



# 1. INTRODUCTION TO PLATFORMS

Darryl Ng

[darryl.ng@nus.edu.sg](mailto:darryl.ng@nus.edu.sg)

- Objectives
  - To introduce the core concepts of Platform Engineering
  - To discuss the difference between Software Platforms and Software Applications
  - To introduce the paradigm shift in Platform Thinking
- Topics
  - Definition of Software Platforms
  - Platforms vs Applications
  - Platform thinking
  - Business ecosystems
  - Exercise: Analyse examples of platforms



# What is a Platform

## Contemporary Understanding

*“A software platform is a software-based product or service that serves as a foundation on which outside parties can build complementary products or services”*

- It is an extensible software-based system that provides the core functionality shared by “apps” that interoperate with it, and the interfaces through which they interoperate
- Both internal consumption and external complementary services

*Ref: Baldwin and Woodard, 2009 ; Tiwana et al., 2010*



# What is a Platform

## Contemporary Understanding

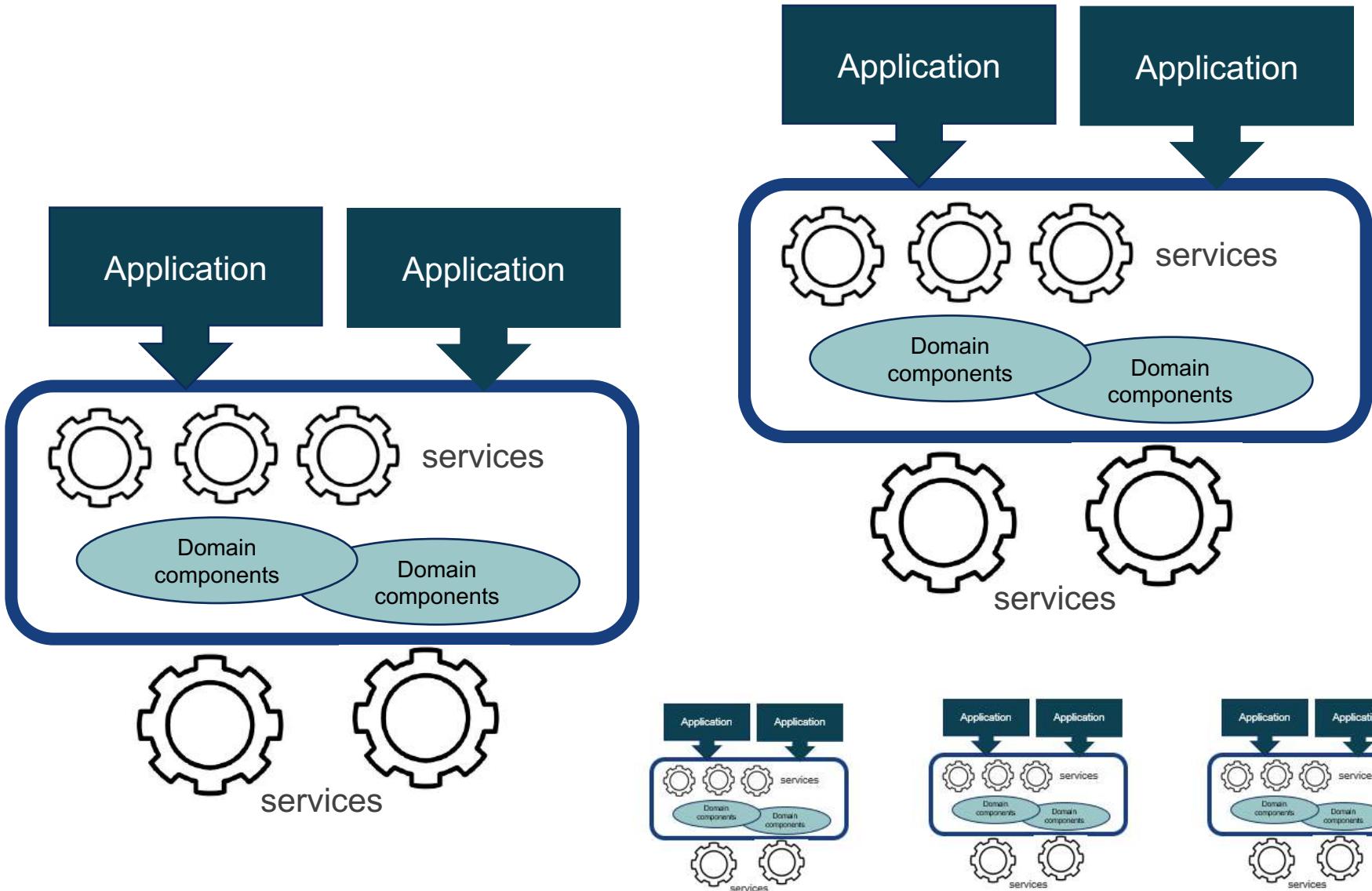
*“A platform is a set of services on which applications can be built and run on top of. Modern platforms blur the line between infrastructure architecture and application architecture, needing to address complex issues such as service discovery, resource coordination, container orchestration and usage reporting”*

- Platform engineering, then, is the practice of designing, building and operating platforms for Application development teams to leverage.
- Both an application architecture and an infrastructure to host the services

Ref:<http://www.kovarus.com/article/platform-engineering/>



# What is a Platform





# Examples of Software Platforms

- Services Platform: AirBnB, Uber, Grab etc
- Marketplace Platform: eBay, Amazon, Carousell etc.
- Payment Platform: PayPal, Stripes, PayNow, etc.
- Social Networking: Facebook, Twitter, LinkedIn etc.
- Communication Platform: WhatsApp, Skype, WeChat etc.
- Content Platforms: YouTube, Amazon Kindle, iTunes etc.
- Software Development Platforms: Linux, Android etc.
- And so on...

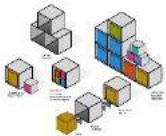
Why do you think these are Platforms?





# Characteristics of a Platform

Contains Composable Services and components



Rich User Community



Stays Relevant and Data-Driven



Quick to start using



Promotes interaction amongst partners



Secure and Compliant



Enables Business Agility





# Characteristics of a Platform

- The platform is **composable**, containing components and discrete services that can be used independently.
- The platform is **quick and cheap to start** using, with an easy on-ramp (e.g. Quick start guides, documentation, code samples)
- The platform has a rich internal user community for sharing
- Actively promotes interactions among different partners
- The platform is secure and compliant by default
- The platform is extensible and enables business agility
- The platform is up to date and provides feedback loop by collecting and analysing data



# Benefits of building a Platform

Interoperability  
With disparate systems



Service Model  
fit into value-chain  
and no longer  
sequential



Separation of  
Intra (external)  
vs inter (internal)  
platform changes



Quicker response  
to opportunities



Data and information  
available to support  
decision making



Agile Governance and  
Risk Management



Team, skills and culture  
re-imagined





# Benefits of building a Platform

- Embrace the rapid changes in technology with technologies that are **interoperable**
- Allows intra platform change to happen at a different pace to inter platform, without overly impacting each other
- Enables business to **implement new ways of working**, enabling them to respond to opportunities quickly
- Enables agile **Governance and Risk management**, while still protecting the needs of stakeholders
- Teams, skills and culture are reimagined to leverage this new capability
- Supports Data-driven decision-making
- The service model, across value chains, is flexible enough to respond to opportunities and counter threats.



# Benefits of building a Platform

- Embrace the rapid changes in technology with technologies that are interoperable
- Allows intra platform change to happen at a different pace to inter platform, without overly impacting each other
- Enables business to implement new ways of working, enabling them to respond to opportunities quickly
- Enables agile Governance and Risk management, while still protecting the needs of stakeholders
- Teams, skills and culture are reimagined to leverage this new capability
- Supports Data-driven decision-making
- The service model, across value chains, is flexible enough to respond to opportunities and counter threats.



# Platforms, Frameworks and Applications

- Platforms
  - An environment that software is designed to run in
  - Provide services and reusable components to build applications.
  - Provide means to extend the services and features of application by consuming and exposing services to external parties
  - Provides the software and hardware tools needed to run an application
  - Typically it is a part of a hosted service accessed by APIs
  - Collect data and provide tools to perform analytics to improve and monitor services
- Framework
  - Pre-defined skeleton with predefined functions and (abstract) classes
  - Provides guidelines and a specific way(s) of architecting applications supporting certain design patterns.
  - Needs to be downloaded and installed in a specific environment
  - Can be a part of a platform
- Libraries
  - Calling and using of reusable codes at the appropriate point of your applications

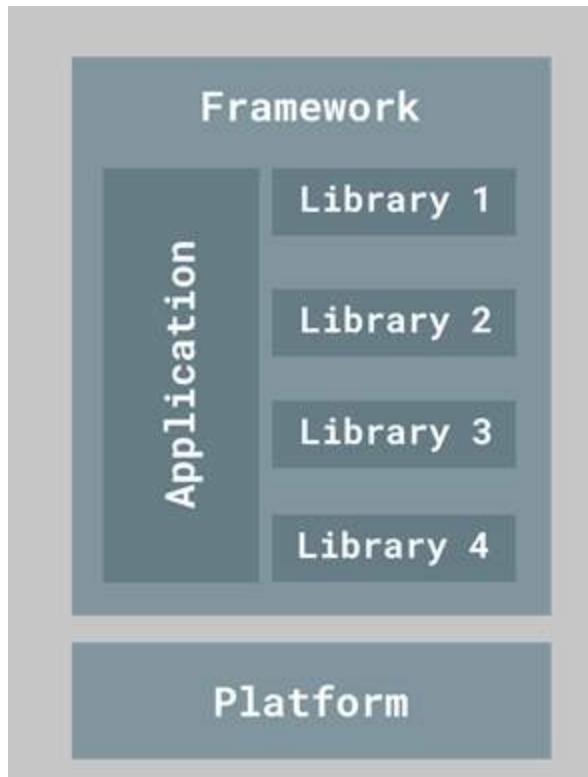


# Traditional -> Agile Approach

- Traditional Software to SaaS Software
- Example
  - Outsystems
- IaaS
- PaaS
- SaaS



# Platforms, Frameworks and Applications



**You use a framework.**

e.g. Spring, Apache Struts, .NET framework, ReactJS, AngularJS, Flask, Django, vue.js

**You build an application by calling libraries**

e.g. jQuery, Numpy, Keras, PyTorch, TensorFlow, Matplotlib

**You work on a platform**

e.g.

technology platform – AWS, AZURE

computing platforms – iOS, Android, Windows

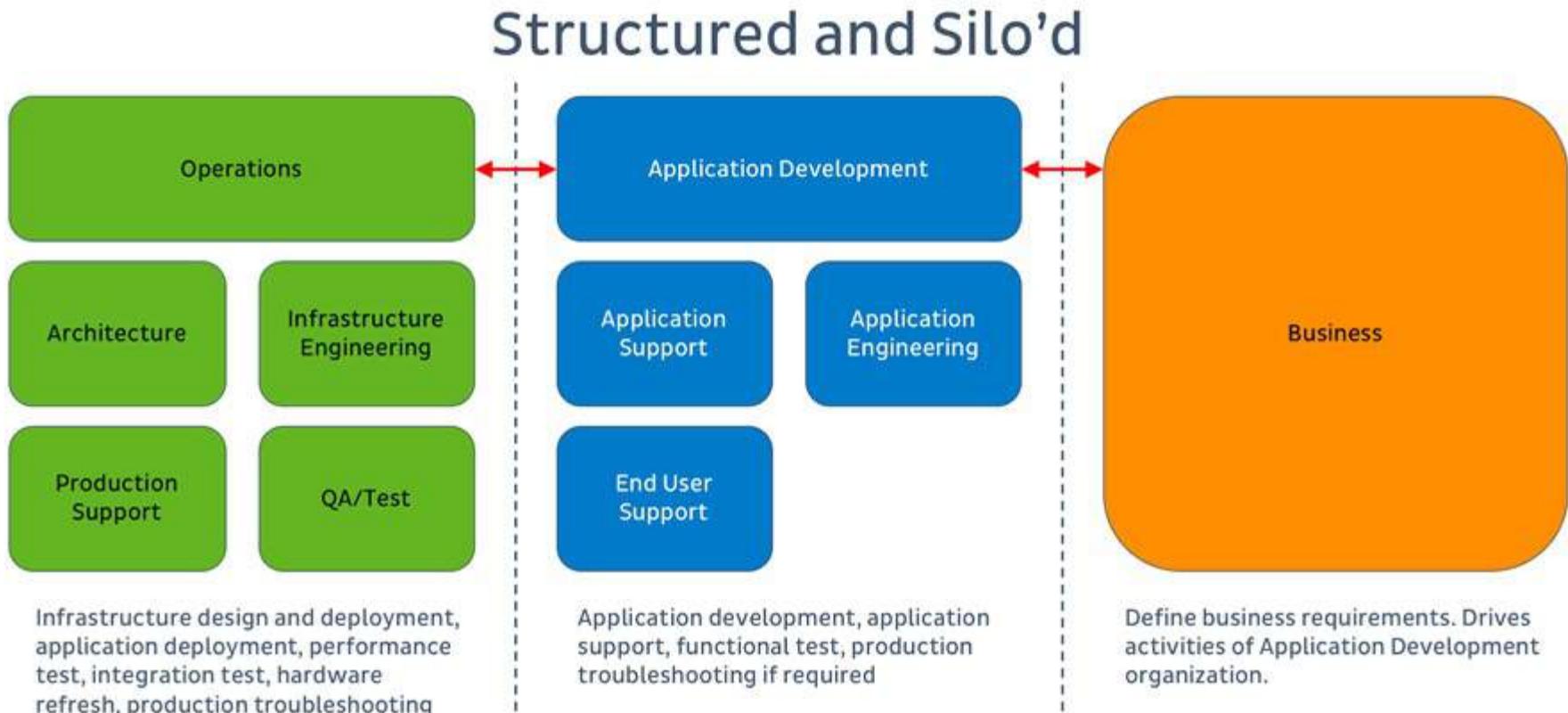
utility platforms – Google Search, Kayak

interaction platform – Wechat, Facebook,

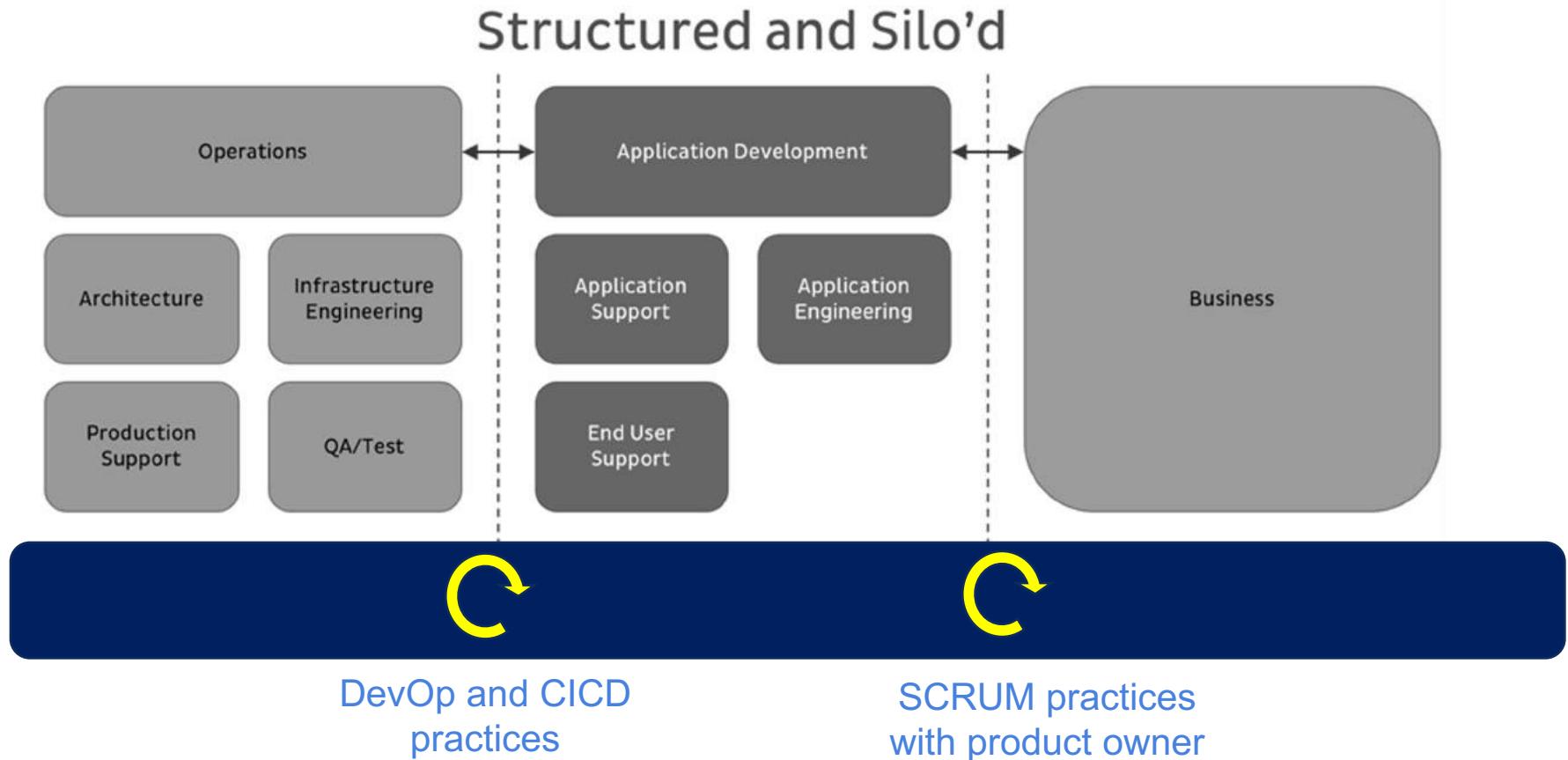
<https://medium.com/@i.e.rahu1/eli5-platform-vs-framework-vs-library-154539de8b>



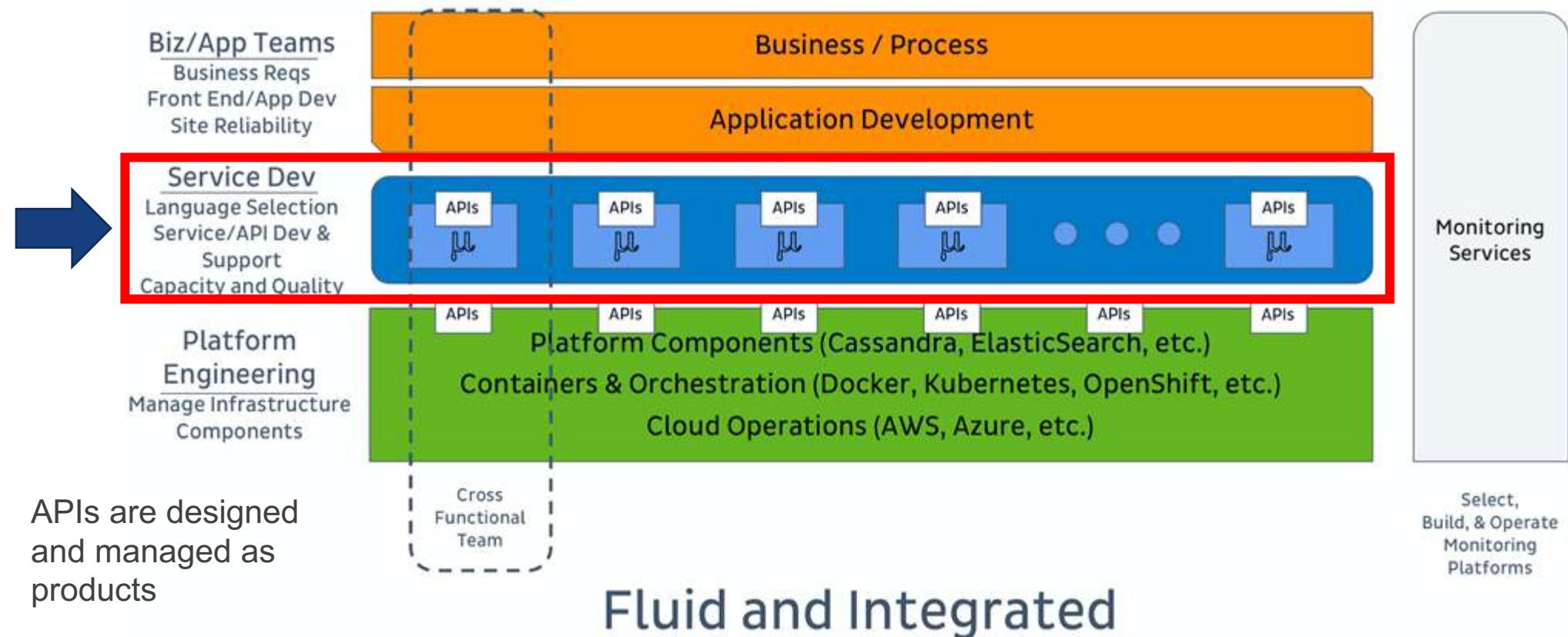
# Traditional Application Delivery



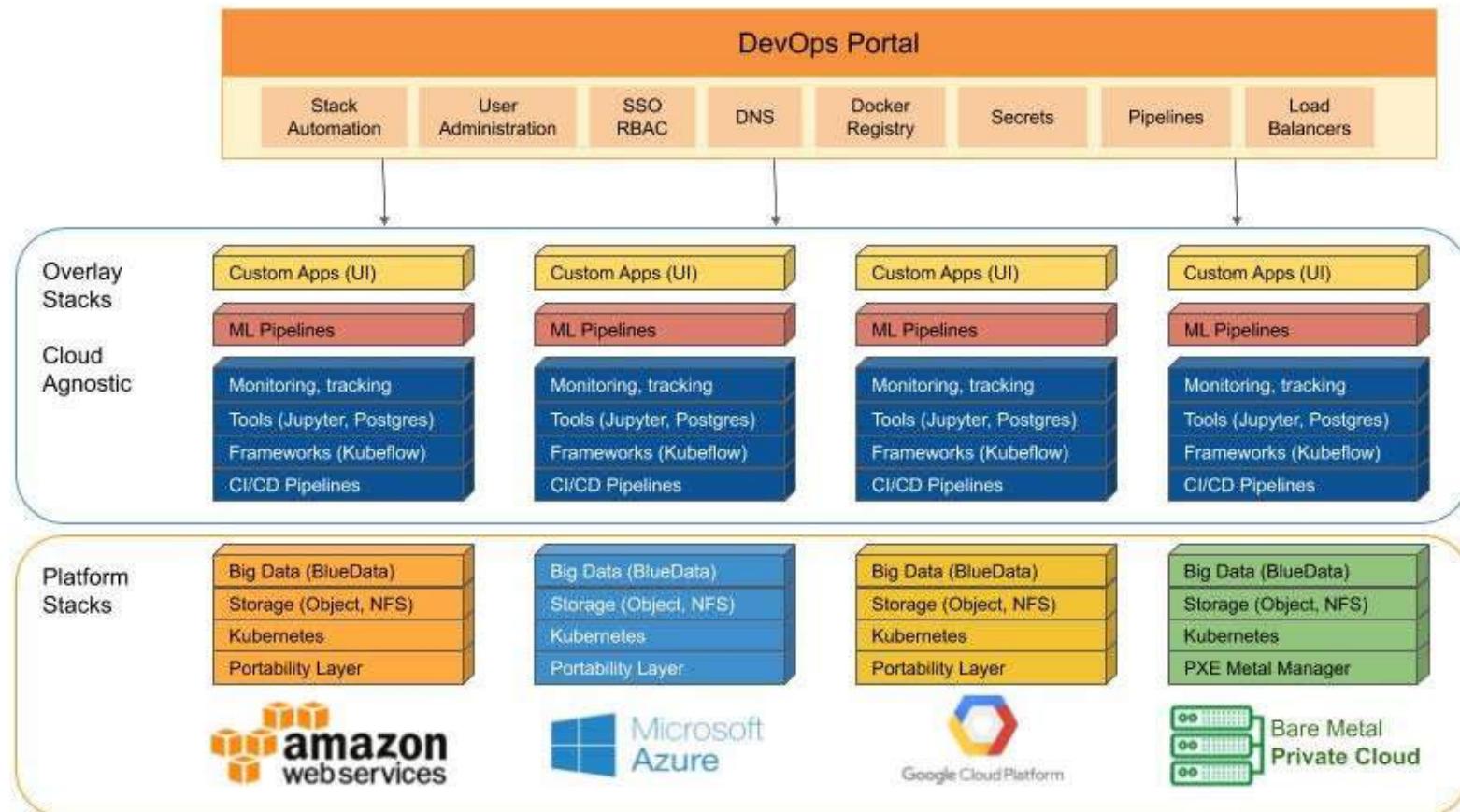
<https://www.instana.com/blog/the-new-application-delivery-organization/>



# New Application Delivery based on Platform Engineering



<https://www.instana.com/blog/the-new-application-delivery-organization/>





# Characteristic of Google Pay

Apart from

Enables Business Agility – easy to integrate payment into ecommerce apps & mobile payment app

Promotes interaction amongst partners –merchants and buyers

Rich User community – high adoption rate (Android users – 72.2% as of 2021)

Quick to start using and contains composable services and components

- Ranging from setup guides to tutorial to test & deploy (even with the test card suite)

Guides

Setup

Tutorial

Brand guidelines

UX best practices

Test and deploy

Integration checklist

Request production access

Deploy your application

Home > Products > Google Pay > Google Pay for Payments > Android

Rate and review  

## Test card suite

The new test card suite empowers Google Pay developers to run integration and API tests without the need to add a real credit card in their Google Account.

 **Note:** Developers must use real credit cards in a **PRODUCTION** environment. The test card suite is only intended for use in **TEST** environments.

This resource supports the following:

- Visa, Mastercard, Discover, and Amex card networks
- PAN\_ONLY and CRYPTOGRAM\_3DS authentication methods
- A single billing address in the US
- Shipping addresses in the US, Australia, Brazil, Canada, Hong Kong, Japan, Poland, Russia, and Singapore

<https://developers.google.com/pay/api/android/overview>

[https://static.googleusercontent.com/media/pay.google.com/en//about/business/static/data/gpay\\_worldpay-whitepaper.pdf](https://static.googleusercontent.com/media/pay.google.com/en//about/business/static/data/gpay_worldpay-whitepaper.pdf)



# Characteristic of Google Pay

## Secure and Compliant

- using ML to identify **phishing and fraud risks**
- using virtual account number for payment, instead of the actual credit card number
- Encrypts data exchanged during payment transaction
- Provide easy to use privacy control setting
- Comply to standard Payment Card Industry (PCI) compliance for processing payments
- as mandated by the Payment Card Industry Security Standards Council (PCI SSC)

## Stays relevant and data-Driven

A.I.-powered analysis of payments and spending over time



<https://www.thehindubusinessline.com/info-tech/google-pays-new-app-icon-is-meant-to-reflect-evolution-of-the-product-google-pay-exec/article33058587.ece>



# Benefits of building a Platform

Quicker response to opportunities

Service Model fit into value-chain and no longer sequential



Payment platform  
Between merchants and buyers  
through credit card



Payment platform  
Peer to peer payment through messaging  
Transfer \$ to anyone with paynow  
Top up \$ to Grabpay  
Manage expenses and savings  
Look for offers with promo codes

<https://joyofandroid.com/evolution-of-google-pay/>

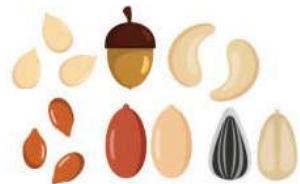
<https://blog.google/products/google-pay/reimagined-pay-save-manage-expenses-and-more/>

- Platforms are where **Users** create and exchange **Value** in the **Ecosystem**
- It is an infrastructure which enables **Interactions** between **Producers** and **Consumers** via information (**Seed**)

Think Roles not Markets



Producers



Platforms



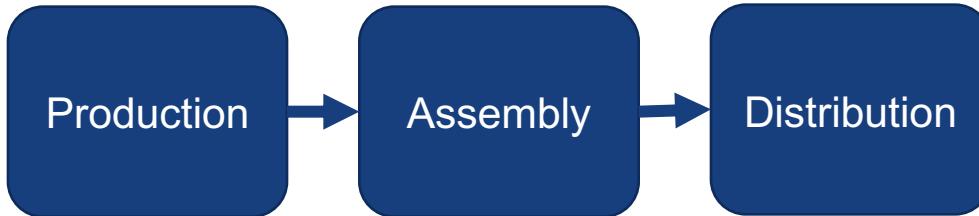
Consumers



Ref: <http://platformed.info>



# Platform Business – Linear to Network



Linear Business Model  
Focus is on Process



Network Business Model  
Focus is on Interactions



# Network Business Model - 1

- Start with one Seed
- Identify the Producers and Consumers
- Identify the Interaction with the Platform
  - Producer ↔ Platform ↔ Consumer
- Technology is only an enabler

## YouTube Example

Seed: ?  
Producer: ?  
Consumer: ?  
Interaction: ?

## The Three Main Components

### Magnet



### Toolbox



### APIs

### Matchmaking



### Data Analytics



# Network Business Model - 2

## Problem

- Accommodation
- Transportation
- Tele-communication
- News
- Computing

## Magnet

- Getting the two roles into the Platform
- Converting Consumers to Producers over time
  - ❖ Saves the marketing spend
- Getting the Producers to produce more seeds
  - ❖ Scaling the production



## ToolBox

- Tools to enable Interactions
- Tools for Producers to create
- Tools for Consumers to consume Value
  - ❖ Design and Build the right APIs
- Remove barriers to production and consumption
- Use Co-Design principle
  - ❖ Get external parties to contribute

## Matchmaker

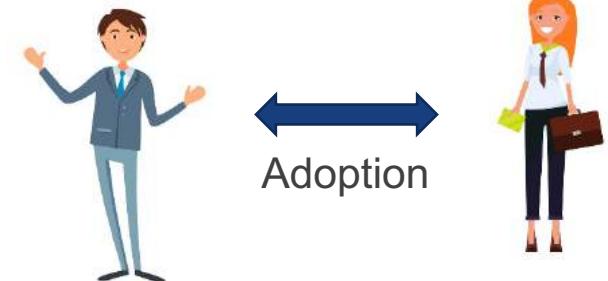
- Accurate matching
  - ❖ Minimize false-positives
- Curate content
  - ❖ Differentiate High Quality vs Low quality seeds
  - ❖ Use social media ranking e.g voting, rating etc.
- Build Trust mechanisms
- Use Machine Learning techniques/algorithms





# Scaling Platform Business - 1

## Linked-in Example



- Build one interaction at a time
  - ❖ Professional to Professional interactions
  - ❖ Enabled conversations interactions
  - ❖ Recruiters and Job Seekers
  - ❖ Follow Thought leaders



Discussion groups and posts



Monetization





# Scaling Platform Business - 2

- ❖ Who is responsible of moderating the content?

- ❖ Employees  
(in-house development)



- ❖ Algorithms  
(Automate with intelligence)



- ❖ Users



Example: Moderation  
of Platform Content



# Platforms – Enabler of Business Ecosystems - 1

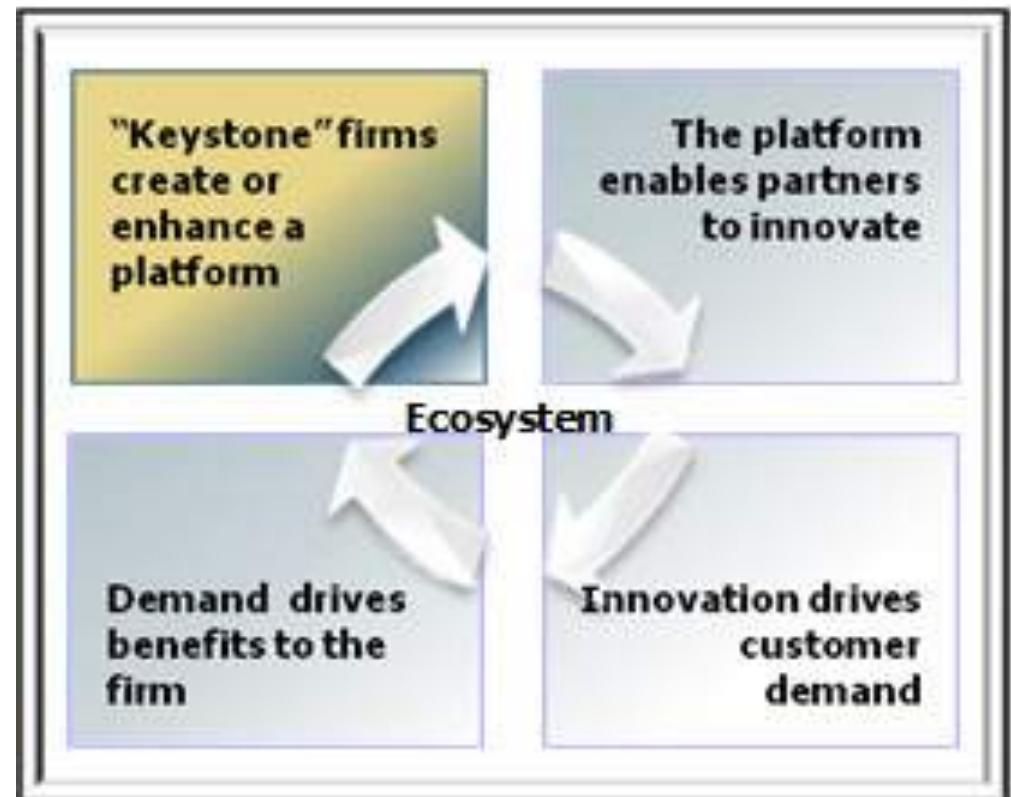
- Platform ecosystems are composed of externally produced complements that augment the capabilities of the platform.
- A software platform is an extensible software product or service that serves as a foundation on which independent outside parties can build complementary products or services that interoperate through the platform's interfaces.
- The collection of the platform and apps that interoperate with it represents the platform's ecosystem.



# Platforms – Enabler of Business Ecosystems - 2

A business ecosystem is the network of organizations

- including suppliers, distributors, customers, competitors, government agencies, and so on.
- involved in the delivery of a specific product or service through both competition and cooperation.



You may view this video for more:

<https://www.youtube.com/watch?v=qIHRiB3eZhM>

Platform Thinking - Lecture at MIT Media Labs by Sangeet Paul Choudary.mp4

Platform is not about building functionality or technology  
Platform is about Enabling Interactions



# Engineering perspective of Platform

Platform Aspects	Engineering Aspects
Seed	Domain entities
Producers and Consumers	Domain entities, Scalability, Performance
Interactions	Service Design, Architectural and Exposing Services, Business Process Scalability, Performance
Magnet	Service usability design, Design of Incentive components, Design and architecture of Data Analytics components
ToolBox	Service usability design, design of SDK, API design, Integration with other Services
Matchmaker	Curation Components, Algorithms
Scaling Platform Business	Design of Modular & Extensible components; Agile architecture and engineering
Platform Ecosystem	Architecting for Availability, Security, Scalability



# Great Agility -> Great Responsibility

- Can you tell me some benefits of a digital platform?
- Can you tell me an example of a platform (seed, producer, consumer)?
- What are the magnets, toolset, and matchmaker?



# Summary.

- Platforms provide composable components, services and infrastructure that can be used to build and host applications
- Platforms enable business ecosystems to be built
- They provide means for extensibility, innovation and co-creation
- They provide components to provide data analytics in order to improve the business
- Platform thinking is important for engineering a platform as a business



# Exercise: Platform thinking (20-30 minutes)

- Think of a new idea to create a platform
- You may use this idea for the Graduate certificate Practice Project

- Identify the following and submit your document describing the below in LumiNUS in your Team folder.

Name your file as 'Platform Thinking-Teamxx' where 'xx' is your team number.

- Seed (s)
- Producers and Consumers
- Interactions (producer <> platform <> consumer)
- **Magnet (attract and retain p and c)**
- ToolBox (web app, mobile app, APIs, integration etc)
- Matchmaker (what data collection and what analysis)



## 2. REUSABLE ASSETS AND FRAMEWORKS

Boon Kui HENG

[boonkui.heng@nus.edu.sg](mailto:boonkui.heng@nus.edu.sg)

Total Slides: 60

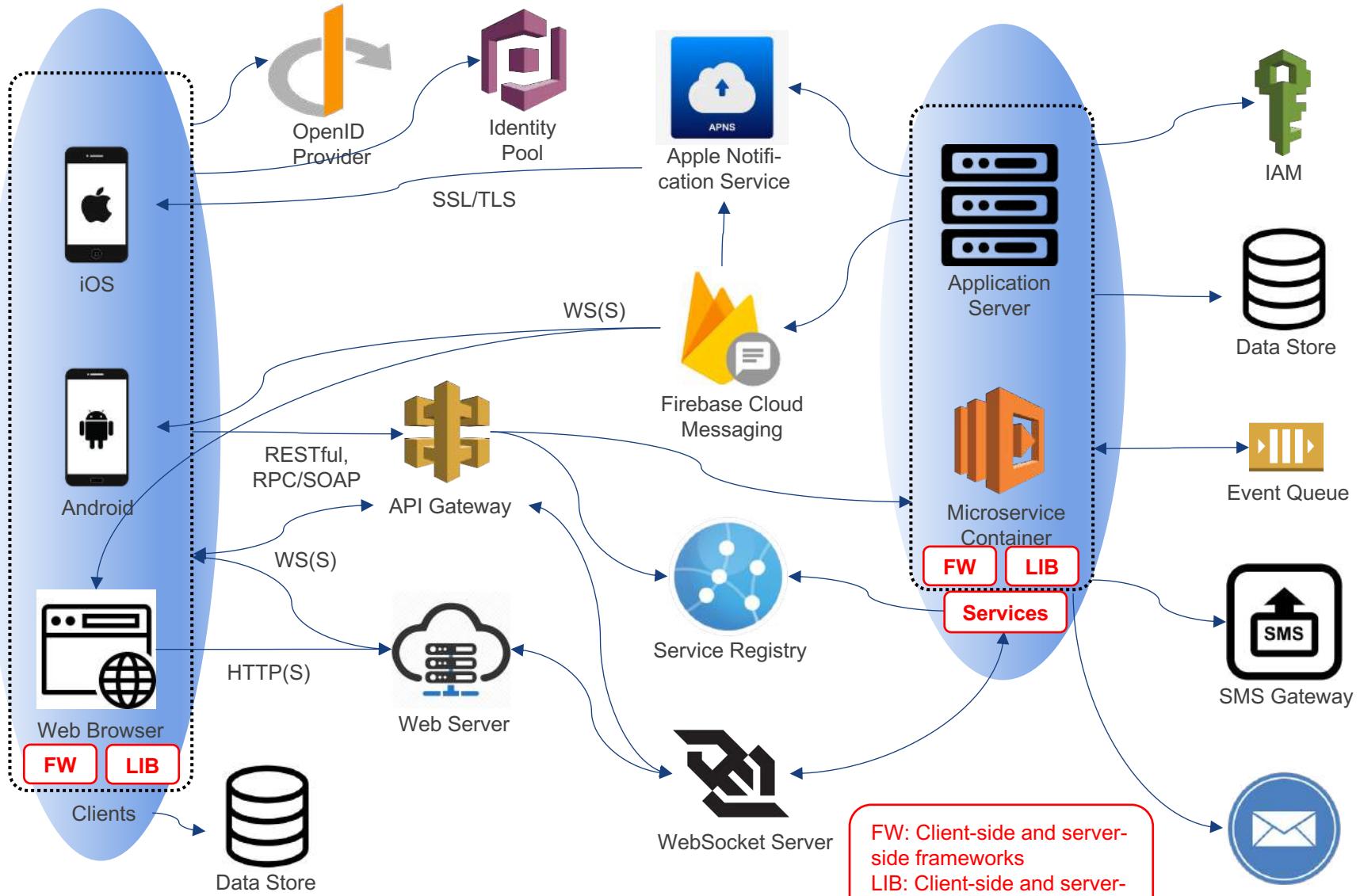
# Agenda

- Objectives
  - To design reusable components (assets) based on Domain Analysis
  - To design common services which can be used by applications
  - To architect frameworks on which applications can be built
- Topics
  - Reusable Assets in Platforms
  - Reusable Services
    - Workshop: Identify the reusable services for a given platform business
  - Reusable Frameworks
  - Shared Libraries

# Topics

- **Reusable Assets in Platforms**
- Reusable Services
  - Workshop: Identify the reusable services for a given platform business
- Reusable Frameworks
- Shared Libraries

# Reusable Assets in Platforms–1



Arrow: Direction of trigger in steady state, disregarding setup and teardown.

FW: Client-side and server-side frameworks  
LIB: Client-side and server-side shared libraries

# Reusable Assets in Platforms–2.

- Services
  - Reusable across **features or use cases**
  - e.g. the *Customer* service can be used by the *Purchase* and *Delivery* services
- Frameworks
  - **Reusable to build multiple clients and services**
  - e.g. the **Spring Boot framework** can be used to build the *Purchase* and *Delivery* services
- Shared Libraries
  - Reusable across multiple **clients and services**
  - e.g. the *Purchase* shared library is used by the *Purchase* client and service to share common entities
  - **Smaller scale** of reuse as compared to frameworks

# Topics

- Reusable Assets in Platforms
- **Reusable Services**
  - Workshop: Identify the reusable services for a given platform business
- Reusable Frameworks
- Shared Libraries

# Characteristics of Reusable Services

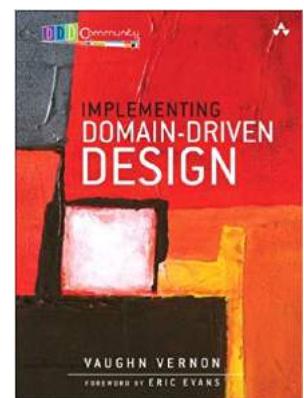
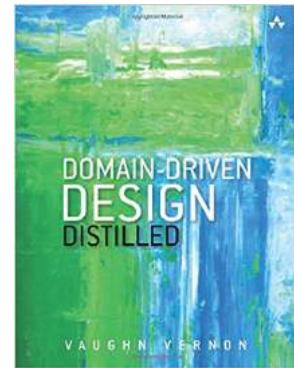
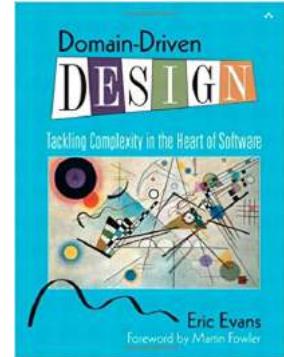
- Services are to be used for the **long haul**
  - It should be **maintainable, extensible and resilient**
- The functions of a service should be **stable** and **cohesive**
  - It should be single-minded → **Single Responsibility Principle**
- A service should be as **decoupled** from or **loosely coupled** with other services as possible
  - It could even be message-driven for the loosest coupling
- A service should be horizontally **scalable** without depending on the scalability of other services

# Identification of Reusable Services

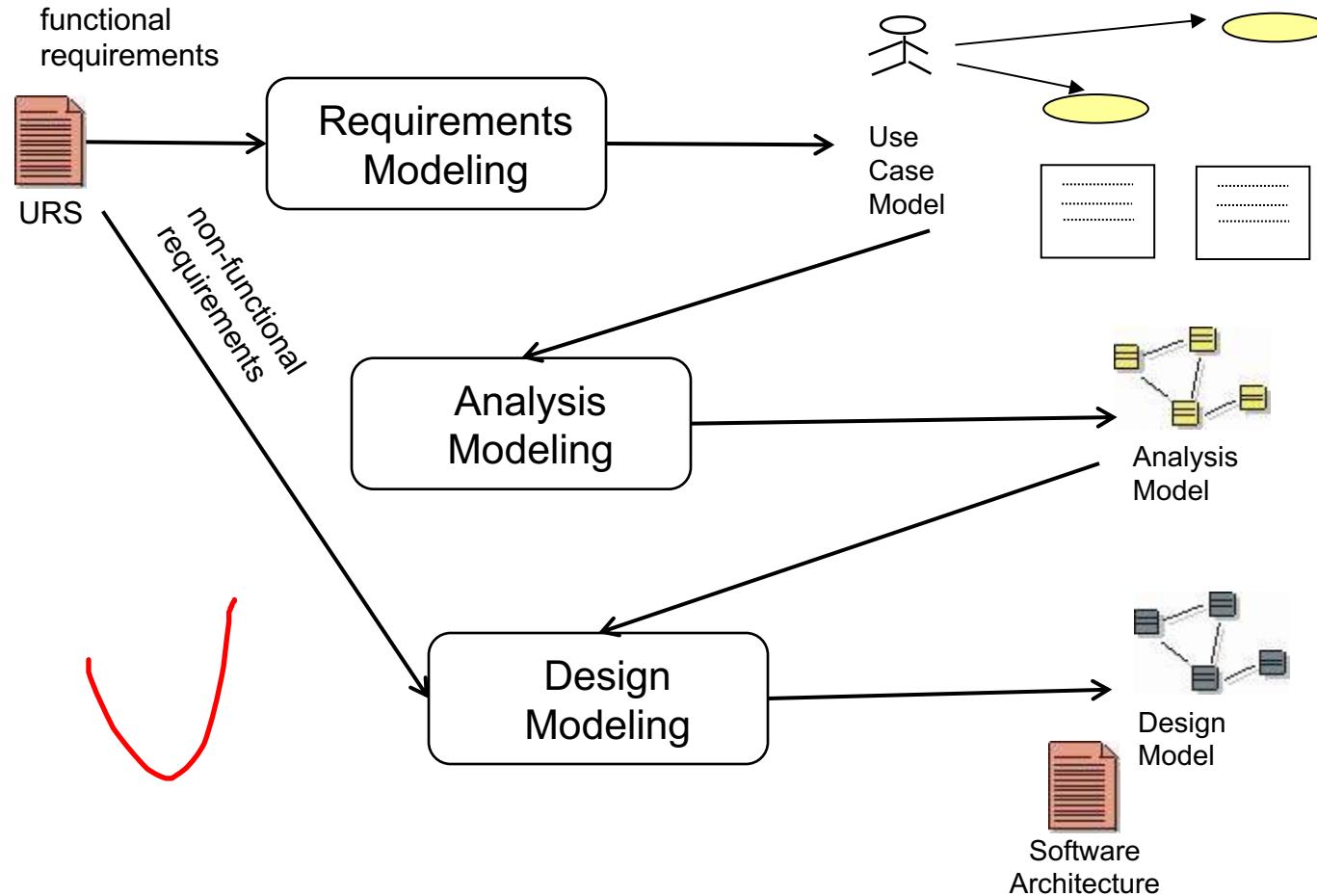
- Reusable services should be aligned with the **business requirements**
- The boundary of the services should facilitate the **desirable characteristics** of reusable services
  - maintainable, extensible, resilient, loosely coupled, scalable
- **Domain Driven Design (DDD)** provides effective techniques
  - to identify such kind of services
  - to **map the dependencies** among the services
  - to group the domain entities into right-sized clusters

# Domain-Driven Design (DDD)

- An approach to software development for **complex needs** by connecting the **implementation** to an **evolving model**. The premise of domain-driven design is the following:
  - placing the project's primary focus on the **core domain** and **domain logic**;
  - basing complex designs on a **model of the domain**;
  - initiating a creative **collaboration between technical and domain experts** to iteratively refine a conceptual model that addresses particular domain problems
- The term was coined by Eric Evans in his book of the same title



# vs. Object Oriented Analysis & Design–1



**Functional requirements** are structured as **use cases**. **Dependencies** among functional requirements are identified as **relationships** between use cases. The **operation flows** of each use case is described. **Domain objects** are optionally identified.

**Analysis objects** are identified along with their **state** (attributes) and **responsibilities** (operations) *without* considerations for implementation.

**Design strategies** are devised for the **operating environment** to fulfill the **non-functional requirements**. Analysis objects are adapted according to the design strategies to become **design objects** with full **class details**.

# vs. Object Oriented Analysis & Design–2.

Method	Requirements	Analysis	Design
OOAD	Use Case, <b>Domain Objects</b>	Boundary, Control & <b>Entity Objects</b>	Analysis Objects (Boundary, Control & <b>Entity Objects</b> ) + additional Design Objects + additional Framework Objects
DDD			<b>Domain Objects</b>

- DDD focuses squarely on **Domain Objects**, doing away with non-domain related concerns
- While OOAD caters to any architectural style, DDD is optimized for an architectural style ~~that is **service-oriented**~~ with **loosely coupled** and **cohesive** services



# and the IDEALS Principles

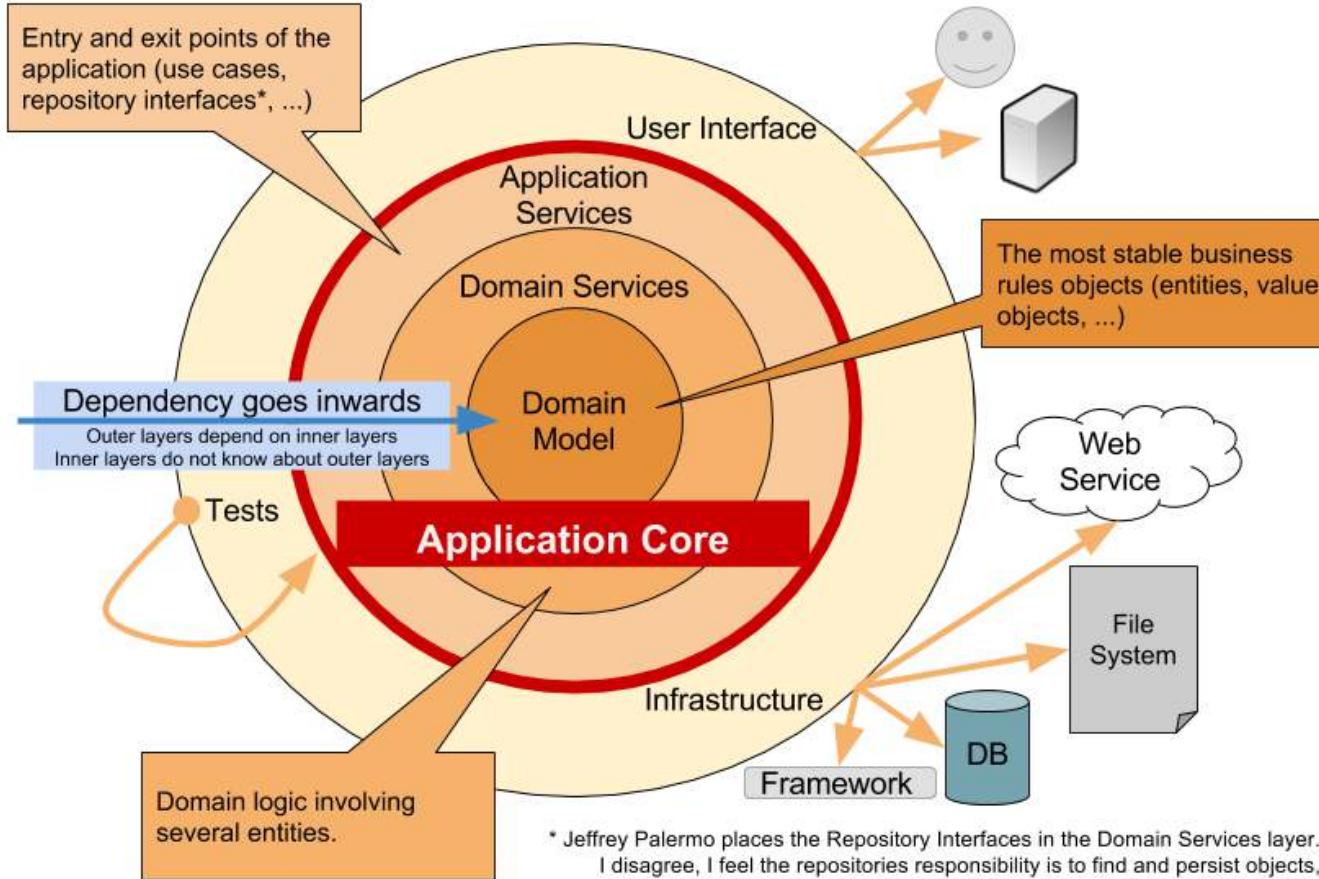
Principle	Goal
Interface Segregation	Each type of frontend sees the service contract that best suits its needs. E.g. Use an API gateway to perform message format transformation, message structure transformation, protocol bridging, message routing, etc.
Deployability	Automation of packaging and deploying software to an appropriate runtime topology. It has become critical due to dramatic increase in the number of deployment units.
Event-Driven	Event-driven microservices are loosely-coupled and are more likely to meet demanding scalability and performance requirements. Reliability also improves as the design copes with temporary outages of microservices, which later can catch up with processing the messages that got queued up.
Availability over Consistency	Prioritize availability over consistency by managing user expectation. Go for eventual consistency via data replication.
Loose-Coupling	Minimize the coupling among microservices for better maintainability.
Single Responsibility	Each microservice should be single-minded and comprising cohesive responsibilities.

→ DDD techniques produce domain-oriented design that complies with the IDEALS principles except for Deployability

→ This would be demonstrated in the later slides

→ Deployability can be achieved with DevOps

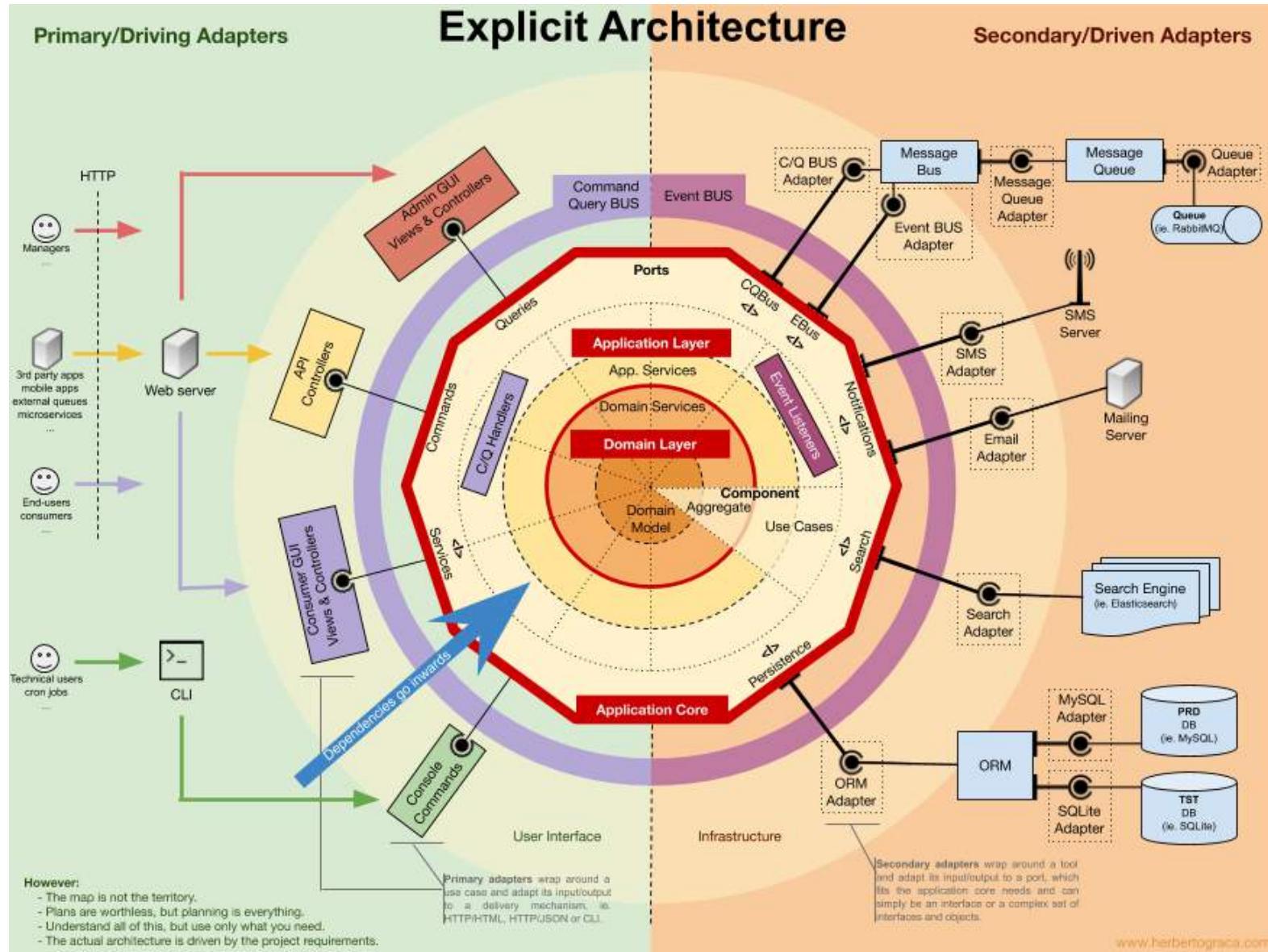
# and Onion Architecture–1



## Key tenets of **Onion Architecture**:

- The application is built around an independent object model
- Inner layers define interfaces. Outer layers implement interfaces
- Direction of coupling is toward the center
- All application core code can be compiled and run separate from infrastructure

DDD focuses on designing the **Domain Services** and **Domain Model** – the core of the Application Core.



*A very comprehensive treatment of DDD and Onion Architecture with Ports, Adapters, Commands, Queries, Events, etc.*

Credit: <https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-cqrs-how-i-put-it-all-together/>

Two levels of design conducted in DDD

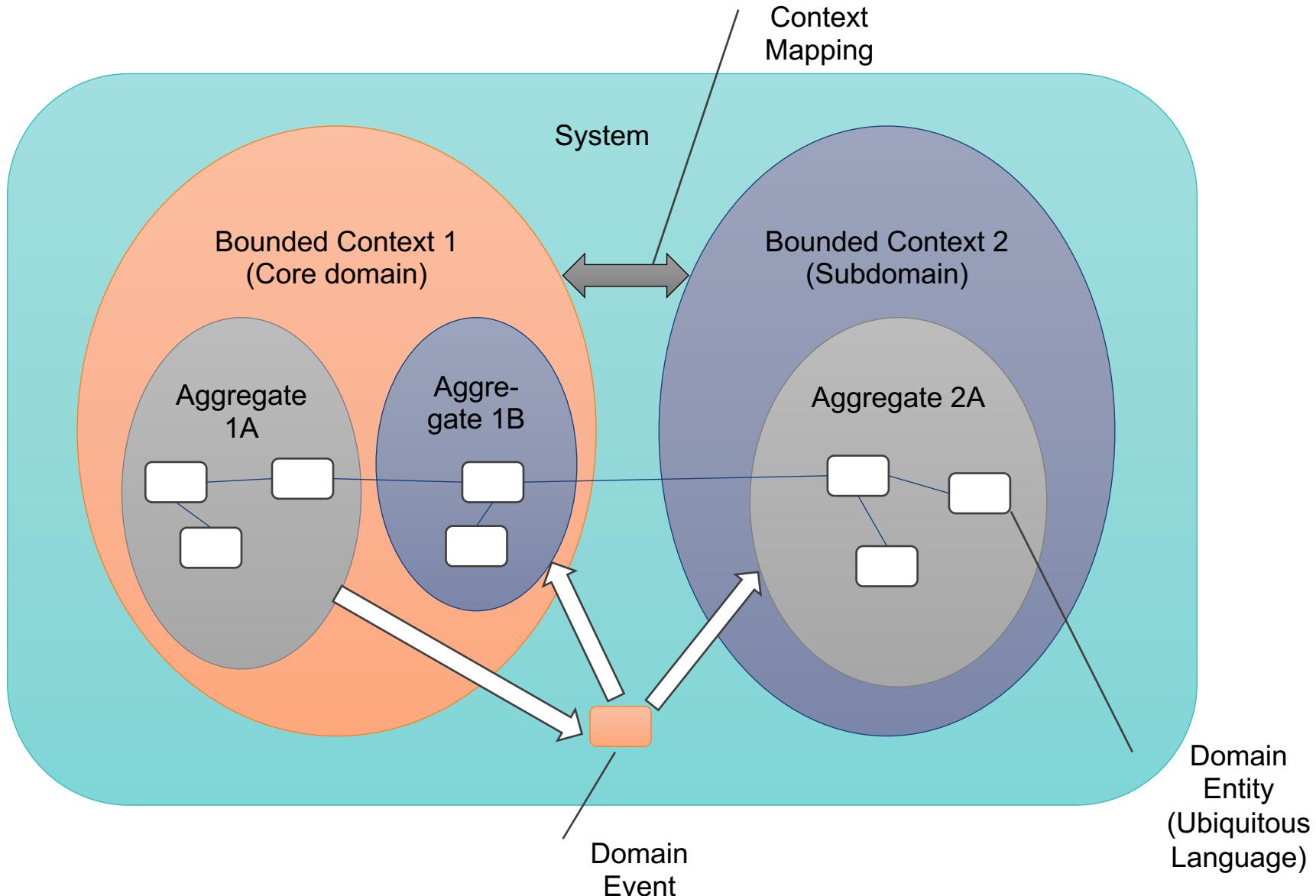
## Strategy Design

- Addressing **strategic concerns** that are **coarse-grained**
- Techniques / Tools
  - Bounded Contexts
    - Divide the domain into multiple **semantic contexts**
  - Ubiquitous Languages
    - Each **identifies the concepts** in a bounded context
  - Subdomains
    - **Core domain** is the bounded context that is **critical** to the business
    - **Subdomains** are the other bounded contexts that are **supporting** the business or **generic**
  - Context Mappings
    - **Integration** among the core domain and/or subdomains

## Tactical Design

- Addressing **tactical concerns** that are **fine-grained**
- Techniques / Tools
  - Aggregates
    - Collections composed of related **entities**
  - Domain Events
    - Records of **business-significant occurrence** in a bounded context

# Strategic Design and Tactical Design–2.

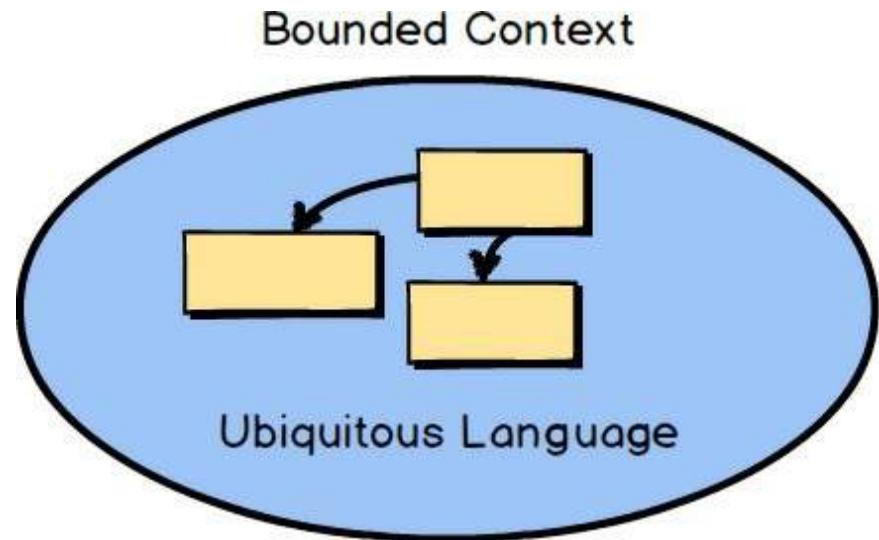


# Issues with Identifying Reusable Services

- How to **extract the domain entities** of a service from the business requirements? Who should be involved?
- How to **segregate the domain entities** into different services?
- How to **consistently name the concepts** within a service?  
How to **resolve name clashes** across services?
- Which **team(s)** should be assigned to work on a service?
- Which **source code repository** should be used for a service?
- Which **database** should be used by a service?  
**→** These issues can be effectively addressed via Bounded Contexts and Ubiquitous Language

# Bounded Contexts and Ubiquitous Language

- DDD is primarily about modelling a **Ubiquitous Language** in an **explicitly Bounded Context**
- Bounded Context (BC)
  - A **semantic contextual boundary**
  - Within the boundary, each **concept** of the software model has a **specific meaning** and does specific things
  - Transcends the **problem** and **solution** spaces
    - BCs are part of discussions and analyses in the problem space
    - BCs are where the solutions are implemented as well



- Ubiquitous Language (UL)
  - The language used among the **team members** and implemented in the **software model**
  - Rigorous – strict, exact, stringent and tight
    - ➔ Think of the official languages of the EU nations

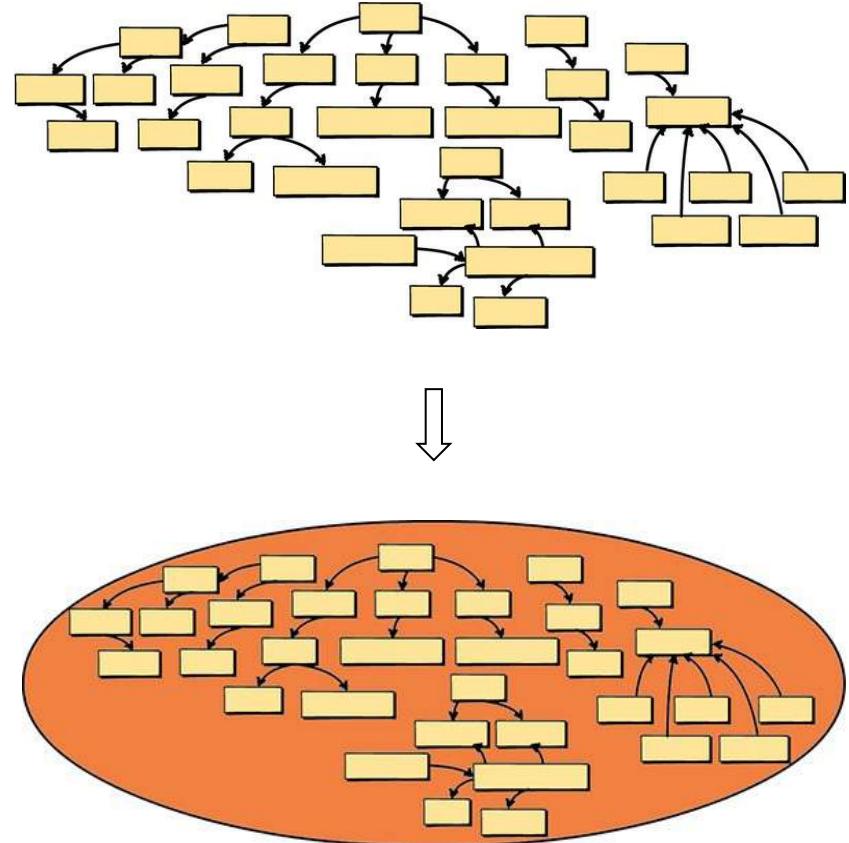
- **One BC** should be assigned to **only one team**
  - Although one team could work on multiple BCs
- A **separate source code repository** should be used for **each BC**
  - Source code, acceptance tests and unit tests should be kept together
- A **separate database schema** should be used for **each BC**
- The **official interfaces** of the BC are defined by **the team**

➔ The above scheme cleanly separates the responsibilities for a BC

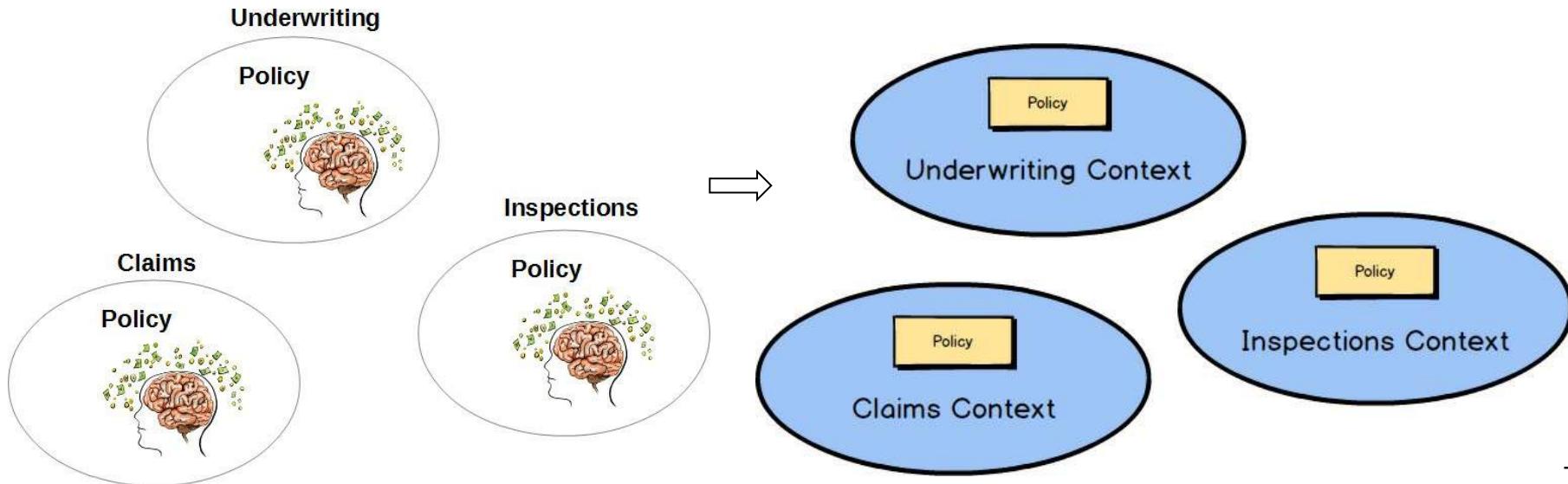
- ↳ Some organizations go further!  
E.g. Amazon's "You built it, you run it"

# Systems without Bounded Contexts

- A system may **start** with a **small set of concepts** in its domain model
- **More concepts** pile onto the **same domain model** as **more features** are added to the system
- Becoming a **Big Ball of Mud**
  - With multiple tangled models
  - Of blurred languages
  - Without explicit boundaries



# Systems with Bounded Contexts

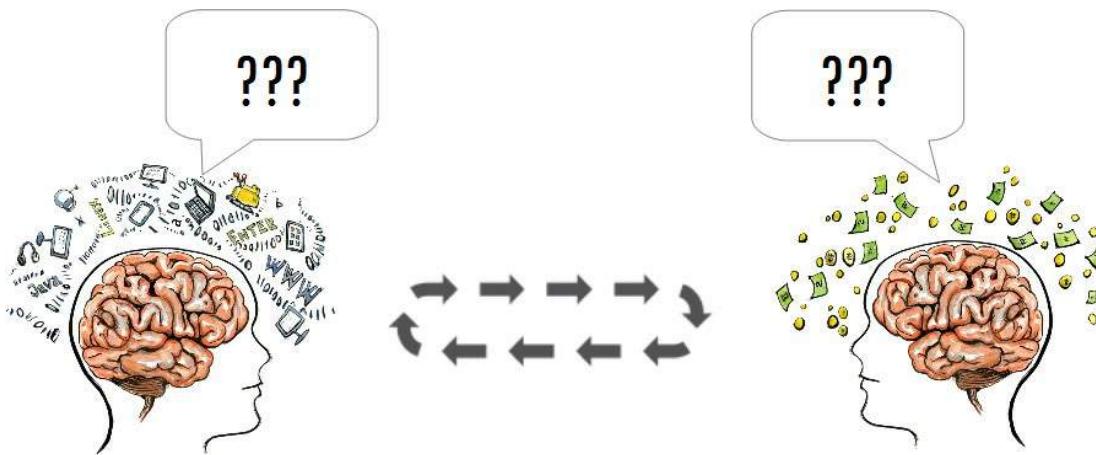


- The concepts of the **same name** may have **different meanings** for different business functions
- **Business** stakeholders and **technical** teams must **align** in their understanding of the concept in different contexts
- DDD embraces such differences by **segregating the concepts into different BCs**
- **Different languages** are used in **different BCs**

# Benefits of BC and UL

- All the concepts in a BC are **core** to the BC
  - Other **non-core** concepts are pushed out to **other** BCs
- The concepts in a BC are specified by **one UL**
- Multiple BCs prevent the design of **monolithic applications**, hence
  - Smaller set of test cases for a BC
  - And other similar benefits
- The boundaries of BCs can be the **boundaries for services, source code repositories** and **databases**
- **Teams** can be **assigned** on a per BC basis

# Identifying BCs for a System

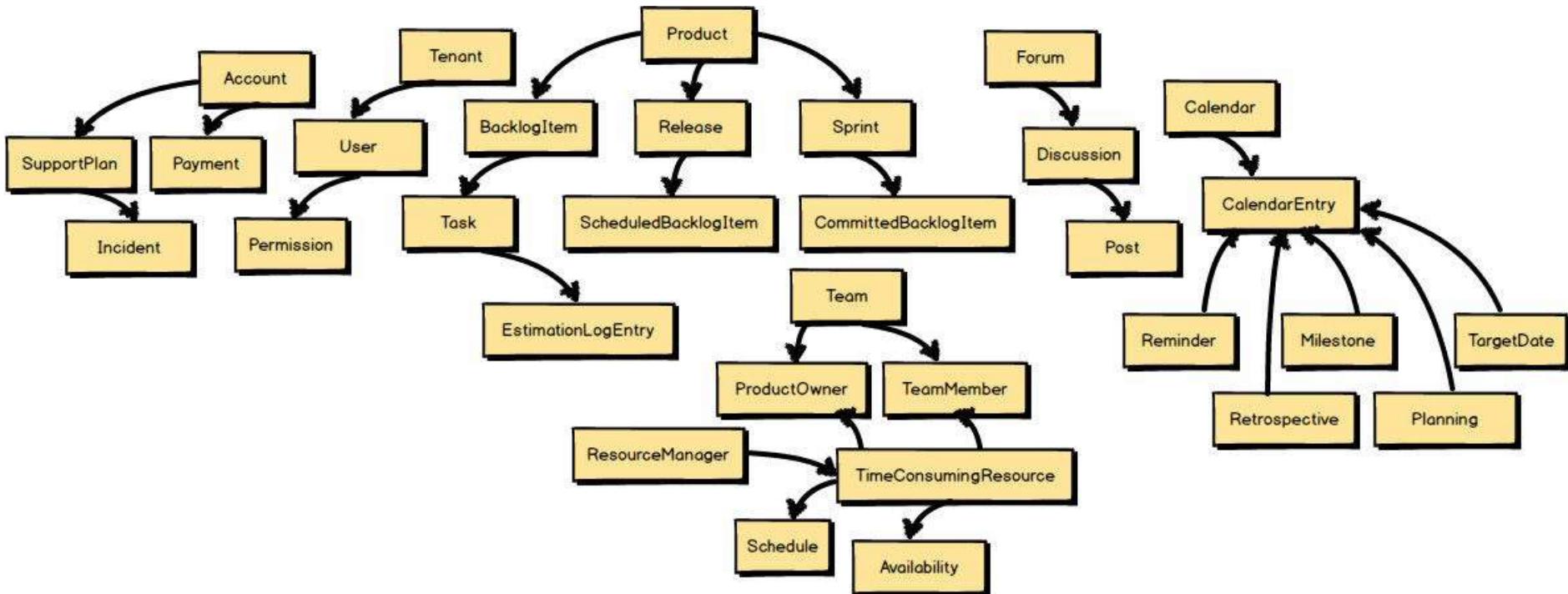


- Require **domain experts** and **software developers**
  - Domain experts focus on **business concerns**
    - Maybe different domain experts for different business functions
  - Software developers focus on **software development**
    - But should **not be consumed by technologies**, ignoring business aspects
    - Domain model is **more complex** than technical aspects
- Based on the **mental model** of domain experts, start to **extract concepts**
- Team **brainstorm** and **iterate** to produce a combined domain model
  - **Challenge** if a concept belongs to the UL

# Case Study: Scrum-based Agile PM Tool

- Support Scrum-based development teams
- Based on concepts like Products, Backlog Items, Releases, Sprints, Tasks, etc.
- Facilitate collaboration among team members via forums, discussions, shared calendars, etc.
- Support multiple tenants with different tiered support plans and the associated payments
- Account for human resource availability, utilization, etc.
- And possibly more features in the future...

