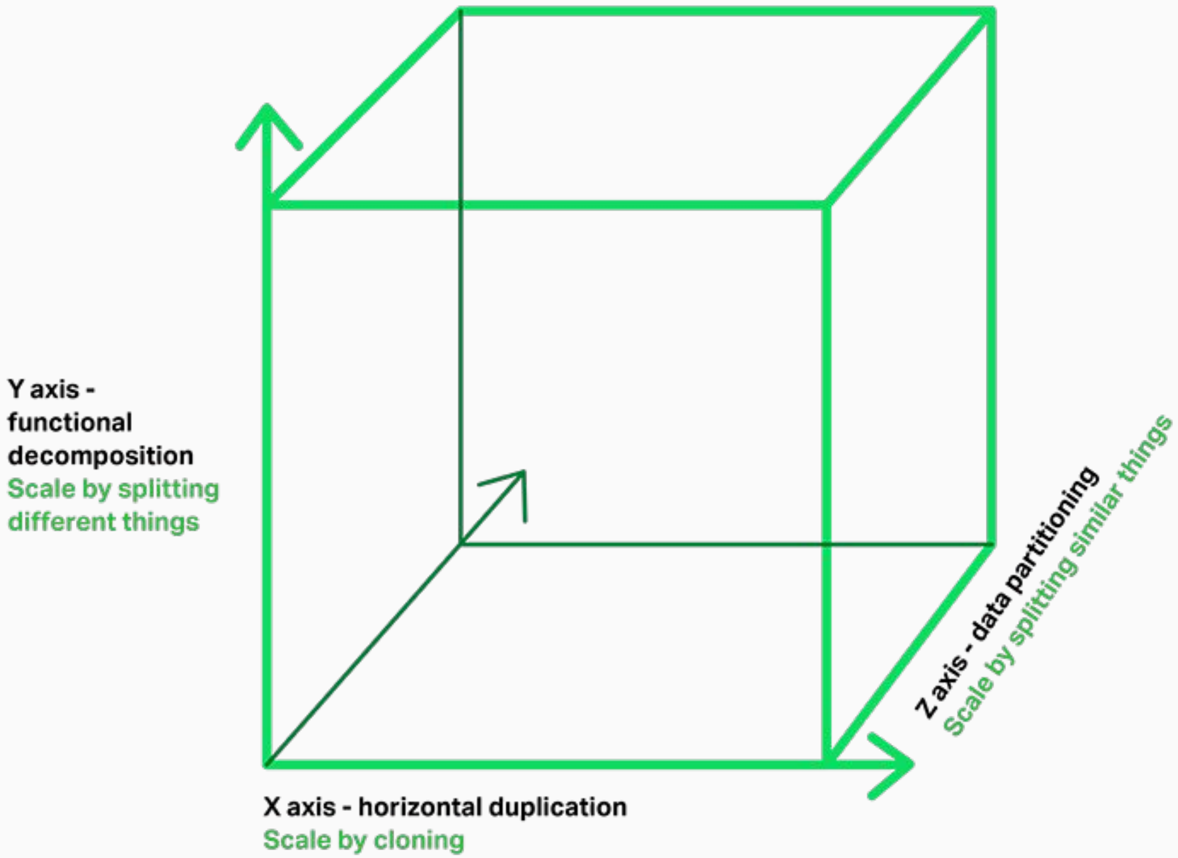
processes

tools



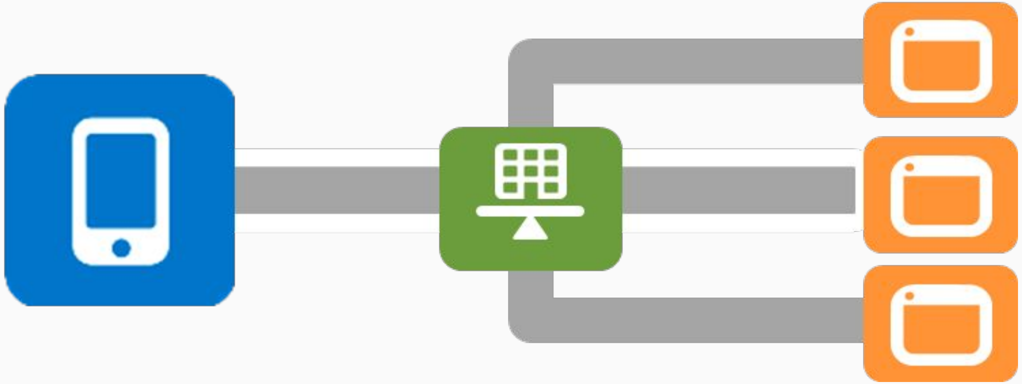
Scale cube

3D model of scalability


Y axis - functional decomposition
Scale by splitting different things
X axis - horizontal duplication
Scale by cloning
Z axis - data partitioning
Scale by splitting similar things

X-axis Horizontal duplication

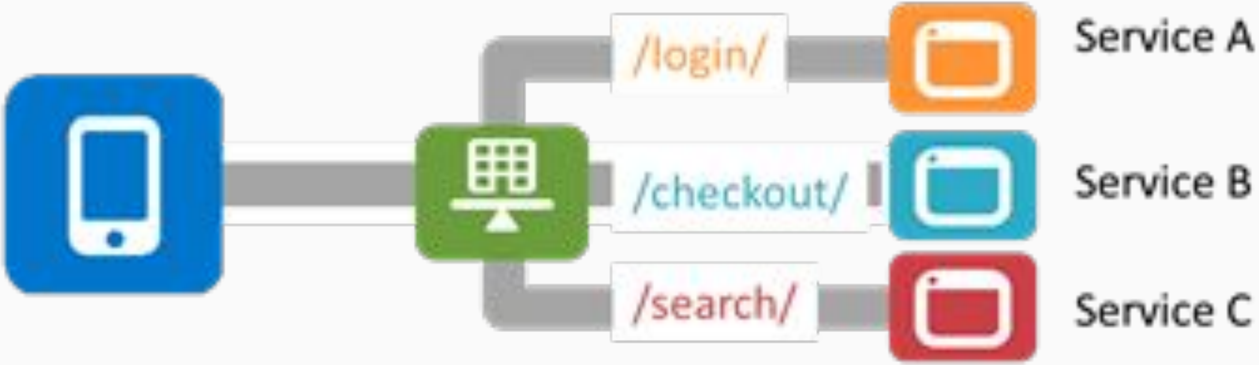
Running multiple copies of an application behind a load balancer



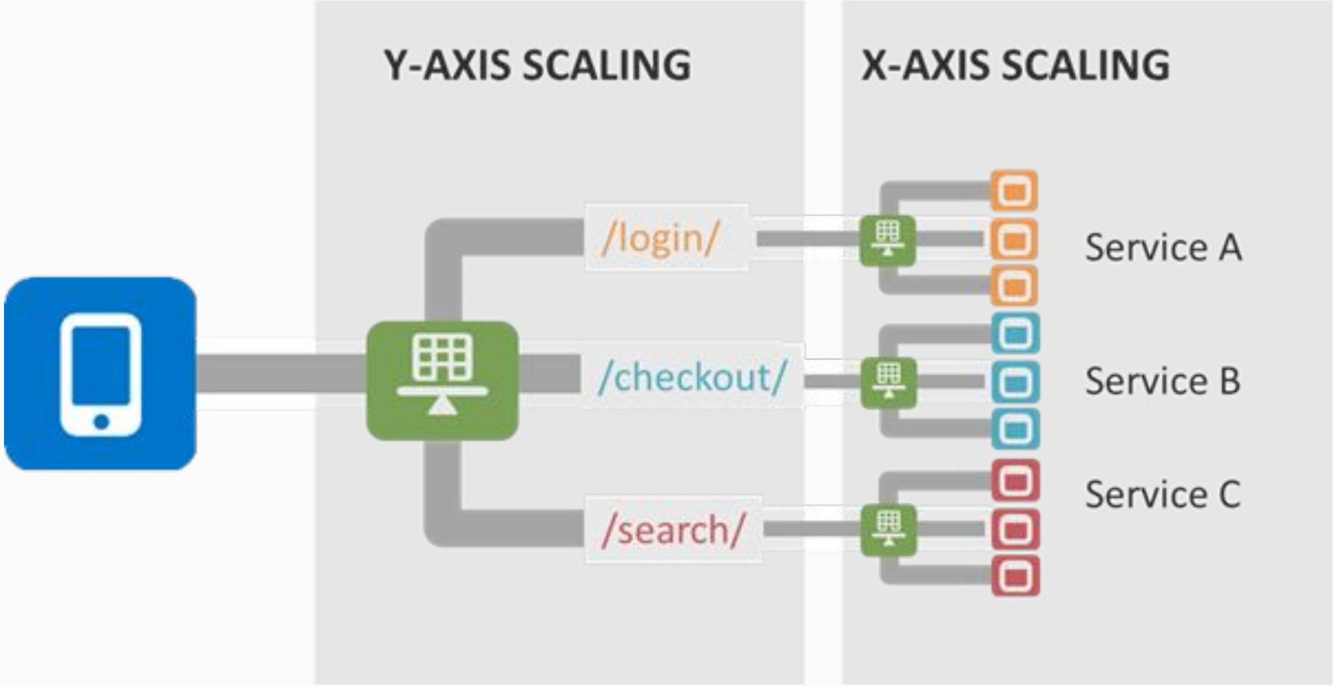
Y-axis Functional decomposition

Splits the application into multiple, different services

Each service is responsible for one or more closely related functions

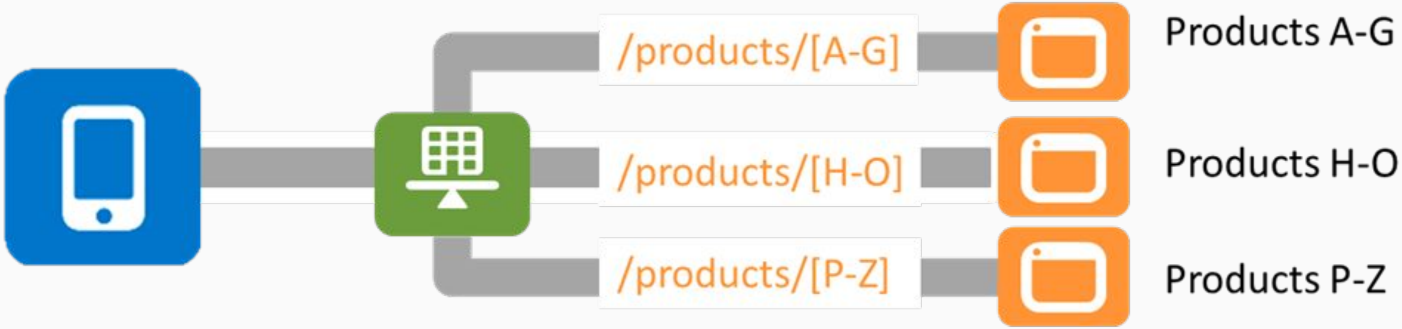


X+Y


Y-AXIS SCALING
X-AXIS SCALING
/login/
/checkout/
/search/
Service A
Service B
Service C

Z-axis Data partitioning

Each server runs an identical copy of the code but responsible for only a subset of the data



