Microservice Application Architecture…

MICROSERVICE APPLICATION ARCHITECTURE

“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.”

- Martin Fowler

© 2020 by Innovation In Software Corporation

MICROSERVICE APPLICATION ARCHITECTURE

By the end of the course you will:

- Have a clear understanding of microservices application architecture
- Have a clear understanding about the use of Docker and Kubernetes and how these technologies relate to microservices architecture design and implementation
- Gain hands on experience working with microservices in a range of labs
- Examine various use cases and architecture patterns for microservice based applications with focused discussions on microservice communications, transactions and state management.
- Have the skills and information necessary to begin designing and working with microservice based applications.

© 2020 by Innovation In Software Corporation

LOGISTICS

Class Hours:

- Start time is 8:30am
- End time is 4:00pm
- Class times may vary slightly for specific classes
- 10 minute breaks roughly at the top of every hour

Telecommunication:

- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Lunch:

- Lunch is noon to 1:15pm
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.

Miscellaneous

- Courseware
- Bathroom
- Fire drills

© 2020 by Innovation In Software Corporation

COURSE STRUCTURE

The course is divided into two components

- Core Concepts and Practices
- Additional Topics

© 2020 by Innovation In Software Corporation

SCHEDULE OF EVENTS

Day 1

- Introductions
- Understanding the Scope of Activities
- Wise Sayings: Once picture is worth 1000 words
- Pros and Cons
- Docker 101
- Kubernetes 101
- Microservices 101
  - Deployment Units
  - Microservice Communication
  - Microservice Orchestration
  - Service Discovery
  - Service Management
- Decomposition Analysis

Optional Lunch-Time Lab

- Installing Docker on your local machine

Demos and Labs

- **Lab:** Containerizing WiseSayings
- **Lab:** WiseSaying Under Kubernetes
- **Lab:** Lambda Microservice Architecture Using GraphQL
- **Lab:** Service Discovery using K8S DNS Part 1
- **Lab:** Service Discovery using K8S DNS Part 2
- **Lab:** Service Mesh on Google Cloud

© 2020 by Innovation In Software Corporation

SCHEDULE OF EVENTS

Day 2

- Review of Day 1
- Data Management
- Decomposition Under the Microscope
- Group Activity
  - Designing a Microservice
  - Implementing a Microservice

Labs

- Decomposing a Monolithic App into an MOA

Group Activity

- Designing and Implementing a Microservice

Lunch-Time

- Office hours
- Supplemental Labs
  - init containers
  - Kubernetes Secrets
  - Kubernetes TLS
  - Using Grafana
  - Creating an API Gateway on AWS

© 2020 by Innovation In Software Corporation

SOURCE CODE AND STUFF

Lab Code

https://github.com/GeorgeNiece/innosoft/microservices-architecture

Case Study Code

https://github.com/reselbob/fortune-cookies

meet the session leader

George Niece

Cloud-native, DevOps, Software Engineering, and IoT Consultant with a Linux sysadmin background. Focused on cloud-native application modernization

Twitter
@georgeniece

LinkedIn
Linkedin.com/in/GeorgeNiece

mail
George.Niece@DigitalTransformationStrategies.net

Expertise

- Cloud
- AppDev
- IoT
- Automation
- CI/CD
- Microservices
- Agile

INTRODUCTION - GTKY

- Name
- Title or your role
- Languages/Areas of expertise
- Why are you taking this workshop?
- Rate yourself (1-5) 1 being a beginner, 5 being a SME on
  1. Containerization
  2. Container Orchestration
  3. Microservices

© 2020 by Innovation In Software Corporation

GEORGE NIECE

Developer, Architect, Industry Analyst

https://www.linkedin.com/in/georgeniece

© 2020 by Innovation In Software Corporation

CORE CONCEPTS & PRACTICES

Core Concepts and Practices are the things we've got to cover no matter what:

- Understanding what a microservice is and how they are used
- Deployment Units
- Microservice Communication
- Service Orchestration
- Service Discovery
- Service Management
- Data Management
- Decomposition
- Basic Design and Implementation

ADDITIONAL TOPICS

Additional topics is concept and practices that you should know, but if we covered each one we'd run out of time and wouldn't get through core concepts and practices.

Additional topics are provided as appendices in this presentation deck. They will be covered provided:

There is time

There is group consensus about which topics to cover

- 4 Types of Distributed Architecture
- Docker Compose
- 12 Factor App
- Messaging
- Monitoring
- Context and Boundary
- Init Containers
- Microservice Security
  Microservice Patterns
- API Gateway
- Service Discovery

Fortune Cookies

A marketing company wants your group to design an application that will deliver a “fortune cookie” to willing recipients. All that’s required is that the user register with the app using either Facebook, Google or LinkedIn credentials. Or the user can custom register by providing standard profile information. The fortune cookie text is provided by the Fortunes service. Each fortune cookie is accompanied by an advertisement from “Ad Auction” service. However, users need to be provided with with the option to not receive advertisements. Also, a feature needs to be implemented supporting a Fortune Cookie presence on Twitter by sending Fortune Cookie tweets out at configurable intervals. Fortune Cookie needs to work on iPhone, Android, standard web pages, Alexa and SMS. In addition, Fortune Cookie needs to provide usage and performance metrics on demand to administrative personnel. Fortune cookie is a free application.

© 2020 by Innovation In Software Corporation

INTERACTIVE LAB ENVIRONMENTS

- We will use **Katacoda**; a free interactive virtual environment that provides virtual machine instances of:
  - Ubuntu with Docker installed
  - Kubernetes

COURSE AND EXPERIMENTS ON GITHUB AT:

https://github.com/GeorgeNiece/innosoft.git

© 2020 by Innovation In Software Corporation

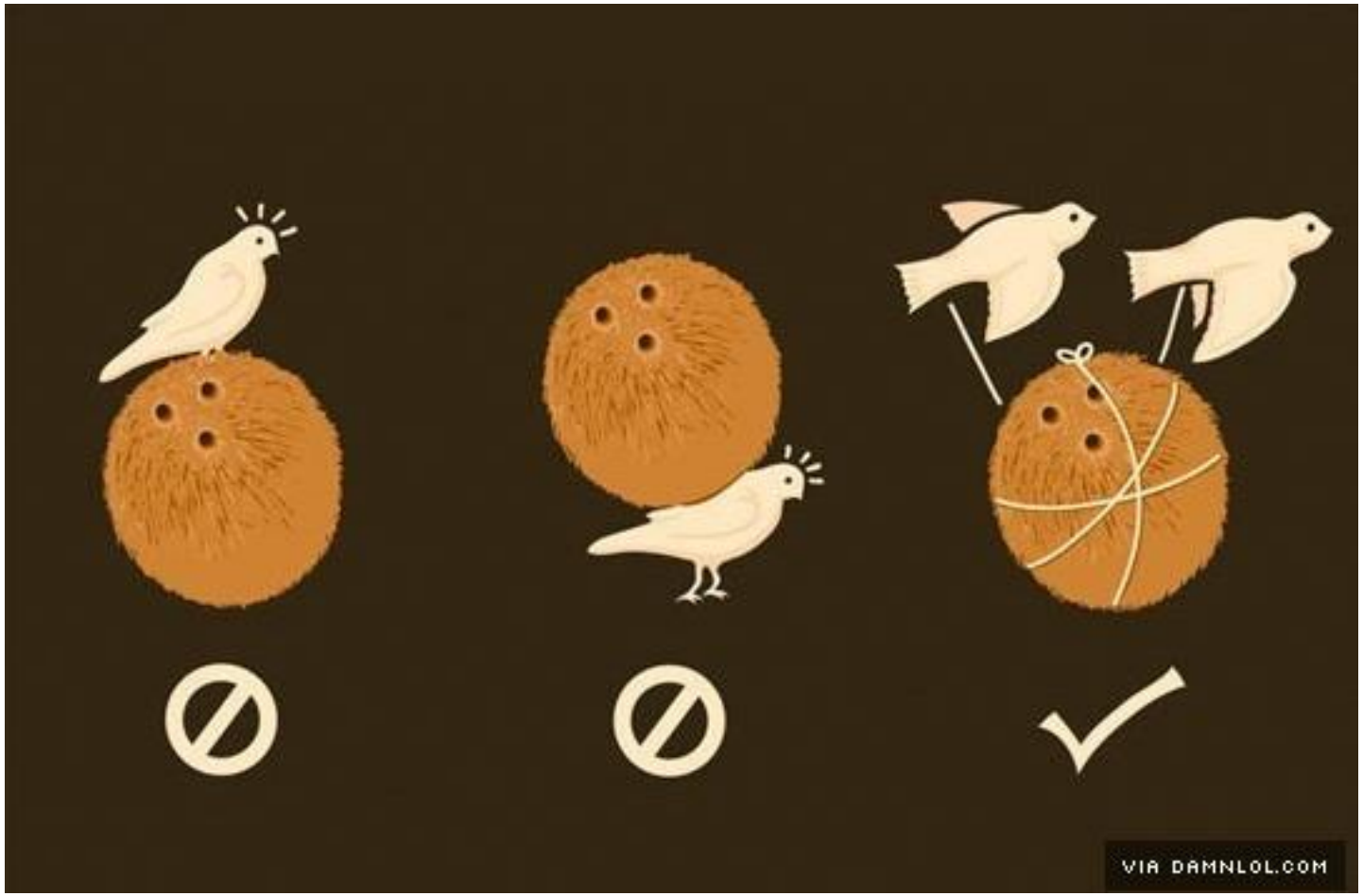
QUESTIONS?

Does anyone have any questions before we begin?

© 2020 by Innovation In Software Corporation

RANDOM QUESTION…

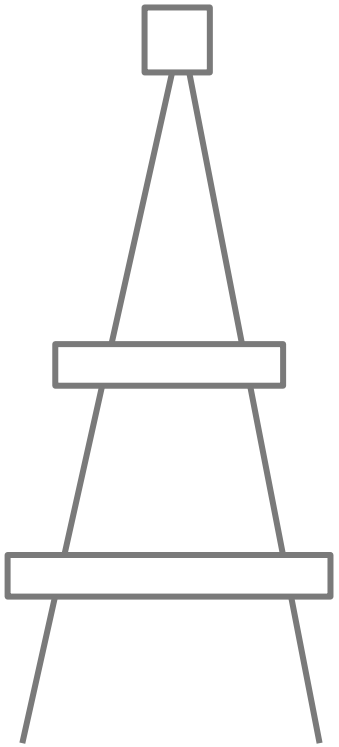
What is the airspeed of an unladen swallow?

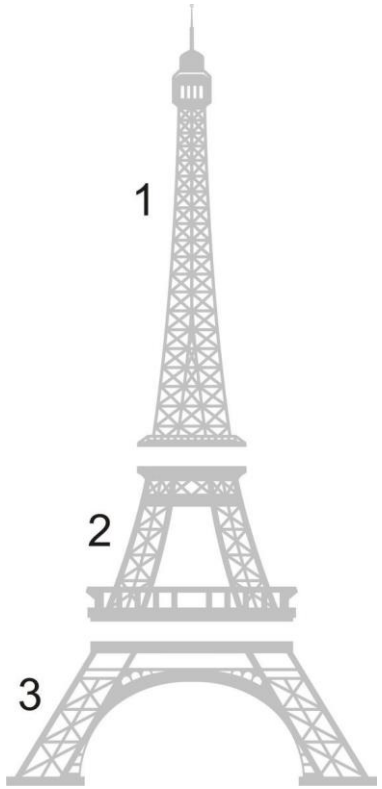


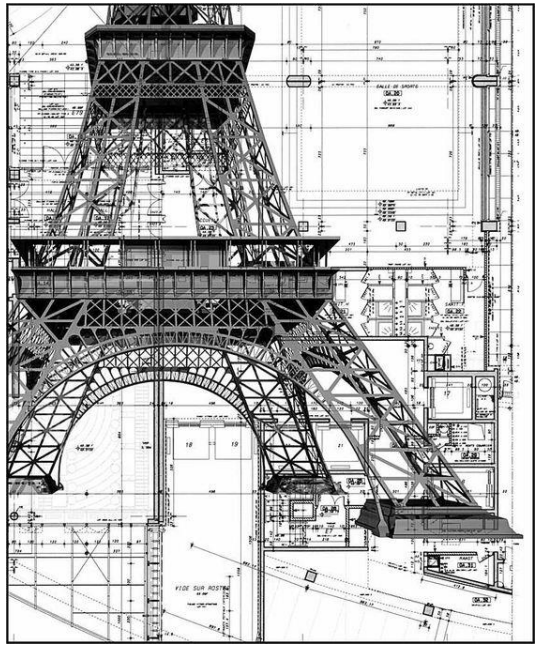
© 2020 by Innovation In Software Corporation

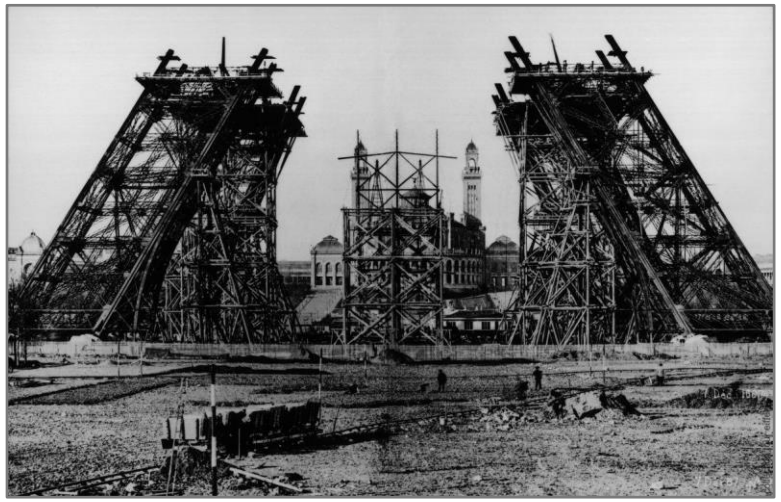
A PHILOSOPHY FOR ARCHITECTURAL DESIGN

Conceptual → Logical → Structural → Concrete





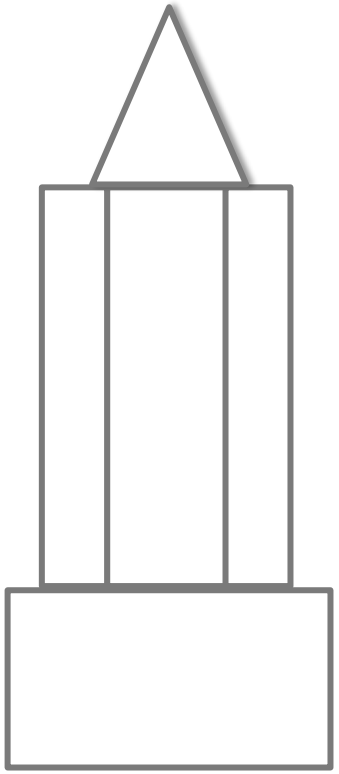


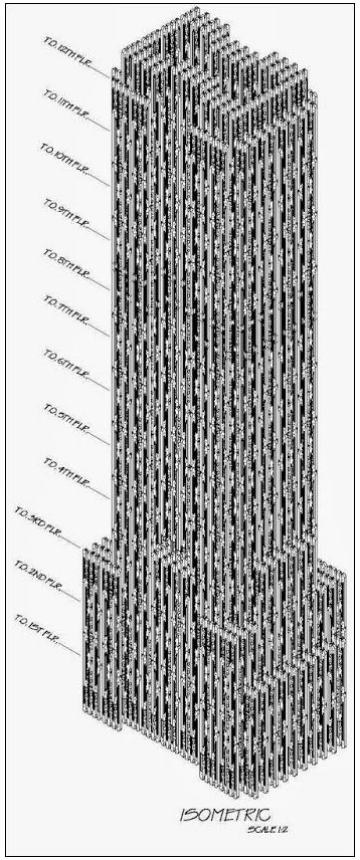


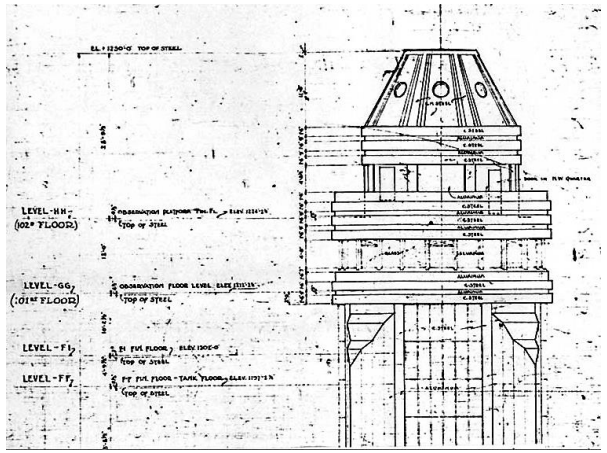
© 2020 by Innovation In Software Corporation

A PHILOSOPHY FOR ARCHITECTURAL DESIGN

Conceptual → Logical → Structural → Concrete



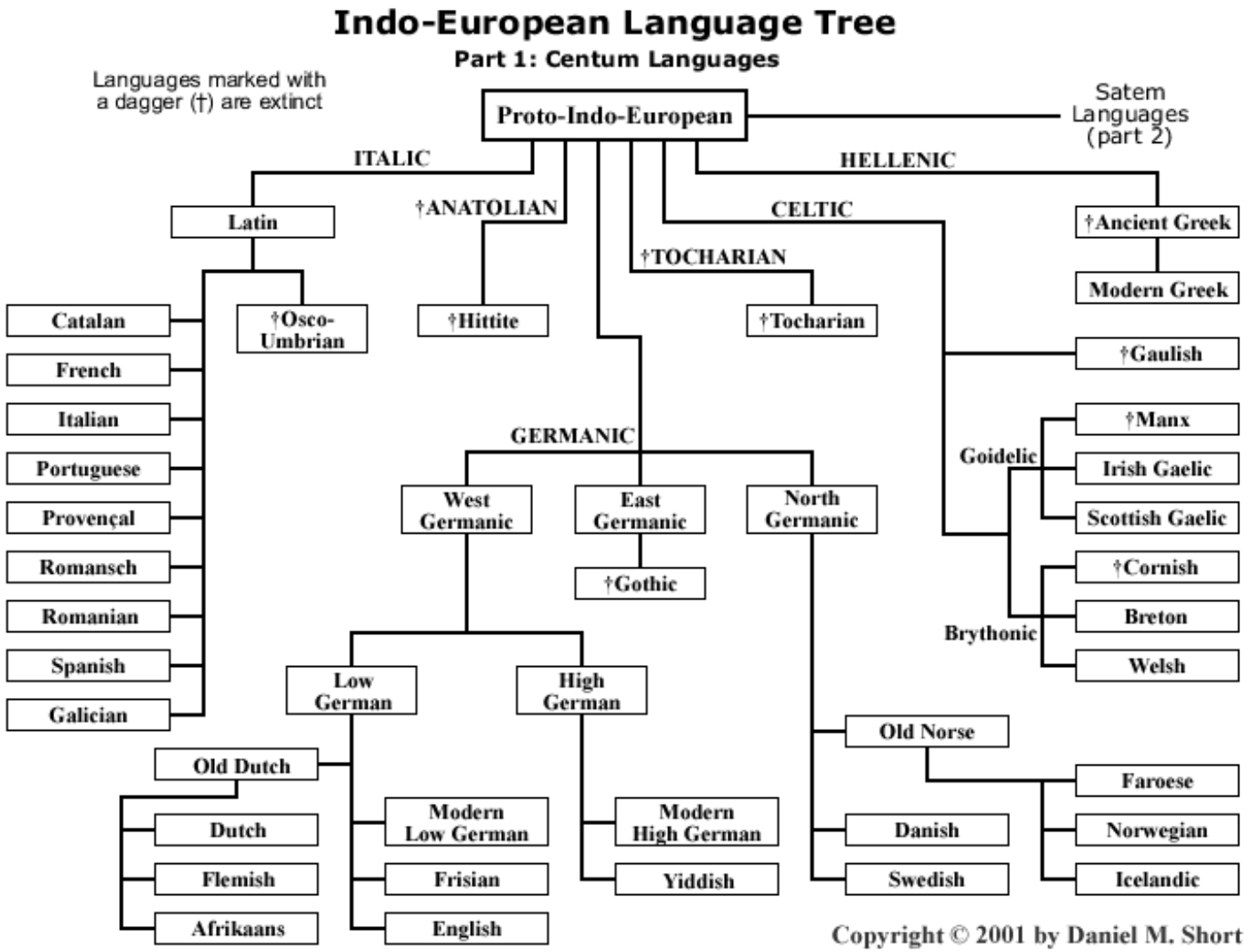




© 2020 by Innovation In Software Corporation

SOMETHING TO CONSIDER……

The higher you go in the abstraction tree, the more things begin to look alike



© 2020 by Innovation In Software Corporation

KNOW YOUR BUSINESS

Where is your company one the spectrum?

Netflix

"Make sure the right person is seeing the right movie"

General Atomics

"Make sure the right person is hitting the right target"

The Spectrum of Software Development

© 2020 by Innovation In Software Corporation

WHY MICROSERVICES?

The driving force behind microservices is to get the highest quality software possible delivered to customers as fast as possible, at web scale.

© 2020 by Innovation In Software Corporation

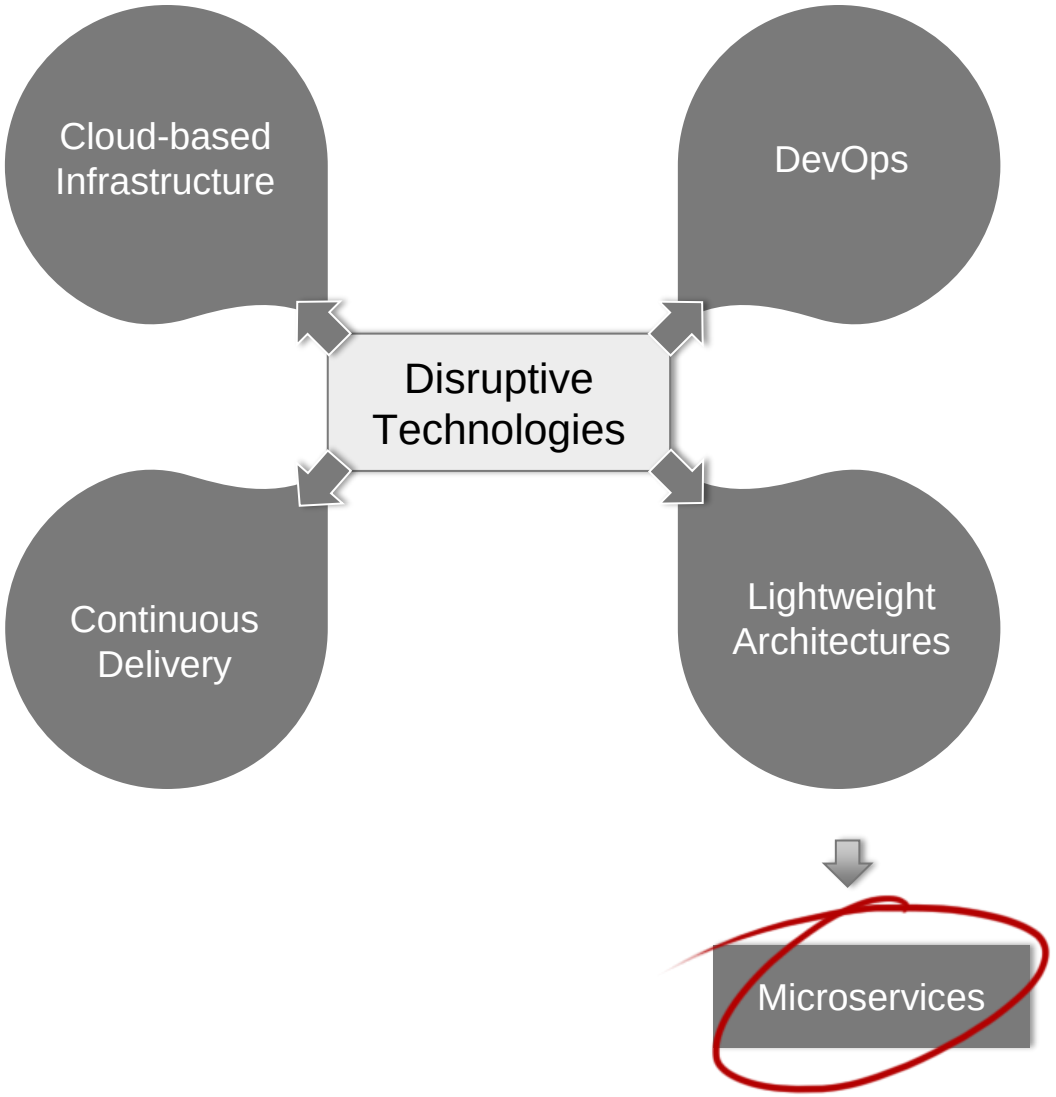
WHY MICROSERVICES?

As new disruptive technologies are introduced, it is critical for DevSecOps teams to evaluate their potential and facilitate experimentation together with the business.

These new disruptive technologies, force IT into a dramatic change and are designed to drive time to market and value for our business teams.

Enterprises who ignore them, will soon or later fall short and loose market share for their competitors.

When combined with Cloud-based Infrastructure, DevOps and Continuous Delivery, Microservices, as a lightweight Architecture approach, enables rapid Enterprise IT transformation and innovation.



© 2020 by Innovation In Software Corporation

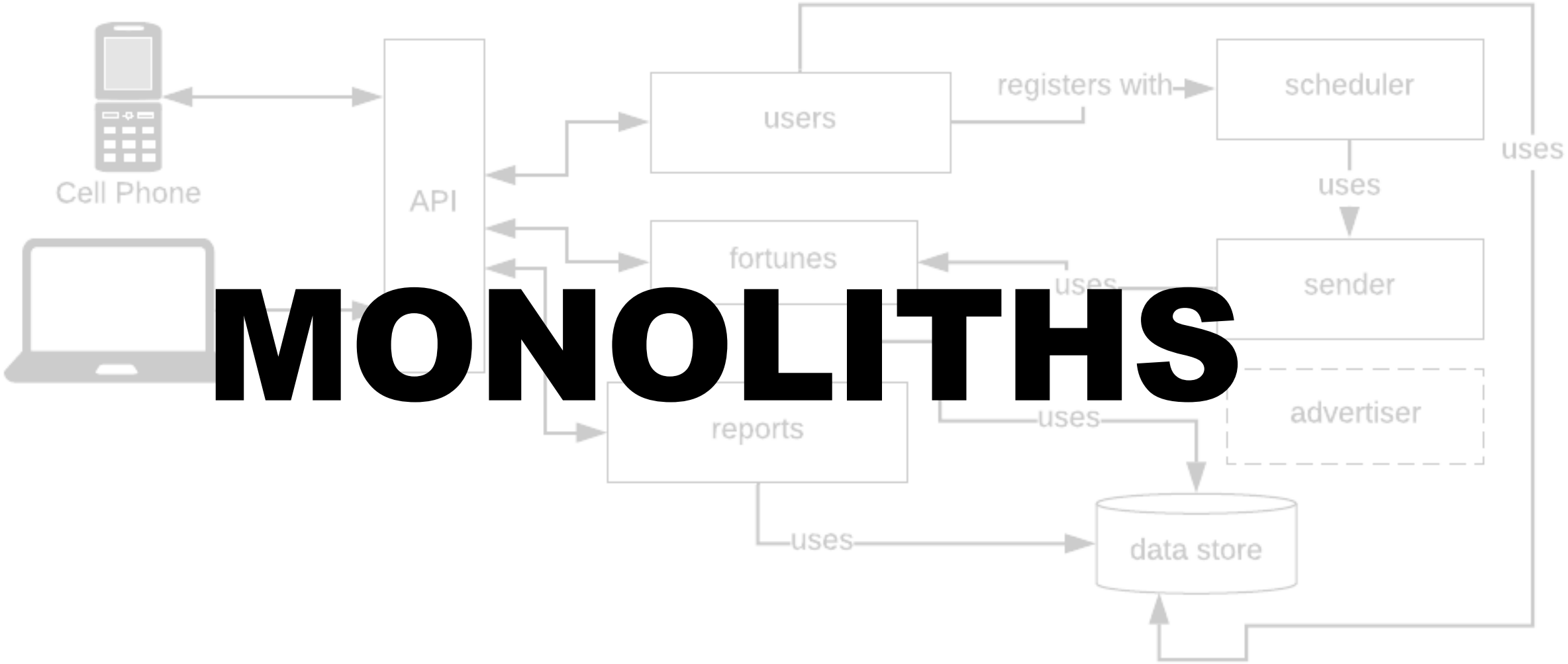
WHY MICROSERVICES ARCHITECTURE?