# Domain Model could be a Big Ball of Mud!



Credit: Vernon, V. Domain-Driven Design Distilled

# Split into Multiple BCs

Let's try to split the Big Ball of Mud into multiple BCs!

- Identify the core domain with core concepts
- Place non-core concepts in subdomains
- Try to make a domain as reusable as possible

*This slide is intentionally left blank. It contains the sample BCs.*

# Developing the UL of a BC.

- Identify a set of **concrete scenarios** for a BC
  - Describing **how** the domain model should work
  - Including **nouns, verbs, adverbs**, etc.
  - Including **pre-conditions, post-conditions, constraints**, etc.
- Identify **concepts**
  - Likely to be **nouns**
  - Some nouns may represent **properties** of a concept
    - Some properties may be **grouped** as a concept
  - Must be **inside** the BC
- Can restate the scenarios as **executable specifications**
  - Serving as acceptance tests
- E.g. concrete scenario:  
*"The product owner commits a backlog item to a sprint. The backlog item may be committed only if it is already scheduled for release, and if a quorum of team members have approved commitment. If it is already committed to a different sprint, it must be uncommitted first. When the commitment completes, notify the sprint from which it was uncommitted and the sprint to which it is now committed."*
- E.g. executable scenario:  
*"The product owner commits a backlog item to a sprint.  
**Given** a backlog item that is scheduled for release  
**And** the product owner of the backlog item  
**And** a sprint for commitment  
**And** a quorum of team approval for commitment  
**When** the product owner commits the backlog item to the sprint  
**Then** the backlog item is committed to the sprint  
**And** the backlog item committed event is created"*

# Topics

- Reusable Assets in Platforms
- Reusable Services
  - **Workshop: Identify the reusable services for a given platform business**
- Reusable Frameworks
- Shared Libraries

# Workshop: Identify domain concepts, BCs and ULs

- **Maid2Order** is a platform that specializes in **connecting customers that require maid services** with the maid agencies that supply maid services. The platform provides the following features:
  - **Management** of agents, customers and maids
  - **Ordering** of maid services and loans
  - **Billing** of and payment for services and loans
- Given the set of concrete scenarios overleaf
  - Identify the **domain concepts** and their **relationships**
  - Identify the **BCs** for these concepts
  - Ensure that the concepts are consistently named based on their **ULs**



# Concrete Scenarios of Maid2Order-1

1. *The system admin registers a new maid agency that enrolls in the Maid2Order platform. While the agency is with the platform, the agent admin maintains the corporate particulars of the agency. The system automatically bills the agency monthly for its usage of the platform. When the agency leaves the platform, the system admin archives its record.*
2. *The system admin registers a new customer that enrolls in the Maid2Order platform. While the customer is with the platform, the customer maintains his/her personal particulars. When the customer leaves the platform, the system admin archives his/her record.*

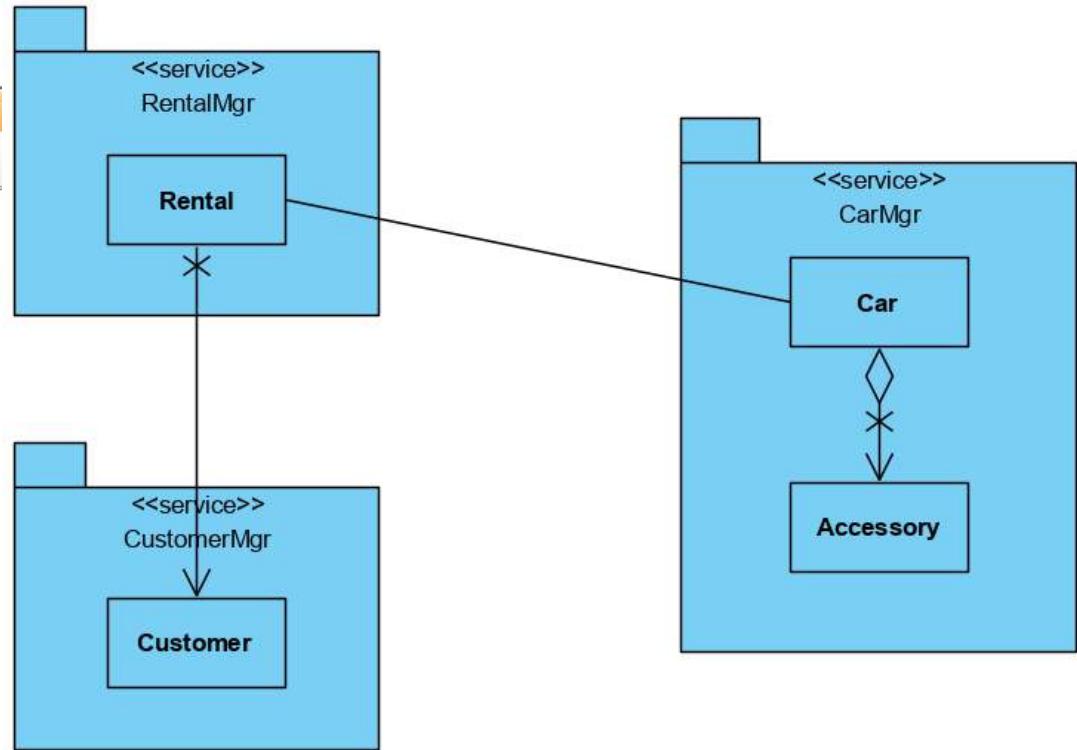
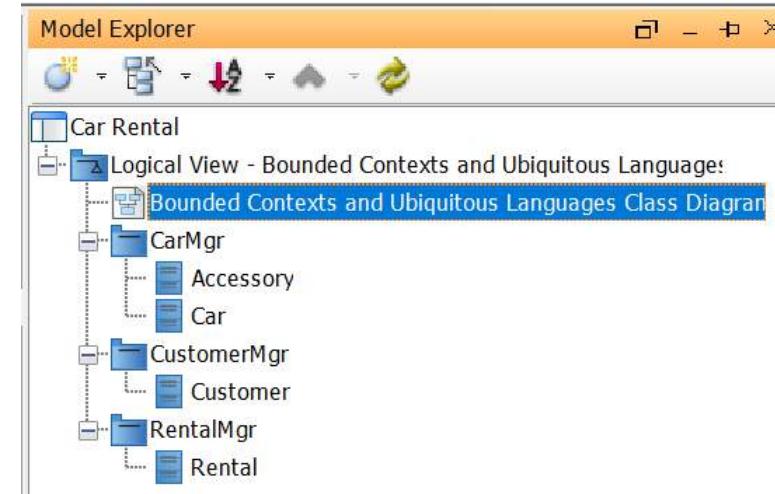
# Concrete Scenarios of Maid2Order–2

3. The agent admin registers a new maid that enrolls in the maid agency. While the maid is with the agency, the agent admin maintains the personal particulars of the maid. More importantly, the agent admin maintains the daily job schedule of the maid. When the maid leaves the agency, the agent admin archives her record.
4. The customer orders the service of a maid from a maid agency. While the customer may make a one-time ad-hoc order, it is normally more economical for the customer to subscribe to a usage plan. There are 3 plans at present with different preferential rates: OnceAFortnight, OnceAWeek, TwiceAWeek. The system automatically bills the customer for his/her usage of maid services; the frequency of billing depends on the types of service consumed.

# Concrete Scenarios of Maid2Order–3.

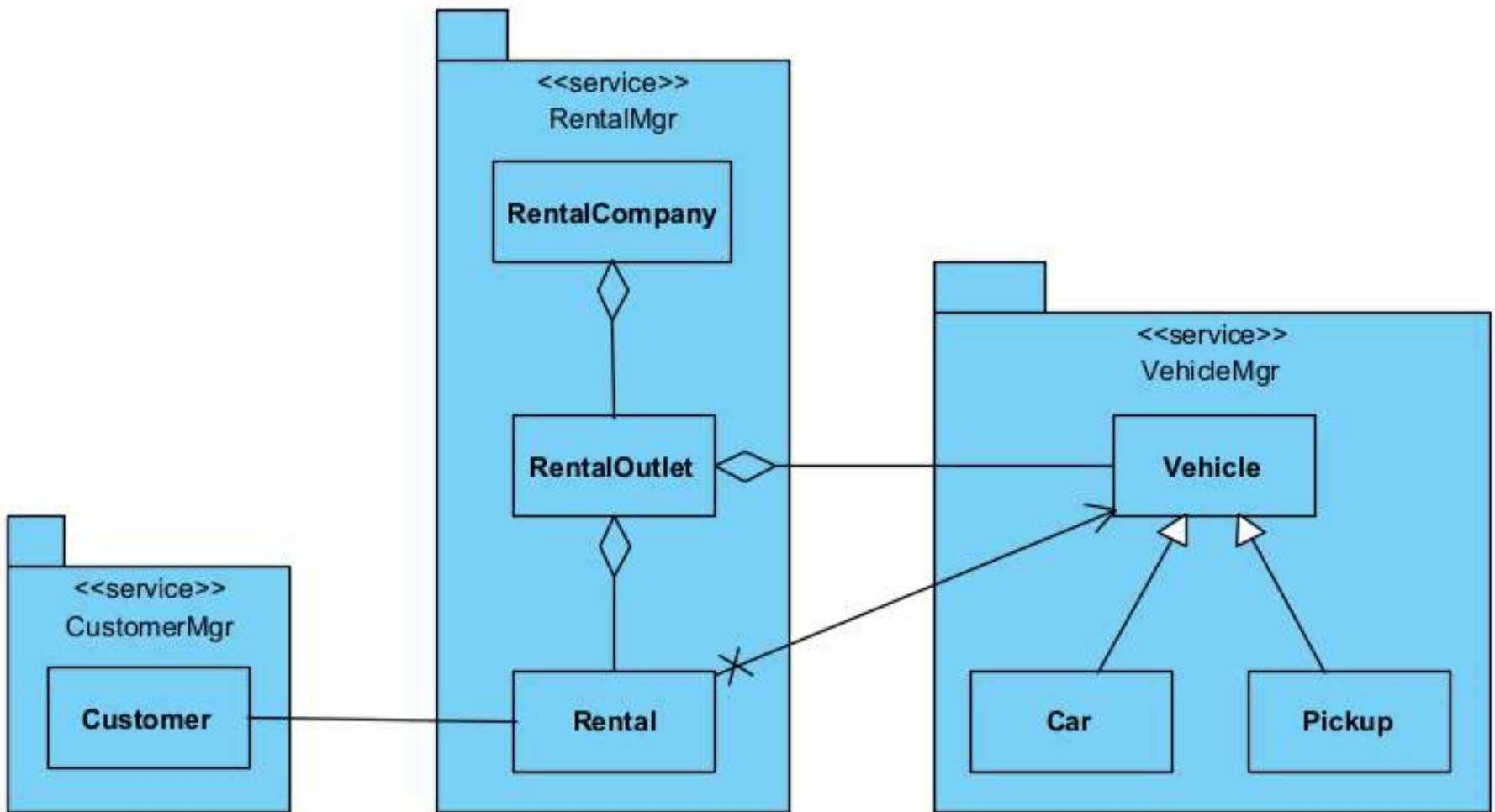
5. The agent admin of a maid agency loans the service of a maid from another maid agency. While the agent admin may make a one-time ad-hoc loan, it is normally more economical for the agent admin to subscribe to a loan plan. There are 2 plans at present with different preferential rates: Weekly, Monthly. The system automatically bills the agency for its usage of maid loans; the frequency of billing depends on the types of loan consumed.
6. The customer and agent admin make payment for their bills via the payment gateway of the system. The payment gateway supports electronic payment via direct bank transfer, credit card and PayNow. They may also choose to mail in a cheque. For an electronic payment, a receipt would be automatically generated. For a cheque payment, the system admin would create a receipt using the system.

# Simple Sample Diagram for Car Rental



Using Visual Paradigm

# Another Sample Diagram for Car Rental



Using Visual Paradigm

# Workshop: Sample Solution.



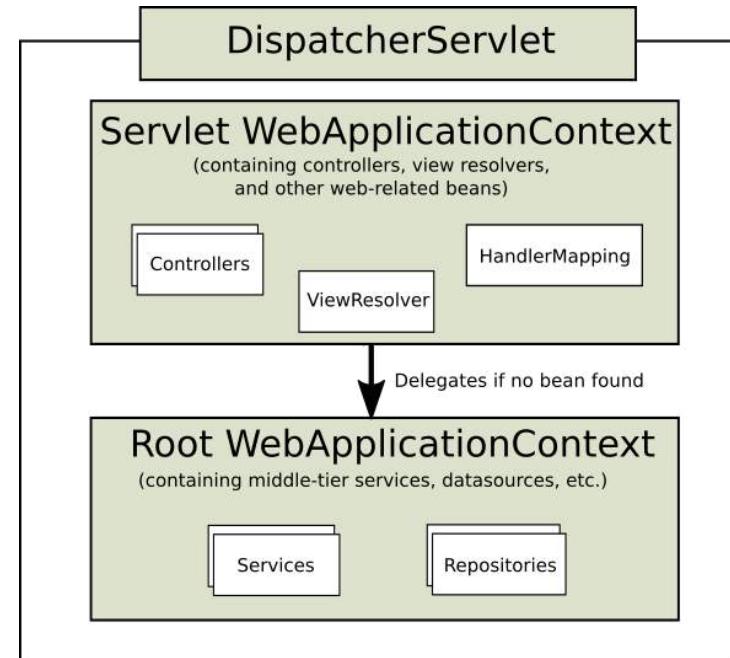
*This slide is intentionally left blank. It contains the sample solution.*

# Topics

- Reusable Assets in Platforms
- Reusable Services
  - Workshop: Identify the reusable services for a given platform business
- **Reusable Frameworks**
- Shared Libraries

# Characteristics of Reusable Frameworks

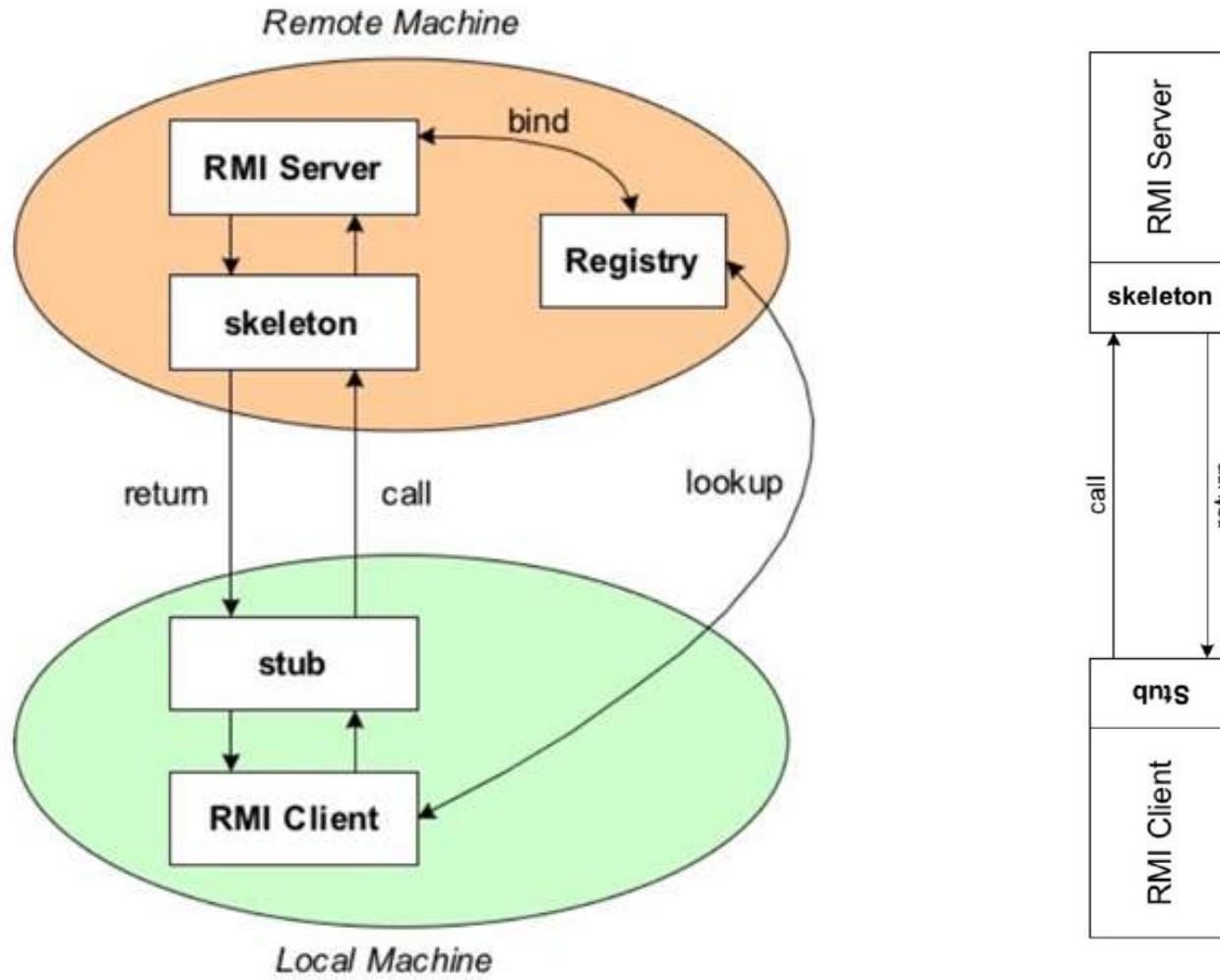
- A framework solves a specific **domain problem** using a set of communicating design patterns
  - E.g. User Interface domain, Data Access domain
- It provides **partial implementation** of domain architecture in a **specific environment**
  - E.g. Spring Web MVC is a framework for building web-based applications
  - **Not economically realistic** to build the classes from **scratch** for an application
- A framework is designed to be **reusable** across multiple systems
- A system may require **multiple frameworks** for its multiple layers
  - E.g. Spring Web MVC for the presentation and business layers
  - E.g. Spring Data for the data access layer



# Reusable Frameworks for Platforms

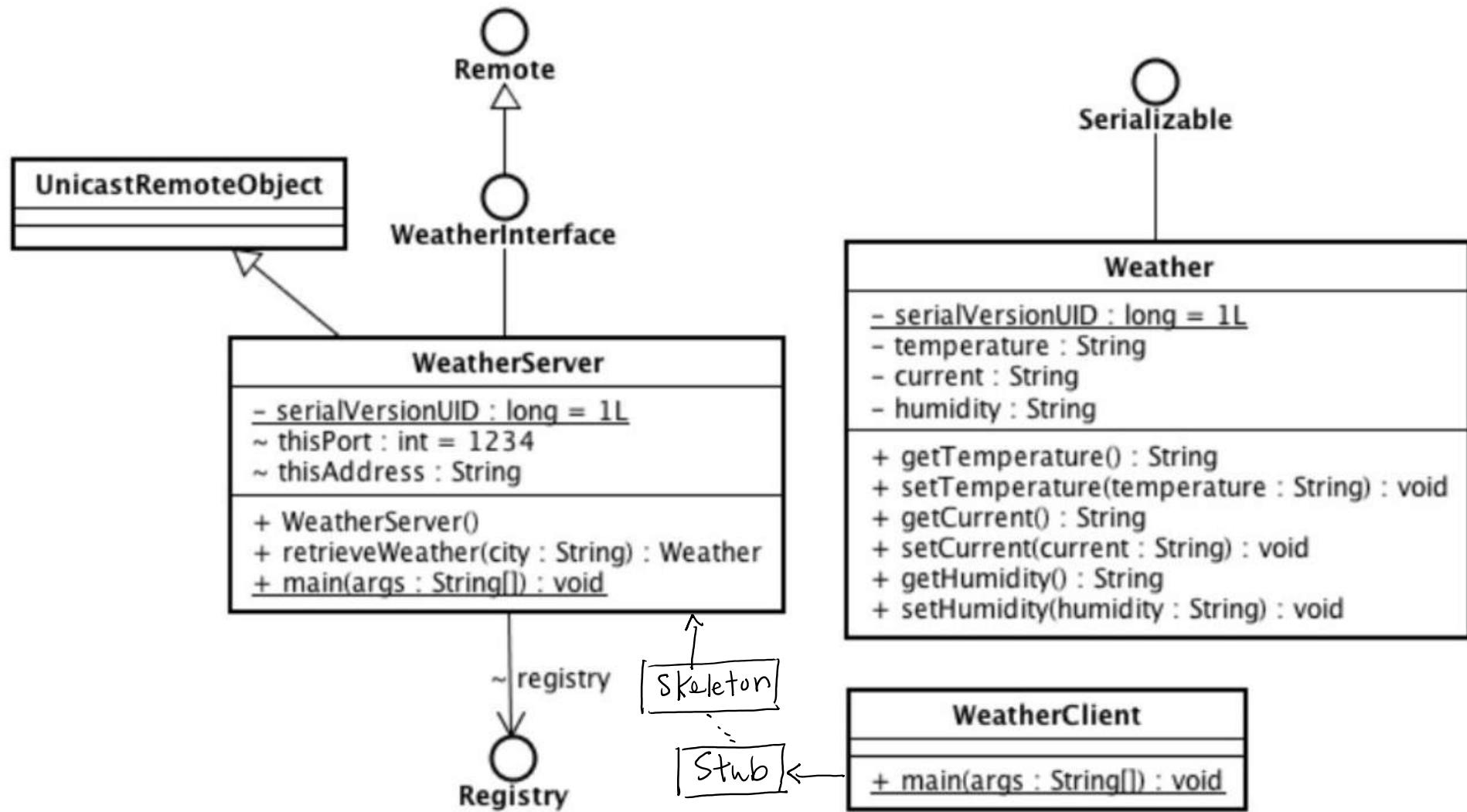
- A platform comprises **multiple constituent systems** that potentially span multiple enterprises and clients
  - Backend, legacy, partner services
  - Web, mobile clients
  - etc.
- Each constituent system may make use of multiple frameworks
- The platform provider should leverage on frameworks to **standardize and reuse**
  - **Common behaviours** shared among platform services and clients
  - **Common interactions** between platform services as well as between platform clients and services

# Example: Java RMI framework–1



# Example: Java RMI framework–2.

## Weather App



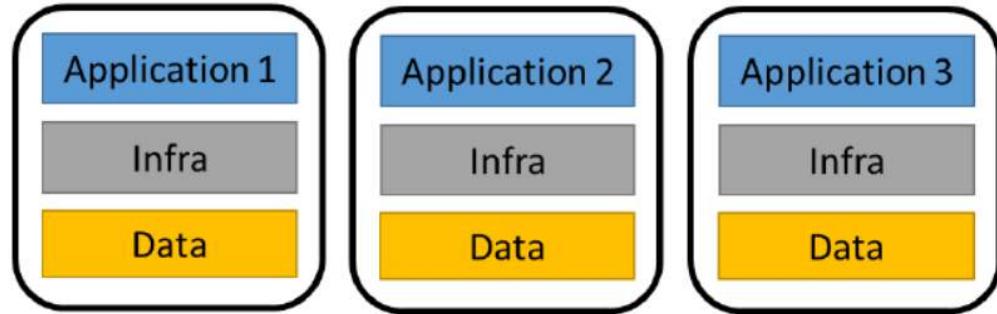
## Example: SGTS-1

- Problem: Government **agencies** develop their **own services and infrastructure** to meet the needs of the specific citizen or business group they serve
- Solution: Singapore Government Tech Stack (SGTS)
  - A **common platform** with a suite of **tools and services** hosted on a **common infrastructure**
  - Enable government agencies to build quickly and effectively **digital services** with more **secure, seamless, consistent and connected** user experience
  - Enable easy **exchange of data** across the government, **aggregating data collection** for richer insights that would help develop policy and monitor operations
- Example services: myinfo.gov.sg, mycareersfuture.sg

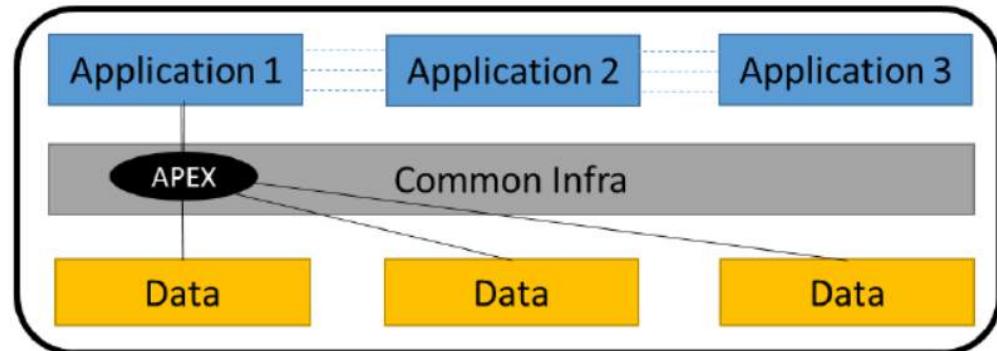
# Example: SGTS-2.



## Previous silo-model



## Using the Tech Stack



# Example: DBS Developers Platform

- An **API platform** with **200+ APIs** across more than **20 categories** such as funds transfer, rewards, PayLah! and real-time payments, relevant for companies from fintech to lifestyle, etc.
  - Are **consumer banking APIs** – which enables functionality for consumer banking accounts
- DBS does not share details on the design of its platform

**Instant redemption of reward points**

without the hassle of physical vouchers and emails.

**e.g: Homage**  
Redeem a free Care Assessment for aging loved ones to determine their home care needs and requirements.

**Funds transfer**

**e.g: soCash**  
All our customers can now skip the queues at DBS/POSB ATMs as they can get access to cash withdrawal points at all SPH Buzz kiosks and other convenience stores and newsstands at bus interchanges, bus stops and MRT stations.

**PayLah! your way to a good meal**

**e.g: McDonald's**  
Use PayLah! to pay for meals 24/7.

**Home planner**

**e.g: PropertyGuru**  
Hunting for a property doesn't need to be so daunting. House hunters can now get an instant affordability assessment from DBS through PropertyGuru as they are able to calculate if they can afford the house that they are viewing on propertyguru.com.

Credit: [www.dbs.com/newsroom/Reimagining\\_banking\\_DBs\\_launches\\_worlds\\_largest\\_banking\\_API\\_developer\\_platform / www.dbs.com/dbsdevelopers/index.html](http://www.dbs.com/newsroom/Reimagining_banking_DBs_launches_worlds_largest_banking_API_developer_platform / www.dbs.com/dbsdevelopers/index.html)

# Qualities of Well-Designed Frameworks–1

- Simple
  - Do **not sacrifice simplicity** to schedule pressure, feature creep, etc.
  - Do not admit **half-baked solutions**; “You can always add; you cannot ever remove”
  - Test if an API is simple by “**client-first programming**”
- Expensive to Design
  - Good framework design consumes **time** and **resources**
  - A **distinct task** separate from application development, staffed by **dedicated framework designers**
- Full of Trade-Offs
  - **No** such thing as **perfect design**
  - Design decisions made by **trading off various options**, based on their benefits and drawbacks

# Qualities of Well-Designed Frameworks–2

- Borrow from the Past
  - Borrow from and build on top of existing **proven designs** that withstood the **test of time**
    - E.g. architectural and design patterns
    - E.g. experiences from designing frameworks
  - **Novel solutions** may be desirable but should be treated with utmost **caution**
    - May not want to innovate in library design, do so in application functionality
- Designed to Evolve
  - Take into account (but not overdo) likely **future evolution** of the framework
  - Prevent **degradation** (decay) over time and preserve **backward compatibility**

# Qualities of Well-Designed Frameworks–3.

- Integrated
  - Play well with an **ecosystem** of **different** development tools, programming languages, application models, etc.
- Consistent
  - Consistency in the **designs** of the **various parts** of framework promotes **productivity**
  - Facilitate **transfer of knowledge** between parts of the framework
  - Facilitate identification of the **design** that is truly **unique to a part** of framework
    - That is, if a part of framework is inconsistent in design, it is more likely for the deviation to be due to some unique consideration
- Testable
  - **Applications** built using a framework should be **testable**
  - Desirable to **build framework** in a **test-driven manner**

- Do design frameworks that are both **powerful** and **easy to use**
  - *Simple things should be simple and complex things should be possible!*
    - Make implementing **simple scenarios** easy
    - Does not prohibit implementing **more advanced scenarios**
  - 80/20 rule: Concentrate effort on the **important 20%** of scenarios and APIs
- Do understand and explicitly design for a broad range of developers with **different programming styles, requirements and skill levels**
- Do understand and design for the broad variety of **programming languages**
  - E.g. Some languages use **dynamically typed types** may have problems with a strongly typed API

- The Principle of **Scenario-Driven Design**
  - Design framework around a set of **common usage scenarios**, based on **sample client codes**
    - In a **test-driven** manner
    - **Not** starting with the **object model**
    - Focusing on the **usability** of the APIs
- The Principle of **Low Barrier of Entry**
  - An API offers a low barrier to entry for **non-expert users** through **ease of experimentation**
    - Allow **easy identification** of the right set of **types** and **operations**
    - Allow a developer to **easily experiment** with it without extensive initial setup
    - Allow for **easy finding** and **fixing of mistakes** caused by incorrect usage of an API

- The Principle of **Self-Documenting Object Models**
  - Design APIs that do **not** require developers to **read documentation** for simpler usage scenarios
    - Care must be taken when choosing names, types, designing exceptions, etc.
- The Principle of **Layered Architecture**
  - **Layered design** makes it possible to provide both **power and ease of use** in a **single framework**
    - **Lower layer** APIs with **low-level types** that expose all the richness and power
    - **Higher layer** APIs **wrap the lower layer** with more convenient APIs
    - Cannot be achieved by **single-layer** APIs

# Process of Framework Design–1

- Like the design of most reusable software
- Start with analysing the domain to collect **application examples**
- Design and develop **initial whitebox** framework to implement the application examples
- Use the framework to build **other applications**
  - Identifying **weak points** and leading to **improvements** in the framework
  - Usually making the framework **more blackbox**
- **Release** the framework when the suggestions for improvement become rare

Two process models for framework design

## Framework designers also design applications

- FW designers **alternate** between phases
  - to extend the framework by **building applications** on it
  - to **revise the framework** by consolidating desirable extensions
- Ensure FW designers **understand the problems** with their framework
- Suitable for **small groups** that can **budget** for effort on framework revision

## Framework designers does not design applications

- FW designers **design and test** frameworks
  - App designers extend the framework by **building applications** and **provide feedbacks** to the FW designers
  - FW designers **revise the framework** based on the feedbacks
- Ensure FW designers are having **sufficient time** to revise the framework
- Suitable for **larger groups**, in particular those targeting **external users**
- This model is **more popular**

# Pitfalls of Framework Design.

- **Immature domain** of framework results in mistakes in understanding, hence more iterations
- **Unstable interfaces** of a framework implicate application classes, leading to more iterations
  - **Unstable behaviours** in framework components are more readily revised
- **Insufficiently representative original application examples** may result in a framework unable to support a specific valid application
- **Prematurely publishing** a framework while its design is changing
  - Do so only when it has **proven** itself
  - Should validate it via some **small pilot projects**

# Topics

- Reusable Assets in Platforms
- Reusable Services
  - Workshop: Identify the reusable services for a given platform business
- Reusable Frameworks
- **Shared Libraries**

# Characteristics of Shared Libraries

- Classes that are shared among platform elements
  - Among **services**
  - Among **clients**
  - Across **clients and services**
- Used for sharing the following across platform elements:
  - **State specification** of a class
    - Via the attributes of a super class
  - **Behavioural specification** of an interface or a class
    - Via the abstract operations of a super class or an interface
  - **Behavioural implementation** of a class
    - Via the implementation of the operations of a super class
- Are also essential elements used to **build reusable frameworks**
- May be provided by **platform providers** or **third parties**

# Drawbacks of Shared Libraries

- Loss of true **technology heterogeneity**
  - Programming language-specific
  - Operating platform-specific
- Ease of **scaling** parts of system **independently** from each other is curtailed
  - If the parts are tied by some construct (e.g. static variable, mutex) used in the library
- Cannot deploy a new library without **redeploying the entire process**
  - Less an issue for dynamically linked libraries
- Lack of obvious **seams** around which to erect **architectural safety** measures to ensure system resiliency
- Shared code used to **communicate** between clients/services is a point of **coupling**
  - **Brittleness** in RPC implementations like Java RMI
  - A **change in the RPC endpoint** frequently requires **new stubs** to be generated, unnecessarily **impacting the RPC clients** that do not require the change

- **DRY:** don't repeat yourself
  - Avoid duplicating system **behavior** and **knowledge**
- Deceptively **dangerous** for reuse **across services**
  - Shared code among services results in **coupling**
  - E.g. A library of common domain objects shared by all the services. A change in the library would impact all services.
- One preventive measure is **not to share** code across the **service boundary**
- A general rule of thumb: **don't** violate DRY **within a service**, but be **relaxed** about violating DRY **across services**

# Pitfalls of Client Libraries

- Client libraries make it **easy** for clients to **consume services**, and **avoid duplicating code** across the clients
- Possible for **server logic** to **creep** into client library
  - Especially likely if the **same team** works on client library and server
  - **Cohesion** of server starts to break down
  - Need to change **multiple clients** to roll out **fixes for server**
- **Limited technology choices** for clients if the use of client library is mandated

# Client Library Examples: AWS, Netflix

## AWS

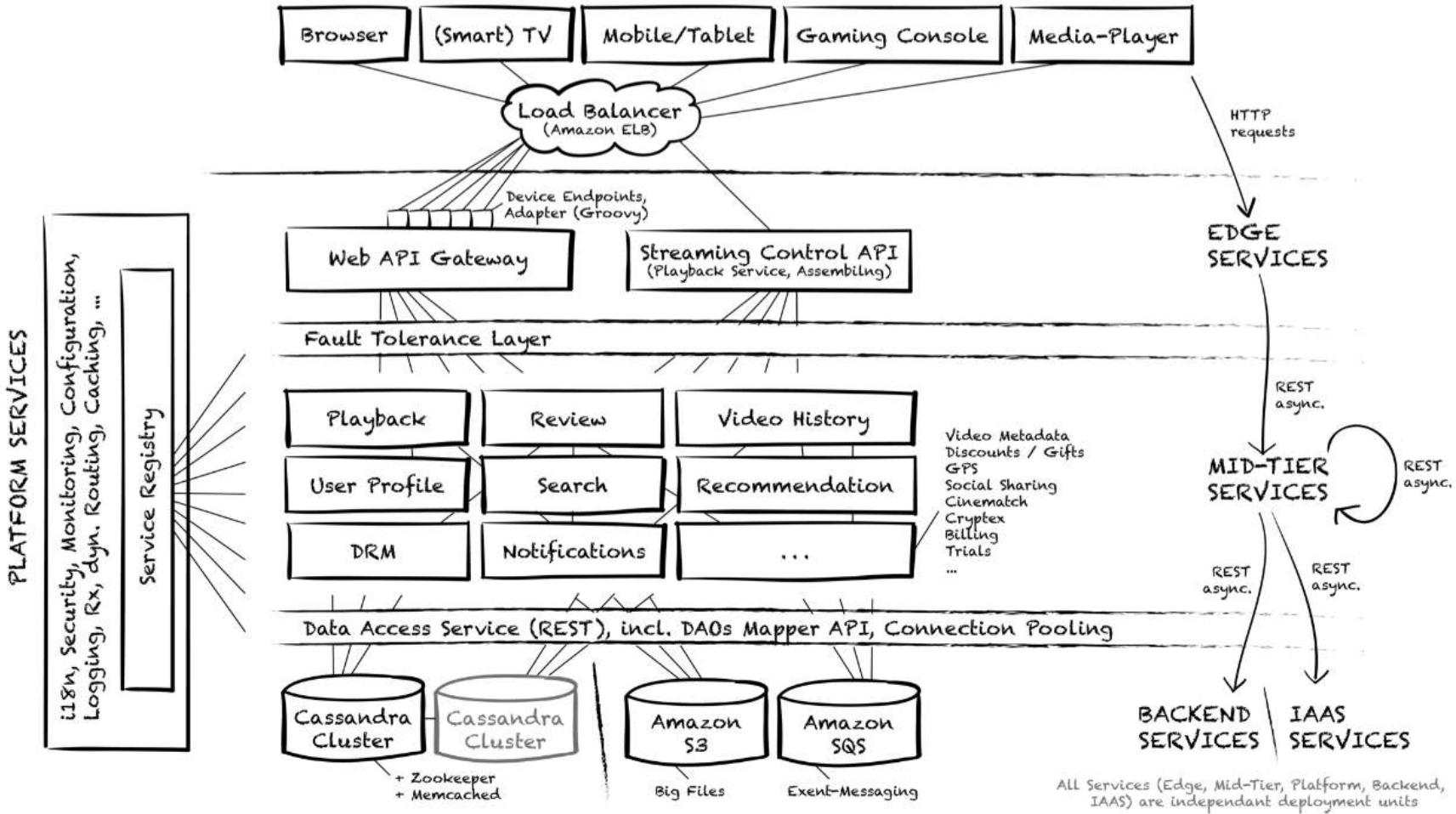
- AWS can be invoked via **SOAP/REST** web services
- Various SDKs are written by **community** or AWS team which does **not work** on the APIs
  - Seems to result in sufficient separation between client library and APIs
- **Clients decide** when to **upgrade** the client libraries

## Netflix

- While avoiding **code duplication**
- Ensuring **reliability** and **scalability** of system by handling
  - Service discovery
  - Failure modes
  - Logging
  - etc.
- Easy to get **clients up and running** and ensuring **system behaves well**
- But the degree of **coupling** between clients and server is worrisome

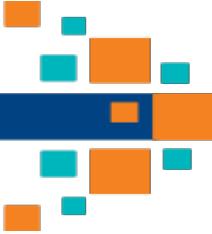
# Example: Architectural Overview of Netflix

## Netflix Architectural Overview (simplified)



# Guidelines for Client Libraries.

- **Separate out** client code to handle the underlying **transport protocol** from things related to the **destination service** itself
- May allow clients using **different technology stacks** to make calls to the **underlying APIs**
- Clients are in charge of **when to upgrade** their client libraries
  - Need to ensure the ability to release services independently of each other!



---

## 3. API DESIGN

Darryl NG

darryl.ng@nus.edu.sg

Total Slides: 91

- Objectives
  - To be able to design APIs in an effective way
  - To be able to apply best practices and patterns for designing APIs
  - To be able to perform API versioning
- Topics
  - API standards
  - Best practices for API design
  - API versioning
  - Securing APIs



# WHAT IS AN API?

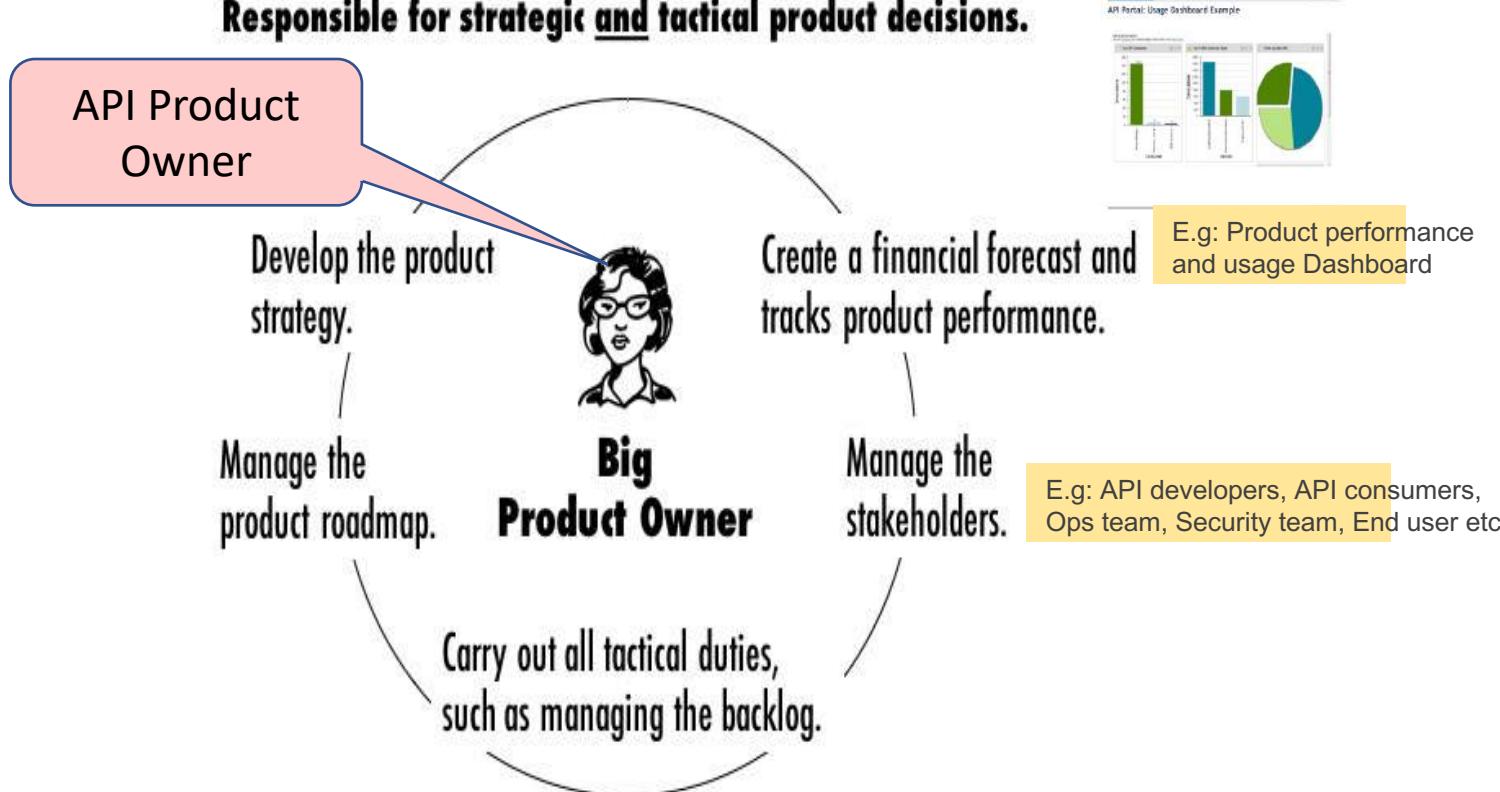
---

API stands for Application Programming Interface.

APIs let software applications communicate with other websites and applications for additional functionality.



# API Product Mindset



*"APIs should be managed like a **Product** and not Project."*

<https://www.romanpichler.com/blog/the-product-owner-responsibilities/>

ATA/SPL/APIDesign.pptx V2.0 © 2019-23 National University of Singapore. All Rights Reserved  
<https://cloud.google.com/files/apigee/apigee-api-product-mindset-ebook.pdf>

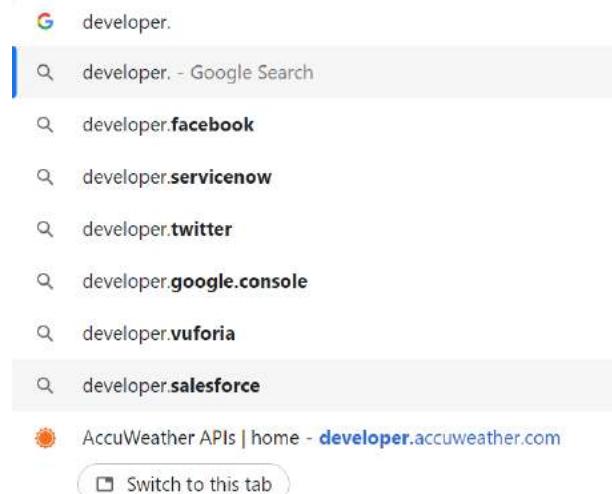
## Contributions to revenue

- Salesforce - 50%
  - has a marketplace (AppExchange) for apps created by its partners (~300) that work on its platform
- eBay - 60%
  - list its auctions on other websites, get bidder information about sold items, collect feedback on transactions, and list new items for sale
- Expedia – 90%
  - allow people using third-party websites to tap its functionality in order to book flights, cars, and hotels



# API Product Mindset - Pillars

- **Outside-in API strategy:**
  - Considers the needs and preferences of the consumers of the APIs (developers) and end-users
    - As opposed to designing the APIs based on exposing existing resources and functionality
  - Data-driven – usage and customer analytics
    - How often are the APIs called? What kind of calls? Are developers changing behaviour
  - Reach out to developers through conference, forum, etc
  - Success is measured by its adoption and business impact
  - Easy to discover
- **Time to Market**
  - Use MVP strategy to launch APIs
- **Mature APIs through iterations**
  - Improve APIs and update backlog
  - grooming through feedback and iterations





# API Standards - Documentation

## OpenAPI Specification (OAS)



- Created by a consortium of industry experts to standardize how REST APIs are described
- Promotes a vendor-neutral description format
- <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md>

## Swagger (by SmartBear Software)



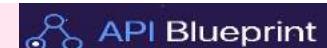
- Swagger Specification is the basis of this Open Specification
- Is a set of open source tools for creating, designing, documenting and architecting APIs based on RESTful standards

## RAML (by MuleSoft)



- RESTful API Modeling Language (RAML) for managing API lifecycle
- A machine readable API design that is also human friendly  
(Merges with Open API Initiative – Apr 2017)

## API Blueprint (Open API under MIT License)



- Offers a clean documentation format
- ‘Design first’ philosophy
- Decouple elements of API interface from backend implementation



# RESTful Guiding Principles

Client  
Server

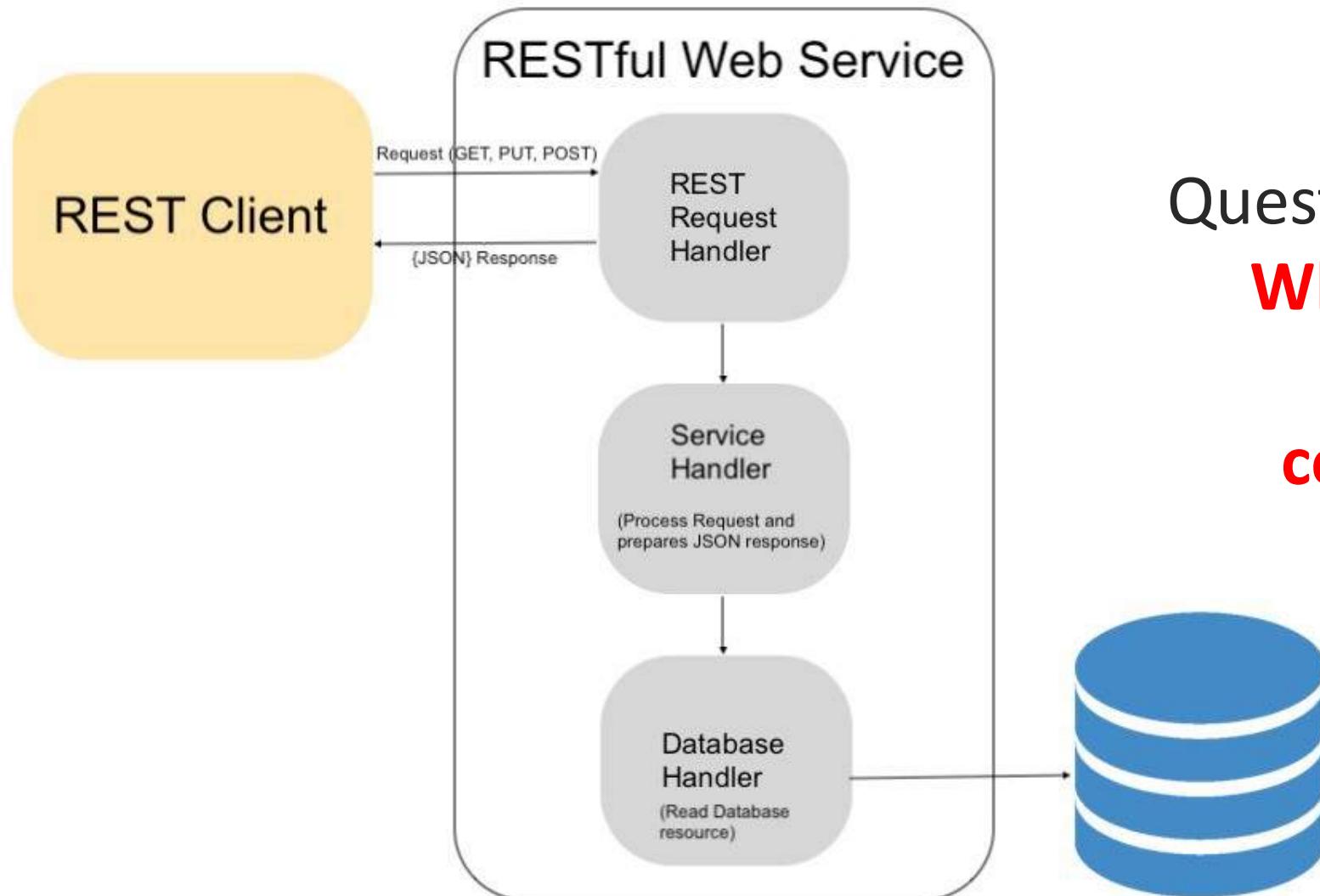
Stateless  
Model

Uniform  
Interface

Layered  
Architecture

Caching

HATEOAS



Question to think about:  
**When do we need  
client-server  
communication?**



# Stateless Model

- HTTP protocol
- Client holds the application state.

## Advantages

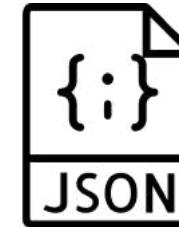
- **scaling APIs** to huge number of concurrent users by deploying it to multiple servers.
- **less complex** as there is absolutely no server-side state synchronization logic.
- **ease of caching** improves application performance



**REST statelessness means being free from the application state.**



## What is a resource?



- Identification of resources (Resource Identifier)
- Self-descriptive messages
- Manipulation of resources through representations
- Hypermedia as the engine of application state (HATEOAS)



# Uniform Interface – HTTP method

- Complies to HTTP methods

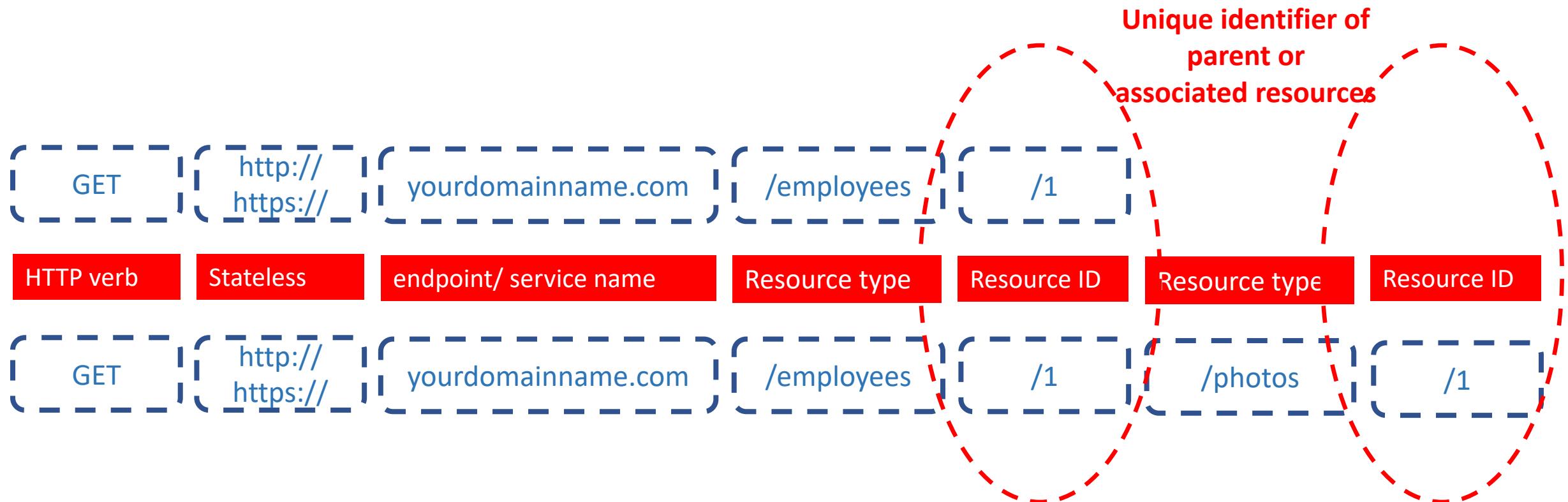
HTTP Method	Request has body	Response has body	Safe	Idempotent	Cachable
GET	NO	YES	YES	YES	YES
POST	YES	YES	NO	NO	YES
PUT	YES	YES	NO	YES	NO
DELETE	YES	YES	NO	YES	NO
TRACE	NO	YES	YES	YES	NO
OPTIONS	NO	YES	YES	YES	NO
CONNECT	NO	YES	NO	NO	NO
PATCH	YES	YES	NO	NO	NO

URI	HTTP Verb	Outcome
.../api/employees	GET	Gets list of employees
.../api/employees/1	GET	Gets employee with Id = 1
.../api/employees	POST	Creates a new employee
.../api/employees/1	PUT	Updates employee with Id = 1
.../api/employees/1	DELETE	Deletes employee with Id = 1



# Uniform Interface – URL + Resource Identifier

- Identification of resource





# REST Resource Naming Best Practices

Scenario	HTTP Method	URL Format
Add a customer	POST	/customers
Retrieve all customers	GET	/customers
Retrieve a specific customer	GET	/customers/{id}
Delete a customer	DELETE	/customers/{id}
Get all orders of a customer	GET	/customers/{id}/orders
Get specific order details of a customer	GET	/customers/{id}/orders/{id}
Get the address of a customer	GET	/customers/{id}/address
Edit details of a specific customer	PATCH	/customers/{id}

naming conventionß

- **Resources should be nouns, not verbs**
  - ✓ API endpoint as `/customers`
  - ✗ API endpoint as `/getCustomers`
- **Use plural nouns (exception if the resource is singleton)**
  - ✓ `/customers/{id}/address`
- **Use kebab-case for resource names that have more than one word**
  - ✓ `/customers/{customer-id}/orders/{order-id}/order-items`
- **Use path variables instead of using parameters whenever possible**
  - ✓ `/customers/{customer-id}/orders/{order-id}/order-items`
  - ✗ `/customers?customer-id={cust-id}&orders-id={order-id}`
- **Consistency**
  - ✓ `/customers`
  - ✗ `/customers/`



# Layered Architecture

## Project Structure

### Core

- Entities
- Interfaces
- Specifications
- ValueObjects
- Exceptions

### Application

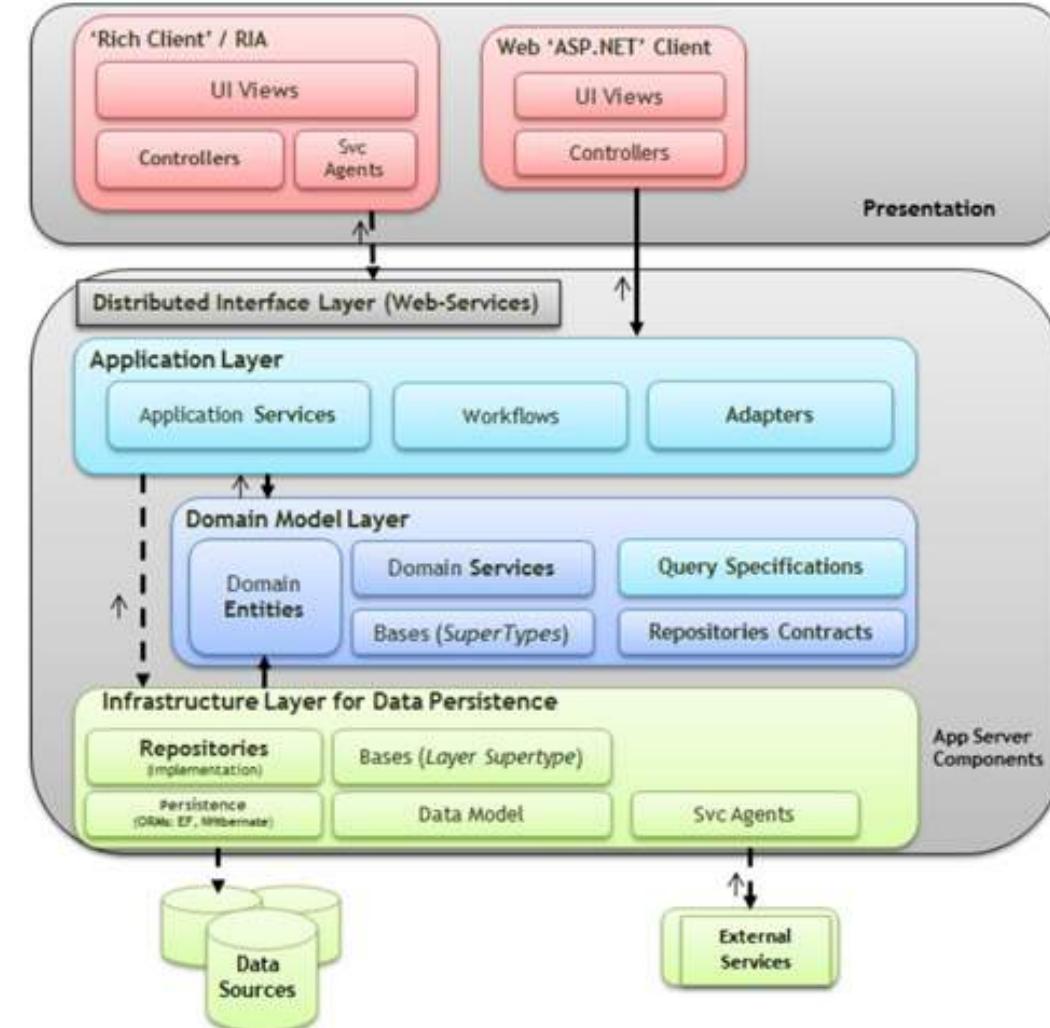
- Interfaces
- Services
- DtOs
- Mapper
- Exceptions

## Infrastructure

- Data
- Repository
- Services
- Migrations
- Logging
- Exceptions

### Web

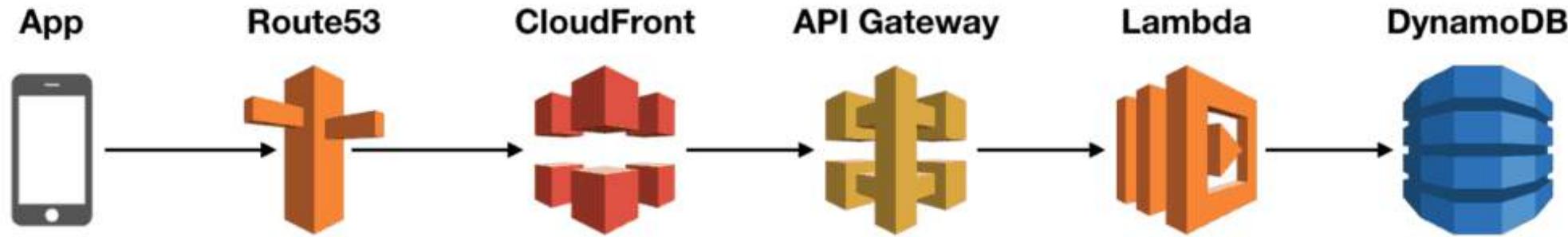
- Interfaces
- Services
- Pages
- ViewModels
- Extensions
- Mapper



<https://medium.com/aspnetrun/layered-architecture-with-asp-net-core-entity-framework-core-and-razor-pages-53a54c4028e3>



# Caching



1. Client app
2. CloudFront (CDN)
3. API Gateway (API Management)
4. Lambda (code/application-level)
5. Elasticache/DAX

Amazon Route 53 is a scalable and highly available domain name system (DNS) web service offered by Amazon Web Services (AWS). It provides domain registration, DNS routing, and health checking of resources to help route incoming internet traffic to the appropriate resources.

A CDN, or Content Delivery Network, is a network of distributed servers strategically positioned in various locations around the world. The primary purpose of a CDN is to deliver web content, such as images, videos, stylesheets, JavaScript files, and other assets, to users in a faster and more efficient manner.

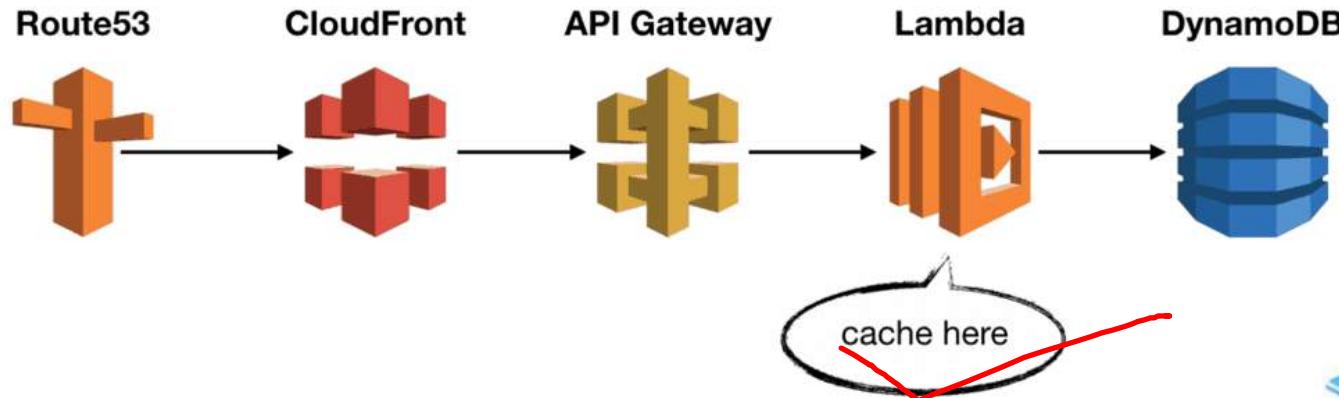
Caching: API Gateway can cache responses from your backend services. This reduces the load on your services and improves response times, especially for frequently requested data.

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). It enables you to run code without provisioning or managing servers. Lambda is event driven.

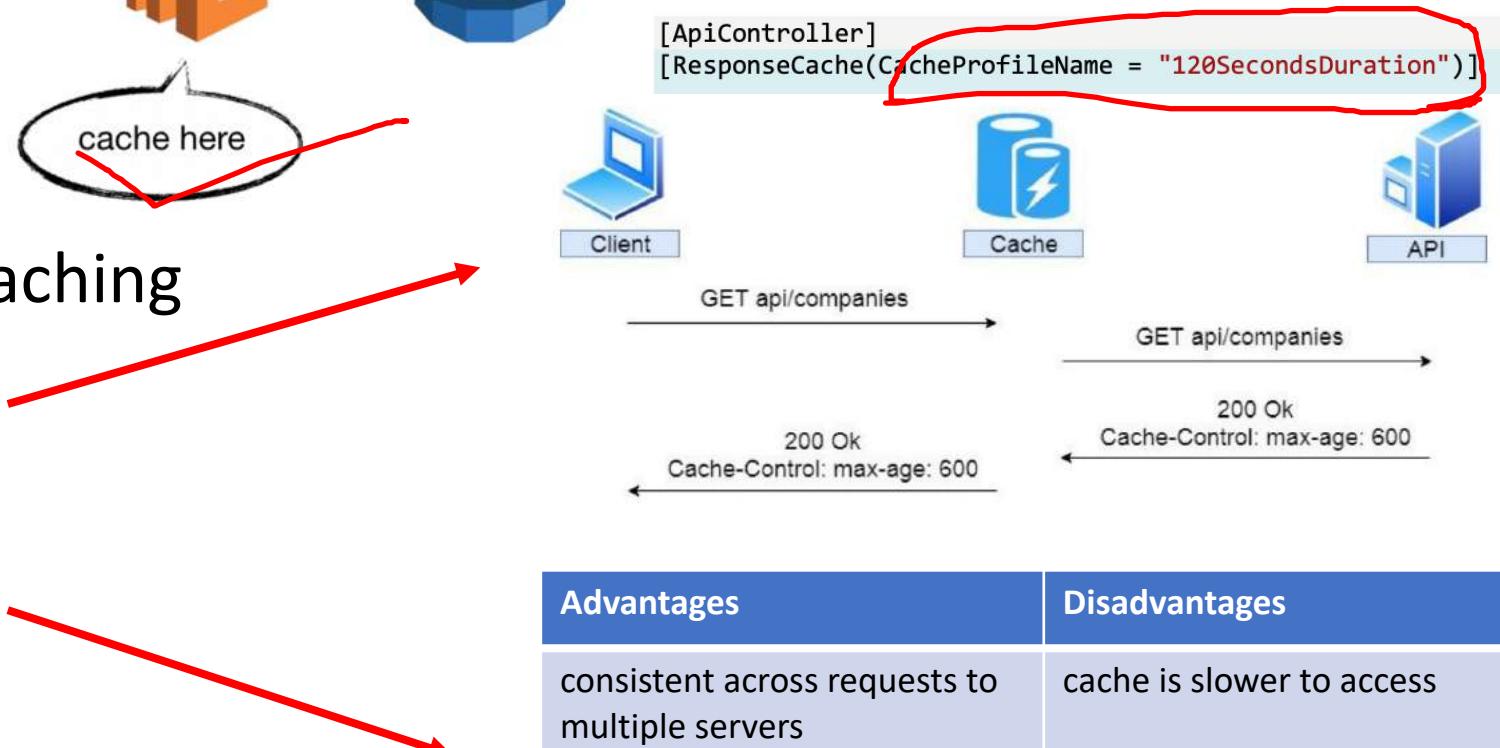
DAX stands for Amazon DynamoDB Accelerator. It is a fully managed, in-memory caching service that is designed to enhance the performance of read-intensive workloads using Amazon DynamoDB, a NoSQL database service provided by Amazon Web Services (AWS).



# Caching



- Application/API-level caching
  - Response caching
  - In-memory caching
  - Distributed caching



Advantages	Disadvantages
consistent across requests to multiple servers	cache is slower to access
Survives server restarts and app deployments	implement a separate cache service – some complexity involved
Doesn't use local memory	



- **Hypermedia links in API response**
- **loose coupling**
  - server drives the application state and URL structure
  - Discovery and exploration of API
  - Client use links in API response to implement its logic

```
{  
    "Id": "261e1685-cf26-494c-b17c-3546e65f5620",  
    "Name": "Anna Bosh",  
    "DateOfBirth": "1974-11-14T00:00:00",  
    "Address": "27 Colored Row",  
    "Links": [  
        {  
            "href": "https://localhost:5001/api/owner/261e1685-cf26-494c-b17c-3546e65f5620",  
            "rel": "self",  
            "method": "GET"  
        },  
        {  
            "href": "https://localhost:5001/api/owner/261e1685-cf26-494c-b17c-3546e65f5620",  
            "rel": "delete_owner",  
            "method": "DELETE"  
        },  
        {  
            "href": "https://localhost:5001/api/owner/261e1685-cf26-494c-b17c-3546e65f5620",  
            "rel": "update_owner",  
            "method": "PUT"  
        }  
    ]  
},
```

REST also allows client functionality to extend by downloading and executing code in the form of applets or scripts.

The downloaded code simplifies clients by reducing the number of features required to be pre-implemented. Servers can provide part of features delivered to the client in the form of code, and the client only needs to execute the code.



# API Design Best practices - I

## HTTP methods

Describe resource functionality with HTTP methods

Method	Description
GET	Used to retrieve a representation of a resource.
POST	Used to create new resources and sub-resources
PUT	Used to <u>update</u> existing resources
PATCH	Used to <u>update</u> existing resources
DELETE	Used to delete existing resources



# Differences between PUT and POST in REST

PUT	POST
PUT methods are used to request the server to store the enclosed entity in request. In case, the request does not exist, then new resource has to be created. If the resource exists, then the resource should get updated.	POST method is used to request the server to store the enclosed entity in the request as a new resource.
The URI should have a resource identifier. Example: <code>PUT /users/{user-id}</code>	The POST URI should indicate the collection of the resource. Example: <code>POST /users</code>
PUT methods are idempotent.	POST methods are not idempotent.
PUT is used when the client wants to modify a single resource that is part of the collection. If a part of the resource has to be updated, then PATCH needs to be used.	POST methods are used to add a new resource to the collection.
The responses are not cached here despite the idempotency.	Responses are not cacheable unless the response explicitly specifies Cache-Control fields in the header.
In general, PUT is used for UPDATE operations.	POST is used for CREATE operations.



# Recap – Core components of HTTP Request & Response

Method/Verb	GET, PUT, POST, DELETE, etc are some examples	Response Status Code	Example 400 represents a client-side error. 200 represents a successful response.
URI	Uniquely identify the resources on the server	HTTP Version	HTTP protocol version
HTTP Version	HTTP v1.1	Response Header	Metadata of the response message. Data can describe what is the content length, content type, response date, what is server type, etc
Request Header	Details of the request metadata such as client type, the content format supported, message format, cache settings, etc	Response Body	Represents the actual resource/message returned from the server
Request Body	Represents the actual message content to be sent to the server		



# API Design Best practices – II

## Idempotent requests

Getting the same response with repeated operations

Method	Description
GET	Should not impose any side effects. Repeated requests on the same resource should result in the same state.
POST	Used to <u>create new resources</u> and sub-resources, should not have any unrelated side-effects. <u>(Though NOT Idempotent)</u>
PUT	Should not impose any side effects. Repeated requests on the <u>same resource should result in the same state</u> .
PATCH	Should not impose any side effects. Repeated requests on the same resource should result in the same state. <u>(Typically, Need not be Idempotent)</u>
DELETE	Should not impose any side effects. Repeated requests on the same resource should result in the same state. First request might return status code 204 (no content) Subsequent requests might return status code 404 (not found)

HTTP Methods	Idempotent	Safe
OPTIONS	yes	yes
GET	yes	yes
HEAD	yes	yes
PUT	yes	no
POST	no	no
DELETE	yes	no
PATCH	no	no

PUT updates the entire resource or creates a new resource. PATCH updates specific fields of the resource.

<https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-implementation>  
<https://restcookbook.com/>



# API Design Best practices - III

## Meaningful feedback in responses

- Give meaningful feedback in Responses to help developers succeed
  - The **client application** behaved erroneously - (client error - **4xx** response code)
  - The **API** behaved erroneously - (server error - **5xx** response code)
  - The client and API worked - (success - **2xx** response code)
- Give examples of the responses

common code should be handled.

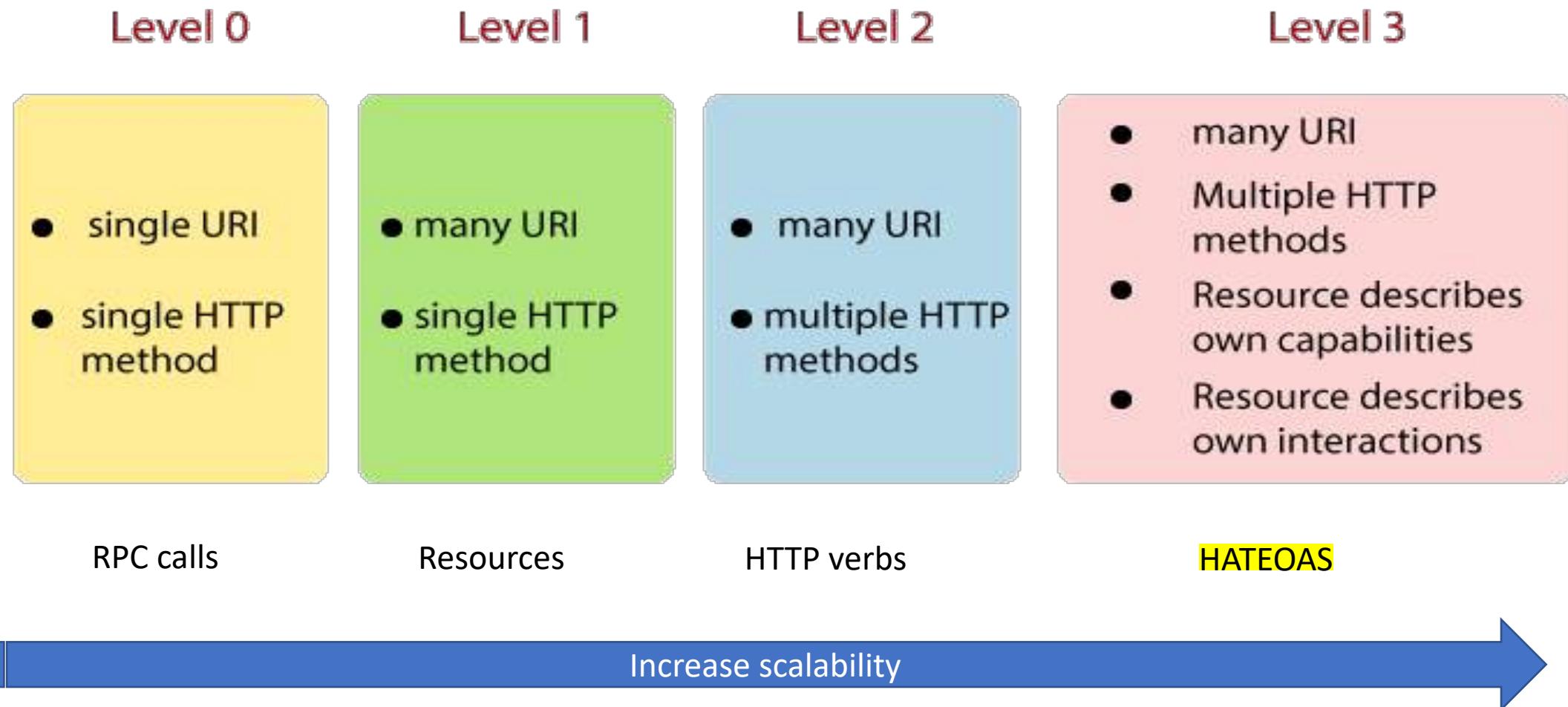
- **200:** Successful request, often a GET
- **201:** Successful request after a create, usually a POST
- **204:** Successful request with no content returned, usually a PUT or PATCH
- **400:** Bad request (client should modify the request)
- **401:** Unauthorized, credentials not recognized
- **403:** Forbidden, credentials accepted but don't have permission
- **404:** Not found, the resource does not exist
- **500:** Server error, generic and worth looking at other 500-level errors instead
- **503:** Service unavailable, another where retry headers are useful
- And more

### Well-designed API

- Improves developer experience
- Faster documentation
- Higher adoption for the APIs

# API Design Best practices - IV

## Richardson Maturity Model





# RMM Level 0: Plain Old XML (POX)

**One-line description:** Uses HTTP as transport, but not to indicate application state akin to classic SOAP and XML-RPC.

**What it's like:** Like going to the only takeaway restaurant in town and ordering a pizza, fried rice, and a curry.

**Natural Language:**

**Customer:** What type of food do you serve?

**Restaurant:** We serve Italian food such as pizzas and calzones, Chinese food like fried rice and prawn crackers and Indian food including curry and poppadoms.

**Customer:** Great. I want a pizza, some fried rice and a curry

**Restaurant:** Here's your pizza, fried rice and curry

```
1 GET/FoodService HTTP 1.1
2
3 {
4     "action" : "GetFoodServed"
5 }

1
2 HTTP 1.1 200 OK
3
4 {
5     "Italian": [ "pizza", "calzone" ],
6     "Chinese": [ "fried rice", "prawn crackers" ],
7     "Indian": [ "curry", "poppadoms" ]
8 }
9

1
2 GET/FoodService HTTP 1.1
3
4 {
5     "action": "MakeFoodOrder",
6     "order": [ "pizza", "fried rice", "curry" ]
7 }

1
2 HTTP 1.1 200 OK
3
4 {
5     "order": [ "pizza", "fried rice", "curry" ]
6 }
7
```



# RMM Level 1: Resources

**One-line description:** Distinguishes between different resources, but uses one HTTP method.

**What it's like:** Like going to different takeaway restaurants and ordering pizza at the Italian restaurant, fried rice at the Chinese restaurant and a curry at the Indian restaurant.