Statelessness

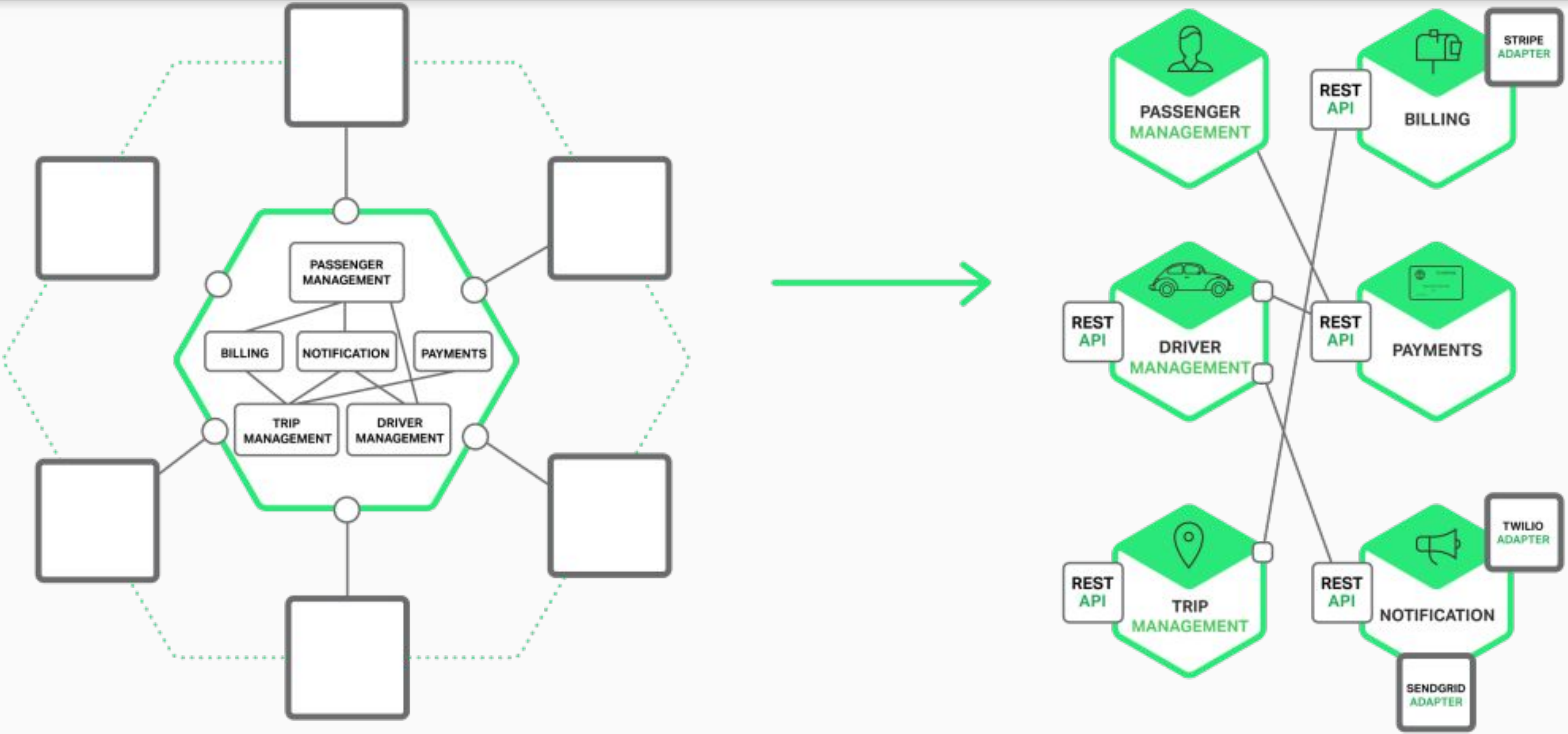
Defer state information to:

- Database
- Service consumer

Separating services from their data → Scalability++

Minimize In-memory data → Availability++

What do you think?



Microservices

Motivation

- New team members must quickly become productive
- The application must be easy to understand and modify
- Practice continuous deployment
- Run multiple copies of the application on multiple machines in order to satisfy scalability and availability
- Take advantage of emerging technologies (frameworks, programming languages, etc)

Microservices characteristics

- Loosely coupled
- Communicate with a lightweight protocol:
  - Synchronous protocols such as HTTP/REST
  - Asynchronous protocols such as AMQP
- Autonomous
- SRP

Monolithic issues

Hard to understand

Hard to maintain

Long Time to Ship

Hard to scale

Failure Cascade

Hard to monitor

Hard to figure out issues

Stuck in a Technology/Language

Microservices benefits

- Developed
- Deployed
- Run
- Scaled

Independently

& Quickly

By a small engineering team

With any technology

Resilient

More maintainable

Drawbacks

Needs communication mechanism implementation

Distributed transactions complexity

Increased memory consumption

Deployment, Management complexity

Decomposition

Identifying Service Boundaries

- Loosely coupled
- Highly cohesive
- Autonomous

Size

Small

enough that it serves a focused purpose.

Big

enough that it minimizes interservice communication

Decomposition

Verb-based

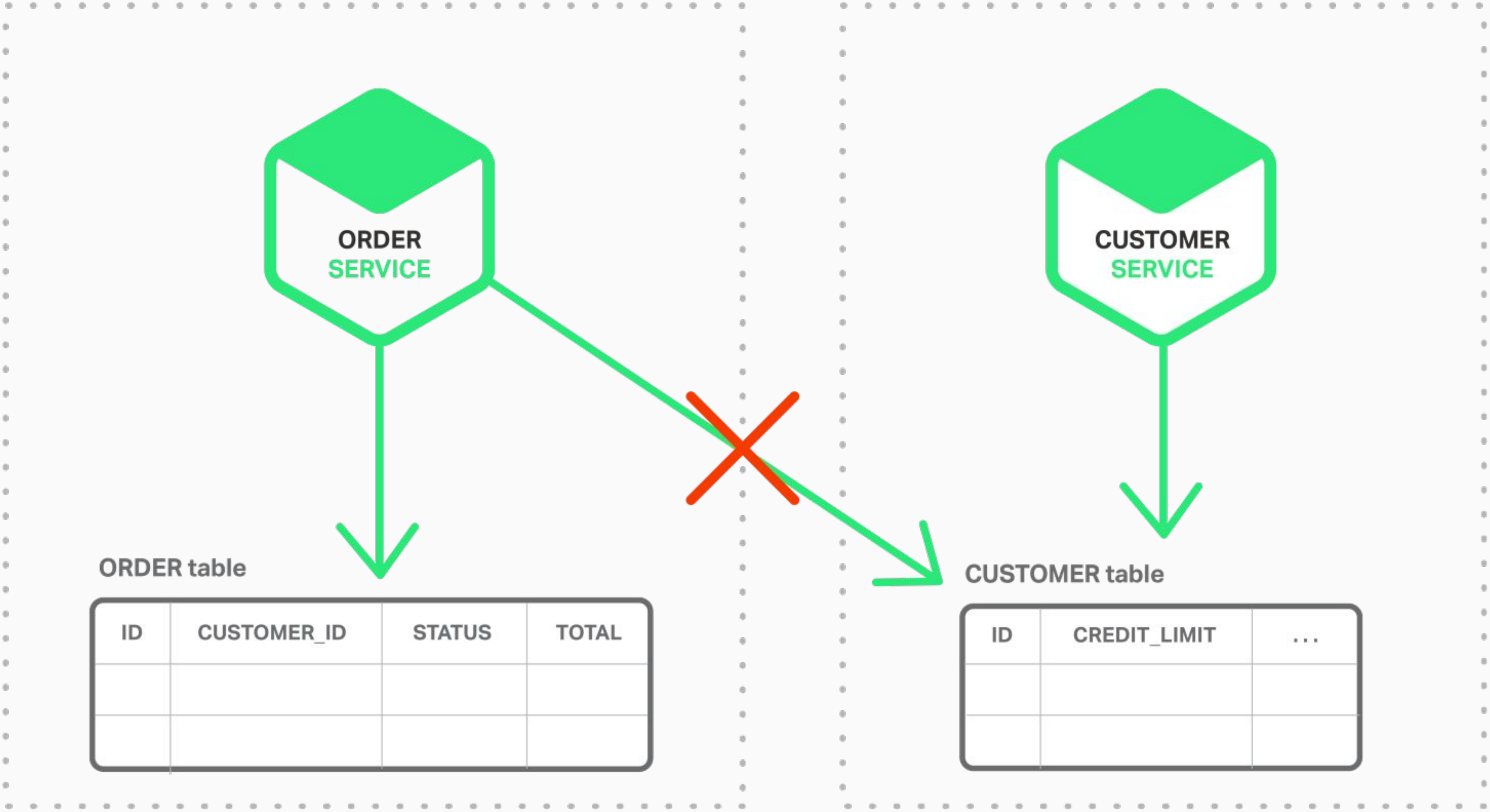
Use case

Business capability

Noun-based

Entity

DDD



ORDER SERVICE

CUSTOMER SERVICE

ORDER table

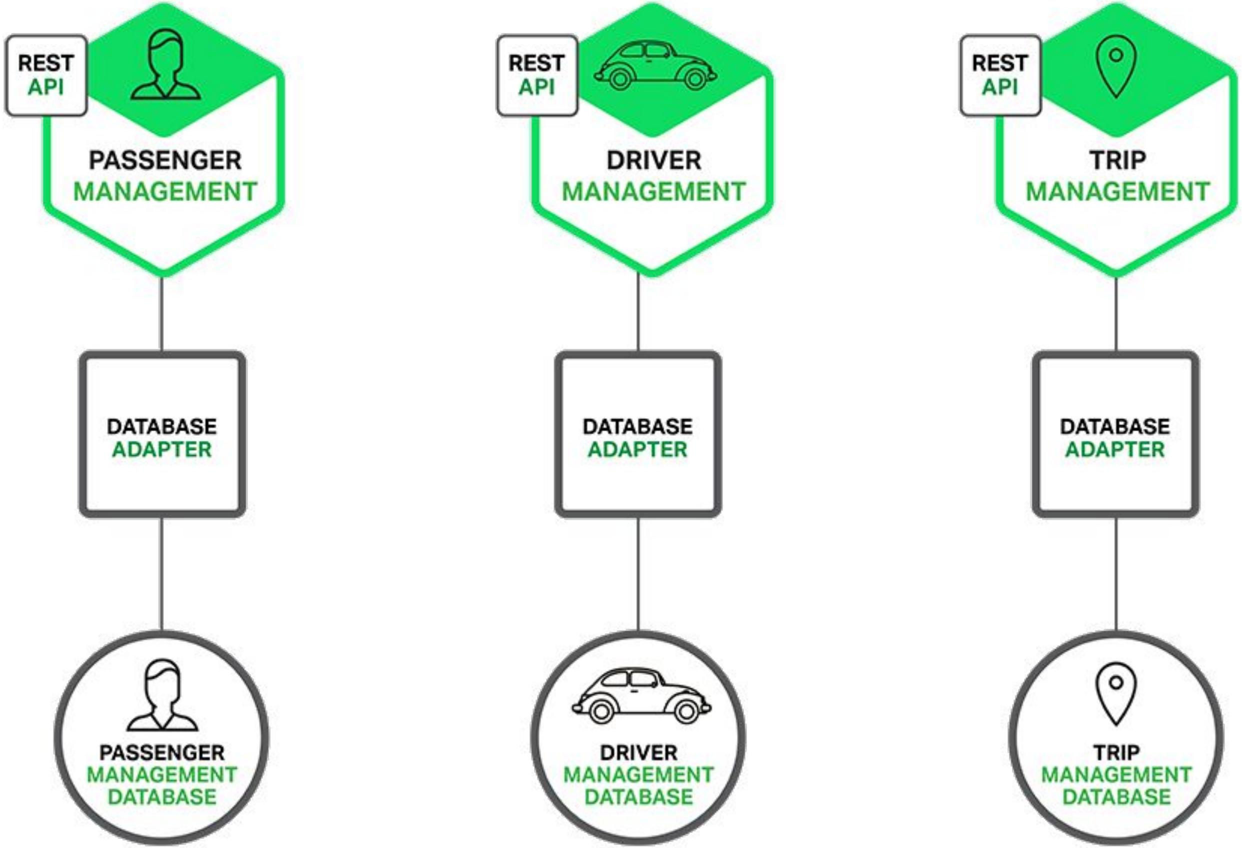
ID	CUSTOMER_ID	STATUS	TOTAL

CUSTOMER table

ID	CREDIT_LIMIT	...

Pattern: Database per service ..

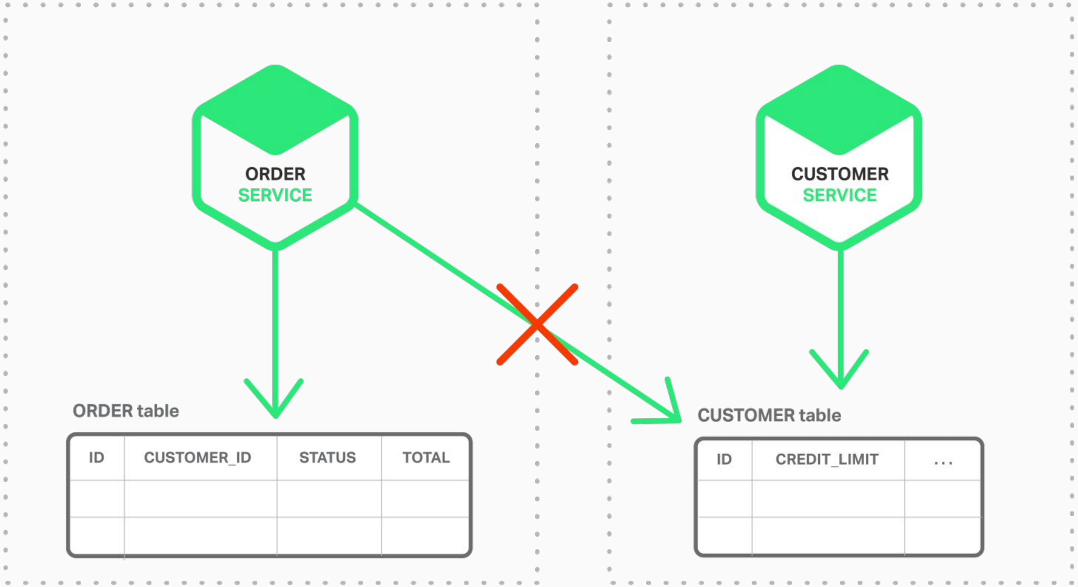
Database to a service is like Private variable to a class



Distributed Transactions Challenges

ACID

- Atomicity
- Consistency
- Isolation
- Durability



ID	CUSTOMER_ID	STATUS	TOTAL

ID	CREDIT_LIMIT	...