Adopting this architecture increases the potential benefits of complex IT initiatives

Agility	**Agility** allows organizations to deliver new products, functions, and features **quicker**
Reduce time to market	Independent components move new features into production quicker and provide **more flexibility** for piloting and prototyping, thus **reducing time to market**
Composability	**Composability** reduces development time, provides a compound benefit through reusability over time, and guarantees **system scalability**
Comprehensibility	**Comprehensibility** of the software system **simplifies development planning**, increases accuracy, and **accelerates new resource integration**
Polyglotism	**Polyglotism** permits the use of the **right tools for the right task**, thus accelerating technology introduction and increasing solution options
Cost reduction	**Reduce cost** at operating speed by reducing the overall cost of designing, implementing, and deploying services

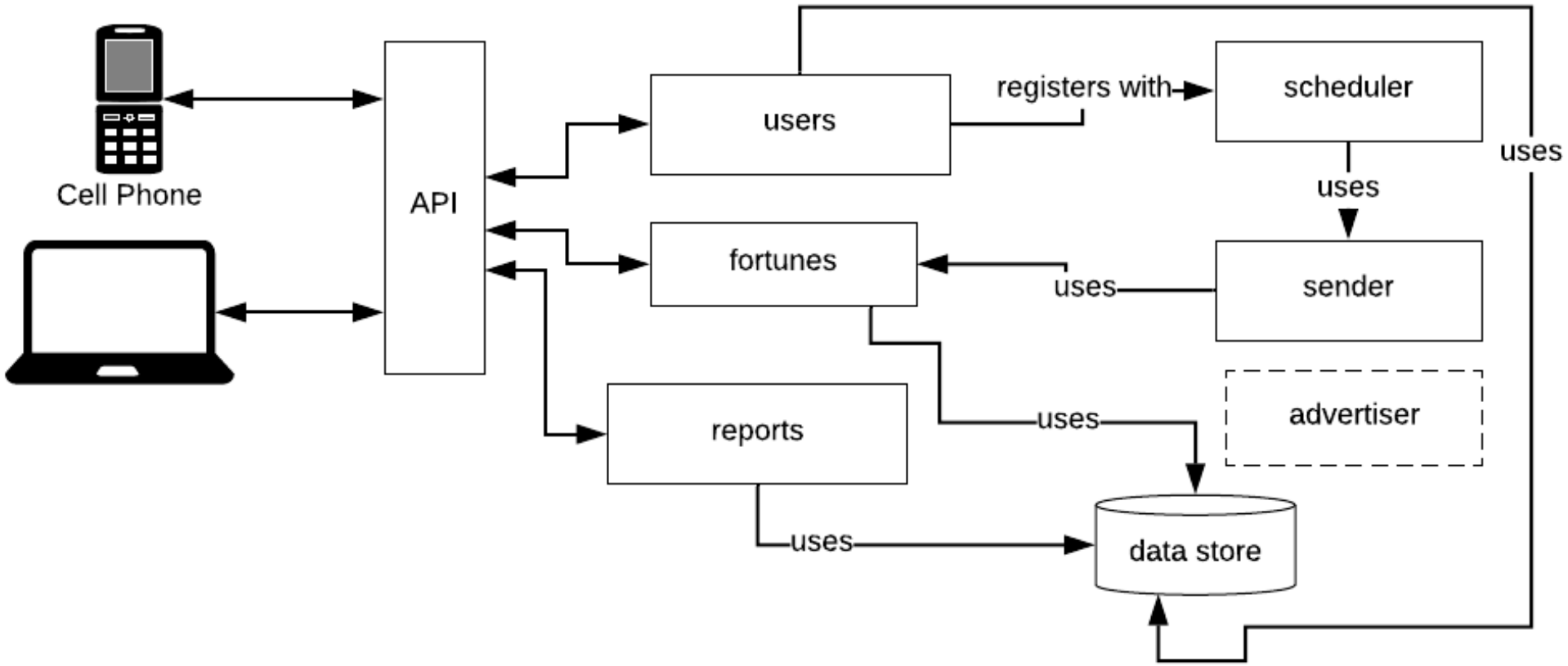
© 2020 by Innovation In Software Corporation

<<WISESAYINGS>>
1 PICTURE IS WORTH 1000 WORDS



MONOLITHS

© 2020 by Innovation In Software Corporation



MONOLITHIC OVERVIEW

Traditional Application Architectures

© 2020 by Innovation In Software Corporation

MONOLITHIC OVERVIEW

There is no single definition of a monolithic application

- A single-tiered software application.
- User interface access, business logic, and data access are combined into a single program from a single platform.
- Self-contained, and independent from other computing applications.

Each of these types of applications have some degree of a monolithic nature

- Enterprise Applications
- Multi-Tier Applications
- Java EE Applications
- SOA Architectures
- Microservices

© 2020 by Innovation In Software Corporation

PROS & CONS OF MONOLITHIC ARCHITECTURES

© 2020 by Innovation In Software Corporation

PROS & CONS OF MONOLITHIC ARCHITECTURES

Pro's

- Easy to develop
- Easy to understand
- Easy to deploy
- Easy to monitor
- Easier to manage transactional integrity
- Single release schedule

Con's

- Difficult to maintain
- Difficult to refactor
- Difficult to scale
- Difficult / impossible to understand
- Requires large team
- Single release schedule

© 2020 by Innovation In Software Corporation

MONOLITHIC CHALLENGES

Lack of Agility

- Small code changes require extensive packaging and testing.
- Slow to deliver features and enhancements.
- Difficult integration with new technology, processes (Mobile, Cloud, Continuous DevOps)
- Monoliths were designed assuming infrequent deployment.

© 2020 by Innovation In Software Corporation

LAB: CONTAINERIZING WISE-SAYINGS

https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/01-wisesayings

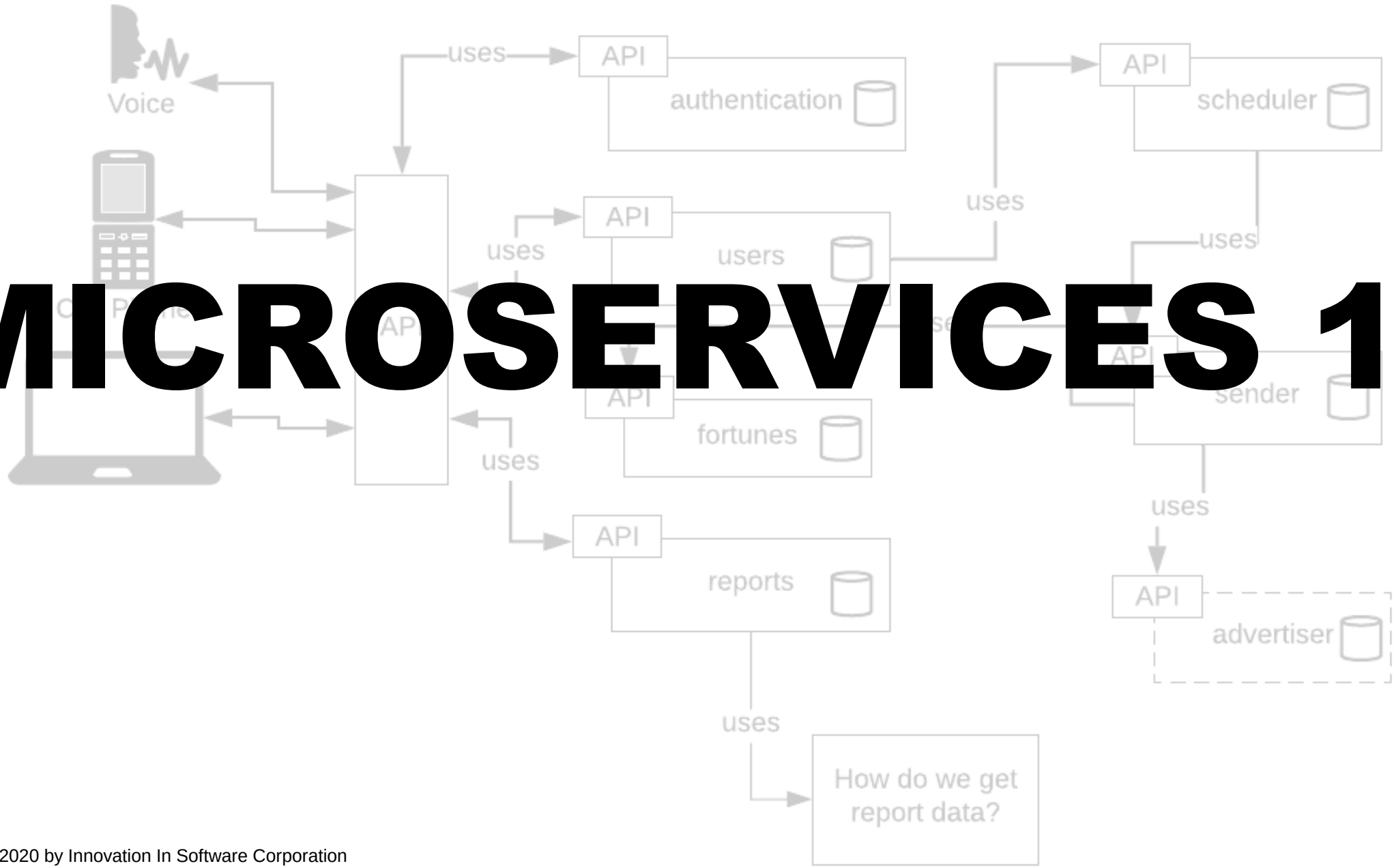
© 2020 by Innovation In Software Corporation

ARCHITECTURAL ALIGNMENT

- **Architecture Reference Model** (what do we intend to use as the building blocks, GitHub, Java, Python, Kubernetes, GKE, ECR, …)
- **Solution Community** – informal group of architects, technical leads from your organization and partners to get all brains in the game
- **Kaizen Skills Matrix** – documented technical capabilities we need to support and enhance the solutions we have now, and in the future
- **Technical Capability Roadmap** – acquisition model for identified required skills and capabilities
- **Knowledge Management** – knowledge management model for our existing and future architecture and application reference models
- **Process Alignment** – Bring technical reference model, capabilities matrix and “all brains in the game” earlier and across the project workflow AKA Agile

© 2019 by Metropolitan Airport Commission IT

MICROSERVICES 101



© 2020 by Innovation In Software Corporation

MICROSERVICES 101

- Microservices are a set of small, independent, composable services.
- Each service can be accessed by way of a well-known request-response approach common to web API, such as REST, GraphQL, gRPC or in response to event notifications.
- This architectural style de-constructs business processes to their most basic level by creating small, separate processes that then replace large, single applications.

© 2020 by Innovation In Software Corporation

MICROSERVICE OVERVIEW

“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.”

- Martin Fowler

© 2020 by Innovation In Software Corporation

MICROSERVICE OVERVIEW (CONT'D)

- Microservices are a set of small, independent, composable services.
- Each service can be accessed by way of a well-known API format such as REST, GraphQL, gRPC or in response to event notifications.
- This architectural style de-constructs business processes to their most basic level by creating small, separate processes that then replace large, single applications.

© 2020 by Innovation In Software Corporation

MICROSERVICE OVERVIEW (CONT'D)

- Breaking a system into smaller parts allows each service to focus on a single business capability.
- Microservices do not live alone.
- They are a part of the larger system.

© 2020 by Innovation In Software Corporation

MICROSERVICE OVERVIEW (CONT'D)

- They work alongside other services to accomplish the tasks normally handled by one large standalone monolith application.
- They communicate with each other synchronously or asynchronously.
- Microservices make applications easier to develop and maintain.
- BUT A LOT HARDER TO MANAGE

© 2020 by Innovation In Software Corporation

CHARACTERISTICS OF MICROSERVICES

- **Decentralized governance and data management.**
- **Automated and designed for failure.**
- **Infrastructure can be built, deployed and verified with CI/CD automation.**

© 2020 by Innovation In Software Corporation

BUSINESS MOTIVATIONS

Business Agility

- Speed of change is faster with a more modular architecture.
- React quicker to feature and enhancement requirements.
- Easy integration with new technology, processes (Mobile, Cloud, Continuous DevOps)

© 2020 by Innovation In Software Corporation

BUSINESS MOTIVATIONS

Accelerate Digital Transformation

- Provide more flexibility through capability reuse.
- Helps reduce technical debt.

© 2020 by Innovation In Software Corporation

BUSINESS MOTIVATIONS

Facilitates fast on-boarding of new personnel provided the development teams have embraced the MOA philosophy

- Reduce learning curve by providing a more understandable architecture.
- Improves developer velocity.

© 2020 by Innovation In Software Corporation

BUSINESS MOTIVATIONS

Enable Polyglot Rewrite (e.g. Java, Node, .NET, etc.)

- Utilize the best technology choice for each service offering.
- Incremental approaches centered on smaller components radically reduces risk.

© 2020 by Innovation In Software Corporation

PROS & CONS

Pro's

- Easy to develop
- Easy to understand
- Easy to deploy
- Easy to monitor each service
- Flexible Release Schedule
- Use standard APIs (JSON, XML)

Provided the development teams have embraced the MOA philosophy

Con's

- Requires more deployments
- Requires translation (JSON, XML)
- Requires more monitoring
- Operations configuration more complex
- System can be very complex

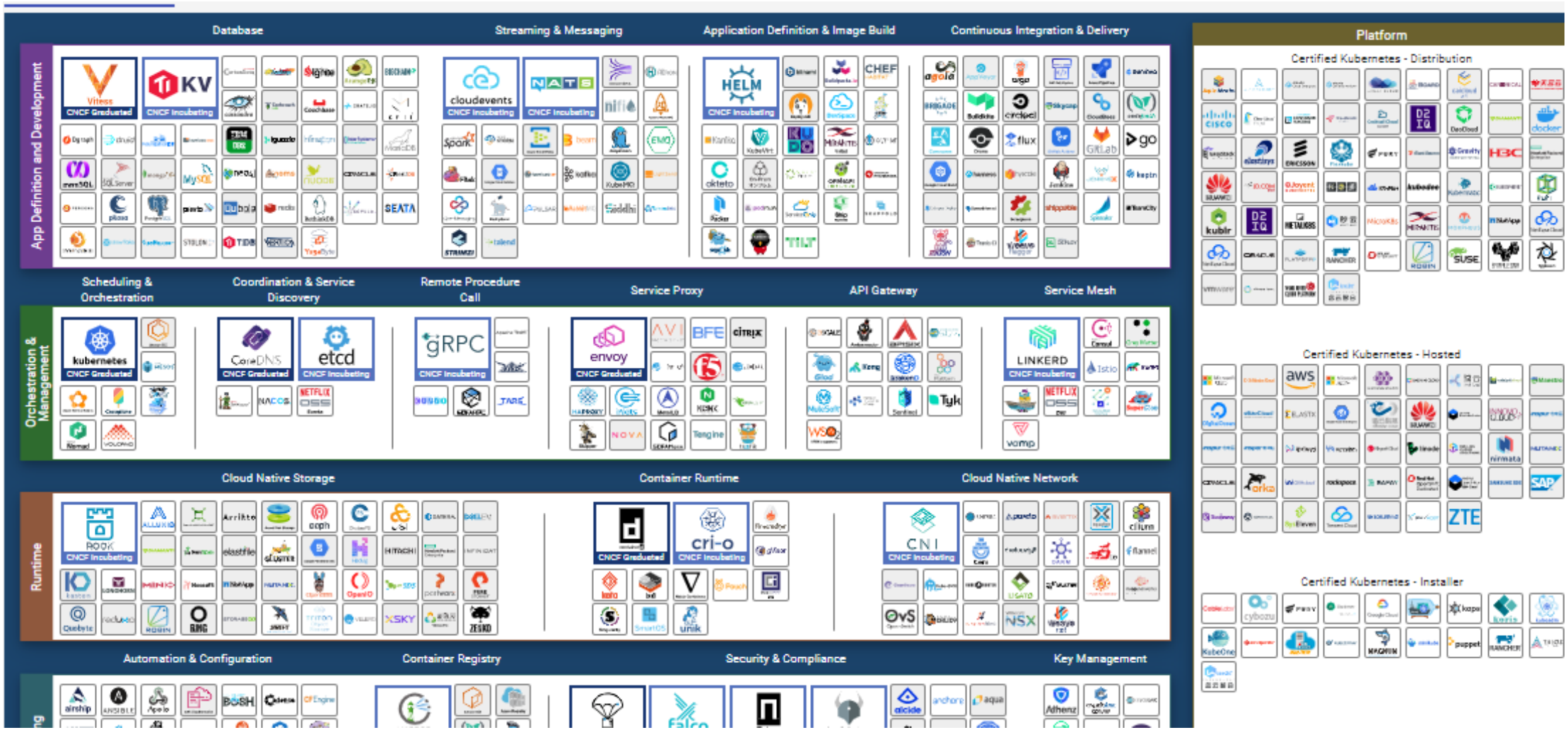
© 2020 by Innovation In Software Corporation

5 KEY CONSIDERATIONS

- What is the semantic definition of the microservice?
- What is the microservice’s SLA?
- How does the microservice expose itself?
- Microservices carry their own data
- Synchronous vs. Asynchronous Interaction

© 2020 by Innovation In Software Corporation

CNCF LANDSCAPE



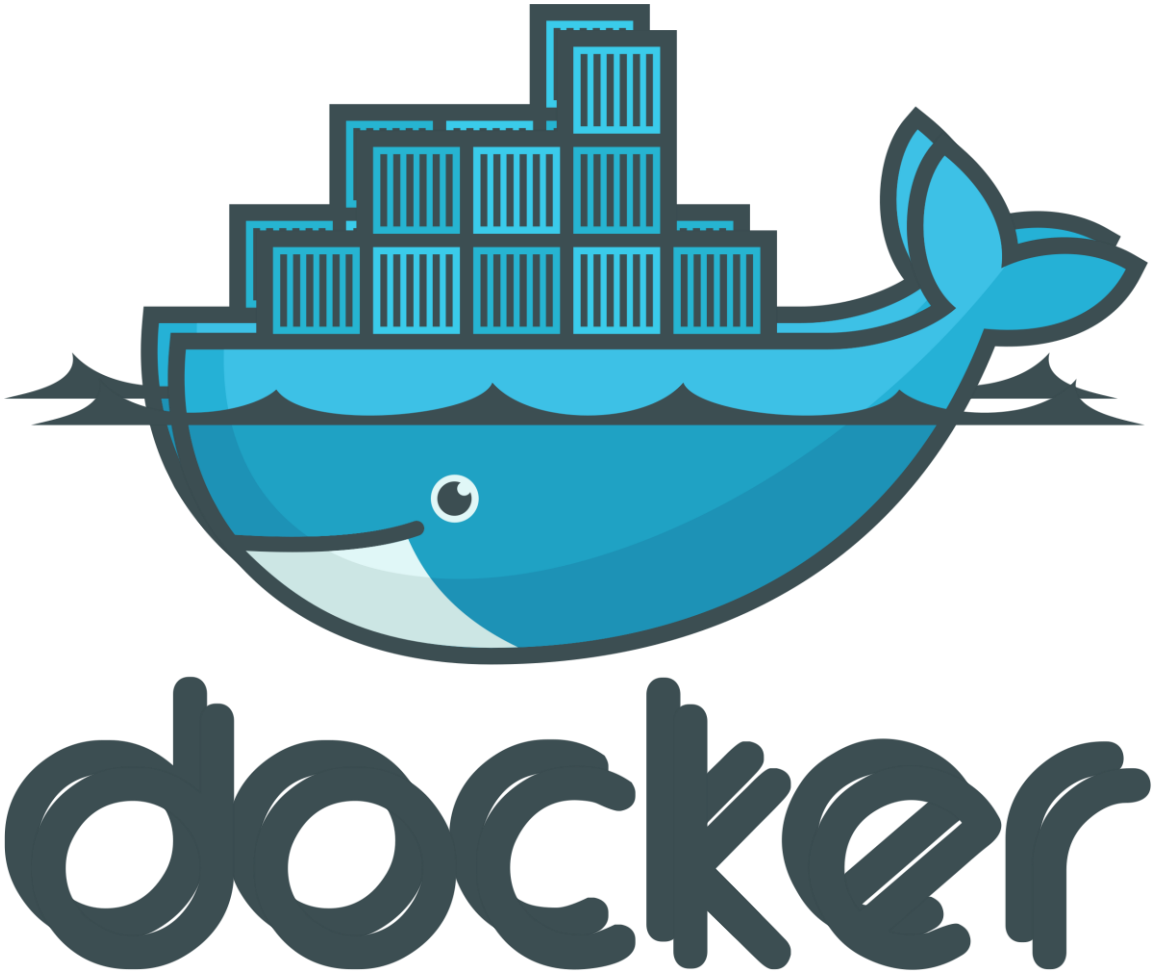
© 2020 by Innovation In Software Corporation

DOCKER 101

© 2020 by Innovation In Software Corporation

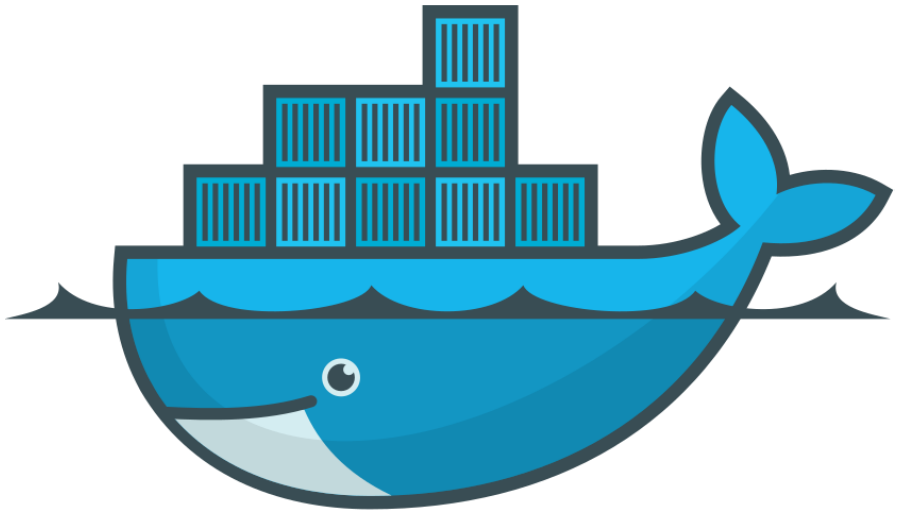
DOCKER

Build, Ship, and Run Any App, Anywhere.



© 2020 by Innovation In Software Corporation

DOCKER



- Get more applications running on the same hardware than other technologies.
- Easy for developers to quickly create ready-to-run containerized applications.
- Easier to manage and deploy applications.

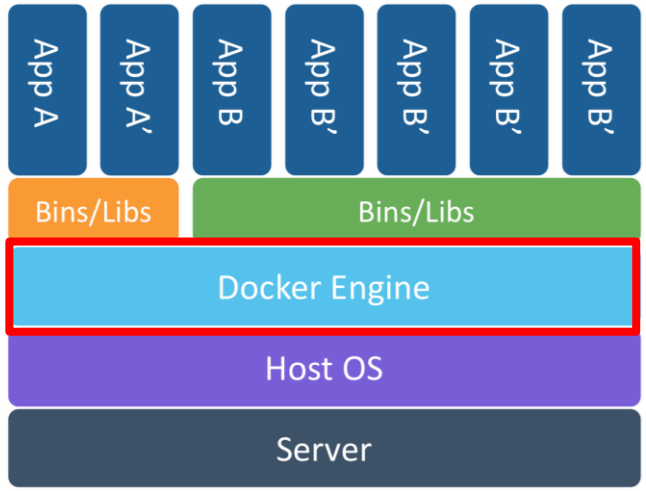
© 2020 by Innovation In Software Corporation

DOCKER ENGINE

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Lightweight runtime program to ***build, ship, and run Docker containers***

- **Also known as Docker Daemon**
- **Uses Linux Kernel namespaces and control groups**
  - **Linux Kernel (>= 3.10)**
- **Namespaces provide an isolated workspace**

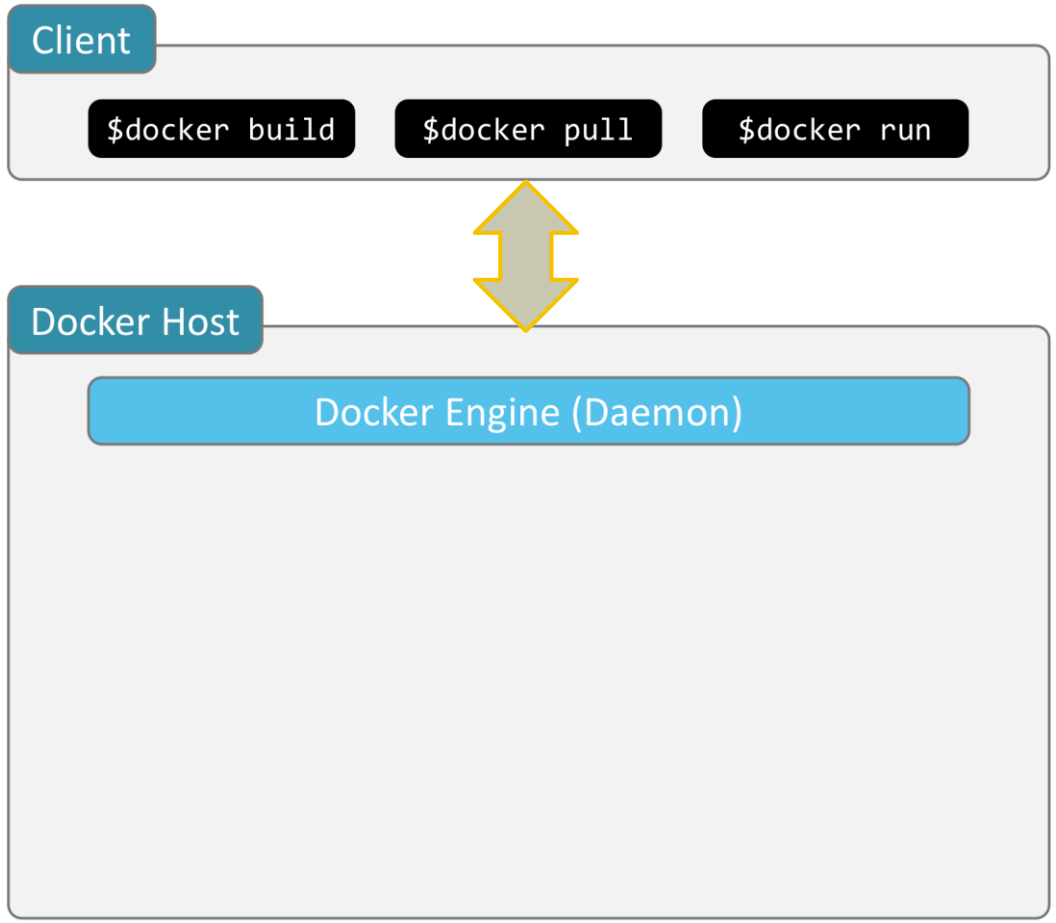


© 2020 by Innovation In Software Corporation

DOCKER ENGINE

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

- **The Docker Client is the `docker` binary**
  - Primary interface to the Docker Host
- **Accepts commands and communicates with the Docker Engine (Daemon)**

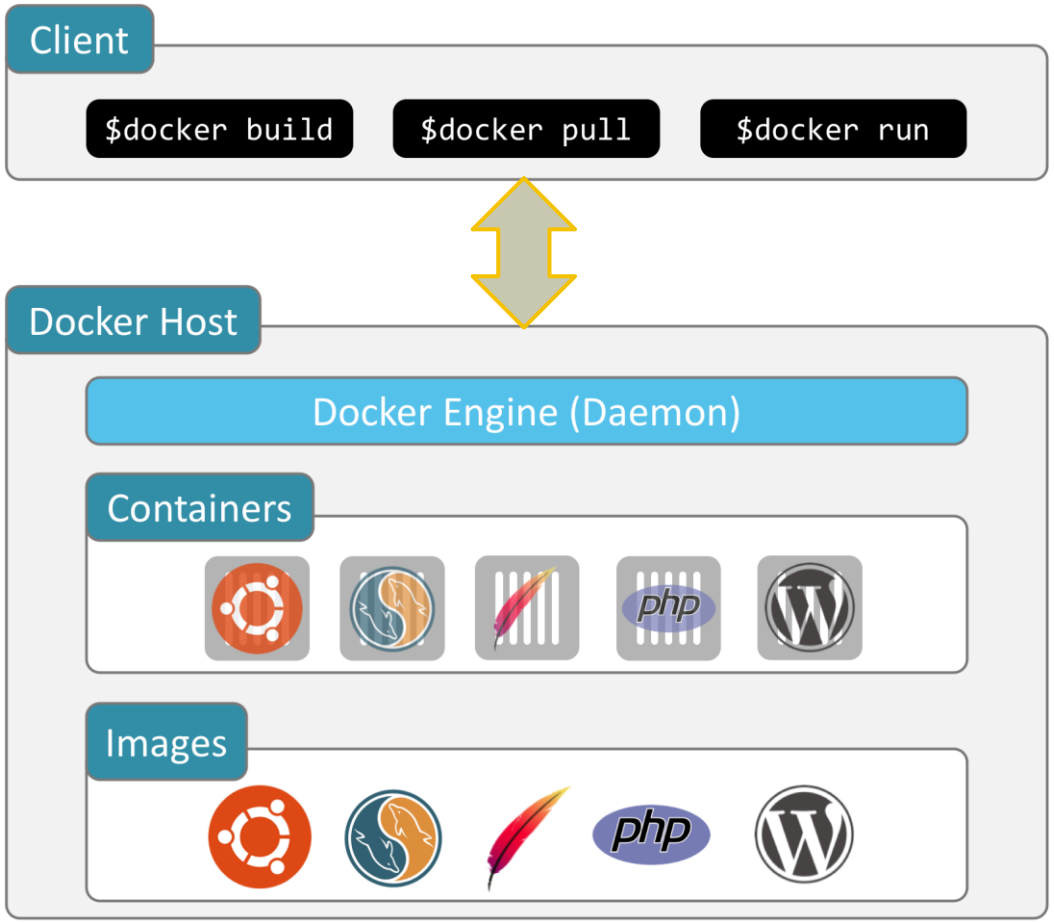


© 2020 by Innovation In Software Corporation

DOCKER ENGINE

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

- **Lives on a Docker host**
- **Creates and manages containers on the host**



© 2020 by Innovation In Software Corporation

DOCKER REGISTRY

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Image Storage & Retrieval System

- **Docker Engine Pushes Images to a Registry**
- **Version Control**
- **Docker Engine Pulls Images to Run**