**Natural Language:**

**Customer:** (goes to Pizza restaurant) I want a pizza with pepperoni and extra cheese toppings.

**Restaurant:** OK. Here's your pizza. That will be £15.00.

**Customer:** (goes to Chinese restaurant) I want fried rice.

**Restaurant:** OK. Here's you rice. That will be £2.50.

**Customer:** (goes to the Indian restaurant) I want a curry.

**Restaurant:** OK. Here's your curry. That will be £5.60.

```
1 POST/restaurants/italian/orders HTTP 1.1
2
3
4 {
5     "order": [
6         {
7             "item": "pizza",
8             "quantity": 1
9         }
10    }
11
12
13
14 HTTP 1.1 200 OK
15
16
17 {
18     "order": [
19         {
20             "orderNo": "123456",
21             "item": "pizza",
22             "toppings": [ "pepperoni", "extra cheese" ],
23             "quantity": 1
24         },
25         {
26             "total": "£15.00"
27         }
28     ]
29 }
```



# RMM Level 1: Resources

**One-line description:** Distinguishes between different resources, but uses one HTTP method.

**What it's like:** Like going to different takeaway restaurants and ordering pizza at the Italian restaurant, fried rice at the Chinese restaurant and a curry at the Indian restaurant.

**Natural Language:**

**Customer:** (goes to Pizza restaurant) I want a pizza with pepperoni and extra cheese toppings.

**Restaurant:** OK. Here's your pizza. That will be £15.00.

**Customer:** (goes to Chinese restaurant) I want fried rice.

**Restaurant:** OK. Here's your rice. That will be £2.50.

**Customer:** (goes to the Indian restaurant) I want a curry.

**Restaurant:** OK. Here's your curry. That will be £5.60.

```
1 POST/restaurants/chinese/orders HTTP 1.1
2
3 {
4     "order": [
5         {
6             "item": "fried rice",
7             "quantity": 1
8         }
9     }
10
11
12 HTTP 1.1 200 OK
13
14 {
15     "order": [
16         {
17             "orderNo": "AW76W09",
18             "item": "fried rice",
19             "quantity": 1
20         },
21         "total": "£2.50"
22     }
23 }
```



# RMM Level 1: Resources



**One-line description:** Distinguishes between different resources, but uses one HTTP method.

**What it's like:** Like going to different takeaway restaurants and ordering pizza at the Italian restaurant, fried rice at the Chinese restaurant and a curry at the Indian restaurant.

## Natural Language:

**Customer:** (goes to Pizza restaurant) I want a pizza with pepperoni and extra cheese toppings.

**Restaurant:** OK. Here's your pizza. That will be £15.00.

**Customer:** (goes to Chinese restaurant) I want fried rice.

**Restaurant:** OK. Here's your rice. That will be £2.50.

**Customer:** (goes to the Indian restaurant) I want a curry.

**Restaurant:** OK. Here's your curry. That will be £5.60.



# RMM Level 2: HTTP Verbs

**One-line description:** Full use of all HTTP verbs combined with resource nouns.

**What it's like:** Like going to the Italian restaurant and specifying the pizza topping, then going to the Chinese restaurant to cancel the fried rice, and then going to the Indian to change the number of poppadoms.

## Natural Language:

**Customer:** (goes to Pizza restaurant) I want a pizza.

**Restaurant:** OK. Here's your pizza. That will be £15.00.

**Customer:** (goes to Chinese restaurant) I want fried rice

**Restaurant:** OK. Here's your rice. That will be £2.50.

**Customer:** (returns to the pizza restaurant) I want to change my order and add extra toppings. I want to have olives and anchovies. Oh, and make that two pizzas.

**Restaurant:** OK we can do that Sir. Gives us your pizza and we will add those toppings.

**Customer:** (goes to Indian restaurant) I want curry and two poppadoms.

**Restaurant:** OK here's your order. That will be £8.20.

**Customer:** I have changed my mind. I want to cancel my order.

**Restaurant:** OK. your order has been cancelled.

```
1  PUT/restaurants/italian/orders/123456 HTTP 1.1
2
3
4
5   "order": [
6     "items": [
7       {
8         "item": "pizza",
9         "toppings": [
10           "pepperoni", "extra cheese",
11           "olives", "anchovies"
12         ],
13         "quantity": 2
14       }
15     ]
16   }

1  HTTP 1.1 200 OK
2
3
4
5   "order": [
6     "items": [
7       {
8         "item": "pizza",
9         "toppings": [
10           "pepperoni", "extra cheese",
11           "olives", "anchovies"
12         ],
13         "quantity": 2
14       }
15     ],
16     "orderNo": "123456",
17     "total": "£34.00"
18   ]
19 
```



# RMM Level 2: HTTP Verbs

**One-line description:** Full use of all HTTP verbs combined with resource nouns.

**What it's like:** Like going to the Italian restaurant and specifying the pizza topping, then going to the Chinese restaurant to cancel the fried rice, and then going to the Indian to change the number of poppadoms.

**Natural Language:**

**Customer:** (goes to Pizza restaurant) I want a pizza.

**Restaurant:** OK. Here's your pizza. That will be £15.00.

**Customer:** (goes to Chinese restaurant) I want fried rice

**Restaurant:** OK. Here's your rice. That will be £2.50.

**Customer:** (returns to the pizza restaurant) I want to change my order and add extra toppings. I want to have olives and anchovies. Oh, and make that two pizzas.

**Restaurant:** OK we can do that Sir. Gives us your pizza and we will add those toppings.

**Customer:** (goes to Indian restaurant) I want curry and two poppadoms.

**Restaurant:** OK here's your order. That will be £8.20.

**Customer:** I have changed my mind. I want to cancel my order.

**Restaurant:** OK. your order has been cancelled.

```
2 POST/restaurants/indian/orders HTTP 1.1
3
4 {
5     "order": [
6         {
7             "items": [
8                 {
9                     "item": "curry",
10                    "quantity": 1
11                 },
12                 {
13                     "item": "poppadoms",
14                     "quantity": 2
15                 }
16             ]
17         }
18     }
19 }
20
21 HTTP 1.1 200 OK
22
23 {
24     "order": [
25         {
26             "items": [
27                 {
28                     "item": "curry",
29                     "quantity": 1
30                 },
31                 {
32                     "item": "poppadoms",
33                     "quantity": 2
34                 }
35             ]
36         },
37         {
38             "orderNo": "89GY7QW8",
39             "total": "£5.60"
40         }
41     ]
42 }
```



# RMM Level 2: HTTP Verbs

**One-line description:** Full use of all HTTP verbs combined with resource nouns.

**What it's like:** Like going to the Italian restaurant and specifying the pizza topping, then going to the Chinese restaurant to cancel the fried rice, and then going to the Indian to change the number of poppadoms.

## Natural Language:

**Customer:** (goes to Pizza restaurant) I want a pizza.

**Restaurant:** OK. Here's your pizza. That will be £15.00.

**Customer:** (goes to Chinese restaurant) I want fried rice

**Restaurant:** OK. Here's your rice. That will be £2.50.

**Customer:** (returns to the pizza restaurant) I want to change my order and add extra toppings. I want to have olives and anchovies. Oh, and make that two pizzas.

**Restaurant:** OK we can do that Sir. Gives us your pizza and we will add those toppings.

**Customer:** (goes to Indian restaurant) I want curry and two poppadoms.

**Restaurant:** OK here's your order. That will be £8.20.

**Customer:** I have changed my mind. I want to cancel my order.

**Restaurant:** OK. your order has been cancelled.

```
1
2 DELETE /restaurants/indian/orders/89GY7QW8 HTTP 1.1
3
4 HTTP 1.1 200 OK
5
6 {
7     "order": [
8         {
9             "items": [
10                 {
11                     "item": "curry",
12                     "quantity": 1
13                 },
14                 {
15                     "item": "poppadoms",
16                     "quantity": 2
17                 }
18             ],
19             "orderNo": "89GY7QW8",
20             "total": "£5.60"
21         }
22     ]
23 }
```



# RMM Level 3: Hypermedia Controls



**One-line description:** Uses **HATEOAS** (Hypermedia As The Engine Of Application State) to direct the application state.

**What it's like:** Like going to the pizza restaurant to order a pizza, then the waiter brings you your pizza, tells you what you just ordered, what other pizzas are available, and how to get more info about each one.

## Natural Language:

**Customer:** (goes to pizza restaurant) I want a pizza.

**Restaurant:** Here's your pizza. You ordered a pepperoni pizza. Would you like to know what extra toppings we have? We have other pizzas: marguerita, four seasons, and four cheeses. Would you like more info about each one?

```
3 POST/restaurants/italian/orders HTTP 1.1
4
5 {
6     "order": {
7         "items": [
8             {
9                 "item": "pizza",
10                "toppings": [ "pepperoni", "extra cheese" ],
11                "quantity": 1
12            }
13        ]
14    }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31 }
```

---

```
1
2
3 HTTP 1.1 200 OK
4
5 {
6     "response": [
7         {
8             "order": [
9                 {
10                    "items": [
11                        {
12                            "item": "pizza",
13                            "toppings": [ "pepperoni", "extra cheese" ],
14                            "quantity": 1
15                        }
16                    ],
17                    "orderNo": "123456",
18                    "total": "£17.00"
19                }
20            ],
21            "links": [
22                {
23                    "rel": "self",
24                    "href": "/restaurants/italian/orders/89GY7QW8",
25                    "method": "GET"
26                },
27                {
28                    "rel": "toppings",
29                    "href": "/restaurants/italian/toppings",
30                    "method": "GET"
31                },
32                {
33                    "rel": "delete_order",
34                    "href": "/restaurants/italian/orders/89GY7QW8",
35                    "method": "DELETE"
36                }
37            ]
38        }
39    ]
40 }
```



# What is GraphQL for APIs

- A query language for APIs and a runtime for fulfilling those queries with your existing data
- Provides a complete and understandable description of the data in your API
- Gives clients the power to ask for **exactly** what they need and nothing more
  - Instead of gathering the data by accessing multiple endpoints
  - send a single query to the GraphQL server (which includes the concrete data requirements). The server then responds with a JSON object where these requirements are fulfilled.
- Makes it easier to evolve APIs over time, and enables powerful developer tools

A Popular Alternative for RESTful APIs

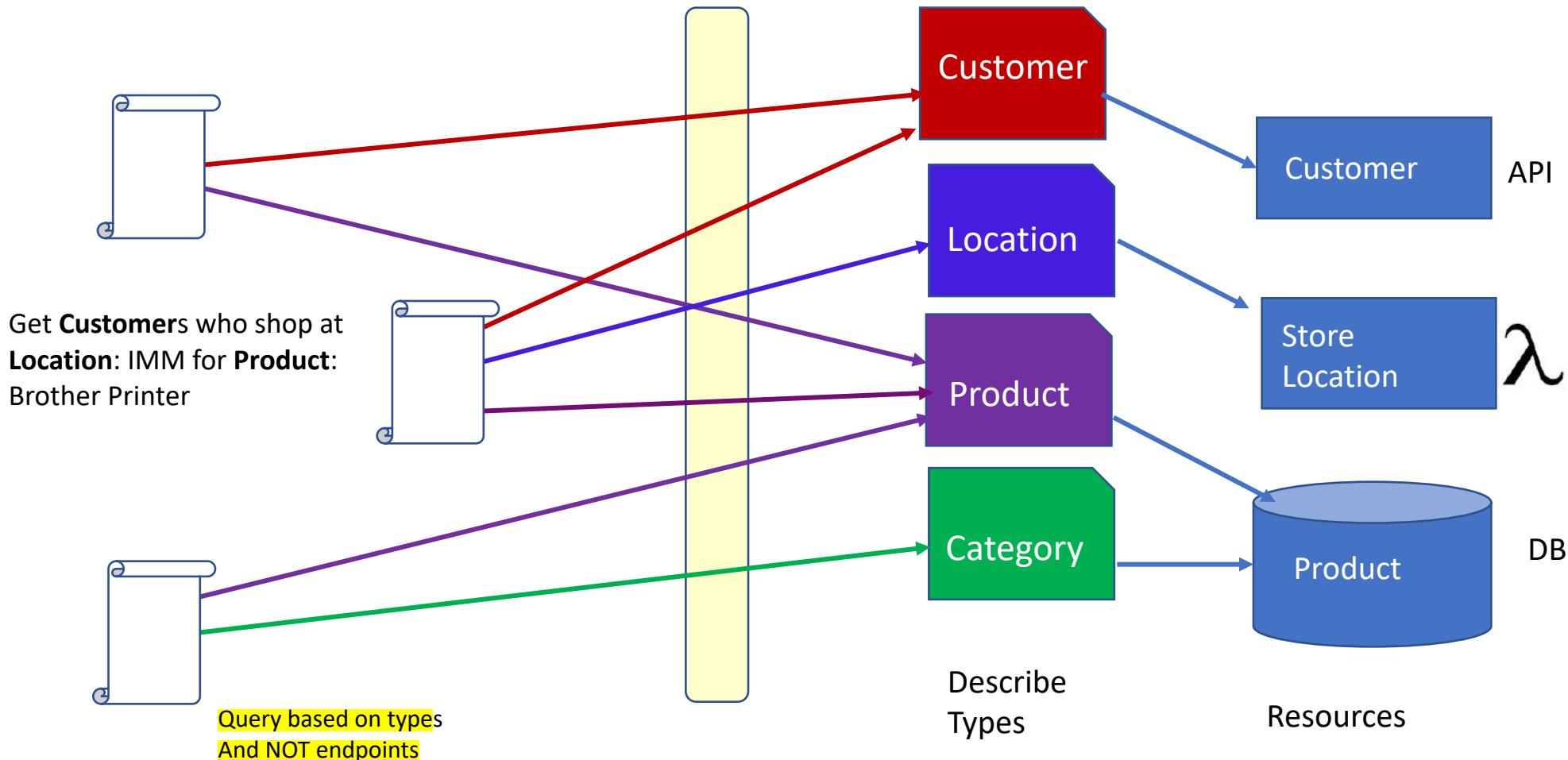
<https://www.howtographql.com/basics/2-core-concepts/>





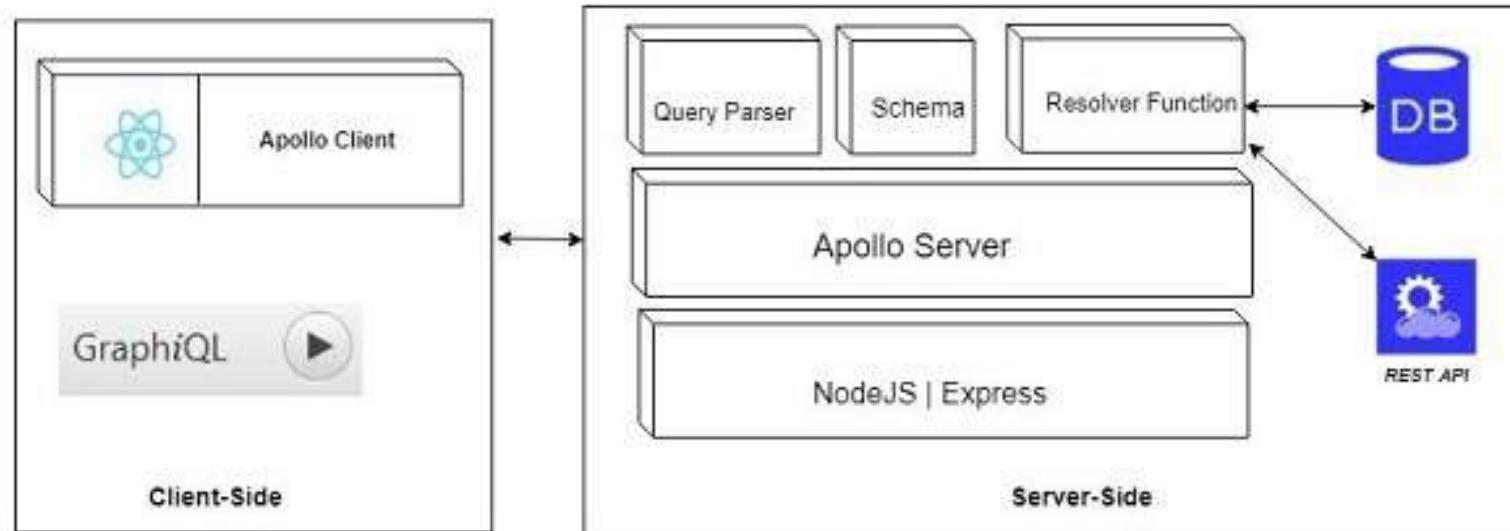
# GraphQL - How it works

One Endpoint E.g.: <https://bestdenki.com/graphql>





# GraphQL: How it works (an example)

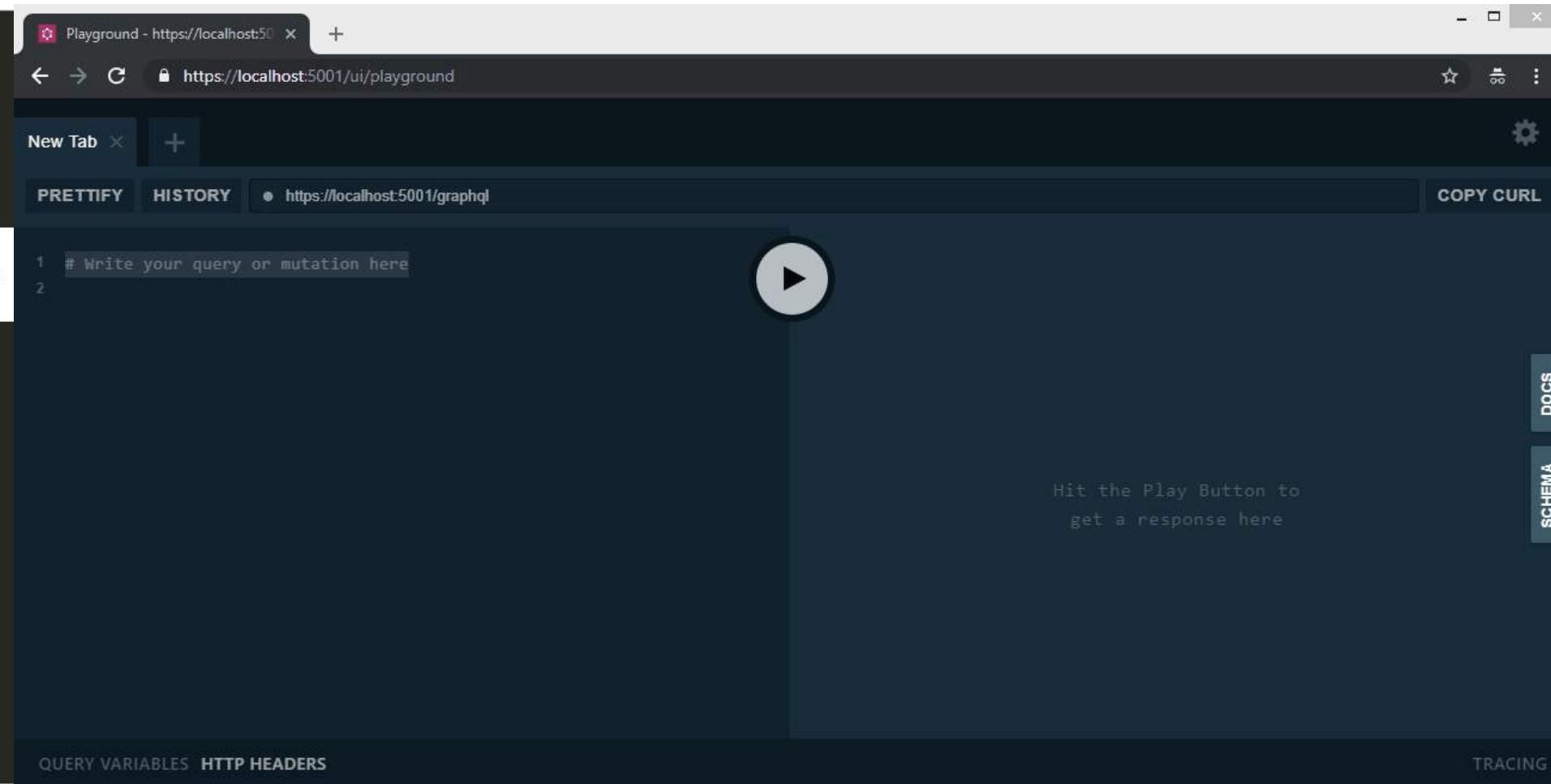


Client-Side		Server-Side	
GraphQL	Browser based interface for editing and testing GraphQL queries and mutations	Schema	Describes the functionality available to the clients which connect to it
Apollo Client	A tool to build GraphQL client applications. Integrates well with all JavaScript front-end.	Query	Request to retrieve data from database or legacy API's.
<a href="https://www.tutorialspoint.com/graphql/graphql_application_components.htm">https://www.tutorialspoint.com/graphql/graphql_application_components.htm</a>			

```
query OwnersQuery {  
  owners {  
    name  
    account {  
      type  
    }  
  }  
}
```

We are 100% sure that we will get this response back:

```
{  
  "data": {  
    "owners": [  
      {  
        "name": "John Doe",  
        "accounts": [  
          {  
            "type": "Cash"  
          },  
          {  
            "type": "Savings"  
          }  
        ]  
      }  
    ]  
  }  
}
```



The screenshot shows a GraphQL playground interface. On the left, a code editor displays a GraphQL query:

```
query OwnersQuery {  
  owners {  
    name  
    account {  
      type  
    }  
  }  
}
```

On the right, the playground interface is shown. It has tabs for "PRETTIFY" and "HISTORY". Below the tabs, it says "https://localhost:5001/graphql". A large input field contains the query. To the right of the input field is a "PLAY" button. Below the input field, it says "Hit the Play Button to get a response here". At the bottom of the playground, there are buttons for "QUERY VARIABLES" and "HTTP HEADERS".



# GraphQL for API vs RESTful API

	GraphQL	REST
Architecture	Client-Driven	Server-Driven
Invocation	Schema & Type system	Endpoints
Operations	<ul style="list-style-type: none"><li>- Query</li><li>- Mutation (create, update or delete data)</li><li>- Subscription</li></ul>	<ul style="list-style-type: none"><li>- Create</li><li>- Retrieve</li><li>- Update</li><li>- Delete</li></ul>
Data Fetching	Specific data with a single API call	Fixed data with multiple API call
Performance	Fast <b>(no more over and under fetching)</b>	Can be slow due to multiple API calls
File Upload	Yes	Yes
Web caching	<ul style="list-style-type: none"><li>- Hard to cache</li><li>- Third party libraries</li></ul>	Yes
Use cases	<ul style="list-style-type: none"><li>- Multiple Micro services</li><li>- Mobile Apps</li></ul>	<ul style="list-style-type: none"><li>- Simple apps</li><li>- Resource-driven apps</li></ul>



# API Versioning

Part	Usage
Major Version	Breaking changes (every time you make a breaking change, increment major version by 1)
Minor Version	Backward compatible features (every time you add a new backward compatible feature, increment minor version by 1)
Patch Version	Backward compatible fixes (every time you apply a backward compatible patch, increment patch version by 1)

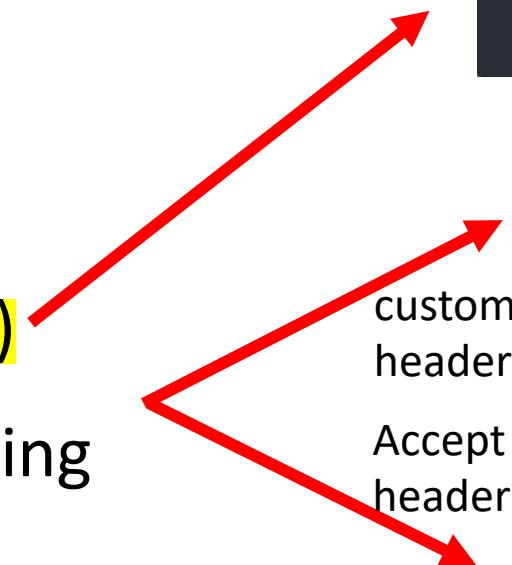
## Rules

1. Don't take anything away abruptly.
2. Don't change the meaning of an existing API.
3. Additions to an existing API is optional.

`http://api.example.com/v1`  
`http://apiv1.example.com`

## Approaches

1. URI Versioning (Recommended)
2. Custom/Accept Header Versioning

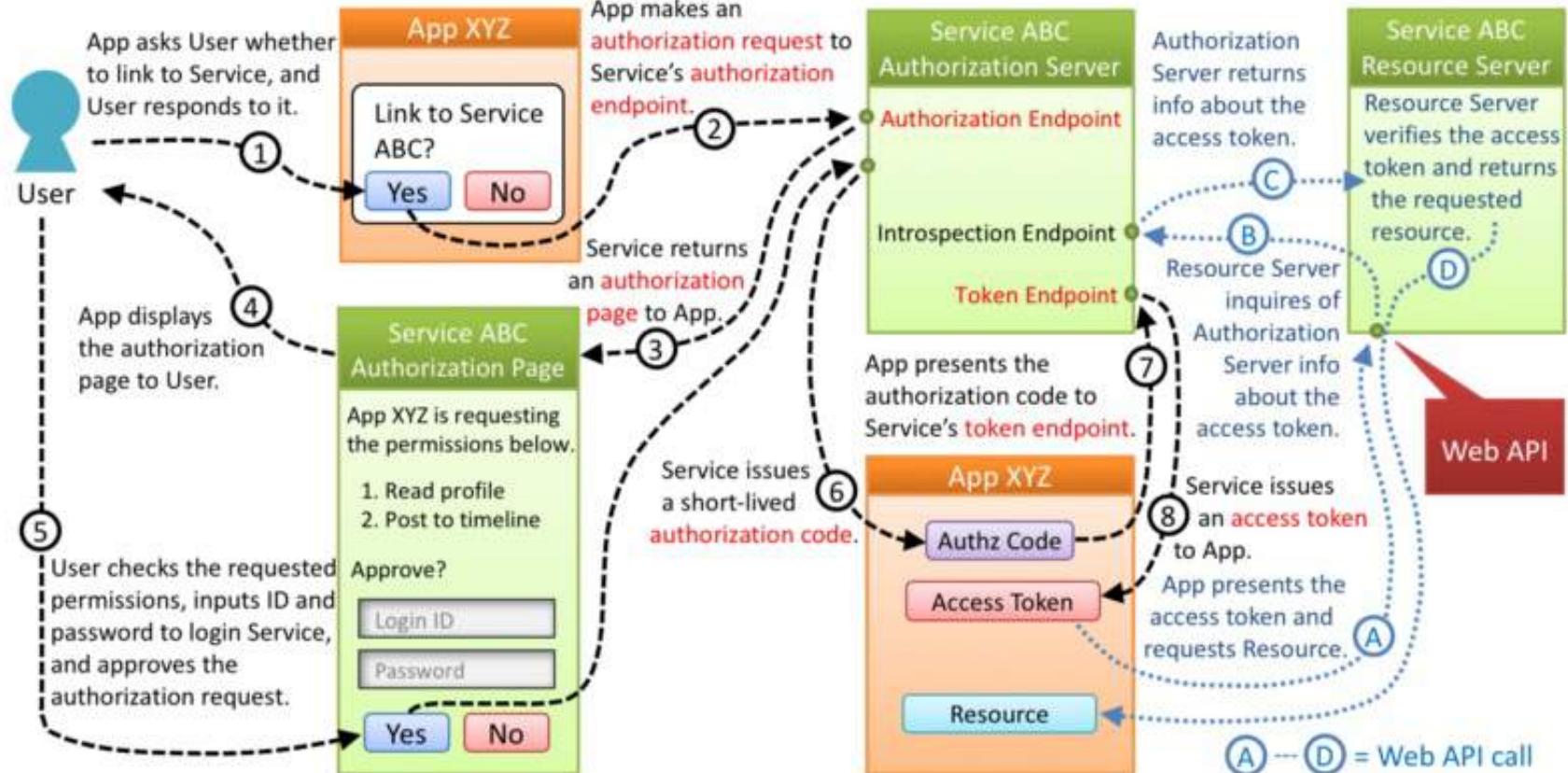


`Accept-version: v1`  
`Accept-version: v2`

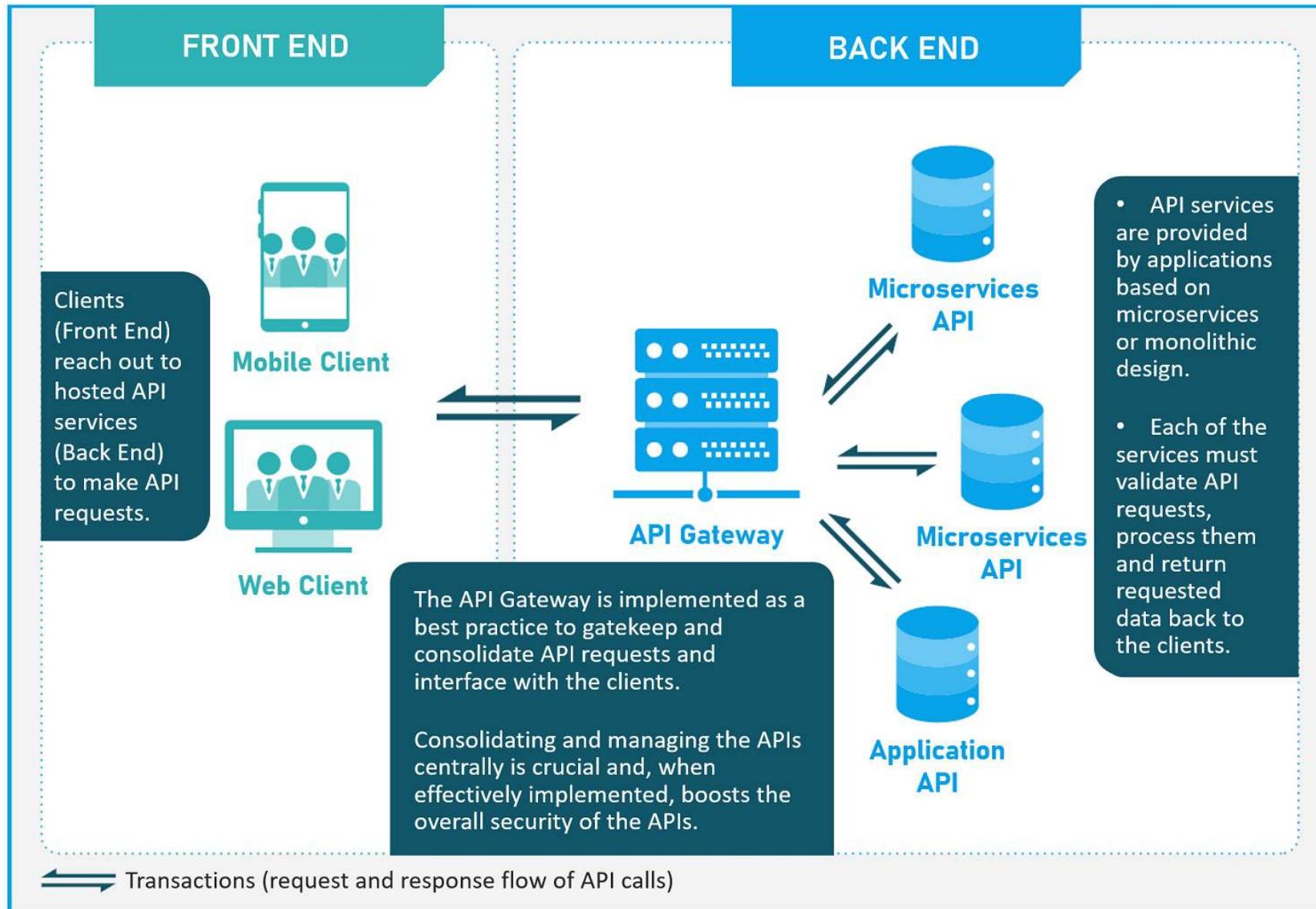
`Accept: application/vnd.example.v1+json`  
`Accept: application/vnd.example+json;version=1.0`

# API Security – Sample OAuth 2.0 Authorization Flow

## Authorization Code Flow (RFC 6749, 4.1)



<https://medium.com/@darutk/diagrams-and-movies-of-all-the-oauth-2-0-flows-194f3c3ade85>





# Summary - 1

- API development with a Product mindset
- API standards
  - Open API, Swagger, RAML, API Blueprint
  - Framework support for API development
- API Best Practices
  - Resource Oriented (nouns), follow HTTP verbs
  - API Maturity Levels
  - **HATEOAS – Hypermedia As The Engine Of Application State**
  - GraphQL vs RESTful APIs
  - Outside-in design, Time-to-market



# Summary - 2

- API design principles
  - Developer Empathy
  - **Granularity (Follow the SOLID principles: SRP, ISP)**
  - Follow industry practices, accepted standards, documentation
- API Versioning
  - URI, Accept header, custom header
- API Security
  - Validate parameters, threat detection, SSL everywhere, proven solutions
  - Authenticate, Authorize (OAuth 2.0 is industry standard)



# Workshop: API Design

## Individual submission

- Refer to MaidToOrder Service case study
- Use the sample solution of the *Workshop: Identify domain concepts, BCs and UIs*
- Design the APIs for the following requirements:

**The customer orders the cleaning service from a cleaning agency. While the customer may make a one-time ad-hoc order, it is normally more economical for the customer to subscribe to a usage plan. There are 3 plans at present with different preferential rates: OnceAFortnight, OnceAWeek, TwiceAWeek.**

- Each team member (individual) should design the APIs in any one of the following style
  - RESTful (Level 2 or 3) for a different resource for the service. Specify: URI, verbs, parameters, HTTP verb, response
  - GraphQL: Specify the Types and show an example of a Query
- Name of your submission file: <Name of the participant>\_API\_Design.docx



# APPENDIX



## Developer Empathy

- User Centric Design
- Simplicity
- Fit for Purpose
- Act on feedback



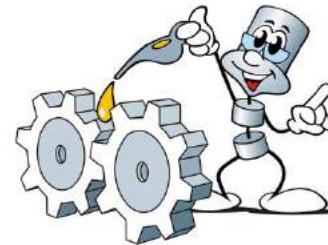
## Granularity

- Re-usable APIs
- Identify key entities that impact the service
- Design at the lowest practical level
- Can be used and combined in different ways
- Single responsibility principle for APIs
- Interface Segregation Principle for APIs



## Ubiquitous standards

- Develop APIs using well-accepted standards
- Make no assumptions of the technology used by consumers
- Maintain SDKs for many different technologies
- Incorporate security controls for sensitive information/APIs



## Endpoint Stability

- Availability of APIs should be same as platform
- API versions must be backward compatible
- AB versions service levels must be explicit



## Security

- Level of Security must be according to its risk levels
- Apply to each granular level of the APIs



## Documentation

- APIs must be discoverable and documented
- Provide link to documentation from the endpoint
- Provide feedback channels
- APIs and documents must be aligned

## Use try...catch block to handle exception

```
[HttpGet]
public IActionResult Get()
{
    try
    {
        _logger.LogInfo("Fetching all the Students from the storage");

        var students = DataManager.GetAllStudents(); //simulation for the data base access

        _logger.LogInformation($"Returning {students.Count} students.");

        return Ok(students);
    }
    catch (Exception ex)
    {
        _logger.LogError($"Something went wrong: {ex}");
        return StatusCode(500, "Internal server error");
    }
}
```

## Handling Errors Globally with the Built-In Middleware

```
public static class ExceptionMiddlewareExtensions
{
    public static void ConfigureExceptionHandler(this IApplicationBuilder app, ILogManager logger)
    {
        app.UseExceptionHandler(appError =>
        {
            appError.Run(async context =>
            {
                context.Response.StatusCode = (int) HttpStatusCode.InternalServerError;
                context.Response.ContentType = "application/json";

                var contextFeature = context.Features.Get<IExceptionHandlerFeature>();
                if(contextFeature != null)
                {
                    logger.LogError($"Something went wrong: {contextFeature.Error}");

                    await context.Response.WriteAsync(new ErrorDetails()
                    {
                        StatusCode = context.Response.StatusCode,
                        Message = "Internal Server Error."
                    }.ToString());
                }
            });
        });
    }
}
```

## A cleaner way to handle API exception

```
public IActionResult Get()
{
    _logger.LogInfo("Fetching all the Students from the storage");

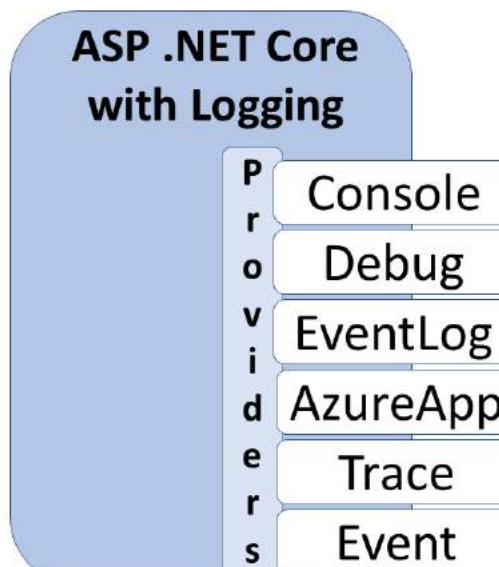
    var students = DataManager.GetAllStudents(); //simulation for the data base access

    throw new Exception("Exception while fetching all the students from the storage.");

    _logger.LogInformation($"Returning {students.Count} students.");

    return Ok(students);
}
```

- Provide a trail of activity leading up to an event
- Help supplement exception information recorded in other systems
- Understand how clients use our application
- Record application metrics



**Log Levels**

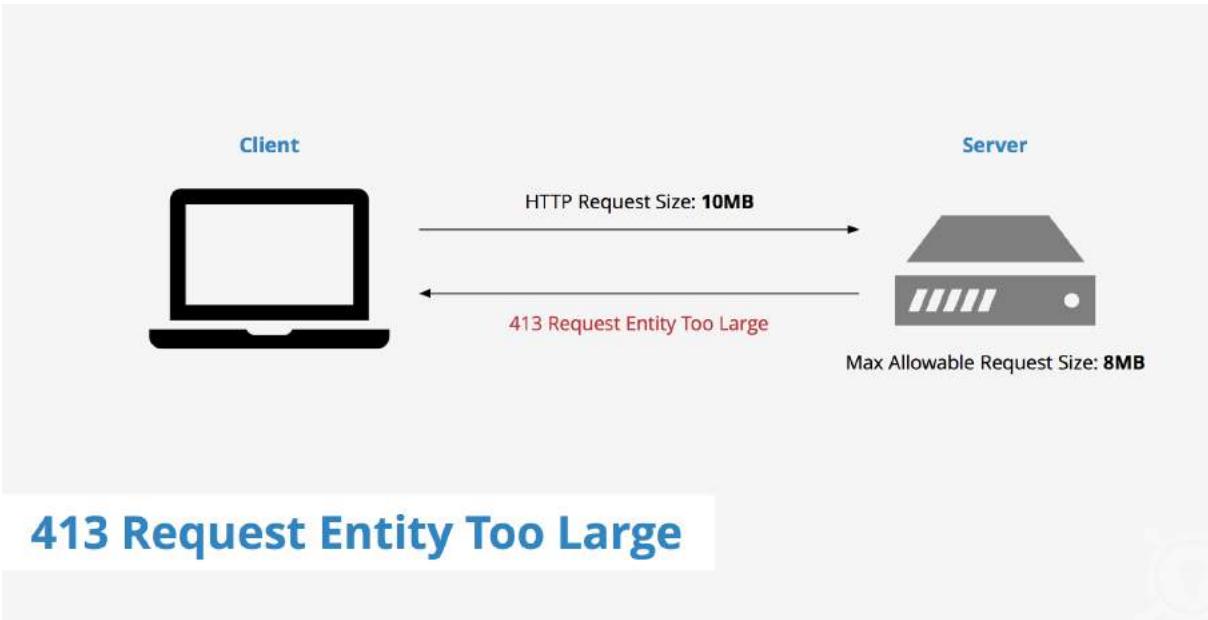
- Trace
- Debug
- Info
- Warning
- Error
- Critical
- None

using Serilog;  
using System;  
  
public static void Main(string[] args)  
{  
 try  
 {  
 Log.Information("Starting the application...")  
 var host = Host.CreateDefaultBuilder(args).Build();  
 host.Run();  
 }  
 catch (Exception ex)  
 {  
 Log.Fatal(ex, "Host terminated unexpectedly");  
 }  
 finally  
 {  
 Log.CloseAndFlush();  
 }  
}

[HttpGet]  
[Route("api/[controller]/[action]")]> Get()  
{  
 logger.LogInformation("Here is info message from the controller.");  
 logger.LogWarning("Here is warning message from the controller.");  
 logger.LogError("Here is error message from the controller.");  
  
 return new string[] { "value1", "value2" };  
}



# API Payload Too Huge Error



**Increase the web server capability to handle  
Larger payload**

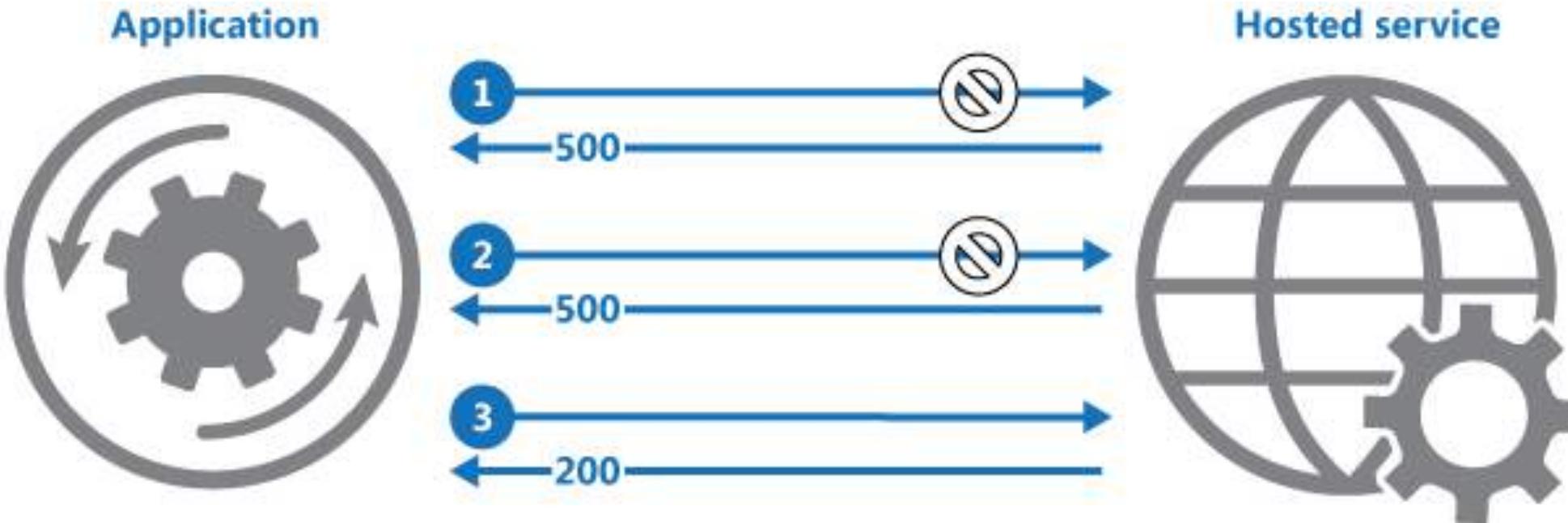
```
<system.webServer>
  <security>
    <requestFiltering>
      <requestLimits maxAllowedContentLength="209715200" />
    </requestFiltering>
  </security>
</system.webServer>
```

**Change the settings at the action level**

```
[HttpPost]
[RequestFormLimits(MultipartBodyLengthLimit = 209715200)]
[RequestSizeLimit(209715200)]
public IActionResult Upload(IFormFile file,
[FromServices] IHostingEnvironment env)
{
  ...
}
```



## • API Retries



- 1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
- 2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
- 3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).



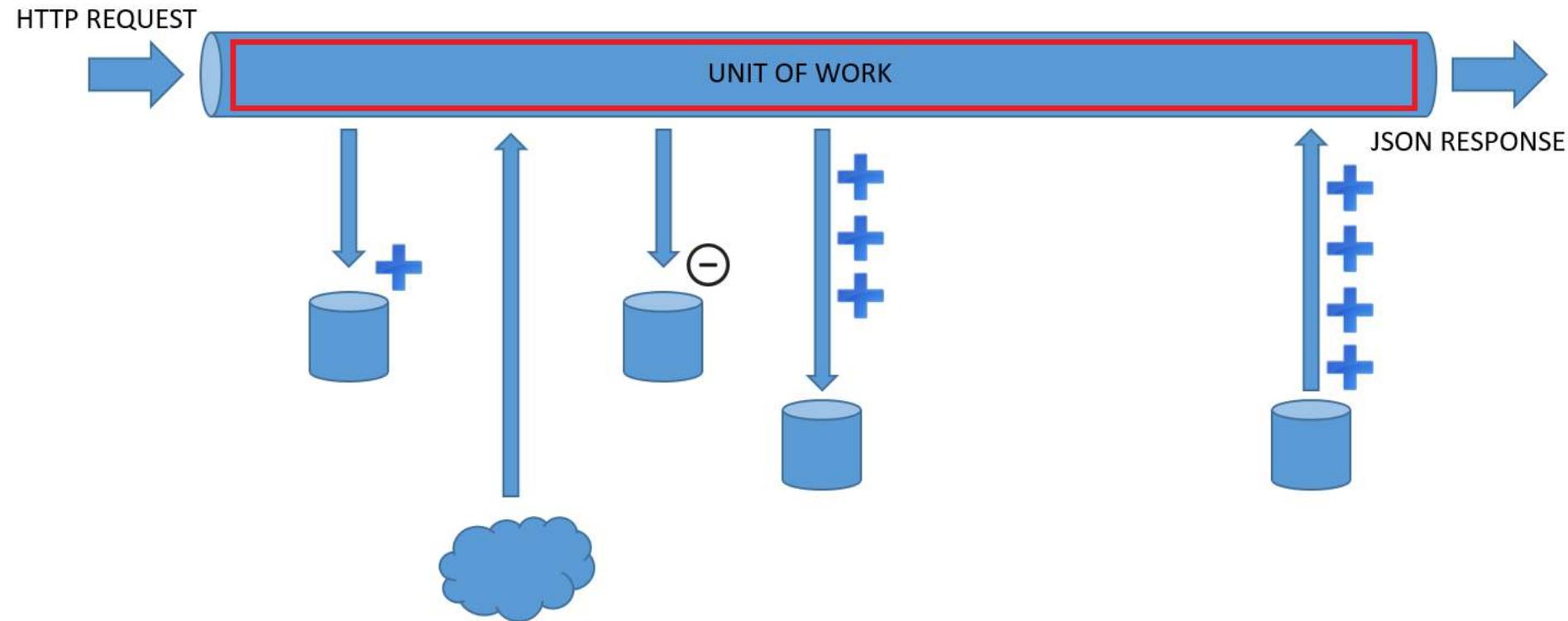
The Polly Project

**API call retries  
with exponential  
backoff and  
policies**

- Retry
- Circuit Breaker
- Bulkhead Isolation
- Timeout
- Fallback



# API Transactions



## Two approaches

- Offset Pagination
- Cursor Pagination

### Pagination Example

1	Lottie Gaylord III	Pitcairn Islands	Opal@katlyn.me	29/07/2015 18:13:59
2	Rory Torp	Serbia and Montenegro	Darryl.Mueller@davonte.co.uk	03/04/1990 01:47:24
3	Jadon Bartoletti	Kazakhstan	Carlo@clovis.info	21/05/2015 00:19:14
4	Enoch Haley	Samoa	Agustina.Bauch@arvel.name	29/08/2005 13:06:34
5	Ava Aufderhar	Cayman Islands	Coleman.Fadel@benjamin.info	12/01/2003 23:44:03
6	Donnie Labadie III	Jamaica	Maeve.Monahan@jan.info	28/04/2013 22:03:55
7	Janice Schaden	Micronesia	Laura_Jewess@brenden.tv	30/10/2006 14:30:50
8	Ms. Adolfo Beier	Israel	Jamey@jane.co.uk	17/08/2001 16:35:28
9	Aurelie Hoppe	Kuwait	Alberto.Schamberger@leola.net	16/07/2014 23:31:18
10	Jessyca Hayes	Switzerland	Kamren_Aufderhar@terrence.info	24/07/2008 02:45:41

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

## Offset Pagination

```
SELECT * FROM CARS WHERE MODEL='audi'  
LIMIT 5 OFFSET 100;
```

## Cursor Pagination

```
SELECT * FROM `cars` WHERE  
`cars`.`createdOn` > 1456323 LIMIT 5
```

- Filtering
  - `https://localhost:5001/api/owner?minYearOfBirth=1975&maxYearOfBirth=1997`
- Sorting
  - `https://localhost:5001/api/owner?orderBy=name desc,dateOfBirth`
- Filtering & sorting
  - `https://localhost:5001/api/owner?minYearOfBirth=1975&maxYearOfBirth=1997&orderBy=name desc,dateOfBirth`



# API Monitoring – Performance Metrics

## Application API Metrics

- Requests Per Minute (RPM)
- Latency
- Failure Rate

## Infrastructure API Metrics

- API Uptime
- Memory & CPU Usage

<https://www.moesif.com/blog/technical/api-metrics/API-Metrics-That-Every-Platform-Team-Should-be-Tracking/>  
<https://stackify.com/top-api-performance-metrics-every-development-team-should-use/>



# Why monitor APIs?

- 1. Proactive monitoring**
- 2. Measure the impact of performance improvements**
- 3. SLO monitoring**
- 4. Third-party SLA monitoring**



# API Performance - Compression using gzip

```
Accept-Encoding: gzip
User-Agent: my program (gzip)
```



# API Performance – Working with partial resources

Full response → Partial Response

Two types of partial requests:

- Partial response
- Patch

```
{  
  "kind": "demo",  
  ...  
  "items": [  
    {  
      "title": "First title",  
      "comment": "First comment.",  
      "characteristics": {  
        "length": "short",  
        "accuracy": "high",  
        "followers": ["Jo", "Will"],  
      },  
      "status": "active",  
      ...  
    },  
    {  
      "title": "Second title",  
      "comment": "Second comment.",  
      "characteristics": {  
        "length": "long",  
        "accuracy": "medium"  
        "followers": [ ],  
      },  
      "status": "pending",  
      ...  
    },  
    ...  
  ]  
}
```

```
{  
  "kind": "demo",  
  "items": [{  
    "title": "First title",  
    "characteristics": {  
      "length": "short"  
    }  
  }, {  
    "title": "Second title",  
    "characteristics": {  
      "length": "long"  
    }  
  },  
  ...  
]  
}
```



# API Performance – Partial Update

## Semantics of a patch request

The body of the patch request includes only the resource fields you want to modify. When you specify a field, you must include any enclosing parent objects, just as the enclosing parents are returned with a partial response. The modified data you send is merged into the data for the parent object, if there is one.

```
PATCH https://www.googleapis.com/demo/v1/324
Authorization: Bearer your_auth_token
Content-Type: application/json

{
  "title": "New title"
}
```

Response:

200 OK

```
{
  "title": "New title",
  "comment": "First comment.",
  "characteristics": {
    "length": "short",
    "accuracy": "high",
    "followers": ["Jo", "Will"],
  },
  "status": "active",
  ...
}
```

- Functional Testing
- API as Contract
  - ✓ Endpoints
  - ✓ Object model reflects Resources and their types
  - ✓ No missing/duplicate functionality
  - ✓ Relationship between resources are reflected
- API Test Plan
  - Manual testing or automated testing of the APIs

## Different types of test flows

1. Isolate Test requests
2. Multi-step workflow with several requests
3. Combined UI and API tests

API Call	Action
GET /users	List all users
GET /users?name={username}	Get user by username
GET /users/{id}	Get user by ID
Test Scenario Category	Test Action Category
Basic positive tests (happy paths)	<b>Verify:</b> <ul style="list-style-type: none"><li>- correct HTTP status code</li><li>- response payload</li><li>- response headers</li><li>- correct application state</li><li>- basic performance sanity</li></ul>
Extended positive testing with optional parameters	
Negative testing with valid input	
Negative testing with invalid input	
Destructive testing	



# Beyond API Functional Testing

- Security and Authorization
  - Security principles: deny-by-default, fail securely, least privilege principle, reject all illegal inputs
  - Positive: correct authorization → auth method → bearer token, cookies, digest
  - Negative: refuse unauthorized calls
- Role Permissions: Specific endpoints' call based on user role
- Protocol: HTTP → HTTPS
- Data leaks: Internal API data representations doesn't leak to public facing APIs
- Rate limiting, throttling and access control policies
- Performance
  - Load Test
  - Stress Test
- Usability Test



# API Monitoring

		Classifications					
Levels	NOT BASED ON APIs			PARTIALLY BASED ON APIs		FULLY BASED ON APIs	
	1	2	3	4	5	6	7
Isolated Applications	Unstructured Integrations	Component-based Architectures	Service-oriented Architectures	Private APIs based on Microservices	Open APIs	APIs as Business	
Monolithic App	Sockets File Sharing Msg Queues DB Replications	EJB RMI COM/COM+ CORBA WebServices	ESB BPEL CEP BAM SOAP	REST AMQP API Gateway Containers	API Platform Dev Portal WAF REST OAS	API Platform Service Mesh Cloud GraphQL gRPC	

# Rounding Up...

## API Security

- Schema validation
- Protocol conformance
- Protection against vulnerabilities
- Fraud prevention
- Defense against DoS & bad bots
- Payload inspection

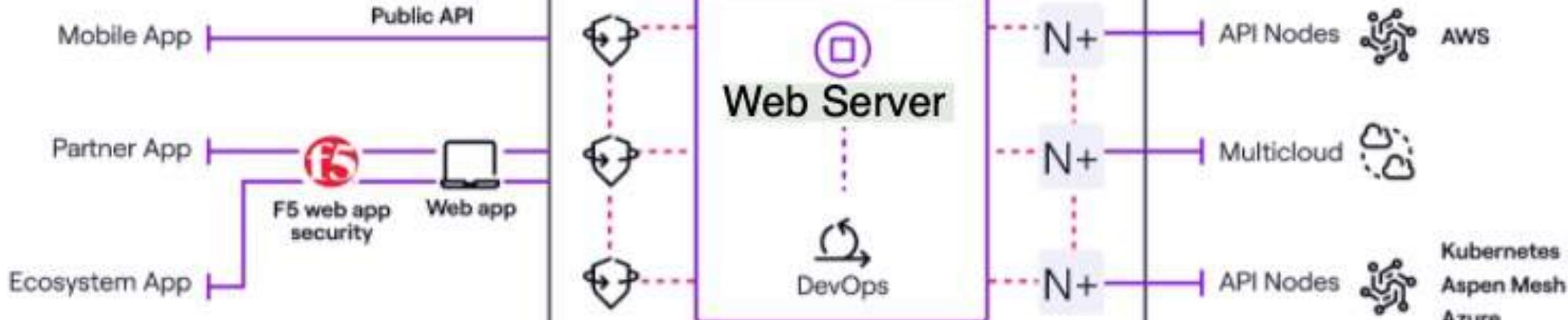
## API Management

- Versioning
- Publishing
- Schemas/definition
- Monitoring and dashboards
- Onboarding and docs

## API Gateway

- Authentication and authorization
- Traffic management
- Rate-limiting/thresholding
- Allow list
- Routing

## API Security and Management



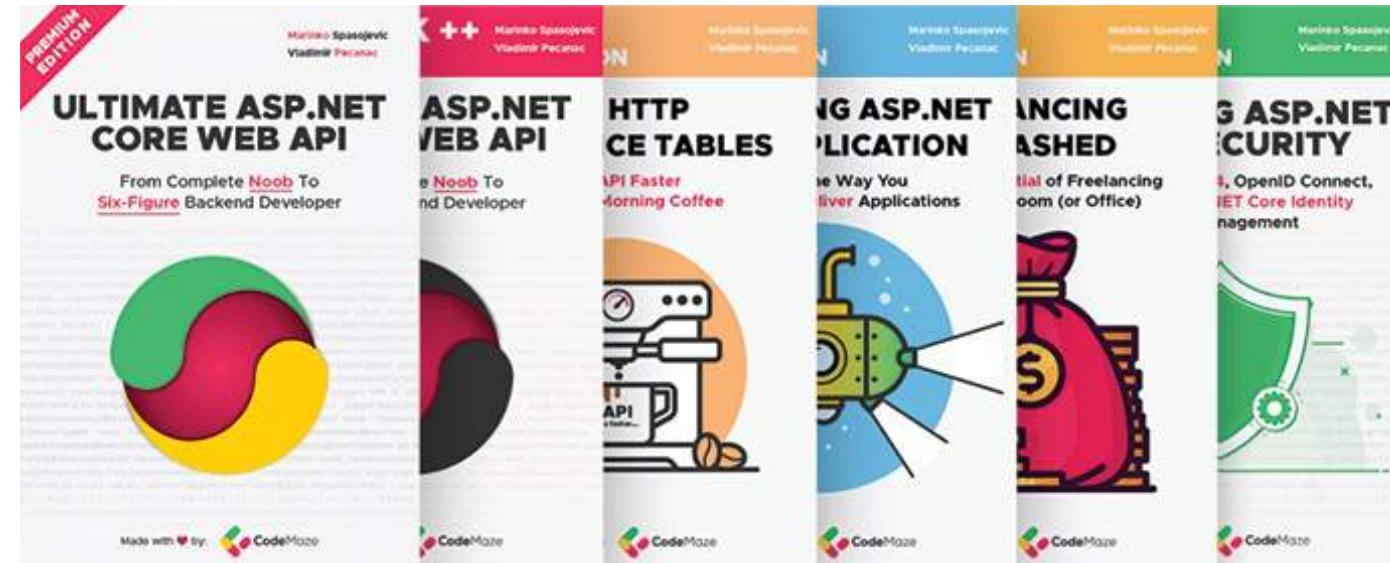


# Check out these resources...

## Security Overview of Amazon API Gateway

AWS Whitepaper

November 2020



GraphQL

<https://graphql.org/>



# Check out these platforms...

  
[Remove All](#)

  
**Butter CMS**  
HEADLESS CMS SOFTWARE  
  
[Visit Website](#)

  
**Sanity**  
HEADLESS CMS SOFTWARE  
  
[Visit Website](#)

  
**Strapi**  
HEADLESS CMS SOFTWARE  
  
[Visit Website](#)

  
**Agility CMS**  
HEADLESS CMS SOFTWARE  
  
[Visit Website](#)

## Overview

Summary	Butter CMS is an easy to use marketing dashboard and fast content API for modern apps. It offers a Global <a href="#">..read more</a>	Sanity is a management system which delivers high-quality content to digital devices and products. It provides <a href="#">..read more</a>	Strapi is an open-source headless CMS coded in JavaScript. Being open-source, it is a developer-focused <a href="#">..read more</a>	Agility CMS features a variety of flexible and fast developmental tools making it easier for the content <a href="#">..read more</a>
---------	---	--	---	--

## Features

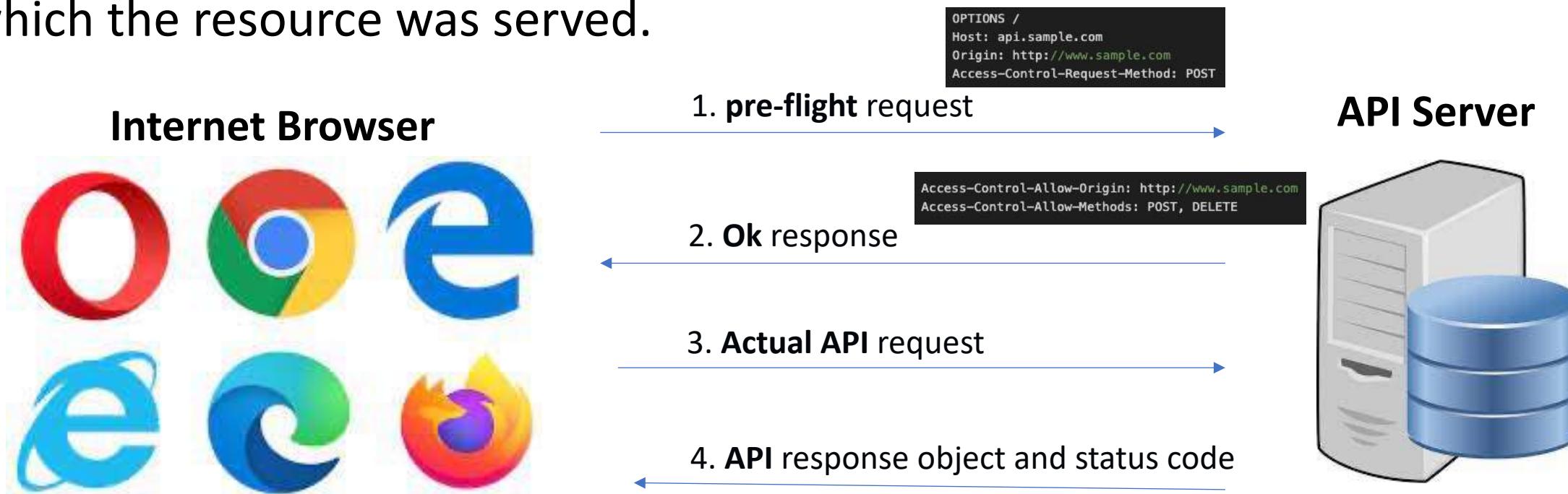
Access Control 	✓	✓	✓	✓
Approval Process Control 	✓	✓	✓	✓
Content Blocks 	✓	✓	✓	✓
Document Management 	✓	✓	✓	✓
Drag and Drop Editor 	✓	✓	✓	✓
Multi-language Support 	✓	✓	✓	✓



# API Security Design Pattern

OAuth	JWT	SAML
OAuth is a framework	JWT is a token in the JSON format.	SAML is a Single Sign-On login standard.
A user accessing an API resource from any client software such as browser-based web apps, native mobile apps, or desktop apps, can use OAuth for identification and authorization purposes.	JWT is one of the tokens that the client can present while using the OAuth framework. It is a JSON object which contains encoded data structure with information about the token issuer, subject (claims), expiration time, and more.	SAML allows logged in users to access applications by transferring the identity details from other sessions.
OAuth centralizes the authorization server, which allows the clients or third-party users to seek permission before accessing the information on a particular server.	JWT is one of the many security mechanisms that can be implemented as part of the OAuth Framework.	SAML uses an XML document, which is digitally encrypted, to provide authentication and deliver the desired message to the client.
OAuth framework specifies different protocols such as HTTP/HTTPS that can be used for presenting the tokens.	JWT specifies the token format in the form of JSON object definition (JWT, JWS, and JWE).	SAML strongly follows HTTP communication protocols and various other protocols such as SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), and more.
OAuth defines security patterns in which the client can obtain an access token from the Authorization Server.	JWT, on the other hand, is independent of the security pattern that the client uses to obtain or present the token.	SAML allows only one computer in the network to perform the security check and share credentials seamlessly with all the connected computers.

CORS is HTTP header-based mechanism that allows restricted resources to be requested from another domain outside the origin domain from which the resource was served.



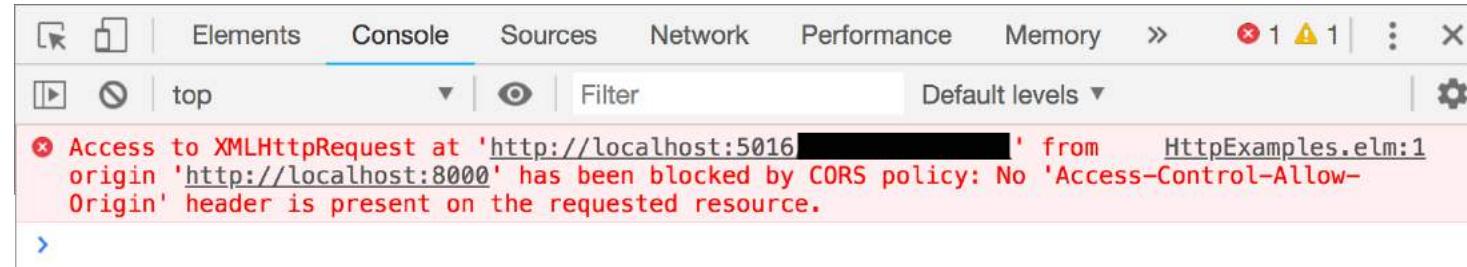
## Using Wildcard (\*) Origins

Unless the request is a public resource and is intended to be used by everyone, you should try to be as specific as possible about the origins of the requests, otherwise you will lose the benefit of implementing CORS in your applications.



# API Security – Handling CORS Error

1. **Server CORS Policies** did not allow the specific resource to be served based on the domain, scheme, port, header, credential or method.
2. **Error on Server:** There was an error on the server, and preflight request failed because the server couldn't serve the pre-flight request.



3. **API Gateway:** Although your web server is sending acceptance back in response to pre-flight requests, your API gateway is not configured properly to relay those responses to the browser.
4. **Credentials:** need to be sent such as cookies or Http Authentication.

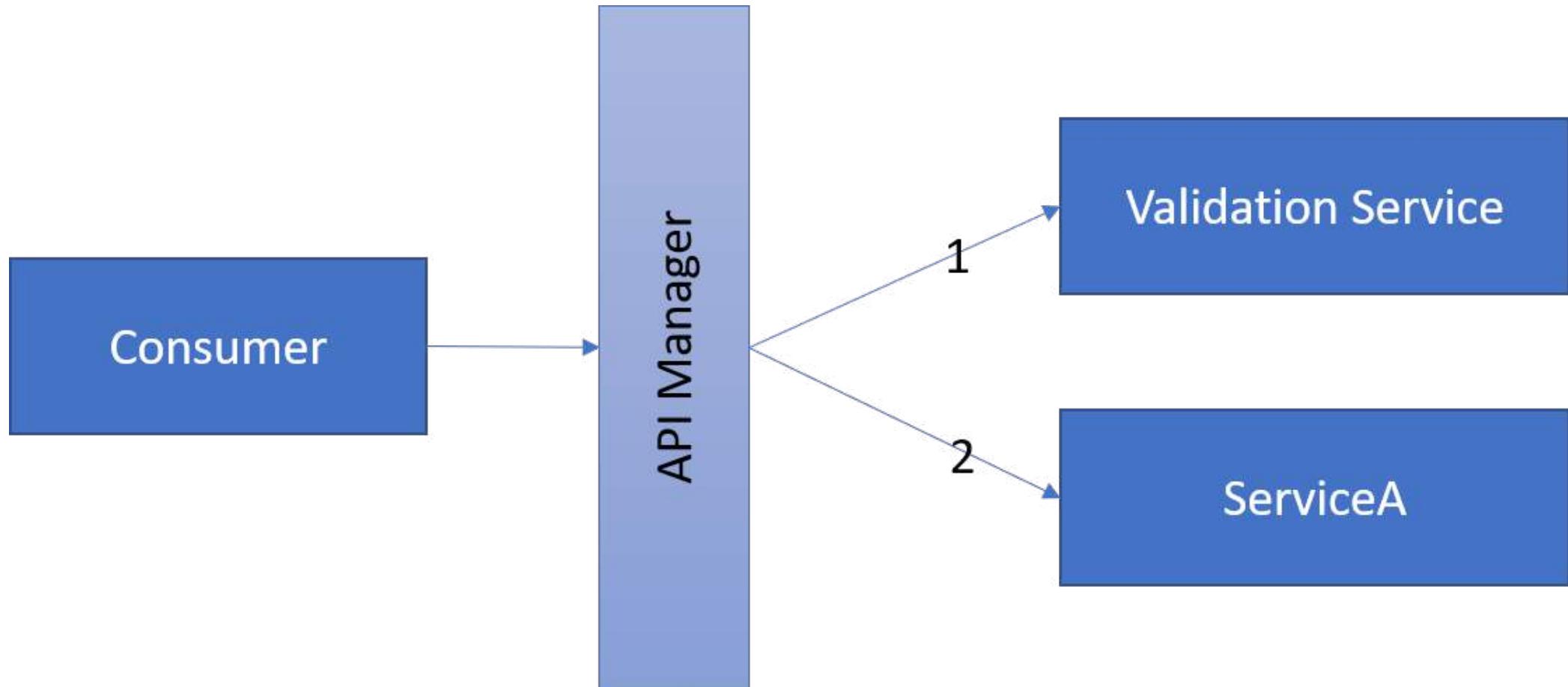
## Validation policies

- [Validate content](#) - Validates the size or JSON schema of a request or response body against the API schema.
- [Validate parameters](#) - Validates the request header, query, or path parameters against the API schema.
- [Validate headers](#) - Validates the response headers against the API schema.
- [Validate status code](#) - Validates the HTTP status codes in responses against the API schema.

```
<validate-parameters specified-parameter-action="prevent" unspecified-parameter-action="prevent" errors-variable-name="requestParametersValidation">
    <headers specified-parameter-action="detect" unspecified-parameter-action="detect">
        <parameter name="Authorization" action="prevent" />
        <parameter name="User-Agent" action="ignore" />
        <parameter name="Host" action="ignore" />
        <parameter name="Referrer" action="ignore" />
    </headers>
</validate-parameters>
```



# API Orchestration





# Transaction Management in Microservices

- Avoiding transactions across Microservices
- Two-Phase Commit Protocol
- XA Standard
- REST-AT Standard Draft
- Eventual Consistency and Compensation

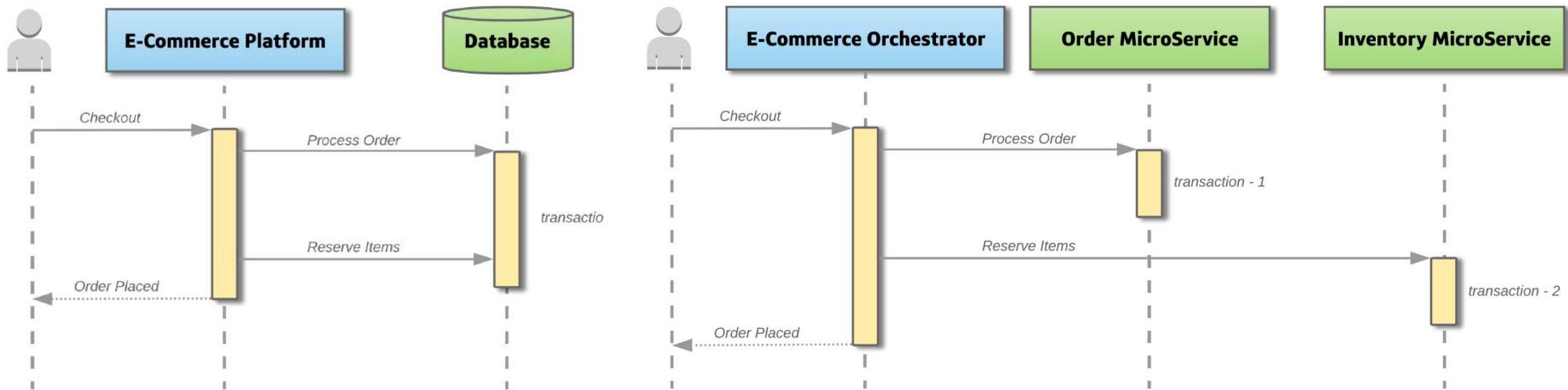
## Service A

- ✓ Call API1 of service B
- ✓ Call API2 of service B
- ✓ Update something in DB
- ✓ Call API1 of service C
- ✓ Update something in DB
- ✓ Call API1 of service D
- ✗ Update something in DB

## Service A

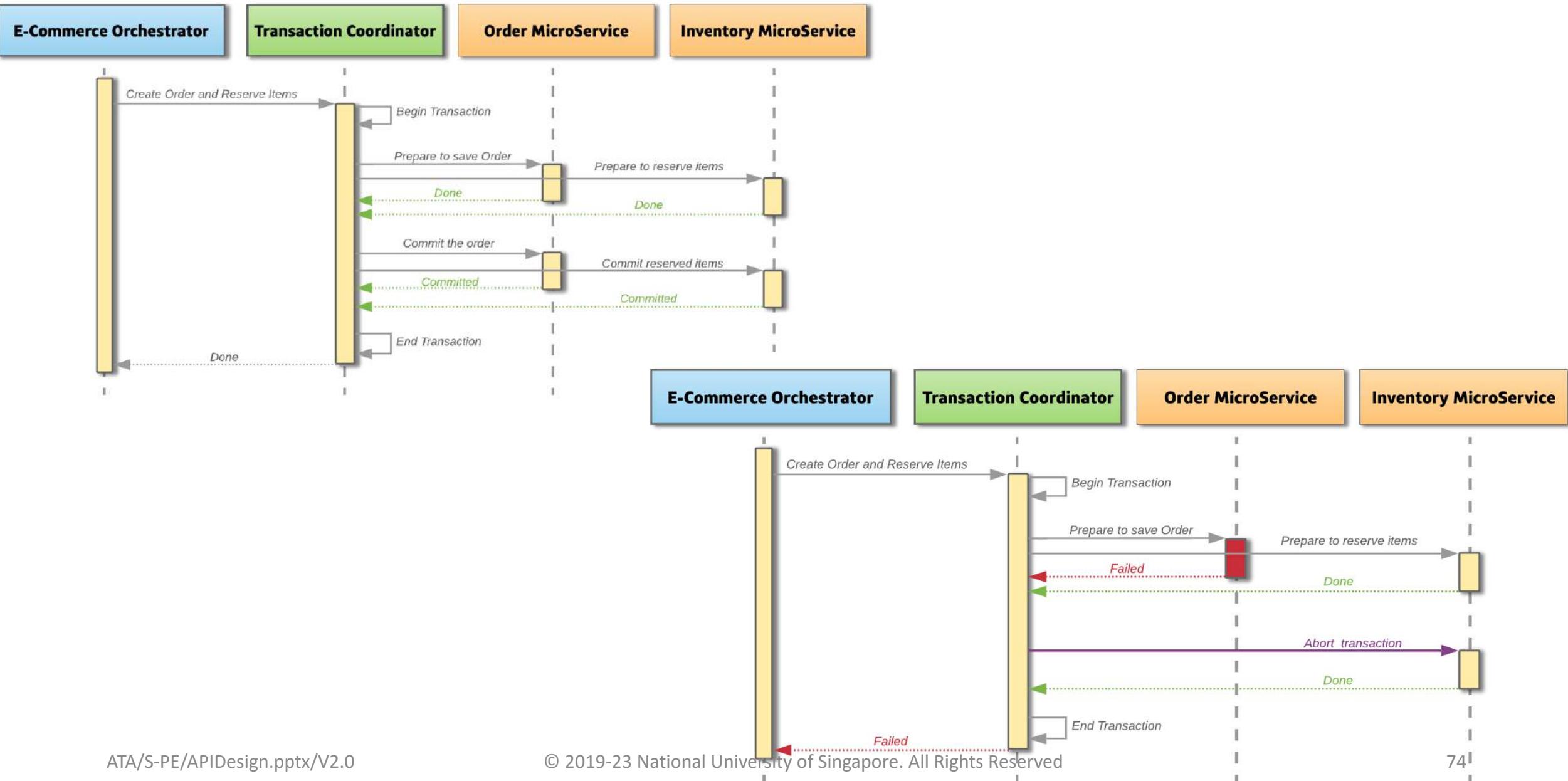
- ✓ Call API1 of service B ?
- ✓ Call API2 of service B ?
- ✓ Update something in DB ↗
- ✓ Call API1 of service C ?
- ✓ Update something in DB ↗
- ✓ Call API1 of service D ?
- ✗ Update something in DB

# Handling Distributed Transactions in the Microservice - I





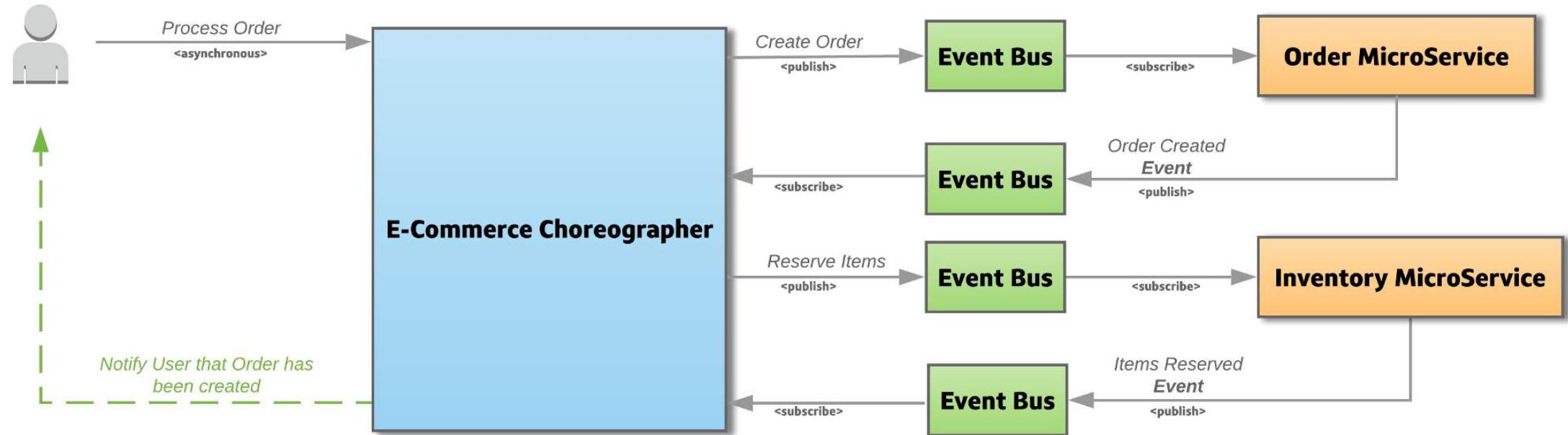
# Two-Phase Commit





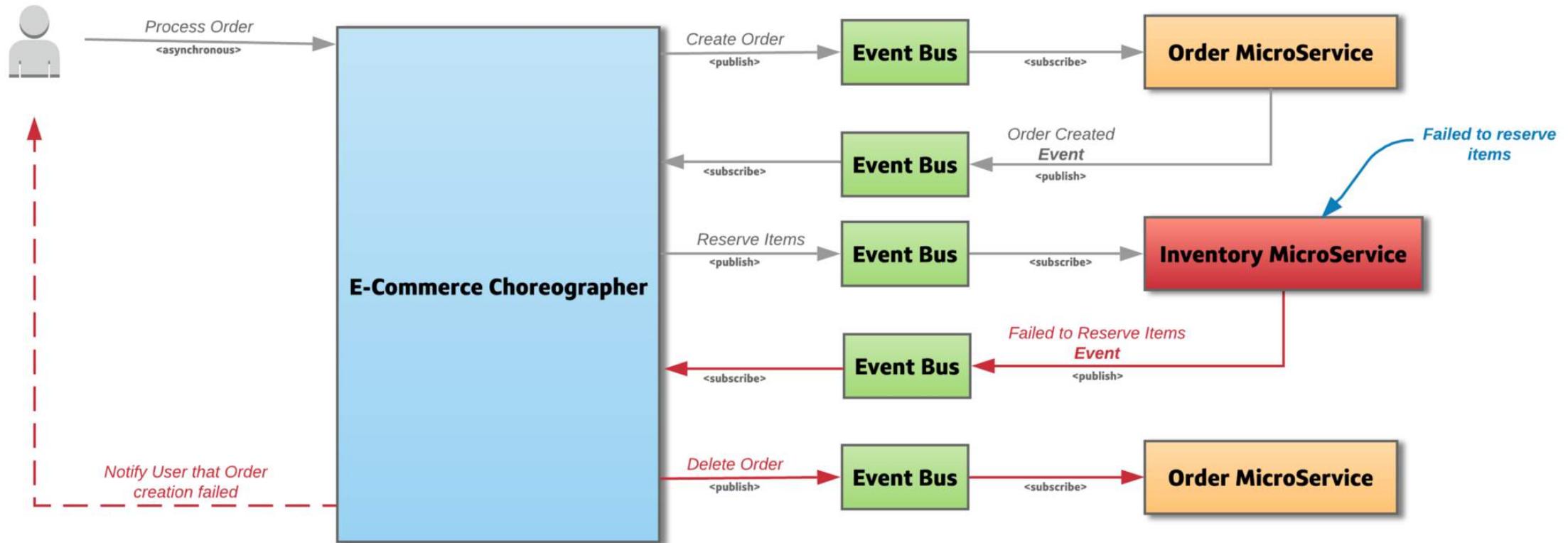
# Eventual Consistency and Compensation

## SAGA - I



# Eventual Consistency and Compensation

## SAGA - II



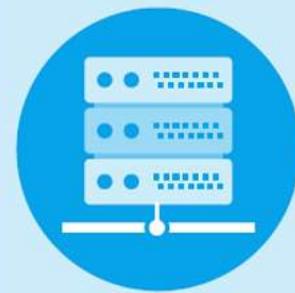


# Risk Approach For Managing APIs - I

ARCHITECTURE	THREAT SCENARIO	MITIGATION MEASURES
FRONT END 	► <b>Untrusted clients</b> could lead to insecure connections risking data theft	<ul style="list-style-type: none"><li>► Ensure authentication is based on <b>multi-factor authentication (MFA)</b>.</li><li>► Use <b>industry-recognised authentication standards</b> such as Simple Authentication Markup Language (SAML) and Open ID Connect (OIDC) protocols as they have undergone scrutiny and have been widely adopted.</li></ul>

# Risk Approach For Managing APIs - II

## TRANSACTION EXCHANGE (API GATEWAY AS BROKER)



► **Insecure interfaces** are prone to data leakage due to sniffing by Attackers (i.e. man-in-the-middle attacks)

► **Brute force or Denial of Service attack** that may risk the availability of services providing APIs

- **Data-in-transit remains encrypted** to mitigate risk whereby an Attacker may sniff the data through the network. Use the latest recommended version of Transport Layer Security (TLS) for encrypting network traffic (e.g. TLSv1.2 and above).
- **Define, configure and test page limit specific to the system/API.** Page limiting is setting a maximum number of records which an API can return per request. Setting thresholds should undergo regular load performance testing as demand increases for use cases, and be justified with verifiable evidence (e.g. based on past statistics, Service Level Agreement (SLA)).
- **Define, configure and test rate limit specific to the system/API.** This limits client requests for API resources to sustain back end system availability. Adopt Content Delivery Network (CDN) or equivalent services to manage demand surges expected for upcoming key events/activities.
- **Monitor API transactions** and look out for suspicious patterns such as querying attempts of certain restricted APIs outside of expected geolocations and frequent API validation failures which could potentially be some sort of brute force attack.