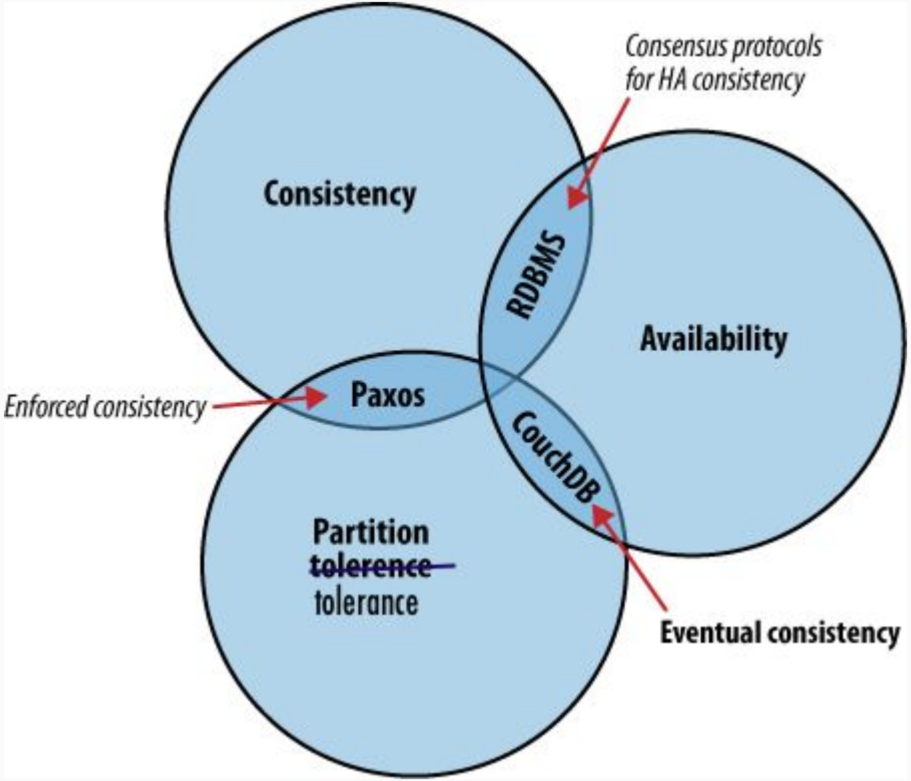
CAP theorem

It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

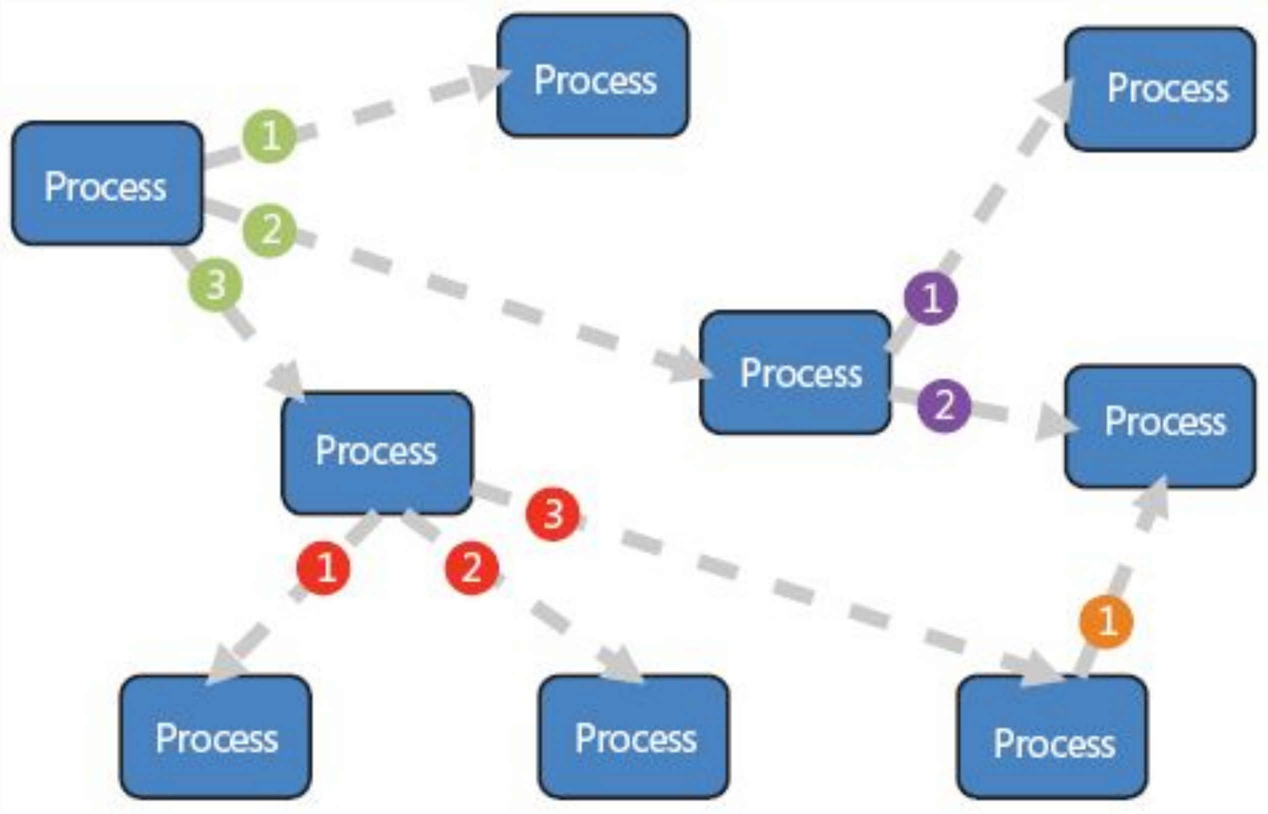
Consistency [2PC]

Availability

Partition tolerance



Event-Driven Architecture

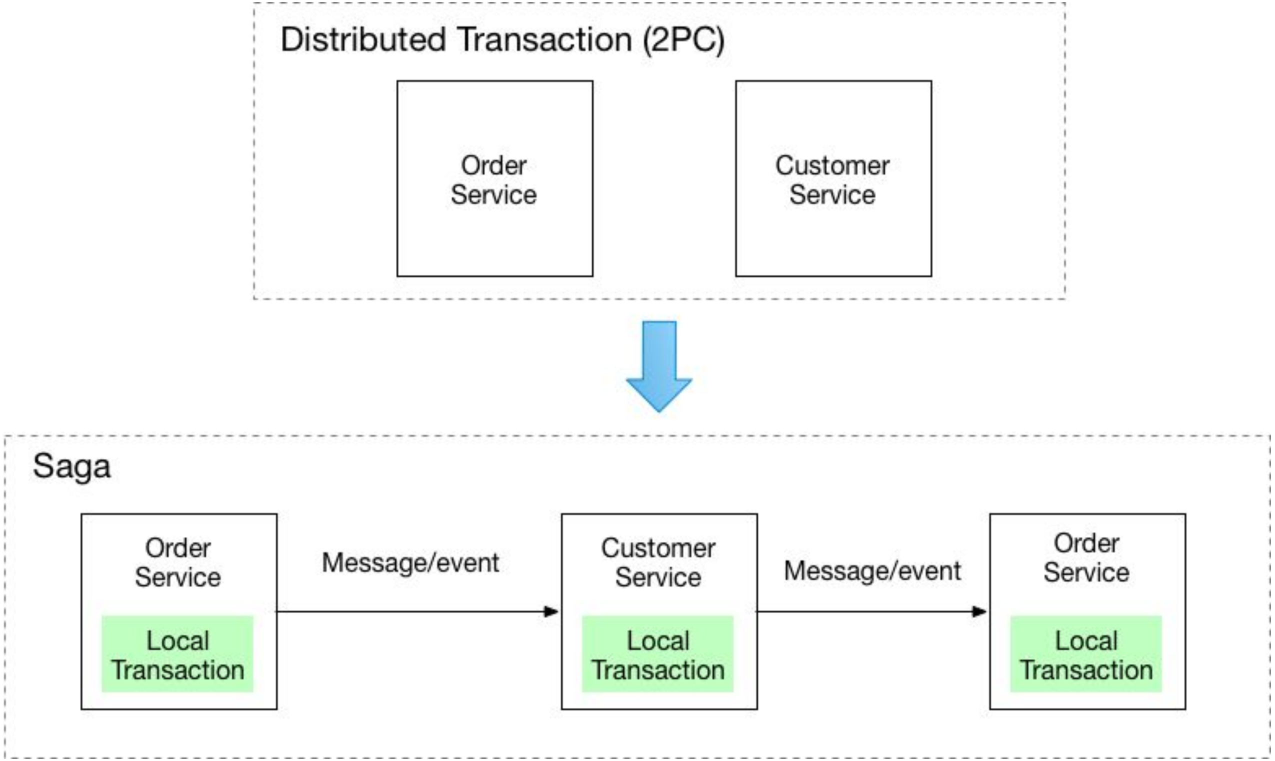


Pattern: Saga (=Story)

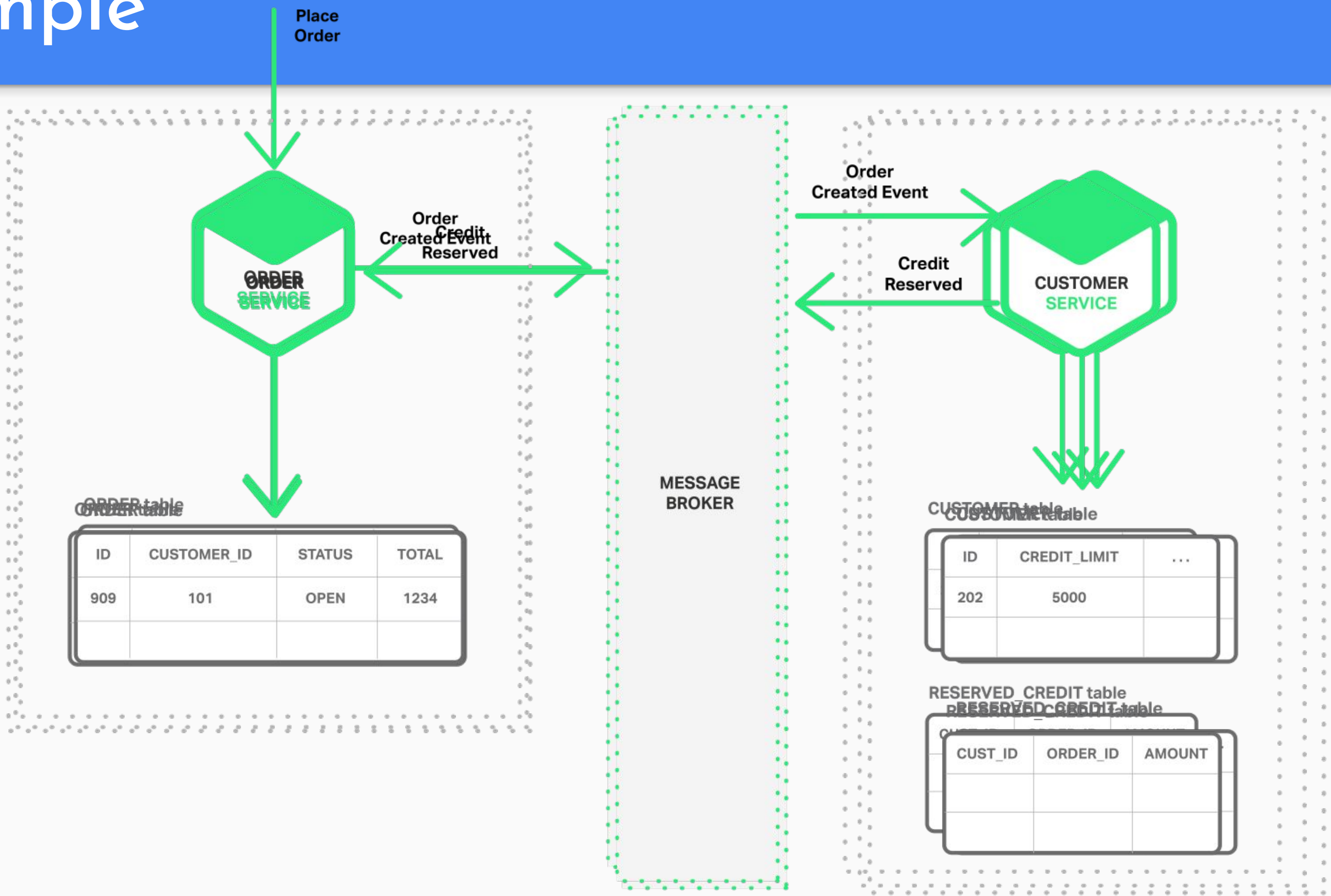
A sequence of local transactions

Each local transaction updates the database and publishes a message or event to trigger the next local transaction

If a local transaction fails then the saga executes a series of compensating transactions that undo the changes