JFrog Artifactory

tutum

QUAY by CoreOS

Nexus

Official Repositories

MySQL mongoDB PostgreSQL

RAILS Ruby Java WordPress

redis NGINX node

ubuntu debian CentOS

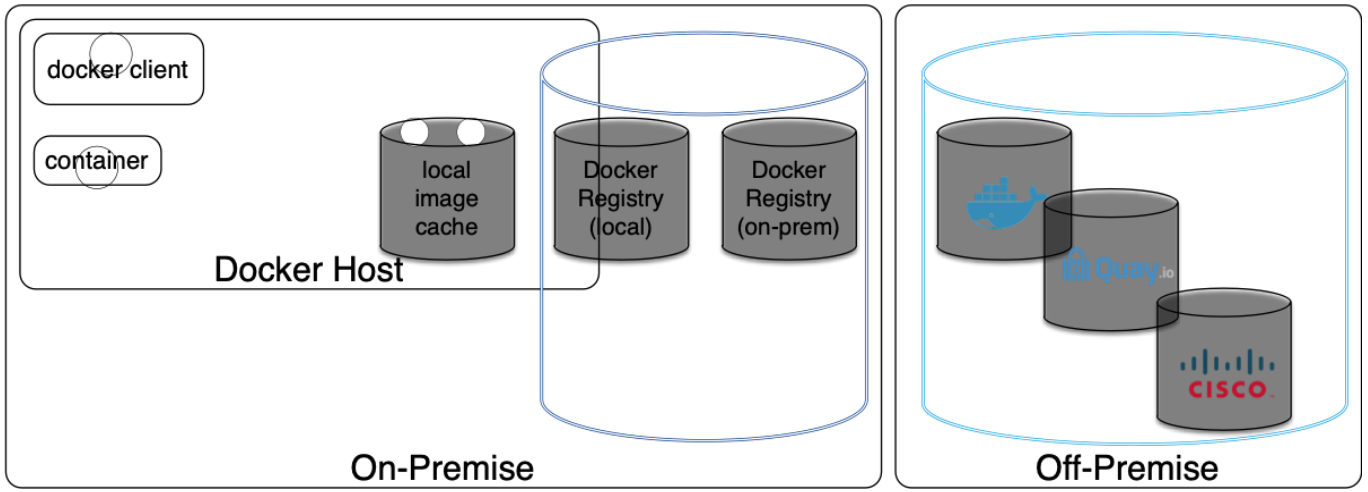
© 2020 by Innovation In Software Corporation

DOCKER REGISTRY

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Types of Docker Registries

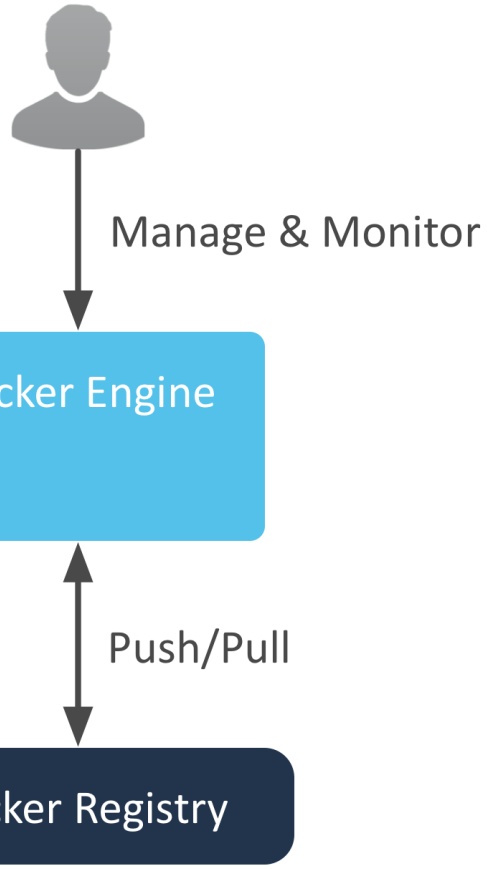
- Local Docker Registry (On Docker Host)
- Remote Docker Registry (On-Premise/Off-Premise)
- Docker Hub (Off-Premise)



© 2020 by Innovation In Software Corporation

DOCKER ENGINE AND REGISTRY

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm



Docker Engine:

- **Pushes** Images to a Registry
- **Pulls** Images to Run

© 2020 by Innovation In Software Corporation

DOCKER REGISTRY

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

The registry and engine both present APIs

- All of Docker's functionality will utilize these APIs

- RESTFUL API
- Commands presented with Docker's CLI tools can also be used with curl and other tools

© 2020 by Innovation In Software Corporation

DOCKER AT THE COMMAND LINE

```
docker build -t GeorgeNiece/wisesayings:beta .

docker run -d -p 3000:3000 georgeniece/wisesayings:beta

docker ps -a

curl localhost:3000
```

© 2020 by Innovation In Software Corporation

LAB: CONTAINERIZING WISE-SAYINGS

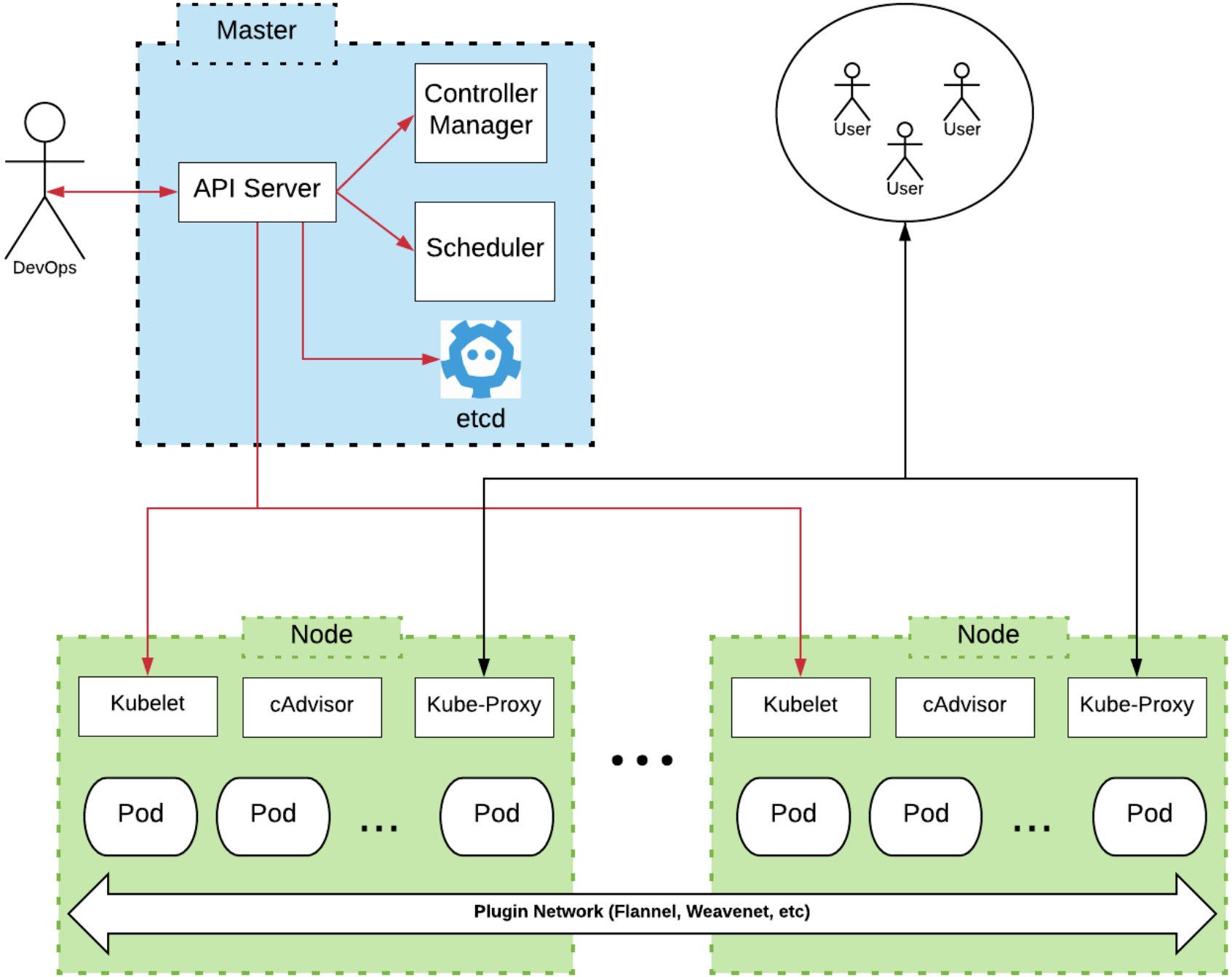
https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/01-wisesayings

© 2020 by Innovation In Software Corporation

KUBERNETES 101

© 2020 by Innovation In Software Corporation

KUBERNETES



© 2020 by Innovation In Software Corporation

KUBERNETES - CONTROL PLANE

etcd

etcd is a persistent, lightweight, distributed, key-value data store developed by CoreOS that reliably stores the configuration data of the cluster, representing the overall state of the cluster at any given point of time. Other components watch for changes to this store to bring themselves into the desired state

API Server

The API server is a key component and serves the Kubernetes API using JSON over HTTP, which provides both the internal and external interface to Kubernetes.The API server processes and validates REST requests and updates state of the API objects in etcd, thereby allowing clients to configure workloads and containers across Worker nodes

Scheduler

The scheduler is the pluggable component that selects which node an unscheduled pod (the basic entity managed by the scheduler) runs on, based on resource availability. Scheduler tracks resource use on each node to ensure that workload is not scheduled in excess of available resources. For this purpose, the scheduler must know the resource requirements, resource availability, and other user-provided constraints and policy directives such as quality-of-service, affinity/anti-affinity requirements, data locality, and so on. In essence, the scheduler’s role is to match resource "supply" to workload "demand"

Controller Manager

The controller manager is a process that runs core Kubernetes controllers like DaemonSet Controller and Replication Controller. The controllers communicate with the API server to create, update, and delete the resources they manage (pods, service endpoints, etc.)

© 2020 by Innovation In Software Corporation

KUBERNETES - NODES

Kubelet

Kubelet is responsible for the running state of each node, ensuring that all containers on the node are healthy. It takes care of starting, stopping, and maintaining application containers organized into pods as directed by the control plane. Kubelet monitors the state of a pod, and if not in the desired state, the pod re-deploys to the same node. Node status is relayed every few seconds via heartbeat messages to the master. Once the master detects a node failure, the Replication Controller observes this state change and launches pods on other healthy nodes

Container

A container resides inside a pod. The container is the lowest level of a micro-service that holds the running application, libraries, and their dependencies. Containers can be exposed to the world through an external IP address.

Kube-proxy

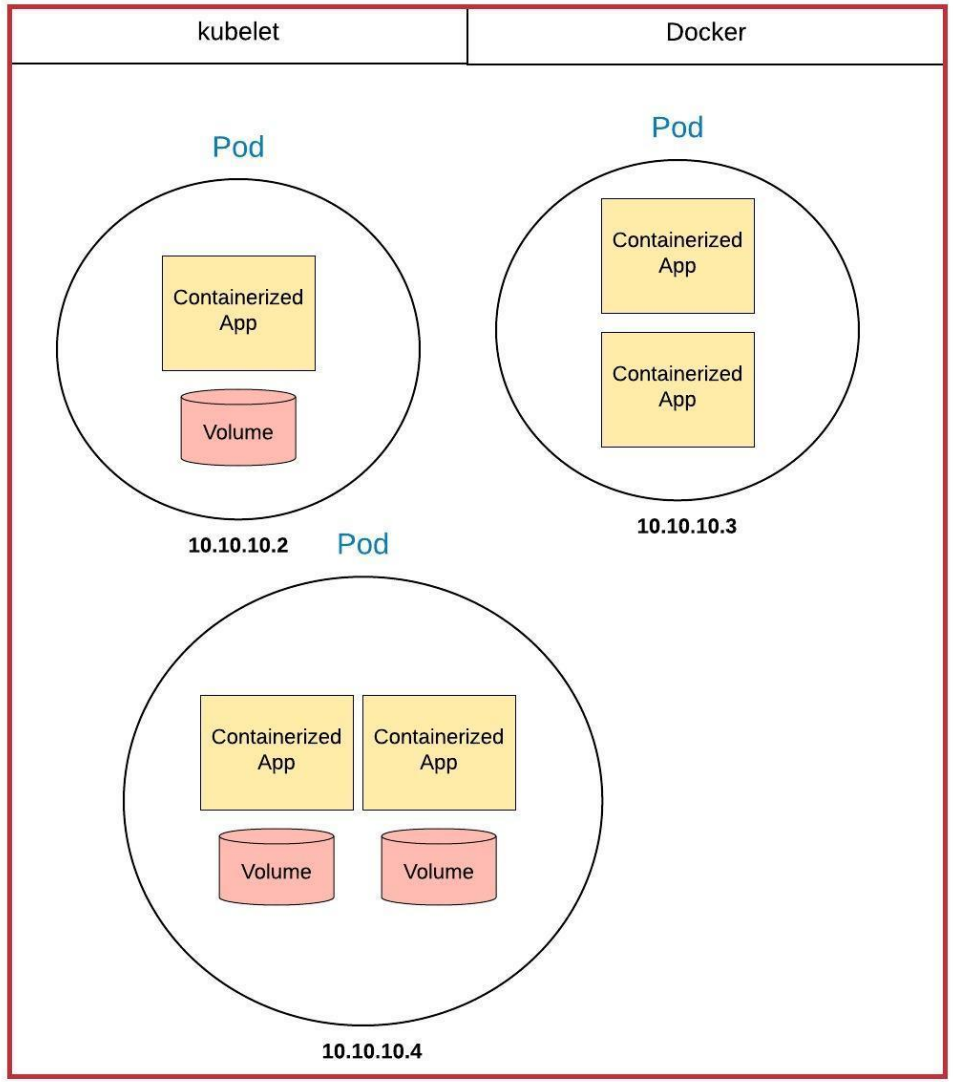
The Kube-proxy is an implementation of a network proxy and a load balancer, and it supports the service abstraction along with other networking operation. It is responsible for routing traffic to the appropriate container based on IP and port number of the incoming request.

cAdvisor

cAdvisor is an agent that monitors and gathers resource usage and performance metrics such as CPU, memory, file and network usage of containers on each node.

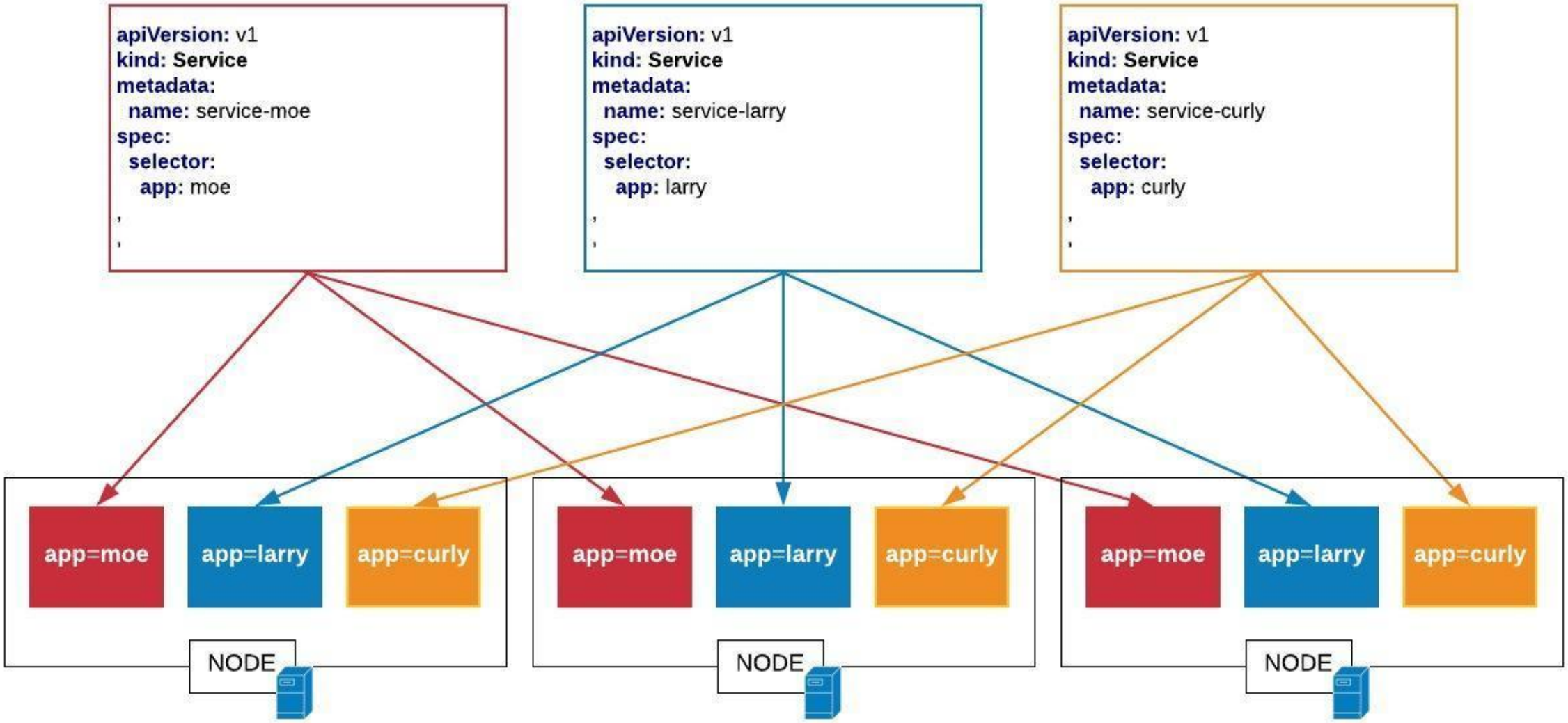
© 2020 by Innovation In Software Corporation

KUBERNETES - UNDERSTANDING PODS AND DEPLOYMENTS



© 2020 by Innovation In Software Corporation

KUBERNETES - UNDERSTANDING SERVICES



© 2020 by Innovation In Software Corporation

KUBERNETES MANIFEST: POD

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: scheduler
  labels:
    app: scheduler
spec:
  containers:
  - name: scheduler
    image: localhost:5000/scheduler
    imagePullPolicy: Always
    ports:
      - name: app
        containerPort: 3000
    env:
      - name: SENDER_API_URL
        value: "http://sender/"
```

© 2020 by Innovation In Software Corporation

KUBERNETES MANIFEST: DEPLOYMENT

```yaml
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: wisesayings
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: wisesayings
    spec:
      containers:
      - name: wisesayings
        image: georgeniece/wisesayings:v0.2
        imagePullPolicy: Always
        ports:
        - containerPort: 3000
```

© 2020 by Innovation In Software Corporation

KUBERNETES MANIFEST: SERVICE

```yaml
apiVersion: v1
kind: Service
metadata:
  name: wisesayings
spec:
  selector:
    app: wisesayings
  ports:
  -
    protocol: TCP
    port: 3000
    targetPort: 3000
  type: NodePort
```

© 2020 by Innovation In Software Corporation

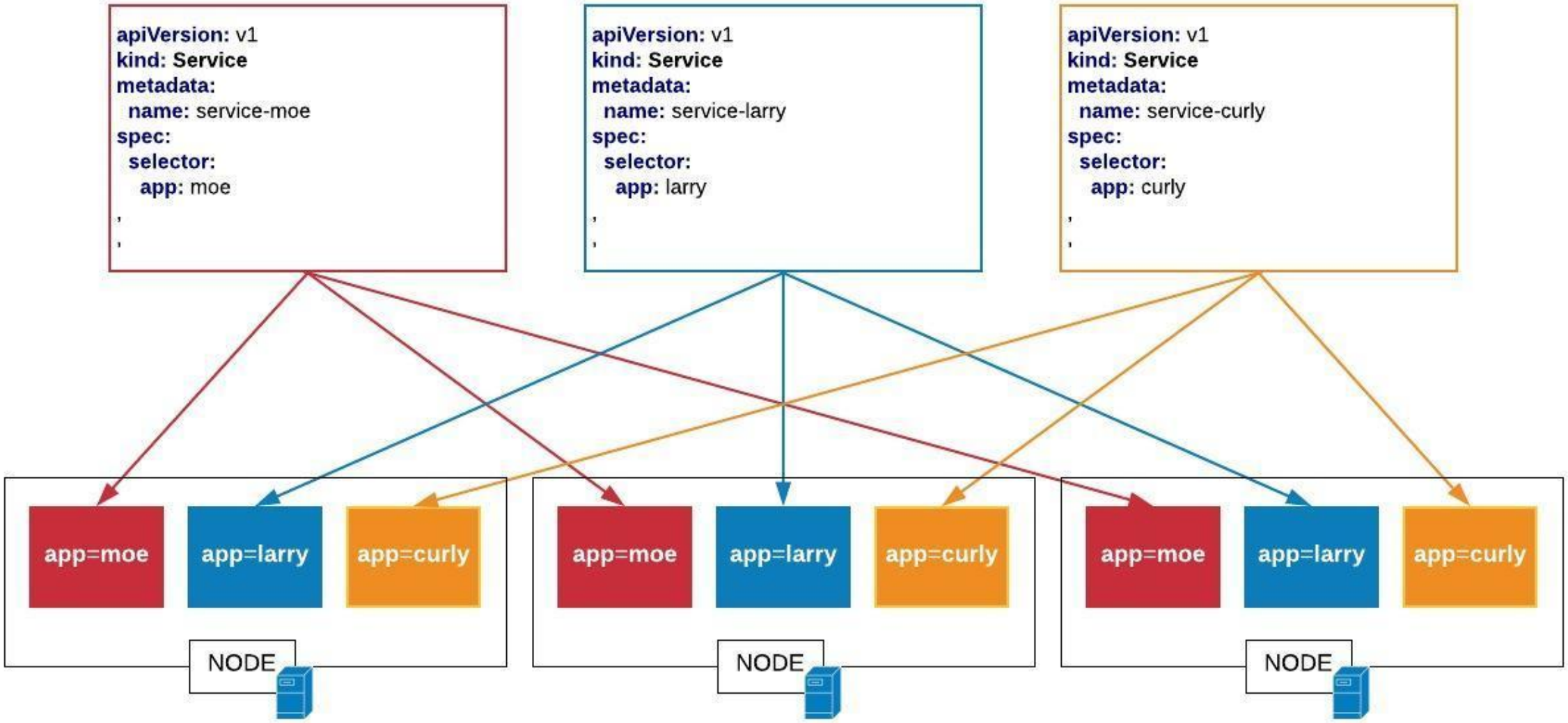
KUBERNETES DEPLOYMENT AND SERVICE

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: wisesayings
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: wisesayings
    spec:
      containers:
      - name: wisesayings
        image: georgeniece/wisesayings:v0.2
        imagePullPolicy: Always
        ports:
        - containerPort: 3000
```

```
apiVersion: v1
kind: Service
metadata:
  name: wisesayings
spec:
  selector:
    app: wisesayings
  ports:
  -
    protocol: TCP
    port: 3000
    targetPort: 3000
  type: NodePort
```

© 2020 by Innovation In Software Corporation

KUBERNETES - UNDERSTANDING SERVICES



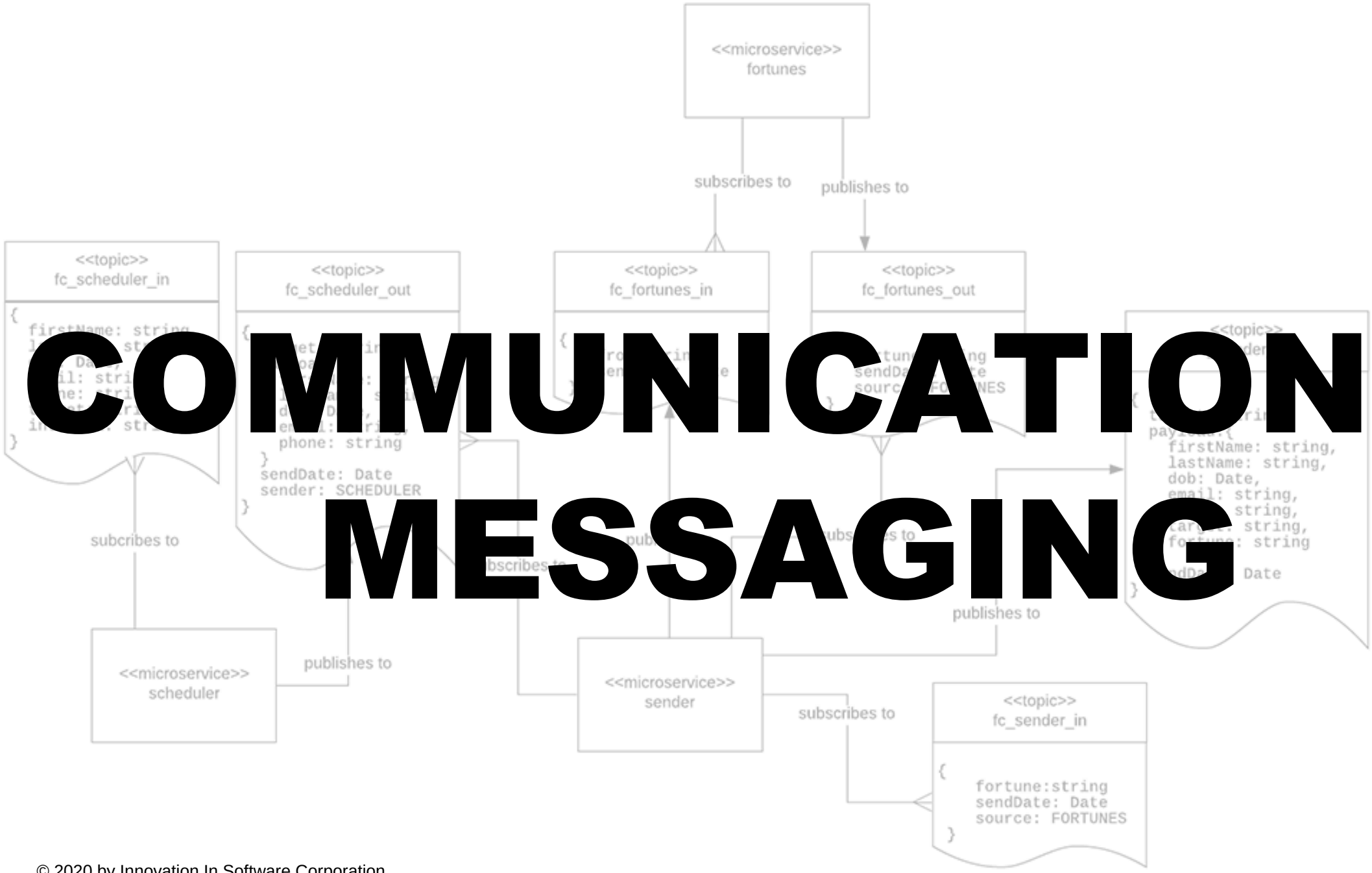
© 2020 by Innovation In Software Corporation

LAB: DEPLOYING WISE-SAYINGS TO K8S

https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/01-wisesayings

© 2020 by Innovation In Software Corporation

COMMUNICATION & MESSAGING



© 2020 by Innovation In Software Corporation

COMMUNICATION STYLES

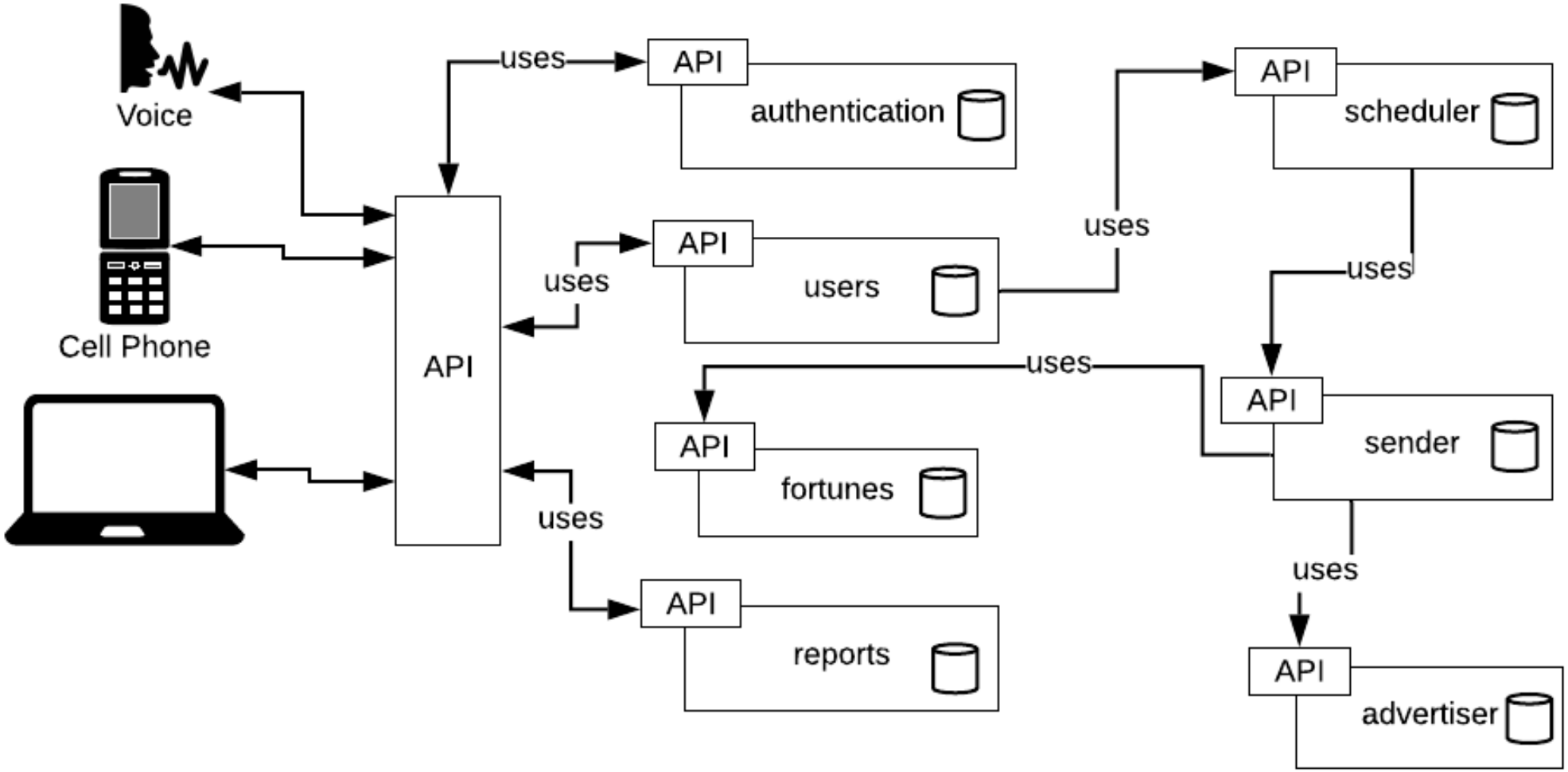
Synchronous

Asynchronous

Both

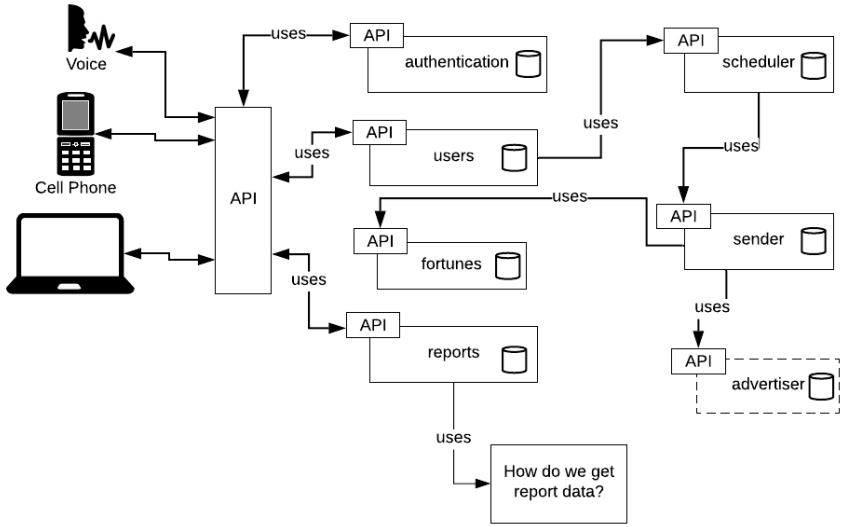
© 2020 by Innovation In Software Corporation