# Risk Approach For Managing APIs - III



- ▶ **Unauthorised access** of APIs and/or data could lead to data breaches and could also come in the form of an insider threat
- ▶ **Excessive data exposure** often leads to violation of data sensitivity
- ▶ **Security misconfiguration** (when security configuration is not optimised to requirements, e.g. using default configurations) leads to exposure of sensitive back end information such as server version

- ▶ **Validate and log APIs** that comprise valid telemetry such as protocol conformance, request, access scheme and location. Ensure services consuming APIs and services providing APIs are mutually authenticated such that their communication is legitimate.
- ▶ **Implement role-based access control (RBAC) and attribute-based access control (ABAC)** to manage access rights of clients to APIs and to request for data/service through the APIs based on principle of least privilege.
- ▶ **Conduct periodic reviews on roles and attributes** to ensure dormant accounts and unnecessary access to inactive clients are promptly removed.
- ▶ **Review security configuration** such that default values are changed according to industry-recognised hardening guidelines and standards to improve security posture.
- ▶ **Perform security testing** periodically and whenever a change is introduced to the APIs. The test coverage should be “end to end” (i.e. from front end to transaction to back end) to identify new vulnerabilities or misconfiguration for early remediation.



# Self-check...

What protocol does REST follow?

- a. FTP
- b. HTTP
- c. SFTP
- d. FTPS

HTTP Code 200 represents which among the following?

- a. Success
- b. Completed
- c. Failure
- d. Warning

Which of the following options are true for REST Services?

- a. Each resource can be identified by multiple URIs.
- b. REST supports only JSON representation.
- c. In REST Services, the client gets access to resources and the server provides access to them.
- d. All of the above.

What method should be used to obtain a list of supported operations in REST services?

- a. GET
- b. DELETE
- c. HEAD
- d. OPTION

What category does the 5xx HTTP code belong to?

- a. Redirection
- b. Warning
- c. Client Error
- d. Server Error

Is it possible to maintain sessions in REST on the server-side?

- a. No
- b. Yes
- c. Depends on situation
- d. I don't know

Which among the below directives belonging to the Cache-Control header of HTTP response provide information to the server that the resources have to be revalidated if max-age has crossed?

- a. no-store
- b. must-revalidate
- c. no-cache
- d. None of the above

What category do 1xx HTTP status codes belong to?

- a. Server Error
- b. Redirection
- c. Client Error
- d. Informational

What does the status code 302 represent?

- a. User can select among multiple links and go to different page.
- b. Not Modified
- c. The resource requested has been found and moved temporarily to new URL location.
- d. The page requested is available only by means of proxy address given in the response.

What constraint is not a strict requirement for a service to be called a RESTful web service?

- a. Uniform Interface
- b. Code on Demand
- c. Stateless
- d. Client-Server
- e. All of the above

Which component of HTTP response has the metadata for stating the type of response message is in the form of key-value pairs?

- a. Status Code
- b. Response Body
- c. HTTP Version
- d. Response Header

Which directive of the Cache-control header in the HTTP Response tells that the resource cannot be cached?

- a. public
- b. max-age
- c. no-cache/no-store
- d. private

# 4. REUSABLE SERVICES AND API GATEWAY

Boon Kui HENG

[boonkui.heng@nus.edu.sg](mailto:boonkui.heng@nus.edu.sg)

Total Slides: 76

# Agenda

- Objectives
  - To architect and choreograph reusable services
  - To expose SDKs for consuming APIs
- Topics
  - Characteristics of Microservices
  - Decoupling Reusable Services
    - Context Mappings
    - Aggregates
    - Domain Events
    - Workshop: Design Aggregates and Domain Events
  - Orchestration vs. Choreography Reusable Services
  - API Gateways
  - SDKs for Consuming Services

# Topics

- **Characteristics of Microservices**
- Decoupling Reusable Services
  - Context Mappings
  - Aggregates
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- SDKs for Consuming Services

# Characteristics of Microservices

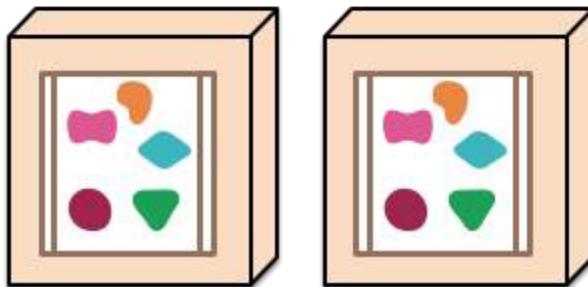
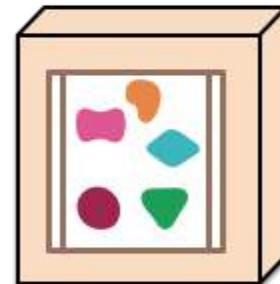
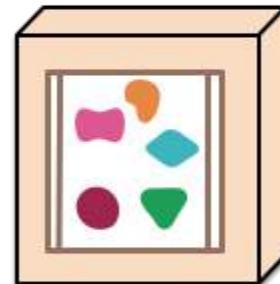
- Not going to discuss the concepts of microservices in details
- Shall highlight the characteristics of microservices
- Shall later explore a few engineering aspects of microservices
- The **microservice** architectural style is an approach to developing a single application as a **suite of small services**, each running in its **own process** and **communicating with lightweight mechanisms**, often an HTTP resource API. These services are built around **business capabilities** and **independently deployable** by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in **different programming languages** and use **different data storage technologies**.

# Componentization via Services-1

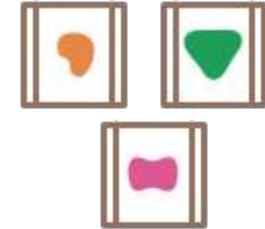
*A monolithic application puts all its functionality into a single process...*



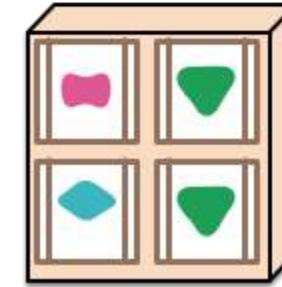
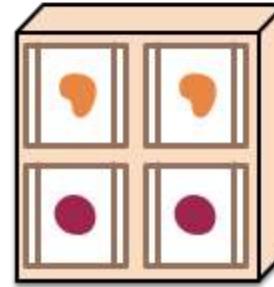
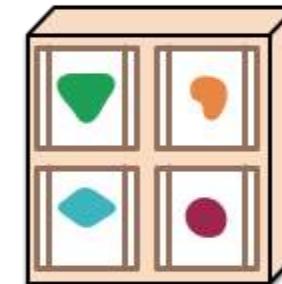
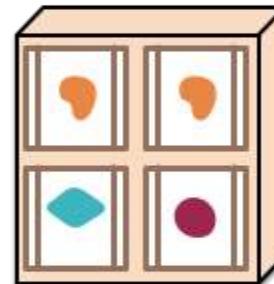
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



*... and scales by distributing these services across servers, replicating as needed.*



# Componentization via Services–2.

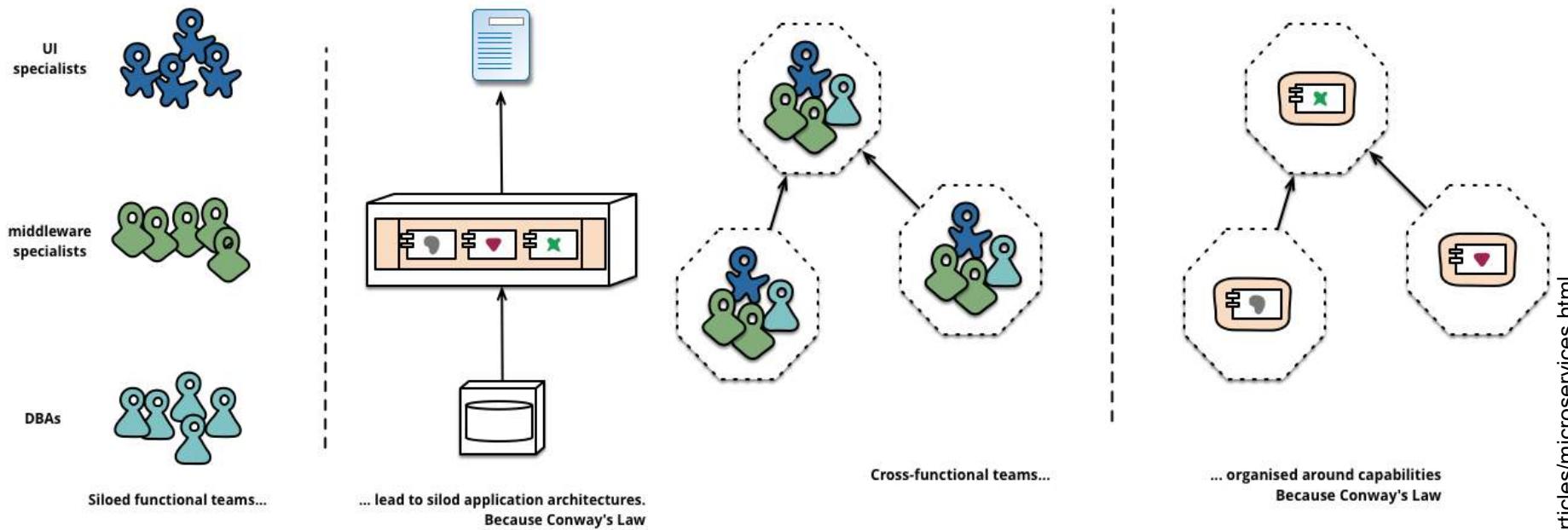
## Advantages

- Services as components are **independently deployable**
  - While **shared libraries** as components are **not**
  - Shared libraries are **linked into a program** and called using **in-memory function calls**, while services are **out-of-process** components who communicate with a mechanism such as a **web service request**, or remote procedure call
- More **explicit component interface**
  - Avoid overly tight coupling between components
  - Preserve the encapsulation of components

## Disadvantages

- Remote calls are **more expensive** than in-process calls, and thus remote APIs need to be **coarser-grained**, which is often more awkward to use
- If you need to change the **allocation of responsibilities** between components, such **movements of behavior** are **harder** to do when you're crossing process boundaries

# Organized around Business Capabilities–1



*Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.  
-- Melvyn Conway, 1967*

## Siloed Functional Team

- Simple changes can lead to a **cross-team project** taking time and budgetary approval
  - A team may work around by **forcing the logic** into whichever application it has access to
- Monolith application can also be **split along business lines** but it requires a **great deal of discipline**

## Cross Functional Team

- Is cross-functional, including the **full range of skills** required for the development of a **service**: user-experience, database, and project management

# Smart Endpoints and Dumb Pipes

## Traditional (e.g. Enterprise Service Bus)

- Put significant **smarts** into the **communication mechanism** itself
  - Enterprise Service Bus (ESB) products often include **sophisticated facilities** for message routing, choreography, transformation, and applying business rules
- Choreographed using **complex protocols** such as WS-Choreography or BPEL or orchestration by a **central tool**

## Smart Endpoints and Dumb Pipes

- Microservices aim to be as **decoupled** and as **cohesive** as possible
- Put significant **smarts** into the **endpoints**
  - Messaging over a **lightweight message bus that is typically dumb** (being a message router only)
- Choreographed using **simple protocols** like REST
  - Used resources can be **easily cached**

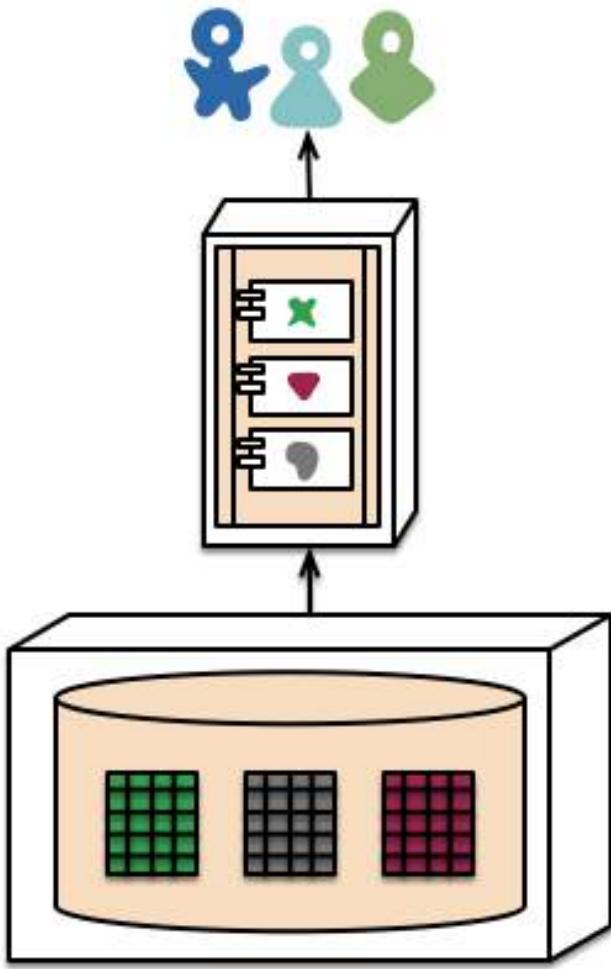
## Centralized Governance

- Tendency to standardise on **single technology** platforms

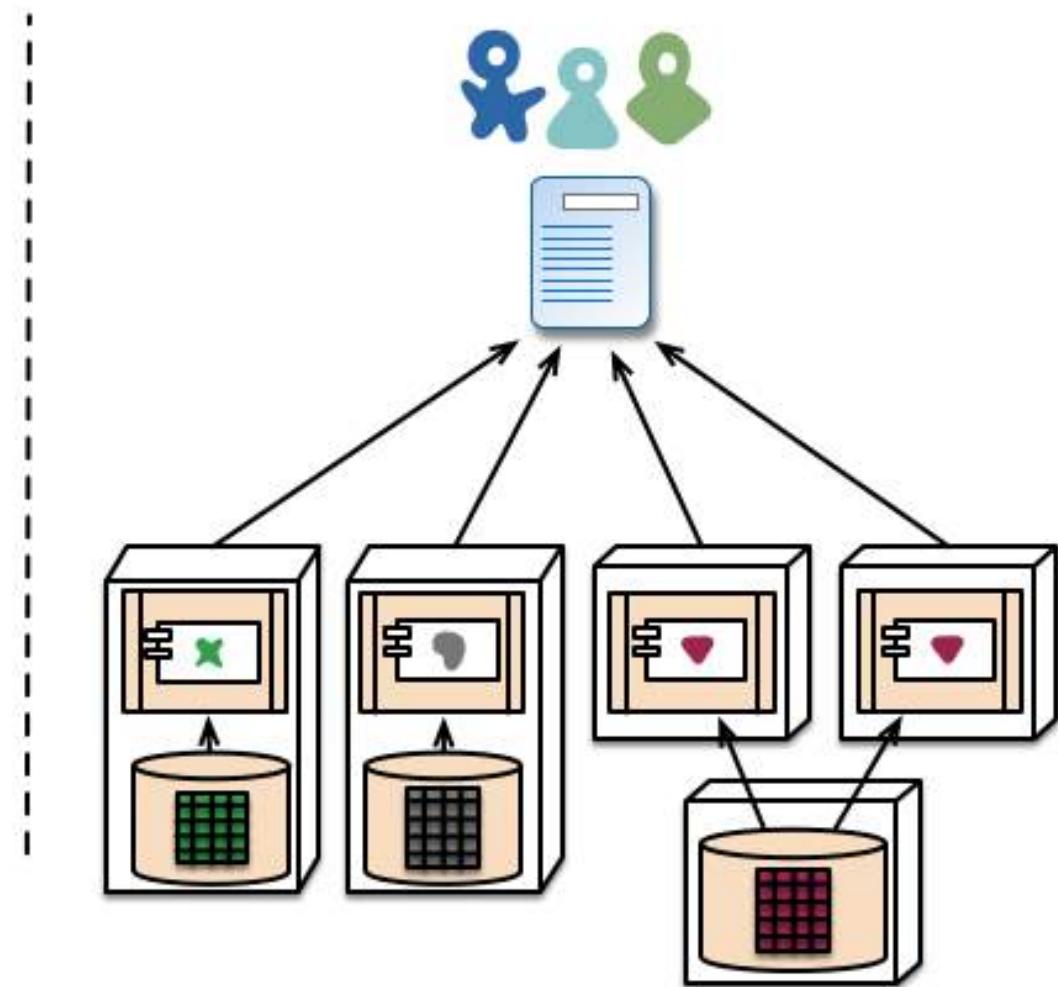
## Decentralized Governance

- Choice of **different technology** platforms for different microservices
- “**build it / run it**” ethos
  - Teams are responsible for **all aspects** of the software they build including **operating** the software 24/7

# Decentralized Data Management-1



monolith - single database



microservices - application databases

# Decentralized Data Management–2.

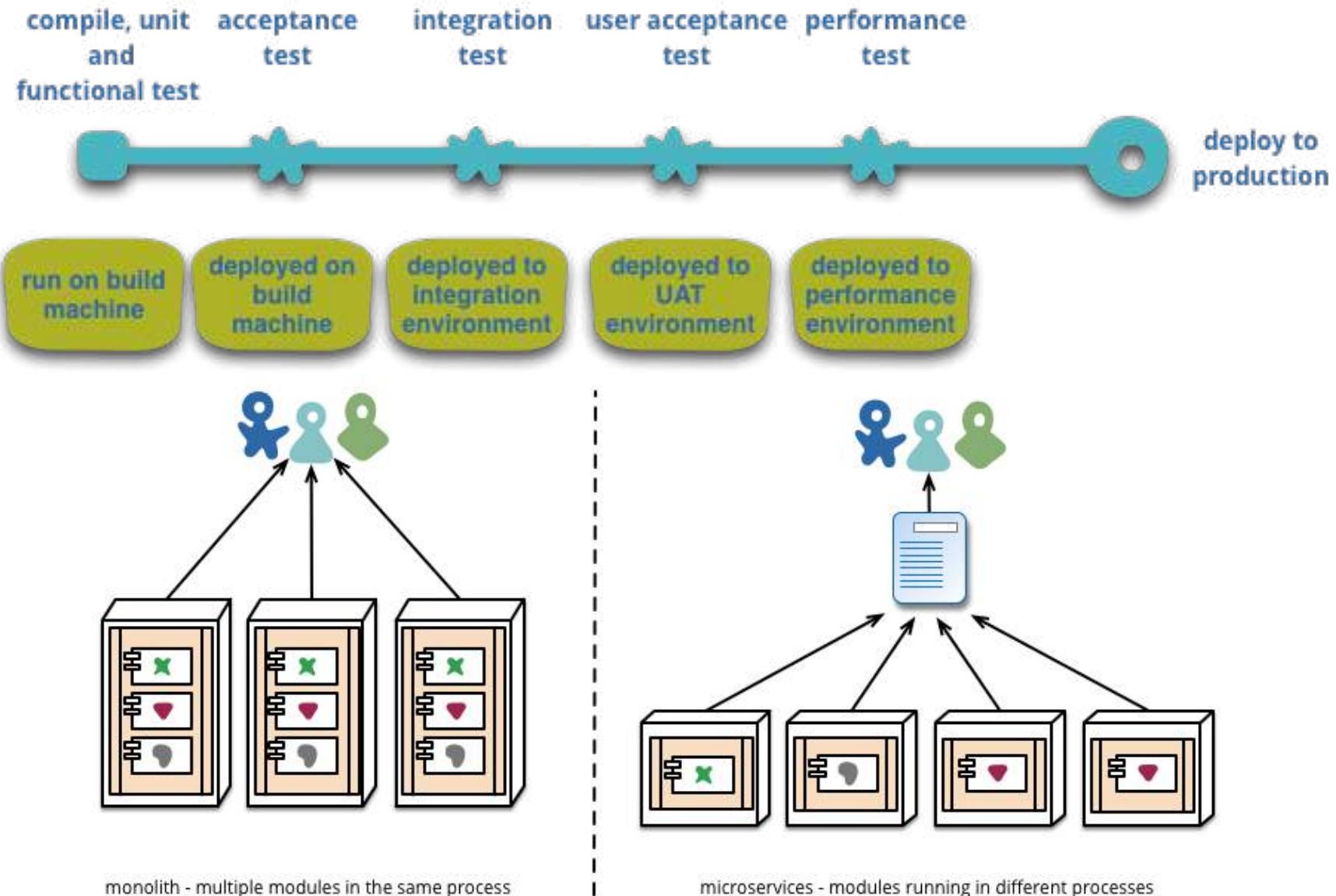
## Monolithic

- Prefer a **single database** over a range of applications
- Rely on **transactions** to guarantee **consistency** when updating multiple resources
  - **Slower response** to business demands

## Microservices

- Prefer **polyglot persistence**
  - Each service manages its **own database** – either **different instances** of the same database technology or **entirely different database systems**
- Rely on **transactionless** coordination between services with **eventual consistency**
  - Coupled with **compensating operations** to deal with inconsistency and mistakes
  - **Quicker response** to business demands, normally more closely **matches the business practice**

# Infrastructure Automation-1



# Infrastructure Automation–2.

- Most teams building microservices use **continuous integration** (CI) and **continuous delivery** (CD)
  - Which apply well to monolithic systems as well
- The build pipeline features **automated tests** and **deployments**
- The **management of microservices** in production is normally **automated** as well

# Design to Handle Failure.

- Microservices applications are normally designed to **tolerate the failure of services**
- Any service call could fail due to **unavailability of the supplier**, the **client** has to respond to this as **gracefully as possible**, mitigating the impact on **user experience**
- Rely **on real-time monitoring** for warning signs on potential **anomalies**
  - Architectural elements (e.g. database throughput)
  - Business metrics (e.g. # orders per minute)

# Topics

- Characteristics of Microservices
- **Decoupling Reusable Services**
  - Context Mappings
  - Aggregates
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- SDKs for Consuming Services

# Decoupling Reusable Services

- Recall that **Bounded Contexts** and **Ubiquitous Languages** help to **identify reusable services** for a platform
- However, a BC **integrates** with other BCs in **multiple ways**
  - A BC may make use of a **domain concept** that belongs to another BC
  - A BC may need to be aware of **events** that occur on a domain concept in another BC
- Such integration between BCs results in **couplings** between BCs, the couplings need to be **appropriately designed** so that the BCs are not unnecessarily coupled
- The **Context Mappings** (CM), **Aggregates** and **Domain Events** (DE) techniques of DDD deal with the design of such couplings

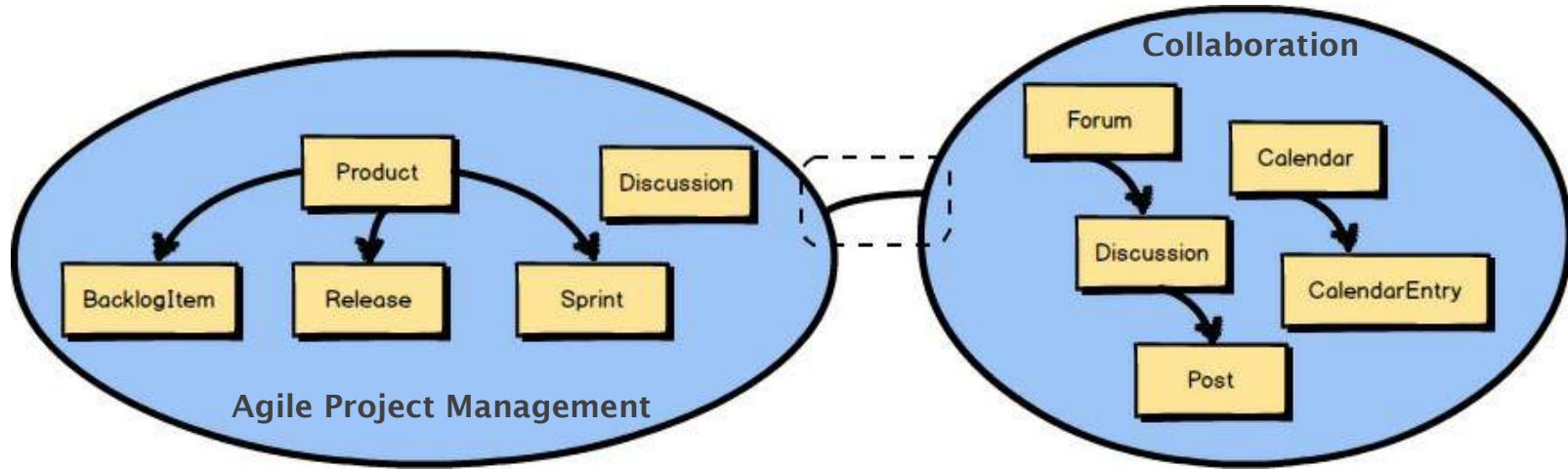
# Issues with Decoupling Reusable Services

- How to **integrate** between BCs?
- What are the **strategies** for teams in charge of BCs to collaborate?
- What are the **types** of communication between BCs?

# Topics

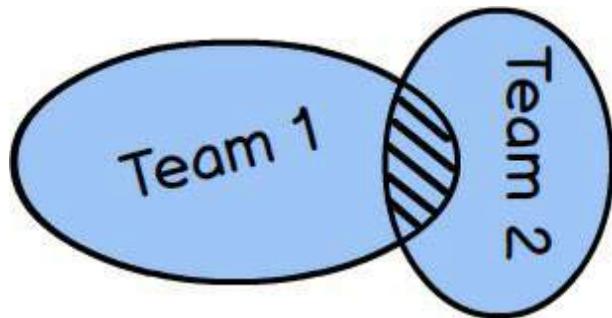
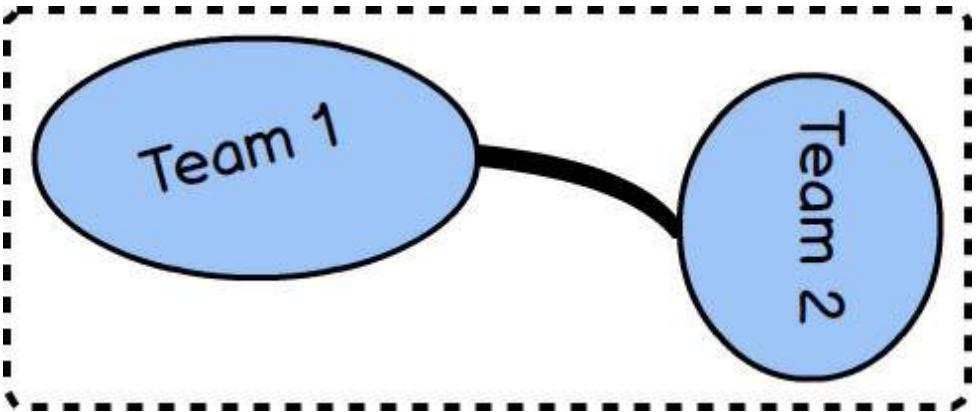
- Characteristics of Microservices
- **Decoupling Reusable Services**
  - **Context Mappings**
  - Aggregates
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- SDKs for Consuming Services

# Context Mapping



- The Agile Project Management context uses the Discussion concept of a **collaboration context**
- Such integration is represented as a **context mapping** between the two contexts
  - The collaboration context is the **source** of the concept
  - The Agile Project Management context is the **consumer** of the concept
- As the two contexts have two **different ULs**, the **context mapping has to mediate** between the two contexts via various means

# Kinds of Context Mappings-1



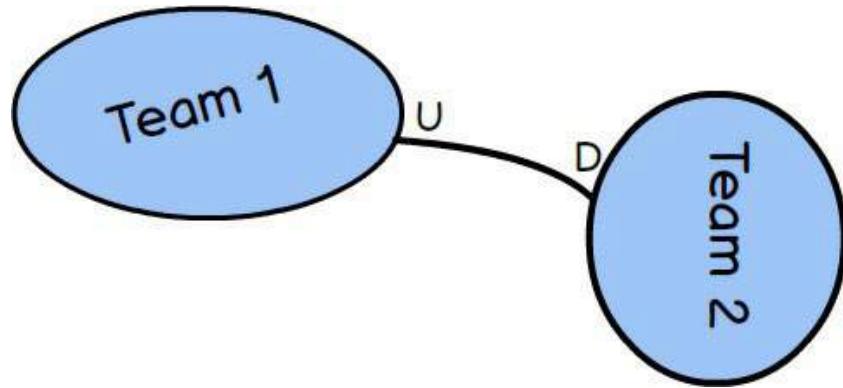
## Partnership

- Both teams **work closely** to **continuously align** the integrations between the contexts
- Very high **commitments** are required from both teams to align **work schedules** and **dependent work**

## Shared Kernel

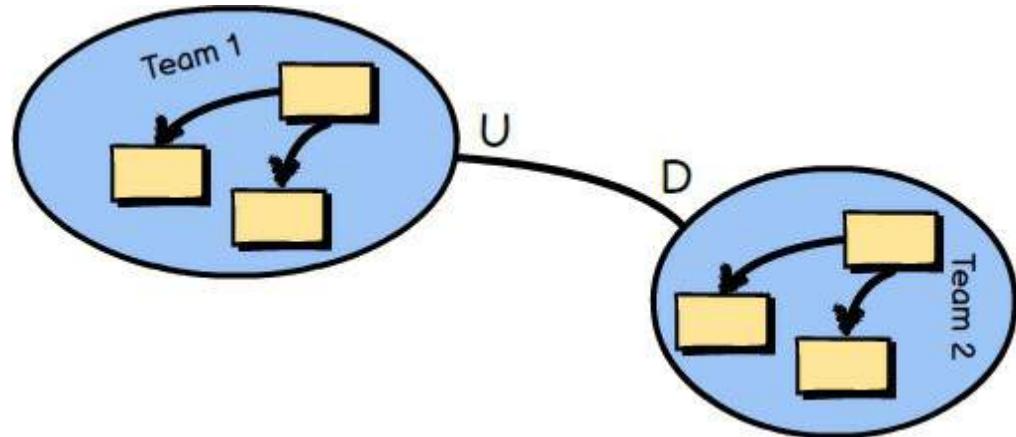
- Both teams share a **small but common model**
- Often very **difficult to conceive** and **maintain**
  - May be maintained by one of the teams
- Require **open communication** and **constant agreement** between both teams

## Kinds of Context Mappings-2



### Customer-Supplier

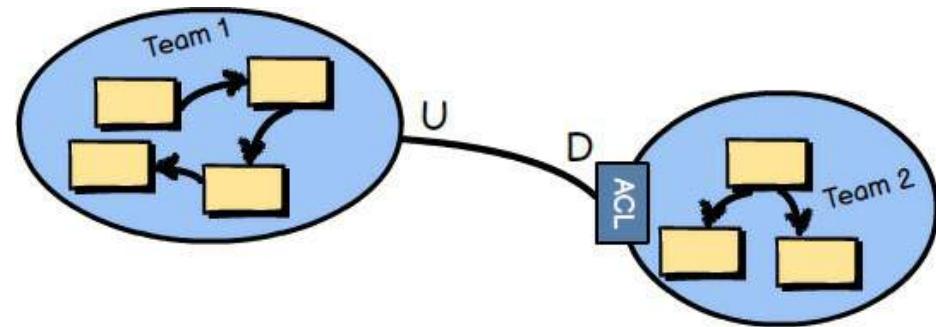
- The **supplier** is the **upstream** (U) while the **customer** is the **downstream** (D)
- Supplier **tries to satisfy** customer's requests but has the **final say**
- Very **typical and practical** as long as supplier does **not** become **completely autonomous** and **unresponsive** to customer



### Conformist

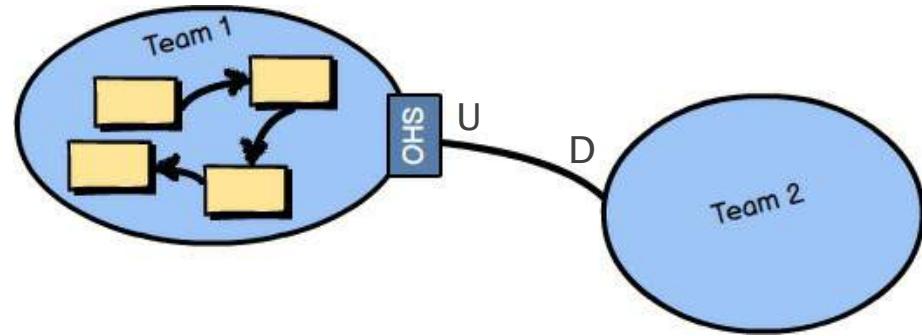
- A special case of Customer-Supplier where
  - The **upstream** team has **no motivation** to support the specific needs of the downstream team
  - The **downstream** team **cannot sustain an effort** to translate the UL of the upstream model to fits its specific need, so the downstream team **conforms** to the upstream model
- Such **upstream** model is usually very **large and complex** model that is **well established**

# Kinds of Context Mappings–3



## Anticorruption Layer

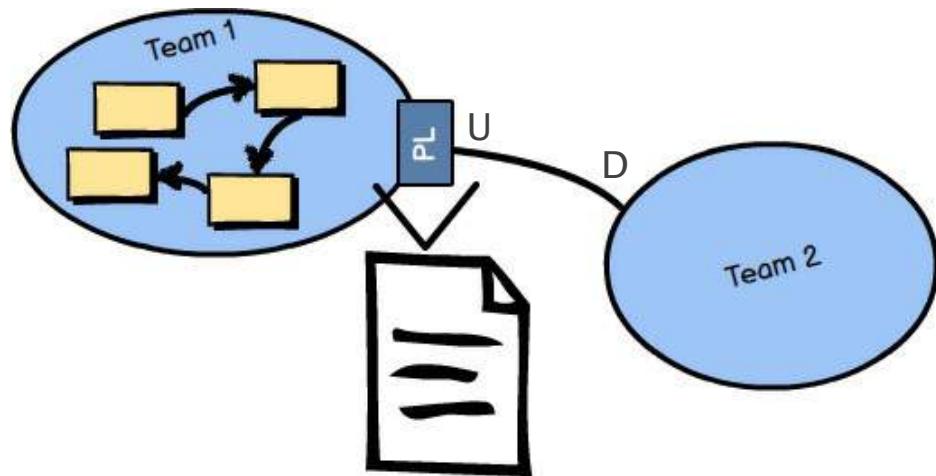
- The **downstream** team **translates** from the upstream UL to its UL
- The **translation layer isolates** the downstream model from the upstream model
- The **cost of maintaining** the translation layer could be **too high**
- A **great way** to integrate with an upstream **Big Ball of Mud** (legacy system)



## Open Host Service

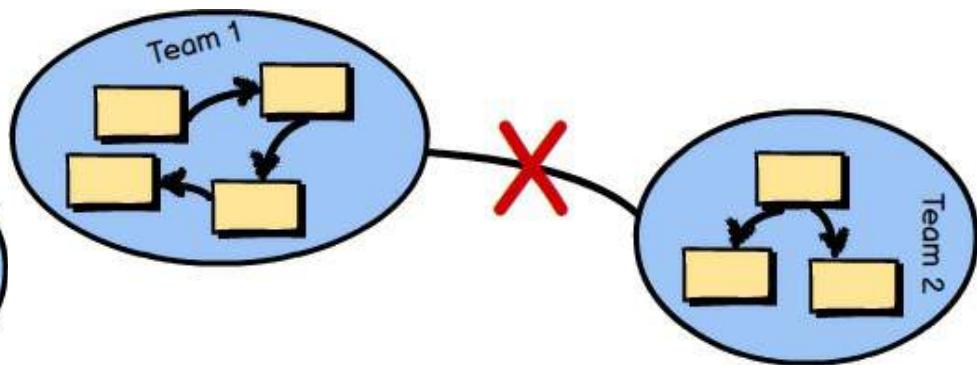
- The **upstream** team creates a **protocol or interface** that give access to its context as a set of **services**
- It is “**open**” as other contexts can **consume** with it with **relative ease**
- **Less crucial** for the downstream team to have an **anticorruption layer**

# Kinds of Context Mappings–4.



## Published Language

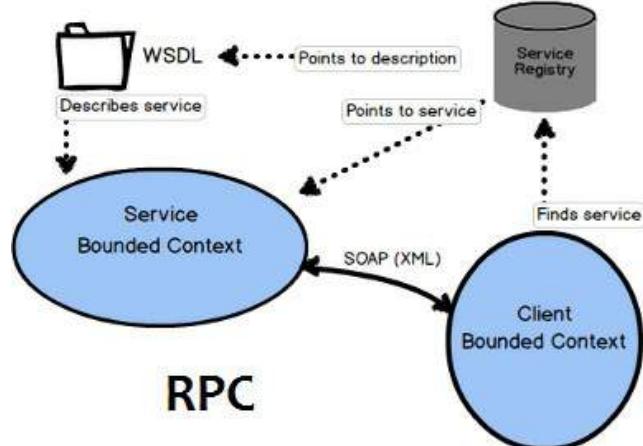
- The **upstream** team publishes its interface using a **well-documented information exchange language**
  - E.g. XML Schema, JSON Scheme
- The **downstream** team easily **consumes / translates** from the upstream UL to its UL
- **Open Host Service** often provides a published language



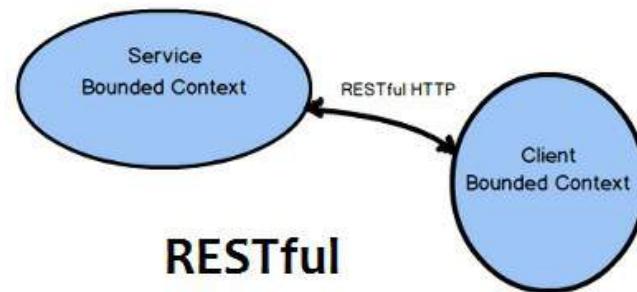
## Separate Ways

- Situation where integration between the contexts will **not** produce **significant payoff**
  - E.g. the functionality sought is not fully provided by the source UL
- Both teams use their **own specialized solutions** without attempting to integrate

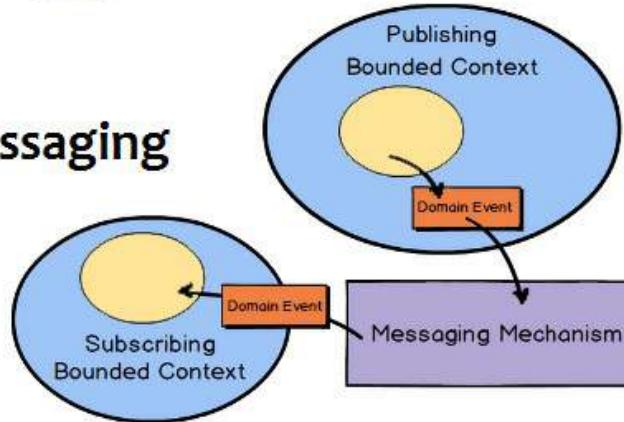
# Types of Integration



**RESTful**



**Messaging**



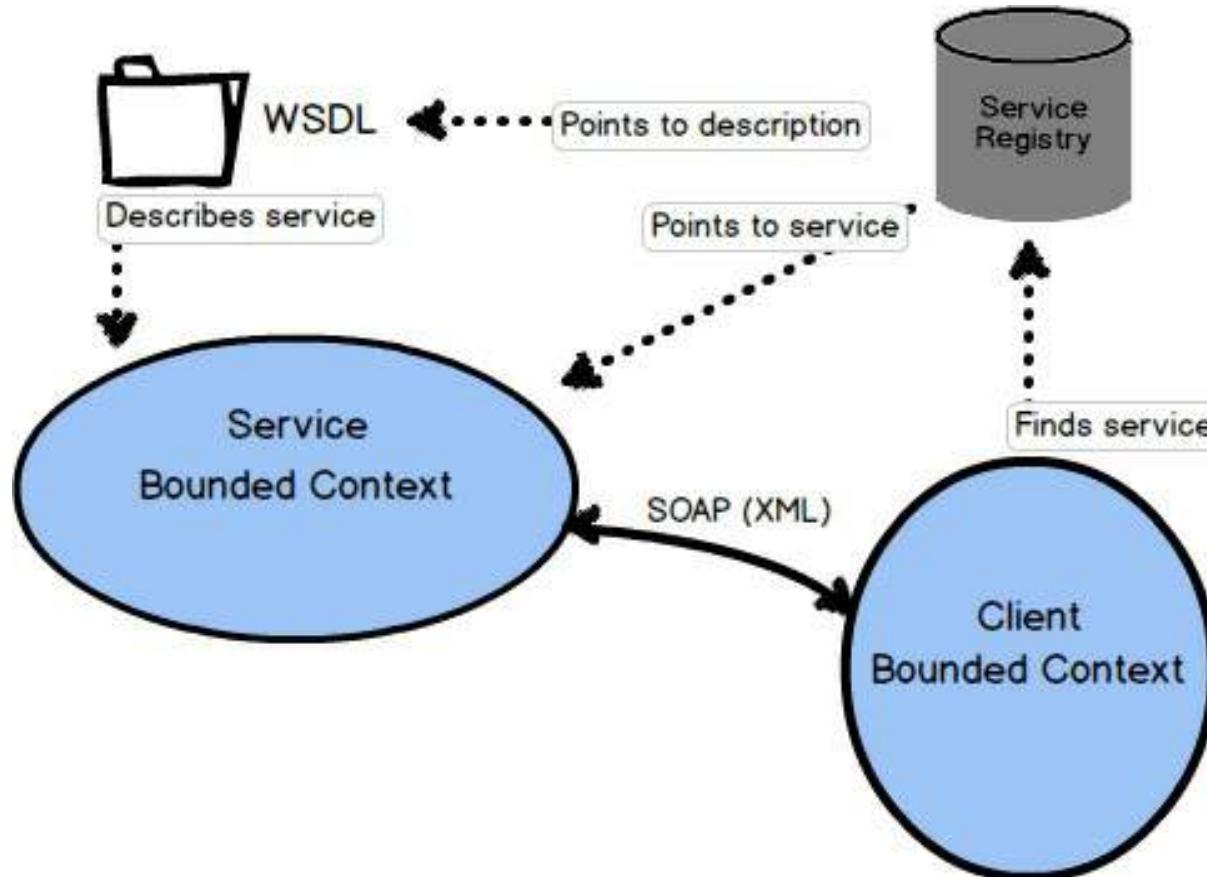
## Typical

- RPC via SOAP
- RESTful
- Messaging

## Best Avoided

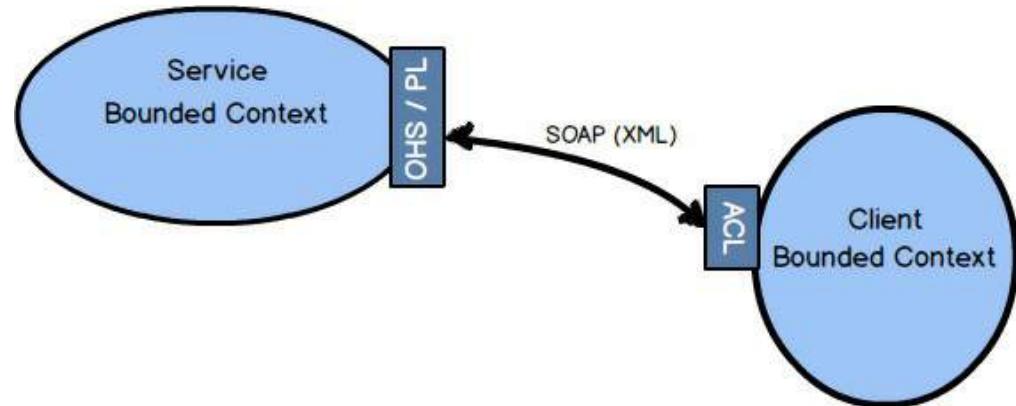
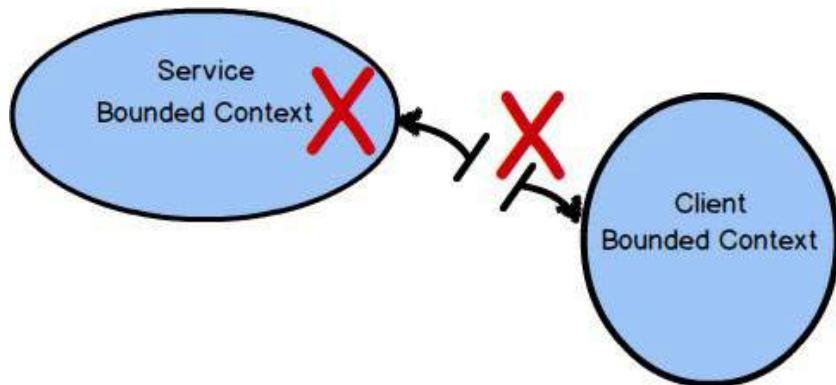
- Database
- File system

# RPC with SOAP-1



- **Simple** for the **client** to call, just like a local procedure
- But the request has to **travel** over the network, **reach** the remote system, **perform** successfully, and **return** results over the network

# RPC with SOAP–2.



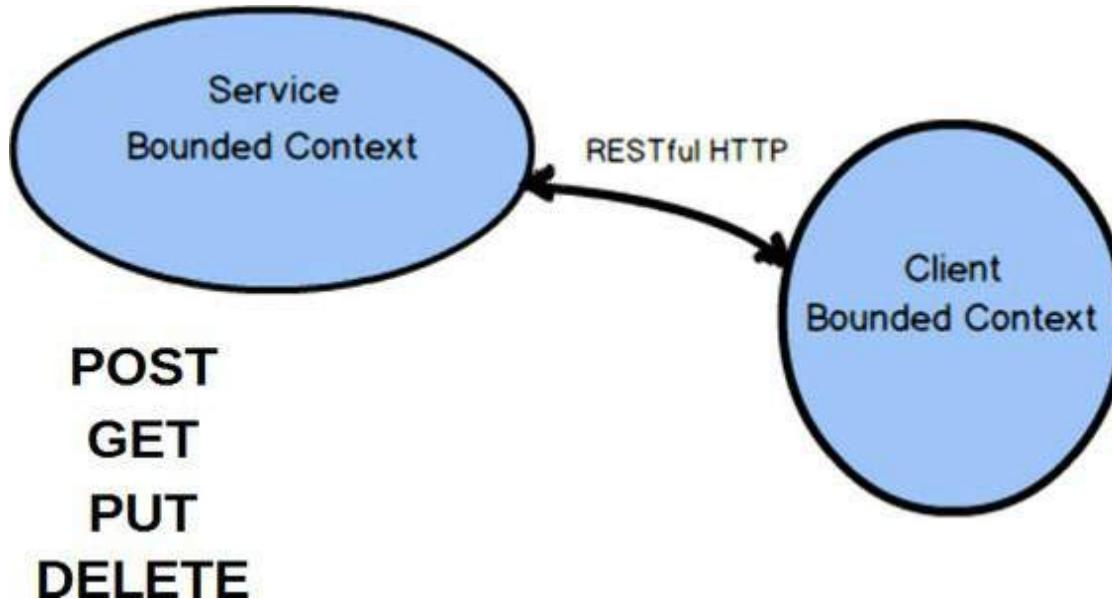
## Disadvantages

- Lack robustness due to **network failure** or **latency**
- Lack robustness if the upstream **service fails**
- **Strong coupling** between upstream service and downstream client
  - Using a specific protocol, standard or otherwise

## Best Practices

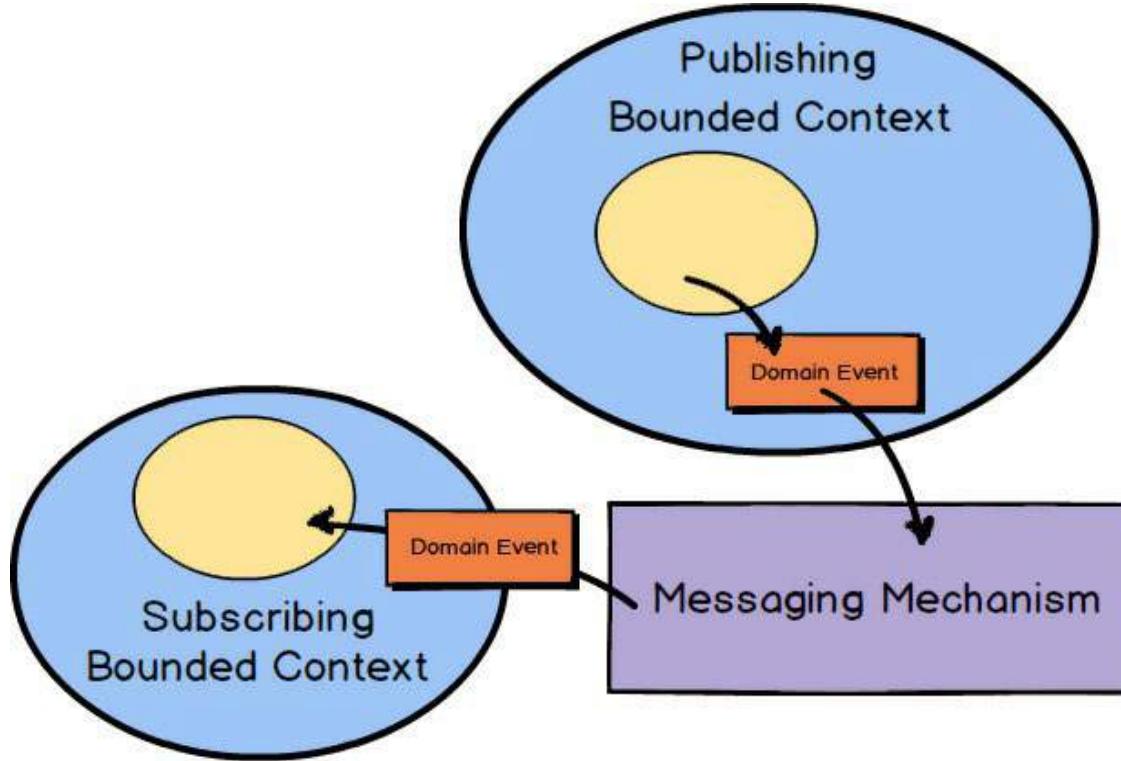
- Upstream service provides an **Open Host Service** with a **Published Language**
- Downstream client designed with an **Anticorruption Layer**

# RESTful HTTP



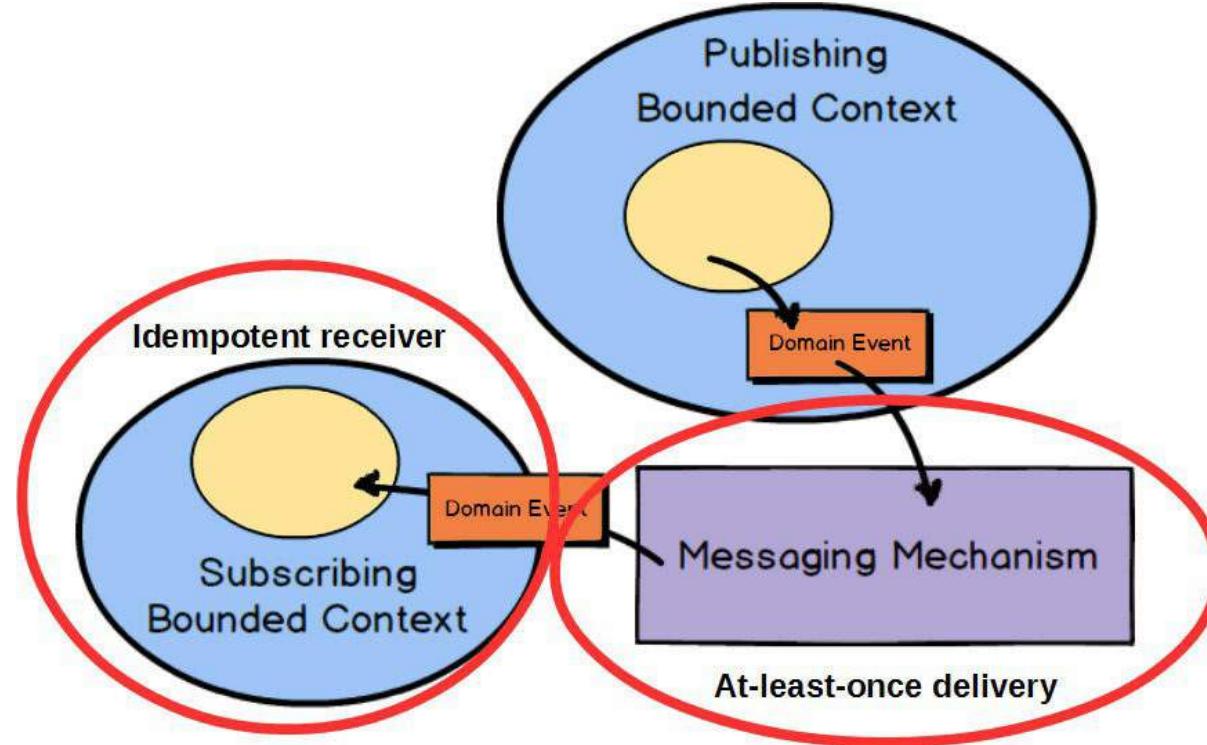
- Focus on the **resources** that are exchanged between the contexts
- With 4 primary **HTTP operations**: POST, GET, PUT and DELETE
- Normally result in **well-defined APIs**
- Susceptible to **network failure** and **latency** as well as upstream **service failure**
  - But RESTful HTTP can mitigate by relying on the **reliability** and **scalability** of the Web

# Messaging-1



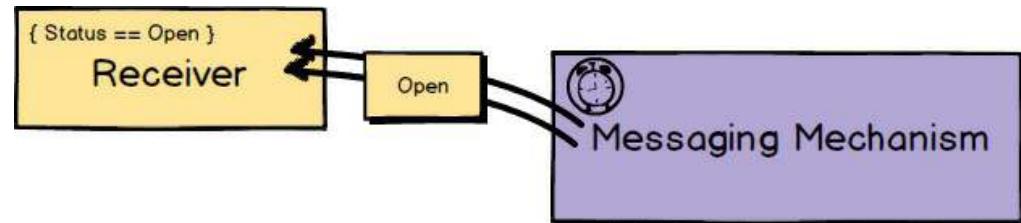
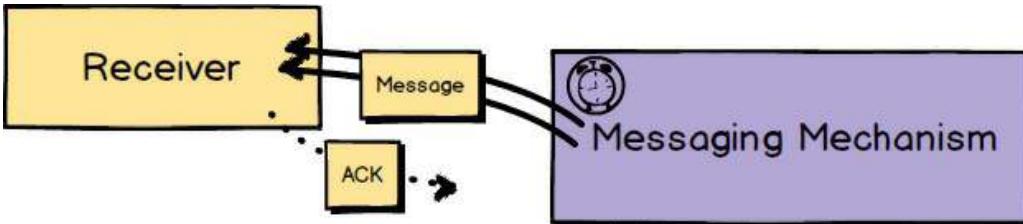
- **Client** context **subscribes** to the **Domain Events published** by another context
  - The events could also be published by the same context
- Tend to be **more robust** form of integration as
  - The **temporal coupling** associated with blocking forms (e.g. REST) is removed
  - The system is more robustly built in **anticipation of the latency** of message exchange

# Messaging-2



- Important for the messaging mechanism to support **At-Least-Once-Delivery** so as to ensure that all messages will be received eventually
- And the subscribing context to be implemented as an **Idempotent Receiver**

# Messaging-3.



## At-Least-Once-Delivery

- Messaging mechanism **periodically redeliver** a given message in cases like
  - Message **loss**
  - **Slow-reacting** or **downed** receivers
  - Receivers **failing to acknowledge** receipt

## Idempotent Receiver

- As a message could be **delivered more than once**, the receiver should be designed to deal correctly with it
- An **Idempotent Receiver** of a request performs an operation in such a way that it produces the **same result** even if it is **performed multiple times**
  - Via de-duplication, safe re-application, etc.

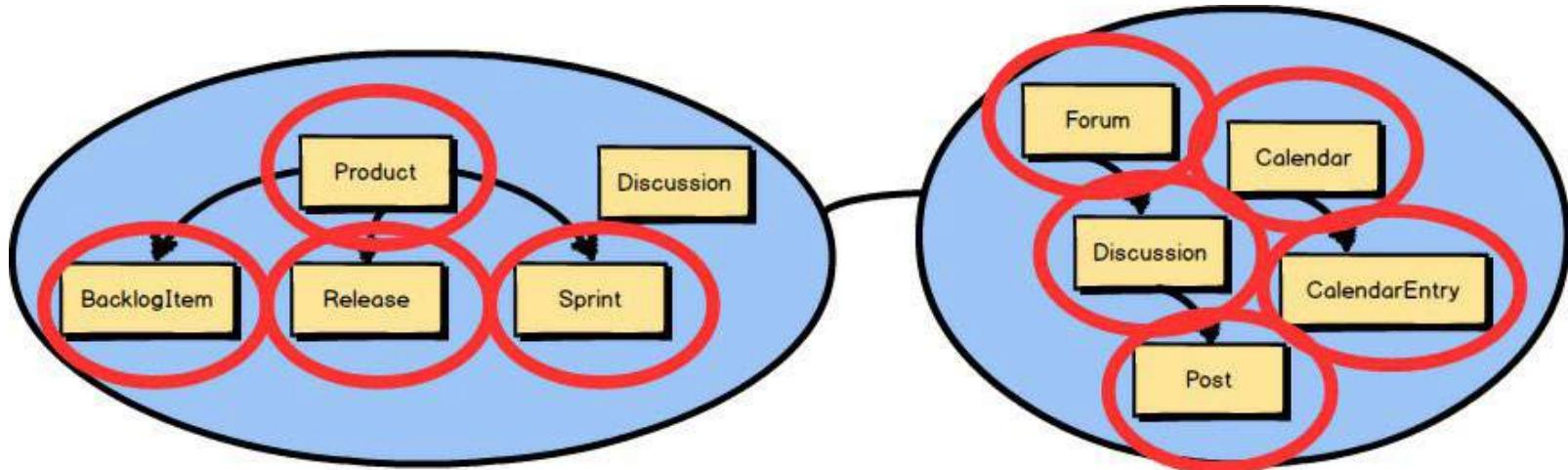
# Topics

- Characteristics of Microservices
- **Decoupling Reusable Services**
  - Context Mappings
  - **Aggregates**
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- SDKs for Consuming Services

# Aggregates-1

- A **bounded context** comprises one or more domain entities
- An **aggregate** is a **cluster** of domain entities, residing within the same bounded context, that should be treated as a **single unit**
  - Normally loaded into memory, modified or saved to storage within **one single transaction**
  - Specifying the **scope of a transaction** where the **integrity of the whole aggregate** is enforced; consequently, integrity across aggregates may be enforced via separate transactions

# Aggregates–2.

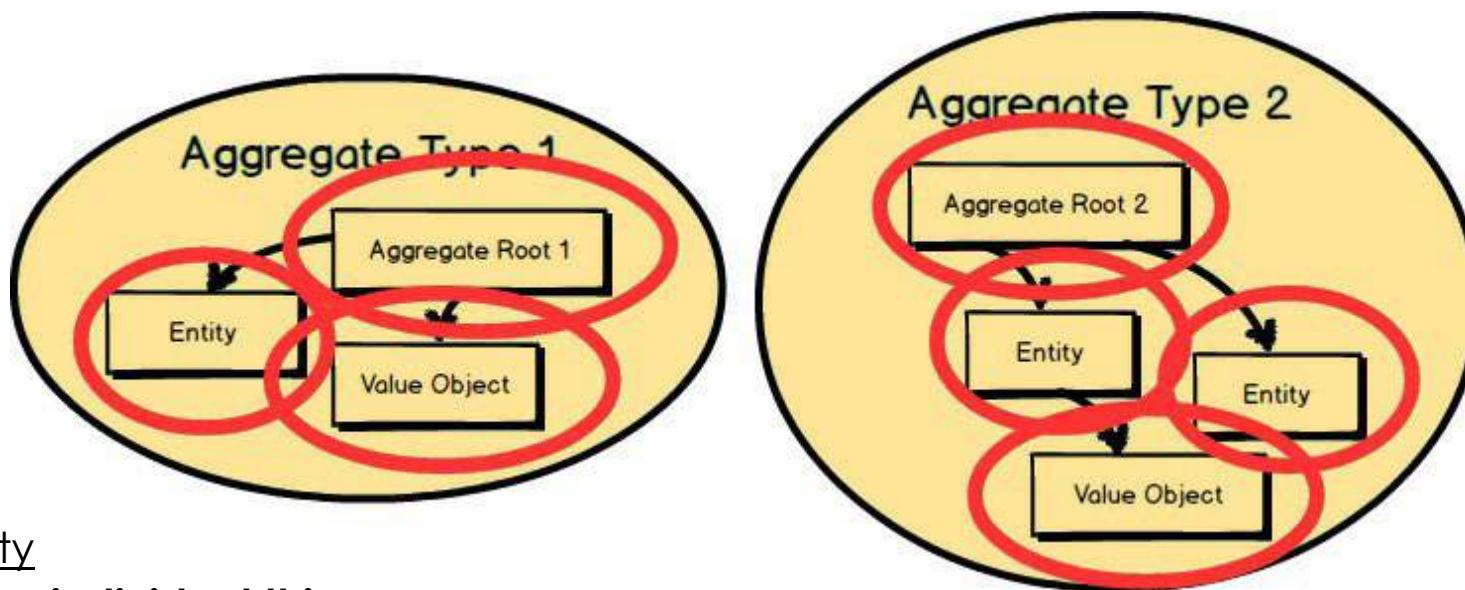


- Each of the above circled concepts could potentially be aggregates
- An aggregate helps to **group entities** within a BC for manageability as a single unit
- **Aggregate**
  - is composed of one or more **Entities**
  - one Entity is called the **Aggregate Root**
  - may also have **Value Objects**

# Issues with Designing Aggregates

- What are the **different elements** of domain concepts?
- How to **reference** from one domain concept to others?
- What is the scope of **transaction** of domain concepts?
- What is the scope of enforcement for **business invariants** for domain concepts?