Example



ORDER table

ID	CUSTOMER_ID	STATUS	TOTAL
909	101	OPEN	1234

CUSTOMER table

ID	CREDIT_LIMIT	...
202	5000	

RESERVED_CREDIT table

CUST_ID	ORDER_ID	AMOUNT

Eventual Consistency

Consistency is hard in a distributed system

Services may have a divergent view of the data at any point in time

They'll eventually converge to having a consistent view

Communication

Interprocess Communication (IPC)

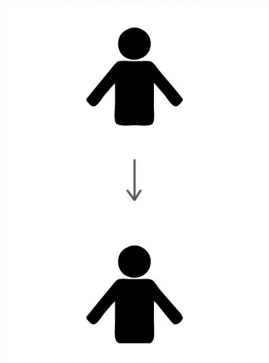
- **Monolithic** → services invoke one another through language-level method
- **Traditional distributed system** → services run at fixed, well known locations (hosts and ports)
- **Microservices** → runs in a virtualized or containerized environments where the number of instances of a service and their locations changes dynamically

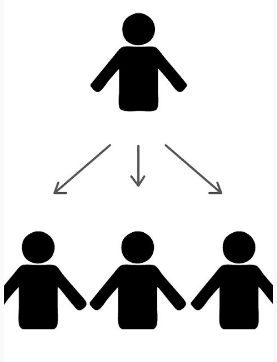
Communication types

Request/response

Request/async response

Notification

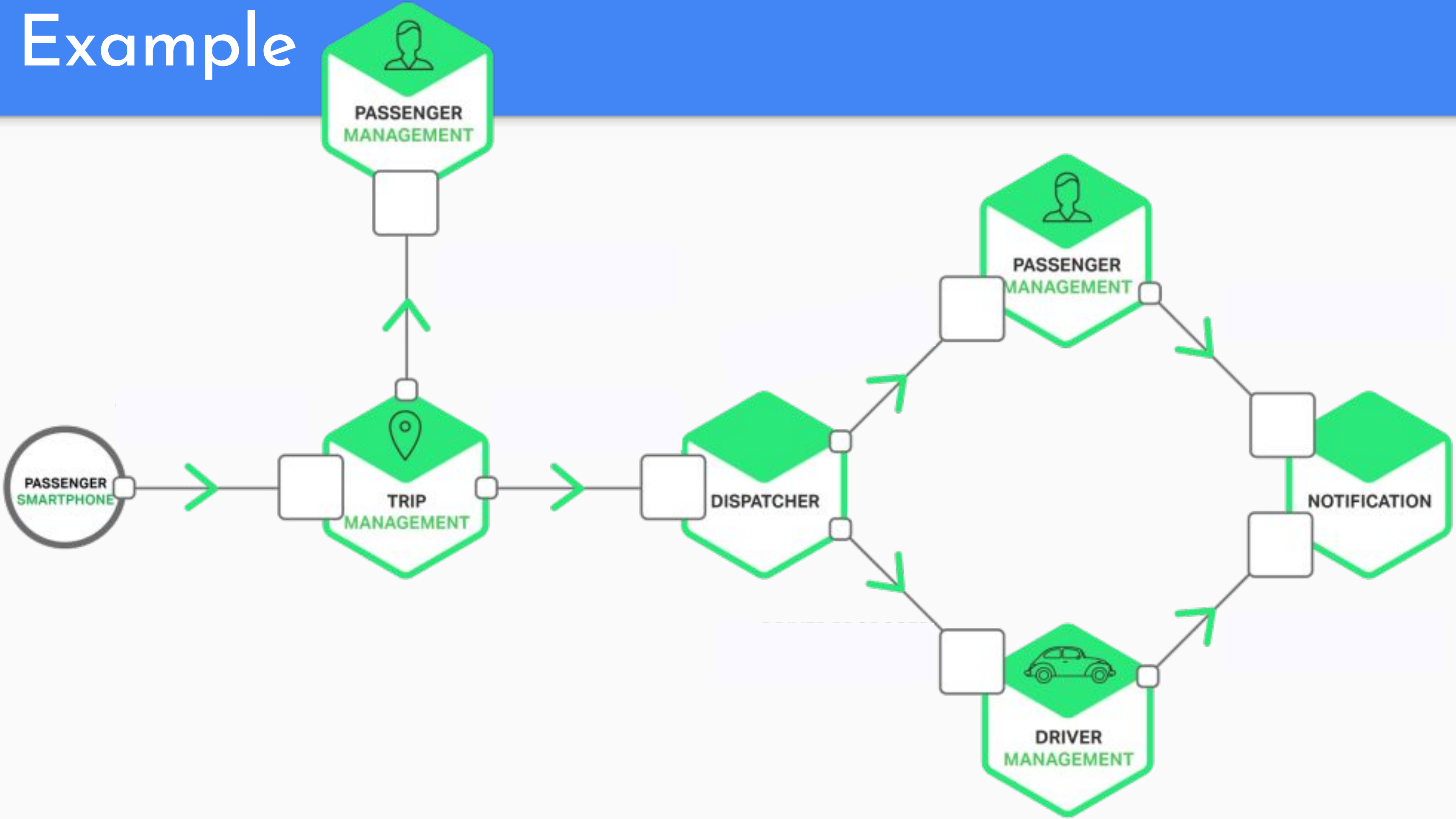




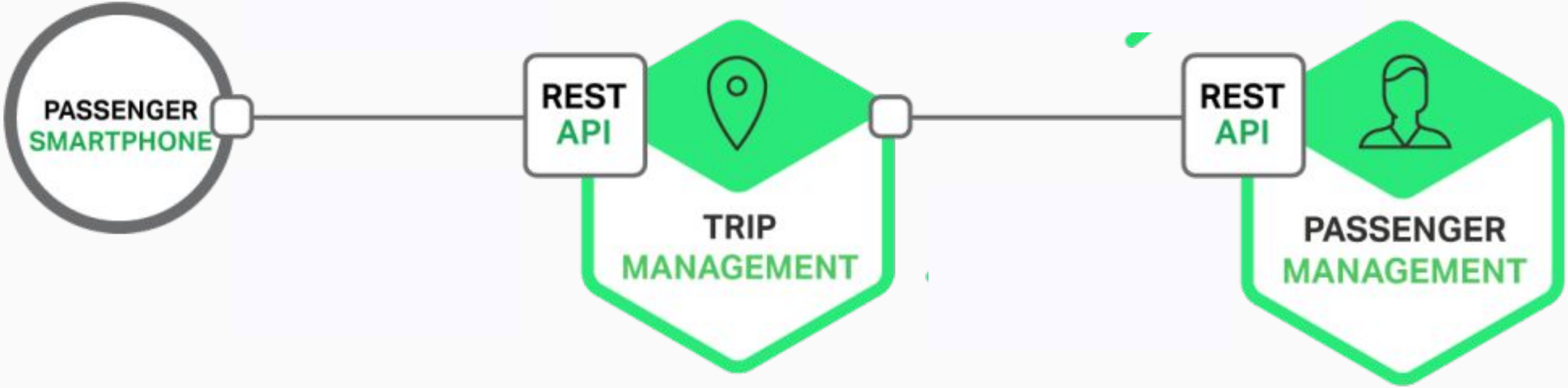
Publish/subscribe

Publish/async responses

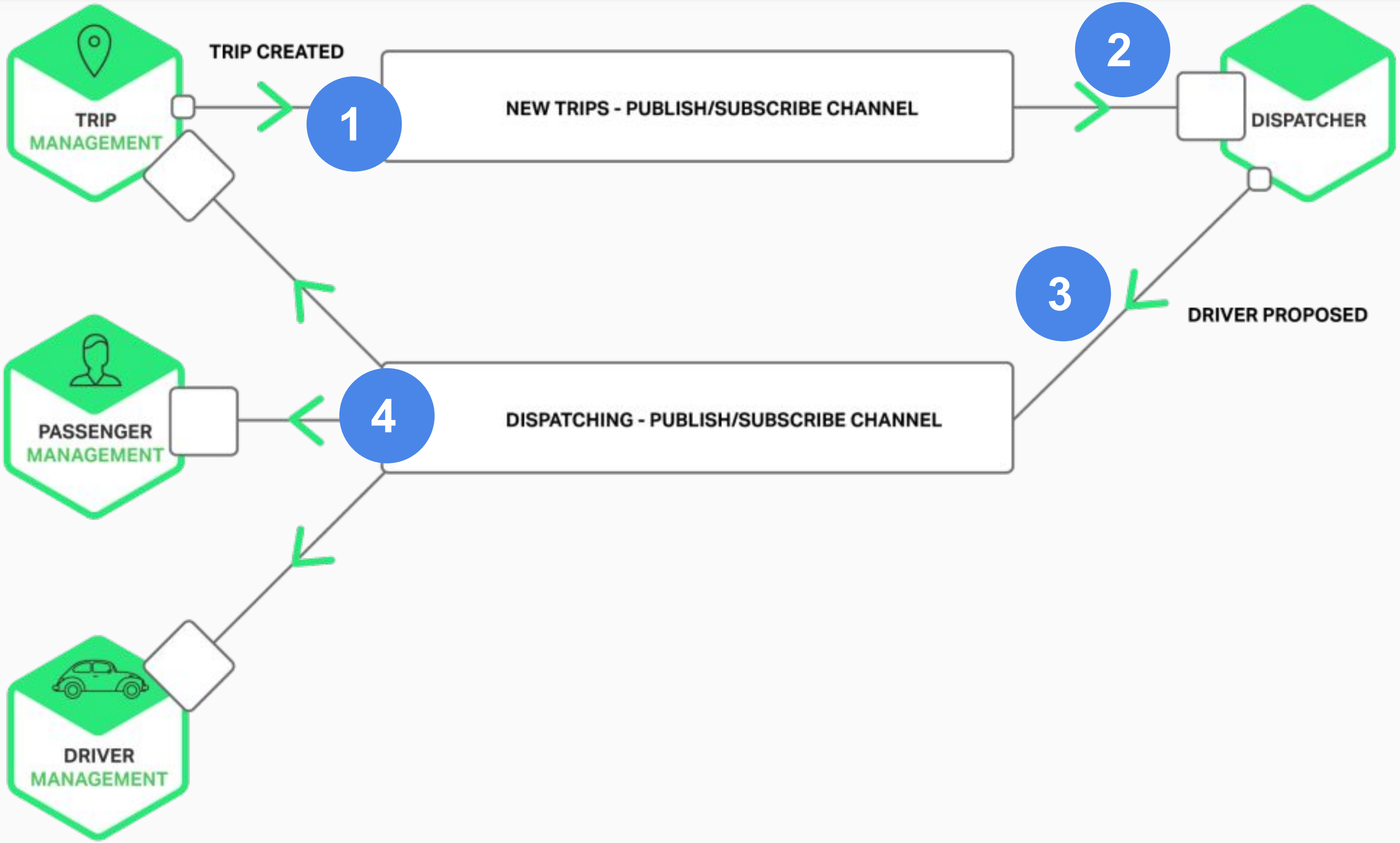
Example


PASSENGER
MANAGEMENT
PASSENGER
SMARTPHONE
TRIP
MANAGEMENT
DISPATCHER
PASSENGER
MANAGEMENT
DRIVER
MANAGEMENT
NOTIFICATION

REST


PASSENGER
SMARTPHONE
REST
API
TRIP
MANAGEMENT
REST
API
PASSENGER
MANAGEMENT

Messaging


TRIP
MANAGEMENT
TRIP CREATED
1
NEW TRIPS - PUBLISH/SUBSCRIBE CHANNEL
2
DISPATCHER
3
DRIVER PROPOSED
PASSENGER
MANAGEMENT
4
DISPATCHING - PUBLISH/SUBSCRIBE CHANNEL
DRIVER
MANAGEMENT