COMMUNICATION STYLE: SYNCHRONOUS



© 2020 by Innovation In Software Corporation

COMMUNICATION STYLE: SYNCHRONOUS



Advantages:

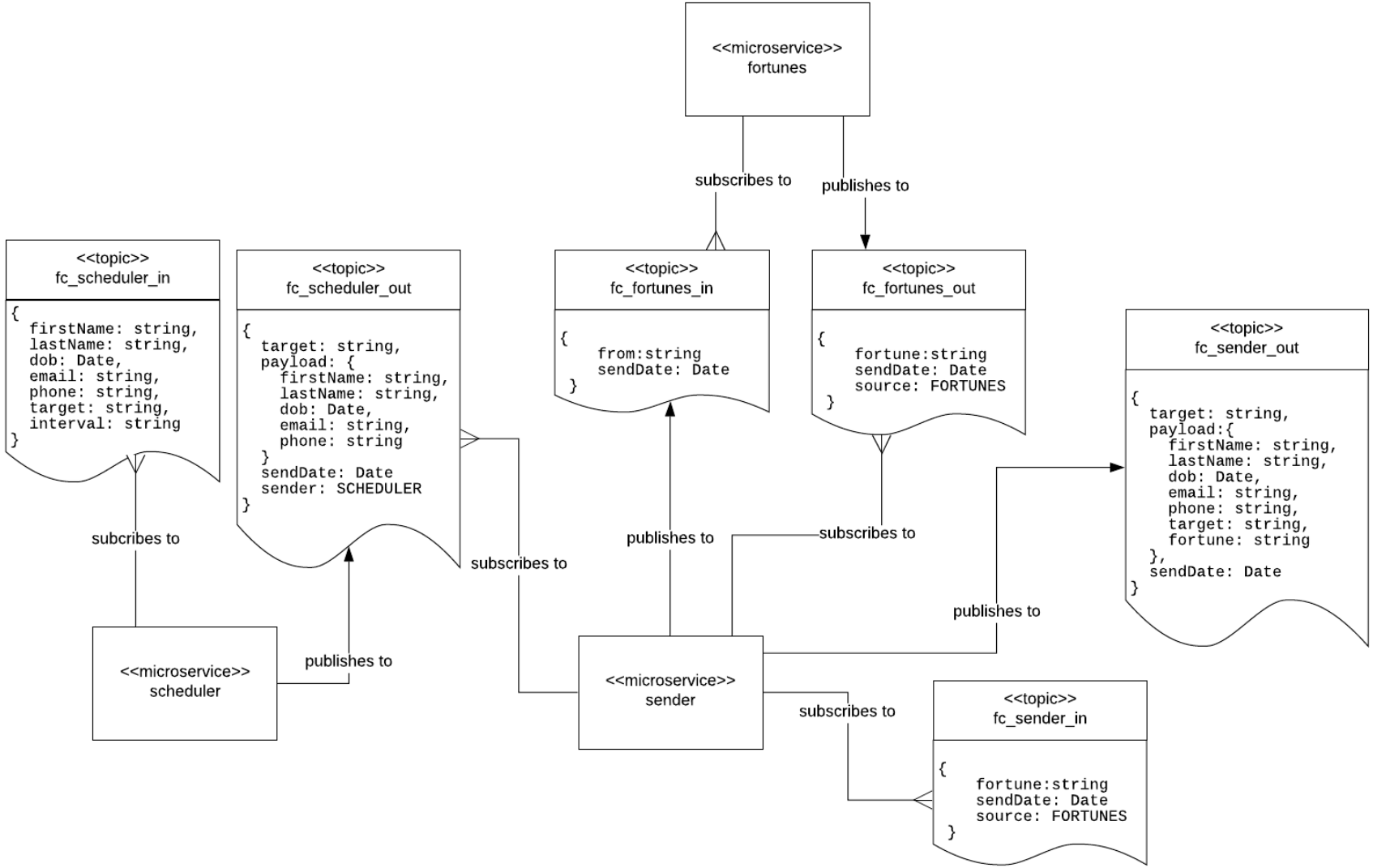
- Service is stateless (relative to caller)
- Allows many active instances
- Lower management overhead since no message server
- Immediate and direct feedback (success or failure)

Disadvantages:

- Caller has to wait, potentially limiting caller scalability
- Can add hard dependencies between services if chained

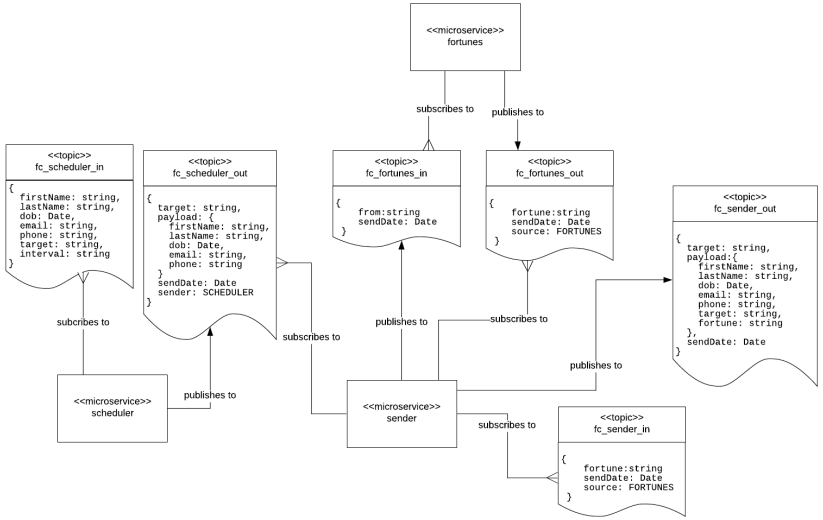
© 2020 by Innovation In Software Corporation

COMMUNICATION STYLE: ASYNCHRONOUS



© 2020 by Innovation In Software Corporation

COMMUNICATION STYLE: ASYNCHRONOUS



Advantages:

- Improves scalability since no coupling between caller and service instance
- Queueing allows for degree of inherent load balancing

Disadvantages:

- Dependent on external message server
- Additional tuning and monitoring is required

© 2020 by Innovation In Software Corporation

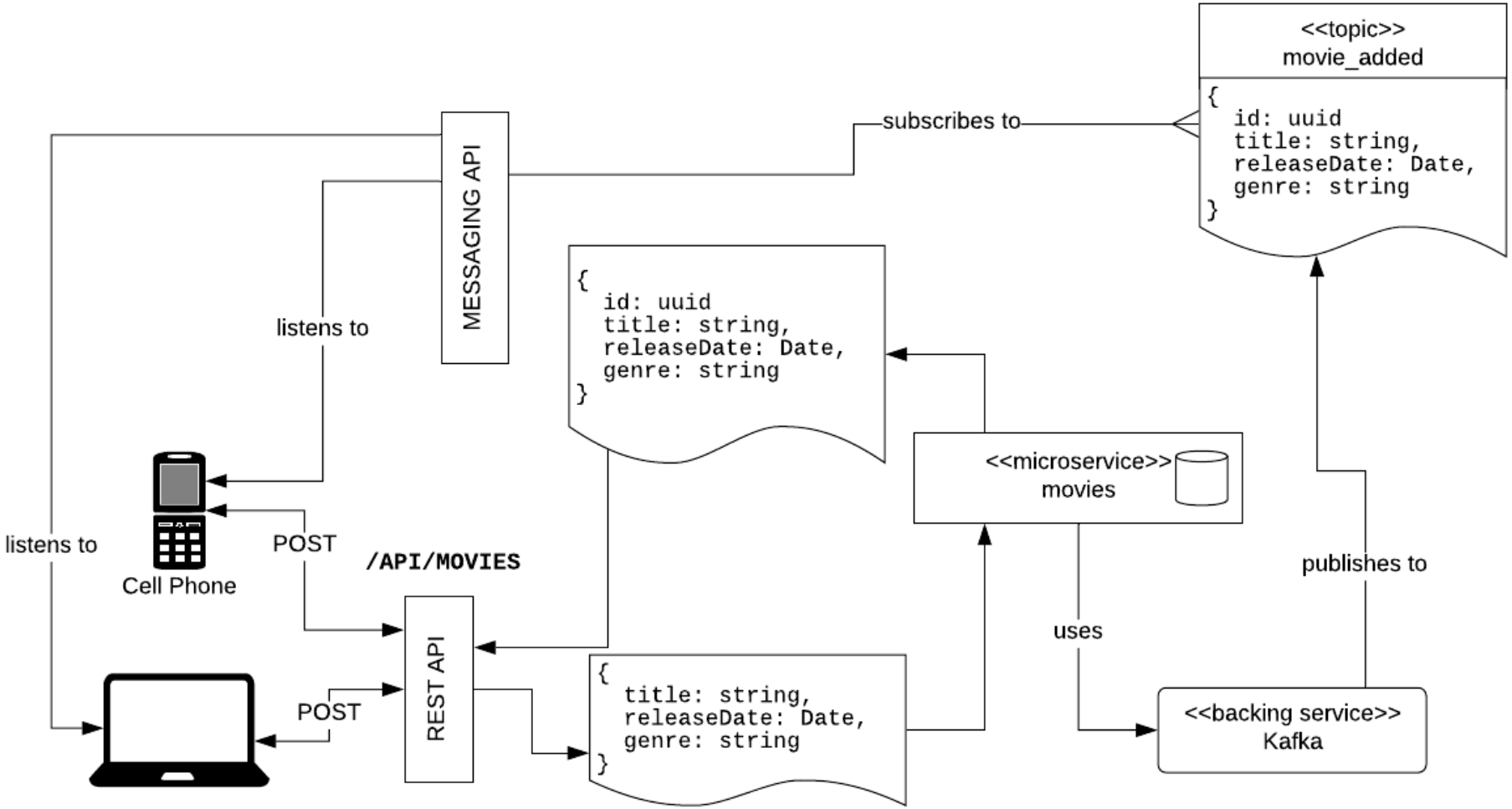
COMMUNICATION STYLE: ASYNCHRONOUS

Communication style is particularly important from the perspective of loose coupling, multi-service communication, and coping with partial server failure.

These are determining factors for inclining microservices towards Async communication. Asynchronous protocols like MQTT, STOMP, AMQP are handled by platforms like Apache Kafka, Pulsar, Amazon Kinesis, or RabbitMQ.

© 2020 by Innovation In Software Corporation

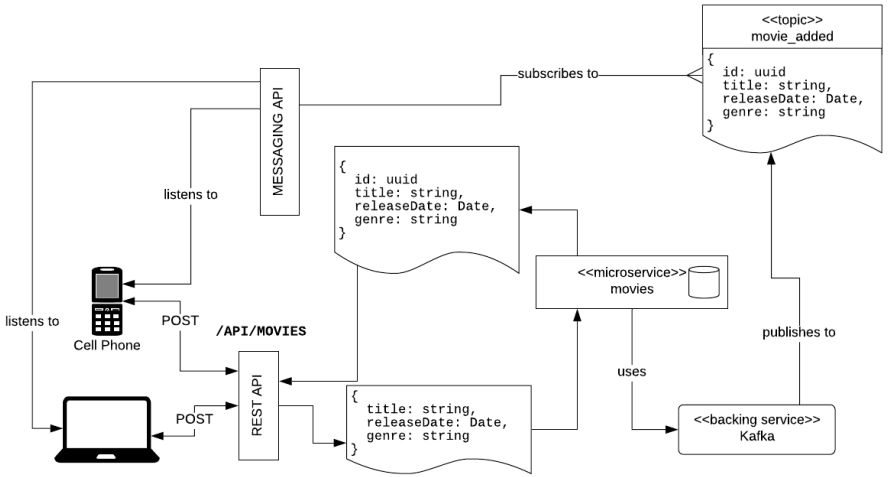
COMMUNICATION STYLE: BOTH



AKA: Lambda Messaging Pattern

© 2020 by Innovation In Software Corporation

COMMUNICATION STYLE: BOTH



Advantages:

- Best of both worlds
- Flexibility in terms of developer experience
- Easy to implement as an API

Disadvantages:

- Hard to implement as a microservices
- There’s no free lunch, you have to contend with more latency issues

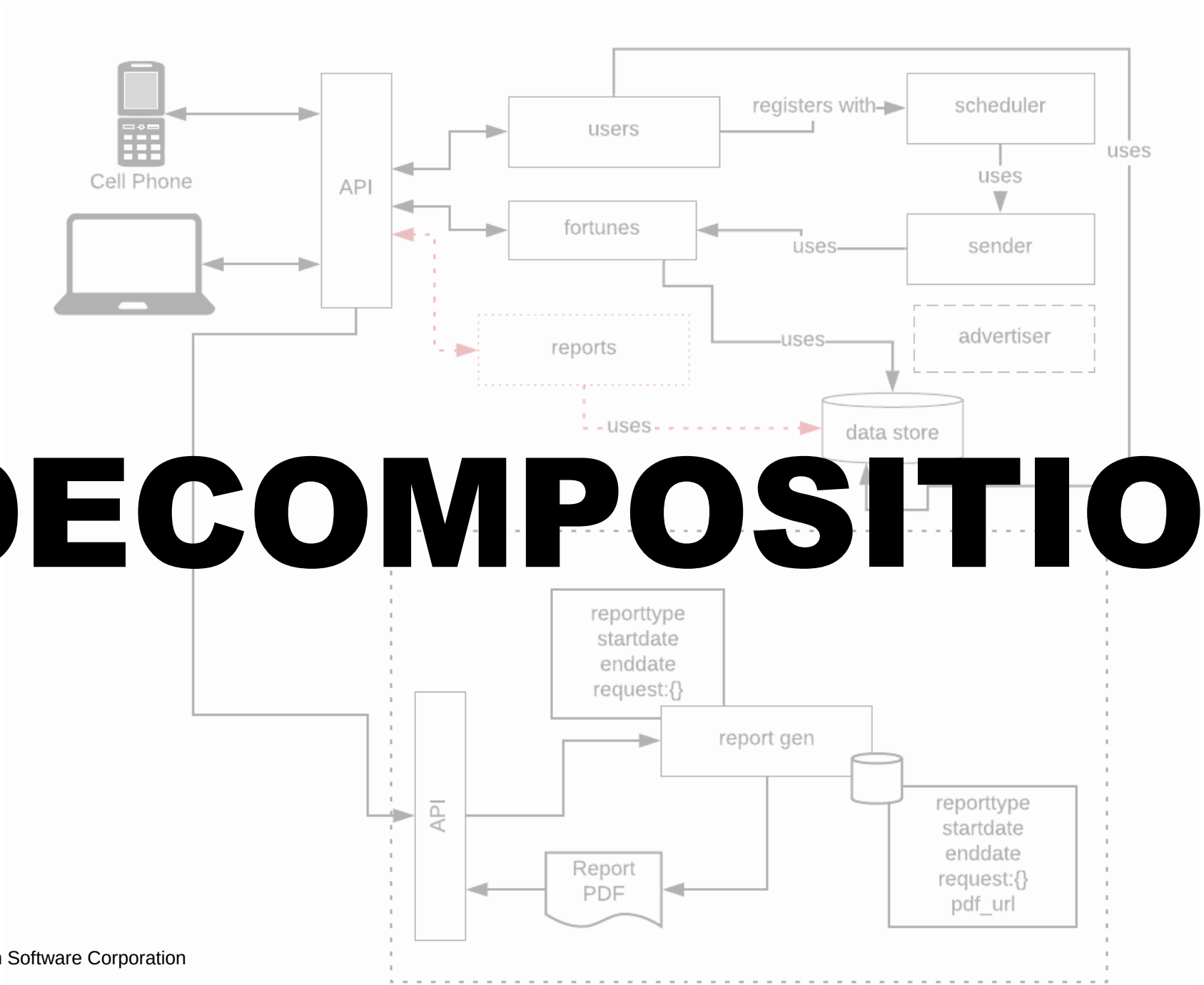
© 2020 by Innovation In Software Corporation

LAB: LAMBDA MESSAGING USING GRAPHQL

https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/02-messaging

© 2020 by Innovation In Software Corporation

DECOMPOSITION



© 2020 by Innovation In Software Corporation

PRINCIPLE: SINGLE RESPONSIBILITY

Single Responsibility Principle

- The Single Responsibility Principle states that a service should have only one reason to change
- The main responsibility of the service is considered to be one reason to change
- If there are two reasons for change, we need to split the functionality of the service into two services

Two services should not share one responsibility

- Replication of functionality becomes maintenance burden

One service should not have multiple responsibilities

- The responsibilities then become tightly coupled

© 2020 by Innovation In Software Corporation

PRINCIPLE: AUTONOMY

A microservice must be

- Self-contained
- Independently deployable
- Have full responsibility for a capability and executing that capability

Autonomy is a primary differentiator between microservices and other architectures

- Minimizes coupling and maximizes abstraction

© 2020 by Innovation In Software Corporation

POTENTIAL CANDIDATES FOR MICROSERVICES

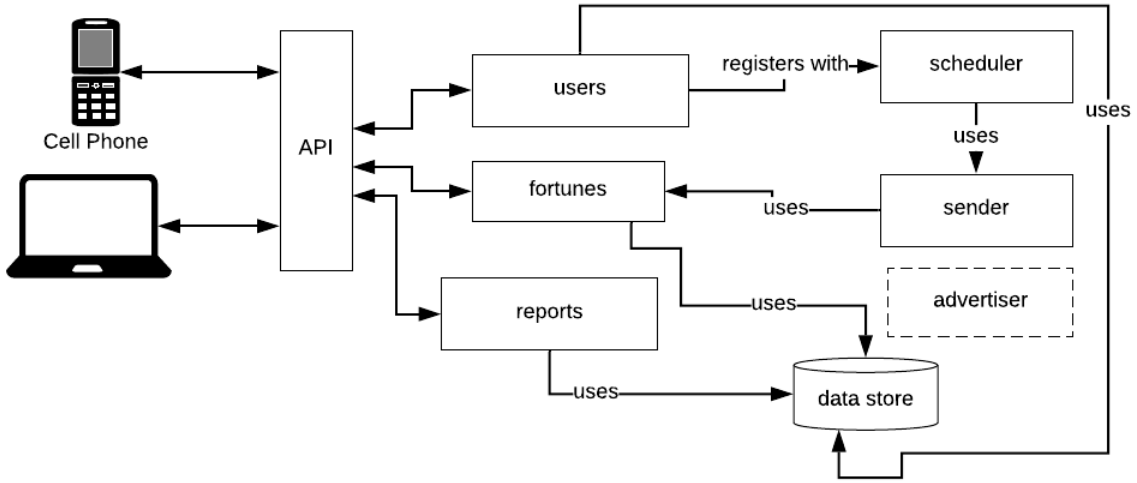
Situations where microservices might not be best

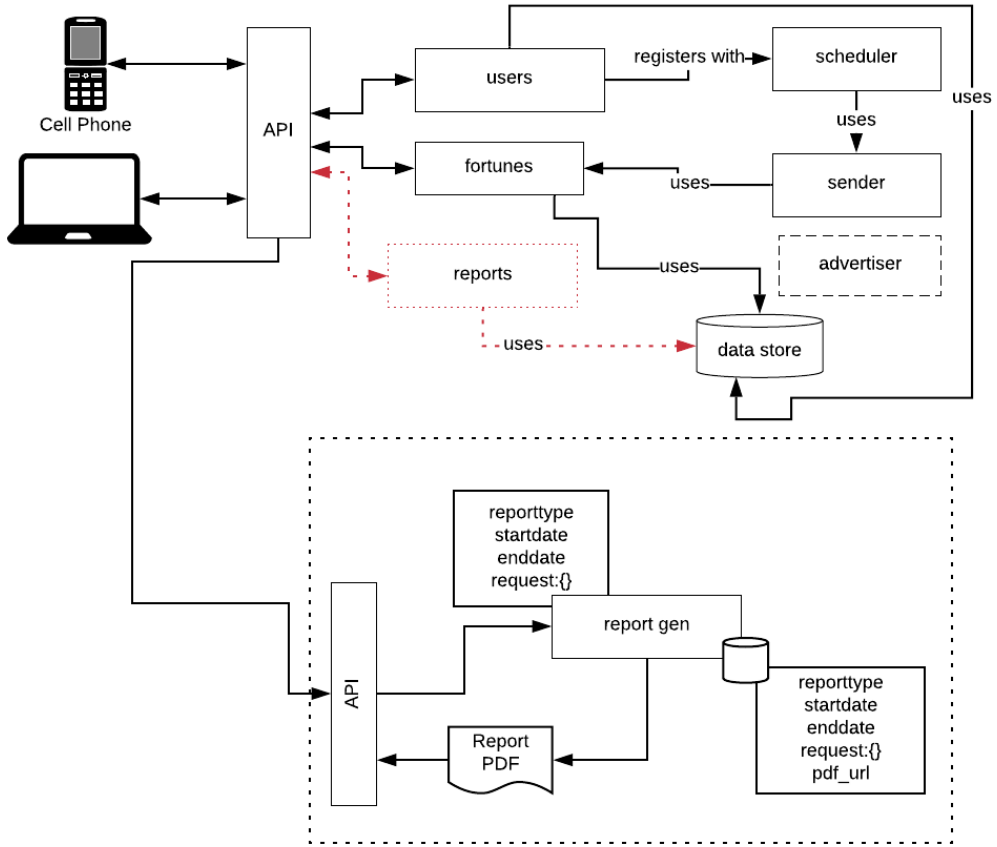
- Organizations with strong policies and practices oriented around centrally managed, heavy or monolithic components
- Organizations where processes are based on waterfall, lengthy development cycles, and heavyweight lifecycle processes

PRO TIP: Go for the low hanging fruit!

© 2020 by Innovation In Software Corporation

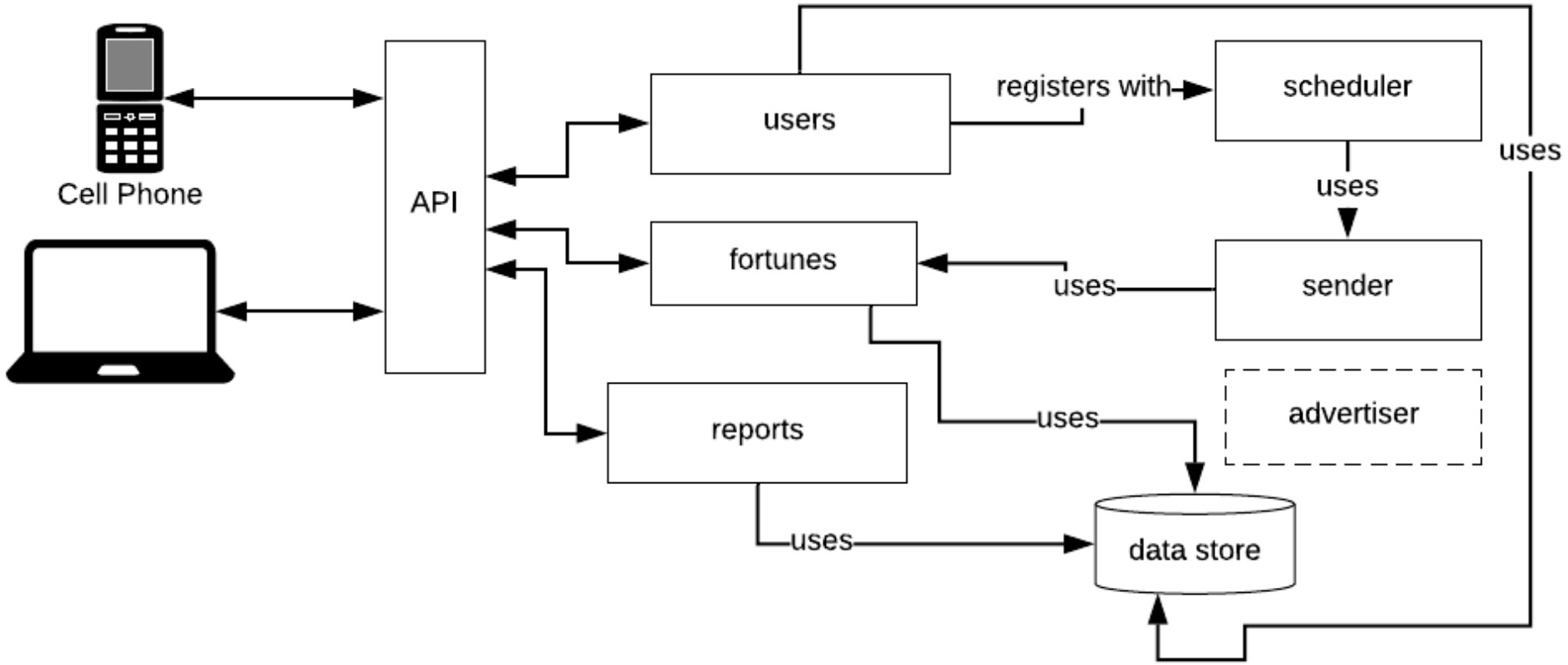
DECOMPOSITION CASE STUDY





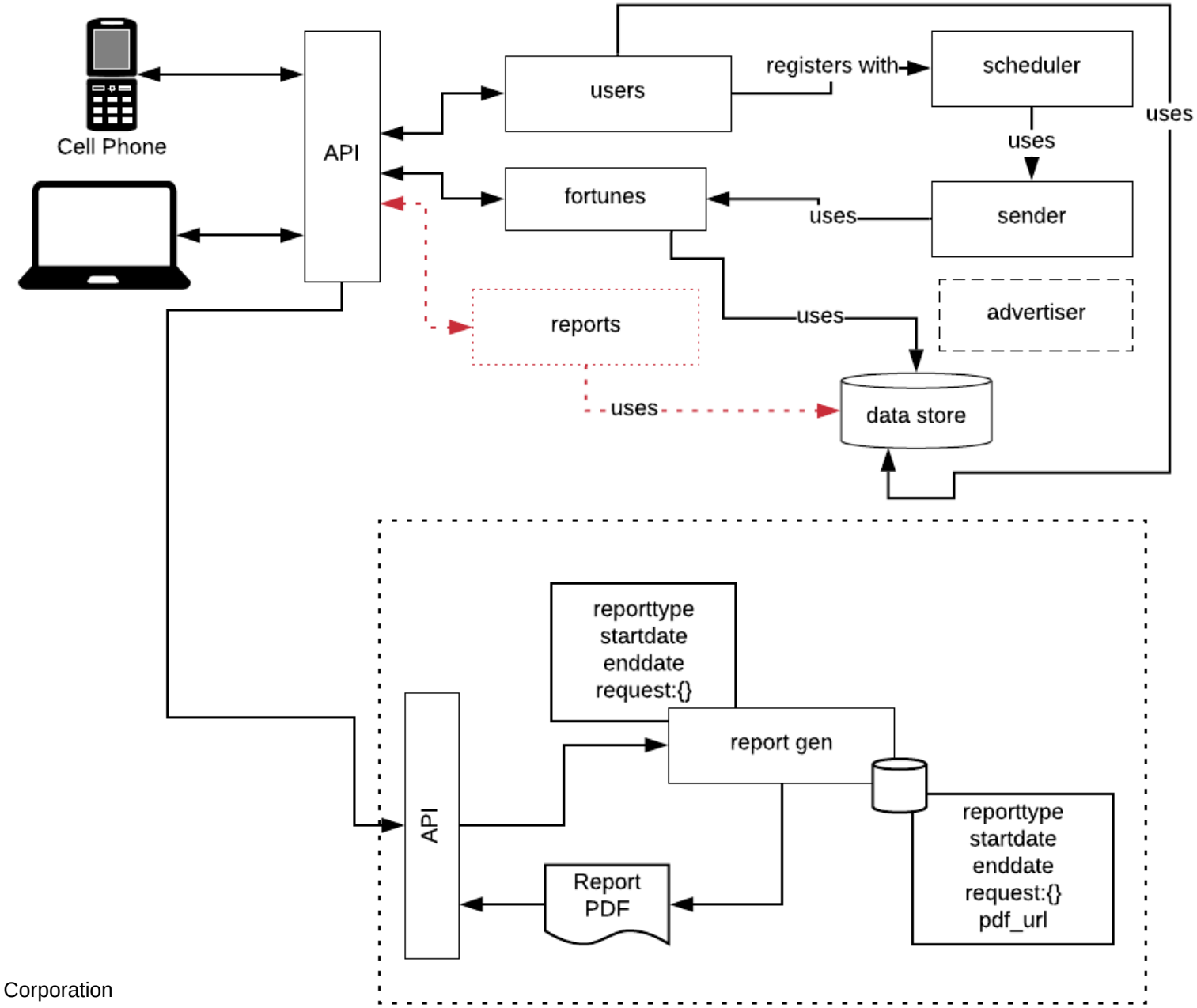
© 2020 by Innovation In Software Corporation

DECOMPOSITION CASE STUDY



© 2020 by Innovation In Software Corporation

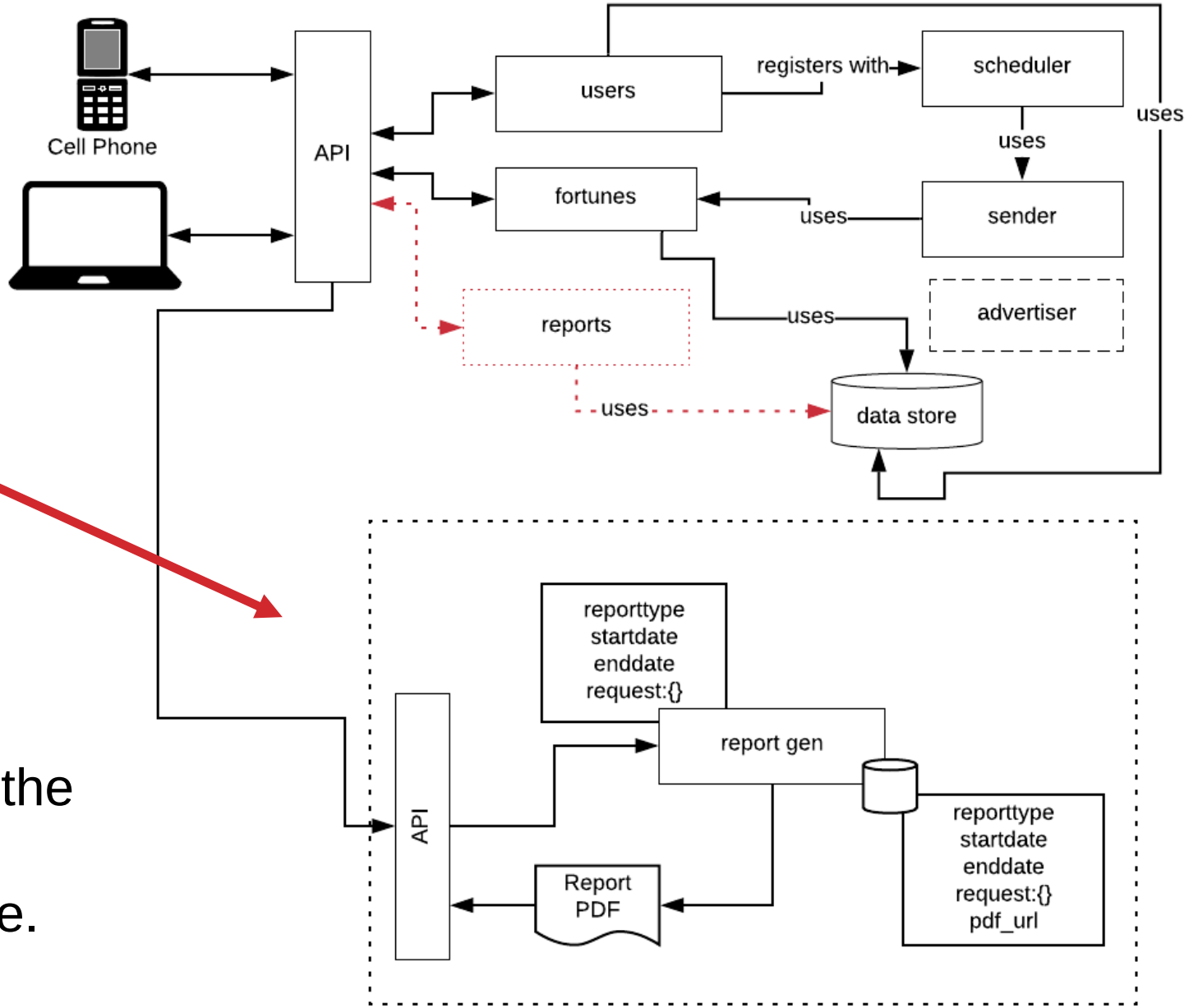
DECOMPOSITION CASE STUDY



© 2020 by Innovation In Software Corporation

DECOMPOSITION EXERCISE

This is a decomposition of Reports as a synchronous microservice.