# Entity, Aggregate Root, Value Object



## Entity

- An **individual thing**
- Has a **unique identity**
  - for distinguishing against other entities of the same or different type
- **Mutable** most of the times
  - i.e. state changes over time

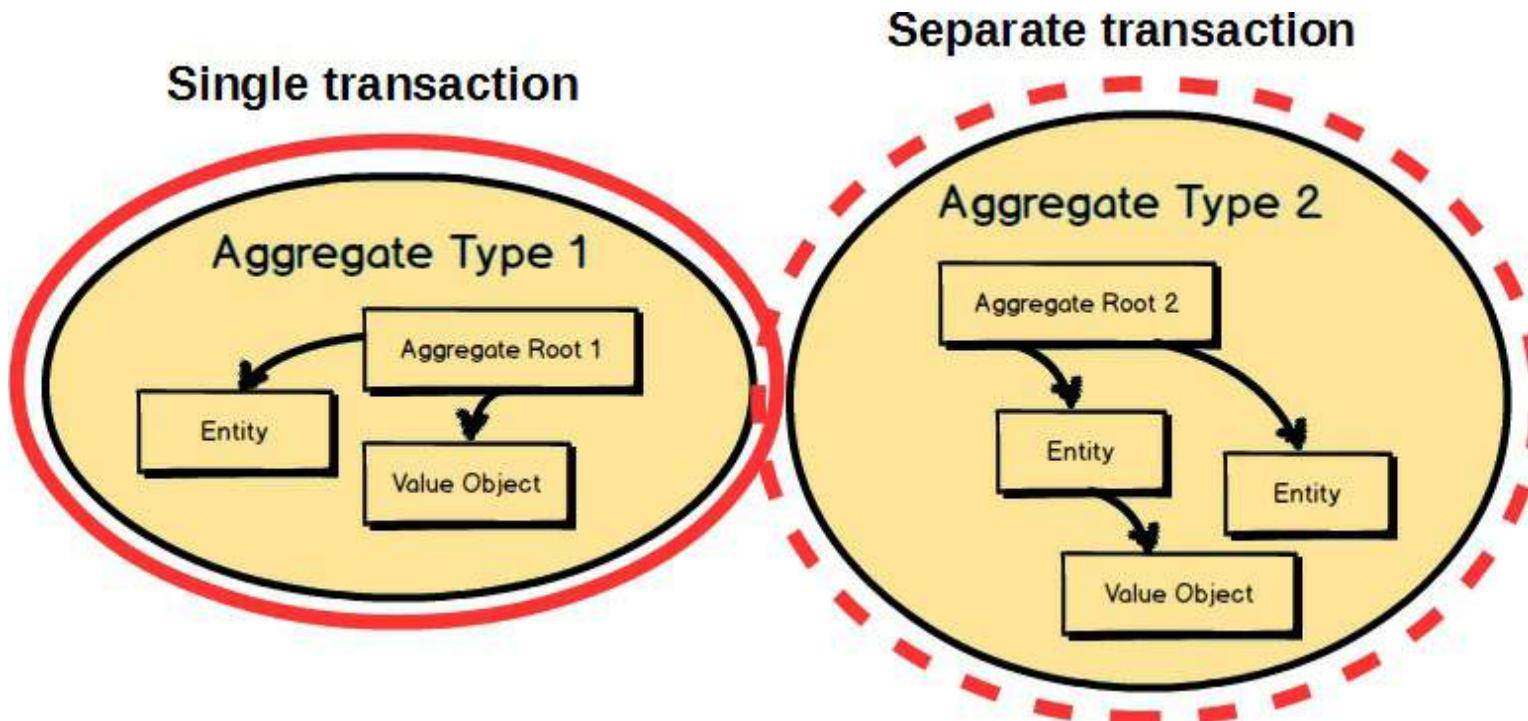
## Aggregate Root

- The **root entity** of an aggregate
- **Owns** all the other elements in the aggregate

## Value Object

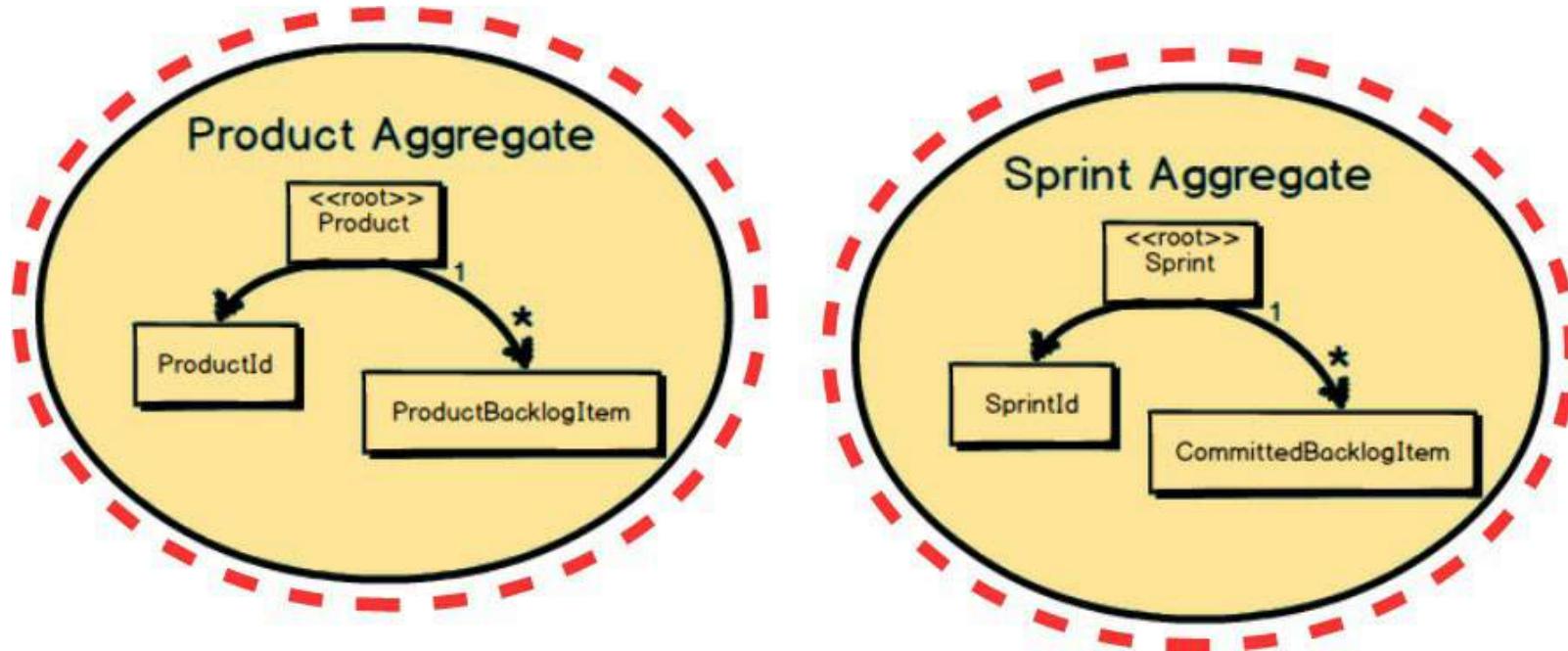
- models an **immutable** conceptual whole
- does **not** have a **unique identity**
- used to **qualify** (describe, quantify or measure) an entity

# Transactional Consistency Boundary-1



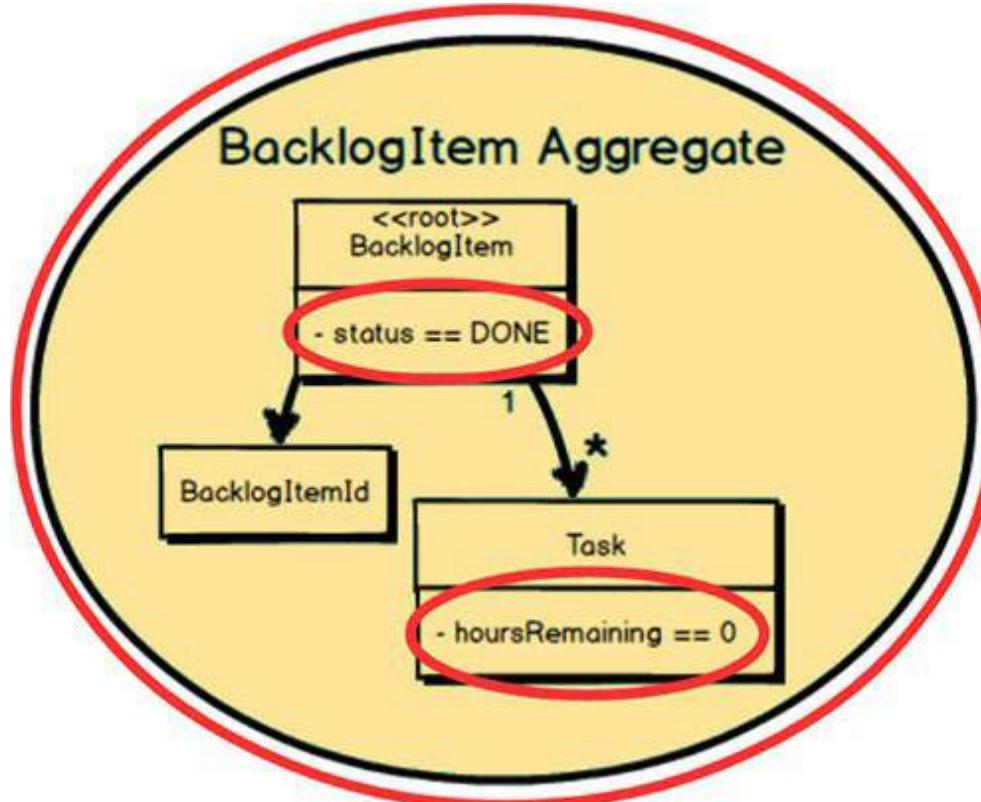
- Each aggregate forms a **transactional consistency boundary**
  - All **composed parts** must be consistent, according to **business rules**, when the controlling transaction is committed to database
- In **one transaction**, modify and commit only **one aggregate**

# Transactional Consistency Boundary–2.



- The above are two **sample aggregates**
- Product is designed such that at the **end of a transaction** all composed `ProductBacklogItem` instances must be **accounted** for and **consistent** with the Product root
- Sprint is designed such that at the **end of a transaction** all composed `CommittedBacklogItem` instances must be **accounted** for and **consistent** with the Sprint root
  - Consistent: See the next slide

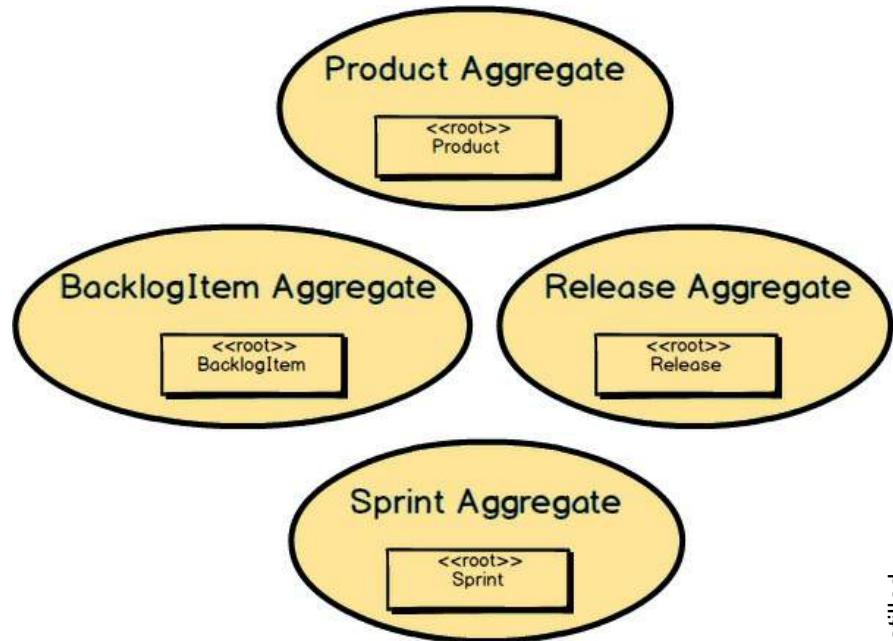
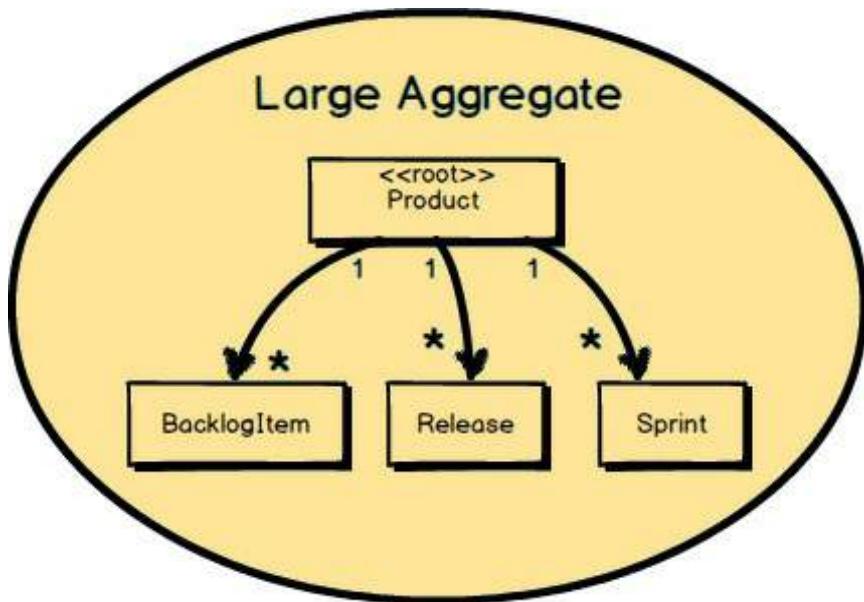
# Aggregate Rules of Thumb-1



Protect business invariants inside aggregate boundaries

- E.g. “When all Task instances have hoursRemaining of zero, the BacklogItem status must be set to DONE .”
- At the **end of a transaction**, this very specific **business invariant** must be met

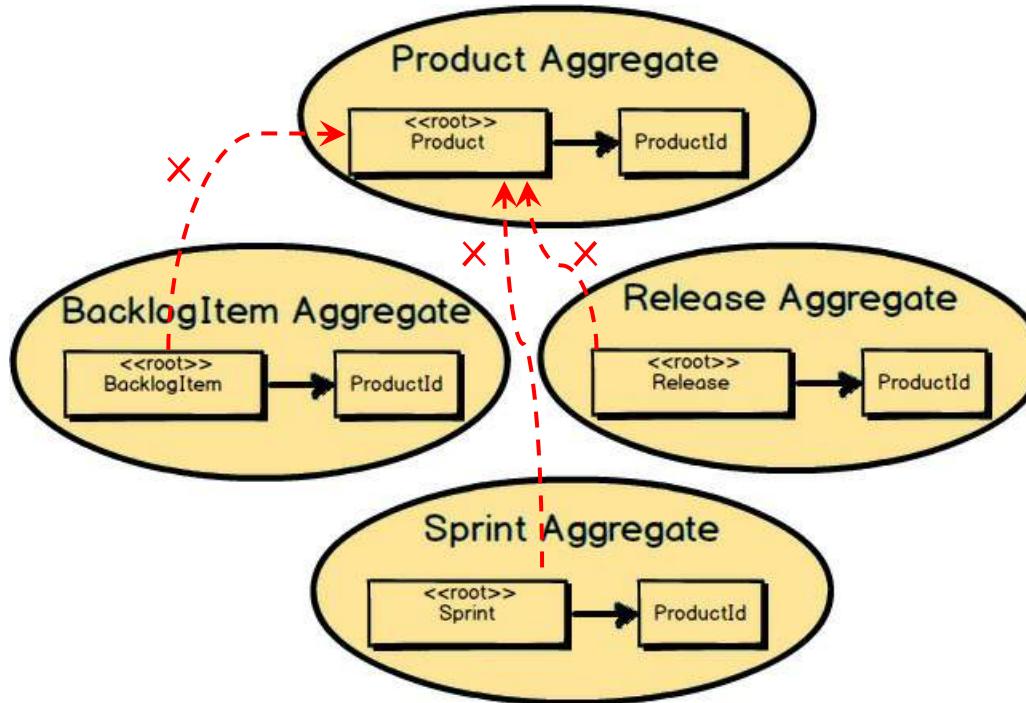
# Aggregate Rules of Thumb–2



## Design Small Aggregates

- E.g. Product literally contains a potentially very large collection of BacklogItem instances, a large collection of Release instances, and a large collection of Sprint instances
- The **memory footprint** and **transactional scope** of each aggregate should be **relatively small**
  - Smaller aggregates **load quickly**, **take less memory** and are **faster to garbage collect**
  - They would also have **more frequent transactional success**
  - They would also be **easier to work on and test**
- An aggregate should be **single-minded** → Single Responsibility Principle

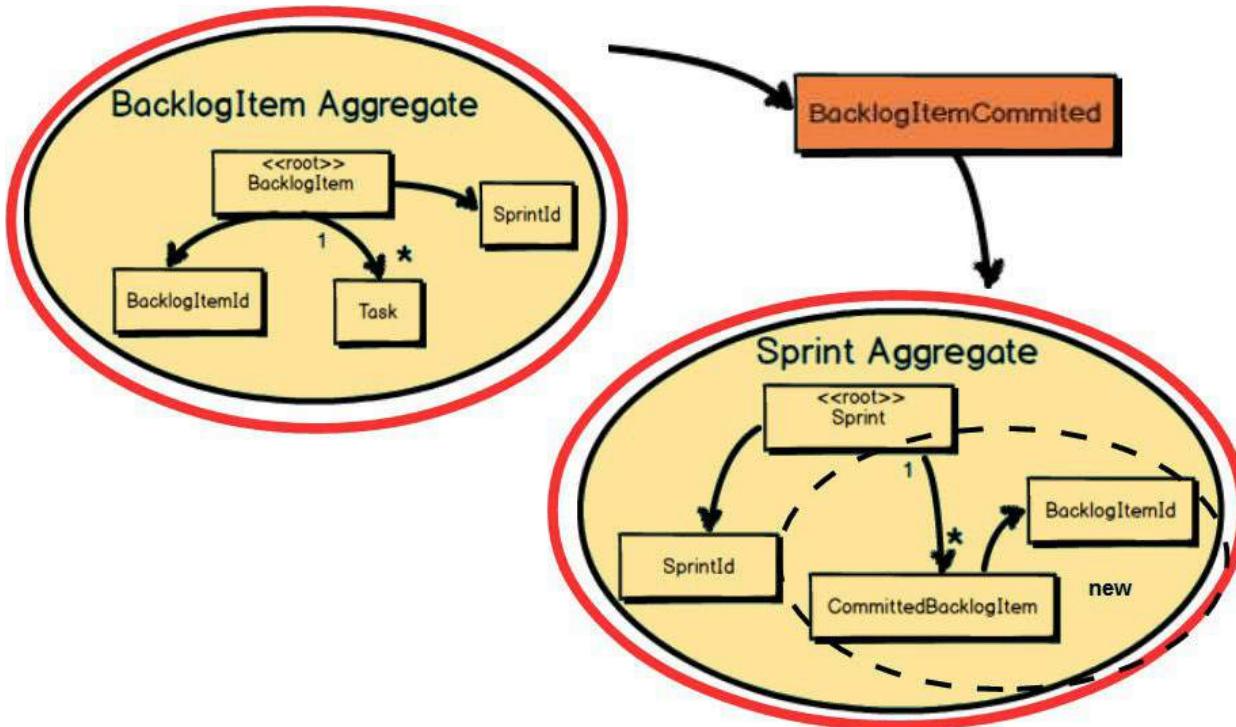
# Aggregate Rules of Thumb-3



Reference other aggregates by identity only

- E.g., BacklogItem , Release , and Sprint all reference Product by holding a ProductId
- The broken up smaller aggregates should **reference** the others by **identity only**
  - **Discourages** aggregates to hold a **direct object reference** to others
  - Identities can be **easily stored** in various kinds of **persistence mechanism**
- The **same rule** applies to referencing **between BCs**
- Identities should be modeled as **value objects** (which are immutable)

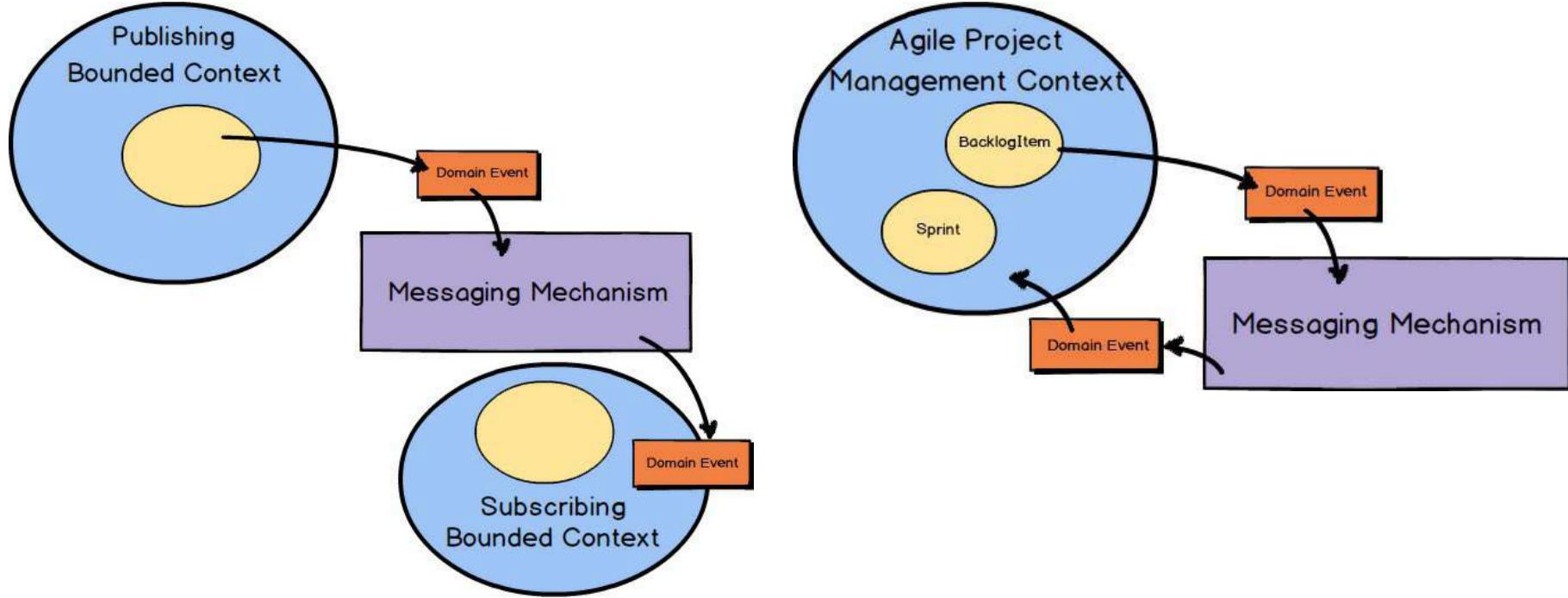
# Aggregate Rules of Thumb-4



Update other aggregates using eventual consistency-1

- E.g. When a `BacklogItem` is committed to a `Sprint`:
  - The `BacklogItem` is modified to contain the `SprintId` of the `Sprint` to which it is committed in **one transaction**
  - The `BacklogItemCommitted` **domain event** is generated by the `BacklogItem` and received by the `Sprint`
  - The `Sprint` should also be updated with the `BacklogItemId` of the newly committed `BacklogItem` in **another transaction**

# Aggregate Rules of Thumb–5



Update other aggregates using eventual consistency–2.

- The **publishing** aggregate and the **subscribing** aggregate could be **within or across BCs**
- Caution! If the business requirements require **two aggregates** need to be **updated immediately** one after the other, consider **merging them into one** aggregate
  - That is, the **elapsed time** of a domain event may result in **unacceptable inconsistency**

# Aggregate Rules of Thumb–6.

## Avoid an Anemic Domain Model

- An Anemic Domain Model has its aggregates having **only public accessors** (getters and setters) but no real business behavior
- Domain business logic should be **kept in the aggregates**
- Domain business logic should **not be leaked** into the **application services** above the domain model
  - Should also **not be delegated** to **helper/utility** classes

# Topics

- Characteristics of Microservices
- **Decoupling Reusable Services**
  - Context Mappings
  - Aggregates
  - **Domain Events**
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- SDKs for Consuming Services

# Domain Events

- A Domain Event (DE) is a **record** of some **business-significant occurrence** in a BC
- The **order** of domain events is important for **causal consistency**
  - That **causally related operations** of a business domain are seen by **every dependent node** of a distributed system in the **same order**

## Causally consistent

1. Sue posts a message saying, “I lost my wallet!”
2. Gary says in reply, “That’s terrible!”
3. Sue posts a message saying, “Don’t worry, I found my wallet!”
4. Gary replies, “That’s great!”

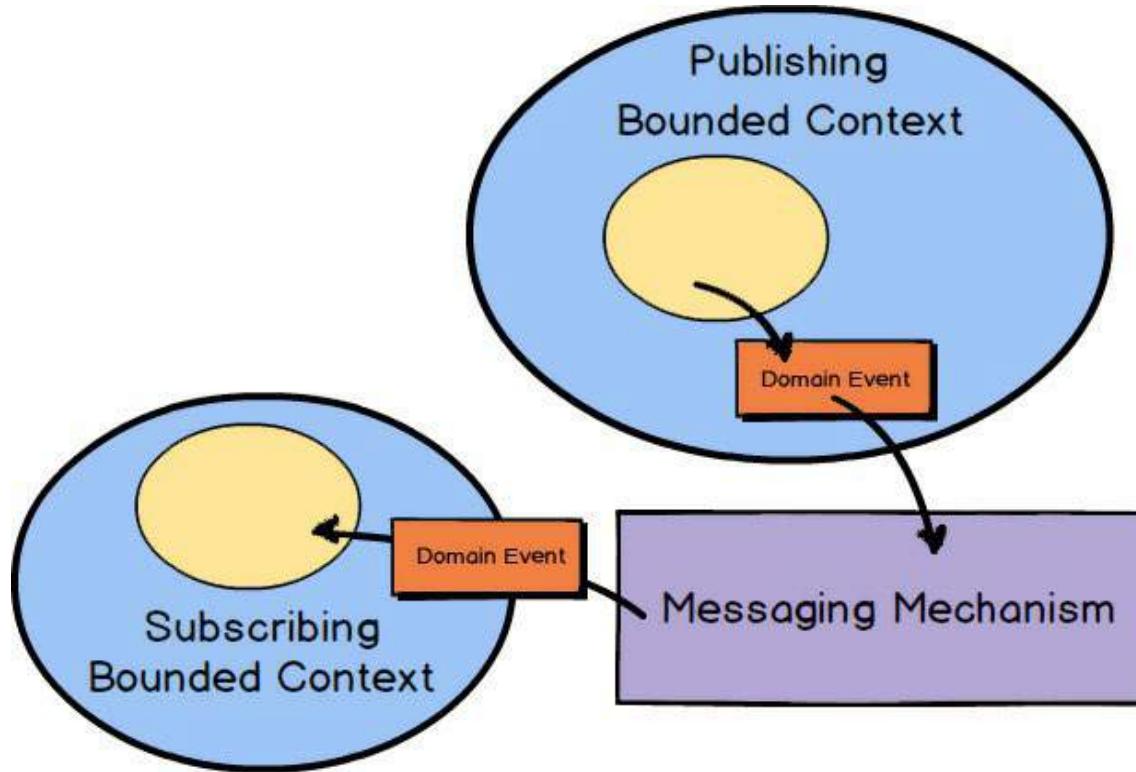
## Causally inconsistent

1. Sue posts a message saying, “I lost my wallet!”
2. *Gary replies, “That’s great!”*
3. Sue posts a message saying, “Don’t worry, I found my wallet!”
4. *Gary says in reply, “That’s terrible!”*

# Issues with Designing Domain Events

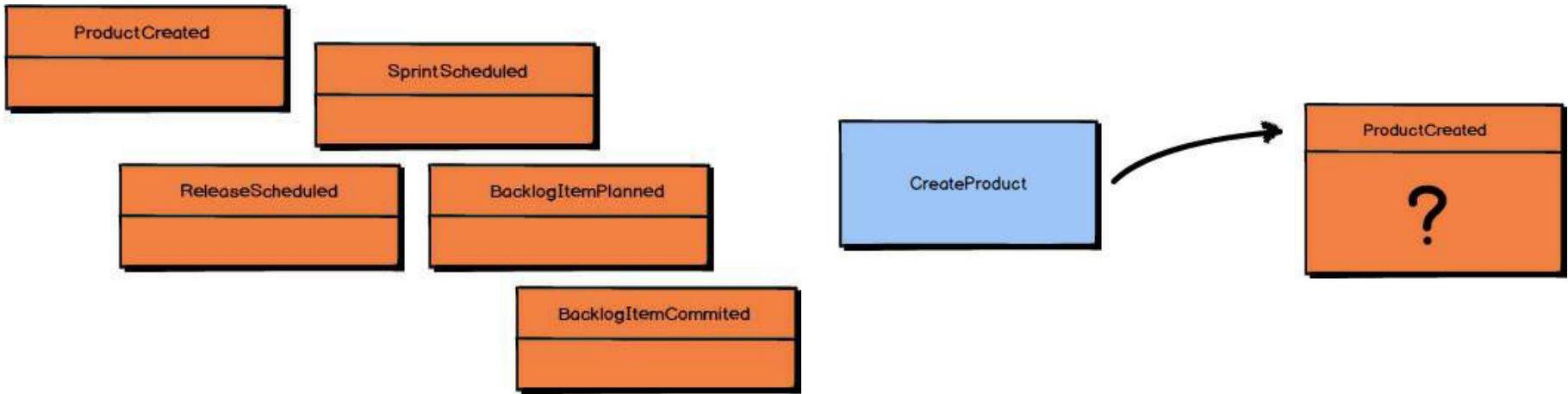
- How to **identify** domain events?
- How to **name** domain events?
- How to **identify the properties** of domain events?
- When to **commit** domain events?

# Identifying Domain Events



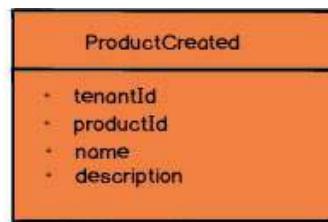
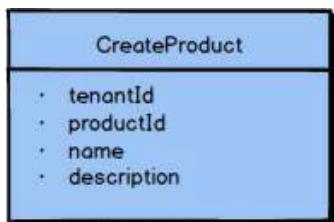
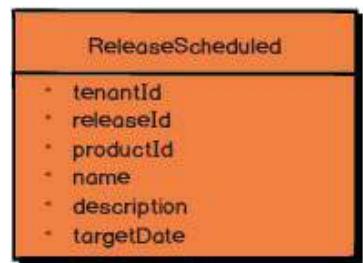
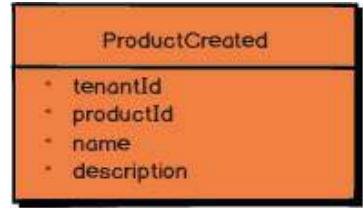
- For each of the **context mappings** of **Messaging** type
  - Identify the business functions that rely on the context mapping
  - For each of the **business functions**
    - Identify the **domain events** that are required

# Naming Domain Events

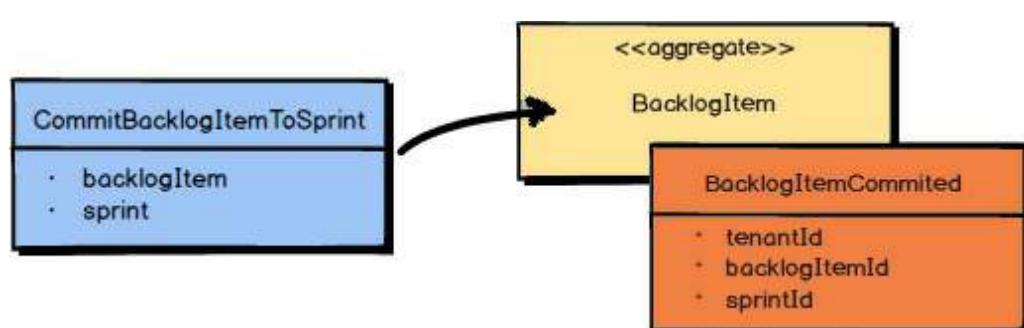


- DEs are **past occurrences**
- It should be stated with a **verb in past tense**
- If the domain event (ProductCreated) is triggered by a command/request (CreateProduct), the event type name can be **specified according to the command name**

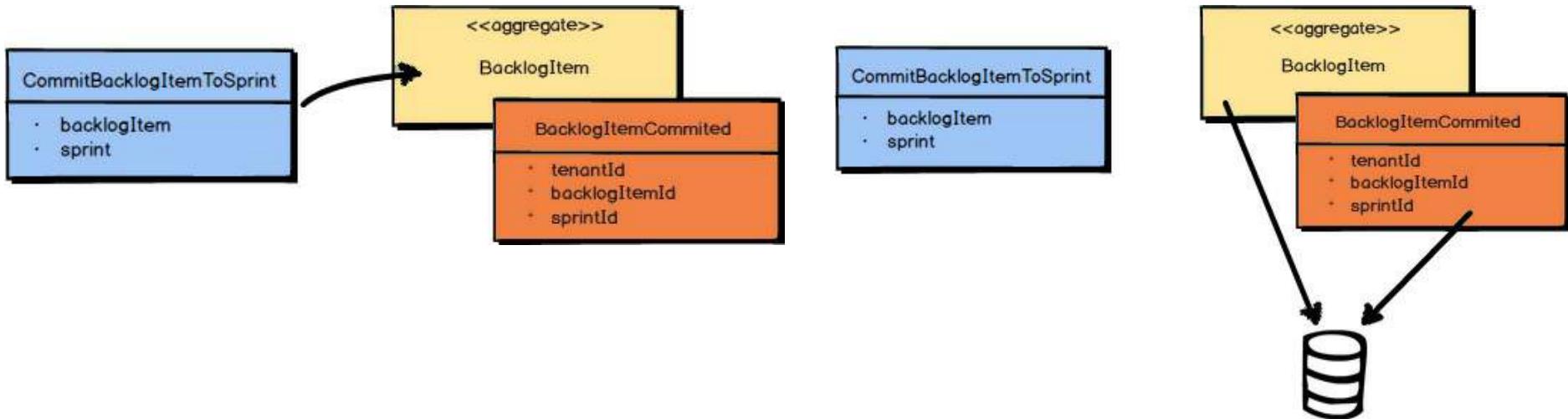
# Properties of Domain Events



- A DE should hold **all the properties** of the **command that caused it to be created**
  - This **fully and accurately** informs all subscribers what happens in the model
- At times, a DE may be **enriched with additional data**
  - Helpful so that consumers do not need to query back the publishing BC for additional data (tenantId)

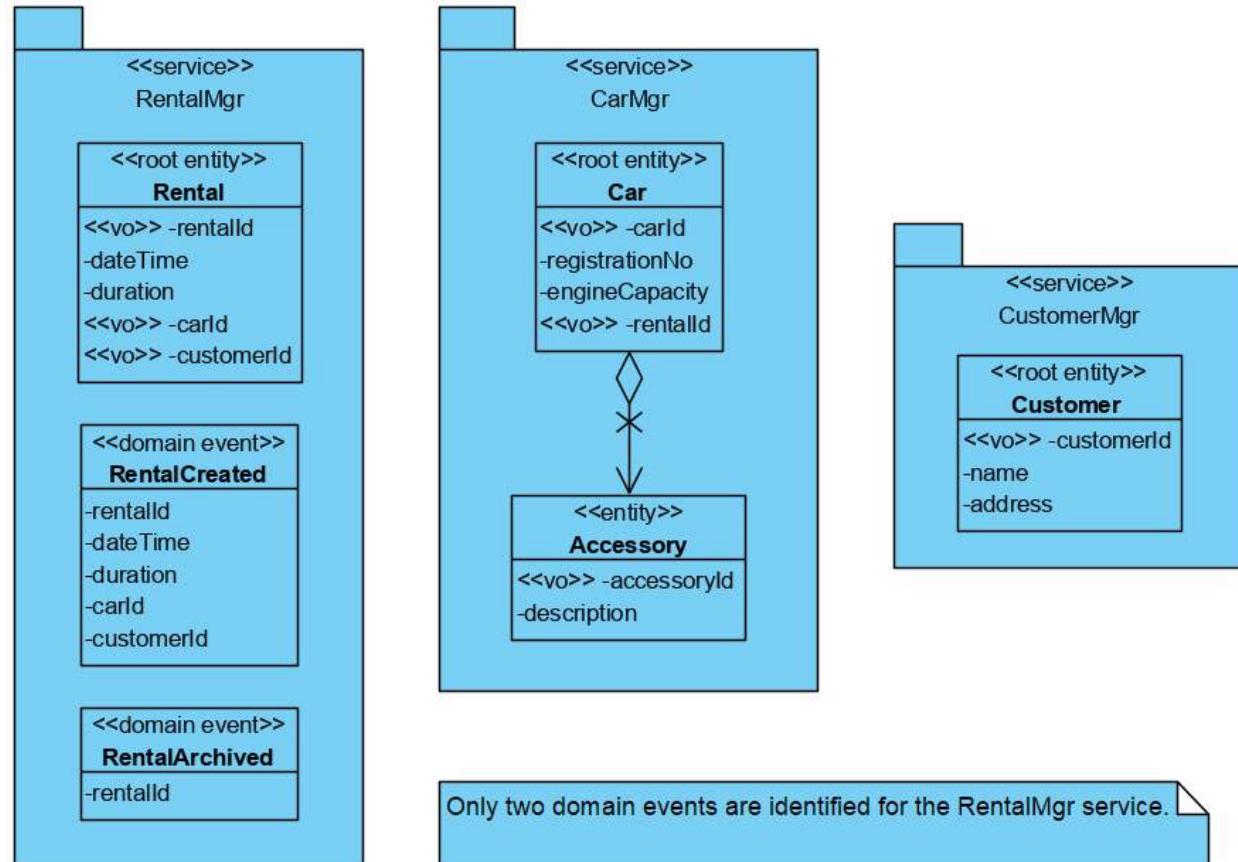
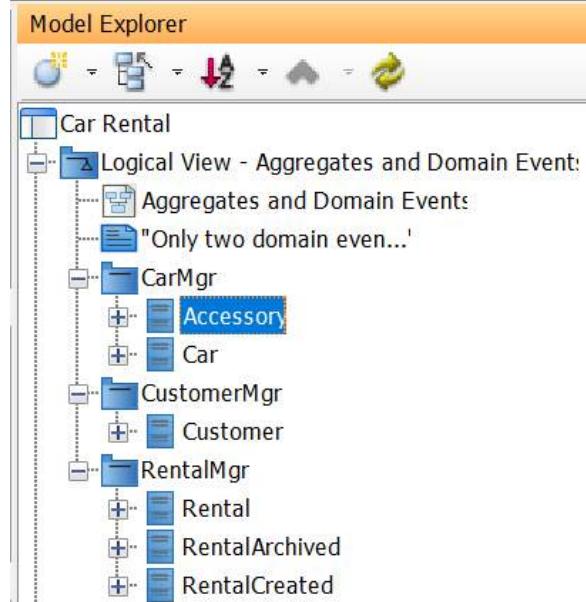


# Transaction Boundary of Domain Events.



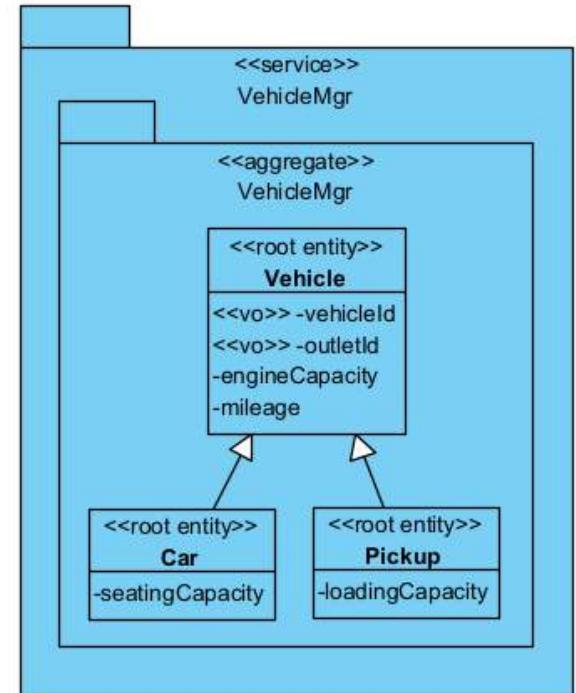
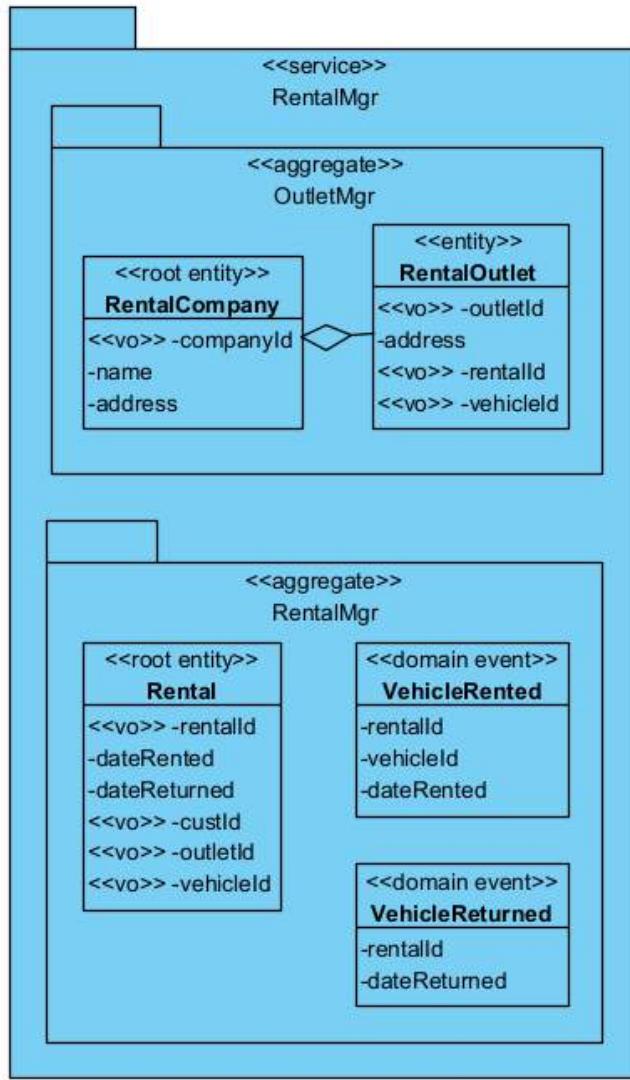
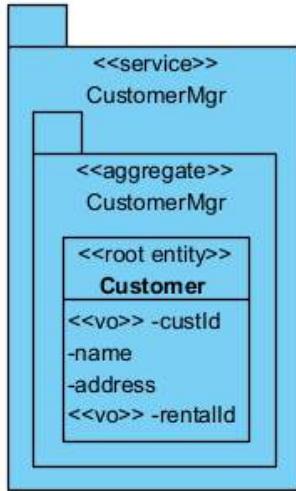
- E.g. The product owner commits a BacklogItem to a Sprint. The command itself causes the BacklogItem and the Sprint to be loaded. Then the command is executed on the BacklogItem aggregate. This causes the state of the BacklogItem to be modified, and then the BacklogItemCommitted domain event is published as an outcome.
- The **modified aggregate** and the **domain event** should be committed in the **same transaction**

# Simple Sample Diagram for Car Rental



Using Visual Paradigm  
(without showing aggregate boundaries)

# Another Sample Diagram for Car Rental



Using Visual Paradigm  
 (showing aggregate boundaries)

# Resultant Architecture for Car Rental.

- RentalMgr microservice
  - OutletMgr transaction scope
  - RentalMgr transaction scope
    - VehicleRented domain event
    - VehicleReturned domain event
  - RentalMgr database / team / repository
- VehicleMgr microservice
  - VehicleMgr transaction scope
  - VehicleMgr database / team / repository
- CustomerMgr microservice
  - CustomerMgr transaction scope
  - CustomerMgr database / team / repository

# Topics

- Characteristics of Microservices
- **Decoupling Reusable Services**
  - Context Mappings
  - Aggregates
  - Domain Events
  - **Workshop: Design Aggregates and Domain Events**
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- SDKs for Consuming Services

# Workshop: Design Aggregates and DEs.

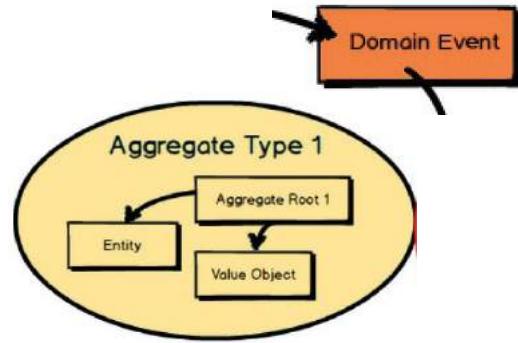
## Objectives

- To design the aggregates for a BC
- To design the DEs for a BC

## Instructions

- Select **two significant services** (i.e. BCs) from the sample solution of “Workshop: Identify domain concepts, BCs and ULs”
 

if two entities --> one to many ---> then should be two different aggregation
- Design the **aggregates** for the two services
  - With aggregate roots, entities and value objects
- Design the **domain events** that are **published** by the core domain
  - ≥3 interesting domain events
  - With event names and properties (assumed)

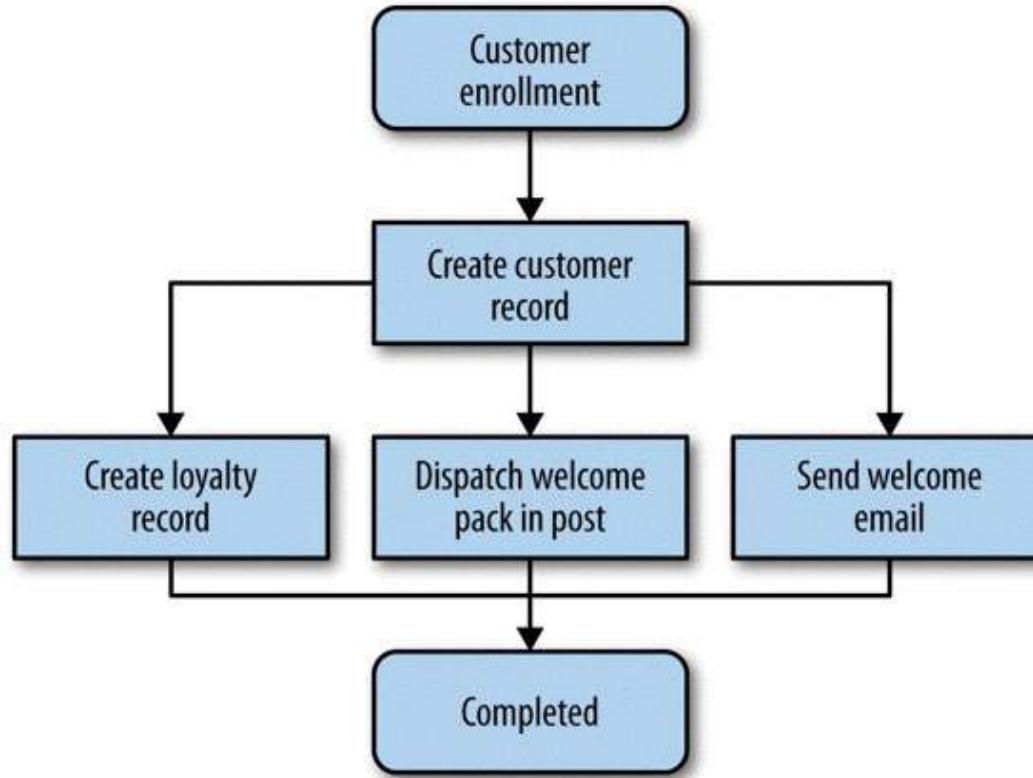


# Topics

- Characteristics of Microservices
- Decoupling Reusable Services
  - Context Mappings
  - Aggregates
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- **Orchestrating vs. Choreographing Reusable Services**
- API Gateways
- SDKs for Consuming Services

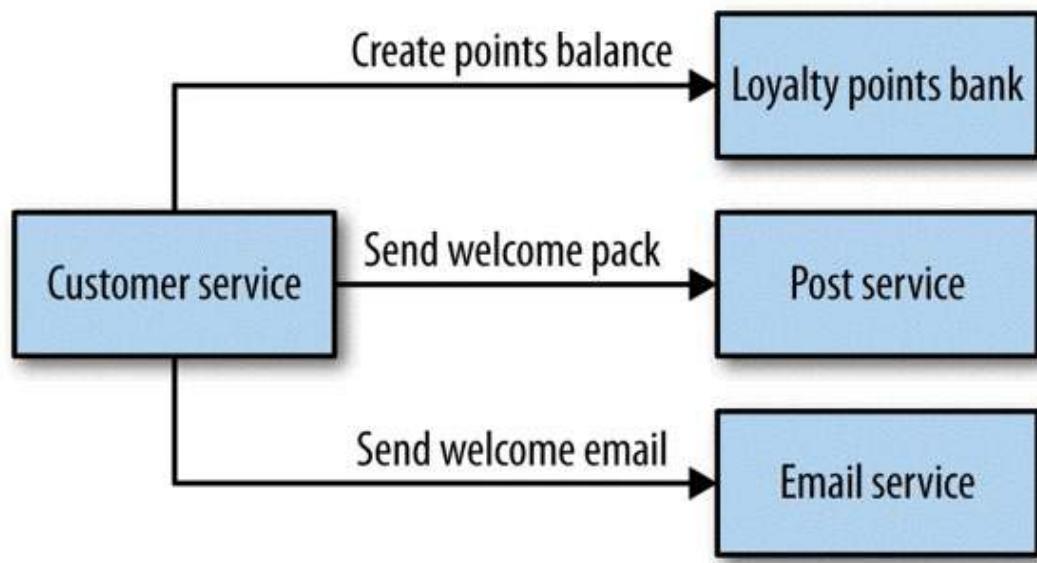
# Orchestrating vs. Choreographing Reusable Services

- As the business logic becomes more complex, the business functions **stretch across the boundary of individual services**
- The flow captured in the following flowchart can be either implemented via **orchestration or choreography**
  - Which may be **mixed and matched** as necessary



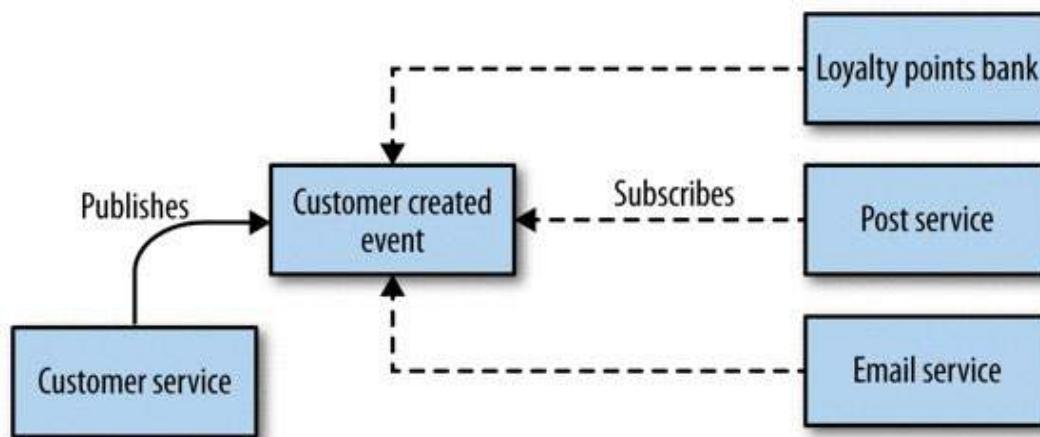
# Orchestrating Reusable Services

- **Simpler** approach
  - **Explicit** flow
  - Can rely on **rules engine**, etc.
- Result in overly smart “**god**” services
- The Customer service acts as the **central brain**
  - Creates the customer record
  - Requests the Loyalty Points Bank service to create loyalty record
  - Requests the Post service to dispatch welcome pack
  - Requests the Email service to send welcome email



# Choreographing Reusable Services.

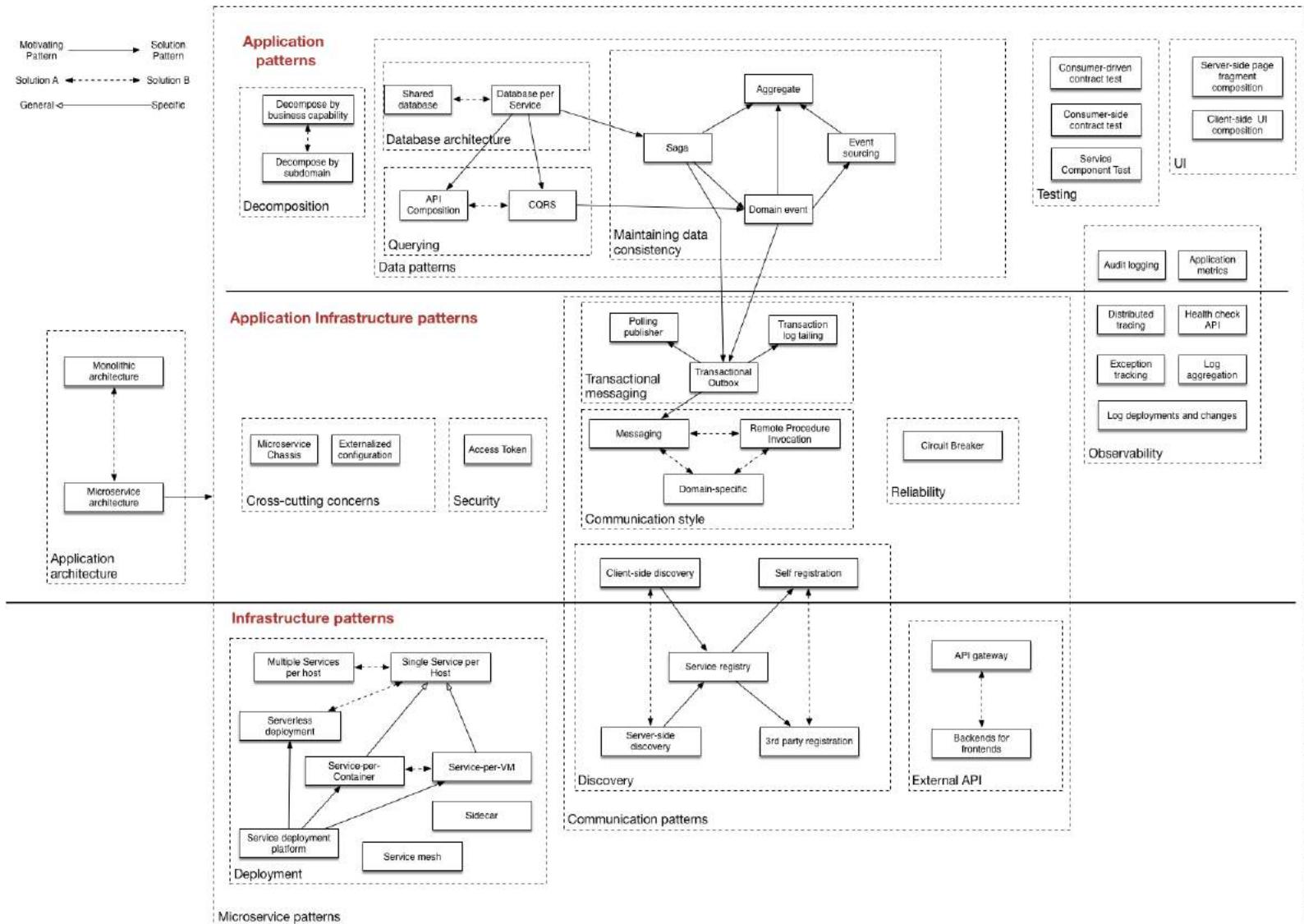
- **More complex** approach
  - **Implicit** flow
  - Rely on **domain events**
  - More **loosely coupled**
- Need to **monitor and track** that the flow is carried out as expected
- The Customer service
  - Creates the customer record
  - Emits an **event**
- The Loyalty Points Bank service receives the event and creates loyalty record
- The Post service receives the event and dispatches welcome pack
- The Email service receives the event and sends welcome email



# Topics

- Characteristics of Microservices
- Decoupling Reusable Services
  - Context Mappings
  - Aggregates
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- **API Gateways**
- SDKs for Consuming Services

# A Pattern Language for Microservices



# Social Multiplication Microservice App

## Welcome to Social Multiplication

Your new challenge is

$88 \times 34$

Result?

Your alias:

  
|

The result is correct! Congratulations!

### Leaderboard

User ID	Score
1	100
4	30
2	20
34	10
33	10

### Your statistics

User ID: 1

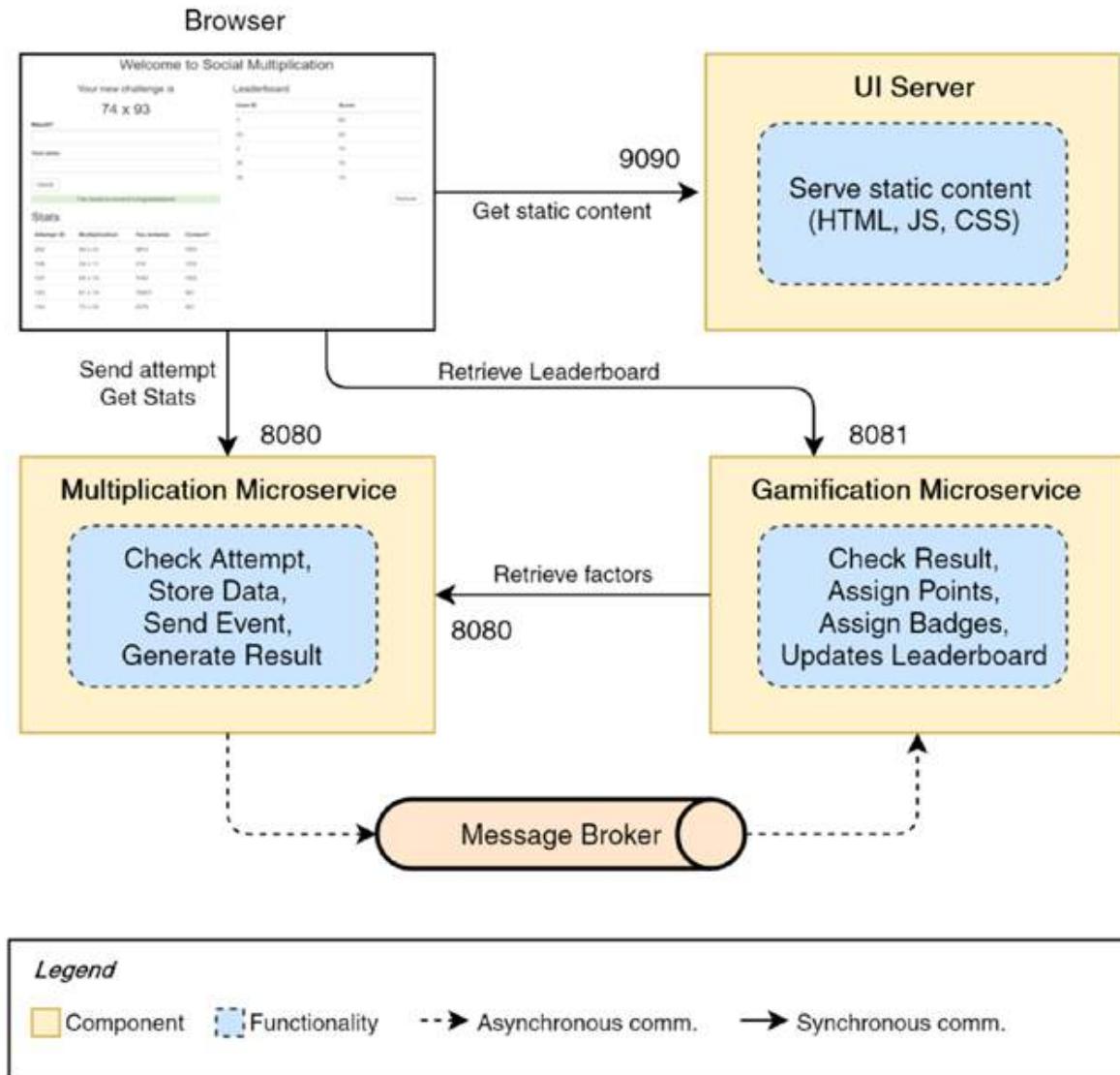
Score: 100

Badges: BRONZE\_MULTIPLICATOR,LUCKY\_NUMBER,FIRST\_WON

### Your latest attempts

Attempt ID	Multiplication	You entered	Correct?
40	55 x 27	1485	YES
39	28 x 46	1287	NO
19	33 x 56	897654	NO
17	42 x 17	714	YES
16	89 x 93	8277	YES

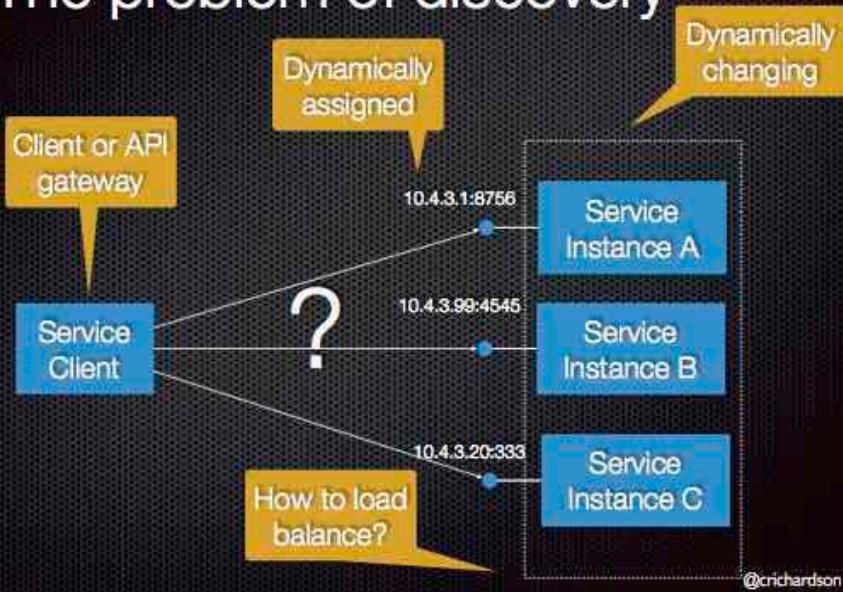
# Not Production Ready S/W Architecture



- Browser and Gamification know the location of Multiplication
- Gamification knows the location of Multiplication
- **Cannot scale horizontally**
  - Need Service Discovery, Load Balancing and API Gateway
- If scaled, Multiplication / Gamification databases should be shared across their service instances
  - Need High Availability for databases
- Message broker is a single point of failure
  - Need High Availability for message broker

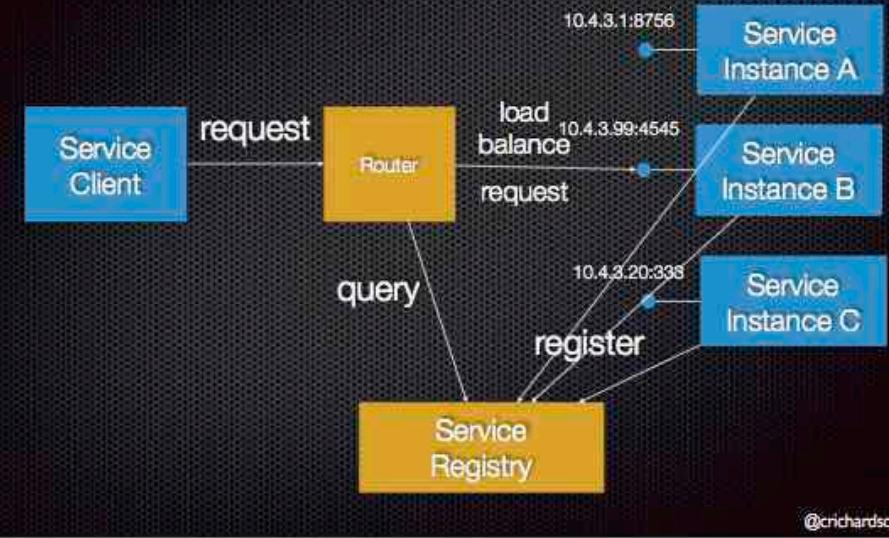
# Server-side Discovery + Service Registry

## The problem of discovery



- Microservice-based application typically runs in a virtualized or containerized environments where the **number of instances** of a service and their **locations** changes dynamically
- Need a mechanism that enables the clients of service to make requests to a **dynamically changing** set of **ephemeral** service instances

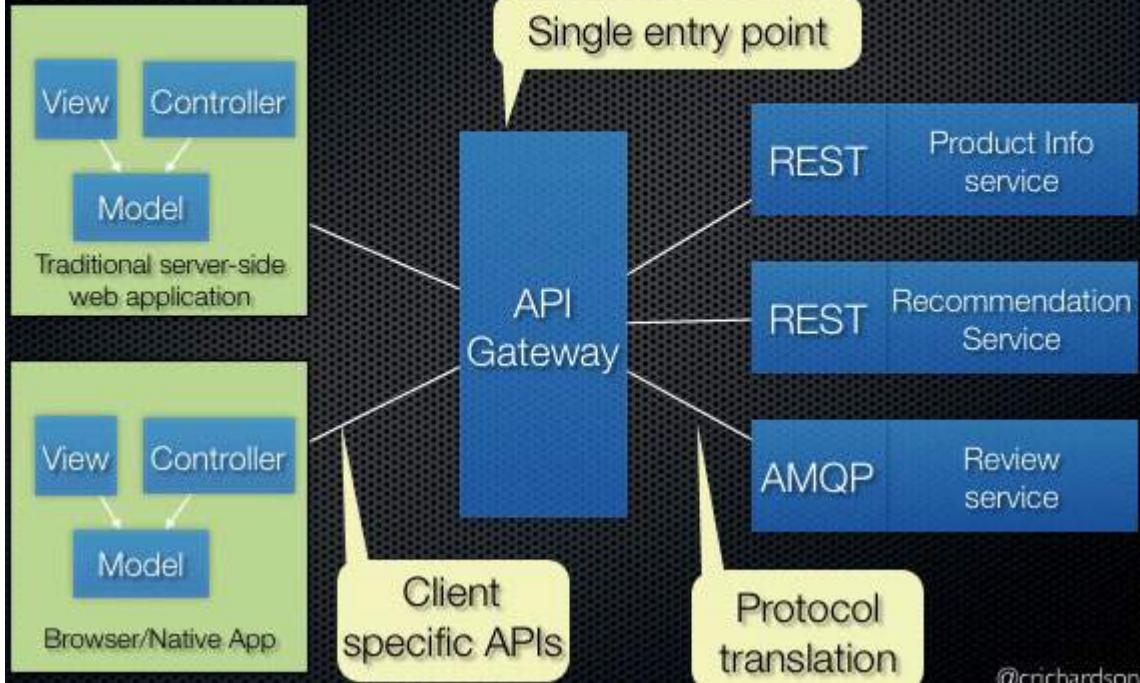
## Pattern: Server-side discovery



- The client makes a request via a **router** (aka **load balancer**) that runs at a **well known location**
- The router queries a **service registry**, applies **load balancing** and forwards the request to a **selected available** service instance

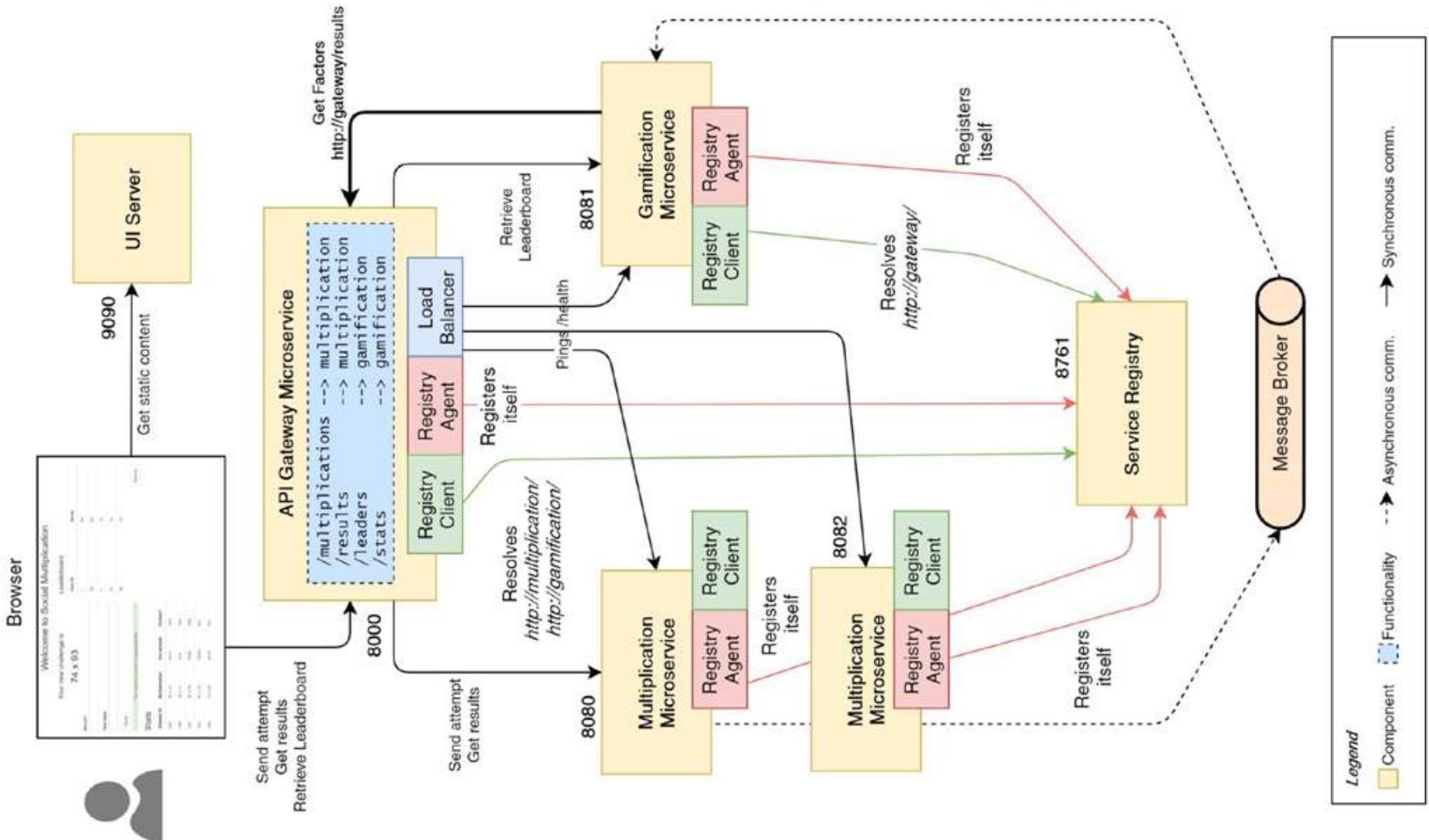
# API Gateway

## Use an API gateway



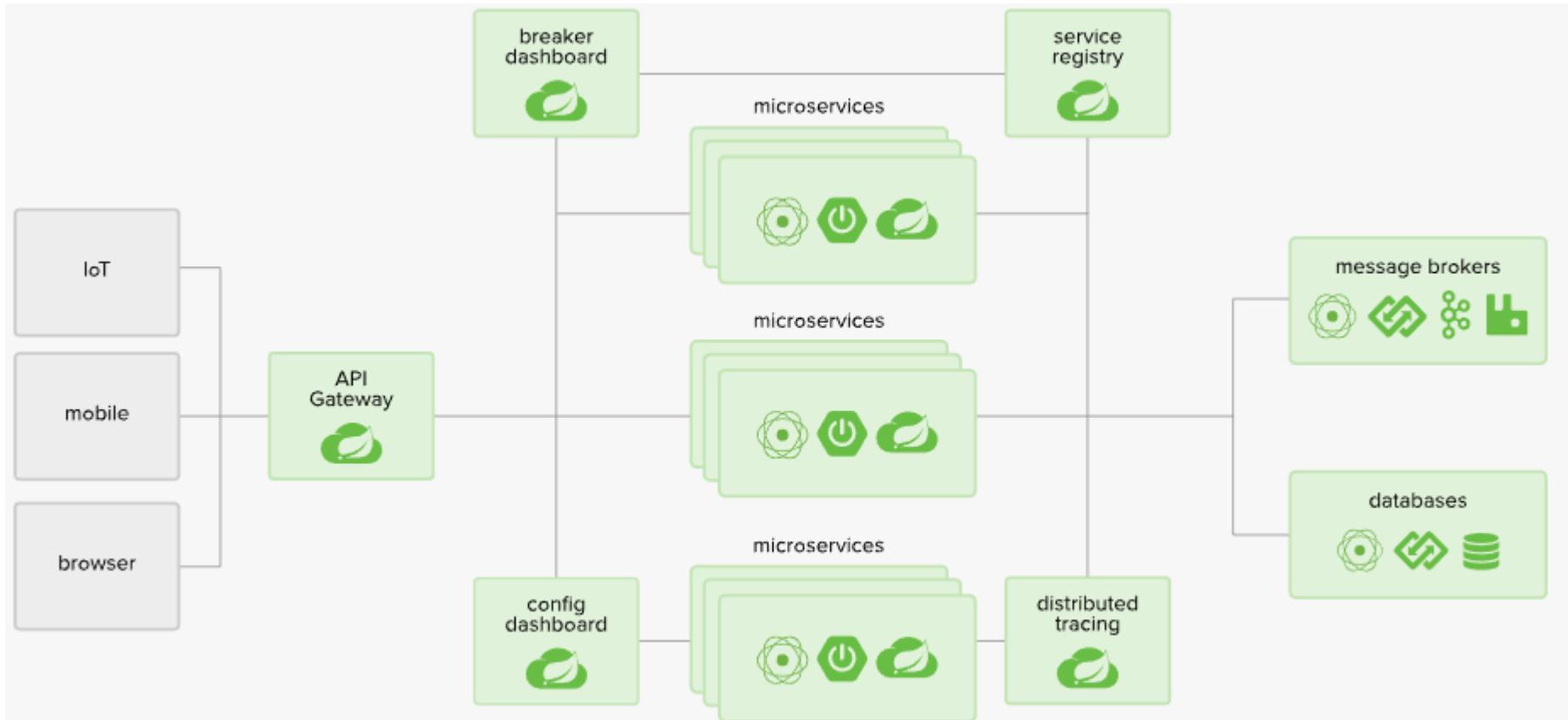
- Is the **router** of Server-side Discovery
- Client is oblivious of **application partitioning**
- Provide the **optimal API** for each client
- Help to access **multiple services** for client in a **single trip**
- Translate public API protocol to whatever **internal protocols**

# Production Ready S/W Architecture



*This architecture is based on the Spring Cloud tools, built on top of Spring Boot*

# E.g. Architecture of Spring Cloud

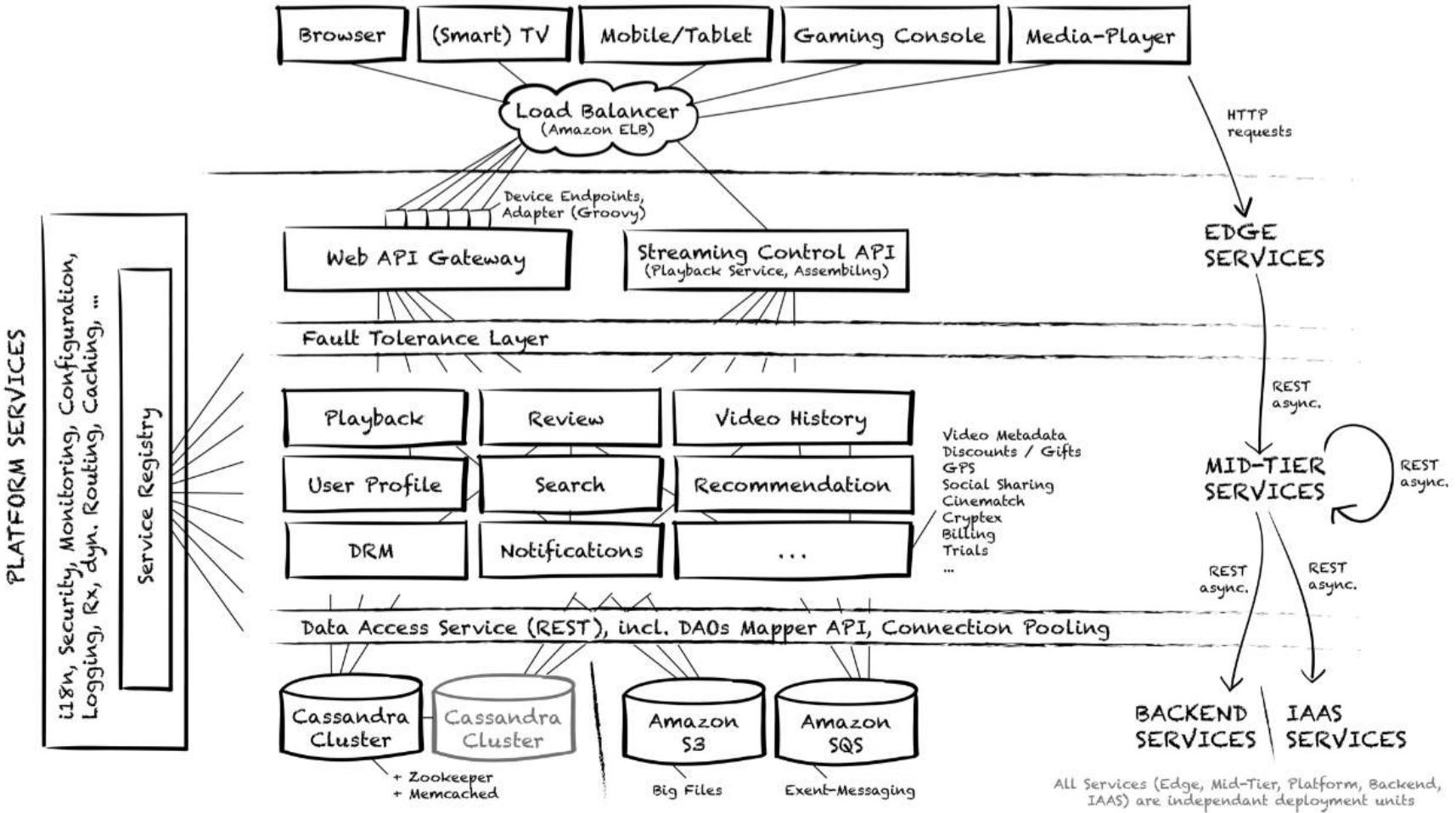


- Spring Cloud provides **tools** for developers to **quickly build** some of the **common patterns** in distributed systems
- Typical **use cases** include Service Registration and Discovery, Routing, Service-to-service Calls, Load Balancing, Circuit Breakers, Distributed Messaging

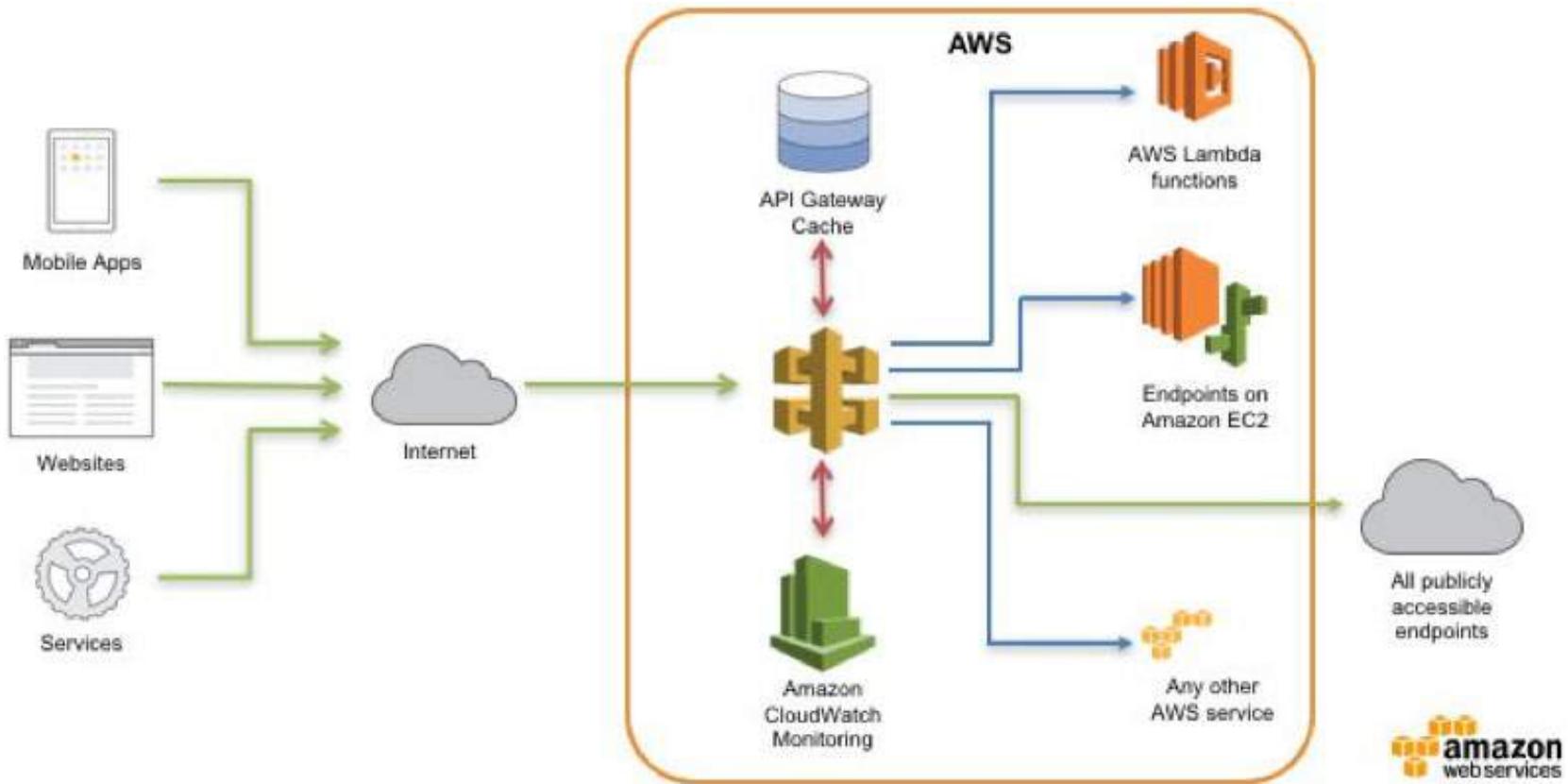
*Some of the Spring Cloud tools originate from Netflix*

# E.g. Architectural Overview of Netflix

## Netflix Architectural Overview (simplified)



# E.g. AWS API Gateway



- Create, publish, maintain, monitor, and secure APIs at any scale
- Accepting and processing concurrent API calls, traffic management, authorization and access control, monitoring, and API version management

# Topics

- Characteristics of Microservices
- Decoupling Reusable Services
  - Context Mappings
  - Aggregates
  - Domain Events
  - Workshop: Design Aggregates and Domain Events
- Orchestrating vs. Choreographing Reusable Services
- API Gateways
- **SDKs for Consuming Services**

# SDKs for Consuming Services

- The use of client libraries to consume services was already discussed
  - under the topic **Shared Libraries** of module **Reusable Assets and Frameworks**
- This topic extends the discussion by including
  - AWS SDK
  - Spring Boot REST Consumers with Feign

- <https://aws.amazon.com/tools/>
- Amazon simplifies **access to AWS services** from applications with SDKs tailored to **various programming languages or platforms**
  - Java, .NET, node.js, PHP, Python, Ruby, browser, Go, C++
  - Android, iOS, React Native, mobile Web

## High Level APIs

Focuses on making common tasks easy

S3 TransferManager

S3 Encryption

DynamoDB ObjectMapper

JavaMail (SES)

Policy API

Flow (SWF)

## Low Level Clients

Secure, robust, and easy to use clients

Amazon S3

DynamoDB

SQS

SNS

EC2

RDS

CloudFront

CloudWatch

Glacier

CloudSearch

Elastic Beanstalk

Cloud Formation

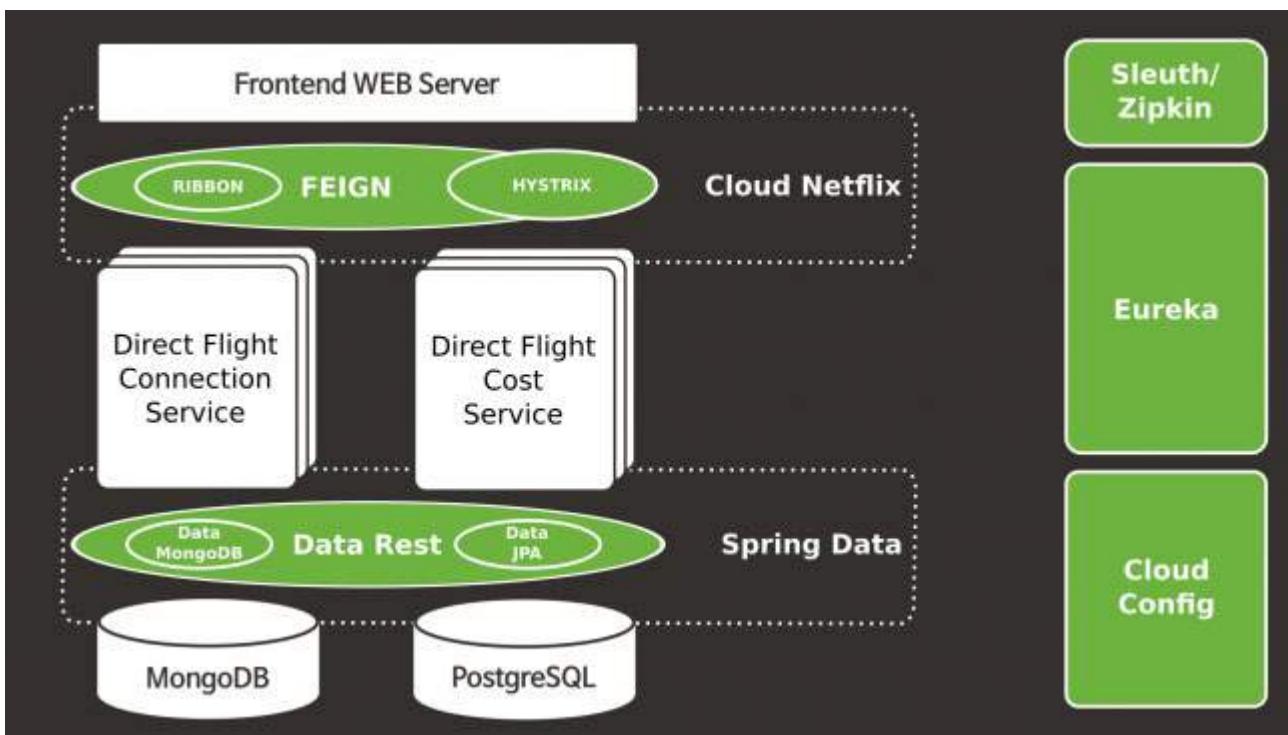
Elastic MapReduce

Simple Email Service

...