API Guidelines

API First approach

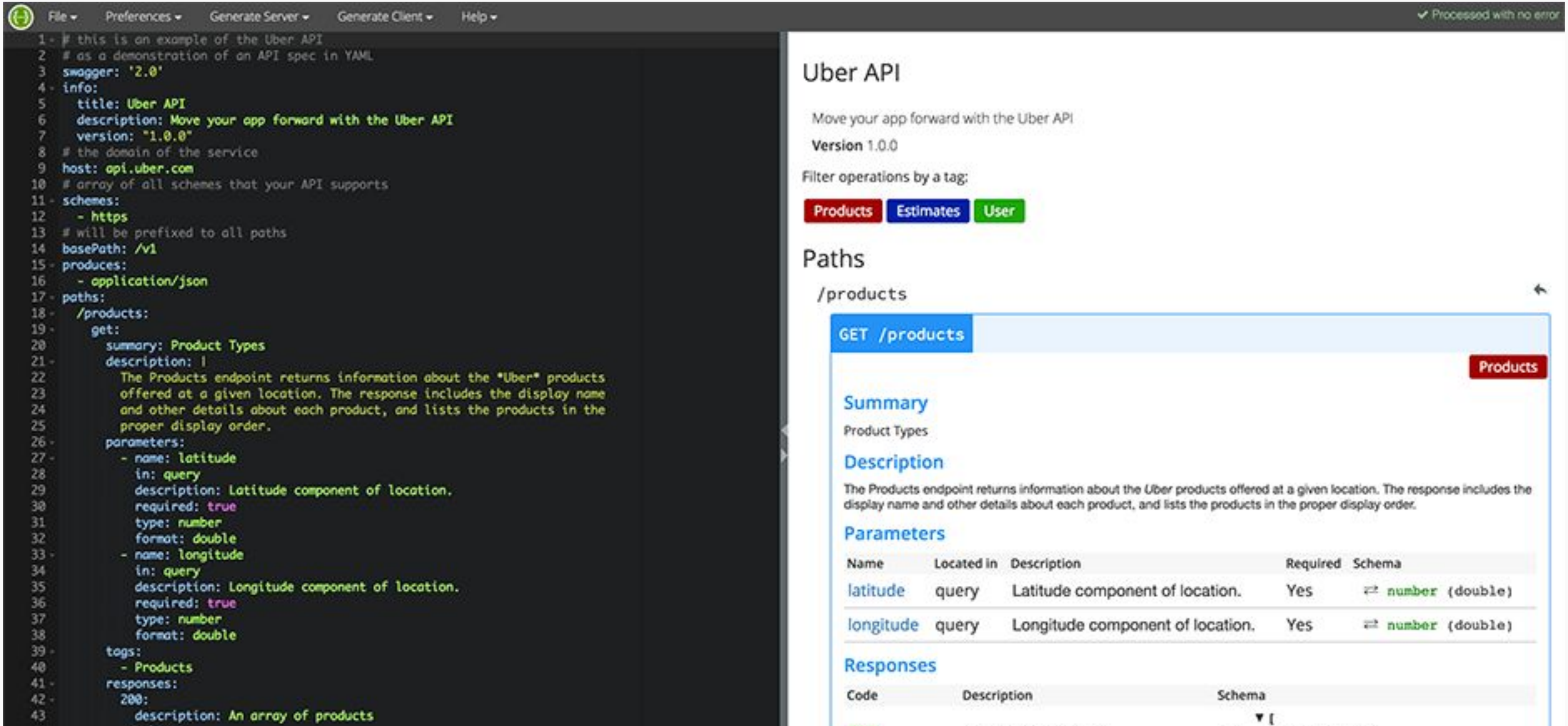
Evolving APIs

- Two versions running concurrently
  - Embed version in the url `/api/v2/trips`
- Robustness principle / Tolerate Unrelated Changes (minor changes)

Common ways for sync communication

Way	Protocol	Interaction Models	
REST	HTTP	Request/Response	
gRPC	HTTP2	Request/Response Request/Stream	Protobuf as content type
RSocket	WebSockets TCP Aeron	Request/Response Request/Stream Channel	Reactive

OpenApi/Swagger



```yaml
# this is an example of the Uber API
# as a demonstration of an API spec in YAML
swagger: '2.0'
info:
  title: Uber API
  description: Move your app forward with the Uber API
  version: "1.0.0"
# the domain of the service
host: api.uber.com
# array of all schemes that your API supports
schemes:
  - https
# will be prefixed to all paths
basePath: /v1
produces:
  - application/json
paths:
  /products:
    get:
      summary: Product Types
      description: |
        The Products endpoint returns information about the *Uber* products
        offered at a given location. The response includes the display name
        and other details about each product, and lists the products in the
        proper display order.
      parameters:
        - name: latitude
          in: query
          description: Latitude component of location.
          required: true
          type: number
          format: double
        - name: longitude
          in: query
          description: Longitude component of location.
          required: true
          type: number
          format: double
      tags:
        - Products
      responses:
        200:
          description: An array of products
```

Uber API

Move your app forward with the Uber API

Version 1.0.0

Filter operations by a tag:

Paths

/products

GET /products

Summary

Product Types

Description

The Products endpoint returns information about the *Uber* products offered at a given location. The response includes the display name and other details about each product, and lists the products in the proper display order.

Parameters

Name	Located in	Description	Required	Schema
latitude	query	Latitude component of location.	Yes	⇄ number (double)
longitude	query	Longitude component of location.	Yes	⇄ number (double)

Responses

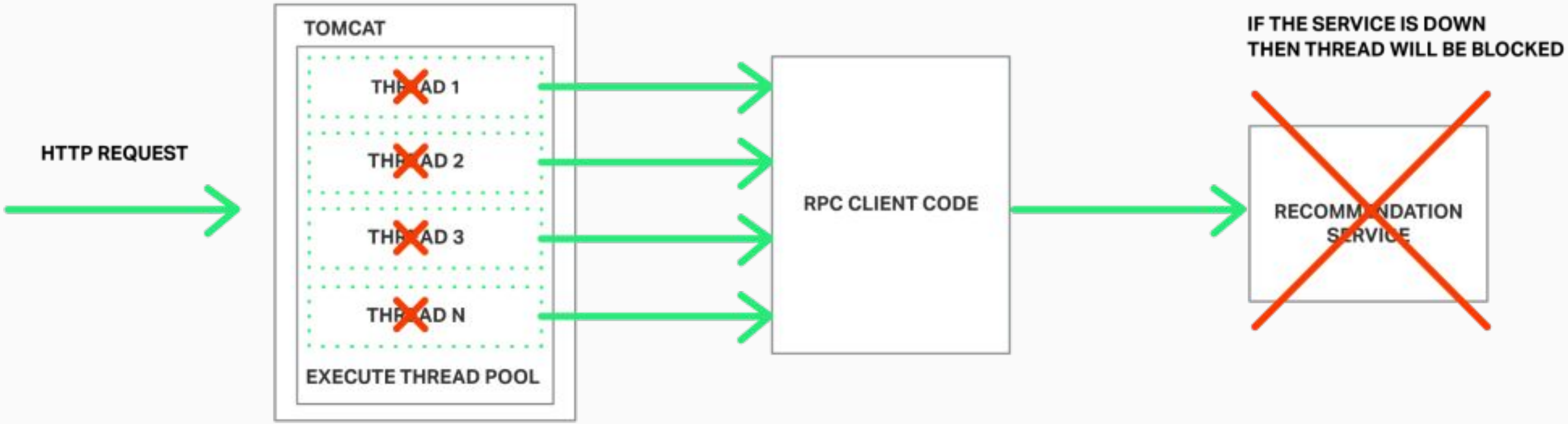
Code	Description	Schema
		▼ [

REST Maturity Levels

Level 0	Single URL for all requests
Level 1	Resources
Level 2	HTTP verbs
Level 3	HATEOAS

Communication challenges

Failure



IF THE SERVICE IS DOWN
THEN THREAD WILL BE BLOCKED

EVENTUALLY ALL THREADS WILL BE BLOCKED

Resilience

Put network timeouts

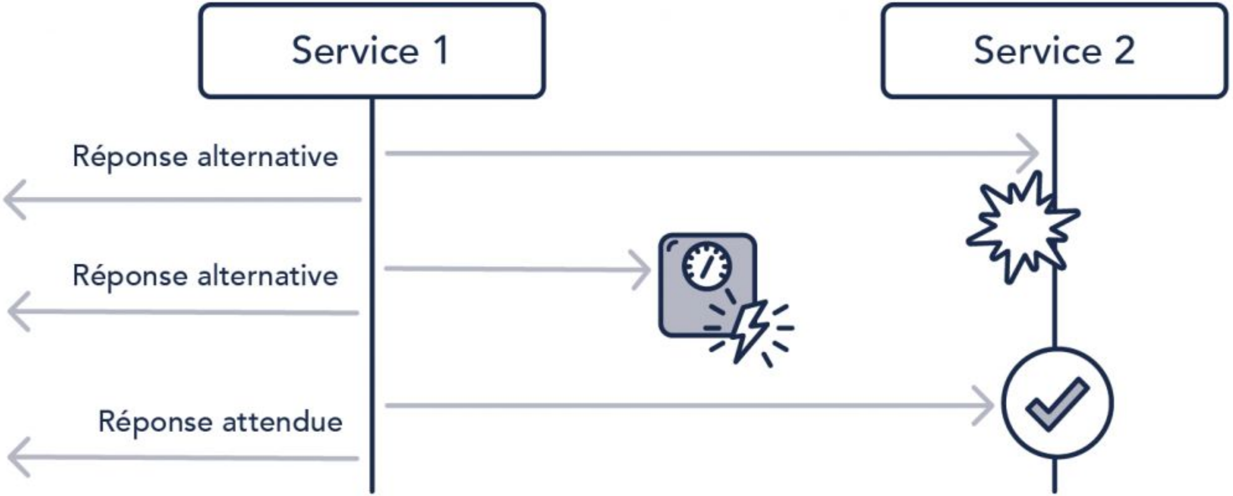
Limit the number of pending requests with a particular service

Provide fallbacks

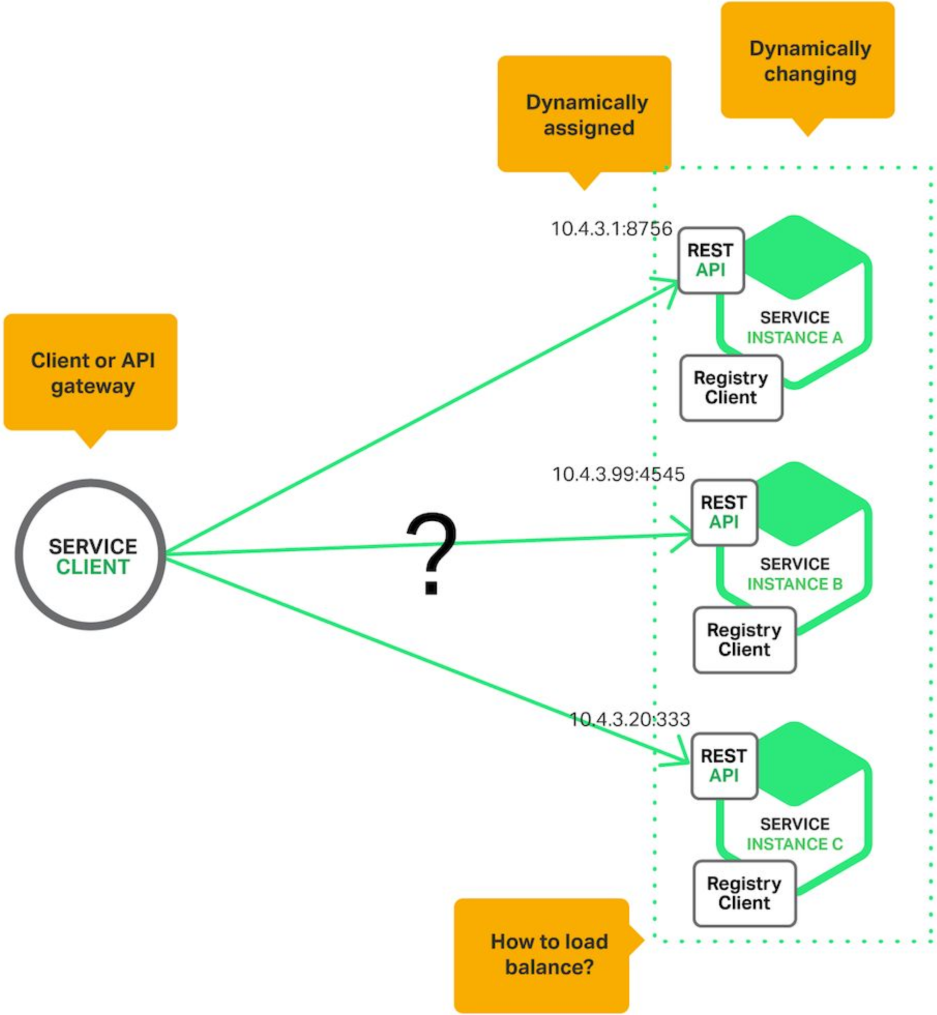
Auto retry

Pattern: Circuit Breaker

- A proxy
- When the number of consecutive failures crosses a threshold, the circuit breaker trips, further attempts should fail immediately.