Objective

Decompose Reports from the monolithic version to an asynchronous microservice.

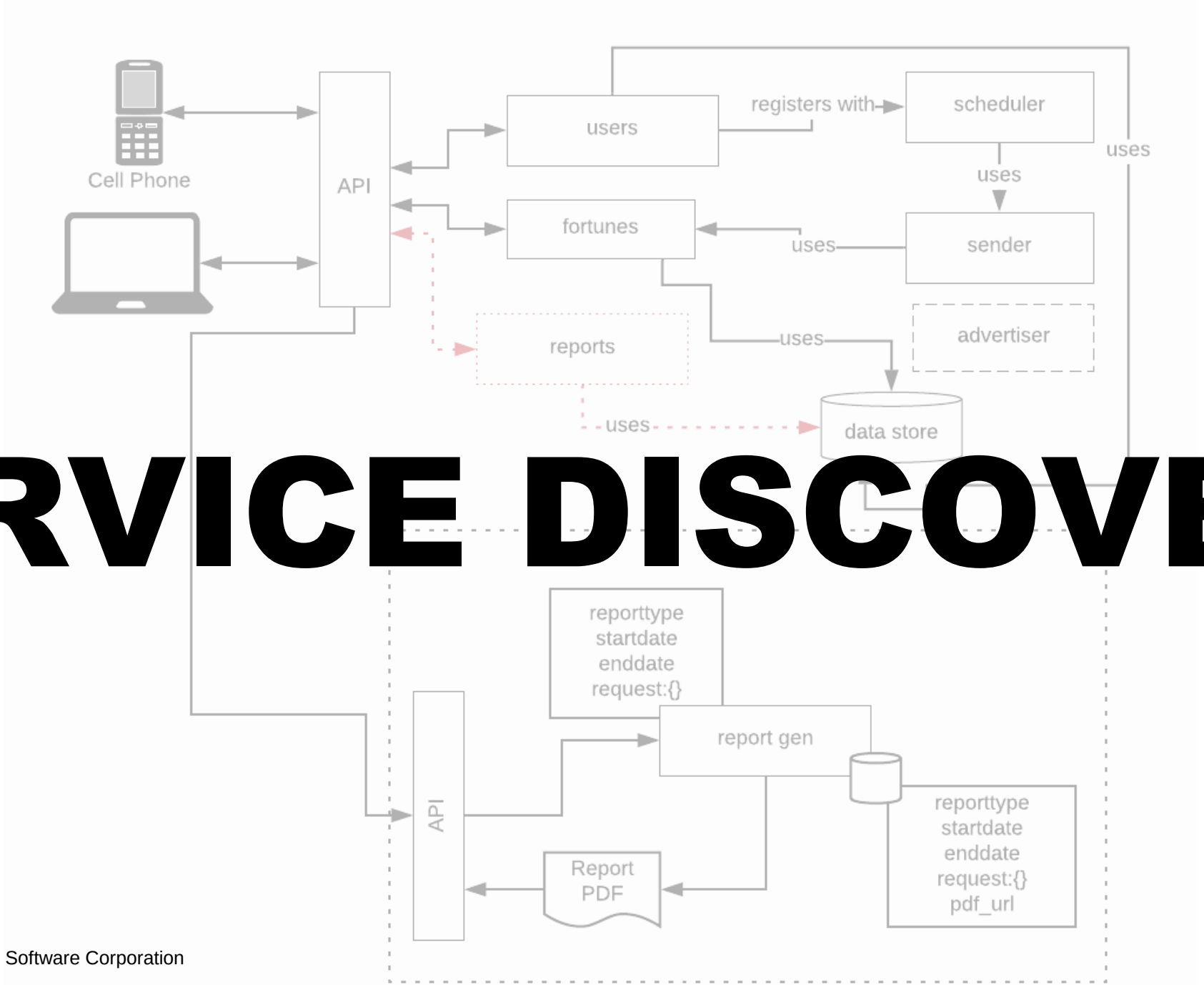
© 2020 by Innovation In Software Corporation

LAB: K8S DEPLOYMENT DISCOVERY

https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/03-deployment-discovery-red-green

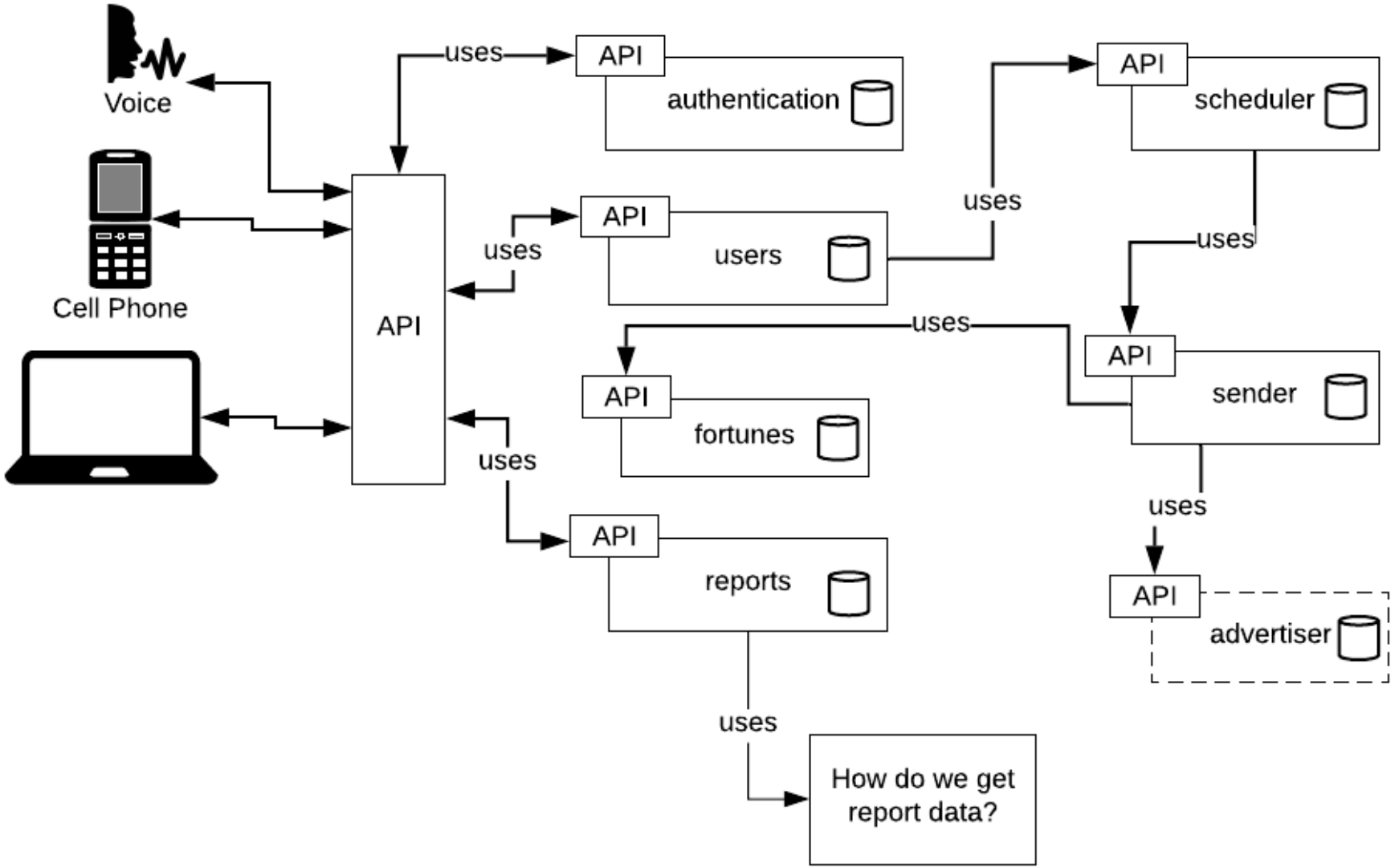
© 2020 by Innovation In Software Corporation

SERVICE DISCOVERY



© 2020 by Innovation In Software Corporation

HOW DO THE SERVICES KNOW HOW TO FIND EACH OTHER?



© 2020 by Innovation In Software Corporation

HOW DO THE SERVICES KNOW HOW TO FIND EACH OTHER?

HashiCorp
Consul

Apache
Zookeeper

BUGLER
ORIGINAL
SINCE 1932
TURKISH & BLENDED
CIGARETTE TOBACCO

Roll your own!

© 2020 by Innovation In Software Corporation

HOW DO THE SERVICES KNOW HOW TO FIND EACH OTHER?

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: sender
  labels:
    app: sender
spec:
  containers:
  - name: sender
    image: localhost:5000/sender
    imagePullPolicy: Always
    ports:
      - name: app
        containerPort: 3000
    env:
      - name: TWITTER_API_URL
        value: "http://twitter/"
      - name: FACEBOOK_API_URL
        value: "http://facebook/"
      - name: LINKEDIN_API_URL
        value: "http://linkedin/"
      - name: INSTAGRAM_API_URL
        value: "http://instagram/"
      - name: SMS_API_URL
        value: "http://sms/"
      - name: EMAIL_API_URL
        value: "http://email/"
      - name: FORTUNES_API_URL
        value: "http://fortunes/"
```

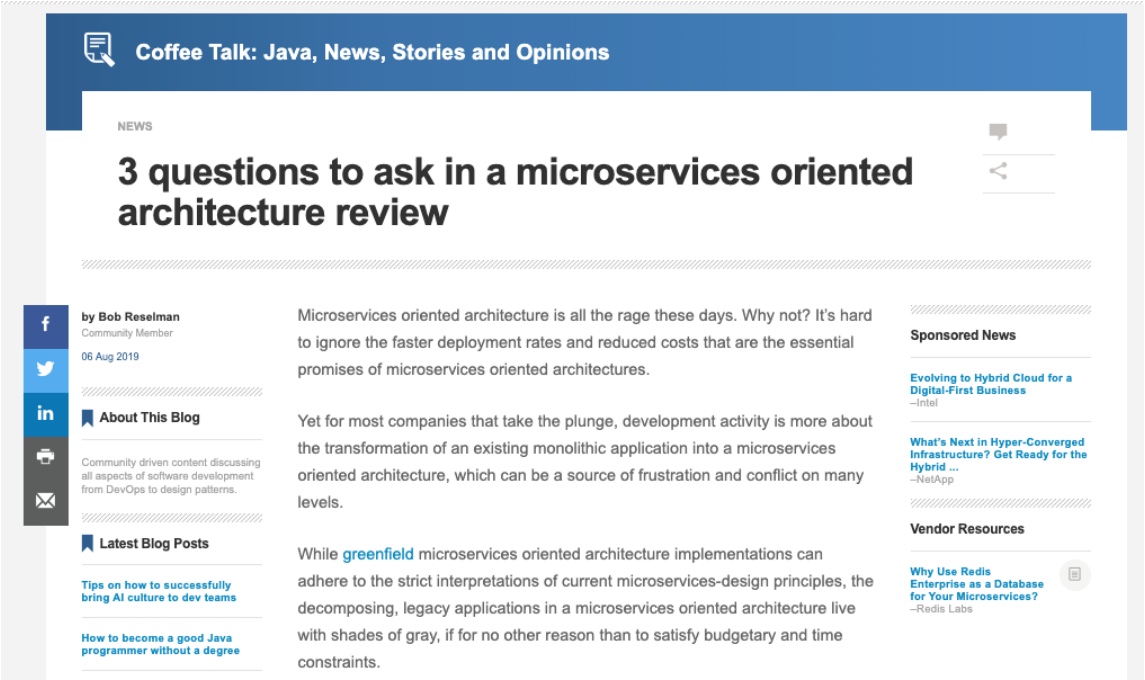
Using K8S implicit service discovery

© 2020 by Innovation In Software Corporation

HOW DO THE SERVICES KNOW HOW TO FIND EACH OTHER?

Any choice of service discovery can work as long as:

- It’s justifiable
- You an identify the risks
- You have a plan to mitigate the risks



Coffee Talk: Java, News, Stories and Opinions

NEWS

3 questions to ask in a microservices oriented architecture review

by Bob Reselman
Community Member
06 Aug 2019

About This Blog

Community driven content discussing all aspects of software development from DevOps to design patterns.

Latest Blog Posts

Tips on how to successfully bring AI culture to dev teams

How to become a good Java programmer without a degree

Microservices oriented architecture is all the rage these days. Why not? It's hard to ignore the faster deployment rates and reduced costs that are the essential promises of microservices oriented architectures.

Yet for most companies that take the plunge, development activity is more about the transformation of an existing monolithic application into a microservices oriented architecture, which can be a source of frustration and conflict on many levels.

While greenfield microservices oriented architecture implementations can adhere to the strict interpretations of current microservices-design principles, the decomposing, legacy applications in a microservices oriented architecture live with shades of gray, if for no other reason than to satisfy budgetary and time constraints.

Sponsored News

Evolving to Hybrid Cloud for a Digital-First Business
–Intel

What's Next in Hyper-Converged Infrastructure? Get Ready for the Hybrid ...
–NetApp

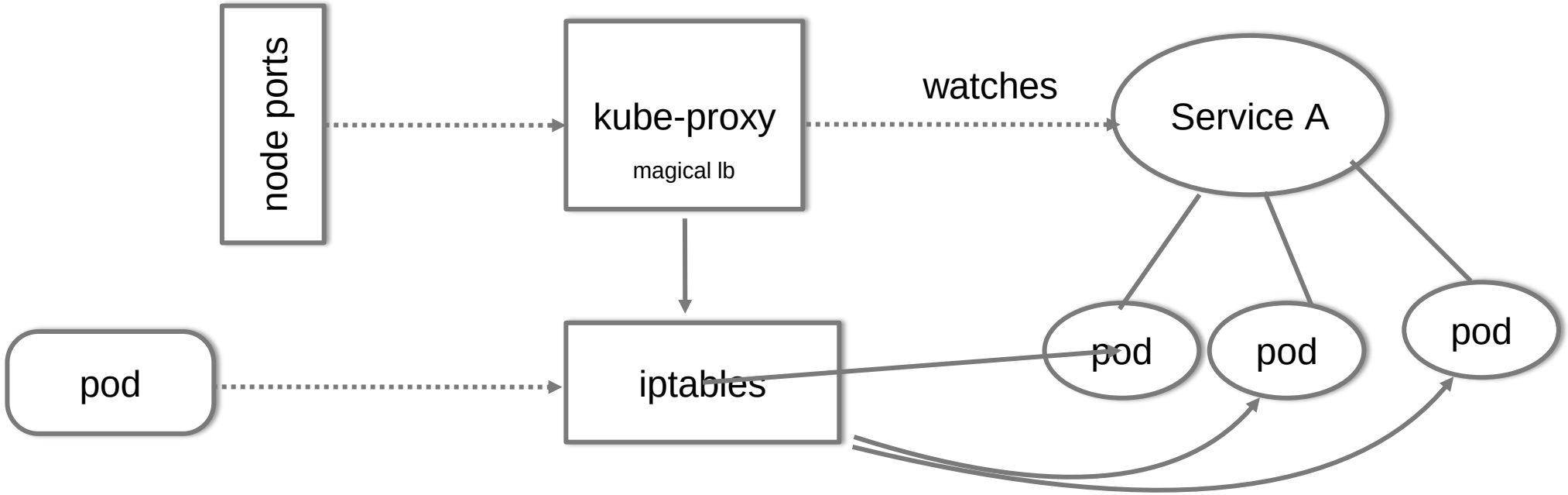
Vendor Resources

Why Use Redis Enterprise as a Database for Your Microservices?
–Redis Labs

https://bit.ly/33f5eOp

© 2020 by Innovation In Software Corporation

HOW K8S DOES SERVICE DISCOVERY USING DNS



© 2020 by Innovation In Software Corporation

SERVICE

[service-name].[namespace].svc.cluster.local

my-service.default.svc.cluster.local

http://my-service

MICROSERVICE DISCOVERY K8S DNS NAMING CONVENTION

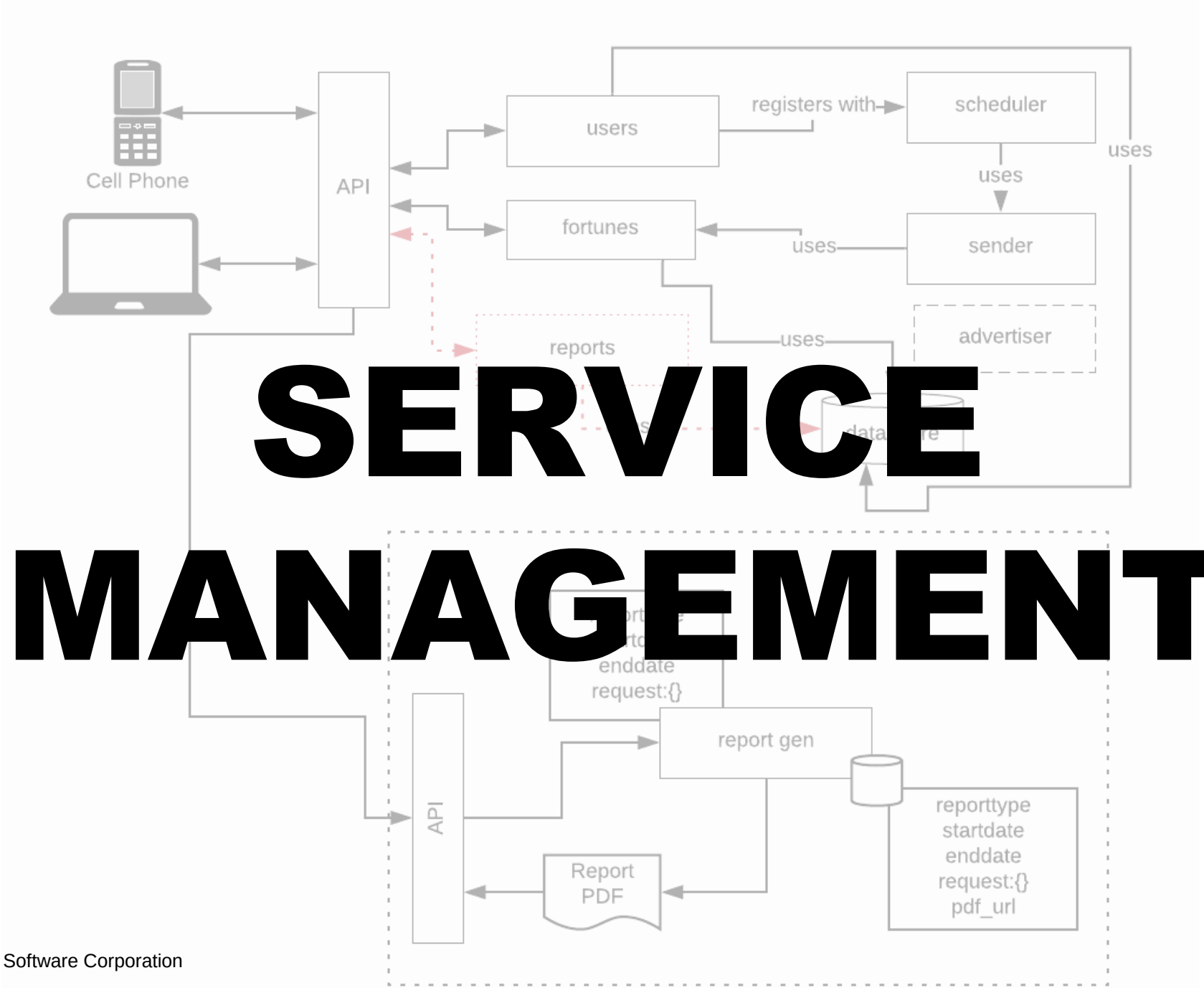
© 2020 by Innovation In Software Corporation

LAB: K8S SERVICE DISCOVERY

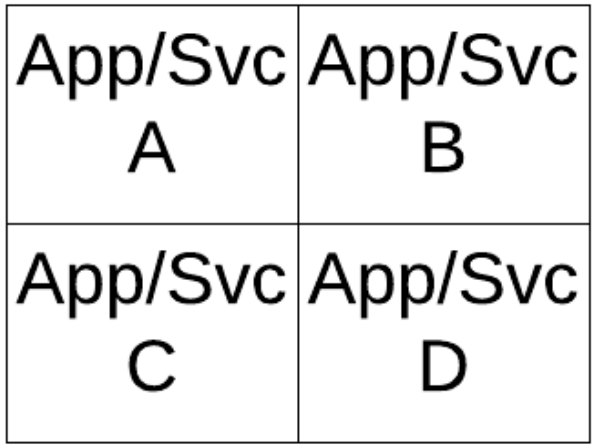
https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/04-service-discovery-k8s

© 2020 by Innovation In Software Corporation

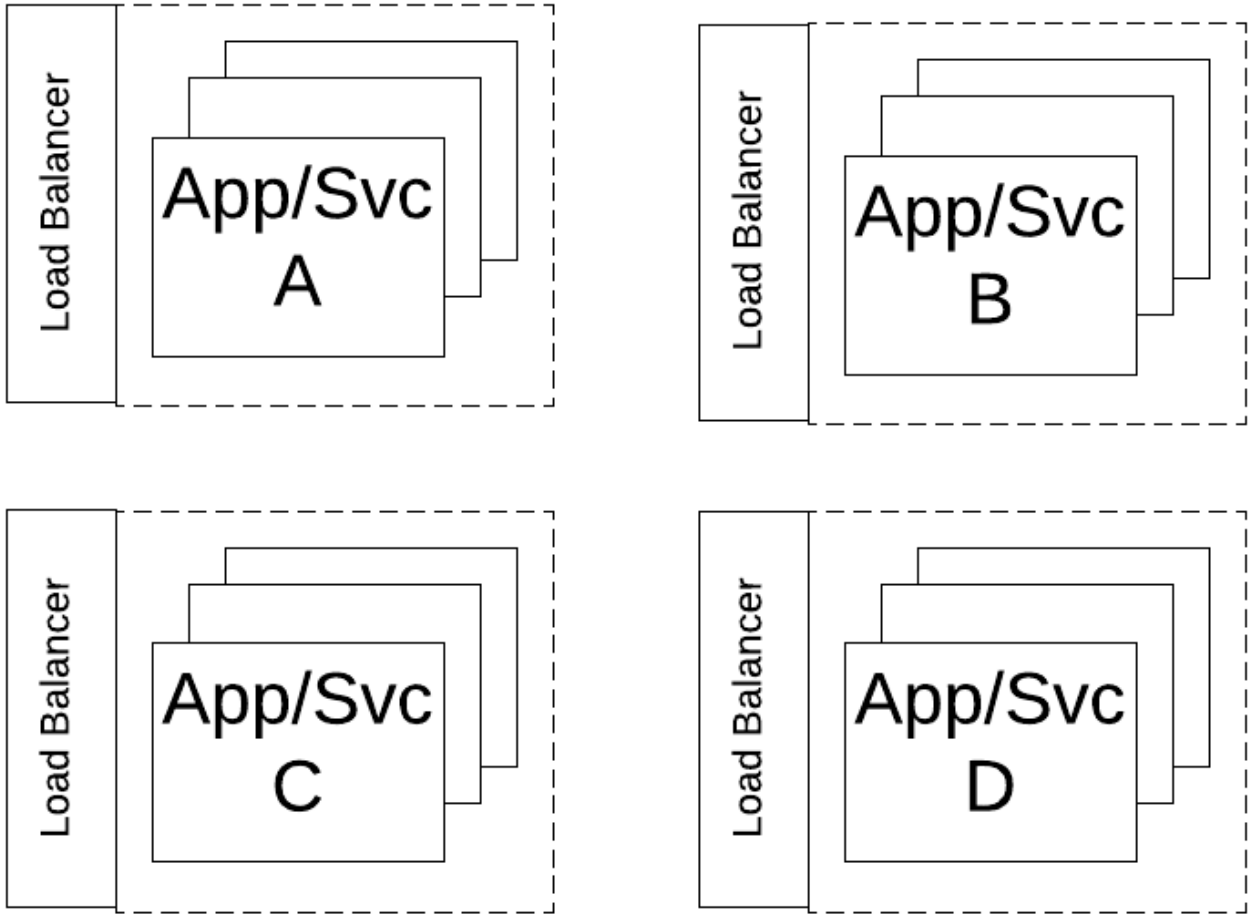
SERVICE MANAGEMENT



© 2020 by Innovation In Software Corporation



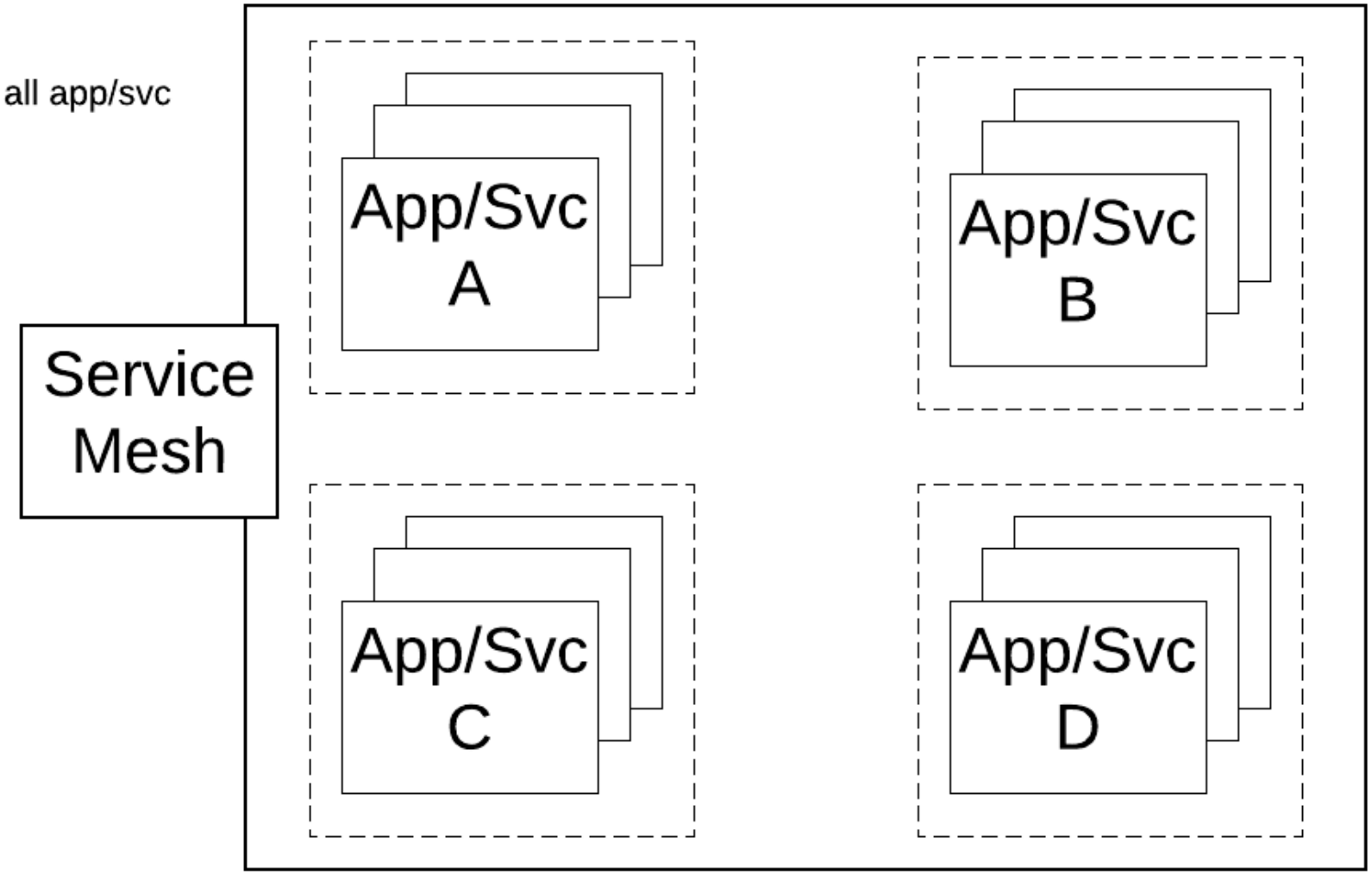
Single Machine
Shared Memory



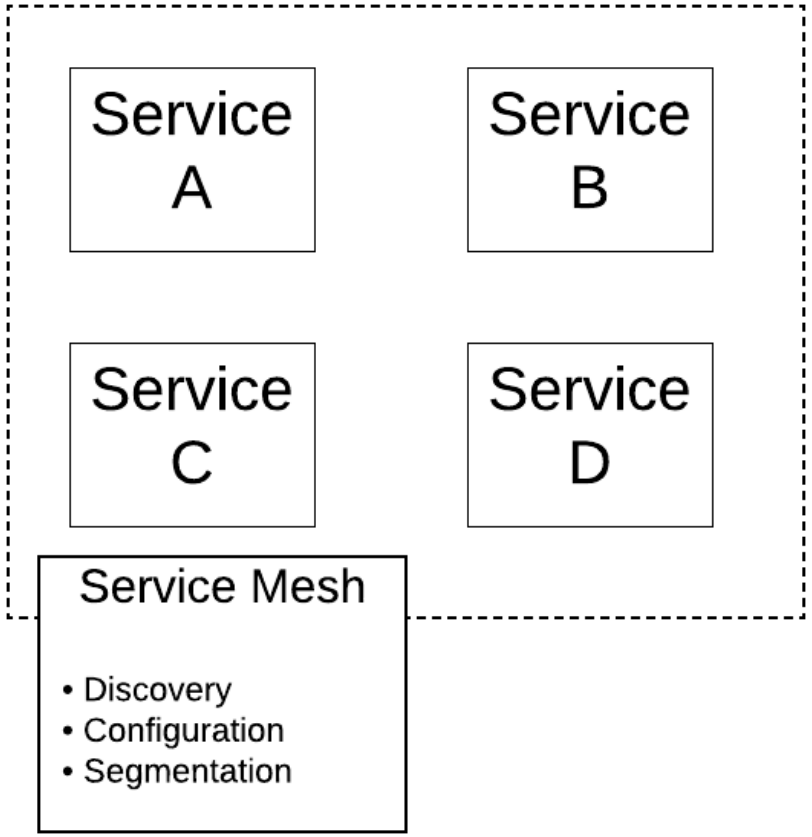
Multiple Machines
Isolated Memory

THE USE CASE FOR A SERVICE MESH

- Knows the location of all instances of all app/svc
- Can route traffic accordingly
- Can enforce access policies
- Can handle service configurations



THE USE CASE FOR A SERVICE MESH



Istio

HashiCorp Consul

Service Fabric Mesh!

AWS App Mesh

MICROSERVICE SERVICE MESH PROJECTS

LAB: ISTIO SERVICE MESH

https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/05-service-mesh-istio

© 2020 by Innovation In Software Corporation

12 FACTOR APP

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATIONS (CONT'D)

The 12-factor methodologies were established, were:

- **Uses declarative formats to cut down the cost and time for developers for setting up Automation.**
- **Partake a clean contract underlying operating system, which offers maximum portability amongst implementing environments.**

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATIONS (CONT'D)

The 12-factor methodologies were established, were:

- **It forsakes the need for servers and system admins as they are suitable for deployment on modern cloud platforms.**
- **Gives maximum agility for continuous deployment by minimizing divergence between development and production.**

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATIONS (CONT'D)

The 12-factor methodologies were established, were:

- It can scale up without any noteworthy changes to tooling, architecture or development practices.
- The 12 factor methodology is language independent.
- Can be applied to application written in any programming language using any combination of backing services (database, queue, memory cache, etc.).

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION

Challenges

- How fast can one fail?
- How on a push of a button, the team can get the work done?
- What process can be used so that code can be made available to production?

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION

Challenges

- **How to automate the above used process with continuous integration (CI), continuous deployment (CD), and even sometimes continuous delivery (CD)?**
- **How to convince the team to change methodology? What are the processes which needs to be defined for the work to be done?**

© 2020 by Innovation In Software Corporation

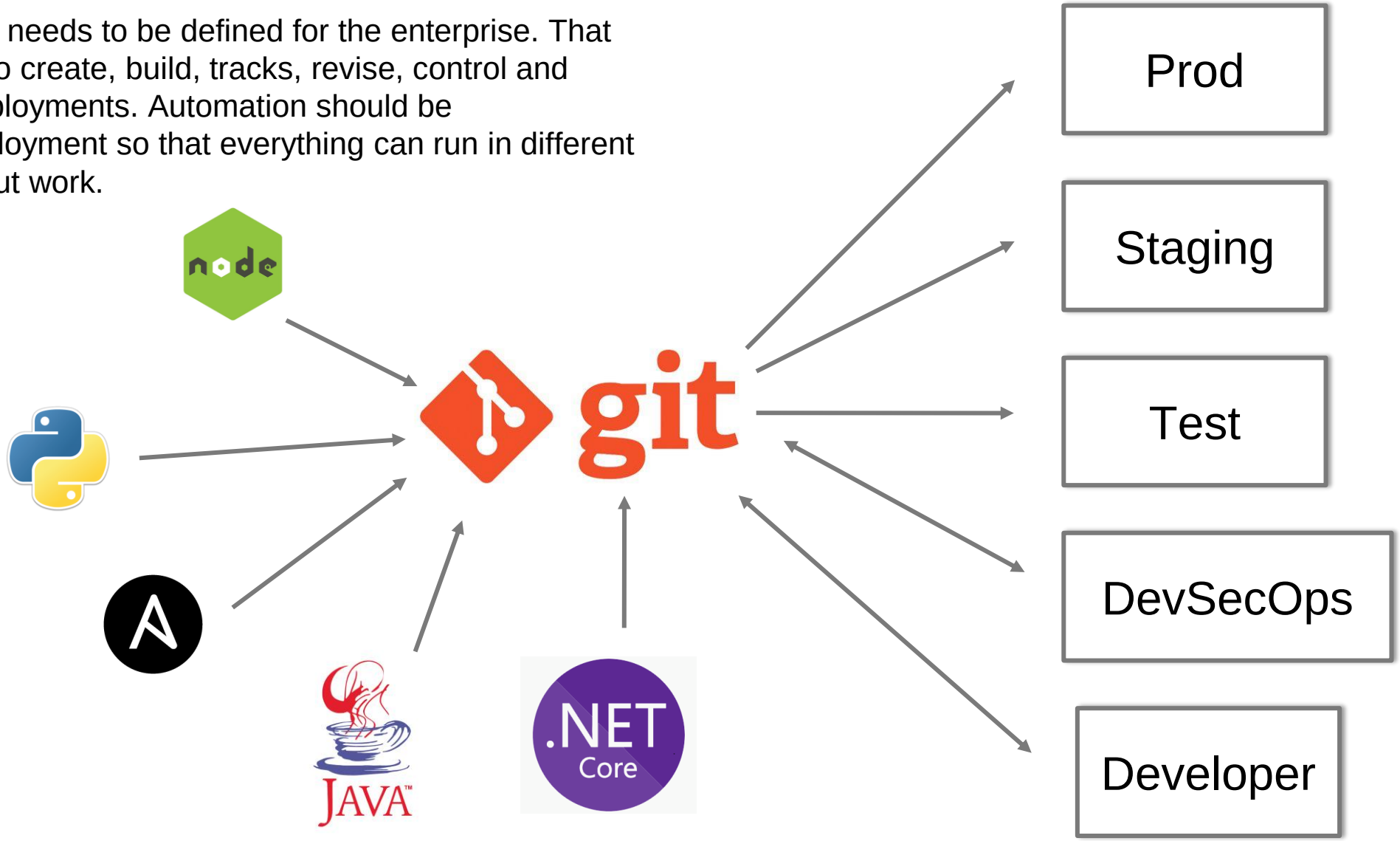
12-FACTOR APPLICATION BUILDING BLOCKS

- Codebase
- Port Binding
- Dependencies
- Concurrency
- Configuration
- Disposability
- Backing Services
- Dev/Prod Parity
- Build, Release, Run
- Logs
- Processes
- Admin Processes

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: CODE BASE

One codebase needs to be defined for the enterprise. That code base is used to create, build, tracks, revise, control and conduct various deployments. Automation should be implemented in deployment so that everything can run in different environments without work.



© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: DEPENDENCIES

Isolate dependencies is all about clearly declaring and isolating the dependencies as an app is a standalone structure, which needs to install dependencies. For this, whatever is required is declared in the code configuration itself.