# Spring Boot REST Consumers with Feign.

- **Feign** is a Spring Cloud framework that originates from Netflix
- It allows a client to **consume REST services** as they were **part of the code**



- Feign **generates Feign Clients** that map to services
- Clients **invoke** services via Feign Clients
- Feign Clients work with Eureka, Ribbon and Hystrix to **find services** and performs **load balancing**

Credit: <https://www.novatec-gmbh.de/en/blog/spring-cloud-sprint-a-fast-and-comprehensive-spring-cloud-services-tutorial/>

# Summary

- After identifying the reusable services (i.e. BCs), they are **not totally decoupled**
  - There are still **relationships** and **interactions** between them
- Context Mappings model the **integration** between reusable services
- Aggregates **break the references** between reusable services and specify the **scope of transaction**
- Domain Events provide a means for **loosely coupled integration** between reusable services
- **More complex business logic** requires either orchestration or choreography of reusable services
- API Gateway provides **server-side discovery, load balancing, client-specific APIs, protocol translation**, etc.
- Client SDKs are a **convenient way for clients to consume services**



# PLATFORM MANAGEMENT

**Yunghans Irawan**

[yunghans@hotmail.com](mailto:yunghans@hotmail.com)

Total Slides: 71

- To understand and appreciate essential tools in the platform development toolbox
  - **Service Discovery** : How does two services tell each other the location where they are running
  - **Deployment strategies** : How to experiment new features and rollout new versions of services
  - **Circuit Breaker** : How to gracefully handle errors
  - **Distributed Tracing** : How to track a user request across multiple components
  - **Logging and Monitoring** : How to capture metrics and logs for debugging and analytical purposes

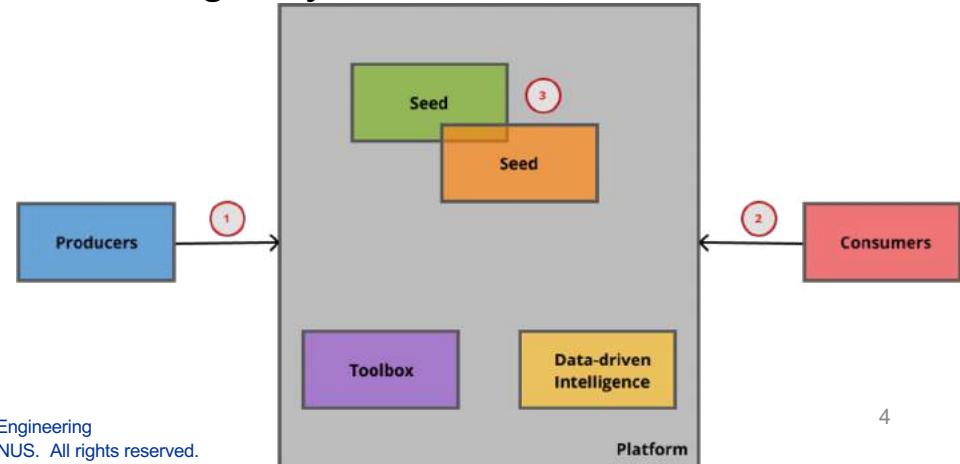


# What is a platform?

“A **platform** is a business model that creates value by facilitating exchanges between two or more interdependent groups, usually **consumers and producers**”

“ Modern platforms **blur the line between infrastructure architecture and application architecture**, needing to address complex issues such as **service discovery, resource coordination, container orchestration and usage reporting** ”

- Simply put, platform just enables the interaction between producers and consumers
- Platforms make the “seeds”, the things that the users are coming to the platform for, easily accessible
  - Videos on Youtube
  - Data and APIs for a Government Health Agency
  - News and opinions - Twitter





# Why are we engineers talking about it?

- Toolbox (aka Services/APIs/Technology)
  - **Scalability** (increasing and limiting the inter)
  - **Availability** (seamless platform experience)
  - **Evolvability** (Releasing bug fixes/new features)
- **Data**



Consistent bad experience would kill your platform



# TOOLBOX

- Service discovery (SA)
- Deployment strategies (E)
  - a. Canary deployments
  - b. Rolling out new versions
- Circuit Breaker & Rate limiting (SA)
- Distributed tracing (A)
- Metrics collection (SA)
- Log aggregation (A)

A canary deployment is a progressive rollout of an application that splits traffic between an already-deployed version and a new version, rolling it out to a subset of users before rolling out fully.

**How are these related to the actual business intent?**

- scaling the interaction
- limiting the interaction
- collecting metrics of these interactions
- giving a seamless experience while enhancing the platform



# SERVICE DISCOVERY

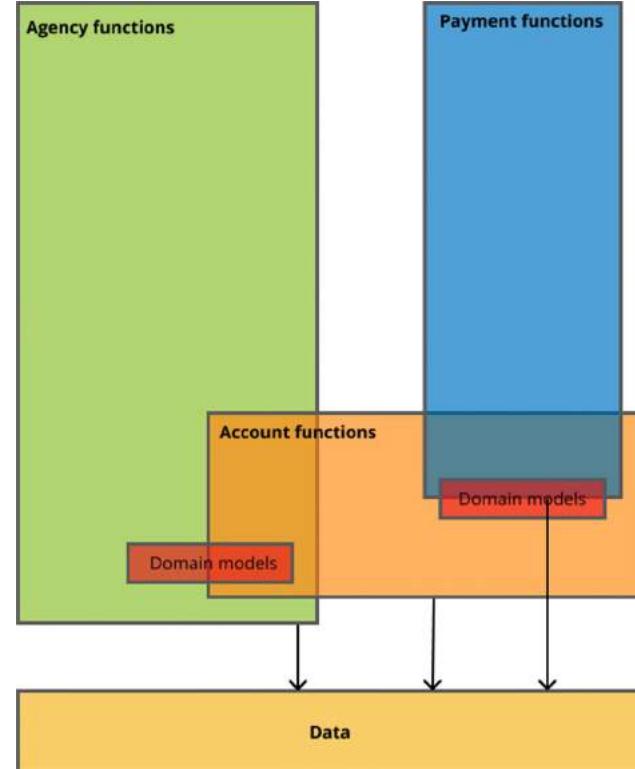


## Pros

- Good performance
- Very less operational overhead
- Code is easier to maintain

## Cons

- Tightly coupled
- Lack of granular scaling





# Service Oriented Architecture

## Pros

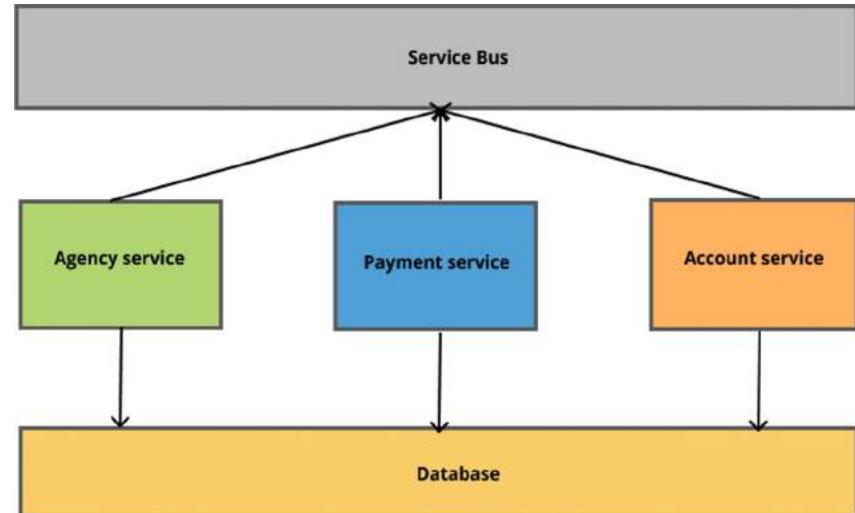
- Better reuse of code
- More ownership
- Supports disparate tech stack

## Cons

- Service bus becomes a bottleneck
- Data resides in a single database

service bus = a message broker

*VM based installations*



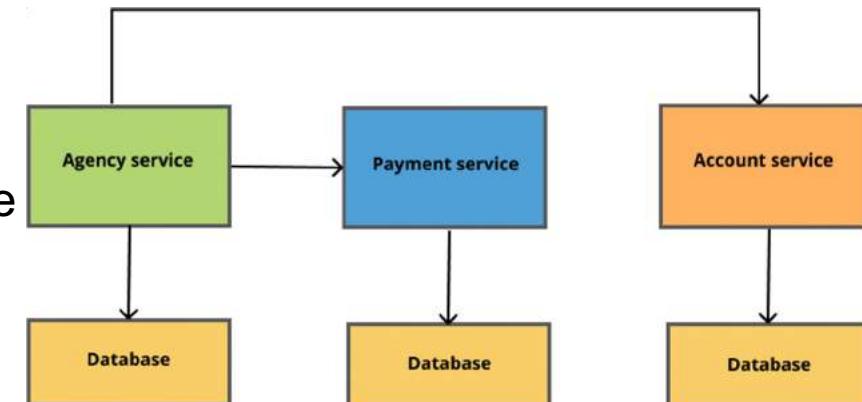


# Microservices

## Pros

- Extreme reuse
- Isolated constructs for building complex systems
- Independent scaling and deployment
- Polyglot tech stack BC!
- Stricter Bounded Context defined
- Ownership - Team culture and morale
  - “You build it, you run it”

*Container based installations*



## Cons

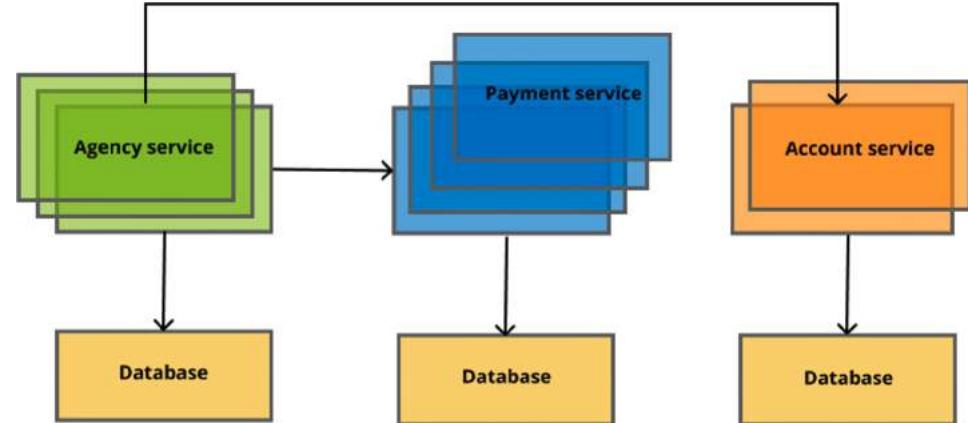
- Significant Operational overhead



# Scaling Microservices

- Each of the services could be scaled independently
- Container based deployments
  - Smaller footprint
- Tech : All cloud providers offer this on their container orchestration offering - Kubernetes, Cloud foundry, AWS ECS

Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms. DevOps teams know applications in containers will run the same, regardless of where they are deployed. Containers allow applications to be more rapidly deployed, patched, or scaled.

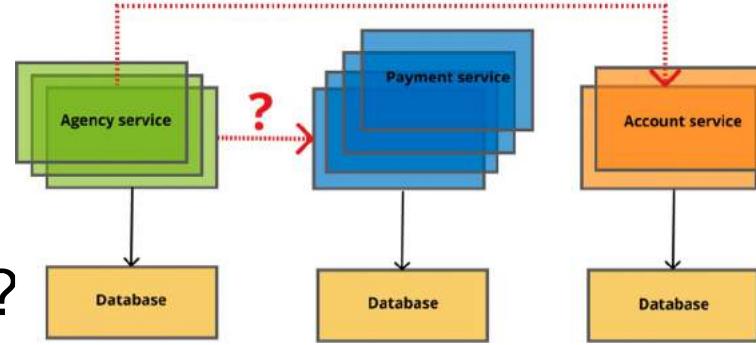




# Problem

## How does the services find each other?

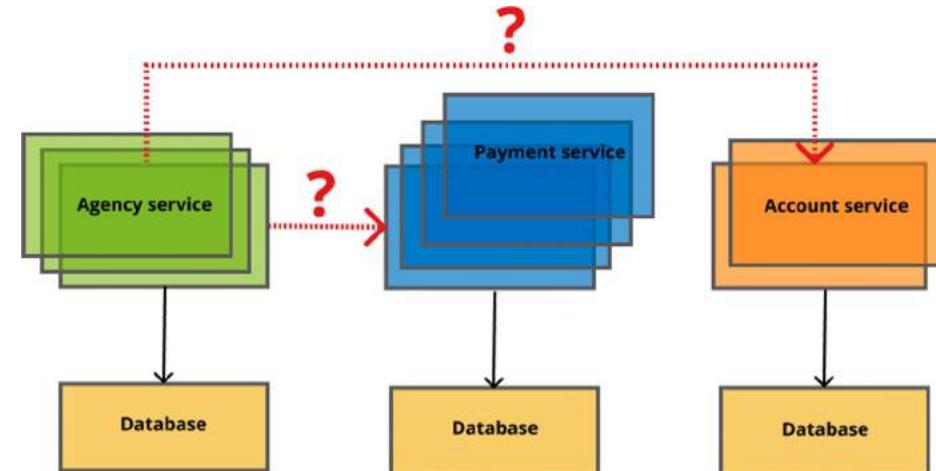
- Would IP address hardcoding work?
  - What happens during auto-scaling during spikes?
    - New instances would be created
    - Can the new instances be used for load balancing
  - What happens during Node/container restarts
    - Cloud environment
    - Fallacies of Distributed computing





# How does the services find each other?

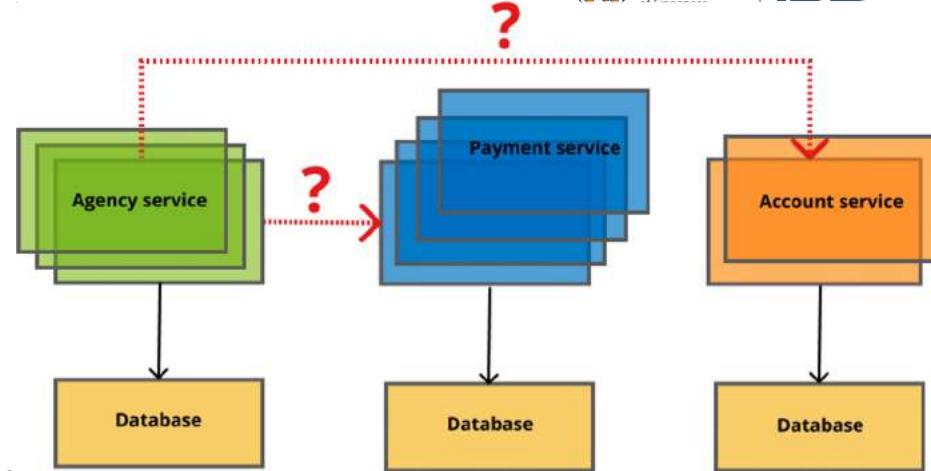
- The solution needs to
  - Be **resilient to node and service failures**
  - **Store** the location (IP) of the services
  - The clients shouldn't need to use the IP address of service for lookups
  - Not be **a Single point of failure**





# How does the services find each other?

- How is the internet doing it?
  - Name based lookups - DNS
- Name based lookups has the following features:
  - Each service has an alias
  - Aliases are stored in a DNS server
  - Callees refer services by alias
  - IP address of the services are looked up from DNS service



The Internet's DNS system works much like a phone book by managing the mapping between names and numbers. DNS servers translate requests for names into IP addresses,

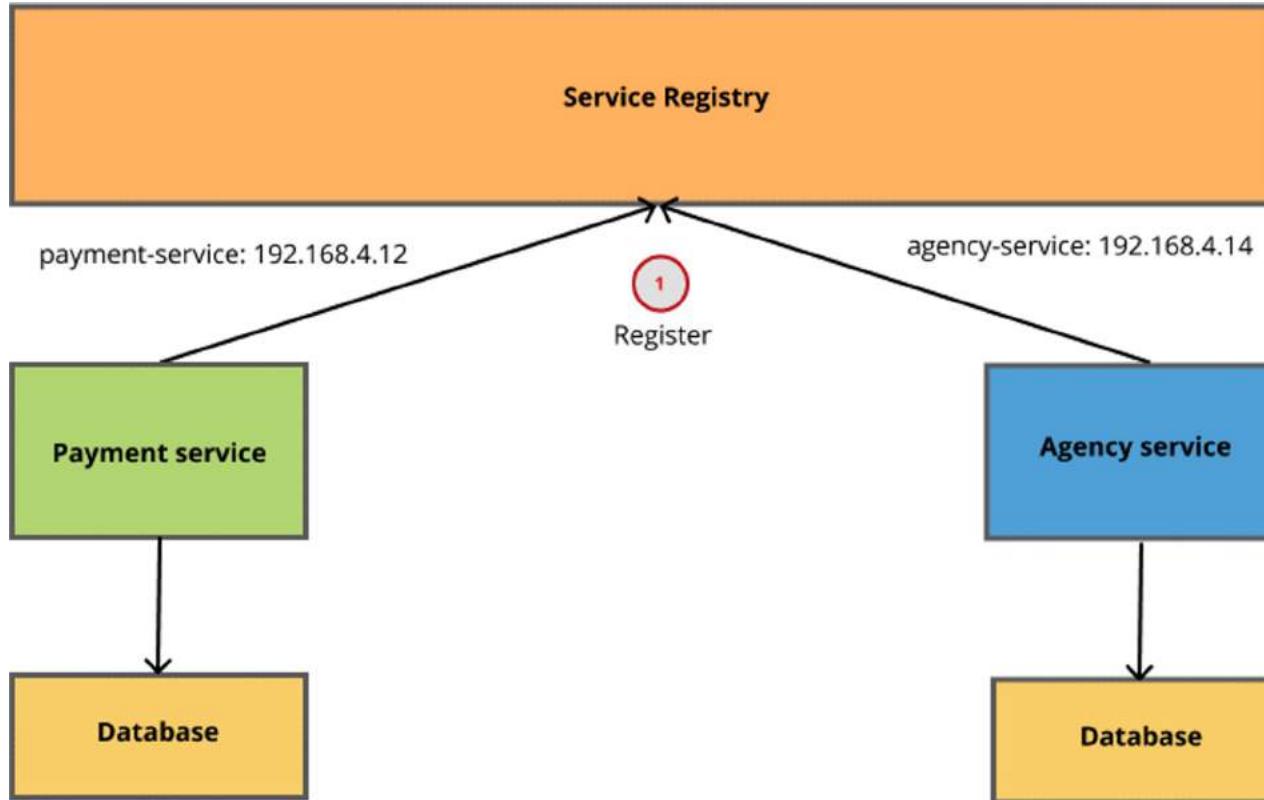


# Service Discovery

- Service Registry daemons are just internal DNS servers
- East-West interactions
  - **East-West** traffic : Traffic **within the services** of the platform
  - North-South traffic: Traffic between users and the platform
- Three-step process of Service Discovery
  - **Register** the service with the service registry
  - **Lookup** the location of the service from the registry
  - **Connect** to the service

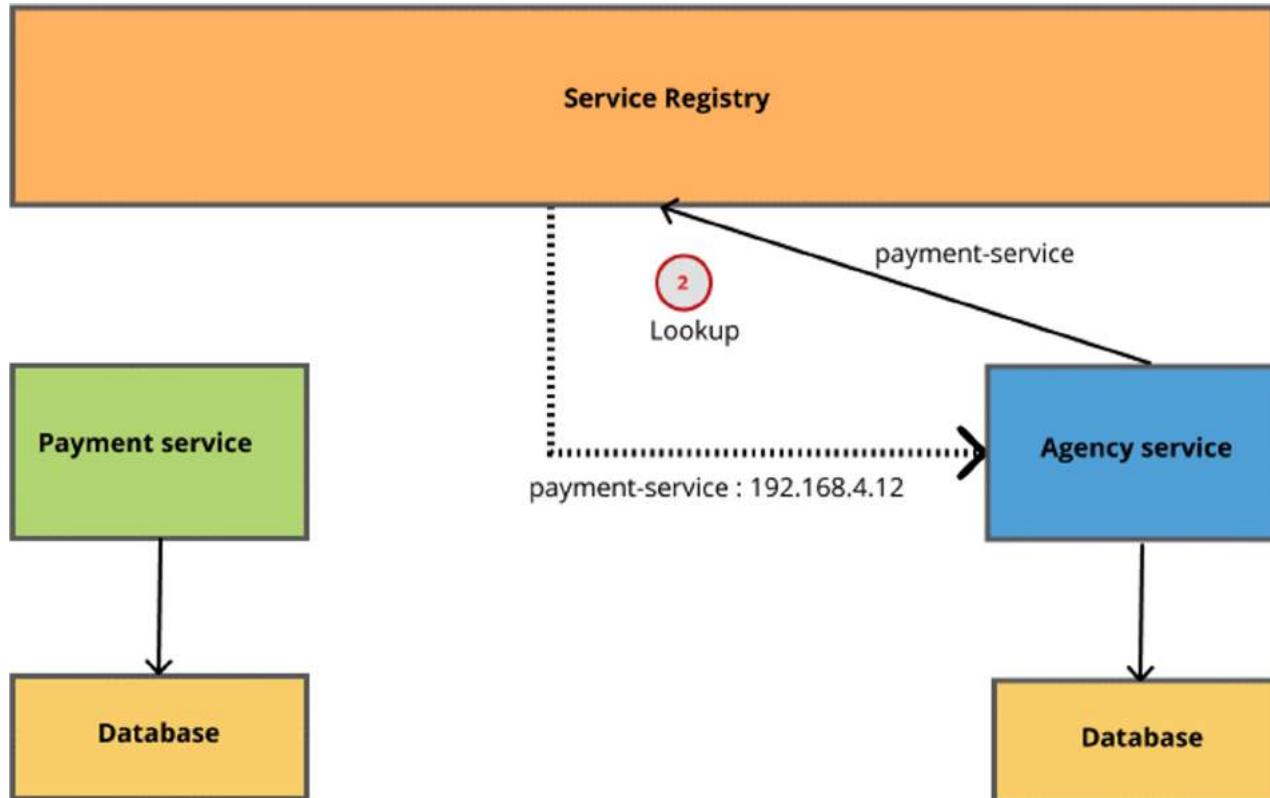


# Service Discovery - 1. Register



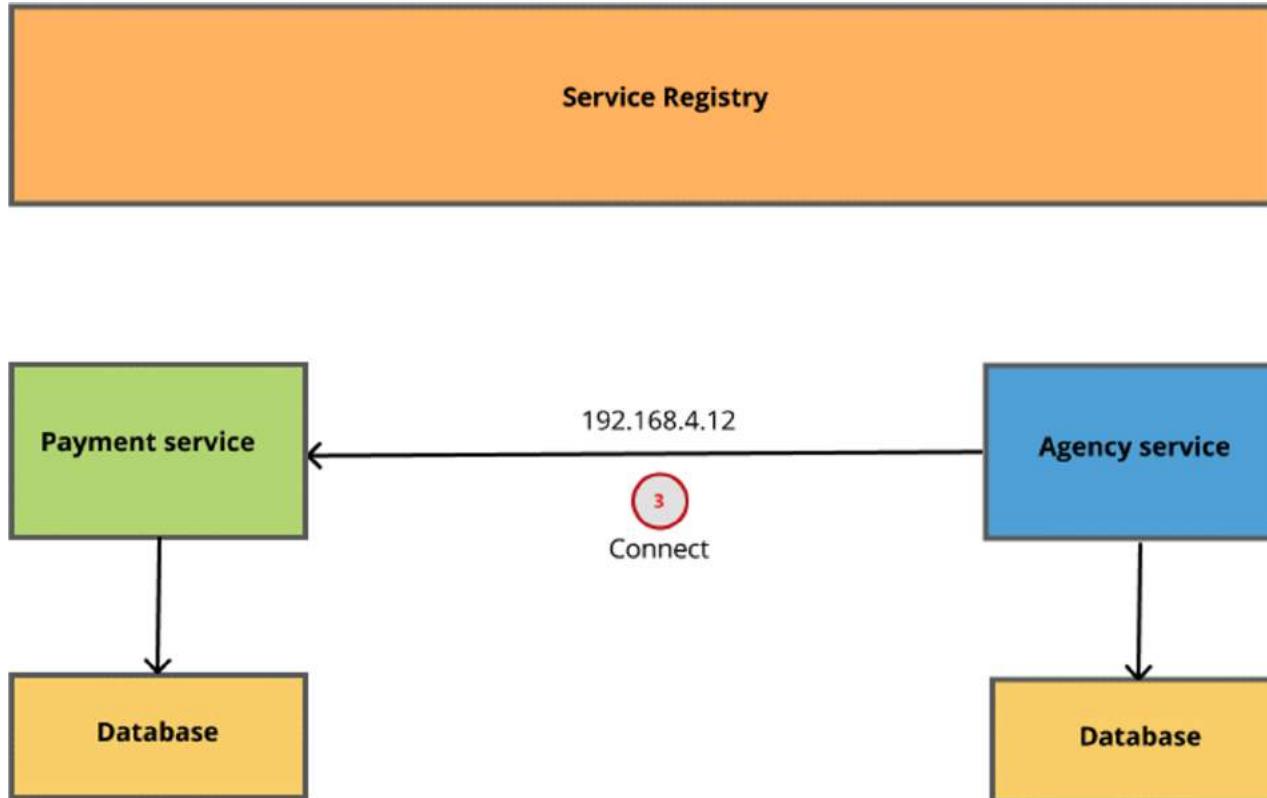


# Service Discovery - 2. Lookup





# Service Discovery - 3. Connect





# Service Discovery – High Availability

- Service discovery service is also Highly Available
  - Typically a cluster of daemons spread over several machines
  - 3,5 or more odd number machines are the norm
- Backed by some consensus protocol (Paxos/Raft/ZAB)
  - Eg. Zookeeper, etcd, consul





# Solutions for Service Discovery

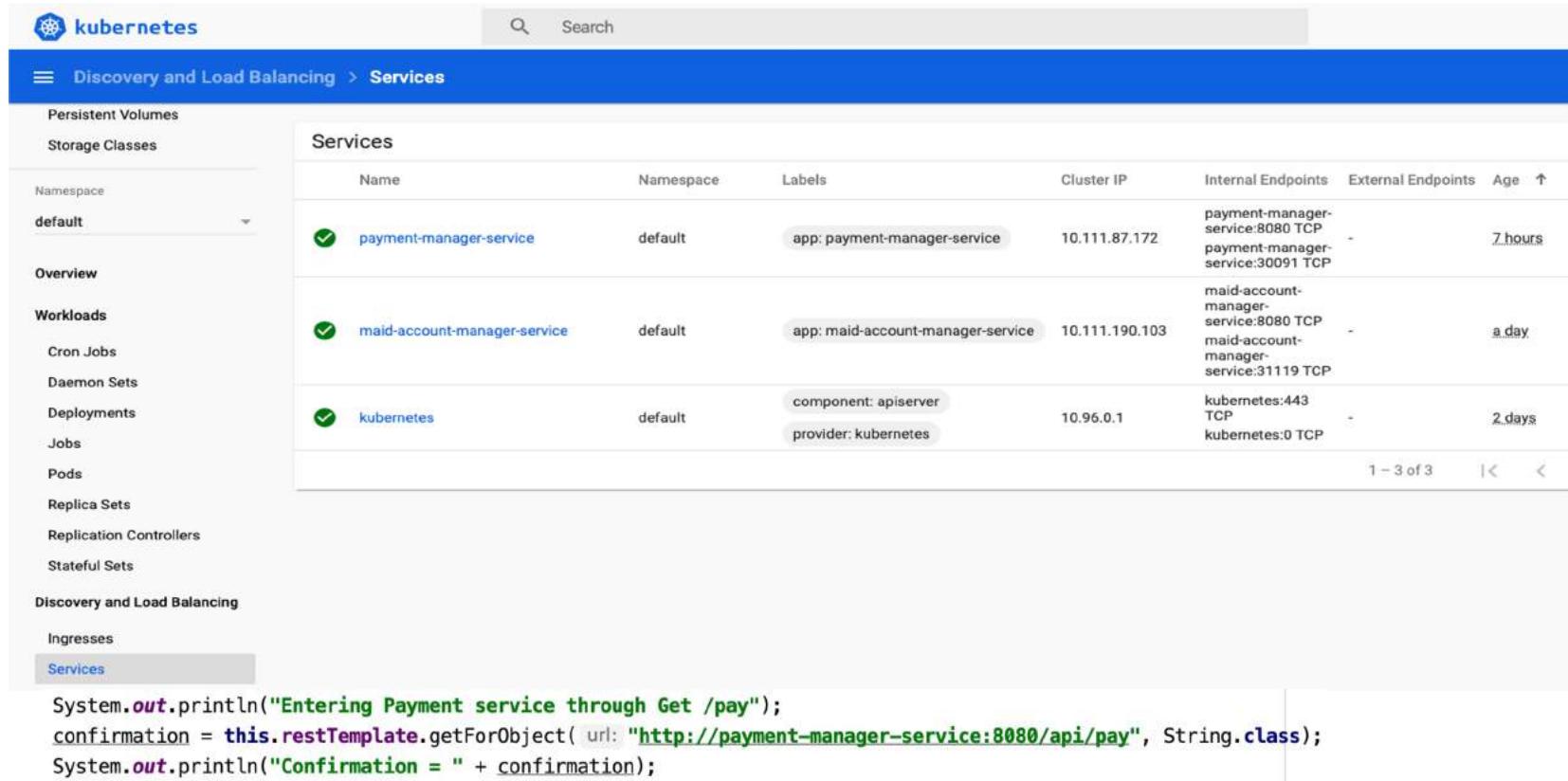
How much code is needed to solve this problem?

Too much code takes away application engineers' time from focussing on business requirements.

- Spring Cloud (rather Netflix) - Eureka server and client - some amount of code
- **Kubernetes has an in-built DNS service** enabling service registry and lookup by alias
  - EKS, AKS, GKS, Openshift
- **Most cloud offerings offer an Out of the box service** discovery for free for their container based deployments - AWS ECS, Azure container service



# Service Discovery on Kubernetes



The screenshot shows the Kubernetes Services page. On the left, there's a sidebar with navigation links like Persistent Volumes, Storage Classes, Namespace (set to default), Overview, Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), and Discovery and Load Balancing (Ingresses, Services). The Services link is currently selected. The main area displays a table of services:

Name	Namespace	Labels	Cluster IP	Internal Endpoints	External Endpoints	Age
payment-manager-service	default	app: payment-manager-service	10.111.87.172	payment-manager-service:8080 TCP payment-manager-service:30091 TCP	-	7 hours
maid-account-manager-service	default	app: maid-account-manager-service	10.111.190.103	maid-account-manager-service:8080 TCP maid-account-manager-service:31119 TCP	-	a day
kubernetes	default	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	2 days

At the bottom, there's a snippet of Java code demonstrating service discovery:

```
System.out.println("Entering Payment service through Get /pay");
confirmation = this restTemplate.getForObject( url: "http://payment-manager-service:8080/api/pay", String.class);
System.out.println("Confirmation = " + confirmation);
```



# DEPLOYMENT STRATEGIES





# Problems

- Who are your customers?
  - Uptime (Nines) - Would downtime of a few minutes upset them? eg. Banks, e-commerce
  - Are your users globally located?
- How often is your release?
  - Are you able to deliver features/enhancements to users as soon as they are built and tested?
- How fast can you rollback your change?
- Can you experiment features in production?
  - A/B tests
  - Passive Machine Learning models

deploy multiple version in production

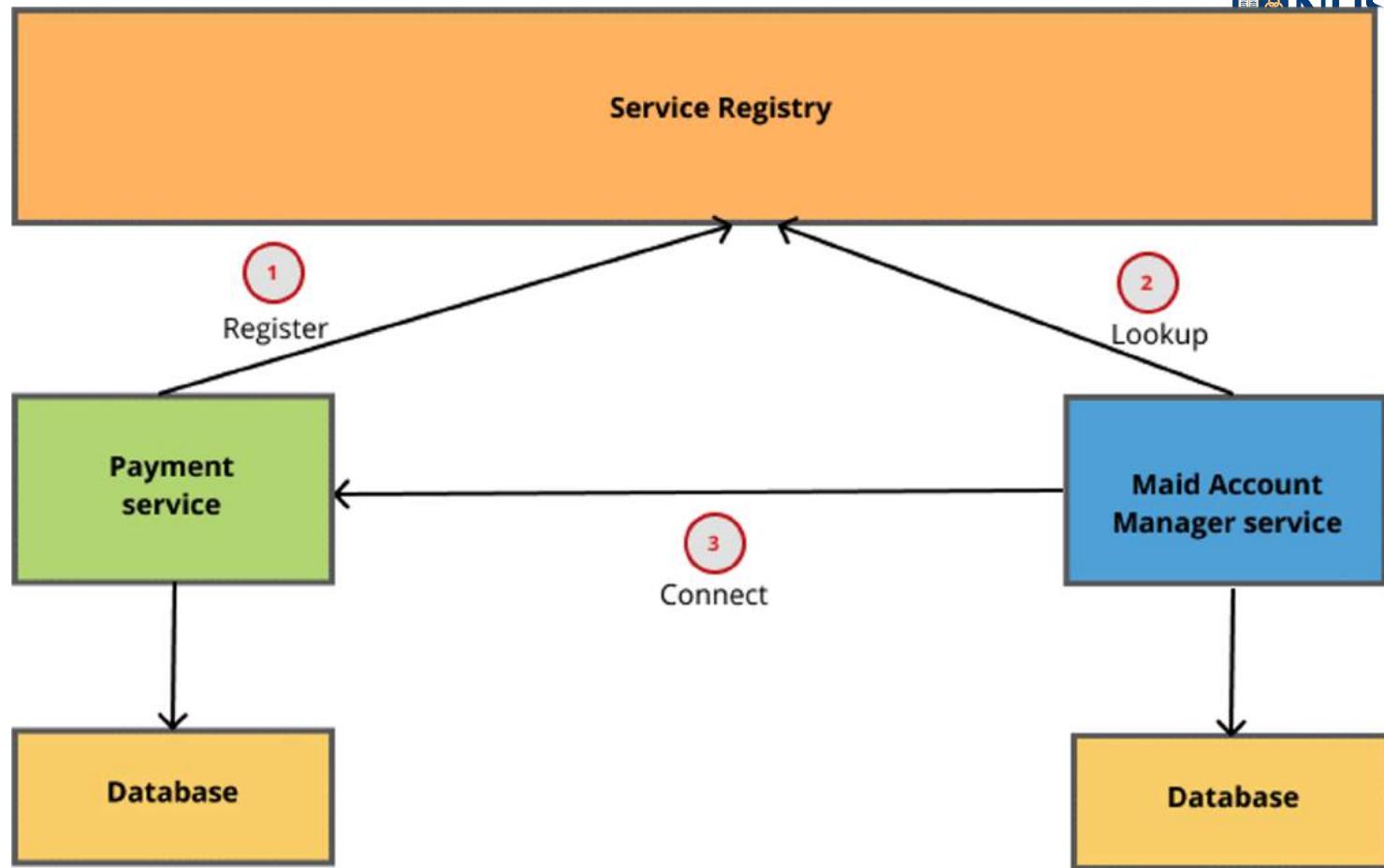


# Deployment strategies

- **Blue/Green** blue/green only expect version 2 work as good as version 1
  - Version 2 is live alongside Version 1 but switched off only after Version 2 is confirmed to be working right
- **Rolling update**
  - Version 2 would progressively replace Version 1 without downtime
- **Canary release**
  - Version 2 handles a small percentage of the overall request that comes to the service.
  - 95-5, 90-10 are popular
- **Shadowing** take longer time than blue/green shadow expect version 2 better than version 1
  - Requests are sent both to Version 1 and Version 2 but Version 2 does not respond to the client



# DEMO - INTRODUCING TWO SIMPLE SERVICES



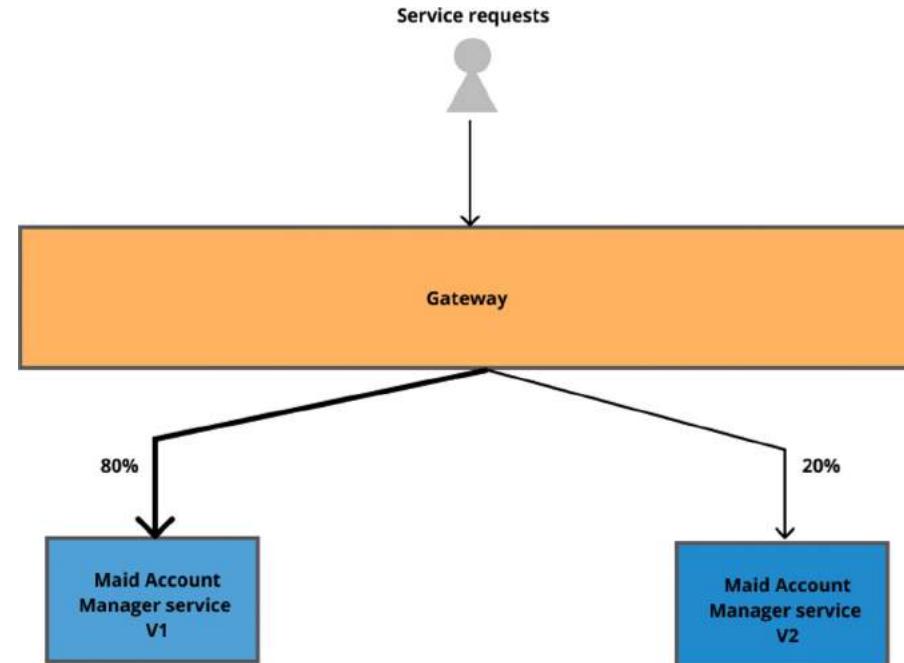


# Canary deployments

- Rolling out releases for a small set of users or servers
- Low risk deployment
- The deployment must be incremental and backward compatible for easier rollbacks

## Primary use-cases :

- Assessing cohort client behavior
- Evaluating new machine learning models
- Experiments/unpredictable behavior of certain features



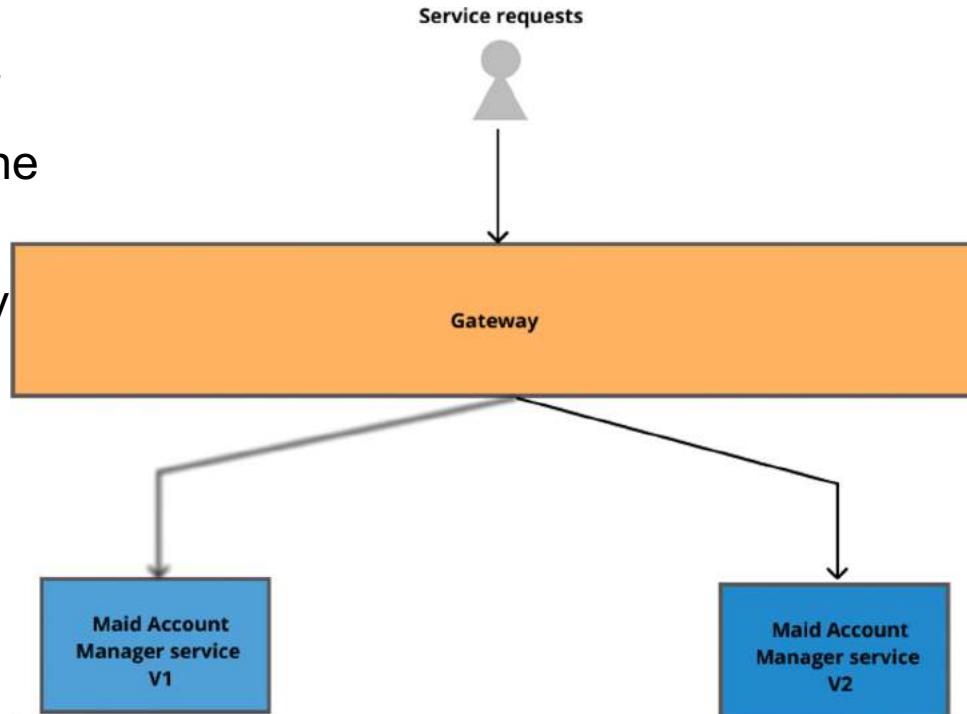


# CANARY DEPLOYMENT - KUBERNETES



# Rolling deployments

- Instead of deploying newer versions of services in one go, gradually replace the old service in servers one by one.
- All traffic to Version 1 will progressively routed to Version 2
- No downtime/loss of requests
- No extra resources (cost)
- Ideal for bug-fixes/patches
- Versions need to be backward compatible for easier rollbacks





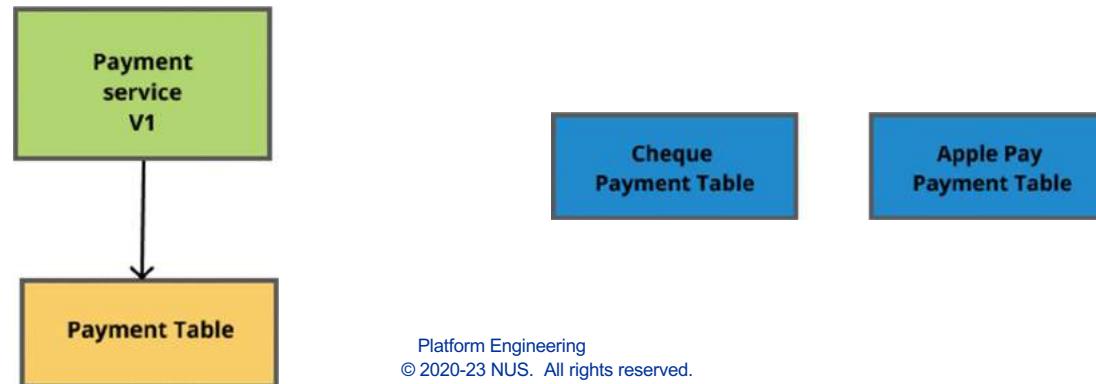
# **SEAMLESS ROLLING DEPLOYMENTS DEMO - KUBERNETES**



# DEPLOYMENT CASE STUDY



- Your Payment microservice had been persisting its data to a Payment table, which stores information for all payment categories - Cheque, ApplePay etc.
- You made your analysis and figured out that you need to split the Payment table into two - one for Cheque and one for Apple Pay because the fields captured were different.
  - Version 1 has been writing to Payment table
  - Version 2 would be writing to ChequePayment and ApplePayPayment



- Old data from Payment table needs to be migrated to new tables
- Change is definitely not backward compatible
  - Retrieval of old payments would not be possible

Requires a fair amount of planning

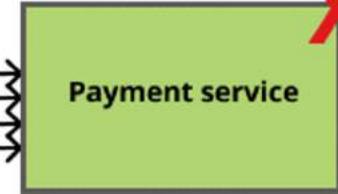
- Make incremental upgrades 1.1, 1.2 ... and finally 2.0



# CIRCUIT BREAKER



Service requests



**Assume the payment service goes down or (even worse) miserably slow that requests time out due to request clogging**

- Agency service would keep making requests and gets errors
- Agency service would have their request threads/thread pool clogged
  - The service cannot make requests even to other services with its threads clogged
  - Requests to Agency service is now clogged - cascading effect



# A possible solution

Service requests

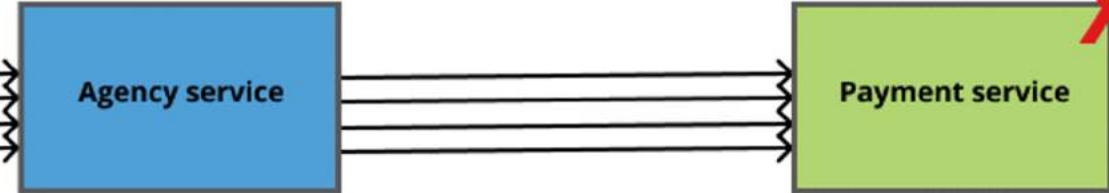


- If the Payment service is returning errors 100% of the time within a window or times out consistently, **don't attempt talking to Payment service for a period of time (interval)**
- Agency service responds to Client **with errors** even without talking to Payment service
- **Retry** - Call the payment service after the interval has expired (for a subset requests)
  - **Exponential backoff** is an approach - try after 50ms, 100 ms, 200ms and so on
- **If the Payment service responds properly, resume calling Payment service for all requests**



# A possible solution

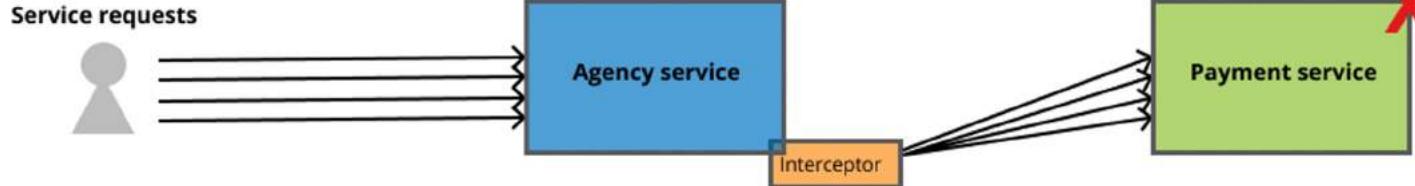
Service requests



- How do we bring in this request forwarding and blocking logic into the Agency service?
  - Could consider coding for it but that's not a business problem
- If Agency is making a REST call directly from code, how do we “intercept” that?
  - New intercepting logic must be written if the protocol is changing -  
[http/gRPC/Thrift](http://gRPC/Thrift)



# Circuit breaker



Circuit breaker libraries are enforced through interceptors

- **Intercepts all requests from Agency service**
- Keeps track of failures within a time window
- **Responds with failures if the error threshold is beyond a threshold**
- **Retries “some” requests with Payment service after a configured time window has elapsed**

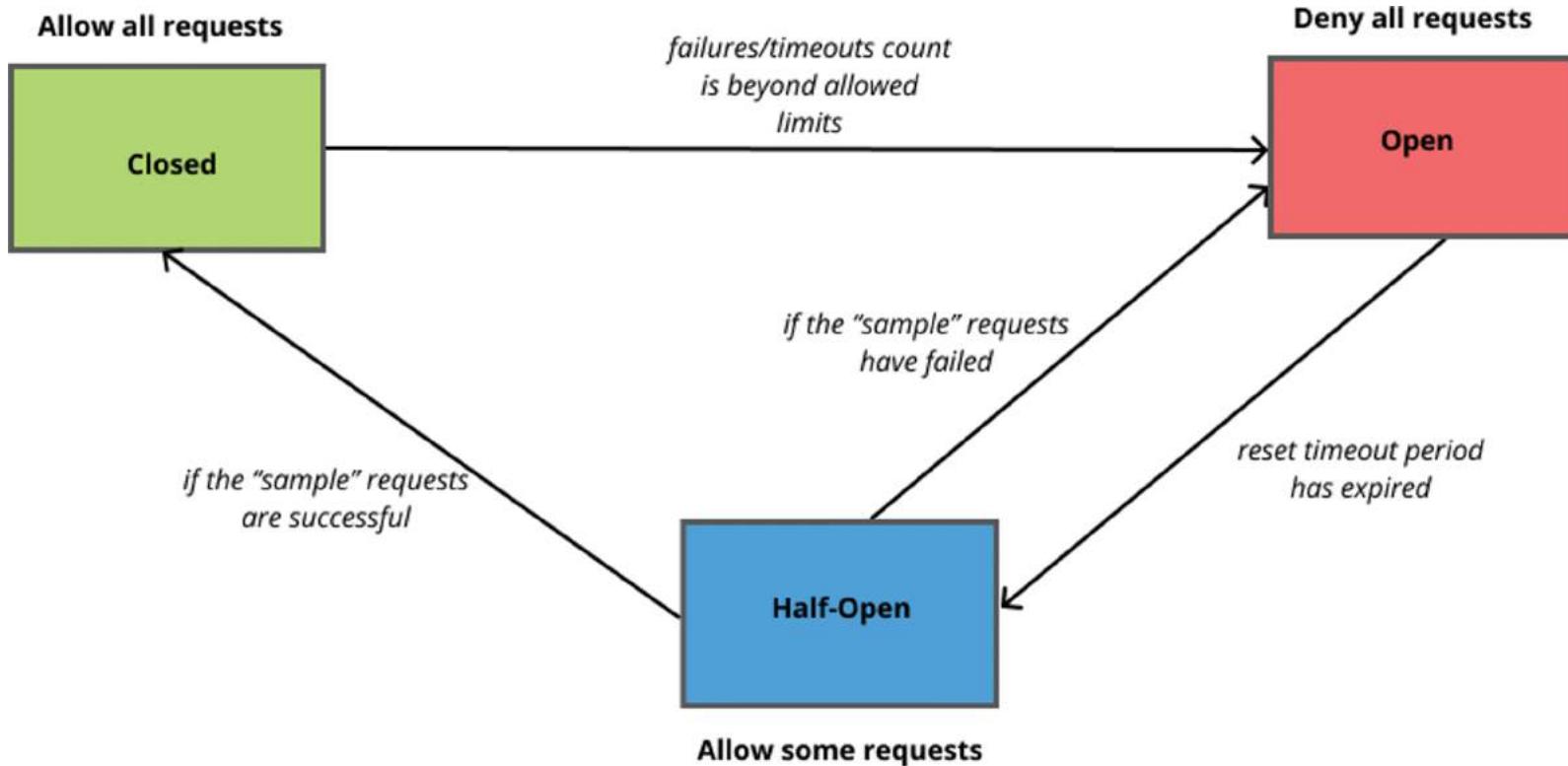
**Very little or no code !**



# Circuit Breaker on Istio

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: payment-manager-service
spec:
  ...
  subsets:
    - name: v1
      labels:
        version: v1
      trafficPolicy:
        connectionPool:
          tcp:
            maxConnections: 1
            connectTimeout: 2s
            ...
        outlierDetection:
          consecutiveErrors: 2
          interval: 2s
          baseEjectionTime: 3s
```

# Circuit breaker is a state machine





# Circuit breaker - Tech options



- Resilience4J
- Sentinel
- Hystrix (Not active)
- Istio service mesh



# DEMO FOR CIRCUIT BREAKING - ISTIO/ENVOY



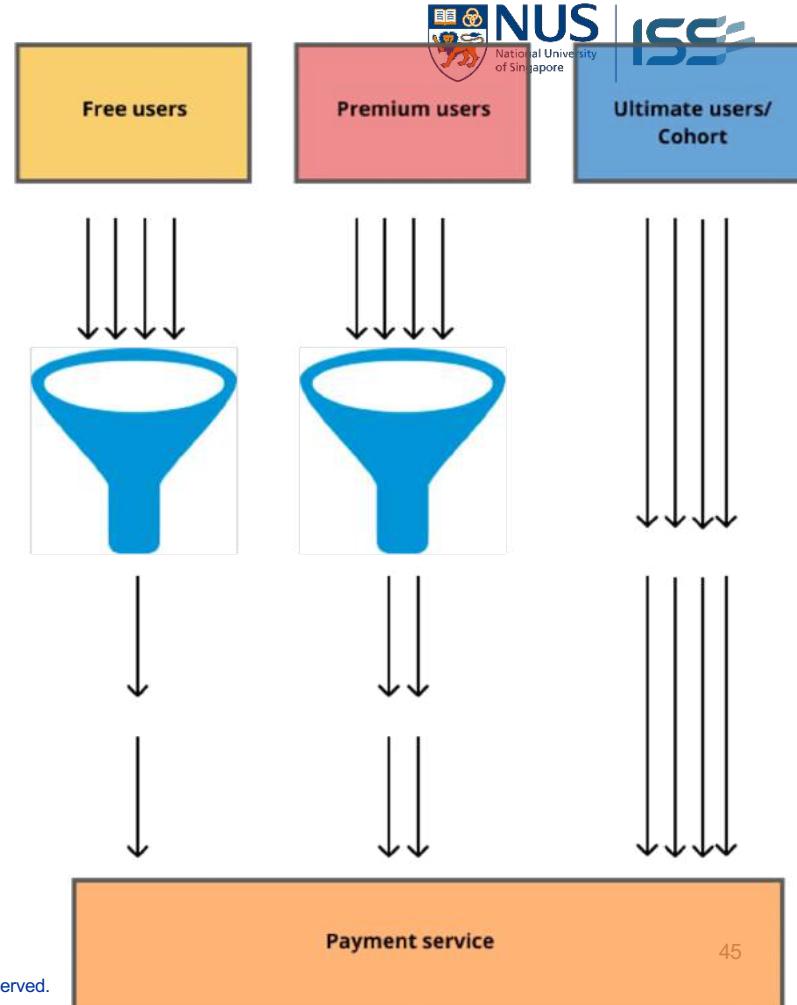
# RATE LIMITING





# Problem

- Not all users in your platform are equal. Some are more equal than others - Conditional rate limits
- Services have limited resources (CPU/Memory/IOPS) and in reality limited spending
- Unrestricted traffic is an invitation to DDoS





# Rate limiting examples



A screenshot of a web browser showing the Twitter developer documentation page for 'Rate limiting'. The URL is developer.twitter.com/en/docs/basics/rate-limit... . The page has a purple header with the Twitter logo and 'Developer' dropdown. The main content area features a large title 'Rate limiting' and a 'Basics' section with a 'Standard' callout box stating 'Standard API endpoints only, does not apply to premium APIs'. Below this are sections for 'Per User or Per Application' and '15 Minute Windows'.



A screenshot of a web browser showing the Facebook developer documentation page for 'Rate Limits'. The URL is developers.facebook.com/docs/graph-api/overvie... . The page has a black header with the Facebook logo and 'facebook for developers' dropdown. The main content area features a large title 'Rate Limits' and a detailed explanation of what rate limits are and how they apply to different API requests.

## Basics

Getting started

Things every developer should know

Frequently asked questions

Twitter developer apps

Developer portal

Authentication

Rate limits

Rate limiting

Response codes

Curorsing

Security

Twitter IDs (snowflake)

# Rate limiting

Standard

Standard API endpoints only, does not apply to premium APIs

## Per User or Per Application

Rate limiting of the standard API is primarily on a per-user basis — or more accurately described, per user access token. If a method allows for 15 requests per rate limit window, then it allows 15 requests per window per access token.

When using [application-only authentication](#), rate limits are determined globally for the entire application. If a method allows for 15 requests per rate limit window, then it allows you to make 15 requests per window — on behalf of your application. This limit is considered completely separately from per-user limits.

## 15 Minute Windows

Rate limits are divided into 15 minute intervals. All endpoints require authentication, so there is no concept of unauthenticated calls and rate limits.

There are two initial buckets available for GET requests: 15 calls every 15 minutes, and 180 calls every 15 minutes.

S-PE/Platform Management.PPT/V2.0

Platform Engineering  
© 2020-23 NUS. All rights reserved.

46

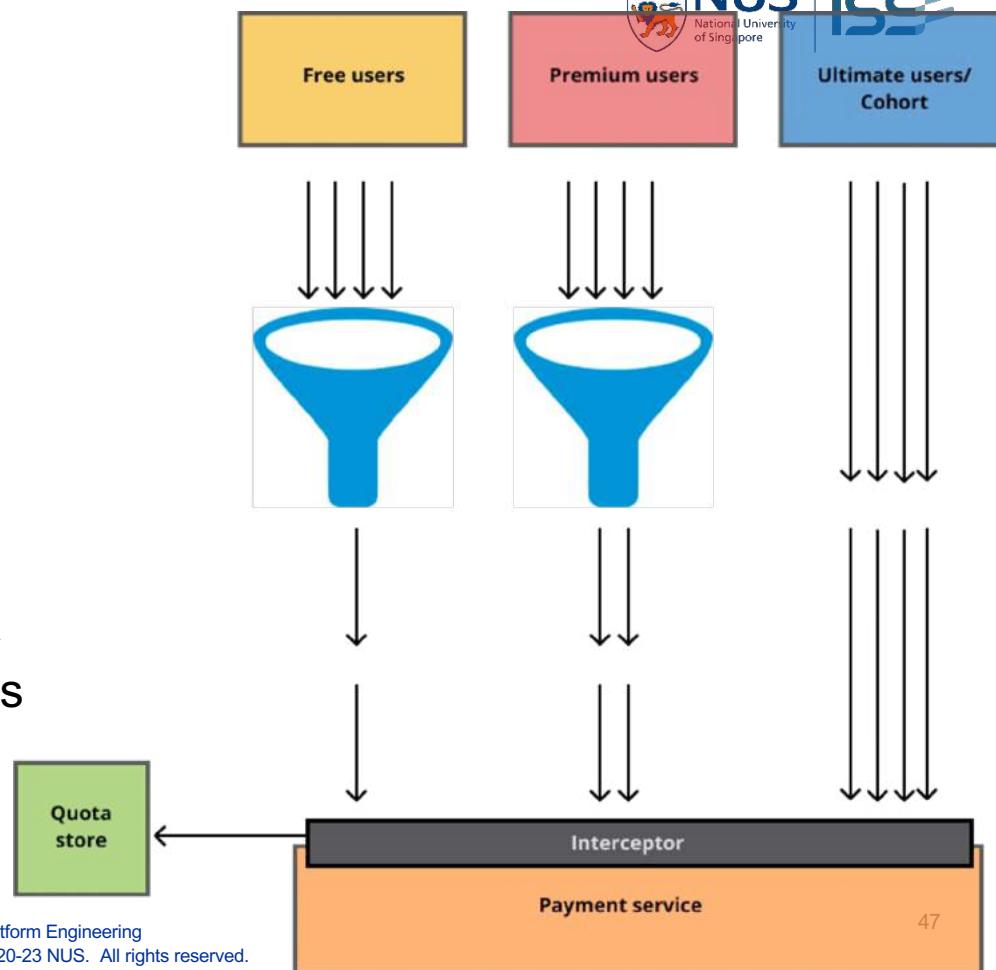


# Rate limiters

Rate limiters are also enforced through interceptors

- Intercepts all requests
- Extracts information such as username, source IP, source service, user plan etc from the request
- Consults a fast (in-memory) Quota database to verify and update limits and usage

Again, very little or no code !





# Rate limiting on Istio

```
apiVersion: "config.istio.io/v1alpha2"
kind: memquota
metadata:
  name: handler
  namespace: istio-system
spec:
  quotas:
    - name: requestcount.quota.istio-system
      maxAmount: 500
      validDuration: 1s
      overrides:
        - dimensions:
            destination: payment
            source: agency
            maxAmount: 1
            validDuration: 5s
        - dimensions:
            destination: payment
            source: "10.28.11.20"
            maxAmount: 200
            validDuration: 1s
```



# Rate limiters - Tech options

- Resilience4J
- Sentinel
- Istio service mesh
- All cloud providers



# DISTRIBUTED TRACING





- Modern platforms have a lot of services that needs to be orchestrated to handle a request
  - Tracing a request across all these services is a hard problem
  - Difficult to debug - Slow turnaround time on fixes
    - MTTR - Mean time to recover increases significantly if the problem could not be traced
  - Loss of effective team communication



# A possible solution



- An UUID created at the root service
- Passed on to nodes and leaf services as HTTP headers
- What is a root service if there are multiple entry-points?
  - Check for header, if it's populated don't generate a UUID
- Distributed tracing libraries do exactly that
  - The root service is generally the library's own daemon that intercepts all requests



# Open tracing - standard distributed tracing

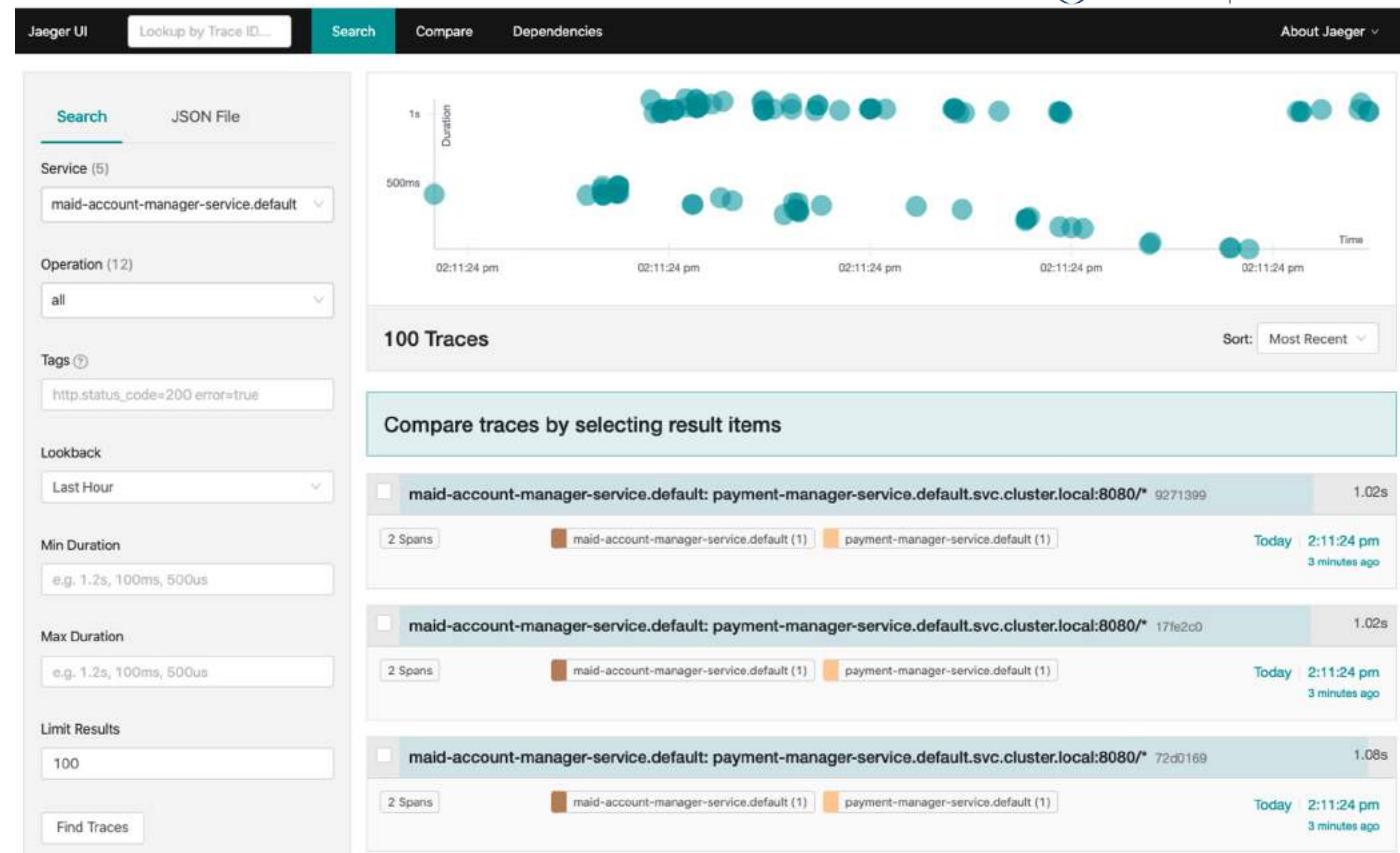


- Vendor neutral
  - Terms
    - **Trace** : represents a transaction that cuts (or spans) across multiple services
    - **Span** : A single unit of work.
      - Typically one service
      - A service can manually create child spans using the Tracer API

```
{
  "data": [
    {
      "traceID": "9271399650b6c72dc8448be74d75305",
      "spans": [
        {
          "traceID": "9271399650b6c72dc8448be74d75305",
          "spanID": "dc8448be74d75305",
          "operationName": "payment-manager-service.default.svc.cluster.local:8080/*",
          "references": [],
          "startTime": 1582438284747849,
          "duration": 1019731,
          "tags": [
            {
              "key": "node_id",
              "value": "1"
            }
          ]
        }
      ]
    }
  ]
}
```

# Tech options

- Jaeger is CNCF
- Spring Sleuth
- Zipkin
- Datadog
- Stackdriver





# DISTRIBUTED TRACING DEMO - JAEGER





# METRICS



- Platform needs to use lesser cost
- APIs need to be faster
- Need to have higher client retention
- Need to deliver better client service
  - SLAs

***“If you can't measure it, you can't improve it”*** - Peter Drucker

- CPU & memory usage
- Request volume – IOPS
- Application availability
- Peak usage time
- Average response time - Outliers and percentiles
- Request success & failure rate
- Request and Response payload size
- Garbage collection time



# Application metrics

- Task success
  - Did the transaction end in success or did the customer drop out?
- Clarity
  - How easy is it for customers to learn and use your product?
  - Heatmaps on websites
- Happiness
  - Returning visitors
  - Frequency of visits



# Key metrics on Scalability and Availability

- Uptime –  $(MTBF - MTTR) / MTBF$ 
  - Mean time between failures & Mean time to recover
  - Measured in nines
  - 5 nines – 5 minutes downtime per year, 4 nines – 50 minutes dpy, 3 nines – 9 hours dpy
- Yield – Successful requests/Total requests
  - Yield is more important than uptime – we could bring down the system when there are no requests
- **Harvest** – Data available/Total data
  - How many servers that has the data are available?

- Prometheus TimeseriesDB + Grafana
- TICK stack - Telegraf, InfluxDB, Chronograf, Kapacitor
- Splunk metrics
- Metrics as a Service platforms



# METRICS DEMO

# PROMETHEUS & GRAFANA



# LOGGING AND MONITORING





# Importance of logging

- Large part of an engineer's life is spent on debugging, analysis and performance tuning.
- Commonly logged information :
  - Authentication/Authorization success and failures
  - Changes to data entities
  - Access to external resources - database, files, message queues
  - Configuration
  - Application errors
  - Feature usage by users



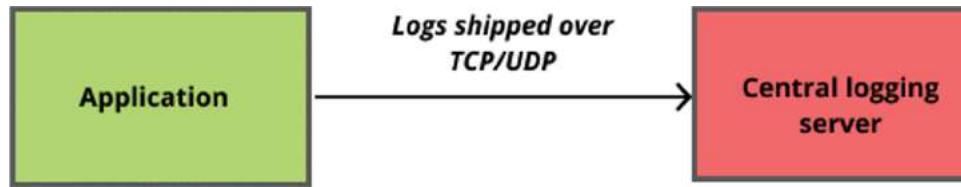
# Log management

- Multiple components make centralized logging very important.
  - Single place to query and analyze logs across several components
  - Alternative is to get into every single machine/container for logs
- Logs are generally shipped to the central log management database from the application



# General log aggregation patterns

- Application ships transformed/formatted logs to Central log server





# General log aggregation patterns

- Application ships raw logs to log transformation and shipping server
- The **log transformation** server parses the logs and transforms to a format that the central log server understands

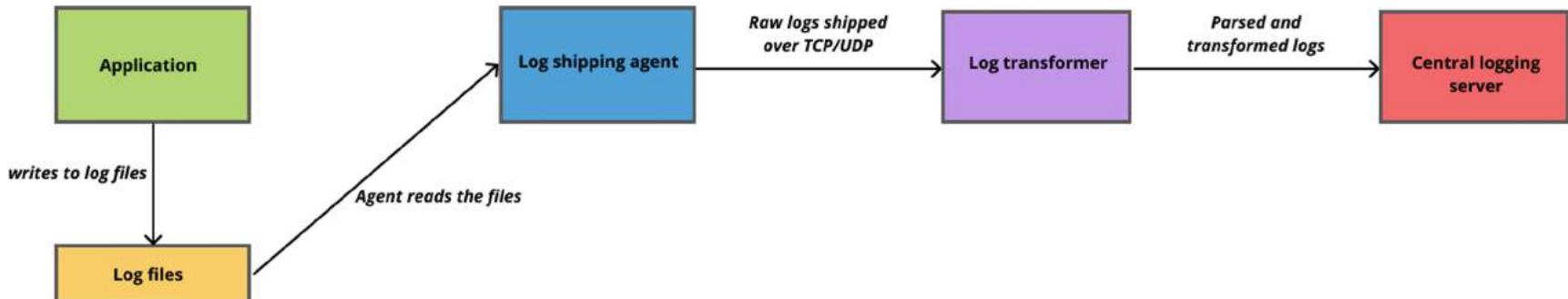


Workshop !



# General log aggregation patterns

- Application writes logs to local file
- A **lightweight log shipping agent** ships raw logs to a remote Log parser/transformer
- The heavy log transformer stores logs to the central log server





# Tech options

- ElasticSearch Logstash Kibana (ELK) and a few other variants of the stack
  - FluentD
  - Filebeat
- Splunk
- Sematext logsene
- AWS Centralized logging
- Google Stackdriver
- Papertrail
- Loggly



# Summary

**Goal : A platform that enables seamless interaction between producers and consumers**

**Seamless interaction** : Deployment strategies - rollout, canary

**Scaling the interaction** : Auto-scaling

**Limiting the interaction** : Circuit breaker, Rate limiting, Retries

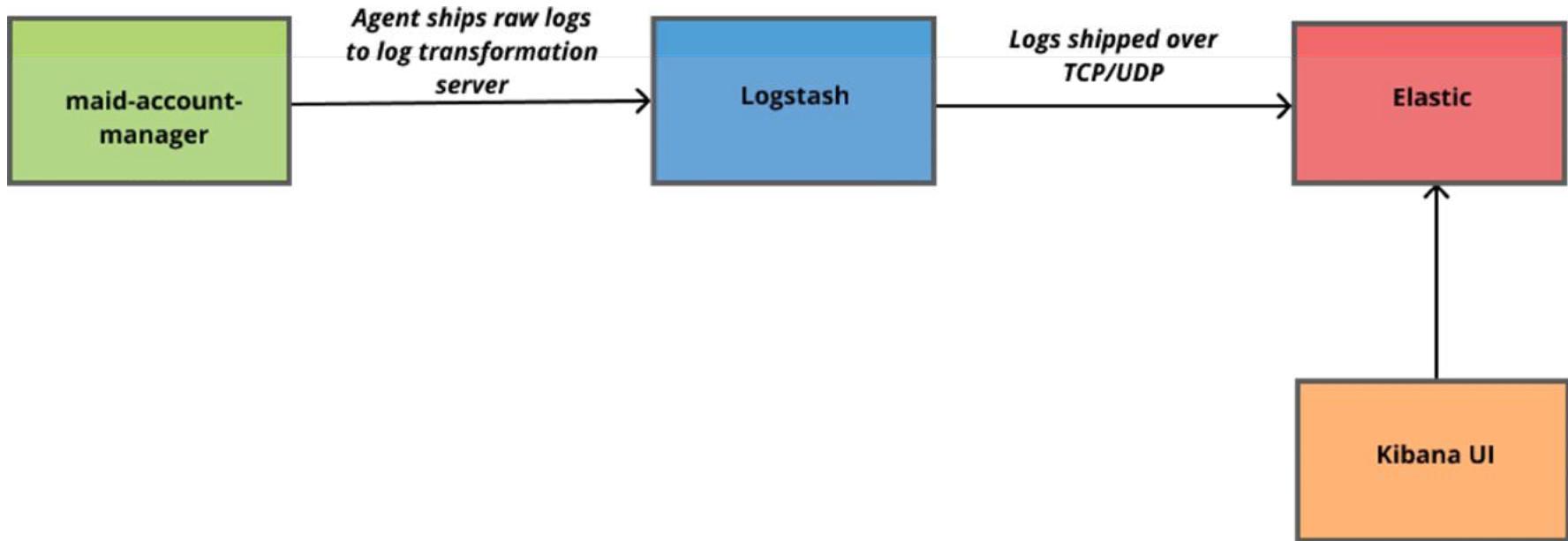
**Experimentation/Traffic management** : Canary deployments

**Data driven intelligence** : Logs and Metrics collection (Observability)

**Lower reaction time to resolve issues** : Logging and monitoring, distributed tracing

**Security** : Mutual TLS, Reducing attack service <Not covered>

# Workshop



# LOGGING & MONITORING WORKSHOP

Platform Engineering

Platform Engineering

## Table of Contents

<b>Clone the workshop .....</b>	<b>2</b>
from codebase <a href="https://github.com/arunma/spring-logmon">https://github.com/arunma/spring-logmon</a> .....	2
<b>Launch all the containers .....</b>	<b>2</b>
on your local machine using docker-compose .....	2
<b>Create dummy logs using Swagger UI.....</b>	<b>2</b>
<b>Check your logs in Kibana .....</b>	<b>4</b>
Create a new Index Pattern.....	5
Let's filter for Errors from the logs.....	6
Try using a richer query criteria .....	7
Filtering all logs from Controller .....	8
<b>Visualize .....</b>	<b>8</b>
Create a pie chart.....	10
<b>End of workshop.....</b>	<b>12</b>
<b>OS Specific instructions .....</b>	<b>13</b>
<b>Common Errors and Solutions.....</b>	<b>18</b>

## Clone the workshop

from codebase <https://github.com/arunma/spring-logmon>

*OS Specific Instructions are available at the bottom of the workshop*

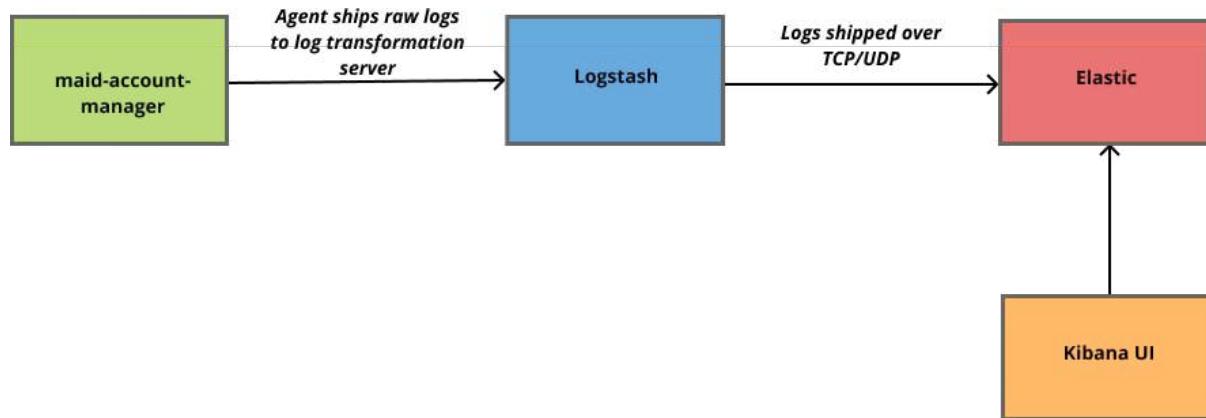
The cloned repository has four containers wrapped in a docker-compose.yml

1. maid-account-manager spring application
2. Elasticsearch datastore
3. Logstash log shipper
4. Kibana UI

## Launch all the containers

on your local machine using docker-compose

```
cd <location of spring-logmon >
# eg. cd /Users/arunma/projects/spring-logmon
docker-compose -f docker/docker-compose.yml up --build
```



## Create dummy logs using Swagger UI

Visit the Swagger UI url on your localhost at <http://localhost:8080/swagger-ui.html>

Expand the maid-account-manager-controller and click “Try it out” a few times.

**Maid Account Manager**

Created by Arun Manivannan  
See more at <https://github.com/arunma>  
[Contact the developer](#)  
[Apache License Version 2.0](#)

**basic-error-controller : Basic Error Controller** Show/Hide | List

**maid-account-manager-controller : Maid Account Manager Controller** Show/Hide | List

**GET /api/bill/get**

**Response Class (Status 200)**  
OK

Model Example Value

```
{  
    "amount": 0,  
    "description": "string",  
    "id": 0  
}
```

Response Content Type: application/json

Response Messages

HTTP Status Code	Reason	Response Model
401	Unauthorized	
403	Forbidden	
404	Not Found	

Try it out!

The MaidAccountController's implementation creates exception roughly 40% of the time.