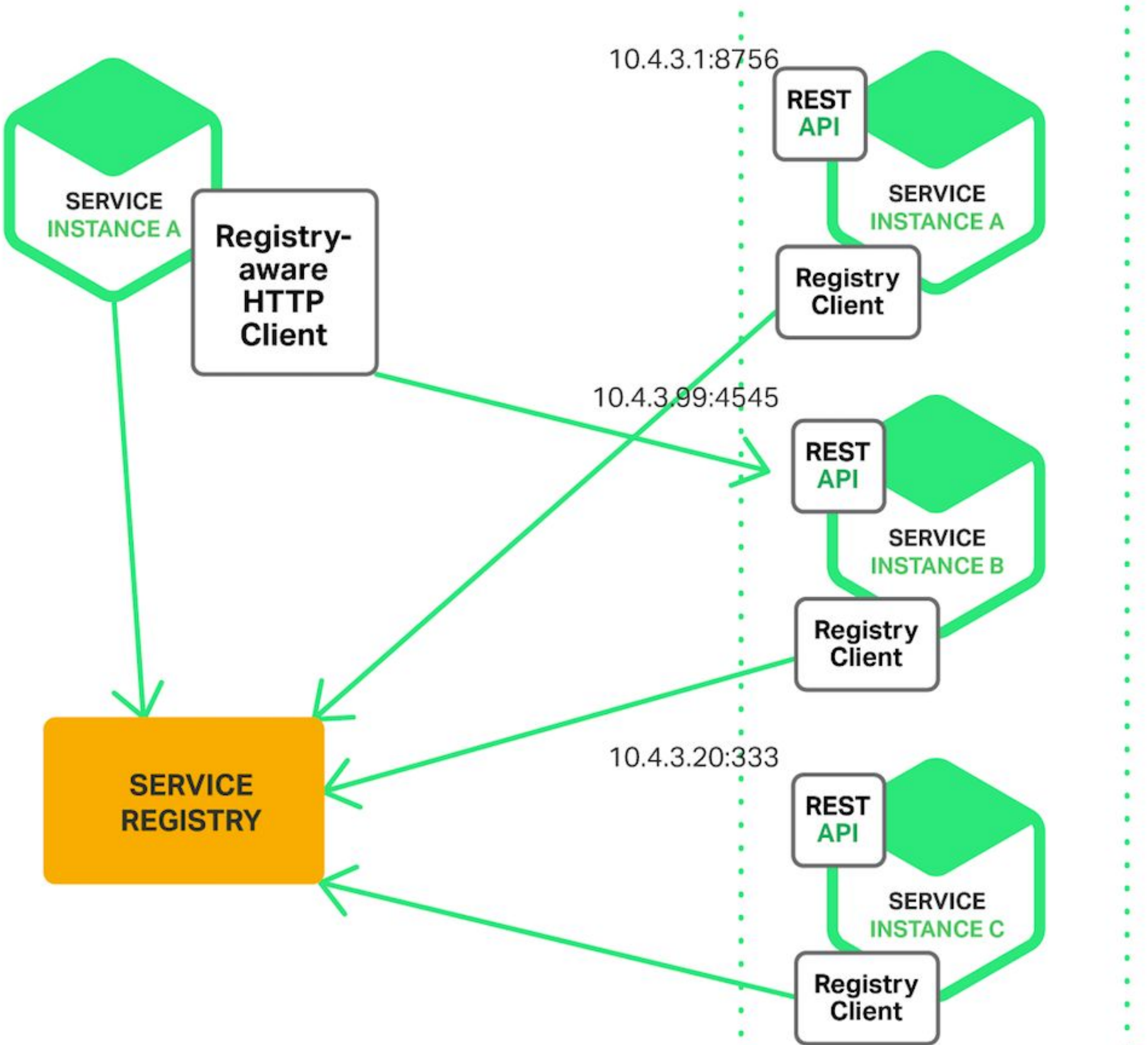
What do you think?



Pattern: Client-side service discovery

Service lookup

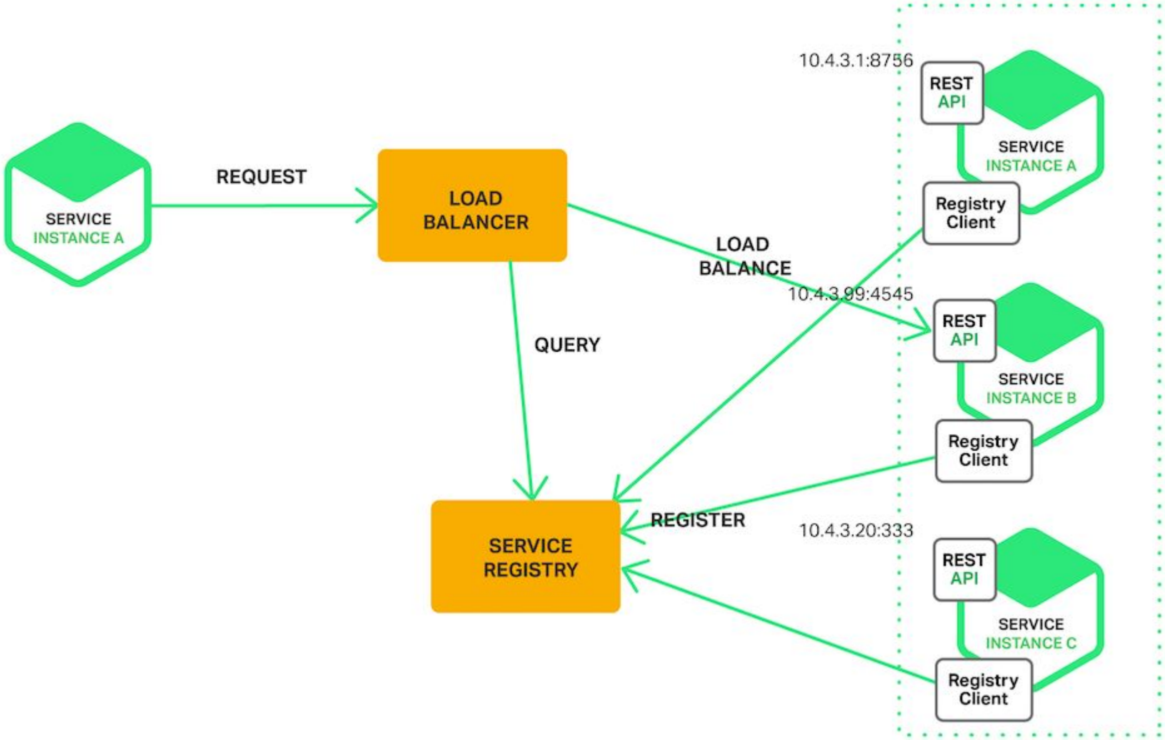
Heartbeat



Pattern: Server-side service discovery

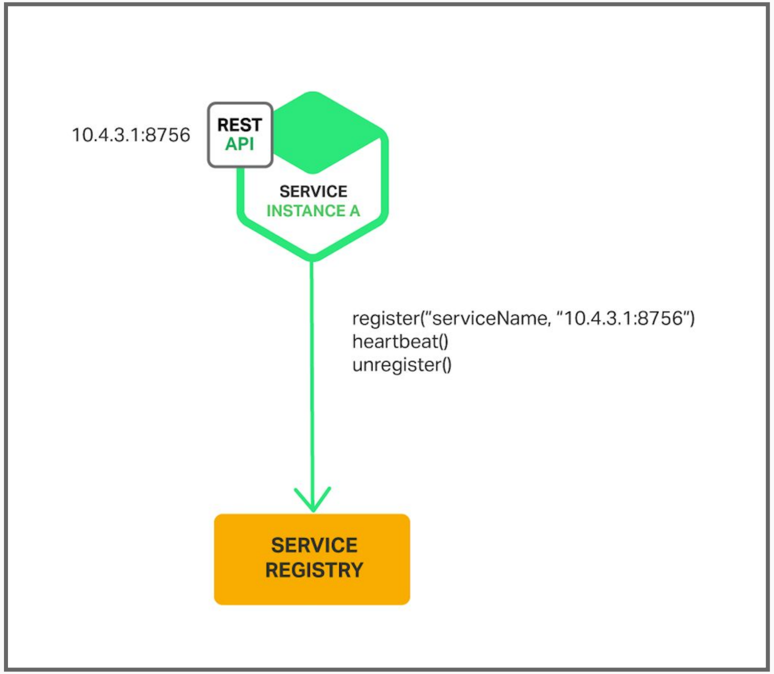
The client makes a request to a service via a load balancer.

The load balancer queries the service registry and routes each request to an available service instance

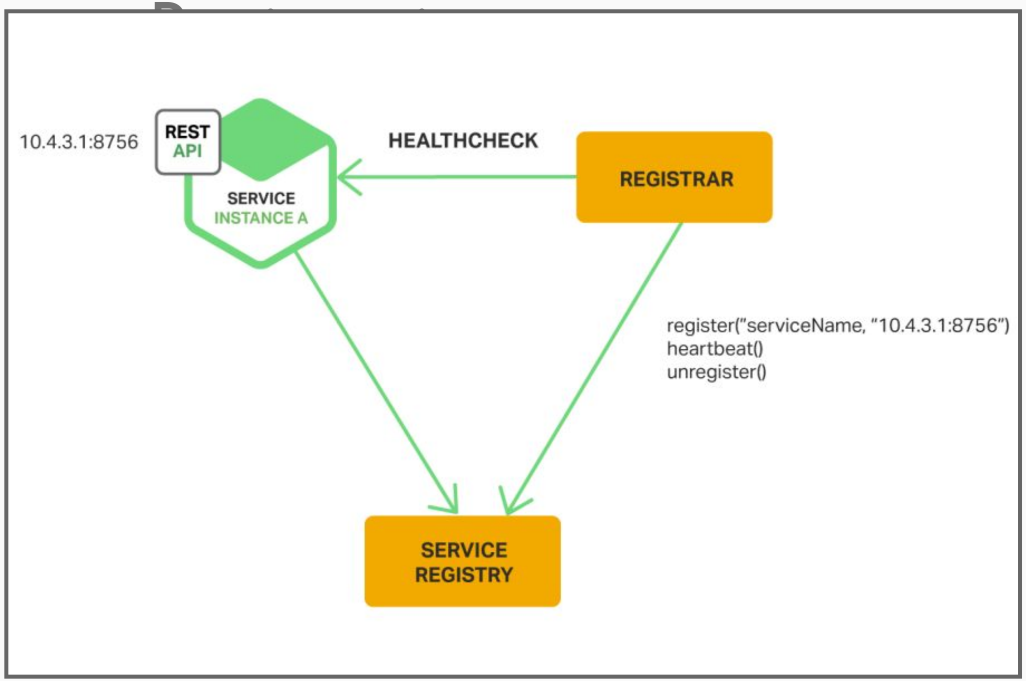


Service Registration Options

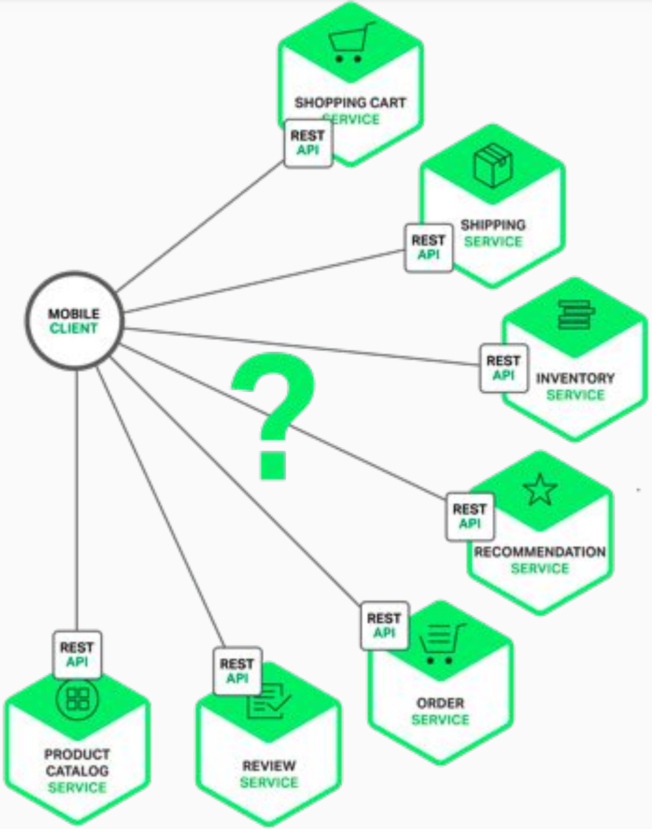
Pattern: Self-Registration

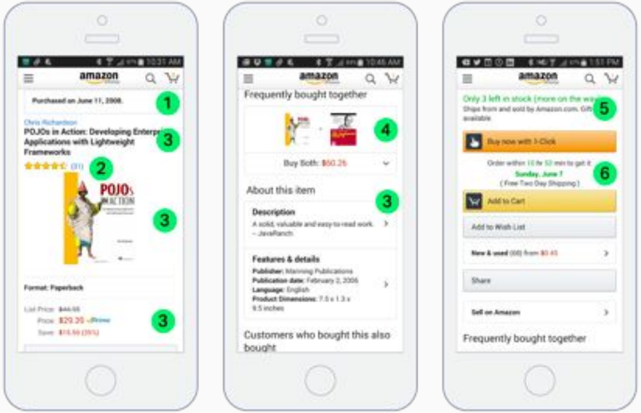


Pattern: Third-Party



Calling a lot of services




amazon
Purchased on June 11, 2008.
1
Chris Richardson
POJOs in Action: Developing Enterpr
Applications with Lightweight
Frameworks
3
(31)
2
POJOs IN ACTION
3
Format: Paperback
List Price: $44.95
Price: $29.39 ✓Prime
Save: $15.56 (35%)
3
amazon
Frequently bought together
4
Buy Both: $60.26
About this item
3
Description
A solid, valuable and easy-to-read work.
– JavaRanch
Features & details
Publisher: Manning Publications
Publication date: February 2, 2006
Language: English
Product Dimensions: 7.5 x 1.3 x 9.5 inches
Customers who bought this also bought
amazon
Only 3 left in stock (more on the wa
Ships from and sold by Amazon.com. Gift
available
5
Buy now with 1-Click
Order within 10 hr 53 min to get it
Sunday, June 7
(Free Two Day Shipping)
6
Add to Cart
Add to Wish List
New & used (88) from $0.45
Share
Sell on Amazon
Frequently bought together