```
"dependencies": {
  "apollo-link": "^1.2.11",
  "apollo-link-ws": "^1.0.17",
  "apollo-server": "^2.4.0",
  "graphql": "^14.2.1",
  "graphql-tag": "^2.10.1",
  "lodash": "^4.17.11",
  "node-fetch": "^2.3.0",
  "subscriptions-transport-ws": "^0.9.16",
  "uuid": "^3.3.2",
  "ws": "^6.2.0"
},
"devDependencies": {
  "chai": "^4.2.0",
  "faker": "^4.1.0",
  "graphql-request": "^1.8.2",
  "mocha": "^5.2.0",
  "supertest": "^3.4.2"
}
```

node: package.json

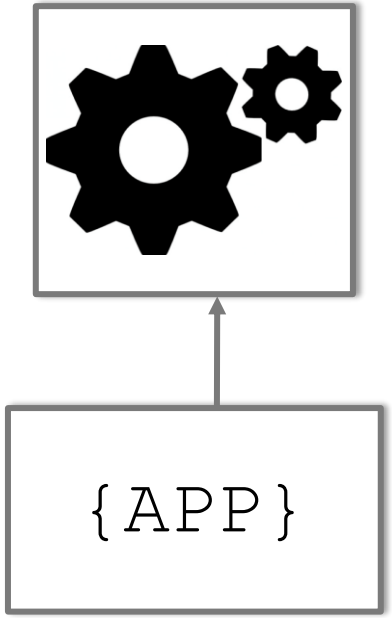
```
ordereddict==1.1
argparse==1.2.1
python-dateutil==2.2
matplotlib==1.3.1
nose==1.3.0
numpy==1.8.0
pymongo==3.3.0
psutil>=2.0
```

python: requirements.txt

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: CONFIGURATION

How to store configuration information in the environment. This factor focuses and how to store "The Data". The URI for a database will be different in Development phase, QA phase and Production.



Helpful Hint: Never store sensitive information in a file on disk

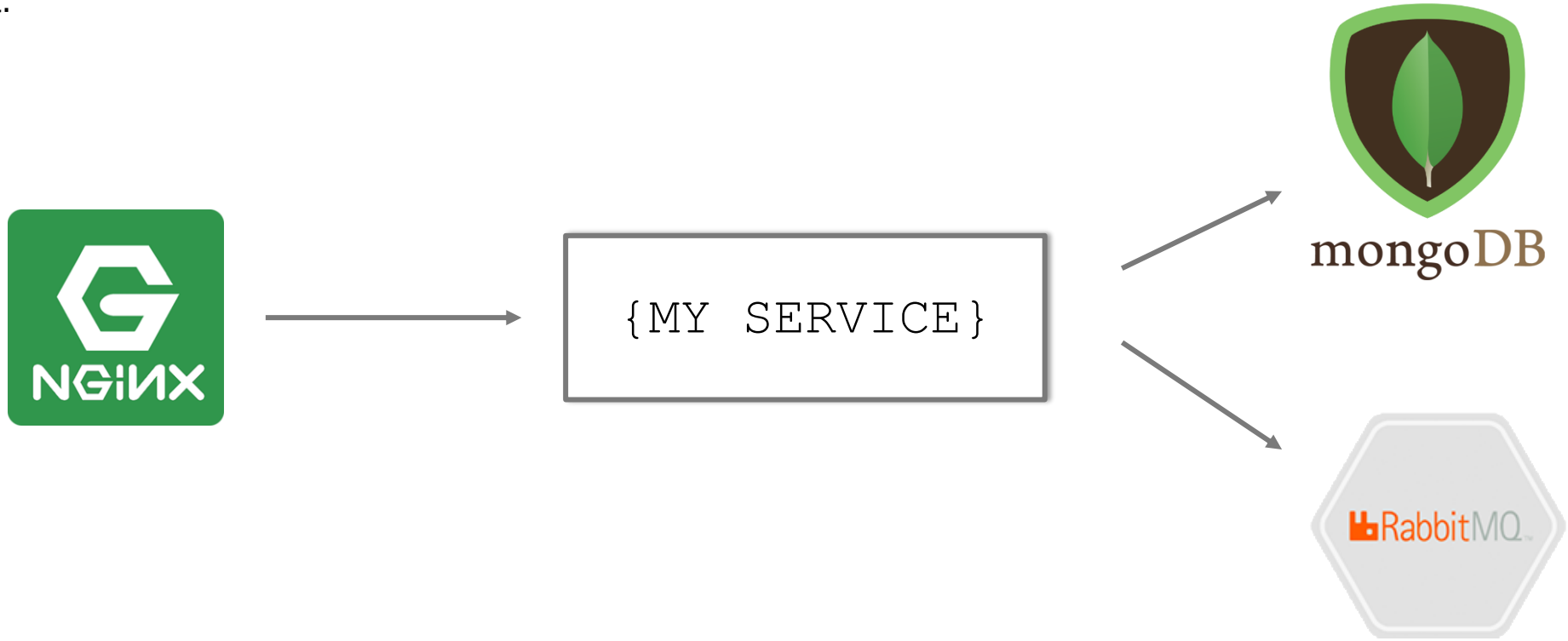
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx
spec:
  tls:
    - secretName: example-com-tls
      hosts:
        - example.com
  rules:
    - host: example.com
      http:
        paths:
        - path: /
          backend:
            serviceName: nginx
            servicePort: 80
```

TLS Certificate embedded in Kubernetes Secret

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: BACKING SERVICES

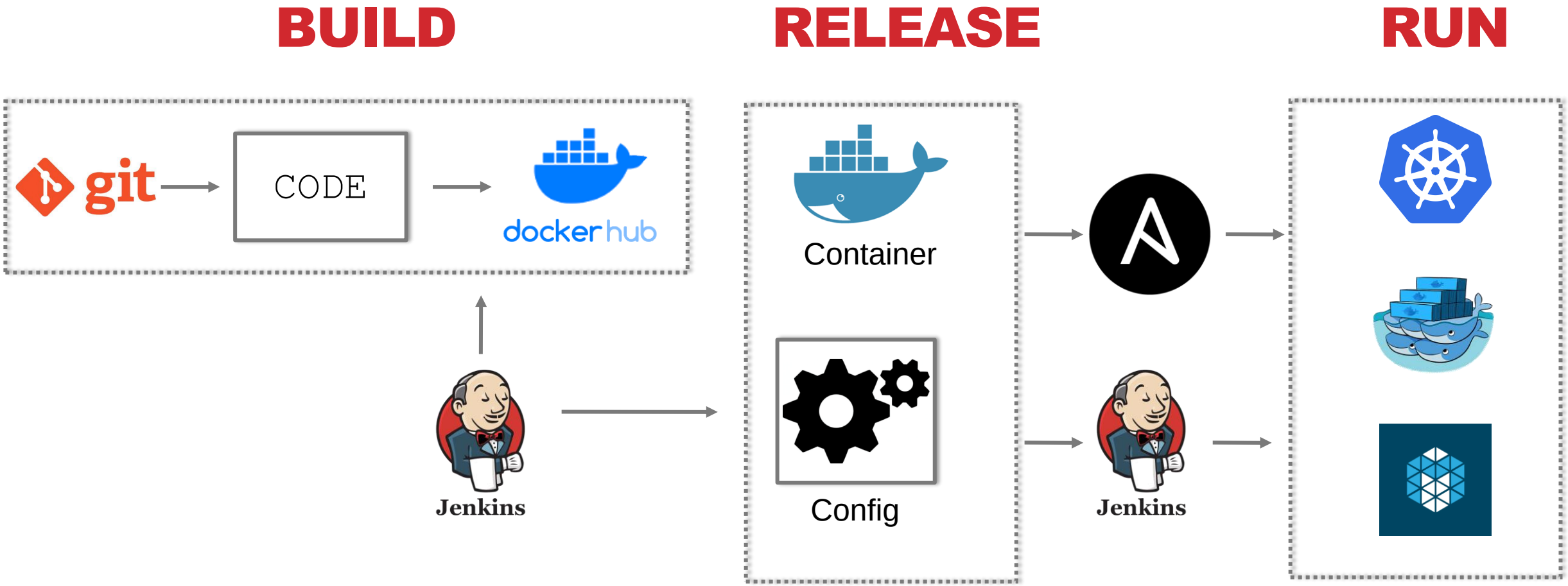
All the backing services should be treated as attached resources. You may want diverse storage resources depending on which team one might be in. With this methodology, each developer can have their own config file as occasionally a dev will want a lot of log files, which a QA will not.



© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: BUILD, RELEASE, RUN

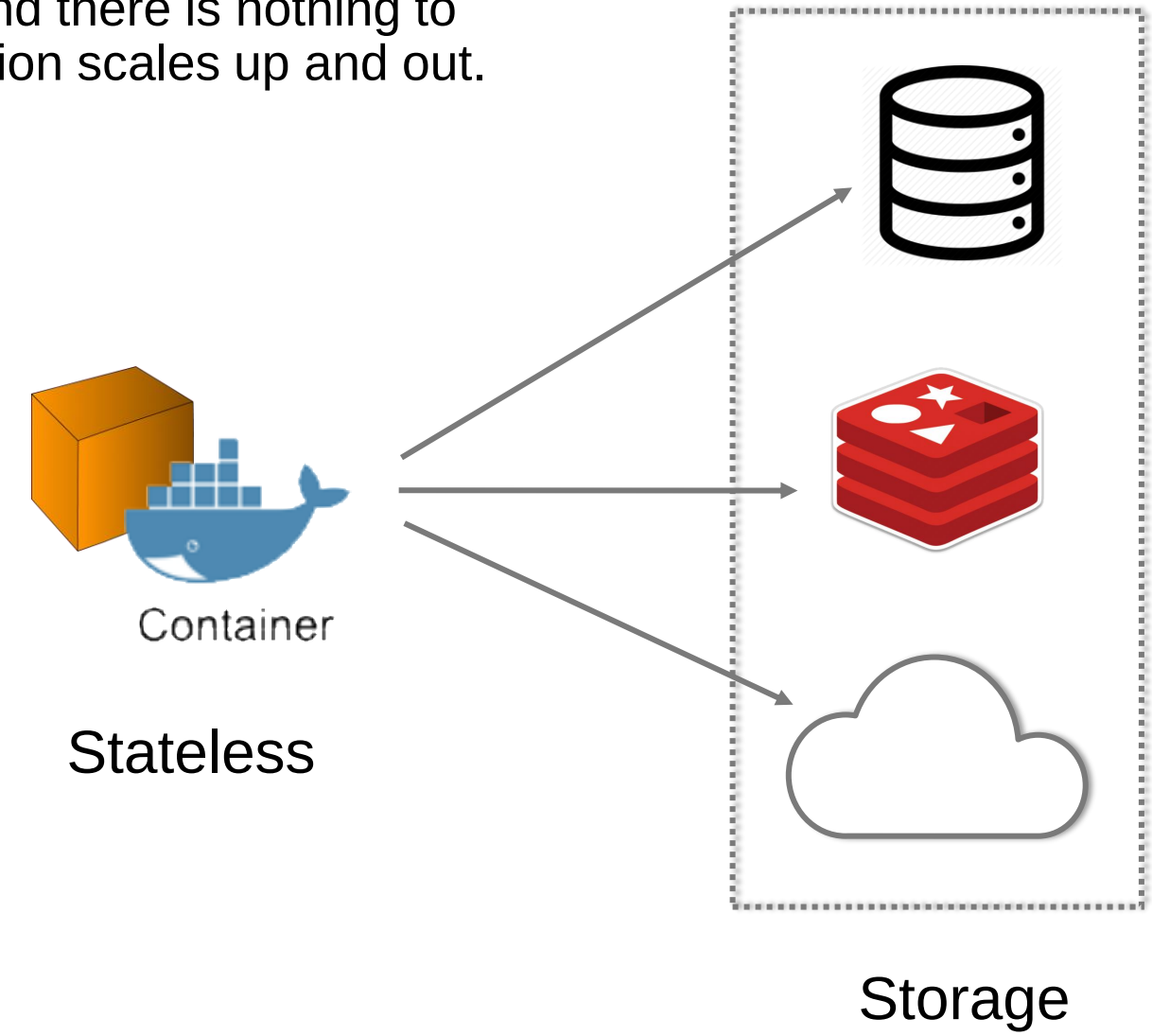
Separate build, release and run stages
making sure all the process has the right libraries.



© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: PROCESSES

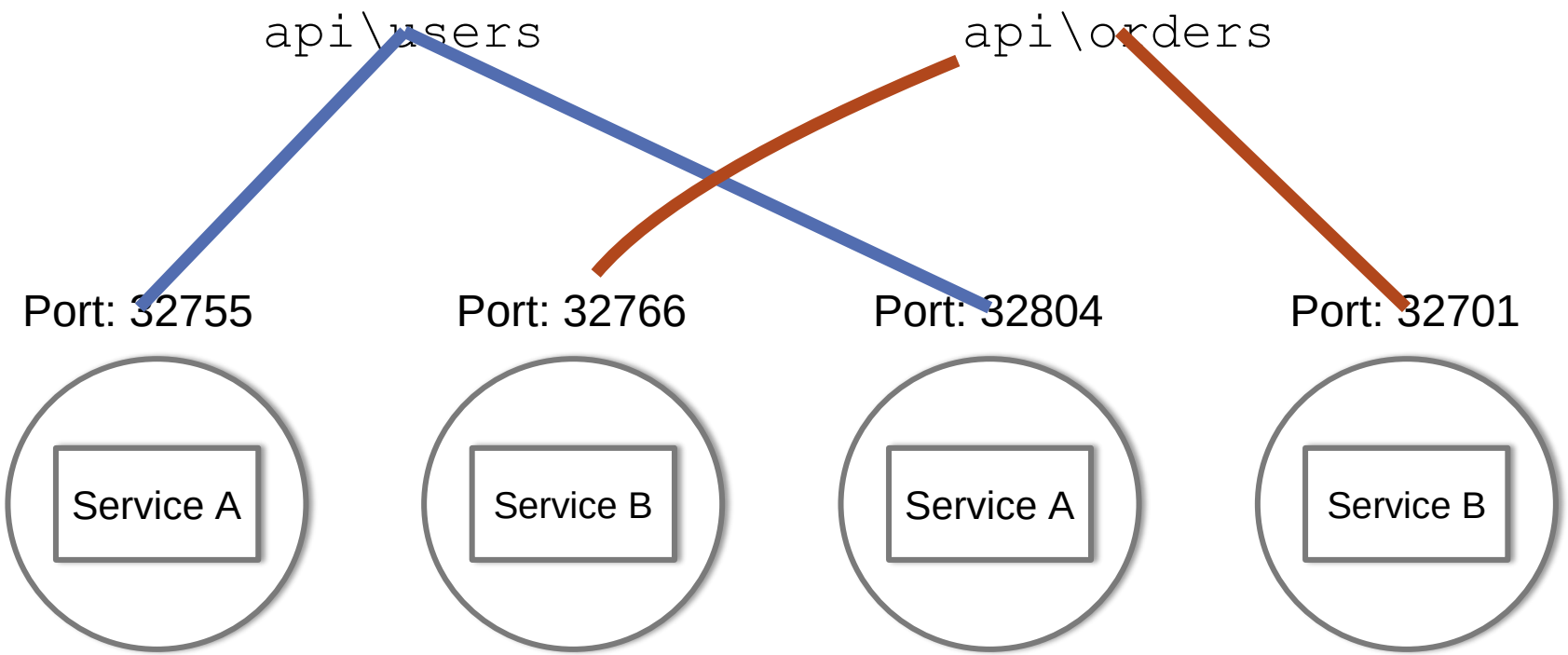
All processes should be stateless; ensure all data is stored in a backing store and there is nothing to pass along when the application scales up and out.



© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: PORT BINDINGS

This means exposing all services via port binding. It allows the internal customers to access the endpoints without traversing security. This also helps in checking the authenticity of the working stack from different ports.

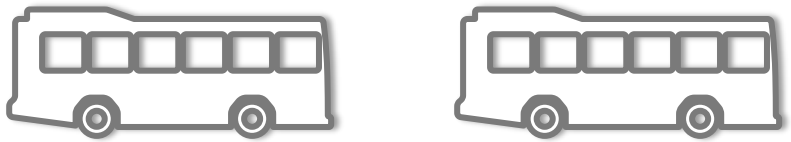


© 2020 by Innovation In Software Corporation

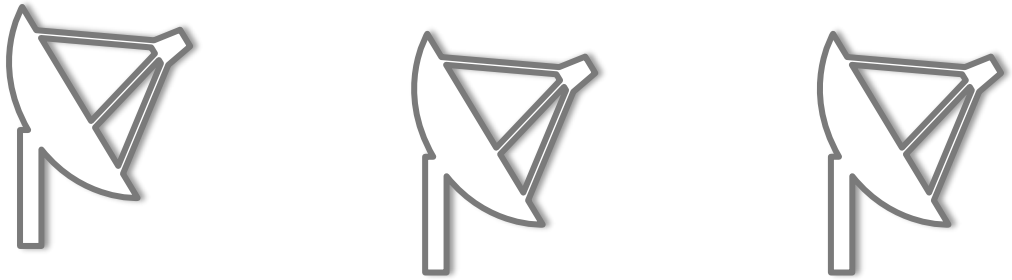
12-FACTOR APPLICATION: CONCURRENCY

Break the app into smaller chunks allowing chunks to scale up or down in order to handle fluctuation in load.

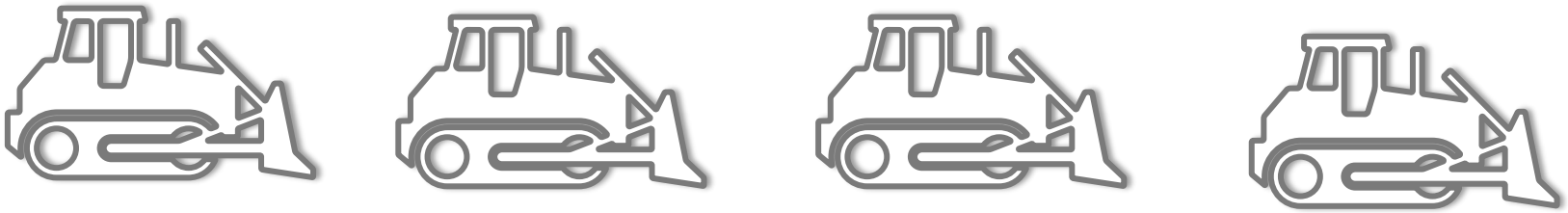
User Interface



Services



Workers



© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: DISPOSABILITY

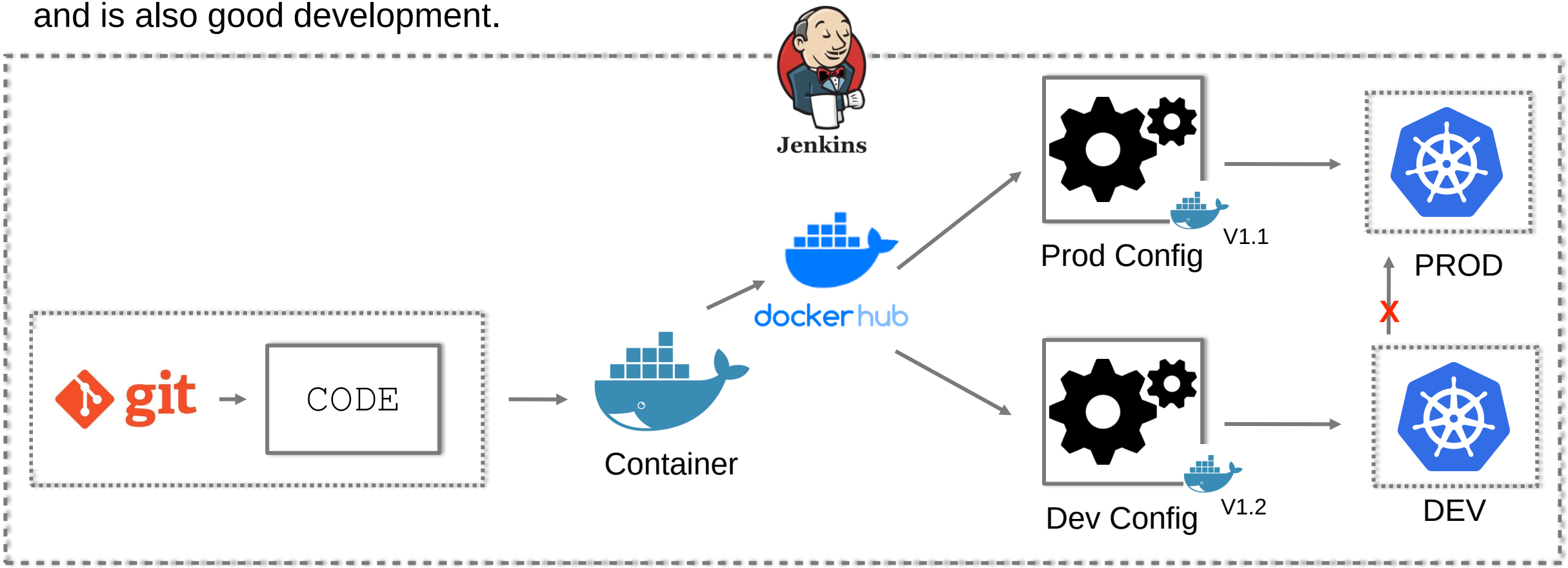
Ensure a high degree of robustness with fast startup, elegant shutdown and gracefully handling a crash.

```
const server = app.listen(3000);
logger.info(`Starting at ${new Date()}`);

process.on('SIGTERM', () =>{
  logger.info(`Shutting down at ${new Date()}`);
  server.close()
});
```

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: DEV/PROD PARITY

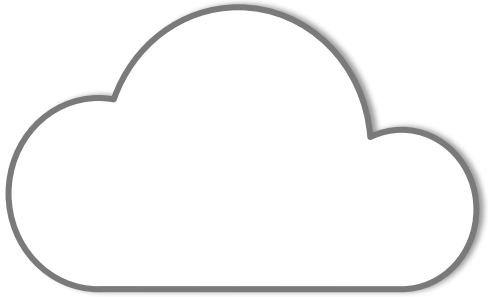
Continuous deployment requires continuous integration built on identical environments limiting divergence and blunders. Keep dev, staging and production as similar as possible helps anyone who comes in to understand the app and release it. Dev/Prod parity permits a lot more scalability and is also good development.



© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: LOGS

Logs can be treated as event streams as the app only concerns itself with creating an even stream. Which log needs to be published is then decided depending upon the configuration.



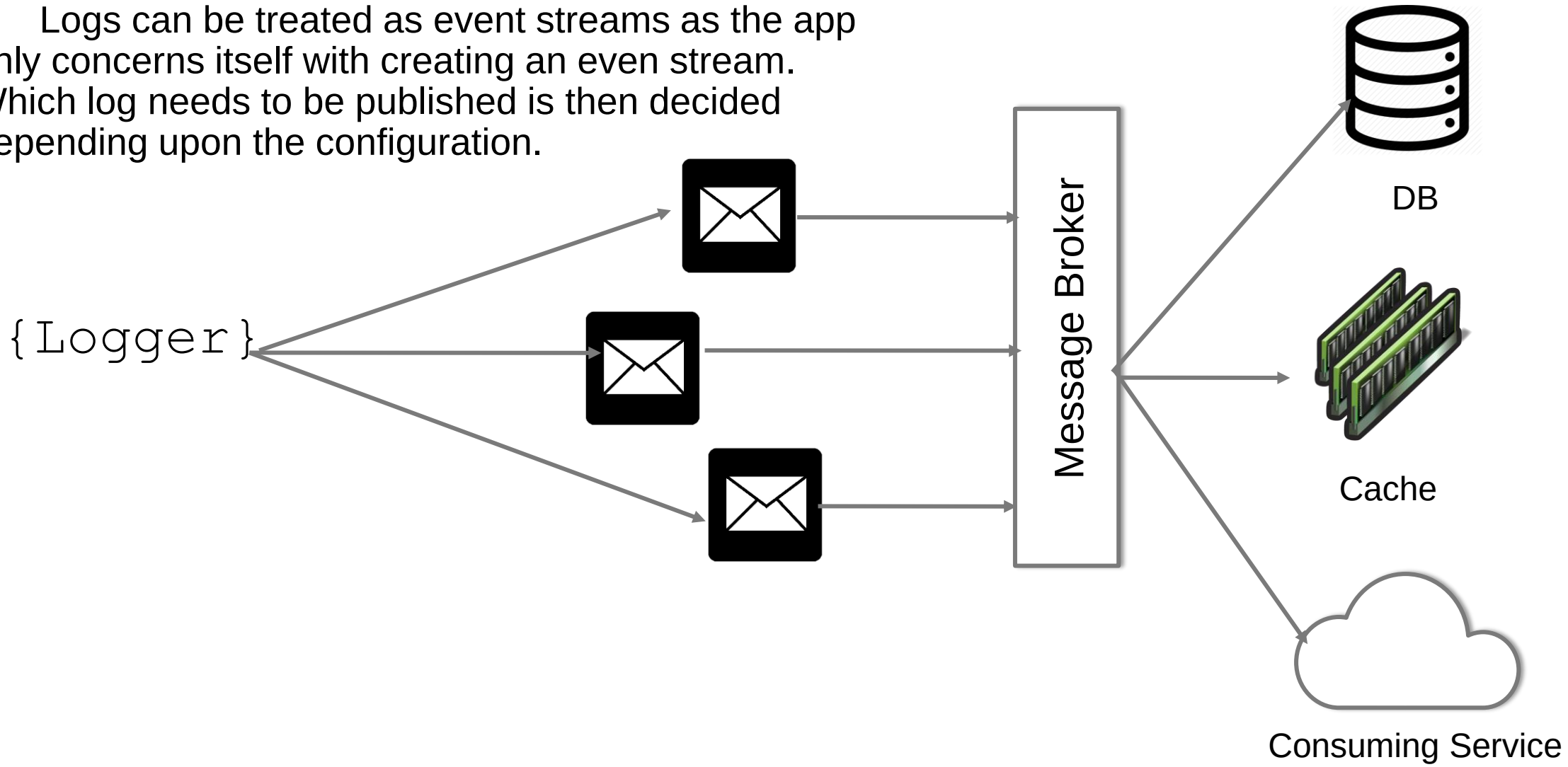
Logging Service

```
{                                   {APP}

        logURL = env.process.LOGGER_URL;
        logUser = env.process.LOGGER_USER;
        logPwd = env.process.PWD;

        const Logger = require(cloud-logger);

        logger = new Loogger({logURL,logUser,logPwd});

        logger.info(`Starting at ${new Date()}`);

}
```

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: LOGS

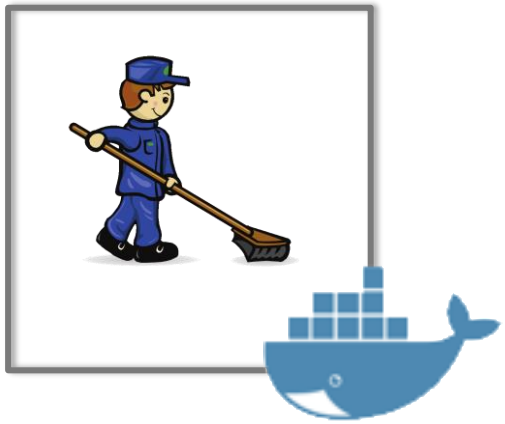
Logs can be treated as event streams as the app only concerns itself with creating an even stream. Which log needs to be published is then decided depending upon the configuration.



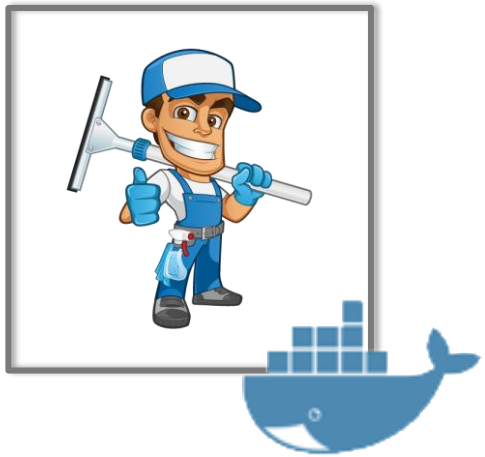
© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION: ADMIN PROCESSES

One of the off-processes can be running and managing admin tasks.



Container



Container

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION BUILDING BLOCKS

I. Codebase

One codebase needs to be defined for the enterprise. That code base is used to create, build, tracks, revise, control and conduct various deployments. Automation should be implemented in deployment so that everything can run in different environments without work.

II. Dependencies

Isolate dependencies is all about clearly declaring and isolating the dependencies as an app is a standalone structure, which needs to install dependencies. For this, whatever is required is declared in the code configuration itself.

III. Config

In this one store configuration information in the environment. This factor focuses and how to store “The Data”. The URI for a database will be different in Development phase, QA phase and Production.

IV. Backing Services

All the backing services should be treated as attached resources. You may want diverse storage resources depending on which team one might be in. With this methodology, each developer can have their own config file as occasionally a dev will want a lot of log files, which a QA will not.

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION BUILDING BLOCKS

V. Build, release, run

Separate build, release and run stages making sure all the process has the right libraries.

VI. Processes

All processes should be stateless; ensure all data is stored in a backing store and there is nothing to pass along when the application scales up and out.

VII. Port binding

This means exposing all services via port binding. It allows the internal customers to access the endpoints without traversing security. This also helps in checking the authenticity of the working stack from different ports.

VIII. Concurrency

Break the app into smaller chunks allowing chunks to scale up or down in order to handle fluctuation in load.

© 2020 by Innovation In Software Corporation

12-FACTOR APPLICATION BUILDING BLOCKS

IX. Disposability

Ensure a high degree of robustness with fast startup, elegant shutdown and gracefully handling a crash.

X. DEV/PROD parity

Continuous deployment requires continuous integration built on identical environments limiting divergence and blunders. Keep dev, staging and production as similar as possible helps anyone who comes in to understand the app and release it. Dev/Prod parity permits a lot more scalability and is also good development.

XI. Logs

Logs can be treated as event streams as the app only concerns itself with creating an even stream. Which log needs to be published is then decided depending upon the configuration.

XII. Admin Processes

One of the off-processes can be running and managing admin tasks.

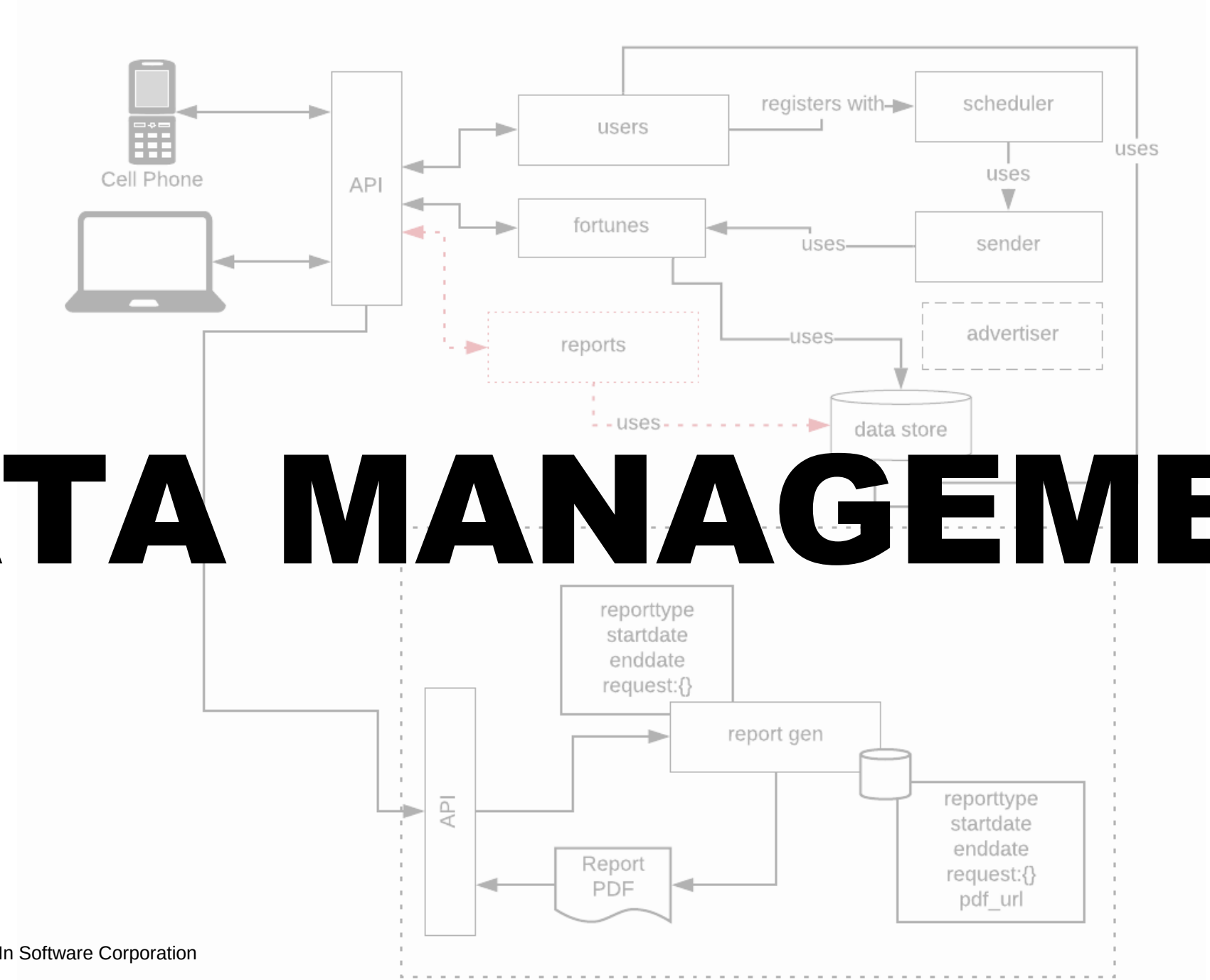
© 2020 by Innovation In Software Corporation

DESIGN SPIKE: DECOMPOSING FORTUNE COOKIES

https://github.com/GeorgeNiece/innosoft/tree/master/microservices-architecture/06-decomposition-to-moa

© 2020 by Innovation In Software Corporation

DATA MANAGEMENT

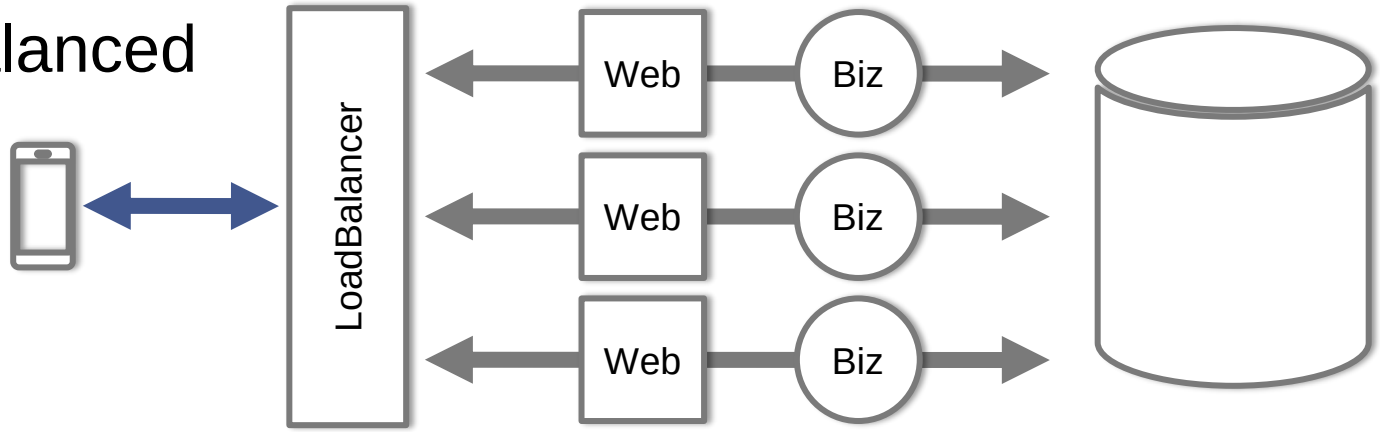


© 2020 by Innovation In Software Corporation

The Problem of Increasing Complexity

Simple

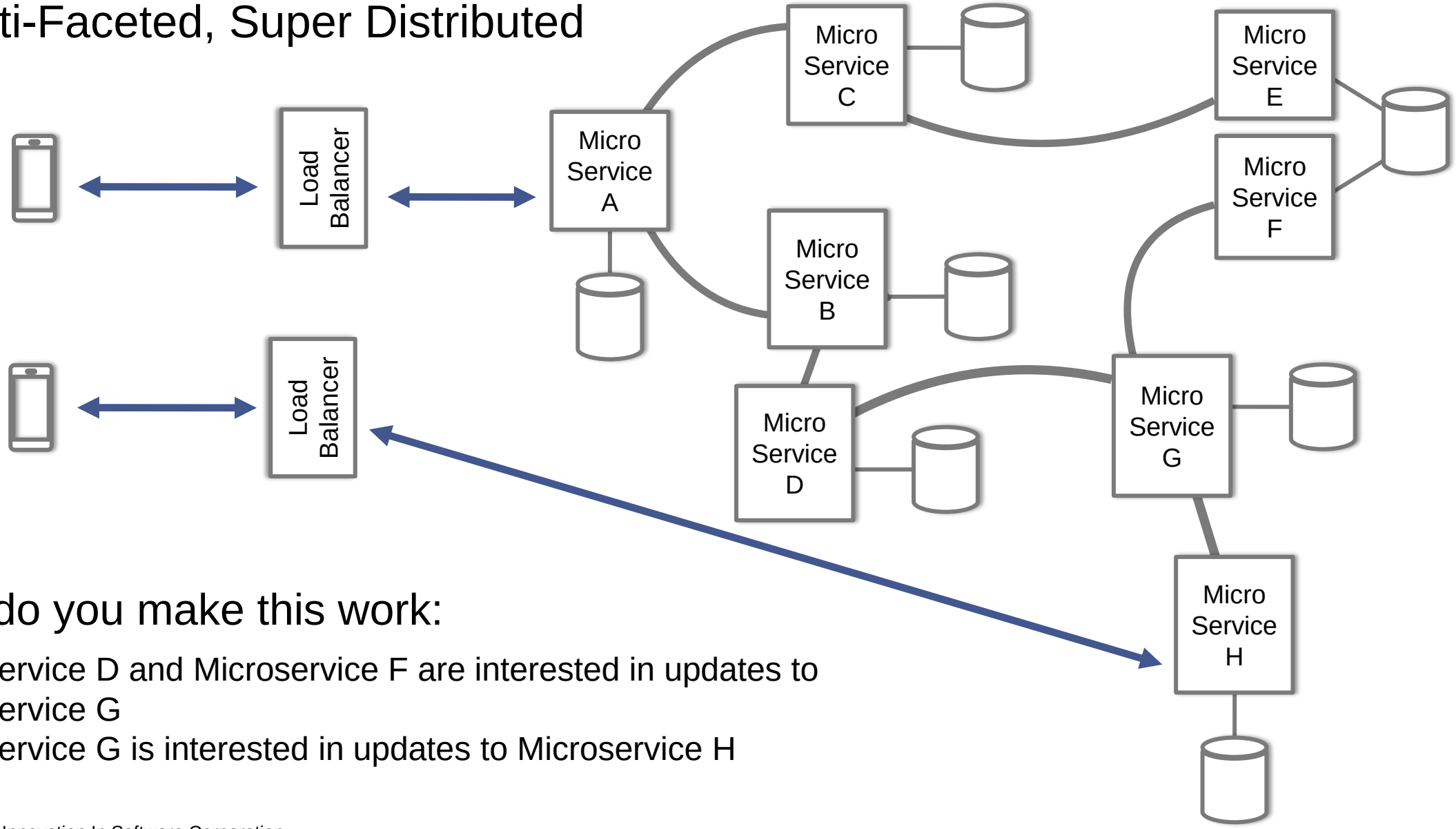
Load Balanced



© 2020 by Innovation In Software Corporation

DATA MANAGEMENT The Problem of Increasing Complexity

Multi-Faceted, Super Distributed

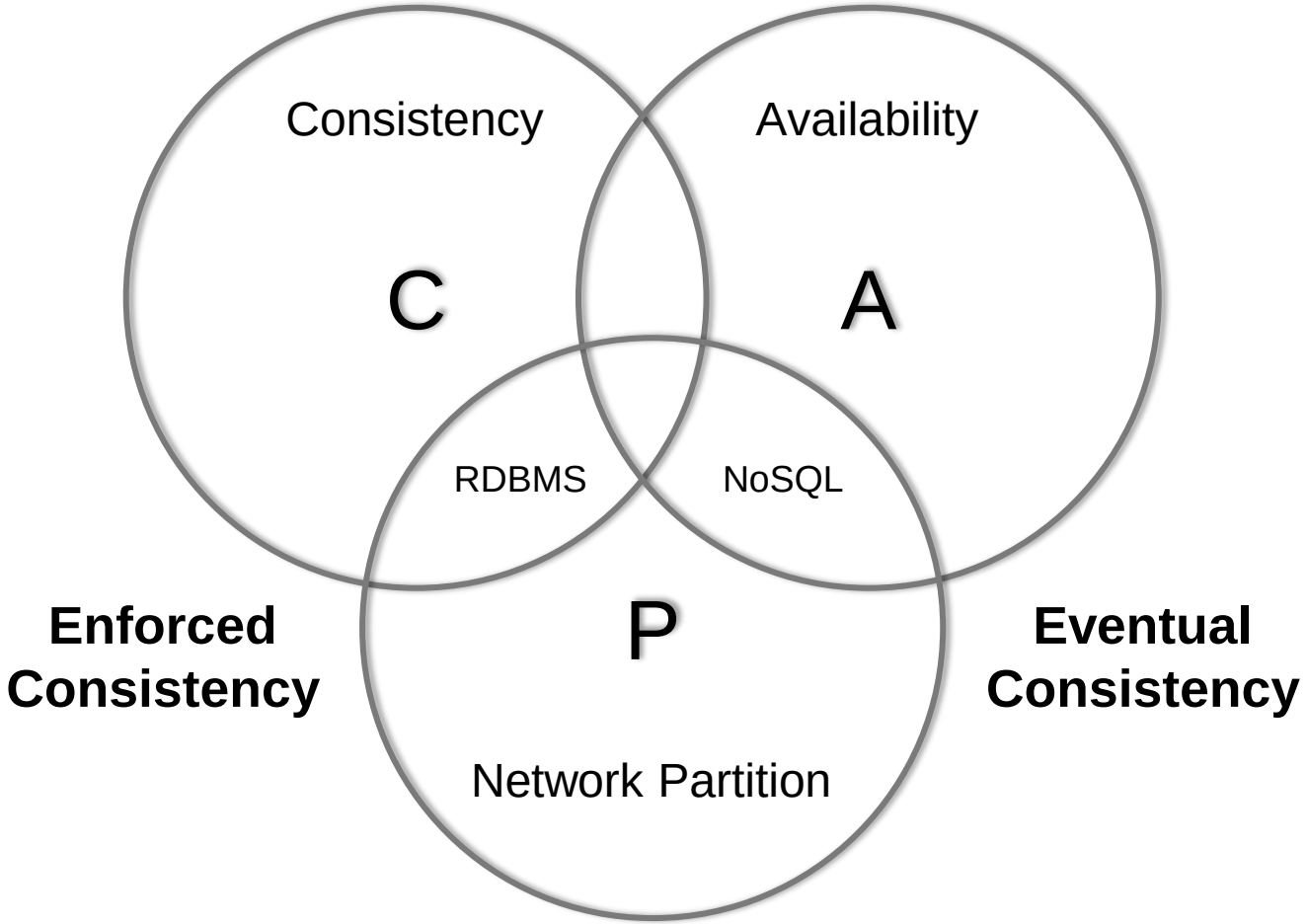


How do you make this work:

- Microservice D and Microservice F are interested in updates to Microservice G
- Microservice G is interested in updates to Microservice H

© 2020 by Innovation In Software Corporation

AN IMPORTANT CONCEPT CAP THEOREM



© 2020 by Innovation In Software Corporation

Consistency Models

Strict	Strict consistency is the strongest consistency model. Under this model, a write to a variable by any processor needs to be seen instantaneously by all processors.
Sequential	A write to a variable does not have to be seen instantaneously, however, writes to variables by different processors have to be seen in the same order by all processors.
Causal	As in sequential consistency, reads do not need to reflect changes instantaneously, however, they need to reflect all changes to a variable sequentially
Processor	A system exhibits processor consistency if the order in which other processors see the writes from any individual processor is the same as the order they were issued
PRAM (FIFO)	PRAM consistency says that there are no guarantees about the order in which different processes see writes, except that two or more writes from a single source must arrive in order, as though they were in a pipeline.
Bounded Staleness	Data consistency is bounded to a set of parameters, for example, a data store is configured to be at most 50 operations behind or 3 seconds delayed against the source of truth data mechanism.
Monotonic Read	Monotonic read consistency guarantees that after a process reads a value of data item x at time t, it will never see the older value of that data item
Monotonic Write	A write operation by a process on a data item X is completed before any successive write operation on X by the same process
Read My Writes	A value written by a process on a data item X will be always available to a successive read operation performed by the same process on data item X
Eventual	It defines that if no update takes a very long time, all replicas eventually become consistent

Strongest

Weakest

DATA MANAGEMENT Consistency Models

Strongest

		Consistency	Performance	Availability
Strong	See all previous writes.	excellent	poor	poor
Consistent Prefix	See initial sequence of writes.	okay	good	excellent
Bounded Staleness	See all “old” writes	good	okay	poor
Monotonic Read	See increasing subset of writes	okay	good	good
Read My Writes	See all writes performed by reader	okay	okay	okay
Eventual	See subset of previous writes.	poor	excellent	excellent

Weakest

Replicated Data Consistency Explained Through Baseball by Doug Terry

Microsoft Research SiliconValley, October 2011
https://www.microsoft.com/en-us/research/wp-content/uploads/2011/10/ConsistencyAndBaseballReport.pdf

© 2020 by Innovation In Software Corporation

DATA MANAGEMENT Consistency Models

Official scorekeeper	Read My Writes
Umpire	Strong Consistency.
Radio reporter	Consistent Prefix & Monotonic Reads
Sportwriter	Bounded Staleness
Statistician	Strong Consistency, Read My Writes
Stat Watcher	Eventual Consistency

Replicated Data Consistency Explained Through Baseball by Doug Terry

Microsoft Research SiliconValley, October 2011
https://www.microsoft.com/en-us/research/wp-content/uploads/2011/10/ConsistencyAndBaseballReport.pdf

© 2020 by Innovation In Software Corporation

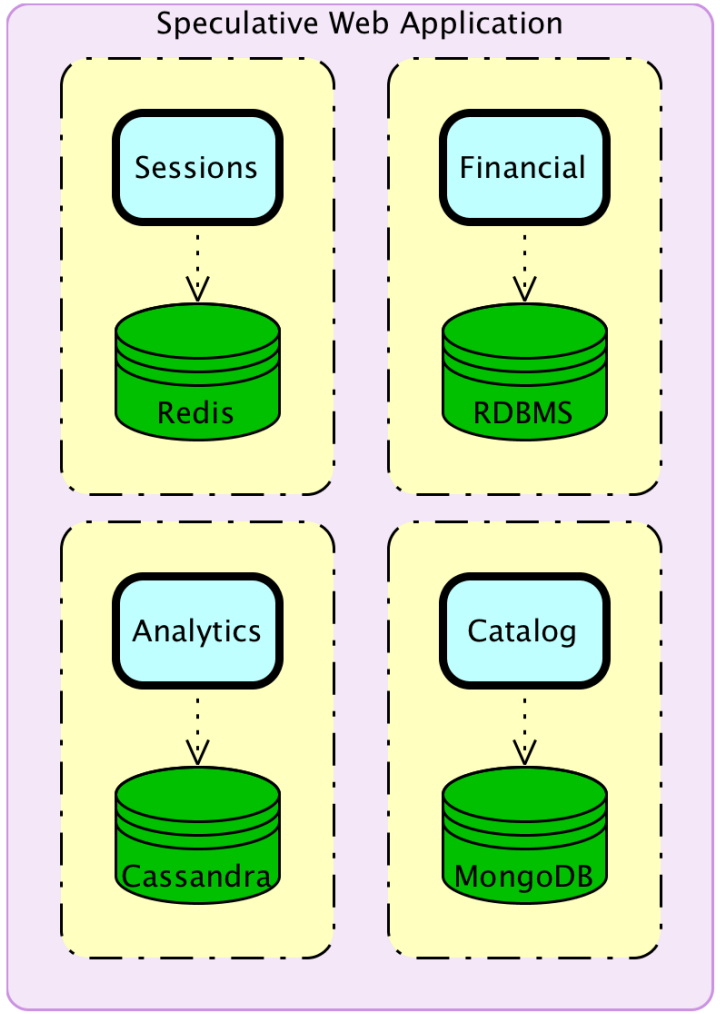
DATA MANAGEMENT

- Microservices are not isolated islands, they come in systems
- Data systems are about exposing data. Services are about hiding data
- The more shared data hidden inside a service boundary, the more complex the interface
- The more complex the interface the harder it will be to join datasets across different services
- Extracting whole dataset into mutable islands, the more they will diverge over time
- Better to take a decentralized approach to data and services with data-streams such as Kafka

© 2020 by Innovation In Software Corporation

POLYGLOT PERSISTENCE

A significant advantage of decentralized data management is the ability to take advantage of polyglot persistence



© 2020 by Innovation In Software Corporation

POLYGLOT PERSISTENCE

Different types of data have different storage requirements:

- Read/Write Balance (Some types of data have a very high write volume. This can require a different type of data store compared to data that has low write volume but high read volume.)
- Data Structure (Some types of highly structured data such as JSON documents may be better stored in a NoSQL database such as MongoDB, while flat relational objects may be more efficiently stored in a SQL database.)
- Data Querying (Some data may be accessible using a simple key value store, while other types of data may require advanced querying based on the values of multiple columns.)
- Data Lifecycle (Some data is temporary in nature and can be stored in a fast, in-memory store such as Redis or Memcached, while other data must be retained for all time and needs very durable storage on disk.)
- Data Size (Some data is made up of fairly uniform rows of consistent byte size, while other data may include large blobs that need to be stored in something like AWS S3.)