```
import java.time.Month;
import java.time.format.TextStyle;
import java.util.Date;
import java.util.Locale;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;

@RestController
@RequestMapping(MaidAccountManagerController.BASE_URL)
public class MaidAccountManagerController {

    public static final String BASE_URL = "/api/bill";
    private Logger log = LoggerFactory.getLogger(MaidAccountManagerController.class);

    @GetMapping(path = "/get")
    @ResponseStatus(HttpStatus.OK)
    public MaidUsageBill getBill() {
        double random = new Random().nextDouble();
        if (random < 0.4) {
            log.error("Random Exception for value " + random, new RuntimeException("Random Exception is thrown @ " + new Date()));
        }
        String randomMonth = Month.of(ThreadLocalRandom.current().nextInt(1, bound: 12)).getDisplayName(TextStyle.FULL, Locale.US);
        String randomYear = String.valueOf(ThreadLocalRandom.current().nextInt(origin: 2000, bound: 2020));
        int id = Math.abs(ThreadLocalRandom.current().nextInt());
        double amount = Math.ceil(ThreadLocalRandom.current().nextDouble(origin: 500, bound: 1200));
        log.info("Returning Maid Usage for the period {} to {} with Id {} and Amount {}", randomMonth, randomYear, id, amount);
        return new MaidUsageBill(
            id,
            amount,
            String.format("Version 2 of the API - Payment for %s %s", randomMonth, randomYear)
        );
    }
}
```

## Check your logs in Kibana

Once enough logs have been created, go to the Kibana UI at  
<http://localhost:5601/app/kibana>

## Create a new Index Pattern

The screenshot shows the Kibana Management interface with the path "Management / Index patterns / Create index pattern". On the left, there's a sidebar with sections for Elasticsearch (Index Management, Index Lifecycle Policies, Rollup Jobs, Transform, Remote Clusters, Snapshot and Restore, License Management, 8.0 Upgrade Assistant) and Kibana (Index Patterns, Saved Objects, Spaces, Reporting, Advanced Settings). A modal window titled "Create index pattern" is open. It contains a sub-section "Step 1 of 2: Define index pattern" with an "Index pattern" input field containing "maid-account-manager-\*". Below the input field, a note says "You can use a \* as a wildcard in your index pattern. You can't use spaces or the characters \, /, ?, \*, <, >, |.". A green success message says "Success! Your index pattern matches 1 index." followed by "maid-account-manager-2020.03.01". At the bottom of the modal, it says "Rows per page: 10".

This screenshot shows the continuation of the "Create index pattern" process. The modal has moved to "Step 2 of 2: Configure settings". It displays the message "You've defined maid-account-manager-\* as your index pattern. Now you can specify some settings before we create it." Below this, a "Time Filter field name" dropdown is open, showing "@timestamp" selected. A tooltip for this field explains: "I don't want to use the Time Filter" and "You can use your own timestamp field instead". There are also "Show advanced options" and "Create Index pattern" buttons at the bottom right.

**Click the discovery button**

Management / Index patterns / maid-account-manager-\*

Help us improve the Elastic Stack

Discover about how usage data helps us manage and improve our products and services, see our Privacy Statement. To stop collection, disable usage data here.

Dismiss

Elasticsearch

- Index Management
- Index Lifecycle Policies
- Rollup Jobs

★ maid-account-manager-\*

Time Filter field name: @timestamp Default

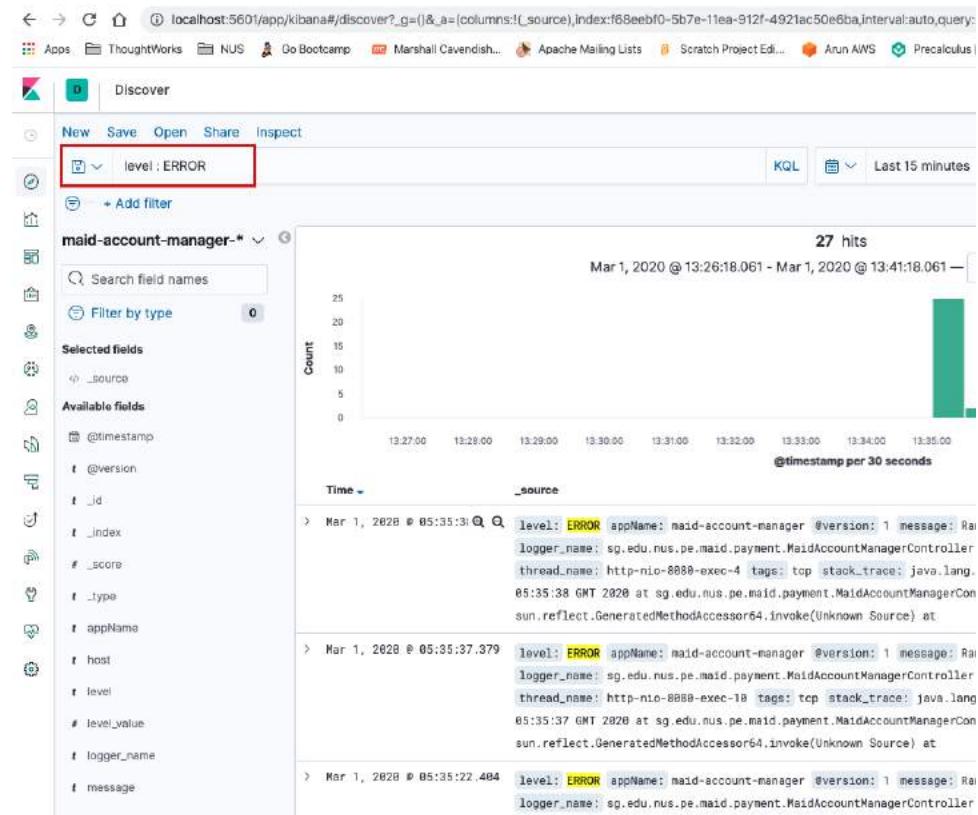
You should see all the logs already captured

Time	_source
Mar 1, 2020 @ 05:35:38.199	app Name: maid-account-manager _version: 1 _index: maid-account-manager-2020.03.01 _id: LKXKJHABKYNUTMWZER _type: applog maid-account-manager-2020.03.01 _score: 1
Mar 1, 2020 @ 05:35:38.198	app Name: maid-account-manager _version: 1 _index: maid-account-manager-2020.03.01 _id: LKXKJHABKYNUTMWZER _type: applog maid-account-manager-2020.03.01 _score: 1

## Let's filter for Errors from the logs

In the “Search” bar, issue the following search criteria. You can also see the count of errors in the histogram.

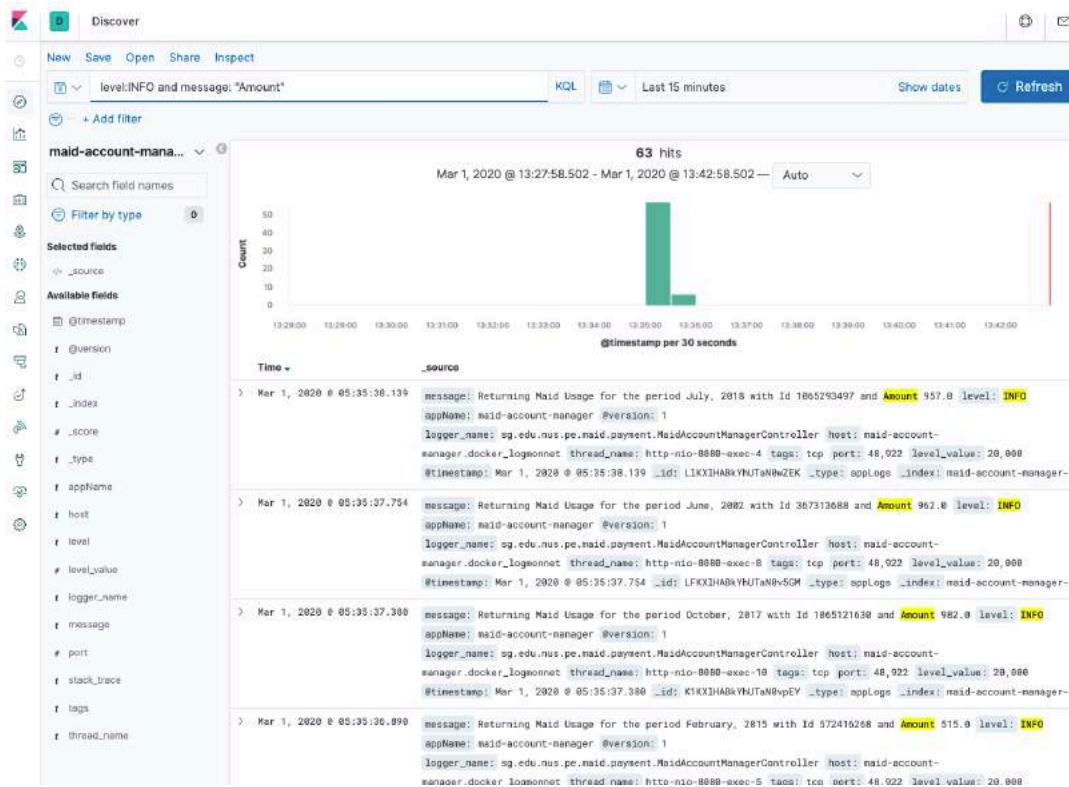
`level:ERROR`



## Try using a richer query criteria

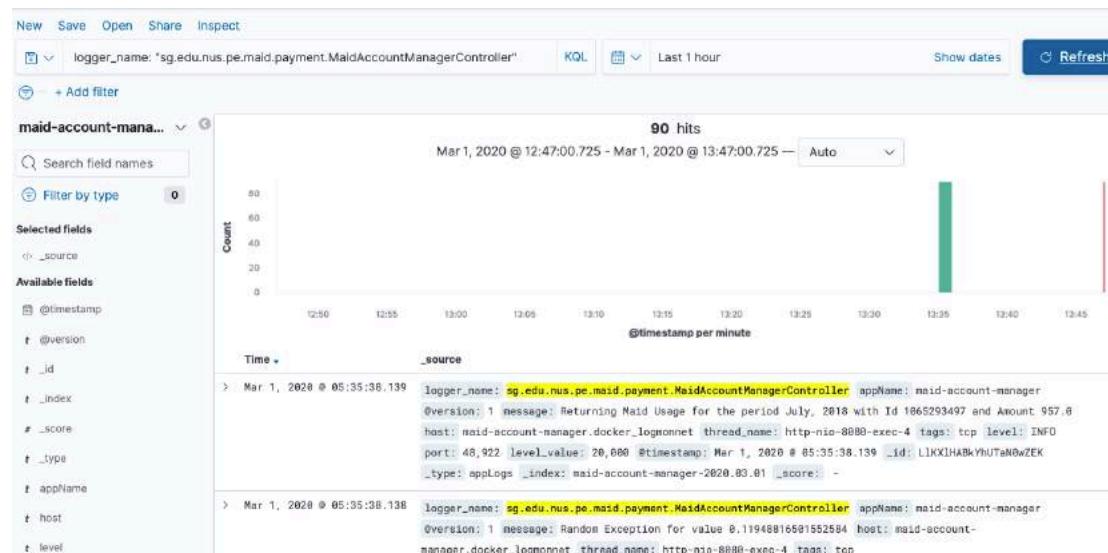
Filter INFO level logs that has "Amount" in the log message

level:INFO and message: "Amount"



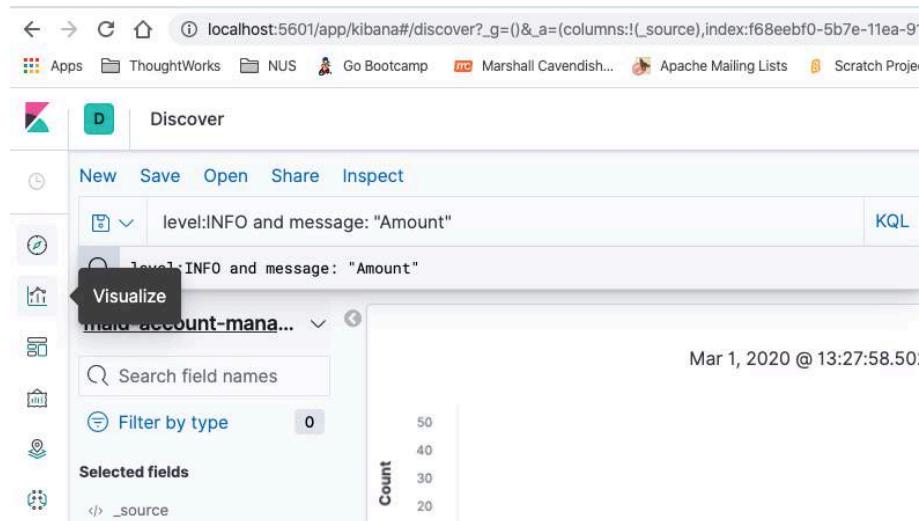
## Filtering all logs from Controller

logger\_name: "sg.edu.nus.pe.maid.payment.MaidAccountManagerController"

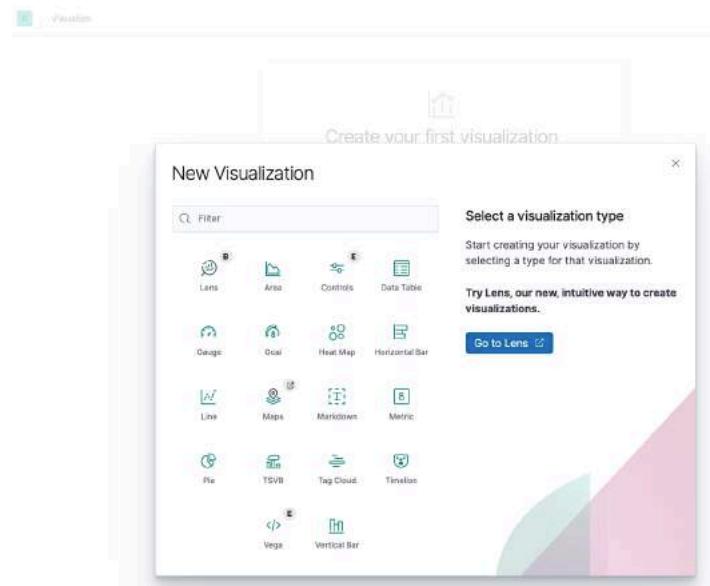


Looks like we have 90 log messages generated from the MaidAccountManagerController. Let's visualize how much of these are errors and how much are good responses.

## Visualize



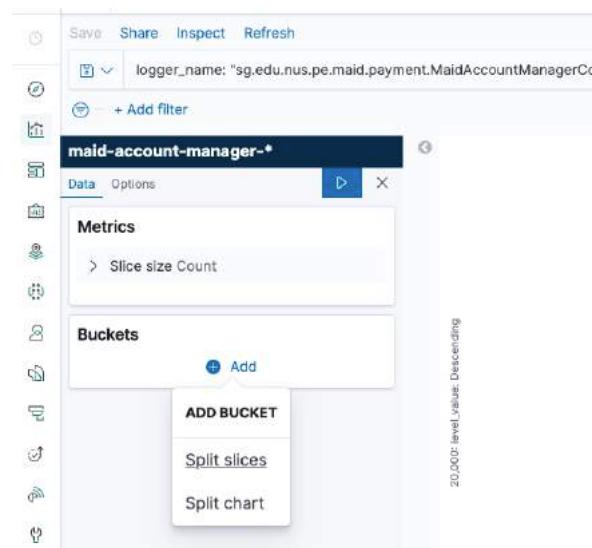
## Create a pie chart



## New Pie / Choose a source



## Create a new Split slice



## Select “Terms”

The screenshot shows a 'Buckets' configuration interface. At the top left is a 'Split slices' section with a dropdown arrow, a refresh icon, and a close button. Below it is an 'Aggregation' section with a dropdown menu labeled 'Select an aggregation'. The menu is open, displaying several options: 'Date Range', 'Filters', 'Histogram', 'IPv4 Range', 'Range', 'Significant Terms', and 'Terms'. The 'Terms' option is highlighted with an underline.

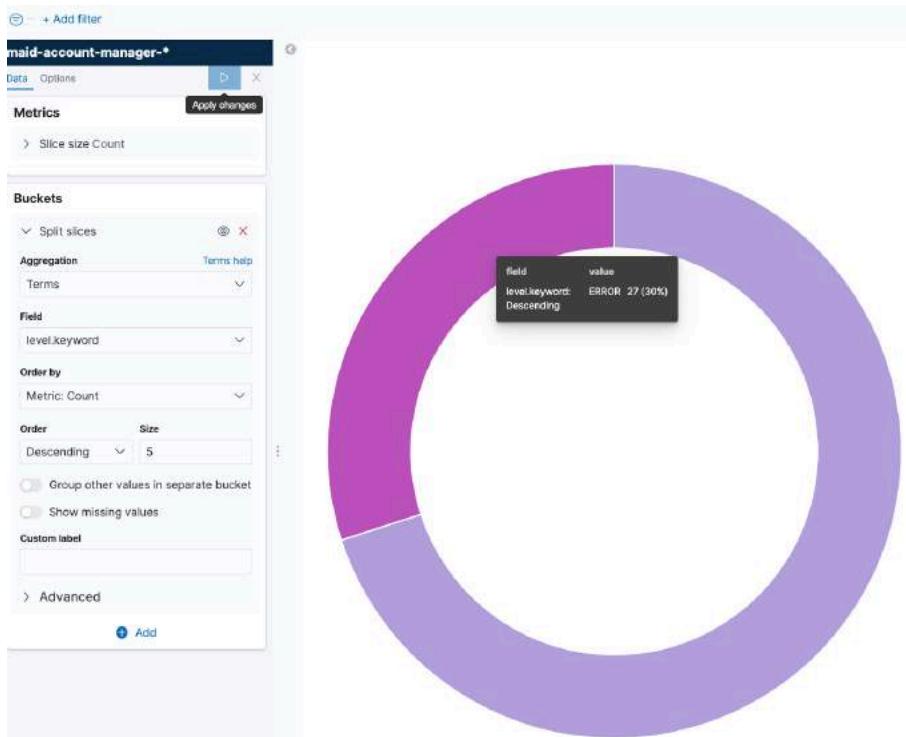
In the Field, select “level\_keyword”

The screenshot shows a dashboard titled 'maid-account-manager-\*'. The top navigation bar has 'Data' and 'Options' tabs, with 'Data' currently selected. The main area is divided into sections: 'Metrics' (containing 'Slice size Count'), 'Buckets', and 'Advanced'. The 'Buckets' section is expanded, showing a 'Split slices' section with a dropdown arrow, a refresh icon, and a close button. Below it is an 'Aggregation' section with a dropdown menu set to 'Terms'. Underneath is a 'Field' section with a dropdown menu set to 'level.keyword'. Further down are 'Order by' and 'Order' settings, both currently set to 'Metric: Count' and 'Descending' respectively. There are also checkboxes for 'Group other values in separate bucket' and 'Show missing values', and a 'Custom label' input field. At the bottom of the 'Buckets' section is an 'Advanced' section with a 'Add' button.

Click “Apply changes”



We now have a live visualisation of the error/good responses.



---

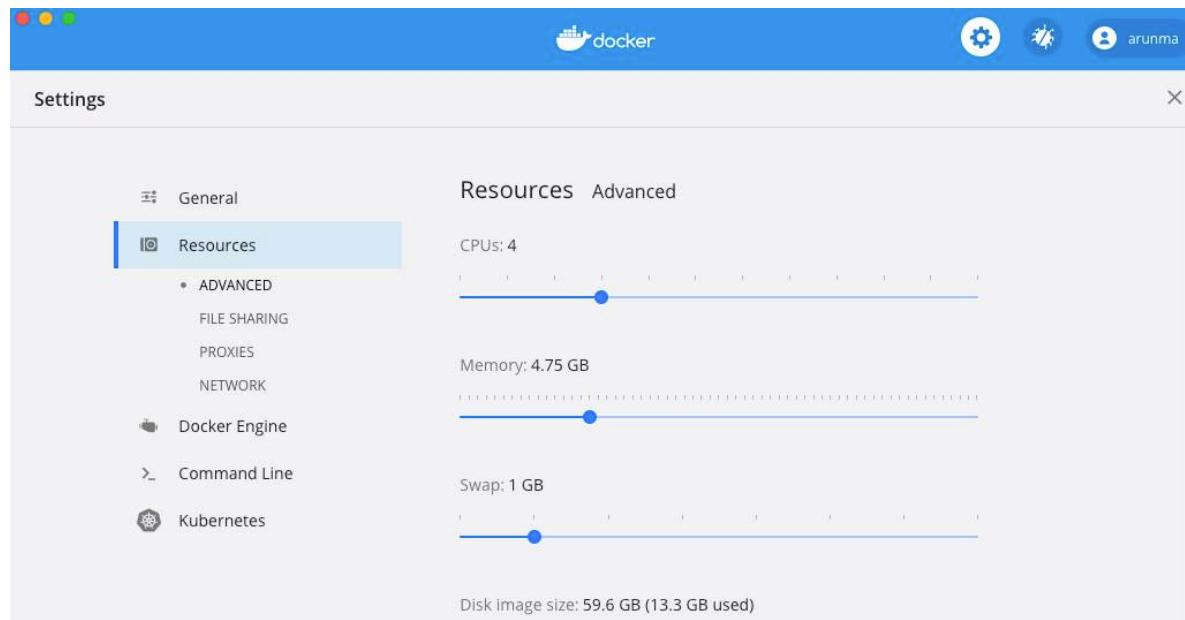
**End of workshop**

## OS Specific instructions

### Mac & Windows Pro Special Instructions:

Extra Adjust your docker settings

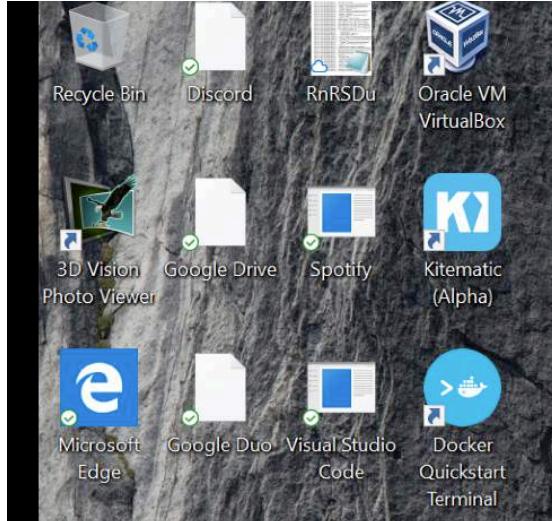
Give at least 4 cores and 3 GB memory



## Windows Home Special instructions (for windows >7)

1. Download Docker toolbox from <https://github.com/docker/toolbox/releases>
2. Install Linux VM

- a. You must see the Docker Quickstart Terminal on your desktop



- b. Upon clicking, it would create a new VM inside your Oracle Virtual Box (which was installed as part of your toolbox installation process)

```
(default) Starting the VM...
(default) Check network to re-create if needed...
(default) Windows might ask for the permission to configure a dhcp server. Sometimes zed in the taskbar.
(default) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...

This machine has been allocated an IP address, but Docker Machine could not reach it successfully.

SSH for the machine should still work, but connecting to exposed ports, such as the Docker daemon port (usually <ip>:2376), may not work properly.

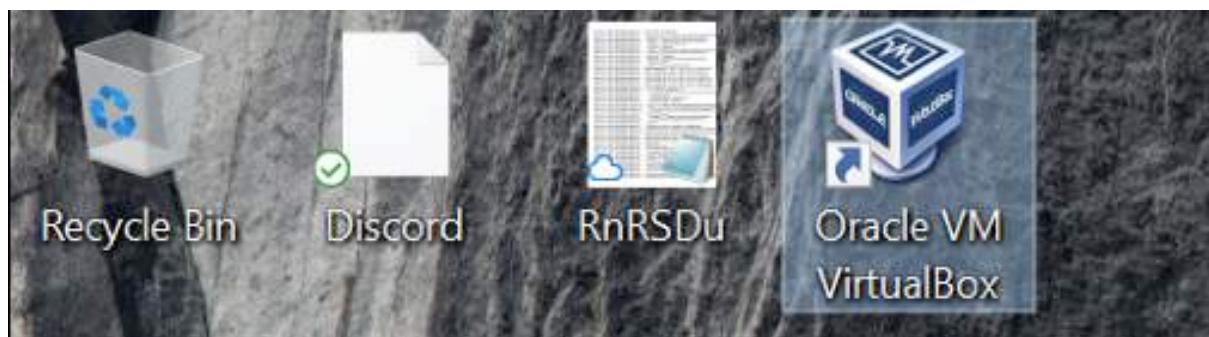
You may need to add the route manually, or use another related workaround.

This could be due to a VPN, proxy, or host file configuration issue.

You also might want to clear any VirtualBox host only interfaces you are not using.
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual
ker Toolbox\docker-machine.exe env default
```

- c. You must see the Docker ASCII art on your screen

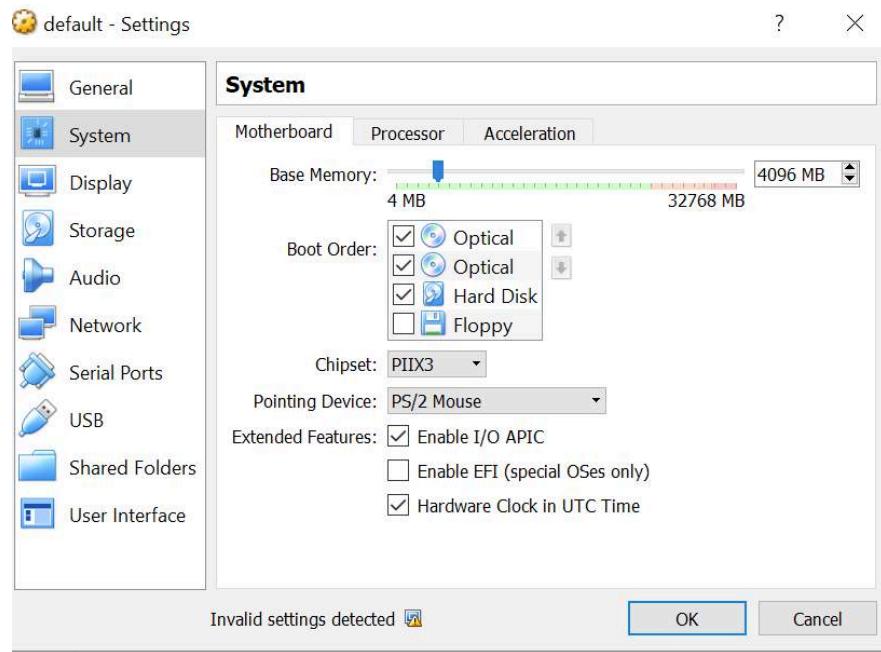
3. Adjust your memory and processor settings:
    - a. Open your virtual box



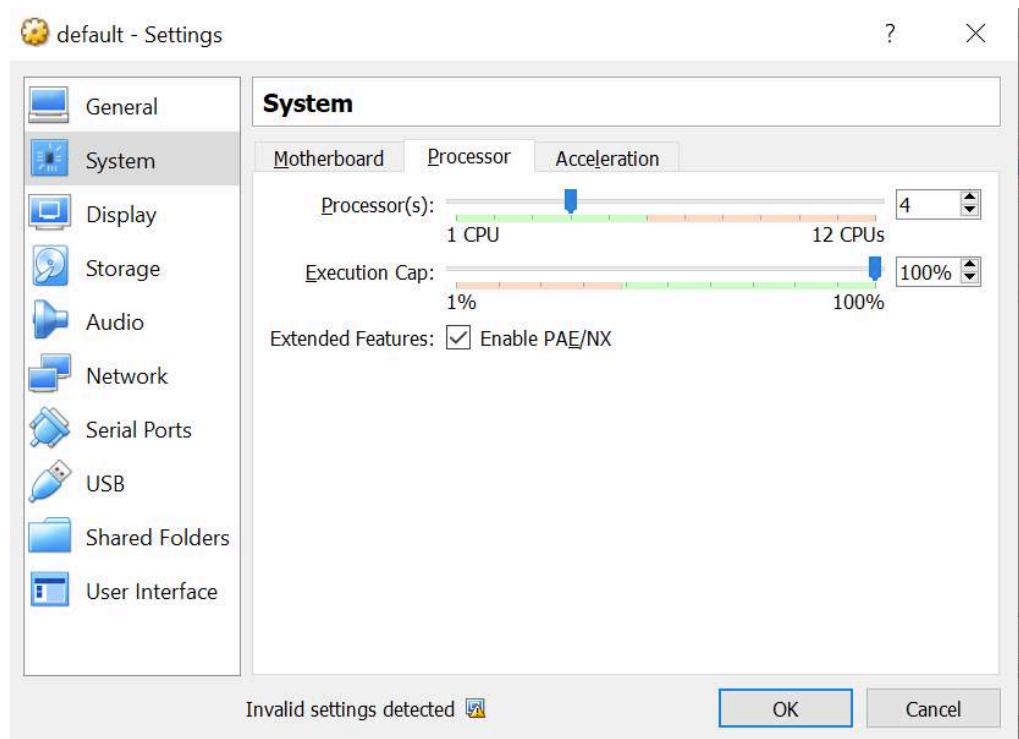
- b. Close and shut down your linux VM



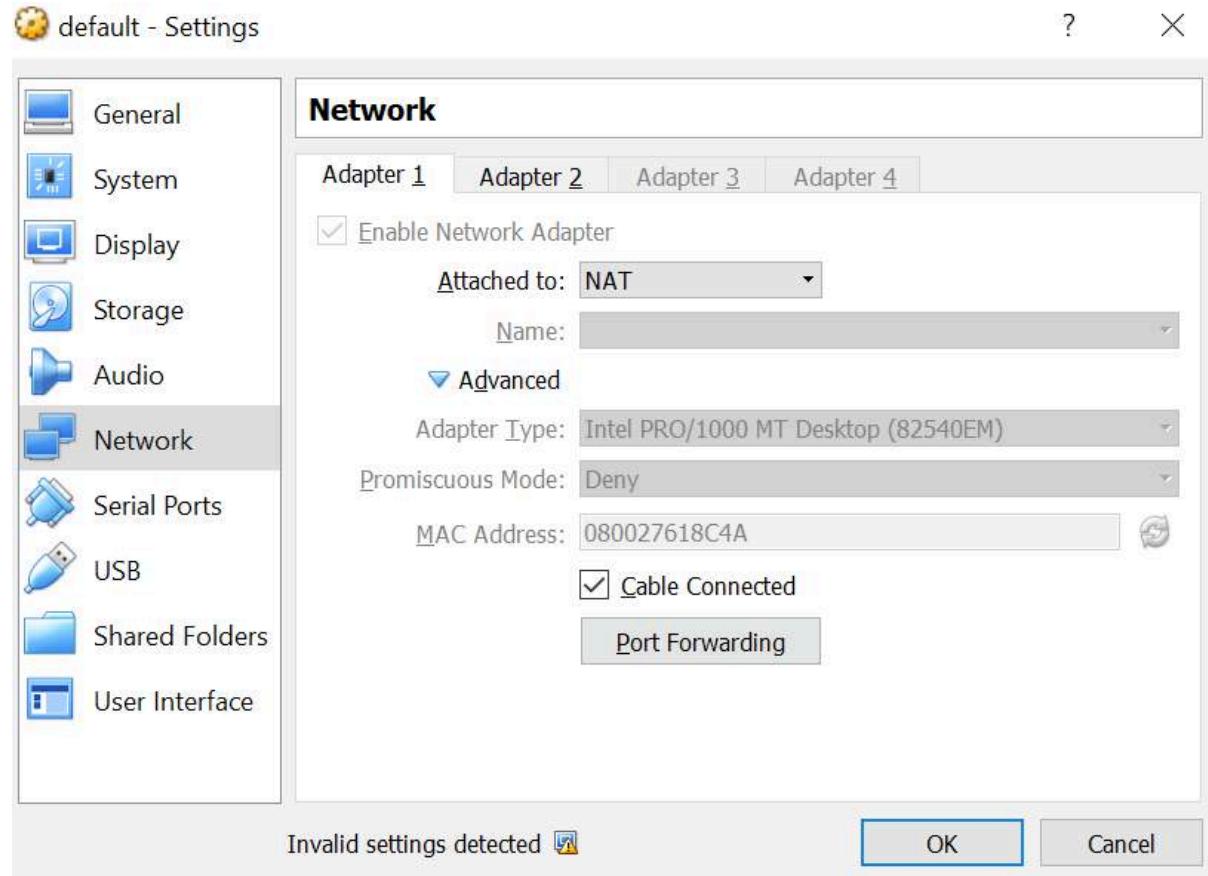
- c. Update memory to be at least 4 GB



d. Update the number of processors to be 4



e. Set “Port Forwarding rules” in your Network

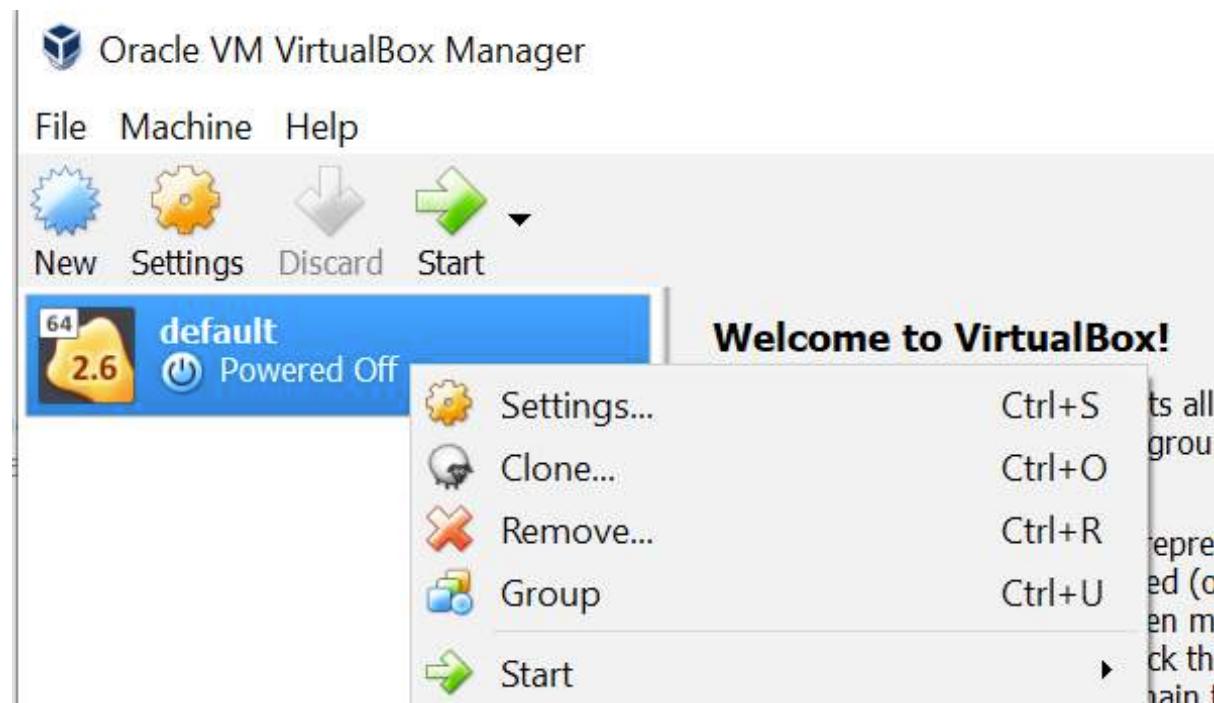


Port Forwarding Rules

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
elastic	TCP	127.0.0.1	9200		9200
kibana	TCP	127.0.0.1	5601		5601
neo4j	TCP	127.0.0.1	7474		7474
neo4j_bolt	TCP	127.0.0.1	7687		7687
springhttp	TCP	127.0.0.1	8080		8080
ssh	TCP	127.0.0.1	54414		22

OK    Cancel

Restart your VM (Headless start)



## Common Errors and Solutions

### 1. Problem

"Forbidden" while creating an index in Kibana

#### Solution

```
curl -XPUT -H "Content-Type: application/json" http://localhost:9200/_all/_settings -d
'{"index.blocks.read_only_allow_delete": null}'
```

### 2. Problem

Error 137 or Kibana not coming up

#### Solution

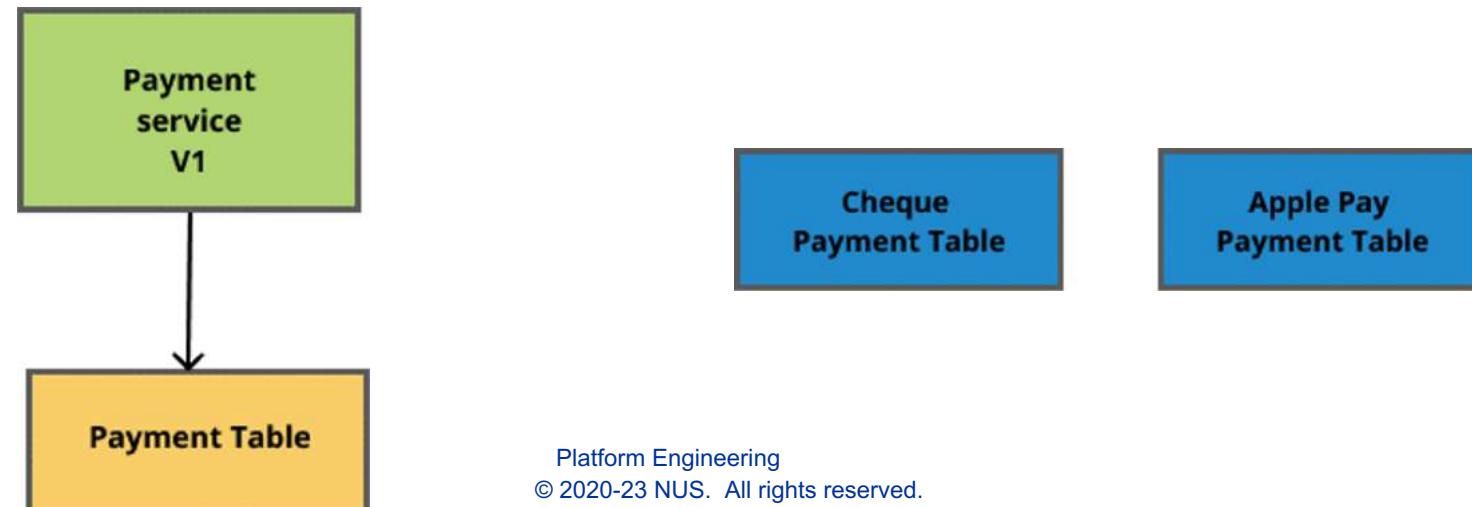
Check if you have increased the Docker memory to at least 4 GB and your CPU to be 4.



# DEPLOYMENT CASE STUDY

Total Slides: 8

- Your Payment microservice had been persisting its data to a Payment table, which stores information for all payment categories - Cheque, ApplePay etc.
- You made your analysis and figured out that you need to split the Payment table into two - one for Cheque and one for Apple Pay because the fields captured were different.
  - Version 1 has been writing to Payment table
  - Version 2 would be writing to ChequePayment and ApplePayPayment



- Old data from Payment table needs to be migrated to new tables
- Change is definitely not backward compatible
  - Retrieval of old payments would not be possible

Requires a fair amount of planning

- Make incremental upgrades 1.1, 1.2 ... and finally 2.0



# An approach

1. Create the two new tables - ChequePayment and ApplePayPayment
  - a. Strategy : Rolling Update - Only Database creation scripts will run
  - b. No code change
  - c. Rollbacks are fine - since only the two new tables would be dropped

Cheque  
Payment Table

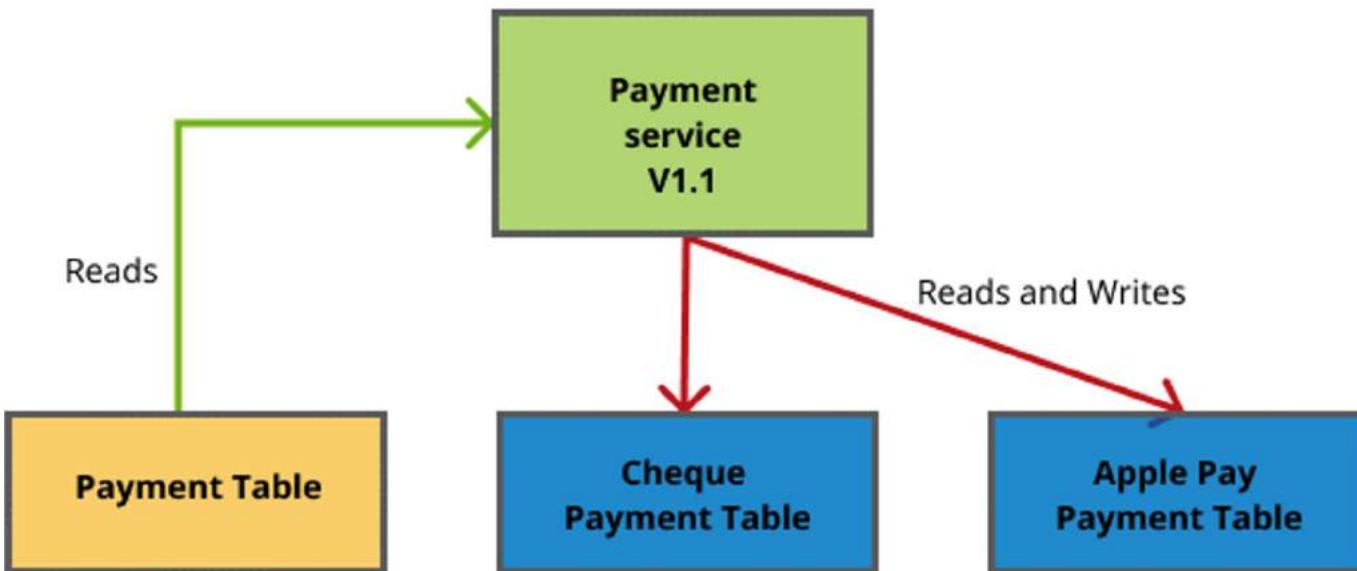
Apple Pay  
Payment Table



# An approach

## 1. Create Version 1.1 that

- a. Writes all new transactions to new tables.
- b. Reads missing transactions from old payments table
- c. Deletes are tombstoned - just marking the row as “deleted” but not physically deleting
- d. Strategy: Canary deployment of V1.1
  - Followed by rollout

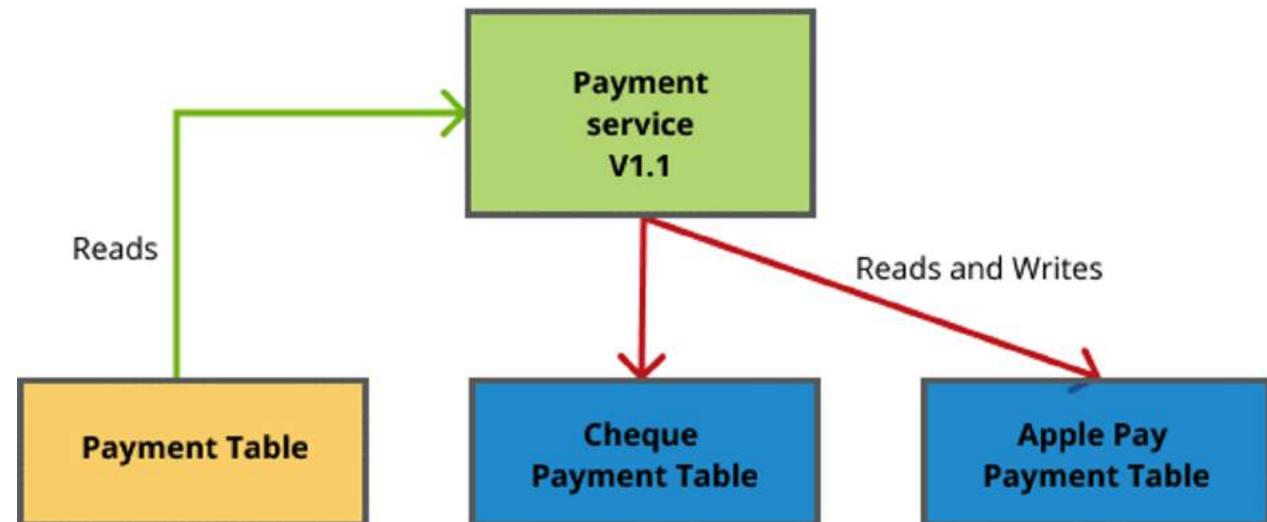




# An approach

Rollbacks are “okay” during canary deployments

- Original payments table untampered
- Surface area : 5-10% transactions





# An approach

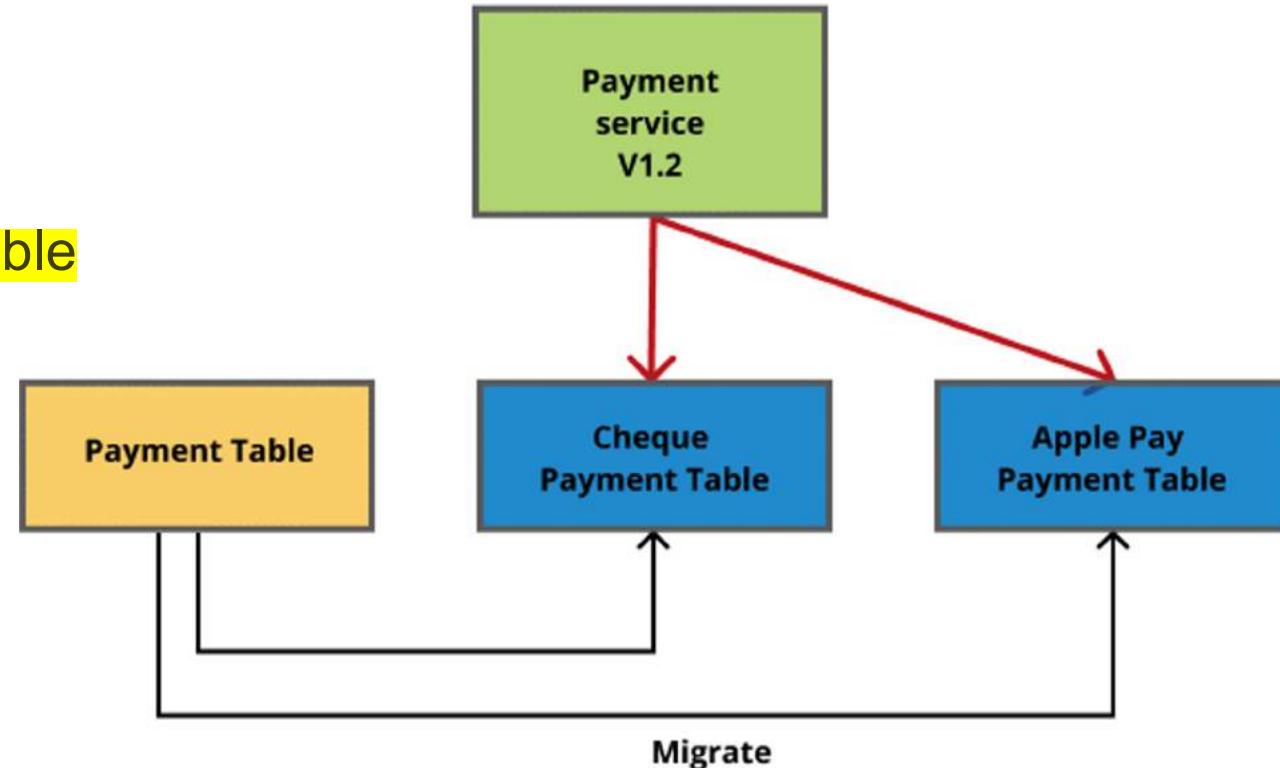
### 3. Migrate old data into new tables

- a. Version 1.1 would still function because only missed records are looked up from Payment table
- b. Rollbacks are fine if carefully planned
  - Drop the old transactions - based on updated date

#### 4. Create Version 1.2 that :

- a. Reads **only** from the new tables
- b. **Strategy : Canary release**
  - Followed by rollout
- c. Rollbacks are fine
  - **Goes back to reading from old table**

Tag 1.2 as 2.0 and rollout





# DATA MANAGEMENT

**Yunghans Irawan**

[yunghans@hotmail.com](mailto:yunghans@hotmail.com)

Total Slides: 65

- To appreciate various alternate data models for storing and retrieving data
- To understand the pros and cons of each data model



# Need for scale

- Your applications
  - Are highly available
  - Are fault-tolerant
  - Are performant
  - Can gracefully degrade
- How about your datastores?
  - Unprecedented amounts of data from variety of systems
  - IoT
  - Analytics-aware organizations

- Key data properties of a good platform
- Scaling databases
- CAP refresher
- **NoSQL**
  - Key-Value
  - Column family
  - Document
  - Search Engines
  - Graph
- **SQL for querying against files**
- Saga pattern - Distributed Transaction Management



# What do we need from datastores?

- Core : Ability to store and read data
- Others :
  - **Scalability (Vertical/Horizontal)**
  - **Consistency**
  - **Availability**
  - **Low latency (performant)**
  - **Evolvability**
  - Able to run analytics
  - Security
  - Able to jump tall buildings

*Scalability and Availability are most often related*



# Types of Data

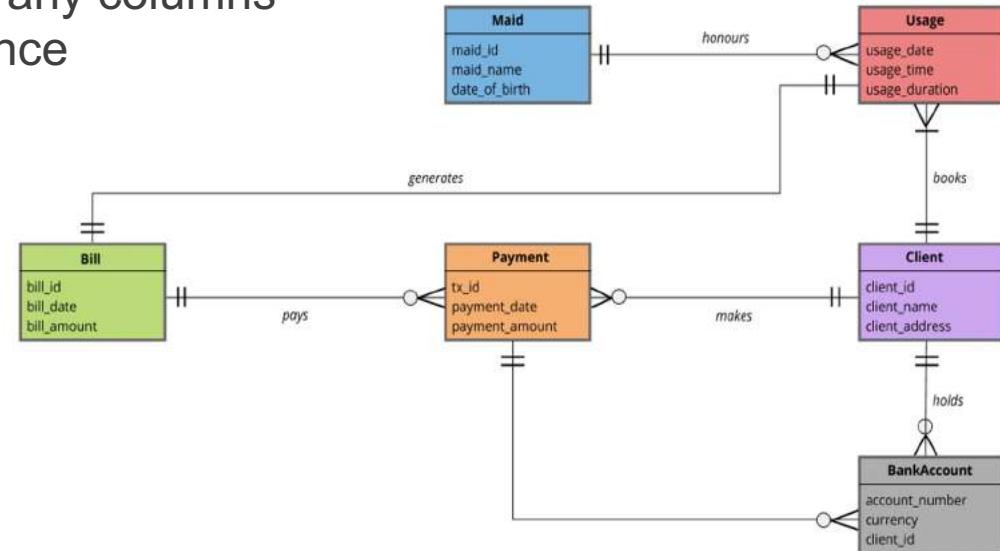
- Structured
  - Defined model - rows, columns
    - aka *Strict Schema*
  - RDBMS Data tables
- Semi-structured
  - Text but has pattern
    - aka *Schema free (Schemaless)*
    - Schema is generally attached to the data
  - JSON, XML, CSV
- *Unstructured*
  - *No inherent structure*
  - ***Plain text, PDF, Images, Binary***



# RDBMS is great for most scenarios

- Each Domain Entity can represent a table
- Friendly tabular structure
  - Structured data can be described in a Logical Data Model
- Structured Query Language (SQL)
- Flexible way of querying using any columns
- Good read and write performance

can do JOIN is an advantage





# Where does RDBMS stand?

- **Scalability (Vertical/Horizontal)** : Scaling RDBMS horizontally while maintaining consistency is hard. Sharding & Replication are the common choices
- **Consistency** : Strong ACID properties
- **Availability** : Replicate data must be available on several machines
- **Low latency** : Good performance for most applications. Read and Write performance required from modern applications are greater
- **Evolvability** :
  - Strict schema: How does the application handle schema evolution in RDBMS?
  - Translation between Domain entities and Tables are required (ORM frameworks)
- RDBMS is not a hammer
  - Graph-like connections eg. Social graph

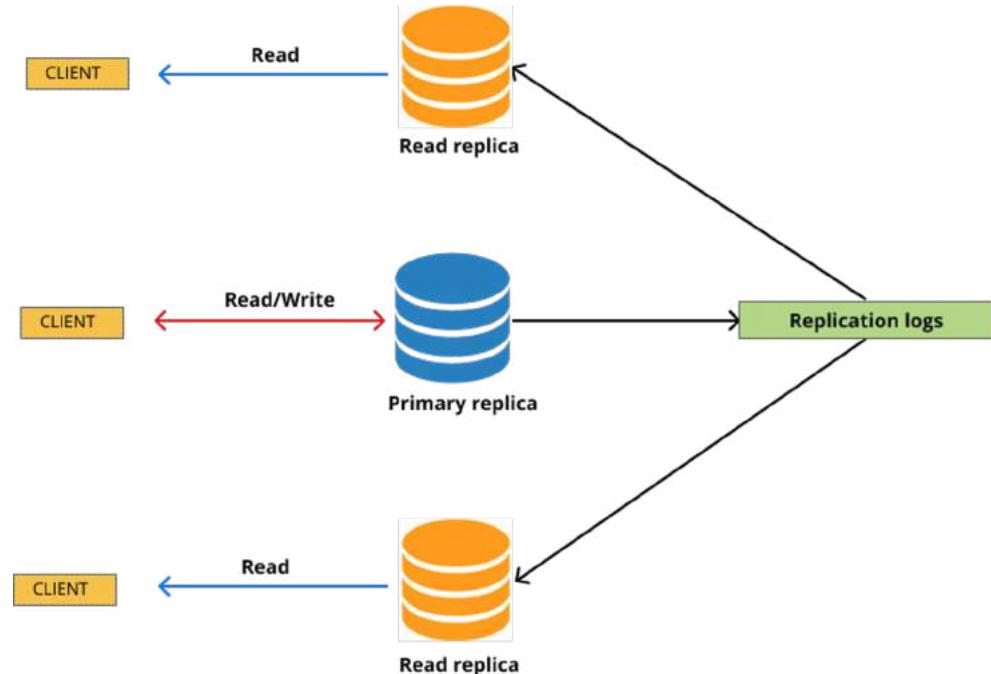


# Scaling databases



# Scaling reads - Read replication

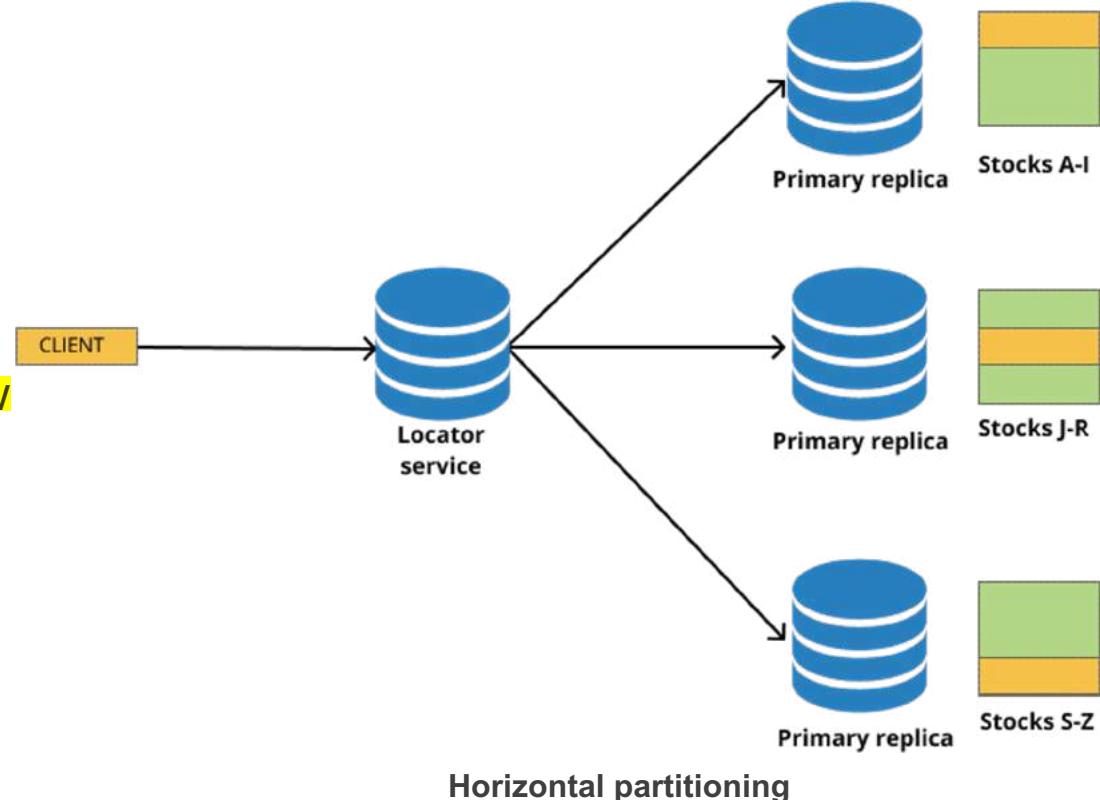
- Writes are accepted only by the primary replica - the master
- Read replica are just read-only copies of the data having full set of data
- For reads, clients can be routed to any of the replicas
- For writes, the request would be routed to the write replica
- If one of the replicas is down, the request is routed to other replicas
- There's a small replication lag





# Scaling writes - Sharding

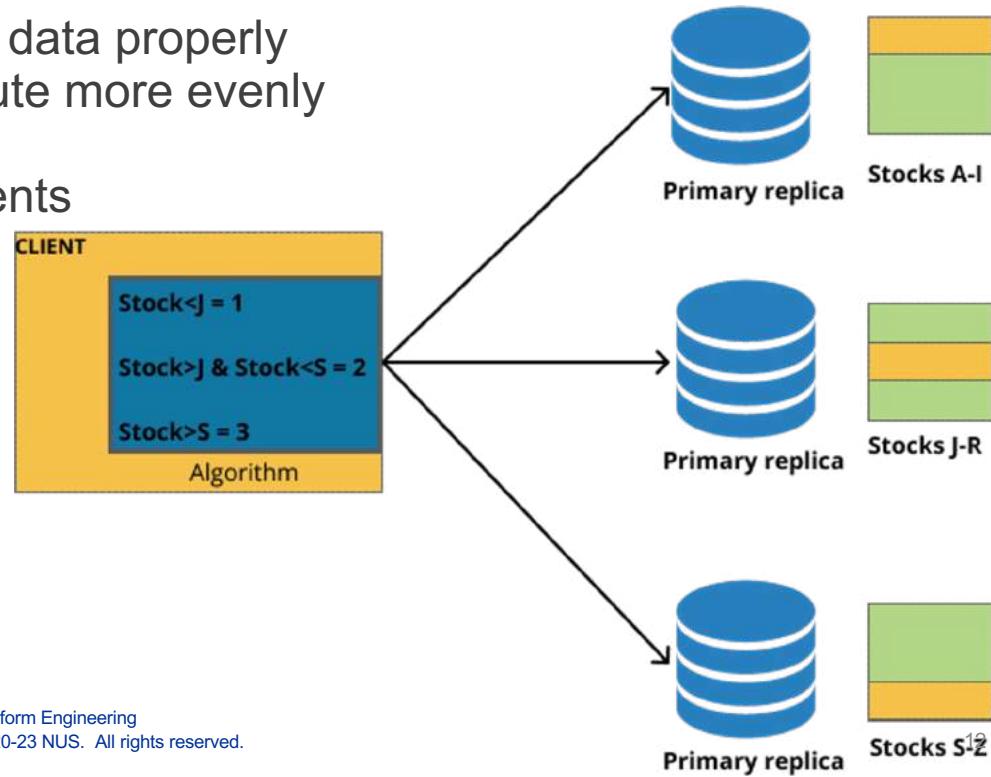
- A shard is a subset of data
- One master per shard
- Partitioning of data across several machines
- Improves both write and read performance
- Locator service to let client know which master is responsible for which shard





# Scaling writes - Algorithmic sharding

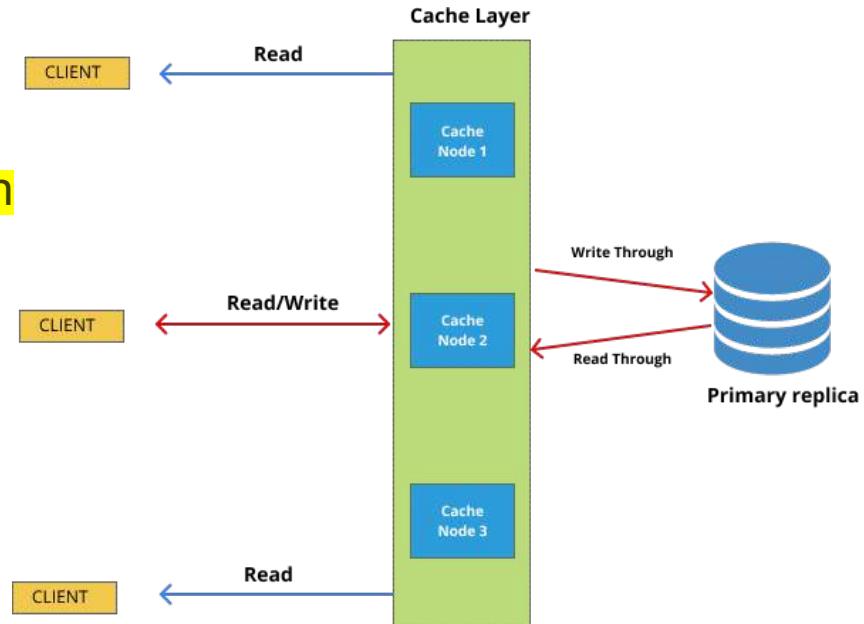
- No need of locator service
- Algorithm resides in Client
- Fixed sharding key may not distribute data properly
- Hash based partitioning would distribute more evenly
- Just like a regular HashMap
- If shard changes, masters publish events





# Scaling reads & writes - Caching

- Caching layer pretends to be the actual database
- In-memory writes are orders of magnitude faster than disk writes
- Read through for cache misses and cache expiry
- Write through for durability
- Cons : Loss of data if cache layer goes down before data gets flushed





# CAP Refresher

- **Consistency**
  - Once the data is committed, every reads the same value
  - There's a risk that some data become unavailable
- **Availability**
  - Some version of data is always available
  - There's a risk that the client reads old data
- **Partition Tolerance** - Resilient to network partitions
  - Single machine - The database doesn't horizontally scale
  - You have to pick partitioning

*If you are talking about huge volumes of data, you have already picked P*

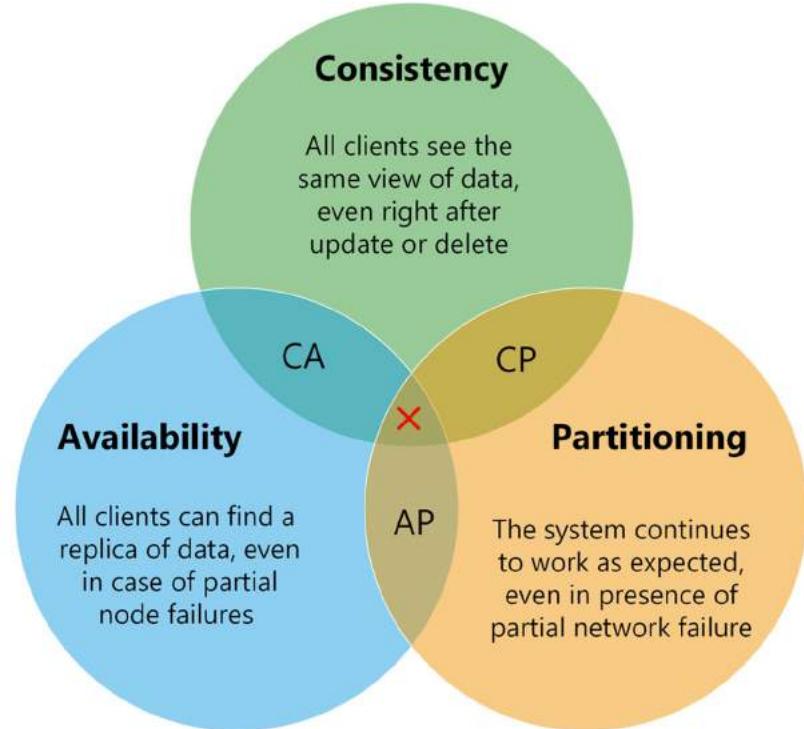
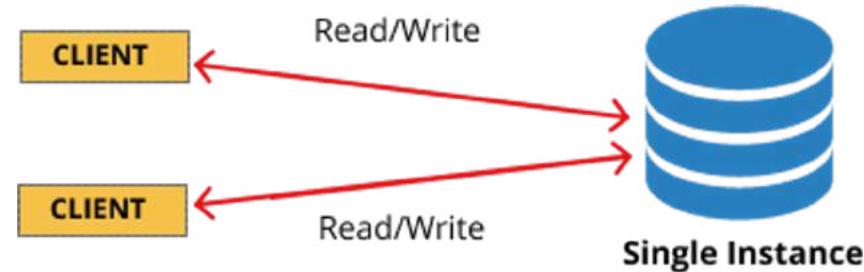


Image Source: [Research Gate](#)

- Consistent
- Available
- Partition Tolerant

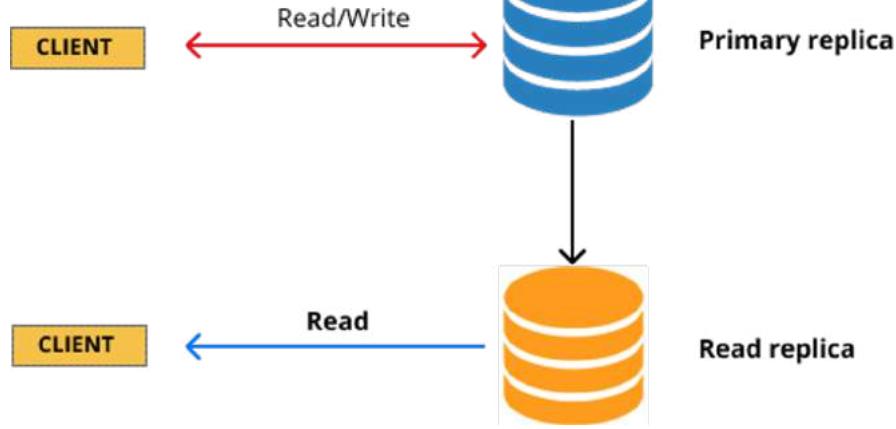


- Consistent
- Available
- Partition Tolerant



- When the master goes down, the replica takes the role of master
- All write requests from client will be **rejected** during this window
  - For sharded data, which is most often the case, this would mean no writes for data that's owned by that master
- The system or part of the system **is not available** during master failure

- Consistent
- Available
- Partition Tolerant



- Reads from replicas would be **stale**
  - Eventual consistency
- The system does **not guarantee consistent data reads**
- Multiple writes are also accepted during network partitions
  - **Last Write Wins**
  - Merging non-conflicting writes
  - Conflicts are pushed to the applications for resolution - DynamoDB



# Consistency is Overrated

- Amazon Shopping cart
  - What if the cart information of a user in a server in one geolocation does not replicate to another?
  - What's more important - availability of cart or accuracy of cart?
- Twitter's timelines
- Facebook feeds
  - Some feeds vs no feeds - Bad user experience
- Google's advertising indices
  - Some ads vs No ads - Loss of revenue

*Consistency vs Availability is a business decision than a technical one.*



# Summary

- You have to pick Partition Tolerance for distributed datastores.
- Simply put, between AP and CP systems, if the read happens on the master alone (no read replica), then the system is CP.



# Non RDBMS databases for Platforms

## NoSQL/NewSQL



# Why NoSQL for data management?

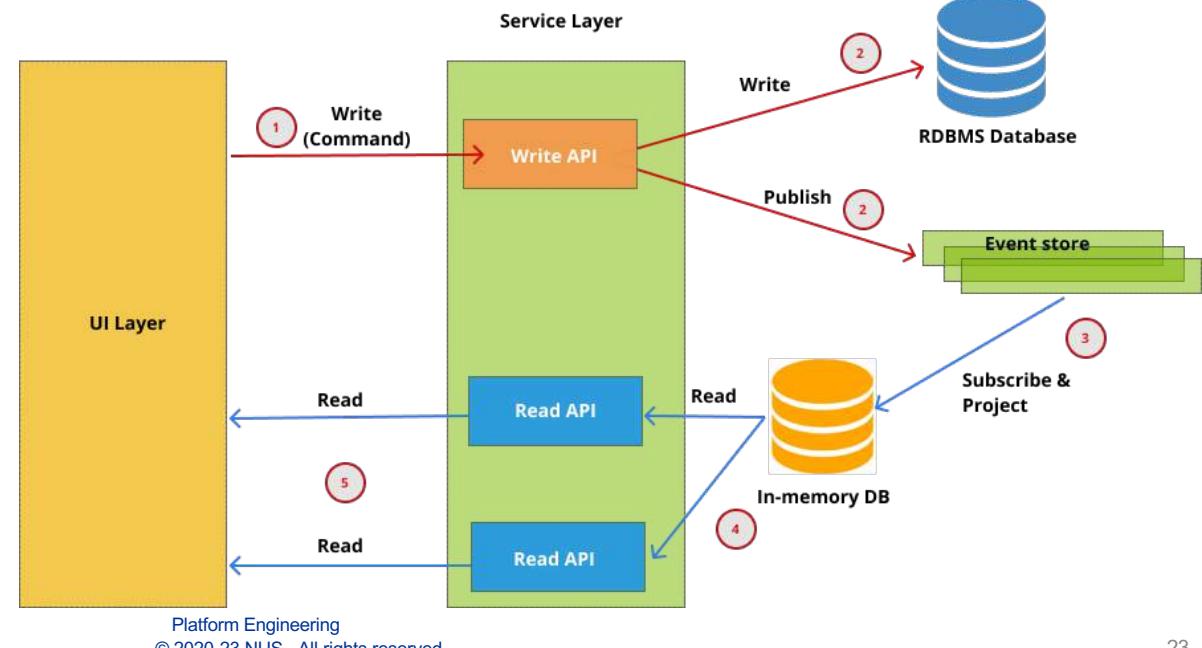
- Each platform and application requirements are unique –
  - write-heavy
  - read-heavy
  - analytics-based consumption
  - varied query patterns
  - Varied scalability and availability requirements
- Most platforms are fast moving and schema changes are the norm

# Most modern platforms use multiple databases



NUS  
National University  
of Singapore

- Even a single application in a platform use multiple databases
- Command Query Responsibility Segregation (CQRS)
- Advantage: Reads and writes can be scaled independently





# Where does NoSQL stand?

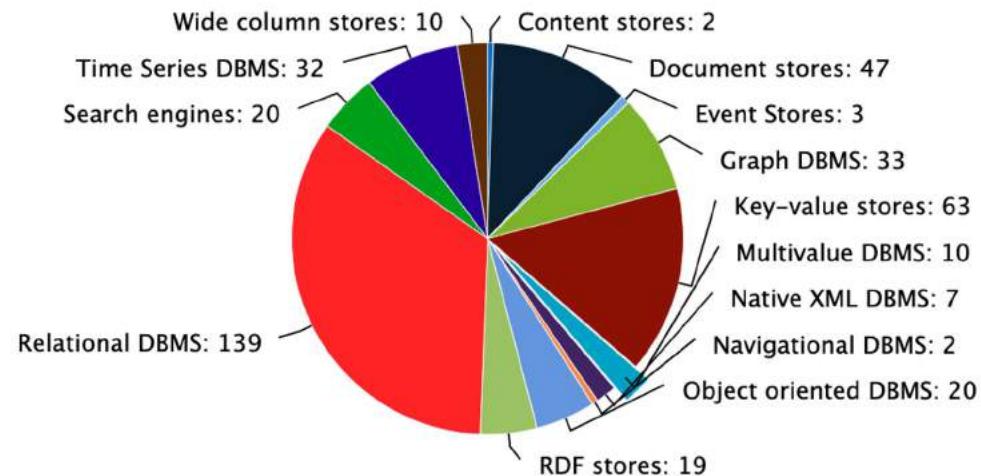
- **Scalability** : Most NoSQL databases **scale linearly** when horizontally scaled. Partitioning and Replication are baked into these systems.
- **Consistency** : ACID properties are fluid
- **Availability** : Most systems support read replicas for scalable reads
- **Low latency** :
  - Several DBs store data exclusively in memory or support in-memory caches.
  - **Optimized for read and write heavy requirements**
- **Evolvability** :
  - Almost all are schema-free or supports rich schema evolution



# NoSQL data models

- Key-Value
- Document
- Column family - Wide column
- Search engines
- Graph

**DBMS popularity broken down by database model**  
**Number of systems per category, February 2020**





# Key Value stores



# Key-Value stores

- **Problem** : Given a key, fetch the entire record as fast as possible (in-memory)
- **Popular applications:**
  - Amazon, Alibaba: Shopping carts
  - Box (like Dropbox): Recently uploaded files
  - Istio: Rate limits
  - Several projects: Cache, Session cache, application configuration
- Roughly a **Distributed HashMap**
  - Key-value pairs replicated across multiple servers
  - **Keys are unique**



# Key-Value stores



- **Where it shines**

- Blazing fast lookup by key (point-query)

- **Where it fades**

- Range queries are expensive by design
- Needs **large amounts of memory**
  - Most KV stores store everything in memory and flush to disk occasionally
  - Persistence affects performance



# Redis KV set and get on Hash - A quick look

- Redis has several rich in-memory datastructures. The example shown here is that of a Hash(table)

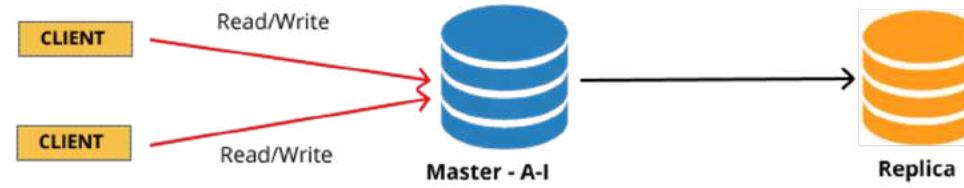
```
127.0.0.1:6379> HSET phones "apple" 11
(integer) 1
127.0.0.1:6379> HSET phones "samsung" 25
(integer) 1
127.0.0.1:6379> HSET phones "LG" 16
(integer) 1
```