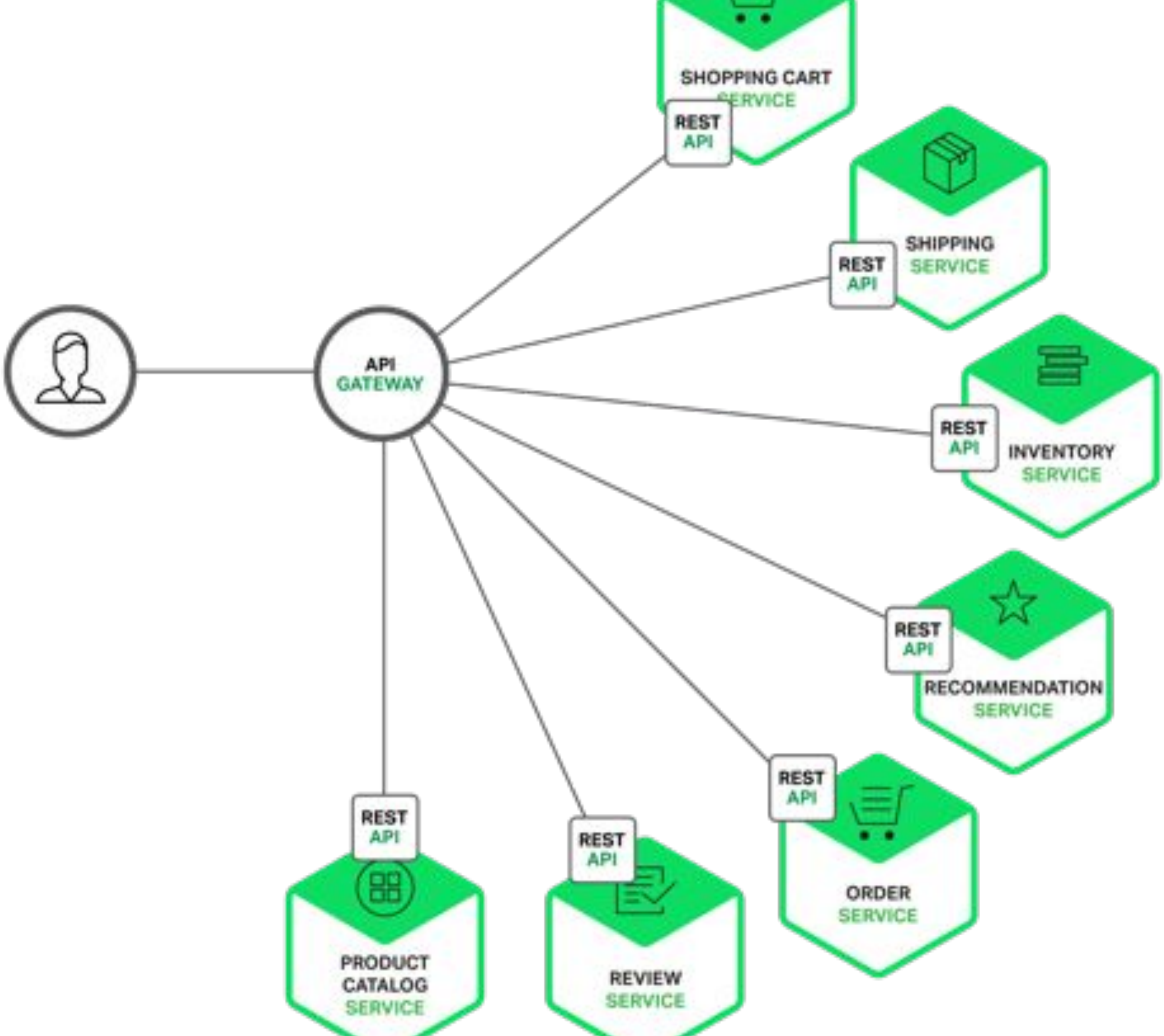
1. ORDER HISTORY
2. REVIEWS
3. BASIC PRODUCT INFO
4. RECOMMENDATION
5. INVENTORY
6. SHIPPING

Pattern: API Gateway

“Aggregation Service” (Facade)

Responsibilities: request routing, composition, and protocol translation

Other responsibilities: authentication, monitoring, load balancing, caching, request shaping



The Twelve factor app

I. Codebase

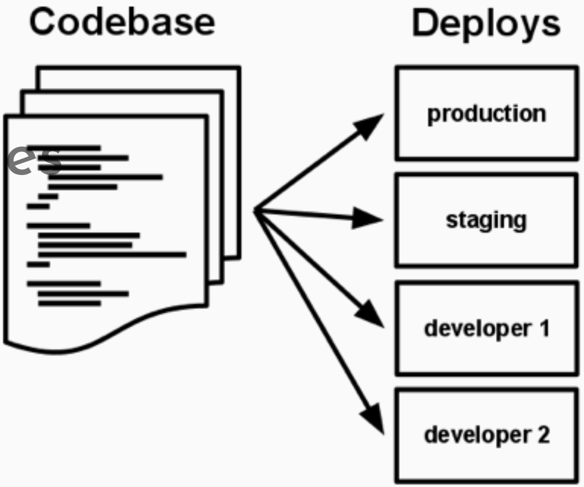
One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Config

Store config in the environment



IV. Backing services

Treat backing services as attached resources

V. Build, release, run

Strictly separate build and run stages

VI. Processes

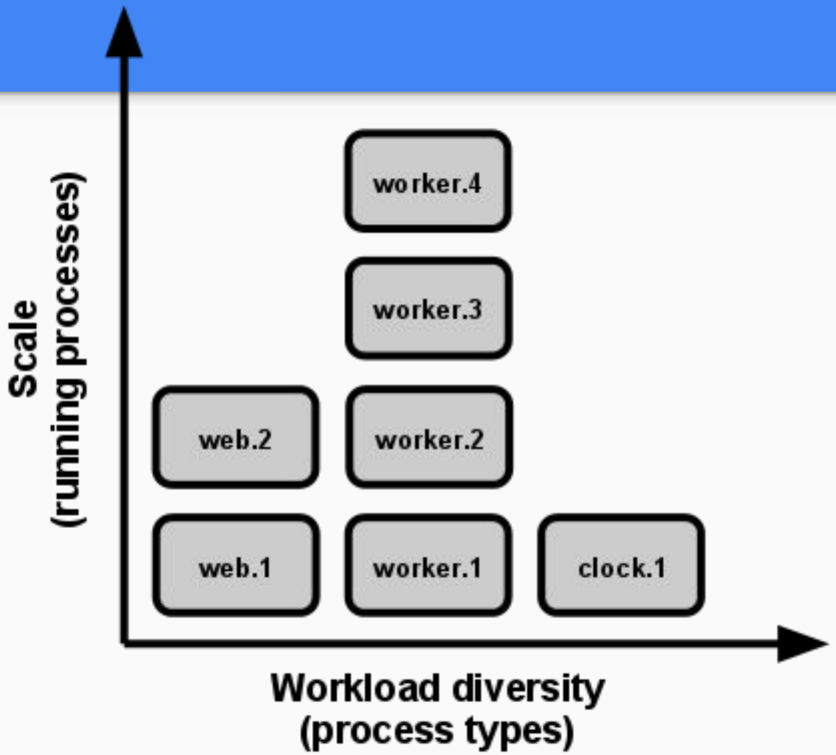
Execute the app as one or more stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model



IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep development, staging, and production as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

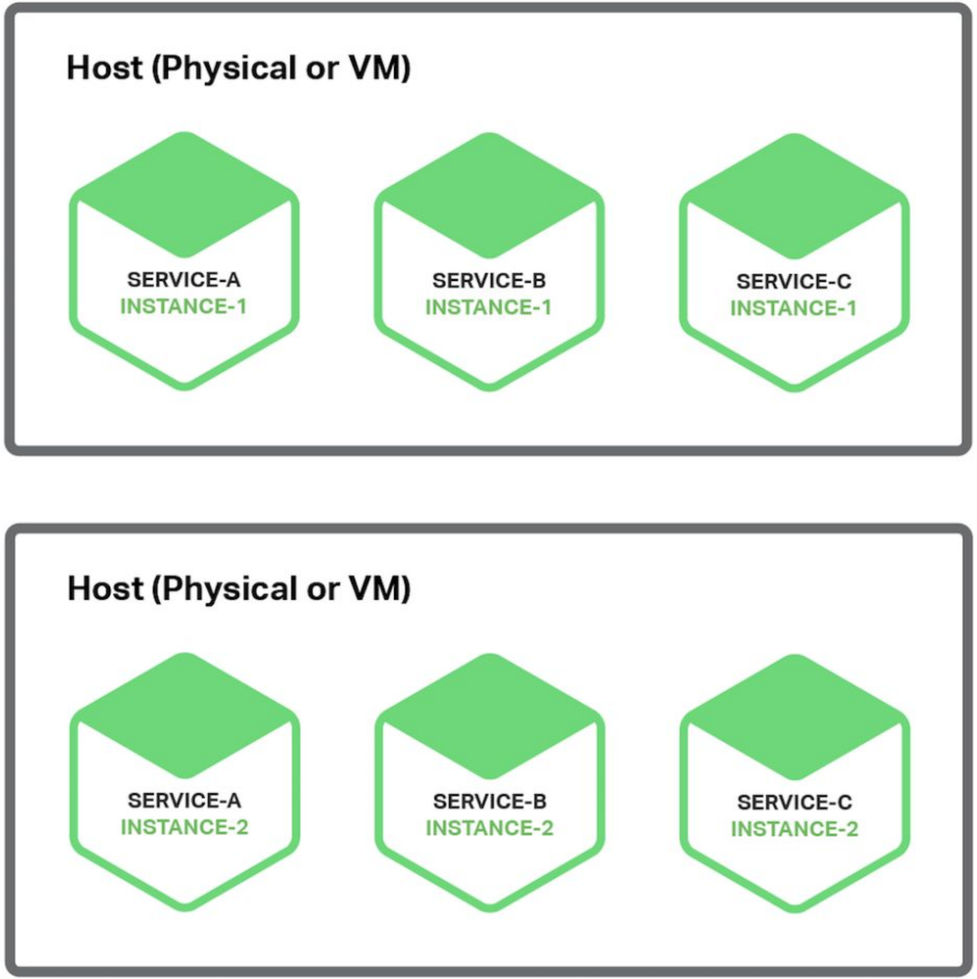
Run admin/management tasks as one-off processes

Deployment

Pattern: Multiple Service Instances per Host

More efficient, Better resources usage, Faster deployment

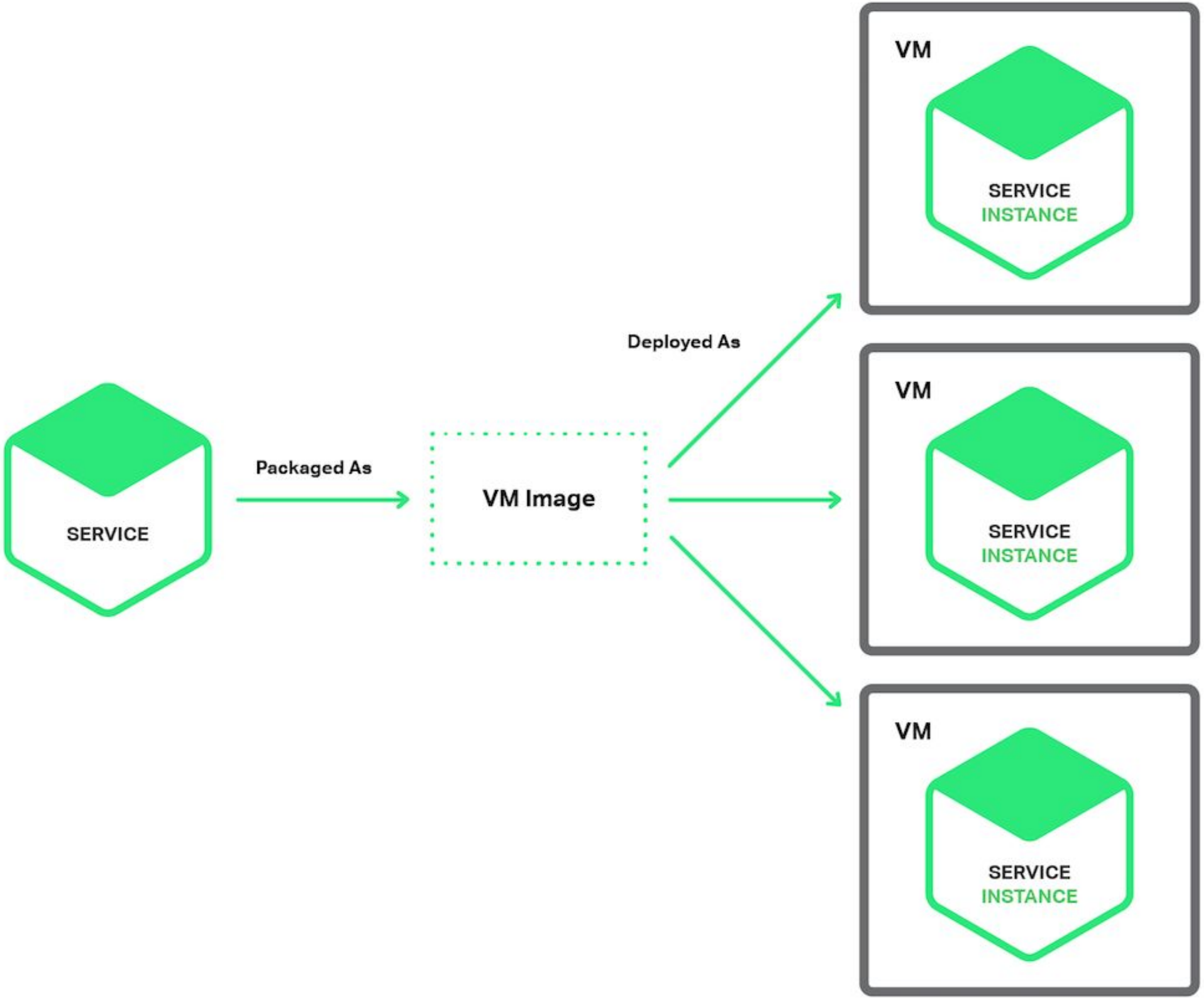
No Isolation



Pattern: Service Instance per Host

Service Instance per Virtual Machine

Service Instance per Container



5

Seconds

to package a Spring Boot application as a Docker container

No lengthy OS boot mechanism

Pattern: Serverless Deployment

No IT infrastructure configuration needed

Like: AWS Lambda, Google Cloud Functions, Azure Functions

Deployment

Mobile

External configuration/secrets

Environment variables, Filesystem, Shared key/value store

Feature flag

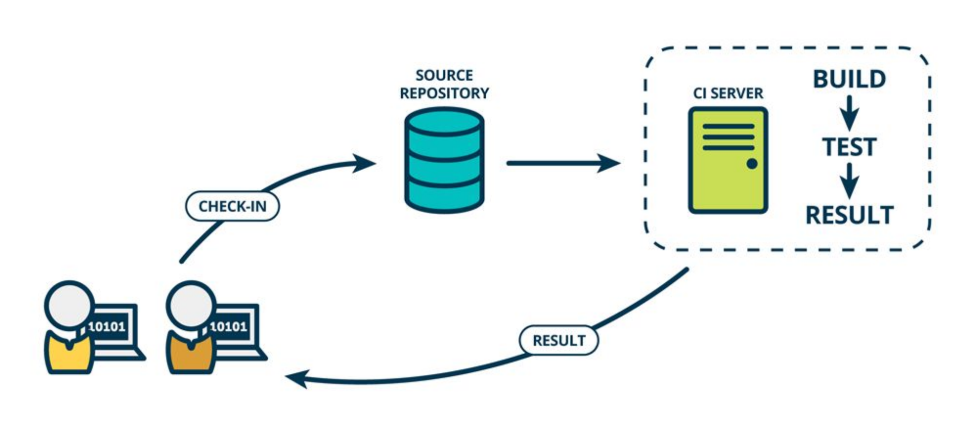
Zero-downtime updates

Continuous Integration & Delivery

Release smaller sets of changes faster, Rather than tackle a large piece of work in one go

Small changes in releases are

- Easy to test
- Simplifies code review
- Easier to release and deploy



Other concerns

Caching

Centralized Logging, Correlation IDs

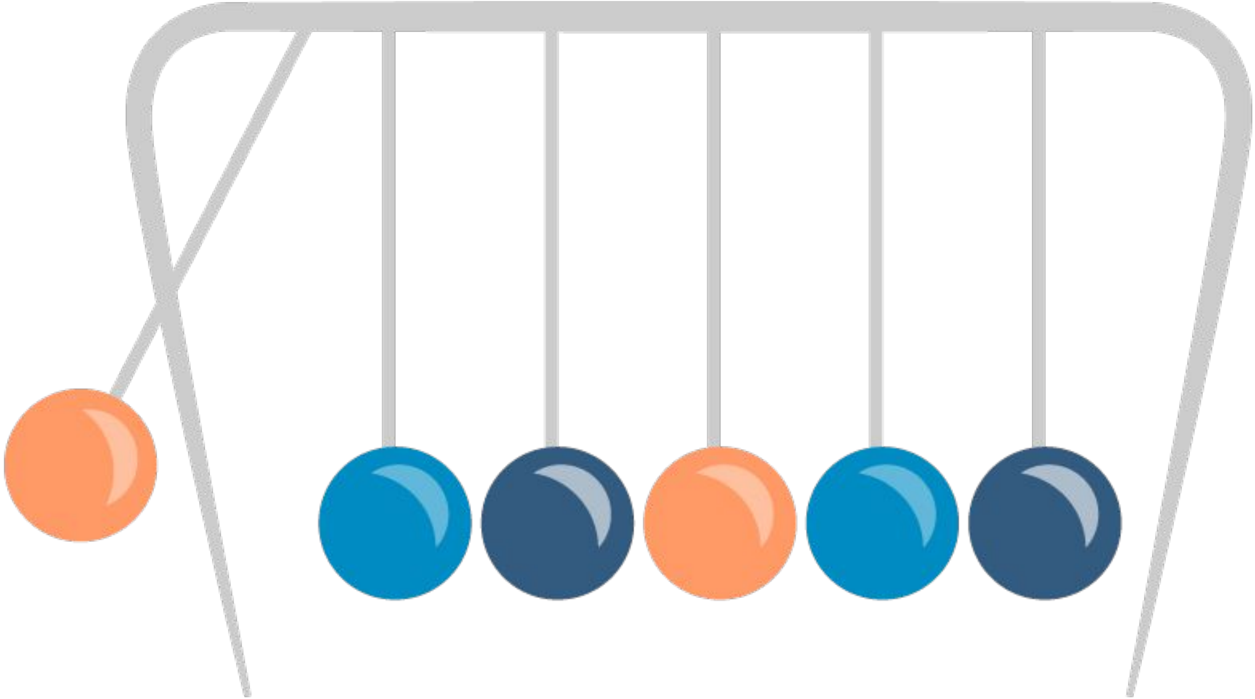
Centralized Monitoring

Shared libraries

Reactive manifesto

Reactive Systems are:

- Responsive
- Resilient
  - Isolation
  - Replication
- Elastic
- Message Driven
  - location transparency
  - back-pressur

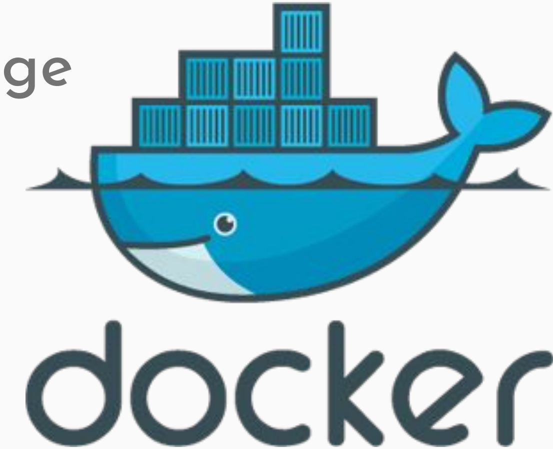


Tools

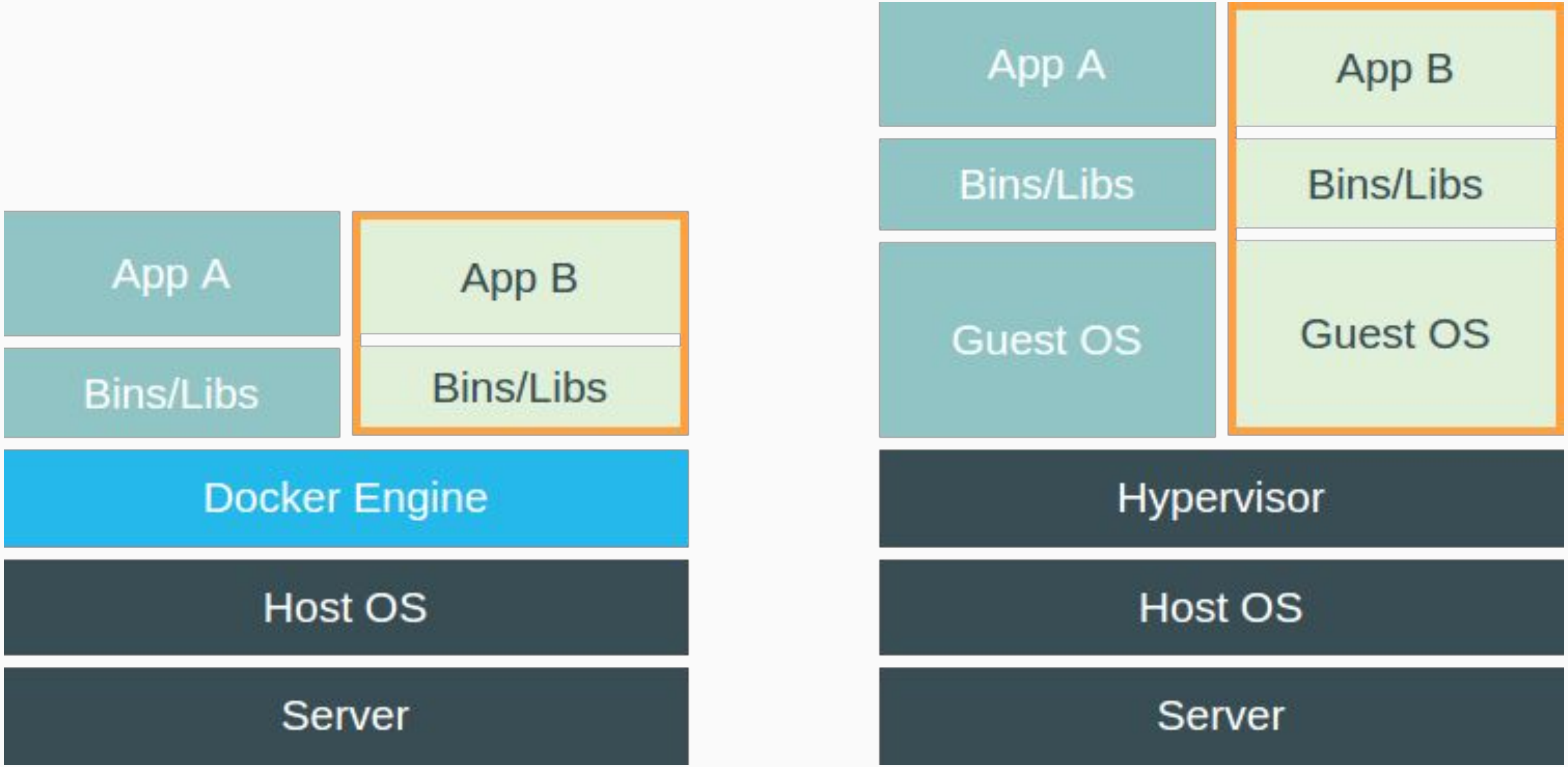
Docker

A tool designed to make it easier to create, deploy, and run applications by using containers

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package



Docker vs VM



App A

App B

Bins/Libs

Bins/Libs

Docker Engine

Host OS

Server

App A

App B

Bins/Libs

Bins/Libs

Guest OS

Guest OS

Hypervisor

Host OS

Server

Simple dockerfile example

```
docker build -t tomcat .

docker run -p 8080:8080 tomcat
```

```
FROM java

MAINTAINER Bala

RUN curl -O http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.55/bin/apache-tomcat-7.0.55.tar.gz

RUN tar xzf apache-tomcat-7.0.55.tar.gz

ADD sample.war apache-tomcat-7.0.55/webapps/

CMD apache-tomcat-7.0.55/bin/startup.sh && tail -f apache-tomcat-7.0.55/logs/catalina.out

EXPOSE 8080
```

More simple one

```
FROM tomcat:8-jre8

MAINTAINER "xxx <xxx@gmail.com">

ADD sample.war /usr/local/tomcat/webapps/
```

Container Orchestration

Google Kubernetes

Docker Compose

Docker Swarm (integrated in Docker)

Kubernetes

A system for automating deployment, scaling, and management of containerized applications across multiple hosts

It groups containers that make up an application into logical units for easy management and discovery

References

Introduction to microservices

https://www.nginx.com/blog/introduction-to-microservices/

Microservices Patterns

http://microservices.io/

Best Practices for Building a Microservice Architecture

http://www.vinaysahni.com/best-practices-for-building-a-microservice-architecture

Thanks!

Hany Ahmed

hanylink@gmail.com

www.namozag.com