```
127.0.0.1:6379> HGET phones "LG"
"16"
```



# A look at a KV System Architecture - Redis

- Redis is a CP system
  - Writes and Reads go through a single shard master





# Document stores



# Document stores

- **Problem** : Your application needs **richer querying capabilities** on all columns without the restrictions of strict schema
- **Popular applications** :
  - eBay: Search suggestions, Product catalog
  - MetLife: Client 360° view of transactions & policy details
  - Shutterfly: Image store
  - Several organizations: Replacement for RDBMS



# Document stores

- Stores data as a “Document” (**mostly some form of JSON**)
- Child entities are nested (aka embedded) to parent entity
- Schema free - smoother schema evolution

RDBMS	DocumentDB
Table	Collection
Row	Document
Index	Index
Join	Child documents/Embedded documents



# RDBMS vs JSON Document

Employee	
empId	5e5a18de48af7755c31938f7
age	28
eyeColor	blue
name	Dolores Dickson
gender	female
company	ANIVET
email	doloresdickson@anivet.com
phone	+1 (967) 459-2598
favoriteFruit	banana
addressId	5e5a18de16e47f037e5014f2



Address	
addId	5e5a18de16e47f037e5014f2
door	784
street	Macdougal Street
city	Greer
state	Tennessee
pin	4400

```
{  
  "empId": "5e5a18de48af7755c31938f7",  
  "age": 28,  
  "eyeColor": "blue",  
  "name": "Dolores Dickson",  
  "gender": "female",  
  "company": "ANIVET",  
  "email": "doloresdickson@anivet.com",  
  "phone": "+1 (967) 459-2598",  
  "address": {  
    "addId": "5e5a18de16e47f037e5014f2",  
    "door": 784,  
    "street": "Macdougal Street",  
    "city": "Greer",  
    "state": "Tennessee",  
    "pin": 4400  
  },  
  "favoriteFruit": "banana"  
}
```



# Document stores

## Where it shines :

- Richer querying capabilities
- Id based queries are natural and blazing fast
- B+Tree & other index types supported

## Where it fades:

- No Joins
  - Some DBs (like MongoDB) support but most don't.
- No complex transactions
- No SQL, Constraints or Trigger support



NUSIS | [MongoDB](#) +

[Amazon DynamoDB](#) +

[Couchbase](#) +

[Microsoft Azure Cosmos DB](#) +

CouchDB

Firebase Realtime Database

[MarkLogic](#) +

[Realm](#) +

[Google Cloud Firestore](#)

OrientDB

[Google Cloud Datastore](#)

[ArangoDB](#) +

RavenDB

Cloudant

RethinkDB

PouchDB

Apache Drill

CloudKit

Mnesia

Datameer

[InterSystems IRIS](#) +

LiteDB

[FaunaDB](#) +

[FoundationDB](#)

[Amazon DocumentDB](#)

[AllegroGraph](#) +

XAP

[BigchainDB](#)

[MapR-DB](#)



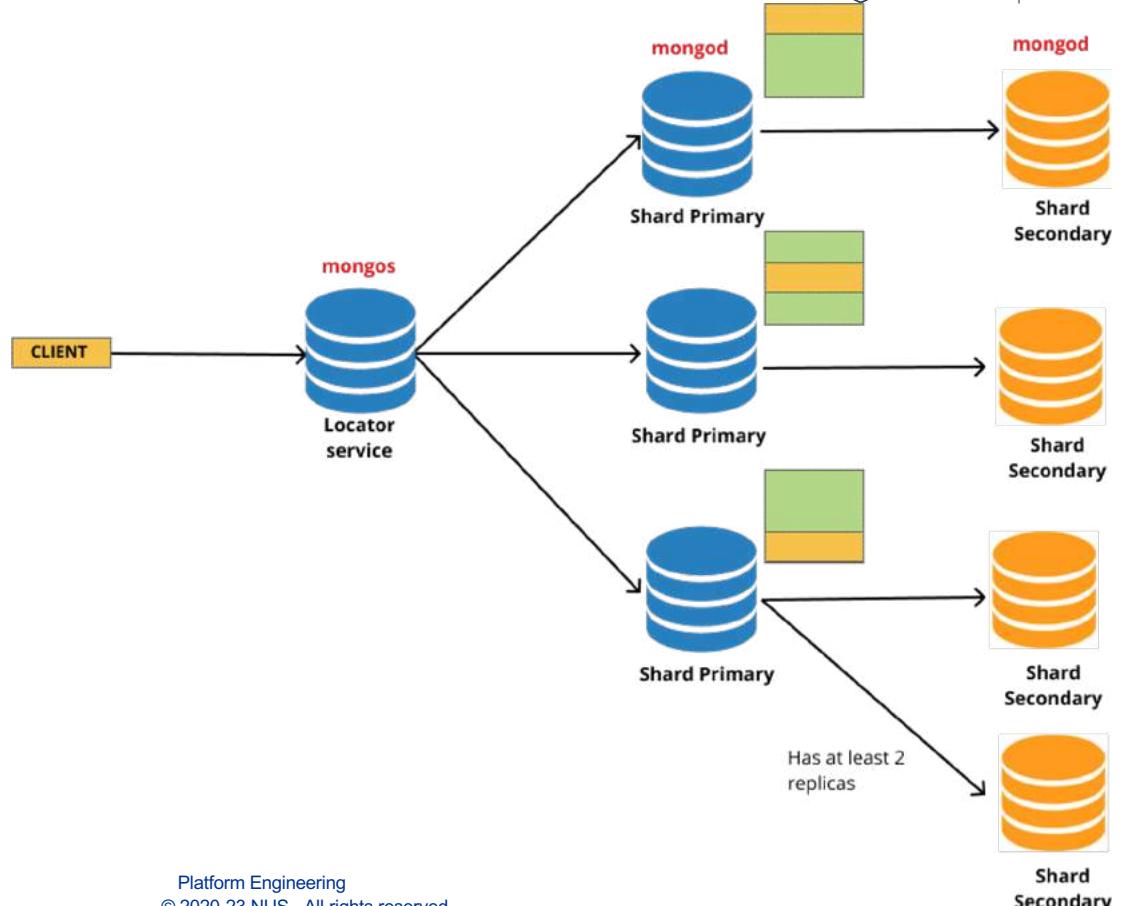
# MongoDB search - A quick look

- MongoDB CRUD operations are in the form of JSON

```
> db.users.find({"eyeColor": "blue"})
{ "_id" : "19", "empId" : "5e5a7628c5d9f93798b8f9df", "age" : 24, "company" : "MIRACLIS", "email" : "jansherman@miraclis.com", "phone" : "5ef79b9f9", "door" : 631, "street" : "Vandam Street", "city" : "Eagle apple" }
```

# A look at a DocumentDB Architecture - MongoDB

- MongoDB is a **CP** system
  - Writes and Reads go through **a single shard master**





# Column family stores



# Column family/Wide column stores

- **Problem :**

- Your application is **write-heavy** - 10s of thousands of records per second.
- Extremely flexible schema and **values are sparse**
- Given a key, fetch the entire record as fast as possible

- **Popular applications**

- Google and Yahoo : Web crawl store
- Facebook: Cassandra and then HBase for Inbox Search, HBase for messenger
- Mozilla: All browser crash data is stored in HBase
- Twitter: **All their Tweets and other production data in RDBMS tables are backed up on HBase for analytics**

Cassandra +  
HBase  
Microsoft Azure Cosmos DB +  
Datastax Enterprise +  
Microsoft Azure Table Storage  
Accumulo  
Google Cloud Bigtable  
ScyllaDB  
MapR-DB  
Alibaba Cloud Table Store



# Wide column/Column family stores

rowKey	emplId	personal			otherInfo		
		age	name	gender	favoriteFruit	eyeColor	company
5e5a18de48af7755c31938f7Dolores	5e5a18de48af7755c31938f7	28	Dolores Dickson	female	banana	blue	ANIVET

Row key	rowKey: 5e5a18de48af7755c31938f7Dolores	rowKey 5e5a18de48af7755c31938f7Dolores
Column	emplId: 5e5a18de48af7755c31938f7	emplId 5e5a18de48af7755c31938f7
Column family	column_family.column_qualifier personal.age: 28 personal.name: Dolores Dickson	personal age name gender 28 Dolores Dickson female



# Where Column family shines

- **Schema free**
  - Database is a collection of key/value pairs. A “huge” distributed nested map !
  - Key consists of 3 parts – row key, column key and a time-stamp (version)
  - Flexible schema - the set of columns is not fixed, and may differ from row-to-row
  - Column key consists of two parts – a column family, and a qualifier (column name)
- **Linearly and massively scalable**
- A few has SQL-like support
  - CQL in Cassandra
  - Phoenix in HBase

Cassandra +  
HBase  
Microsoft Azure Cosmos DB +  
Datastax Enterprise +  
Microsoft Azure Table Storage  
Accumulo  
Google Cloud Bigtable  
ScyllaDB  
MapR-DB  
Alibaba Cloud Table Store



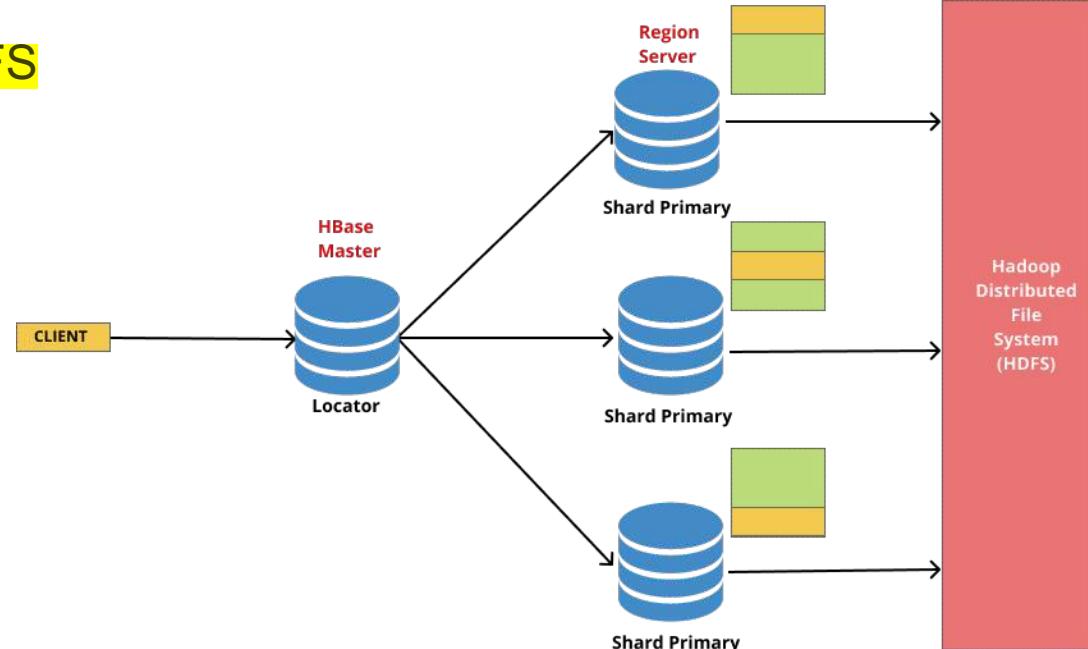
# Where it fades

- **No Joins**
  - All data must reside in the same column family
- Secondary indices are significantly slower and are expensive in terms of cost
- Is **not a general purpose database**



# A look at a CF store Architecture - HBase

- Each Shard/Region is owned by a single server
- Replication is taken care by HDFS (Typically 3 replicas)
- HBase is a CP system





# Search Engines



# Search Engines

- **Problem :**

- Your application users need **advanced query capabilities**
  - Ranking
  - Point
  - Range
  - **Wildcard**
  - Filtering
  - Aggregations

- **Popular applications**

- Stackoverflow: Search results
- **Github: Code search**
- Netflix: Customer service operations and security logs
- LinkedIn & Medium: Debug production issues - Logging and monitoring



Elasticsearch +

Splunk

Solr

MarkLogic +

Sphinx

Microsoft Azure Search

ArangoDB +

Algolia

Amazon CloudSearch

Google Search Appliance

Virtuoso +

Xapian

CrateDB +

SearchBlox

searchxml

Indica

Manticore Search

DBSight

FinchDB

Exorbyte



# Elastic search - A quick look

- ElasticSearch queries are in **JSON format**
- There's a variety of query operators available

```
GET /users/_search
{
  "query": {
    "match": {
      "gender": "male"
    }
  }
}
```

```
"hits" : [
  "total" : {
    "value" : 5,
    "relation" : "eq"
  },
  "max_score" : 0.7801585,
  "hits" : [
    {
      "_index" : "users",
      "_type" : "_doc",
      "_id" : "1",
      "_score" : 0.7801585,
      "_source" : {
        "empId" : "5e5a766afa73dddb81a1c939",
        "age" : 21,
        "eyeColor" : "brown",
        "name" : "Allison Strong",
        "gender" : "male",
        "company" : "ENERFORCE",
        "email" : "allisonstrong@enerforce.com",
        "phone" : "+1 863 502 2602"
      }
    }
  ]
}
```



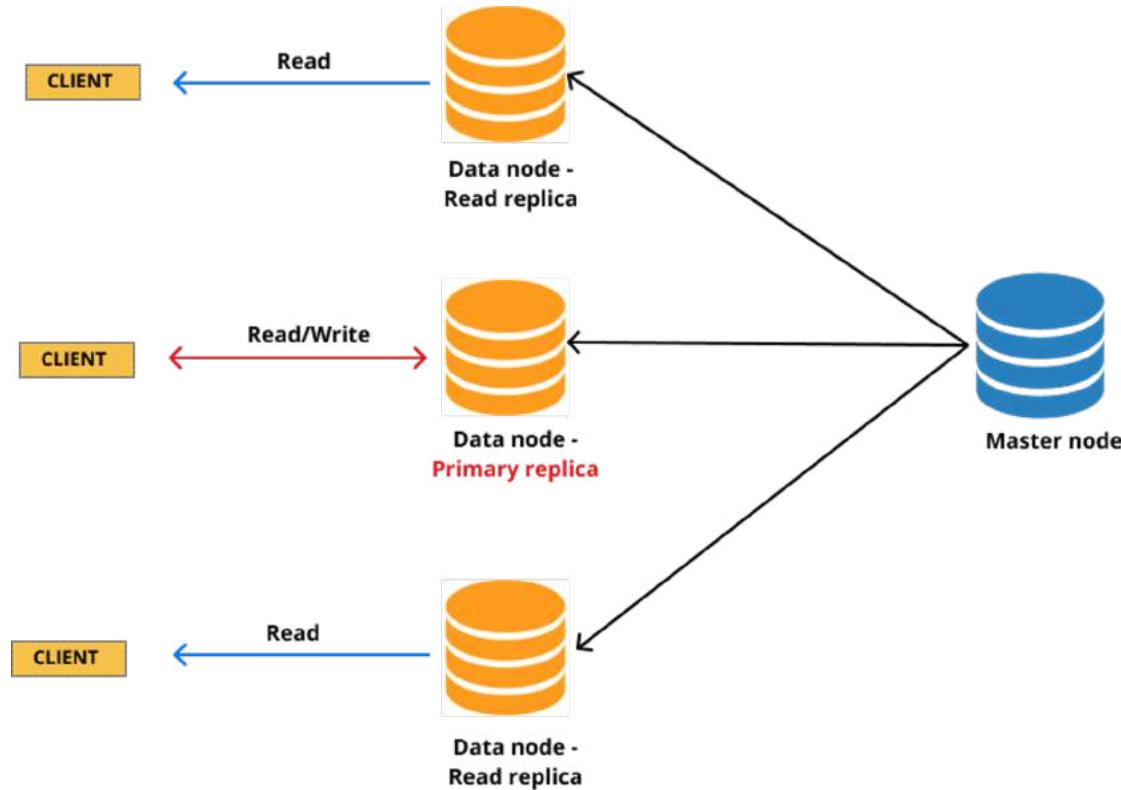
# Where it fades

- **No Joins**
  - Fall back to embedded documents
- **Insertions are slow - tokenization and analysis**
  - Not recommended as a primary datastore
- No transactions



# Elastic Architecture

- Data nodes store data
- Master node is responsible only for **light-weight** cluster wide actions -
  - **creating/deleting indices**
  - tracking data node availability
  - deciding which shards go into which nodes
- ElasticSearch is an **AP system**





# Graph stores



# Where Graph DBs shine

- Graph problems are obvious
  - You care about **relationships between “things”**
    - Person (social graph) and organizations (dependencies)
    - Connected clusters and similarities
    - Key member/organization of a cluster
- **Popular applications**
  - Adidas: Realtime content recommendations
  - eBay: Logistics and Routing
  - Investment Banks – **to query and analyze the inter-relationship between clients and funds**
  - Venture capitalists - to analyze which clusters of investors invest together



NUS Neo4j

Microsoft Azure Cosmos DB

OrientDB

ArangoDB

Virtuoso

Amazon Neptune

JanusGraph

Dgraph

GraphDB

FaunaDB

Giraph

Stardog

TigerGraph

AllegroGraph

Blazegraph

Graph Engine

InfiniteGraph

FlockDB

Grakn

HyperGraphDB

Nebula Graph

TinkerGraph

AnzoGraph

AgensGraph

GraphBase

Sparksee

HGraphDB

VelocityDB

Weaviate

HugeGraph

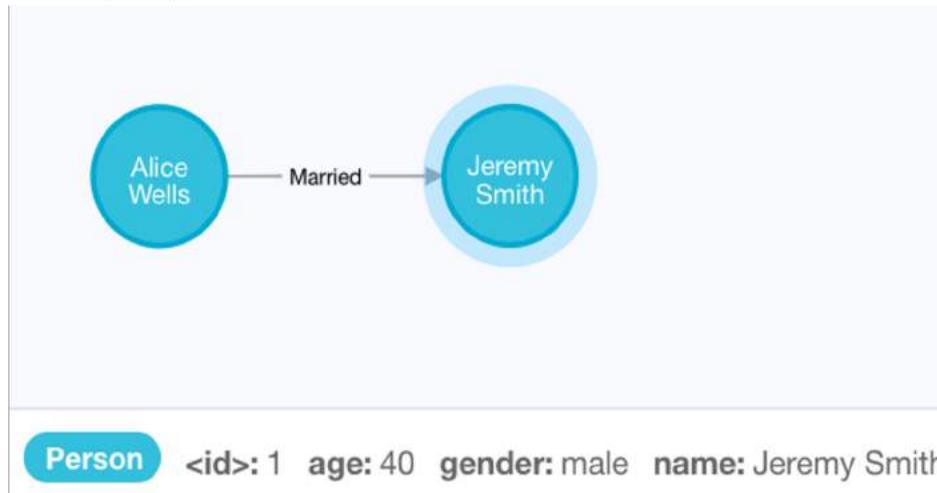


# Graph stores

- Graphs are a natural way to represent certain problems

Person		
Name	Gender	Age
Alice Wells	female	36
Jeremy Smith	male	40

Relationship		
person_id_1	person_id_2	relationship
1	2	Married





# Neo4J search - A quick look



```
MATCH (a:Person) WHERE a.name="Alice Wells" RETURN a
```

The screenshot shows the Neo4J browser interface. At the top, there is a search bar with the query: `MATCH (a:Person) WHERE a.name="Alice Wells" RETURN a`. Below the search results, there is a single node highlighted with a blue circular selection. The node contains the text "Alice Wells". At the bottom of the screen, there is a summary of the node's properties:

Person	<id>: 0	age: 36	gender: female	name: Alice Wells
--------	---------	---------	----------------	-------------------



# Where it fades

- Aggregations require traversing the entire database
  - Nature of Graphs
- Horizontal scaling isn't easy
  - How do we split graphs?
  - Graphs can span across multiple machines
    - Who can do coordination
  - Some Graph databases do this but it comes at a cost

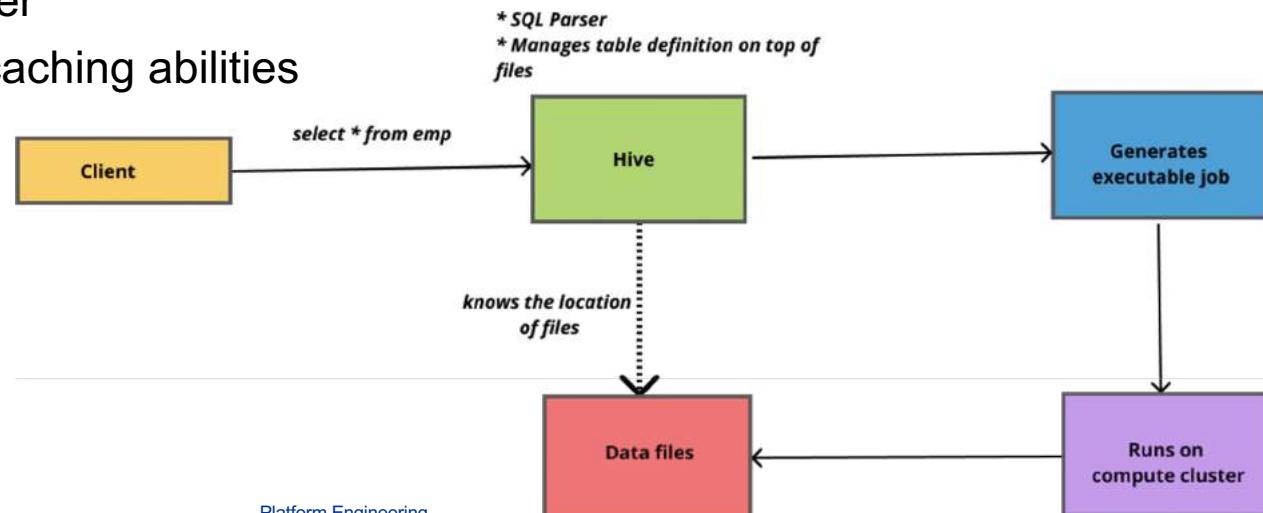


# SQL for accessing files



# SQL for accessing files

- Hive, Athena, Big Query, Snowflake
- Not a database at all. They are a combination of :
  - SQL Parser
  - Job generator
  - Metadata manager
- Some systems have caching abilities





# References

- Designing Data Intensive Applications - Martin Kleppmann
- You can't sacrifice Partition Tolerance - Codahale
- Redis documentation
- Programming Models for Distributed Computing
- Designing highly scalable database architectures
- Please stop calling databases CP or AP

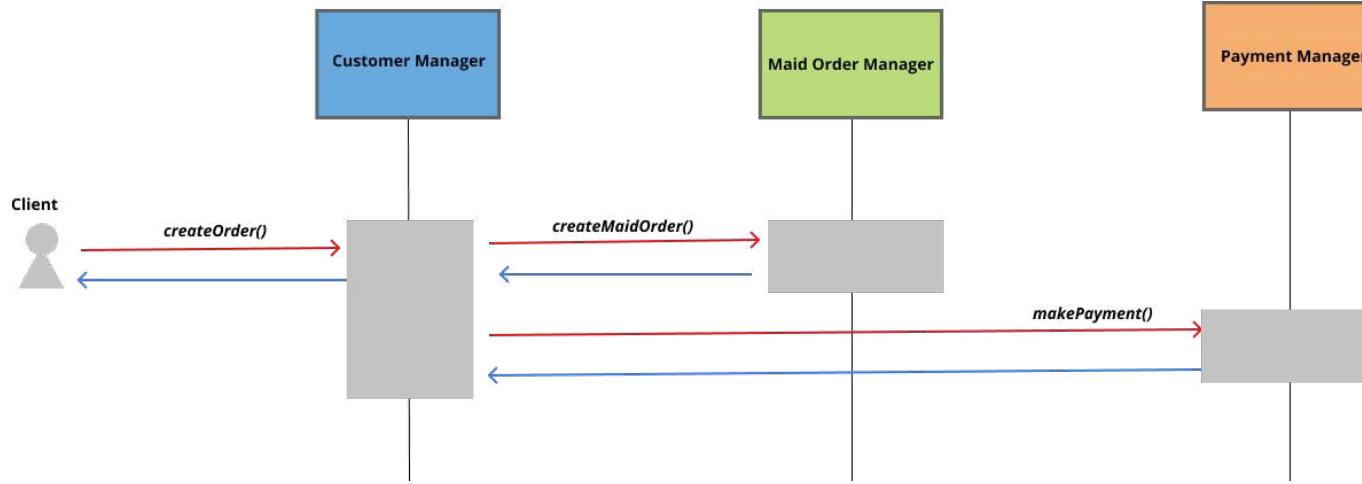


# Distributed transaction management

**Sagas**



# What is a distributed transaction



- Most business transaction spans across multiple services
- Need to maintain consistency
- Traditional way is to do 2 Phase commit – Prepare phase and commit phase



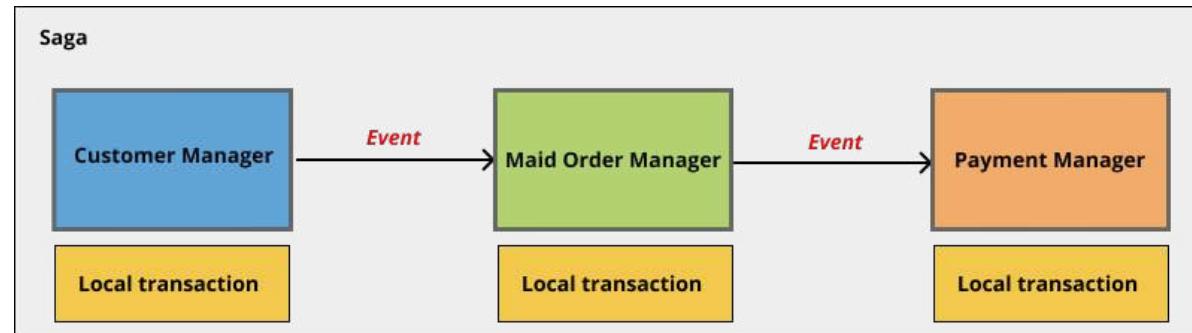
# Problems with distributed transactions

- Not all databases support ACID transactions
- Platform typically uses multiple databases
- 2 Phase commit requires synchronous communication
  - Asynchronous communication is the norm with modern applications
- Can we risk not doing transactions?
  - Partial transaction executions are dangerous



# Saga pattern

- Inspired by the 1987 paper titled “Sagas”
- A Saga typically represents a business transaction



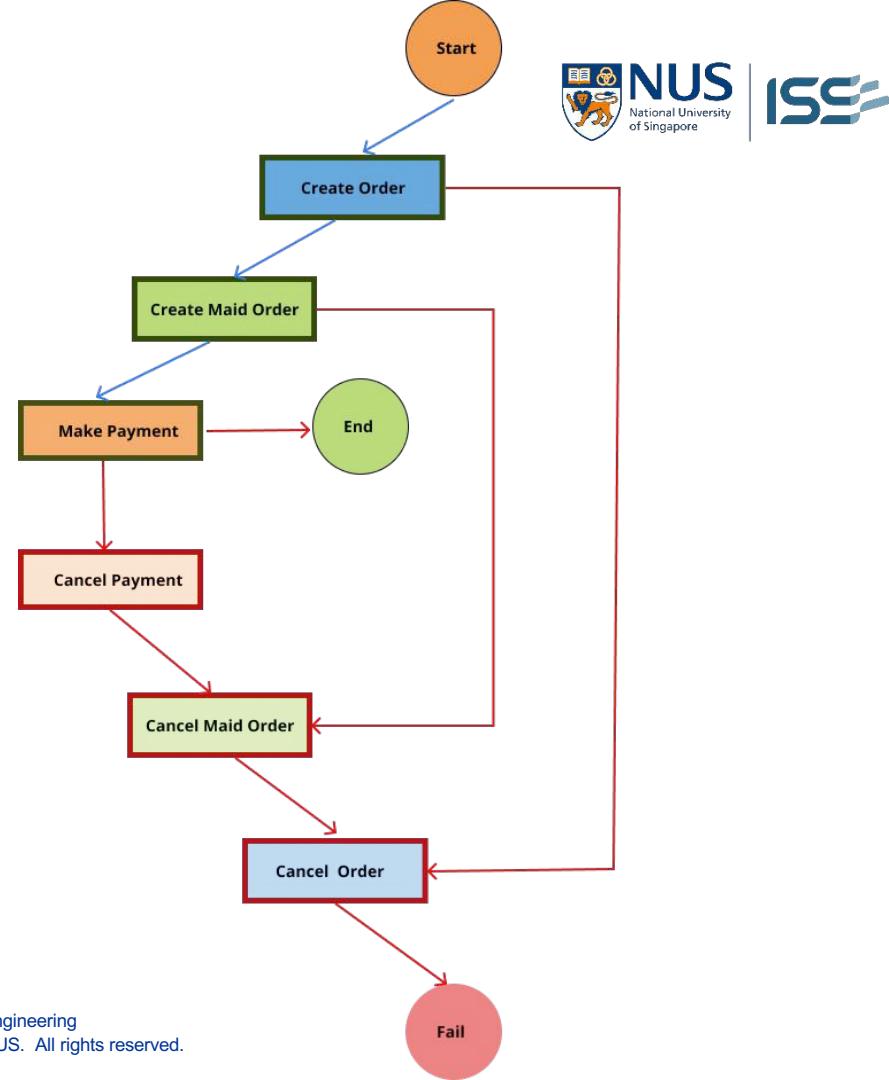


# Sagas are state machines

- Idea: Every transaction has a compensating transaction –  $T_i, C_i$
- All calls are asynchronous – better user experience

## Note :

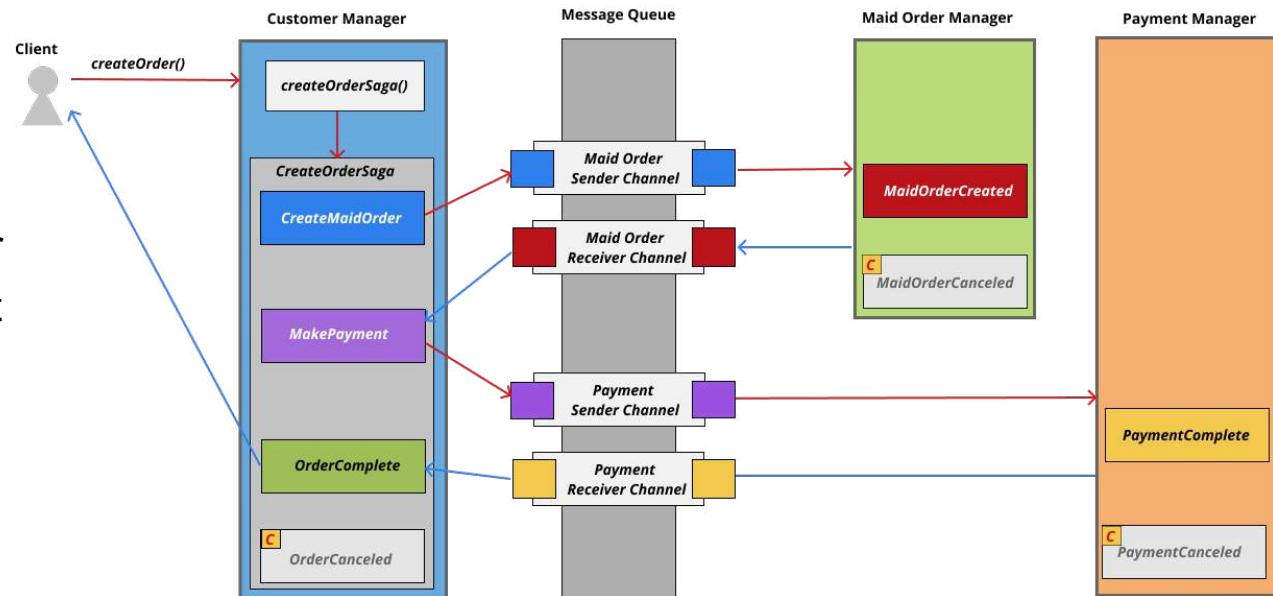
- CreateOrder & CancelOrder belongs to CustomerManager service
- CreateMaidOrder & CancelMaidOrder belongs to MaidOrderManager service
- MakePayment & CancelPayment belongs to PaymentManager service





# Orchestrator based saga

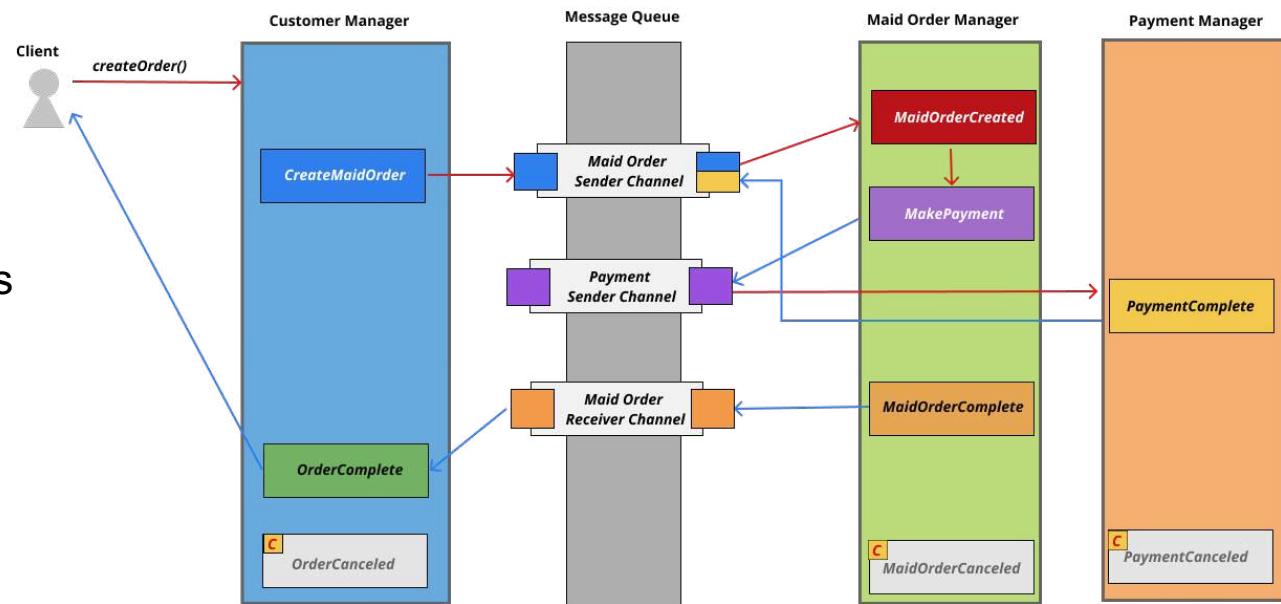
- Orchestrator has all the intelligence for executing the business transaction.
- It creates transaction events and compensating events for each service and publishes it to the respective channel
- **Cons:** Single point of failure/contention





# Choreography based saga

- Each service has some intelligence about how the events must be handled
- **Cons :** Democratized logic is hard to debug





# General expectations with Saga pattern

- **Distributed tracing** with a shared traceld is very important for Sagas
- **No read isolation** – users can read data for incomplete transactions or in-progress canceled transactions



# References

- Microservices patterns – Chris Richardson
- <https://microservices.io/>

# DATA MANAGEMENT WORKSHOP

Platform Engineering

Platform Engineering

## Table of Contents

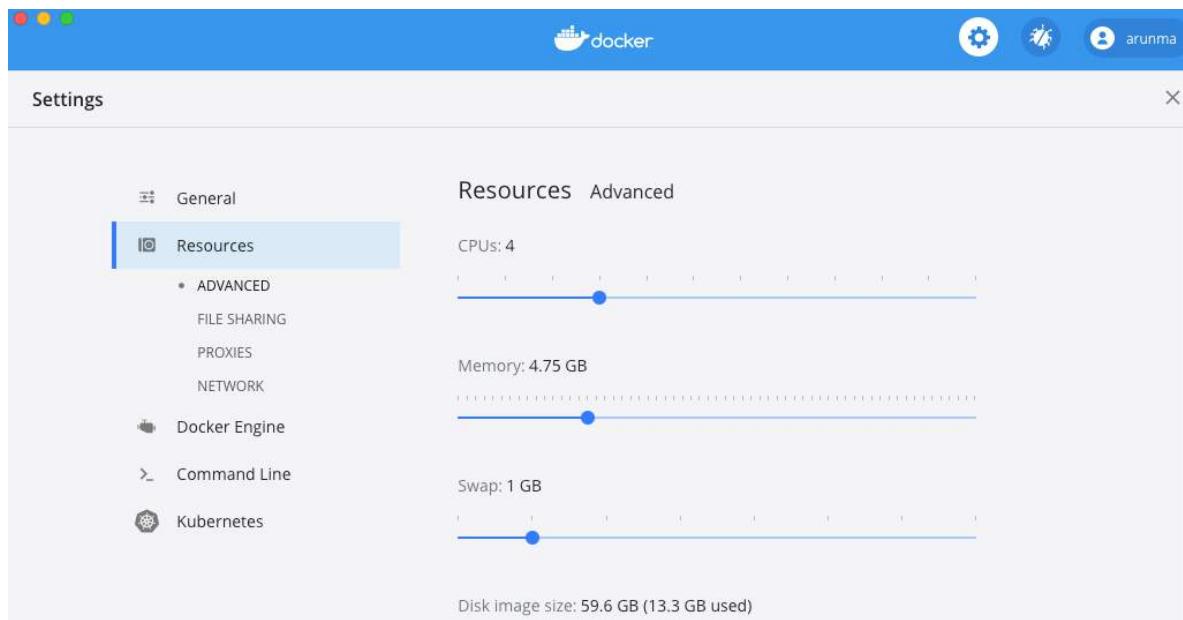
<b>Clone your workshop code from .....</b>	<b>3</b>
<b>Adjust your docker settings to give at least 4 cores and 3 GB memory. ....</b>	<b>3</b>
<b>Launch all databases on your local machine using docker-compose .....</b>	<b>3</b>
<b>ElasticSearch.....</b>	<b>3</b>
Open up Kibana .....	3
Open ElasticSearch Dev Tools .....	4
Create index (Table/Collection).....	4
Read all documents .....	4
Create document with custom id .....	5
Get document by id .....	6
Update documents.....	6
Bulk upload.....	7
Search all documents .....	9
Search by query.....	10
Delete a Document.....	11
<b>Redis.....</b>	<b>12</b>
Lists.....	12
Push to a List .....	12
Push multiple values to a List .....	12
Get values from a List.....	13
Removing a value from a List .....	13
Removing a specific value from a List .....	13
Find the length of the List .....	13
Sets .....	13
Push to a Set.....	13
Push multiple values to a Set .....	13
Listing members in a Set .....	14
Removing a value from a Set.....	14
Checking membership in a Set .....	14
Hashes .....	14
Push to a Hash.....	14
Pushing multiple values to a Hash.....	15
Get value for a key from a Hash .....	15
Get all values from a Hash.....	15
Removing a value from a Hash.....	16
<b>Neo4J.....</b>	<b>17</b>
Open up Neo4J browser .....	17
Create new Nodes .....	17
Create new Relationship .....	17
Query a single node.....	17
Get all nodes and relationships .....	18
Deleting a node .....	19
Delete Relationship .....	19
Delete Nodes.....	19
Load Graph from CSV .....	20
<b>MongoDB .....</b>	<b>22</b>
Login to your mongodb docker container using the following command .....	22
Create/Switch to a database .....	23
Create a new collection .....	23
Insert a new document .....	23

Bulk insert documents.....	23
Query all documents in a collection .....	25
Find document in a collection matching a criteria .....	25
Find document in a collection matching multiple criteria.....	25
Removing a document .....	26
Updating a document.....	26

## Clone your workshop code from

<https://github.com/arunma/datamanagement>

**Adjust your docker settings to give at least 4 cores and 3 GB memory.**



**Launch all databases on your local machine using docker-compose**

```
cd <location of datamanagement>
# eg. cd /Users/arunma/projects/data-management
```

```
docker-compose -f docker-compose.yml up --build
```

## ElasticSearch

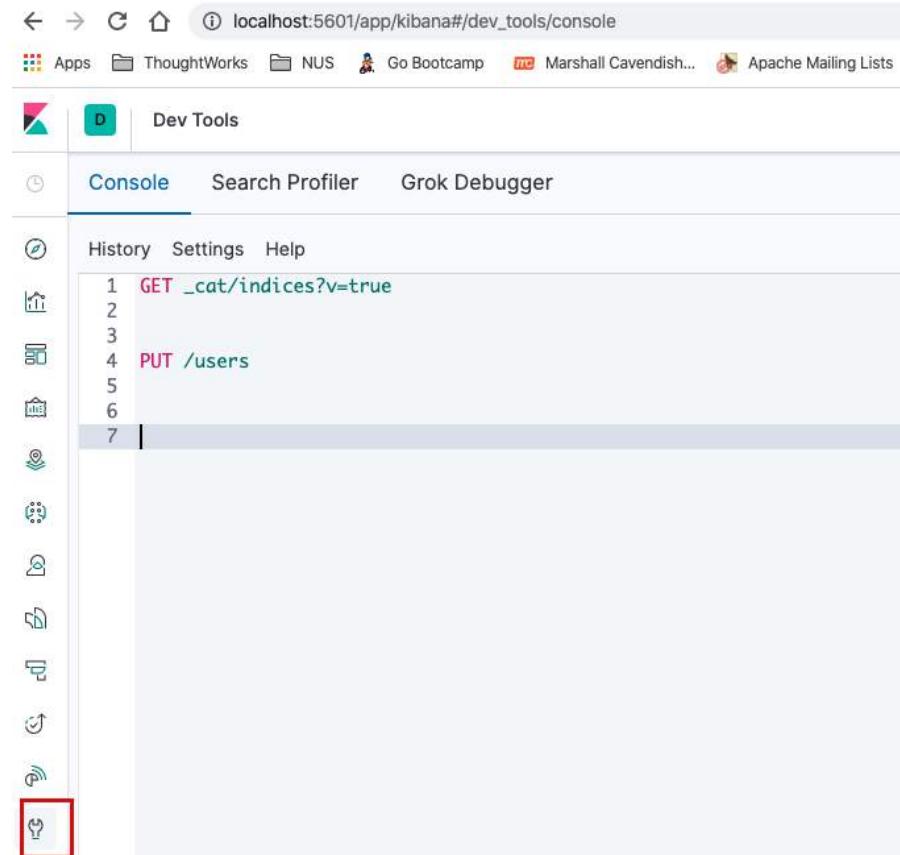
**Open up Kibana**

on your browser by pointing to

```
http://localhost:5601
```

## Open ElasticSearch Dev Tools

Click on the “Spanner” icon to launch the Dev Tools ElasticSearch query page



## Create index (Table/Collection)

```
PUT /users
```

You should get back

```
{  
  "acknowledged" : true,  
  "shards_acknowledged" : true,  
  "index" : "users"  
}
```

## Read all documents

```
GET /users/_search
```

You should get 0 hits as below:

```
{  
  "took" : 1,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 1,  
    "successful" : 1,  
    "skipped" : 0,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : {  
      "value" : 0,  
      "relation" : "eq"  
    },  
    "max_score" : null,  
    "hits" : [ ]  
  }  
}
```

## Create document with custom id

(You can also copy the json from data-management/elasticsearch/users\_single.json)

```
PUT /users/_doc/1  
{  
  "emplId": "5e5a766afa73dddb81a1c939",  
  "age": 21,  
  "eyeColor": "brown",  
  "name": "Allison Strong",  
  "gender": "male",  
  "company": "ENERFORCE",  
  "email": "allisonstrong@enerforce.com",  
  "phone": "+1 (963) 503-2603",  
  "address": {  
    "addId": "5e5a766a0e58c8b905902b40",  
    "door": 822,  
    "street": "Bushwick Place",  
    "city": "Castleton",  
    "state": "Guam",  
    "pin": 2457  
  },  
  "favoriteFruit": "banana"  
}
```

You should get back

```
{
  "_index" : "users",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 4,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 3,
  "_primary_term" : 1
}
```

## Get document by id

GET /users/\_doc/1

```
{
  "_index" : "users",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 4,
  "_seq_no" : 3,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "empId" : "5e5a766afa73dddb81a1c939",
    "age" : 21,
    "eyeColor" : "brown",
    "name" : "Allison Strong",
    "gender" : "male",
    "company" : "ENERFORCE",
    "email" : "allisonstrong@enerforce.com",
    "phone" : "+1 (963) 503-2603",
    "address" : {
      "addId" : "5e5a766a0e58c8b905902b40",
      "door" : 822,
      "street" : "Bushwick Place",
      "city" : "Castleton",
      "state" : "Guam",
      "pin" : 2457
    },
    "favoriteFruit" : "banana"
  }
}
```

## Update documents

- There's no update in elasticsearch
- The old document is replaced (deleted) with the new one.

Try changing some values for doc 1 and re-PUT the document

PUT /users/\_doc/1

```
{
```

```

"emplId": "5e5a766afa73dddb81a1c939",
"age": 21,
"eyeColor": "brown",
"name": "Allison Strong",
"gender": "male",
"company": "ENERFORCE",
"email": "allisonstrong@enerforce.com",
"phone": "+1 (963) 503-2603",
"address": {
  "addId": "5e5a766a0e58c8b905902b40",
  "door": 822,
  "street": "Bushwick Place",
  "city": "Castleton",
  "state": "Guam",
  "pin": 2457
},
"favoriteFruit": "kiwifruit"
}

```

You must see “updated” in the response

```
{
  "_index" : "users",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 5,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 4,
  "_primary_term" : 1
}
```

## Bulk upload

You can also find the json used in this exercise at [data-management/elastic/users\\_bulk.json](#)

```
POST /users/_bulk
{"index":{"_id":"10"}}
{"emplId": "5e5a76285897070407bff0b", "age": 29, "eyeColor": "brown", "name": "Conrad Norman", "gender": "male", "company": "LOCAZONE", "email": "conradnorman@locazone.com", "phone": "+1 (836) 511-3994", "address": { "addId": "5e5a7628b8cbf7a261e08166", "door": 284, "street": "Ludlam Place", "city": "Derwood", "state": "California", "pin": 6076 }, "favoriteFruit": "apple"}
```

```
{"index":{"_id":"11"}}
{"empld": "5e5a762821b4652df875a8d0", "age": 28, "eyeColor": "green", "name": "Atkinson Snider", "gender": "male", "company": "OCTOCORE", "email": "atkinsonsneider@octocore.com", "phone": "+1 (903) 461-3255", "address": { "addId": "5e5a76281d3f5ab14ed92a87", "door": 315, "street": "Flatlands Avenue", "city": "Baker", "state": "Ohio", "pin": 7405 }, "favoriteFruit": "apple"}
{"index":{"_id":"12"}}
{"empld": "5e5a76281fe5397a28480dcf", "age": 30, "eyeColor": "green", "name": "Katina Phelps", "gender": "female", "company": "SENMEI", "email": "katinaphelps@senmei.com", "phone": "+1 (935) 554-3648", "address": { "addId": "5e5a76282d085c36db741fec", "door": 507, "street": "Lewis Avenue", "city": "Rote", "state": "Alaska", "pin": 2168 }, "favoriteFruit": "apple"}
{"index":{"_id":"13"}}
{"empld": "5e5a7628b0afdb763eb508ef", "age": 30, "eyeColor": "brown", "name": "Abbott Cline", "gender": "male", "company": "COMTRACT", "email": "abbottcline@comtract.com", "phone": "+1 (966) 471-2316", "address": { "addId": "5e5a76287c3dc6f68454b035", "door": 837, "street": "Keap Street", "city": "Norvelt", "state": "Kentucky", "pin": 1947 }, "favoriteFruit": "apple"}
{"index":{"_id":"14"}}
{"empld": "5e5a762886f11fc1a5d1f289", "age": 39, "eyeColor": "green", "name": "Copeland Dejesus", "gender": "male", "company": "UNI", "email": "copelanddejesus@uni.com", "phone": "+1 (940) 539-2249", "address": { "addId": "5e5a76289eddff4d1accf774", "door": 479, "street": "Cheever Place", "city": "Grandview", "state": "Maryland", "pin": 2808 }, "favoriteFruit": "apple"}
{"index":{"_id":"15"}}
{"empld": "5e5a7628b085cd6f052aedaa", "age": 40, "eyeColor": "brown", "name": "Debra Lewis", "gender": "female", "company": "DOGTOWN", "email": "debralewis@dogtown.com", "phone": "+1 (985) 586-3140", "address": { "addId": "5e5a7628a3bfce72565b3922", "door": 144, "street": "Fuller Place", "city": "Frizzleburg", "state": "Federated States Of Micronesia", "pin": 830 }, "favoriteFruit": "apple"}
{"index":{"_id":"16"}}
{"empld": "5e5a762841294fa742351a3e", "age": 20, "eyeColor": "green", "name": "Martina Young", "gender": "female", "company": "COASH", "email": "martinayoung@coash.com", "phone": "+1 (898) 595-3459", "address": { "addId": "5e5a7628f3e5a2c30ce52893", "door": 652, "street": "Banker Street", "city": "Diaperville", "state": "Northern Mariana Islands", "pin": 7552 }, "favoriteFruit": "apple"}
{"index":{"_id":"17"}}
{"empld": "5e5a762806a4c092f3cd157b", "age": 31, "eyeColor": "brown", "name": "Terra Goodman", "gender": "female", "company": "TWIGGERY", "email": "terragoodman@twiggery.com", "phone": "+1 (845) 575-3415", "address": { "addId": "5e5a762856baa2032ada8be6", "door": 515, "street": "Horace Court", "city": "Windsor", "state": "Connecticut", "pin": 9206 }, "favoriteFruit": "banana"}
{"index":{"_id":"18"}}
{"empld": "5e5a7628848dee3df1a5ef47", "age": 23, "eyeColor": "green", "name": "Hester Schroeder", "gender": "female", "company": "SUNCLIPSE", "email": "hesterschroeder@suncipse.com", "phone": "+1 (940) 500-3089", "address": { "addId": "5e5a762871308cdbd866f95a", "door": 951, "street": "Harden Street", "city": "Ballico", "state": "Georgia", "pin": 8775 }, "favoriteFruit": "strawberry"}
 {"index":{"_id":"19"}}
 {"empld": "5e5a7628c5d9f93798b8f9df", "age": 24, "eyeColor": "blue", "name": "Jan Sherman", "gender": "female", "company": "MIRACLIS", "email": "jansherman@miraclis.com", "phone": "+1 (986) 466-2510", "address": { "addId": "5e5a7628b230d375ef79b9f9", "door": 631, "street": "Vandam Street", "city": "Eagletown", "state": "Missouri", "pin": 7275 }, "favoriteFruit": "apple"}
```

Your output should look something like this:

```
{  
  "took" : 31,  
  "errors" : false,  
  "items" : [  
    {  
      "index" : {  
        "_index" : "users",  
        "_type" : "_doc",  
        "_id" : "10",  
        "_version" : 1,  
        "result" : "created",  
        "_shards" : {  
          "total" : 2,  
          "successful" : 1,  
          "failed" : 0  
        },  
        "_seq_no" : 6,  
        "_primary_term" : 1,  
        "status" : 201  
      }  
    },  
    {  
      "index" : {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "11",  
        "_version" : 1,  
        "result" : "created",  
        "_shards" : {  
          "total" : 2,  
          "successful" : 1,  
          "failed" : 0  
        },  
        "_seq_no" : 7,  
        "_primary_term" : 1,  
        "status" : 201  
      }  
    }  
  ]  
}
```

## Search all documents

```
GET /users/_search  
{  
  "query": {  
    "match_all": {}  
  }  
}
```

Your output should look something like this:

```
{  
  "took" : 1,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 1,  
    "successful" : 1,  
    "skipped" : 0,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : {  
      "value" : 11,  
      "relation" : "eq"  
    },  
    "max_score" : 1.0,  
    "hits" : [  
      {  
        "_index" : "users",  
        "_type" : "_doc",  
        "_id" : "1",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766afa73dddb81a1c939",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "2",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "3",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "4",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "5",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "6",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "7",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "8",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "9",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "10",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      },  
      {  
        "_index" : "users",  
        "_score" : 1.0,  
        "_type" : "_doc",  
        "_id" : "11",  
        "_score" : 1.0,  
        "_source" : {  
          "empId" : "5e5a766a0e58c8b905902b40",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      }  
    ]  
  }  
}
```

## Search by query

```
GET /users/_search  
{  
  "query": {  
    "match": {  
      "gender": "male"  
    }  
  }  
}
```

Your output should look like

```
{  
  "took" : 2,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 1,  
    "successful" : 1,  
    "skipped" : 0,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : {  
      "value" : 5,  
      "relation" : "eq"  
    },  
    "max_score" : 0.7801585,  
    "hits" : [  
      {  
        "_index" : "users",  
        "_type" : "_doc",  
        "_id" : "1",  
        "_score" : 0.7801585,  
        "_source" : {  
          "empId" : "5e5a766afa73dddb81a1c939",  
          "age" : 21,  
          "eyeColor" : "brown",  
          "name" : "Allison Strong",  
          "gender" : "male",  
          "company" : "ENERFORCE",  
          "email" : "allisonstrong@enerforce.com",  
          "phone" : "+1 (963) 503-2603",  
          "address" : {  
            "addId" : "5e5a766a0e58c8b905902b40",  
            "door" : 822,  
            "street" : "Bushwick Place",  
            "city" : "Castleton",  
            "state" : "Guam",  
            "pin" : 2457  
          },  
          "favoriteFruit" : "kiwifruit"  
        }  
      }  
    ]  
  }  
}
```

## Delete a Document

```
DELETE /users/_doc/1
```

## Redis

Check the name of your container by executing the following command (it must be redis in our case)

```
docker ps
```

Login to your redis docker container using the following command

```
docker exec -it redis /bin/bash
```

You should see the following output

```
› docker exec -it redis /bin/bash
root@redis:/data#
```

Issue the following command to get into the redis-cli prompt

```
root@redis:/data# redis-cli
```

Issue the following command to get into the redis-cli prompt

```
root@redis:/data# redis-cli
127.0.0.1:6379>
```

Redis frequently used datatypes :

1. Lists
2. Sets
3. Hashes

### Lists

#### Push to a List

```
RPUSH users Arun
```

```
127.0.0.1:6379> RPUSH users Arun
(integer) 1
```

#### Push multiple values to a List

```
RPUSH users Alice Daisy Jason
```

```
127.0.0.1:6379> RPUSH users Alice Daisy Jason
(integer) 4
```

## Get values from a List

LRANGE users "Alice" "Daisy" "Jason"

```
127.0.0.1:6379> LRANGE users 0 -1
1) "Arun"
2) "Alice"
3) "Daisy"
4) "Jason"
```

## Removing a value from a List

LPOP users

```
127.0.0.1:6379> LPOP users
"Arun"
```

## Removing a specific value from a List

LREM users 1 "Daisy"

```
127.0.0.1:6379> LREM users 1 "Daisy"
(integer) 1
127.0.0.1:6379> LRANGE users 0 -1
1) "Alice"
2) "Jason"
```

## Find the length of the List

LLLEN users

```
127.0.0.1:6379> LLLEN users
(integer) 2
```

## Sets

### Push to a Set

SADD books "machine learning"

```
127.0.0.1:6379> SADD books "machine learning"
(integer) 1
```

### Push multiple values to a Set

SADD books "machine learning" "microservices" "data science" "machine learning"

```
127.0.0.1:6379> SADD books "machine learning" "microservices" "data science" "machine learning"  
(integer) 2
```

## Listing members in a Set

```
SMEMBERS books
```

```
127.0.0.1:6379> SMEMBERS books  
1) "data science"  
2) "microservices"  
3) "machine learning"
```

## Removing a value from a Set

```
SREM books "microservices"
```

```
127.0.0.1:6379> SREM books "microservices"  
(integer) 1
```

## Checking membership in a Set

```
SISMEMBER books "machine learning"
```

```
SISMEMBER books "no book"
```

```
127.0.0.1:6379> SISMEMBER books "machine learning"  
(integer) 1  
127.0.0.1:6379> SISMEMBER books "no book"  
(integer) 0
```

## Hashes

### Push to a Hash

```
HSET phones "apple" 11  
HSET phones "samsung" 25
```

HSET phones "LG" 16

```
127.0.0.1:6379> HSET phones "apple" 11
(integer) 1
127.0.0.1:6379> HSET phones "samsung" 25
(integer) 1
127.0.0.1:6379> HSET phones "LG" 16
(integer) 1
```

### Pushing multiple values to a Hash

HMSET phones "Oppo" 14 "Huawei" 17 "Redmi" 29

```
127.0.0.1:6379> HMSET phones "Oppo" 14 "Huawei" 17 "Redmi" 29
OK
```

### Get value for a key from a Hash

HGET phones "LG"

```
127.0.0.1:6379> HGET phones "LG"
"16"
```

### Get all values from a Hash

HGETALL phones

```
127.0.0.1:6379> HGETALL phones
1) "apple"
2) "11"
3) "samsung"
4) "25"
5) "LG"
6) "16"
7) "Oppo"
8) "14"
9) "Huawei"
10) "17"
11) "Redmi"
12) "29"
```

## Removing a value from a Hash

```
HDEL phones "Oppo"
```

```
127.0.0.1:6379> HDEL phones "Oppo"
(integer) 1
127.0.0.1:6379> HGETALL phones
1) "apple"
2) "11"
3) "samsung"
4) "25"
5) "LG"
6) "16"
7) "Huawei"
8) "17"
9) "Redmi"
10) "29"
```

# Neo4J

## Open up Neo4J browser

by pointing your browser to

```
http://localhost:7474
```

## Create new Nodes

```
CREATE (alice:Person {name:"Alice Wells", gender:"female", age:36})
CREATE (jeremy:Person {name:"Jeremy Smith", gender:"male", age:40})
```

The screenshot shows the Neo4j Browser interface. At the top, there are two lines of Cypher code:

```
1 CREATE (alice:User {name:"Alice Wells", gender: "female", age:36})
2 CREATE (jeremy:User {name:"Jeremy Smith", gender: "male", age:40})
```

Below the code, a blue bar says "To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)".

The main area shows the command prompt "neo4j\$" followed by the executed query and its results:

```
neo4j$ CREATE (alice:User {name:"Alice Wells", gender: "female", age:36}) CREATE (jeremy:User {name:"Jeremy Smith", gender: "male", age:40})
Added 2 labels, created 2 nodes, set 6 properties, completed after 79 ms.
```

At the bottom left, there are two buttons: "Table" and "Code".

## Create new Relationship

```
MATCH (a:Person), (j:Person) WHERE a.name="Alice Wells" AND j.name="Jeremy Smith"
CREATE (a)-[:Married]->(j)
```

The screenshot shows the Neo4j Browser interface. At the top, there are two lines of Cypher code:

```
1 MATCH (a:User), (j:User) WHERE a.name="Alice Wells" AND j.name="Jeremy Smith"
2 CREATE (a)-[:Married]-(j)
```

Below the code, a blue bar says "To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)".

The main area shows the command prompt "neo4j\$" followed by the executed query and its results:

```
neo4j$ MATCH (a:User), (j:User) WHERE a.name="Alice Wells" AND j.name="Jeremy Smith" CREATE (a)-[:Married]-(j)
Created 1 relationship, completed after 2 ms.
```

At the bottom left, there are two buttons: "Table" and "Code".

## Query a single node

```
MATCH (u:Person) WHERE u.gender='female' RETURN u
```

The screenshot shows the Neo4j Browser interface. At the top, there is a command line input field containing the query: `neo4j$ MATCH (u:User) WHERE u.gender='female' RETURN u`. Below the command line, a message says "To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)". The main area displays a single node named "Alice Wells" represented by a purple circle. On the left side, there is a sidebar with four tabs: "Graph" (selected), "Table", "Text", and "Code". Above the "Graph" tab, there are two status indicators: "(1)" and "User(1)".

## Get all nodes and relationships

```
MATCH (n1)-[r]->(n2) RETURN r, n1, n2
```

(or)

```
MATCH (n) RETURN n
```

```
1 // Get some data
2 MATCH (n1)-[r]→(n2) RETURN r, n1, n2
```

To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)

```
neo4j$ MATCH (n1)-[r]→(n2) RETURN r, n1, n2
```

Graph

\*(2) User(2)  
\*(1) Married(1)

Table

A Text

Code

```
graph LR; AliceWellns((Alice Wells)) -- Married --> JeremySmith((Jeremy Smith))
```

## Deleting a node

(Relationship has to be deleted first before deleting node)

```
MATCH (n1)-[r]->(n2) DELETE r
```

## Delete Relationship

```
neo4j$ MATCH (n1)-[r]→(n2) DELETE r
```

To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)

```
neo4j$ MATCH (n1)-[r]→(n2) DELETE r
```

Table

Deleted 2 relationships, completed after 3 ms.

Code

## Delete Nodes

```
MATCH (n) DELETE n
```

```
neo4j$ MATCH (n) DELETE n
```

To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)

```
neo4j$ MATCH (n) DELETE n
```

Deleted 2 nodes, completed after 1 ms.

Table  
Code

## Load Graph from CSV

Copy the users.csv from data-management/neo4j directory into the neo4j docker container.

### Issue the following command from your terminal

```
docker cp neo4j/users.csv neo4j:/var/lib/neo4j/import/users.csv
```

### Issue the following command from your Neo4J browser window

#### Load CSV

```
LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row
CREATE (:Person{name: row.name, gender: row.gender, company: row.company})
```

```
1 LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row
2 CREATE (:Person{name: row.name, gender: row.gender, company: row.company})
```

To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#)

```
neo4j$ LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row CREATE (:Person{name: row.name, gender: row.gender, company: row.company})
```

Added 10 labels, created 10 nodes, set 30 properties, completed after 220 ms.

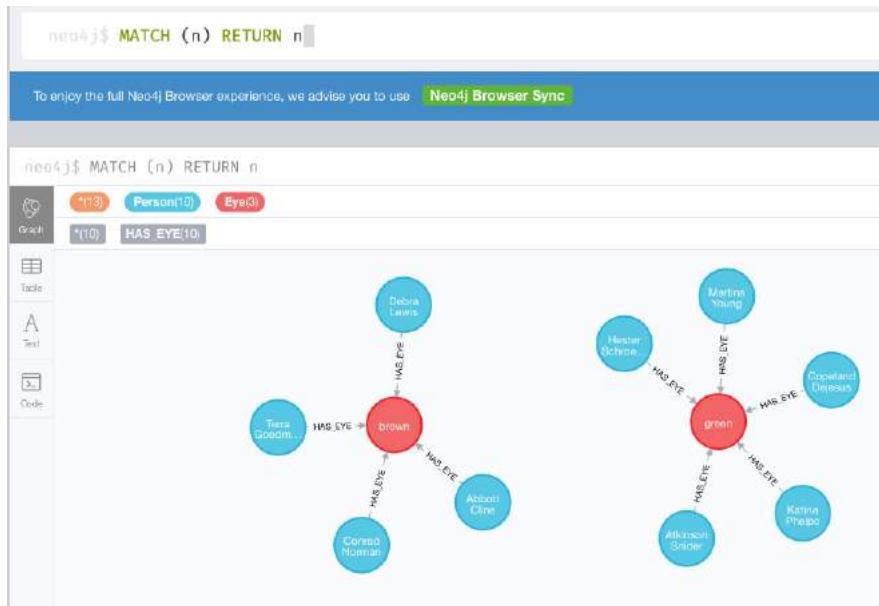
Table  
Code



## Create Nodes and Relationships

```
LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row
CREATE (person:Person{name: row.name, gender: row.gender, company: row.company})
MERGE (eye:Eye{color: row.eyeColor})
CREATE (person)-[:HAS_EYE]->(eye)
```





## MongoDB

Check the name of your container by executing the following command (it must be `mongodb` in our case)

```
docker ps
```

Login to your `mongodb` docker container using the following command

```
docker exec -it mongodb /bin/bash
```

You should see the following output

```
> docker exec -it mongodb /bin/bash
root@mongodb:/#
```

Issue the following command to get into the mongo prompt

```
mongo --username root --password
```

Password is **rootpassword** as specified in your `docker-compose.yml`

```
root@mongodb:/# mongo --username root --password
MongoDB shell version v4.2.3
Enter password:
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("60e37399-3bec-449c-a989-e6680ab3f416") }
MongoDB server version: 4.2.3
Server has startup warnings:
2020-03-01T02:39:59.474+0000 I STORAGE [initandlisten]
2020-03-01T02:39:59.474+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS file system is strongly recommended!
```

## Create/Switch to a database

```
use usersdb
```

## Create a new collection

```
db.createCollection("users")
```

```
> db.createCollection("users")
{ "ok" : 1 }
```

## Insert a new document

You can get a sample document from data-management/mongo/users\_single.json

```
db.users.insert({ "emplId": "5e5a766afa73dddb81a1c939", "age": 21, "eyeColor": "brown", "name": "Allison Strong", "gender": "male", "company": "ENERFORCE", "email": "allisonstrong@enerforce.com", "phone": "+1 (963) 503-2603", "address": { "addId": "5e5a766a0e58c8b905902b40", "door": 822, "street": "Bushwick Place", "city": "Castleton", "state": "Guam", "pin": 2457 }, "favoriteFruit": "banana" })
```

(Insert is a function call. Notice the parenthesis before the JSON argument)

```
> db.users.insert({ "emplId": "5e5a766afa73dddb81a1c939", "age": 21, "eyeColor": "brown", "name": "Allison Strong", "gender": "male", "company": "ENERFORCE", "email": "allisonstrong@enerforce.com", "phone": "+1 (963) 503-2603", "address": { "addId": "5e5a766a0e58c8b905902b40", "door": 822, "street": "Bushwick Place", "city": "Castleton", "state": "Guam", "pin": 2457 }, "favoriteFruit": "banana" })
WriteResult({ "nInserted" : 1 })
```

## Bulk insert documents

Bulk insertion is done by passing an array into the insert function. You can get a sample document from data-management/mongo/users\_bulk.json

```
db.users.insert([
... { "_id": "10", "emplId": "5e5a76285897070407bffd0b", "age": 29, "eyeColor": "brown", "name": "Conrad Norman", "gender": "male", "company": "LOCAZONE", "email": "conradnorman@locazone.com", "phone": "+1 (836) 511-3994", "address": { "addId": "5e5a7628b8cbf7a261e08166", "door": 284, "street": "Ludlam Place", "city": "Derwood", "state": "California", "pin": 6076 }, "favoriteFruit": "apple" },
```

```

... {"_id": "11", "emplId": "5e5a762821b4652df875a8d0", "age": 28, "eyeColor": "green", "name": "Atkinson Snider", "gender": "male", "company": "OCTOCORE", "email": "atkinsonsniider@octocore.com", "phone": "+1 (903) 461-3255", "address": { "addId": "5e5a76281d3f5ab14ed92a87", "door": 315, "street": "Flatlands Avenue", "city": "Baker", "state": "Ohio", "pin": 7405 }, "favoriteFruit": "apple"},  

... {"_id": "12", "emplId": "5e5a76281fe5397a28480dcf", "age": 30, "eyeColor": "green", "name": "Katina Phelps", "gender": "female", "company": "SENMEI", "email": "katinaphelps@senmei.com", "phone": "+1 (935) 554-3648", "address": { "addId": "5e5a76282d085c36db741fec", "door": 507, "street": "Lewis Avenue", "city": "Rote", "state": "Alaska", "pin": 2168 }, "favoriteFruit": "apple"},  

... {"_id": "13", "emplId": "5e5a7628b0afdb763eb508ef", "age": 30, "eyeColor": "brown", "name": "Abbott Cline", "gender": "male", "company": "COMTRACT", "email": "abbottcline@comtract.com", "phone": "+1 (966) 471-2316", "address": { "addId": "5e5a76287c3dc6f68454b035", "door": 837, "street": "Keap Street", "city": "Norvelt", "state": "Kentucky", "pin": 1947 }, "favoriteFruit": "apple"},  

... {"_id": "14", "emplId": "5e5a762886f11fc1a5d1f289", "age": 39, "eyeColor": "green", "name": "Copeland Dejesus", "gender": "male", "company": "UNI", "email": "copelanddejesus@uni.com", "phone": "+1 (940) 539-2249", "address": { "addId": "5e5a76289eddff4d1accf774", "door": 479, "street": "Cheever Place", "city": "Grandview", "state": "Maryland", "pin": 2808 }, "favoriteFruit": "apple"},  

... {"_id": "15", "emplId": "5e5a7628b085cd6f052aedaa", "age": 40, "eyeColor": "brown", "name": "Debra Lewis", "gender": "female", "company": "DOGTOWN", "email": "debralewis@dogtown.com", "phone": "+1 (985) 586-3140", "address": { "addId": "5e5a7628a3bfce72565b3922", "door": 144, "street": "Fuller Place", "city": "Frizzleburg", "state": "Federated States Of Micronesia", "pin": 830 }, "favoriteFruit": "apple"},  

... {"_id": "16", "emplId": "5e5a762841294fa742351a3e", "age": 20, "eyeColor": "green", "name": "Martina Young", "gender": "female", "company": "COASH", "email": "martinayoung@coash.com", "phone": "+1 (898) 595-3459", "address": { "addId": "5e5a7628f3e5a2c30ce52893", "door": 652, "street": "Banker Street", "city": "Diaperville", "state": "Northern Mariana Islands", "pin": 7552 }, "favoriteFruit": "apple"},  

... {"_id": "17", "emplId": "5e5a762806a4c092f3cd157b", "age": 31, "eyeColor": "brown", "name": "Terra Goodman", "gender": "female", "company": "TWIGGERY", "email": "terragoodman@twiggery.com", "phone": "+1 (845) 575-3415", "address": { "addId": "5e5a762856baa2032ada8be6", "door": 515, "street": "Horace Court", "city": "Windsor", "state": "Connecticut", "pin": 9206 }, "favoriteFruit": "banana"},  

... {"_id": "18", "emplId": "5e5a7628848dee3df1a5ef47", "age": 23, "eyeColor": "green", "name": "Hester Schroeder", "gender": "female", "company": "SUNCLIPSE", "email": "hesterschroeder@suncipse.com", "phone": "+1 (940) 500-3089", "address": { "addId": "5e5a762871308cdbd866f95a", "door": 951, "street": "Harden Street", "city": "Ballico", "state": "Georgia", "pin": 8775 }, "favoriteFruit": "strawberry"},  

... {"_id": "19", "emplId": "5e5a7628c5d9f93798b8f9df", "age": 24, "eyeColor": "blue", "name": "Jan Sherman", "gender": "female", "company": "MIRACLIS", "email": "jansherman@miraclis.com", "phone": "+1 (986) 466-2510", "address": { "addId": "5e5a7628b230d375ef79b9f9", "door": 631, "street": "Vandam Street", "city": "Eagletown", "state": "Missouri", "pin": 7275 }, "favoriteFruit": "apple"},  

... ])}

```

```
... {"_id": "19", "empId": "5e5a7628c5d9f93798b8f9df", "company": "MIRACLIS", "email": "jansherman@miraclis.com", "street": "Vandam Street", "city": "Eagle Rock", "state": "CA", "zip": "90041", "door": 631, "pin": 2457, "favoriteFruit": "apple"}, ... ]
BulkWriteResult({
    "writeErrors" : [ ],
    "writeConcernErrors" : [ ],
    "nInserted" : 10,
    "nUpserted" : 0,
    "nMatched" : 0,
    "nModified" : 0,
    "nRemoved" : 0,
    "upserted" : [ ]
})
```

## Query all documents in a collection

```
db.users.find()
> db.users.find()
{ "_id" : ObjectId("5e5b2577d48d8c4f88ae0c9c"), "empId" : "5e5a766a0e58c8b905902b40", "name" : "Conrad Norman", "gender" : "male", "company" : "ENERFORCE", "address" : { "addId" : "5e5a766a0e58c8b905902b40", "door" : 284, "pin" : 2457 }, "favoriteFruit" : "banana" }
{ "_id" : "10", "empId" : "5e5a76285897070407bffd0b", "name" : "Janson Sherman", "gender" : "female", "company" : "LOCAZONE", "email" : "conradnorman@locazone.com", "street": "Ludlam Place", "city": "Eagle Rock", "state": "CA", "zip": "90041", "door": 631, "pin": 2457, "favoriteFruit": "apple" }
```

## Find document in a collection matching a criteria

```
db.users.find({"eyeColor": "blue"})
> db.users.find({"eyeColor": "blue"})
{ "_id" : "19", "empId" : "5e5a7628c5d9f93798b8f9df", "name" : "Debra Sherman", "age" : 24, "company" : "MIRACLIS", "email" : "jansherman@miraclis.com", "street": "Vandam Street", "city": "Eagle Rock", "state": "CA", "zip": "90041", "door": 631, "pin": 2457, "favoriteFruit": "apple" }
```

## Find document in a collection matching multiple criteria

### With regular expression matching

```
#Second condition is equivalent to "Debra%"
db.users.find({$and:[{"eyeColor": "brown"}, {"name": /Debra/}]})

#Second condition is equivalent to "%Lewis"
db.users.find({$and:[{"eyeColor": "brown"}, {"name": /Lewis$/}]})
```

```
#Second condition is equivalent to "%Lew%"  
db.users.find({$and:[{"eyeColor": "brown"}, {"name": /Lew/}]})
```

```
> db.users.find({$and:[{"eyeColor": "brown"}, {"name": /Debra/}]})  
{ "_id" : "15", "empId" : "5e5a7628b085cd6f052aedaa", "age" : 40,  
  "company" : "DOGTOWN", "email" : "debralewis@dogtown.com", "phone"  
  "565b3922", "door" : 144, "street" : "Fuller Place", "city" : "Friz  
, "favoriteFruit" : "apple" }
```

## Removing a document

```
db.users.remove({"eyeColor": "blue"})
```

```
> db.users.remove({"eyeColor": "blue"})  
WriteResult({ "nRemoved" : 1 })  
> db.users.find({"eyeColor": "blue"})
```

## Updating a document

```
db.users.update({"name": "Debra Lewis"}, {$set:{"favoriteFruit": "kiwi"})
```

```
> db.users.update({"name": "Debra Lewis"}, {$set:{"favoriteFruit": "kiwi"})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
> db.users.find({"name": "Debra Lewis"})  
{ "_id" : "15", "empId" : "5e5a7628b085cd6f052aedaa", "age" : 40, "eyeColor" :  
  "blue", "company" : "DOGTOWN", "email" : "debralewis@dogtown.com", "phone" : "+1 (985  
 72565b3922", "door" : 144, "street" : "Fuller Place", "city" : "Frizzleburg", "  
 30 }, "favoriteFruit" : "kiwi" }
```

---

## End of workshop