© 2020 by Innovation In Software Corporation

MICROSERVICE PATTERNS

© 2020 by Innovation In Software Corporation

MICROSERVICE ARCHITECTURE PATTERNS

Guiding for MOA/MSA

- **Reduce Cost:** MSA reducing the overall cost of designing, implementing, and maintaining IT services.
- **Increase Release Speed**: MOA increasing the speed from idea to deployment of services by allowing for smaller deployable units, use of platform as a service e.g. Istio/Knative/Kubernetes/Docker
- **Improve Resilience:** MSA will improve the resilience of our service network by the use of the monitoring/tracing/scaling mechanisms available in both service mesh and container orchestration platforms
- **Enable Visibility:** MSA support for better visibility on your service and network. The combination of tracing, logging, monitoring and visualization framing observability

© 2020 by Innovation In Software Corporation

MICROSERVICE ARCHITECTURE PATTERNS

Guiding principles for MOA/MSA

- Scalability
- Availability
- Resiliency
- Flexibility
- Independent, autonomous
- Decentralized governance
- Failure isolation
- Auto-Provisioning
- Continuous delivery through DevOps

© 2020 by Innovation In Software Corporation

MICROSERVICE ARCHITECTURE PATTERNS

Pattern Areas for MOA/MSA

- **Decomposition:** How to guide the conversion of brownfield applications AKA monoliths
- **Integration**: How to guide the binding of services to deliver business capabilities
- **Data Management:** How to inform the use of data management for cloud-native
- **Observability:** How to see across the landscape of MOA/MSA
- **Cross-Cutting Concerns:** How to create high performance delivery and operation of your service mesh

© 2020 by Innovation In Software Corporation

DECOMPOSITION: BUSINESS CAPABILITY

Define services by business capability

A business capability is based on MSA/MOA modeling. The service capability is something that a business does to create value.

A business capability often corresponds to a business object, e.g.

- **Order Management is responsible for orders**
- **Customer Management is responsible for customers**

© 2020 by Innovation In Software Corporation

DECOMPOSITION: BUSINESS CAPABILITY

© 2020 by Innovation In Software Corporation

DECOMPOSITION: SUBDOMAIN-DRIVEN

Subdomains can be classified as follows:

Core — key differentiator for the business and the most valuable part of the application

Supporting — related to what the business does but not a differentiator. These can be implemented in-house or outsourced

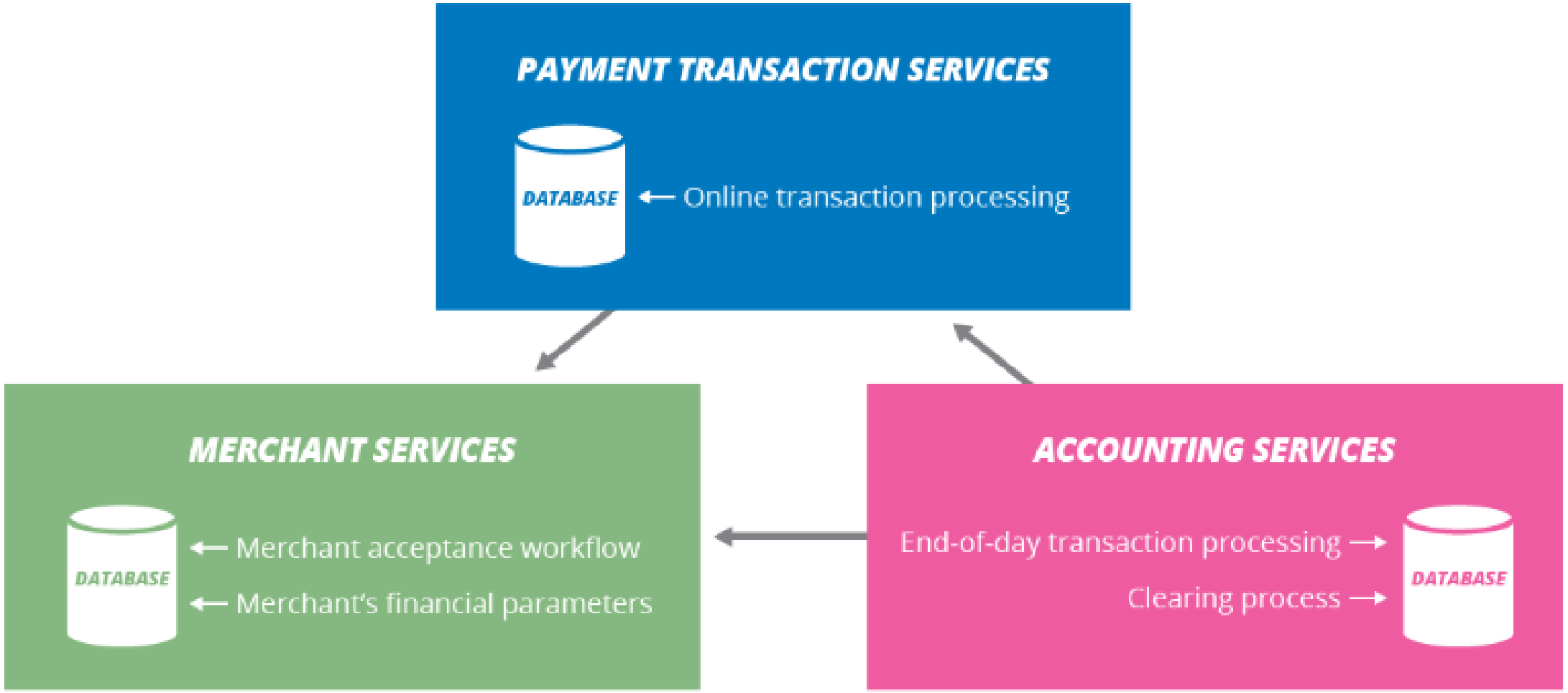
Generic — not specific to the business and are ideally implemented using off the shelf software

To decompose a business capability for Ordering services might include:

- **Product catalog**
- **Inventory**
- **Logistics**
- **Order management**

© 2020 by Innovation In Software Corporation

DECOMPOSITION: SUBDOMAIN-DRIVEN



© 2020 by Innovation In Software Corporation

DECOMPOSITION: TRANSACTIONS

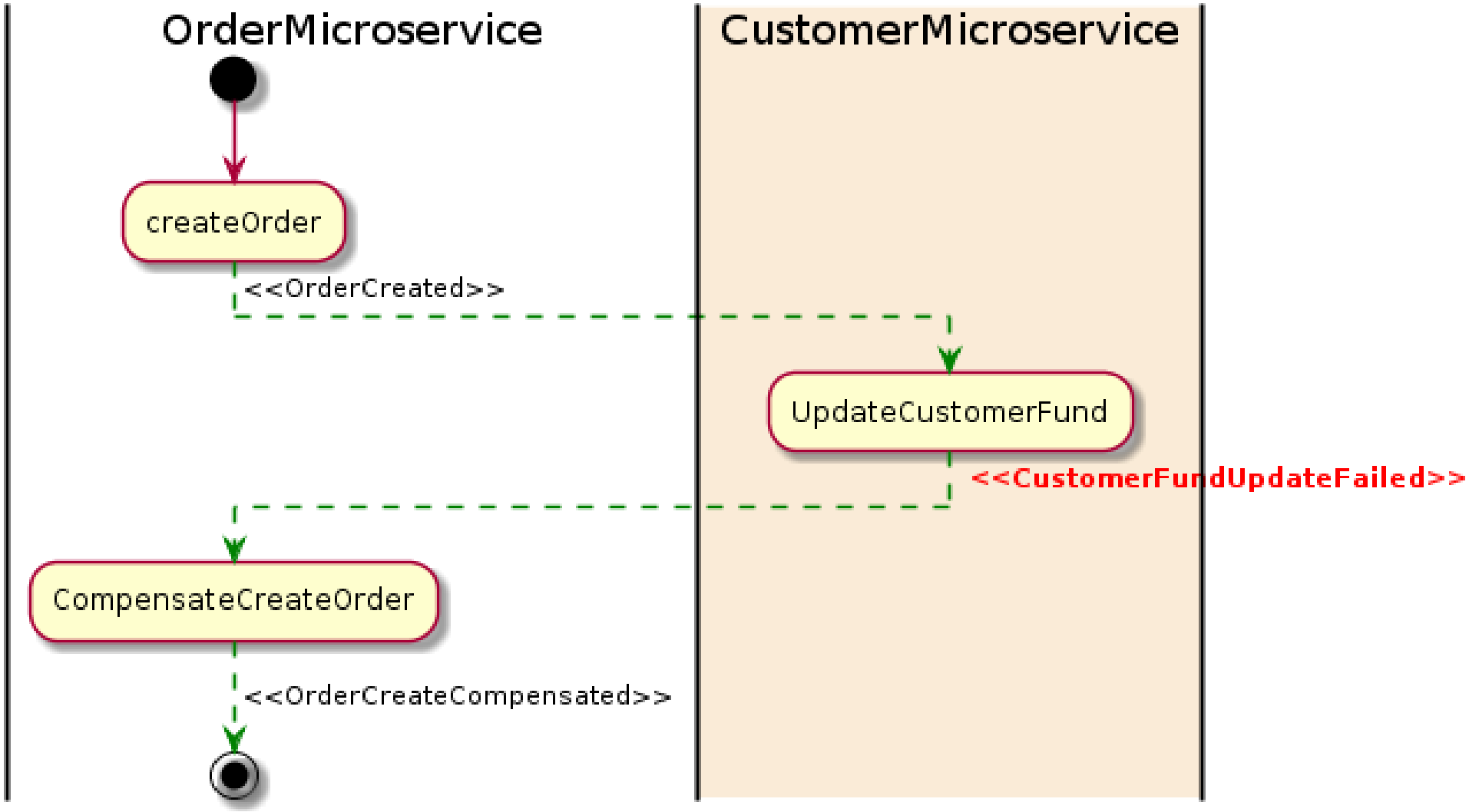
Transactions are based on phases

- **Prepare phase — during this phase, all participants of the transaction prepare for commit and notify the coordinator that they are ready to complete the transaction**
- **Commit or Rollback phase — during this phase, either a commit or a rollback command is issued by the transaction coordinator to all participants**

The complexity of transaction (2+pc) require careful consideration on the load that these services will be put under. Further, and service decomposed in this or an equally complex manner requires a well-planned performance/load testing approach

© 2020 by Innovation In Software Corporation

DECOMPOSITION: TRANSACTIONS



© 2020 by Innovation In Software Corporation

DECOMPOSITION: STRANGLER

Decomposing huge applications can be done more simply than big-bang by slowly recreating functionality as services alongside. That processing involves

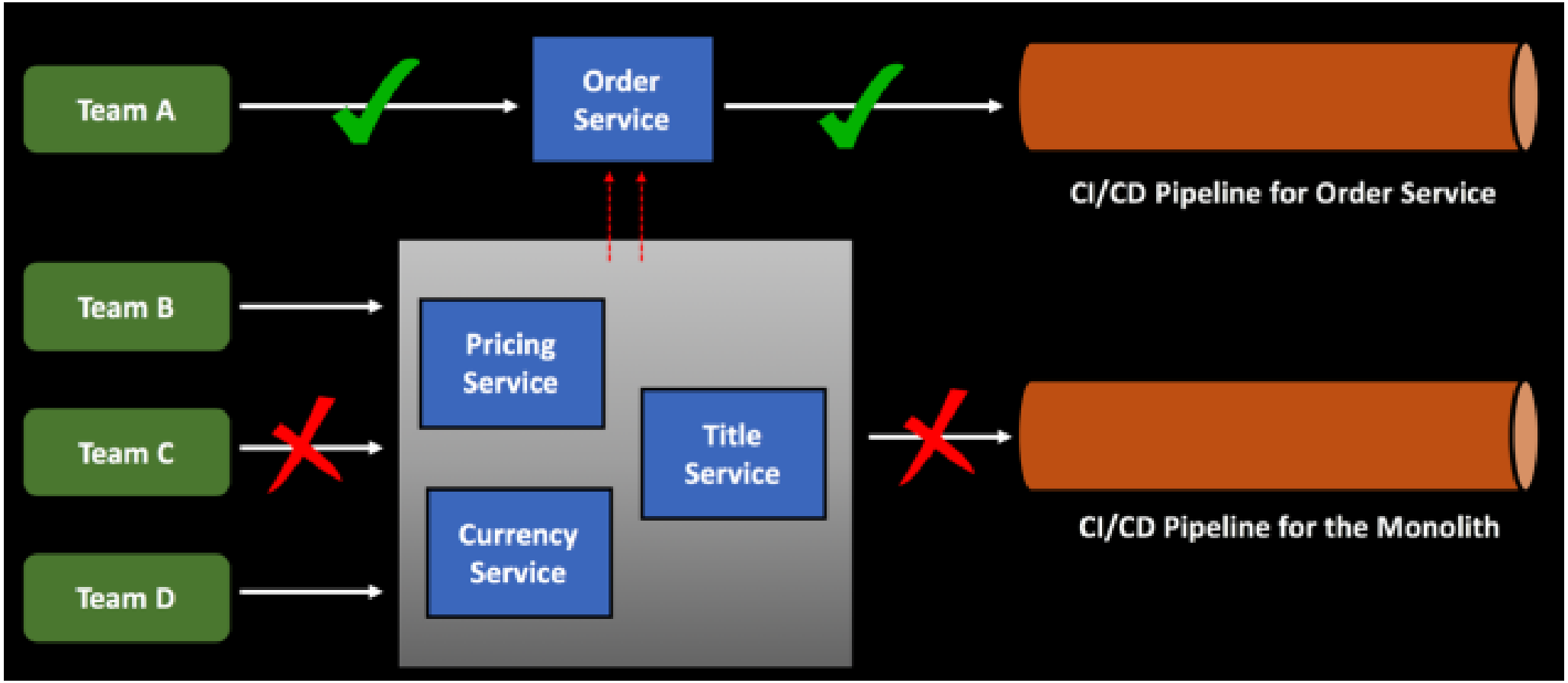
Transform — Create a parallel new development with modern cloud-native approaches

Coexist — Leave the existing application where it is for a time. Redirect from the existing application services to the new, as the incremental software engineering is completed

Eliminate — Remove the old functionality from the existing site, as the new functionality has been road-tested

© 2020 by Innovation In Software Corporation

DECOMPOSITION: STRANGLER

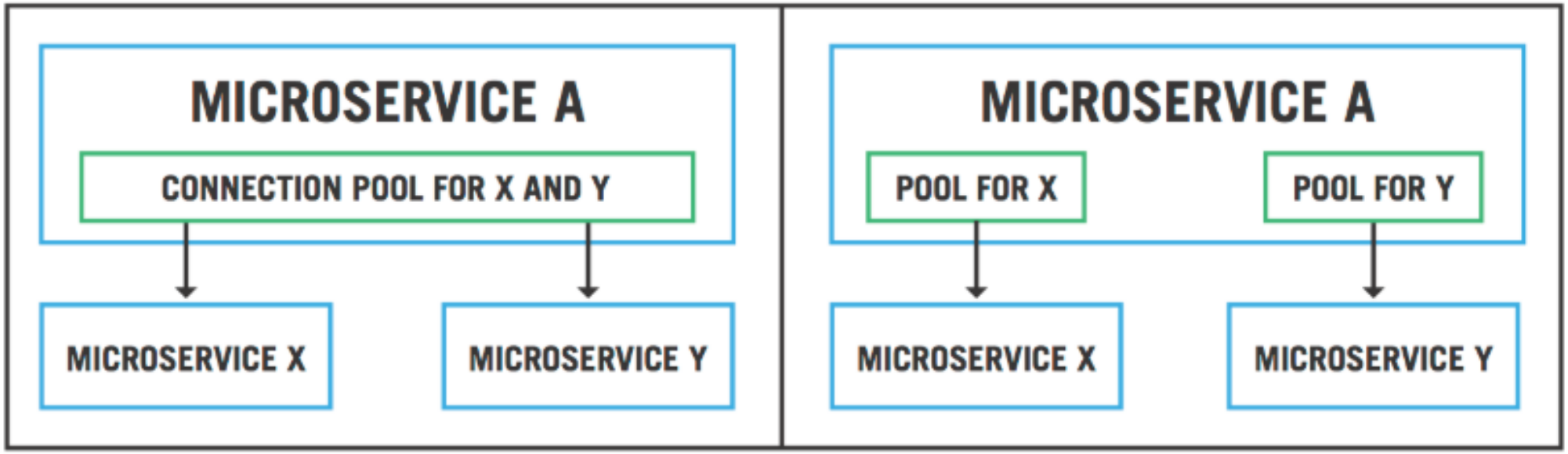


© 2020 by Innovation In Software Corporation

DECOMPOSITION: BULKHEAD

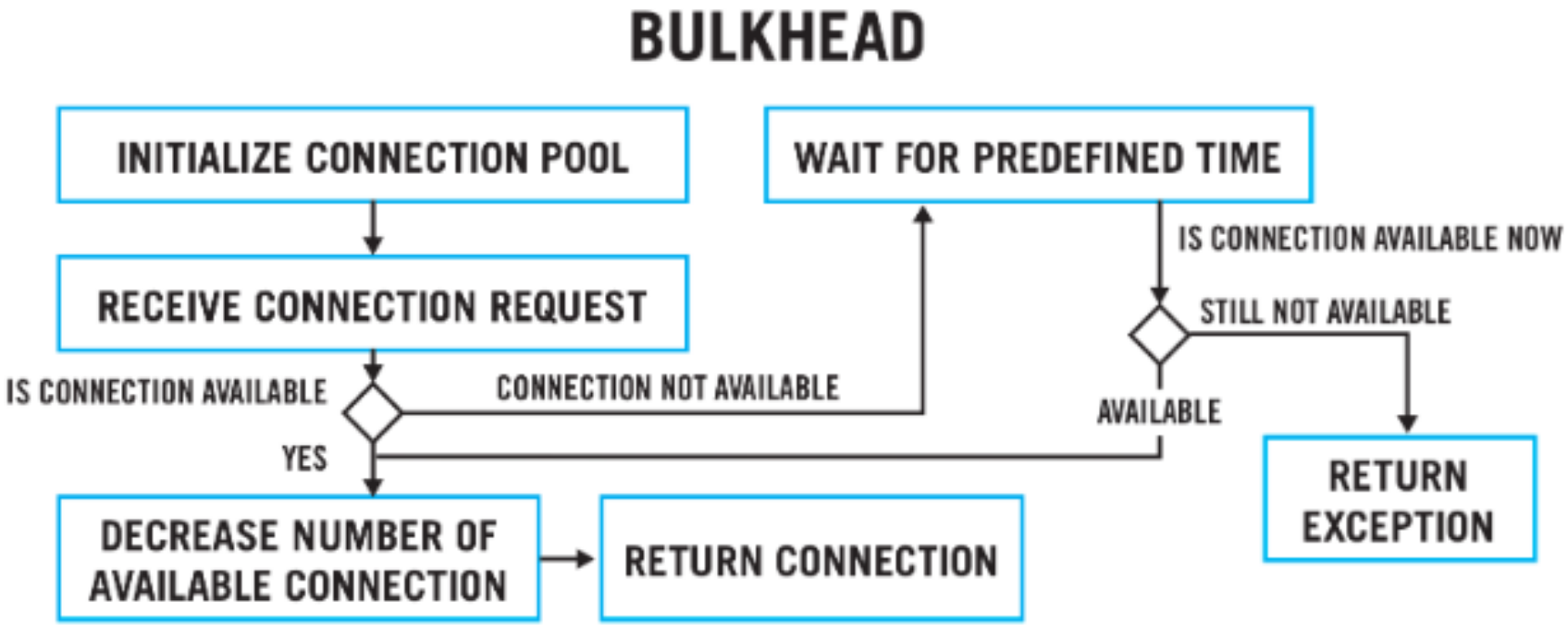
Isolation of application elements into pools so that if one fails, the others will continue to function.

Partition service instances into different groups, based on consumer load and availability requirements. This design helps to isolate failures, and allows you to sustain service functionality application service consumers, even during application stress or failure.



© 2020 by Innovation In Software Corporation

DECOMPOSITION: BULKHEAD



- Any application that needs to connect to a component will request a connection to that component, controlled by the individual bulkhead
- Check for availability for the component and allocate the connection or wait for a defined interval to re-check
- Alert/notify if a connection availability is impaired or never becomes available and creates failure situations

© 2020 by Innovation In Software Corporation

DECOMPOSITION: SIDECAR

- Deploy components of an application into a separate container to provide isolation and encapsulation
- Microservice applications are then composed of heterogeneous components and technologies through containerization
- Sidecars are attached to a parent microservice container and provide supporting features for the service
- Sidecar shares the same lifecycle as the parent application, is created and retired alongside the parent

© 2020 by Innovation In Software Corporation

DECOMPOSITION: SIDECAR

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sidecar-service-two
  labels:
    app: application
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sidecar-service-two
  template:
    metadata:
      labels:
        app: sidecar-service-two
spec:
    containers:
      - name: sidecar-app
        image: sidecar2-application:v1
        imagePullPolicy: Never
        ports:
          - containerPort: 6000
        env:
          - name: PORT
            value: "6000"
      - name: sidecar
        image: sidecar2-sidecar:v1
        imagePullPolicy: Never
        ports:
          - containerPort: 80
        env:
          - name: PORT
            value: "80"
          - name: TARGET
            value: "6000"
```

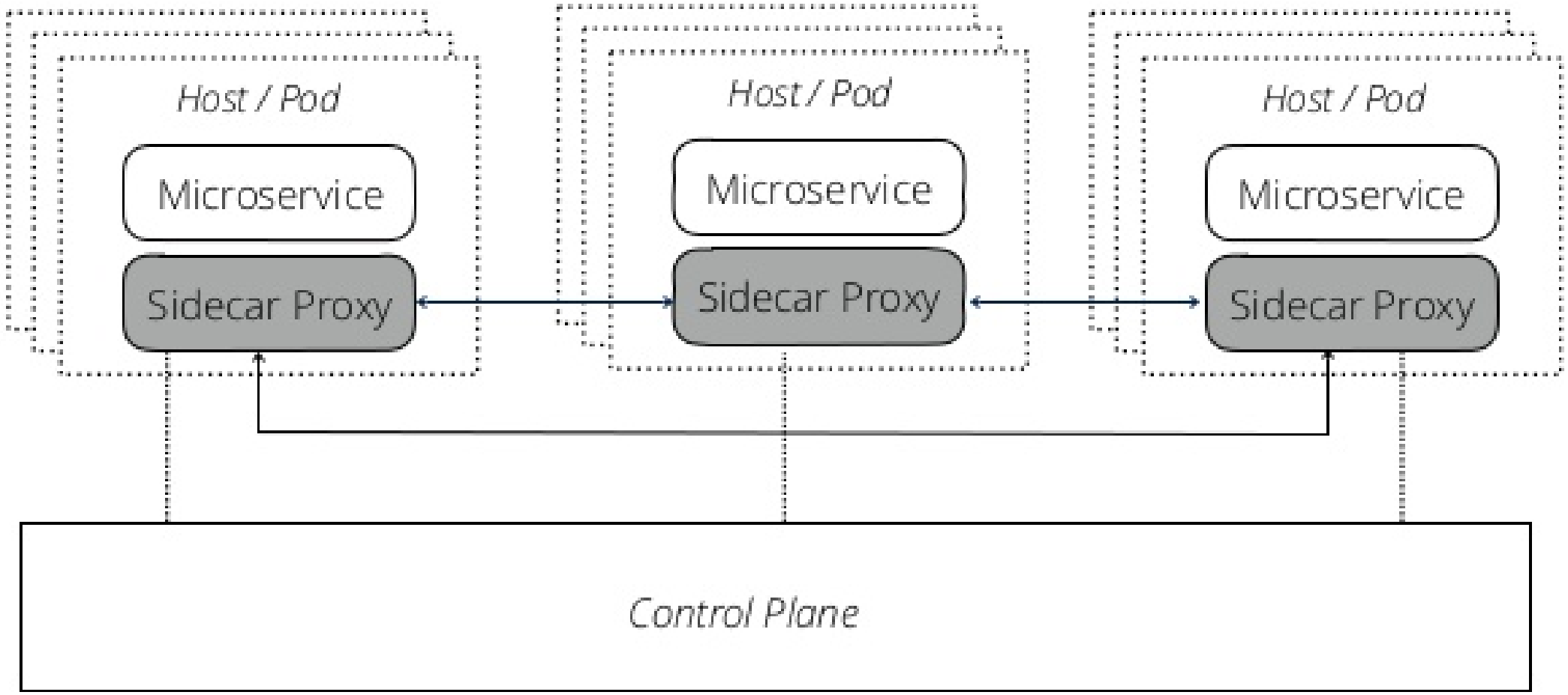
© 2020 by Innovation In Software Corporation

DECOMPOSITION: SIDECAR

```
FROM golang:alpine as builder
RUN apk update && apk upgrade && \
  apk add --no-cache bash git openssh gcc musl-dev
RUN mkdir -p /usr/distributed-patterns/sidecar-http-grpc/app
WORKDIR /usr/distributed-patterns/sidecar-http-grpc/app
COPY . .
RUN go get
RUN CGO_ENABLED=0 GOOS=linux go build main.go
FROM alpine
RUN mkdir -p /usr/app
WORKDIR /usr/app
COPY --from=builder /usr/distributed-patterns/sidecar-http-
grpc/app/main .
ENV PORT 80
CMD ["./main"]
```

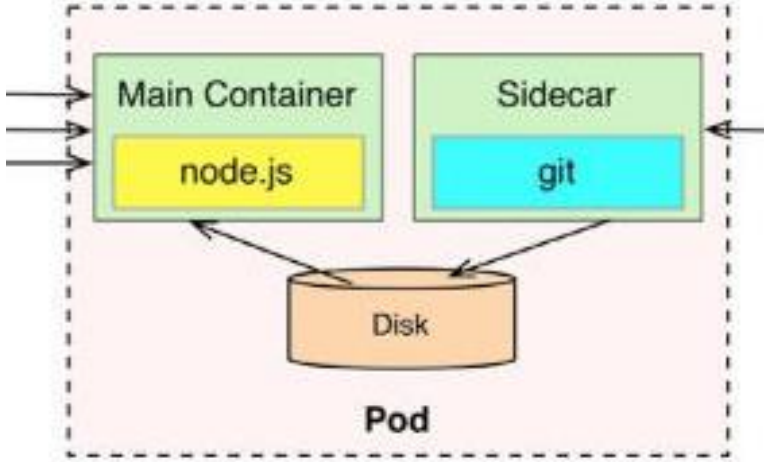
© 2020 by Innovation In Software Corporation

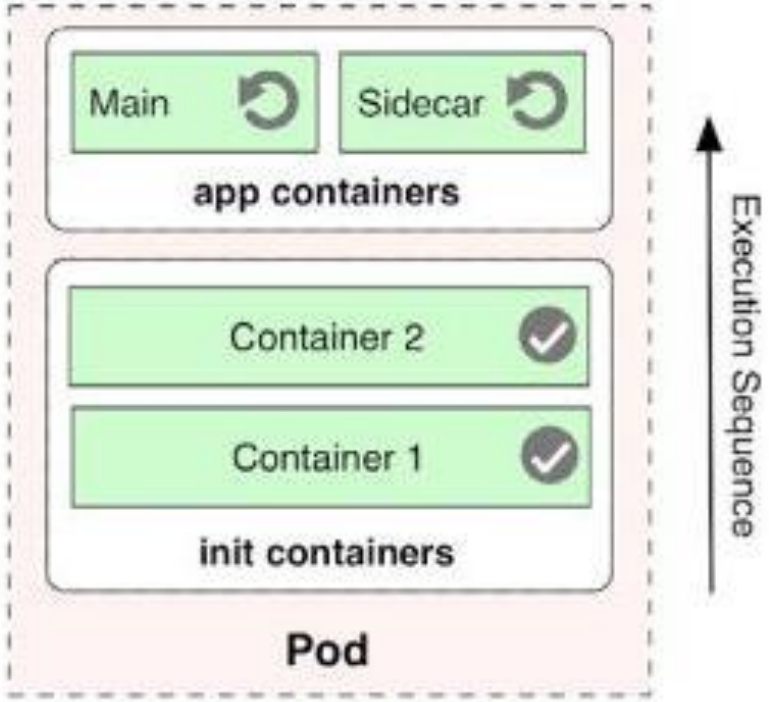
DECOMPOSITION: SIDECAR

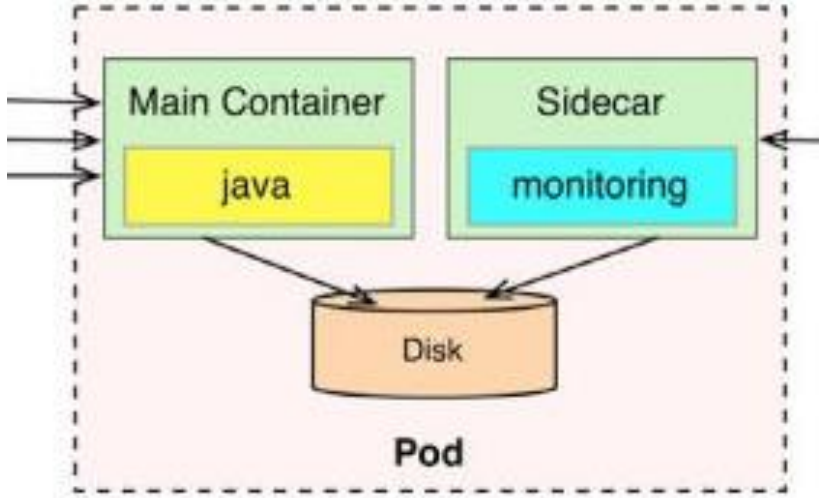


© 2020 by Innovation In Software Corporation

DECOMPOSITION: SIDECAR







© 2020 by Innovation In Software Corporation

INTEGRATION: API GATEWAY

- An API Gateway is the single point of entry for any microservice call
- Work as a proxy service to route a request to the concerned microservice
- Aggregate the results to send back to the consumer
- Create a fine-grained API for each specific type of client
- Convert the protocol request and respond
- Offload the authentication/authorization responsibility of the microservice

© 2020 by Innovation In Software Corporation

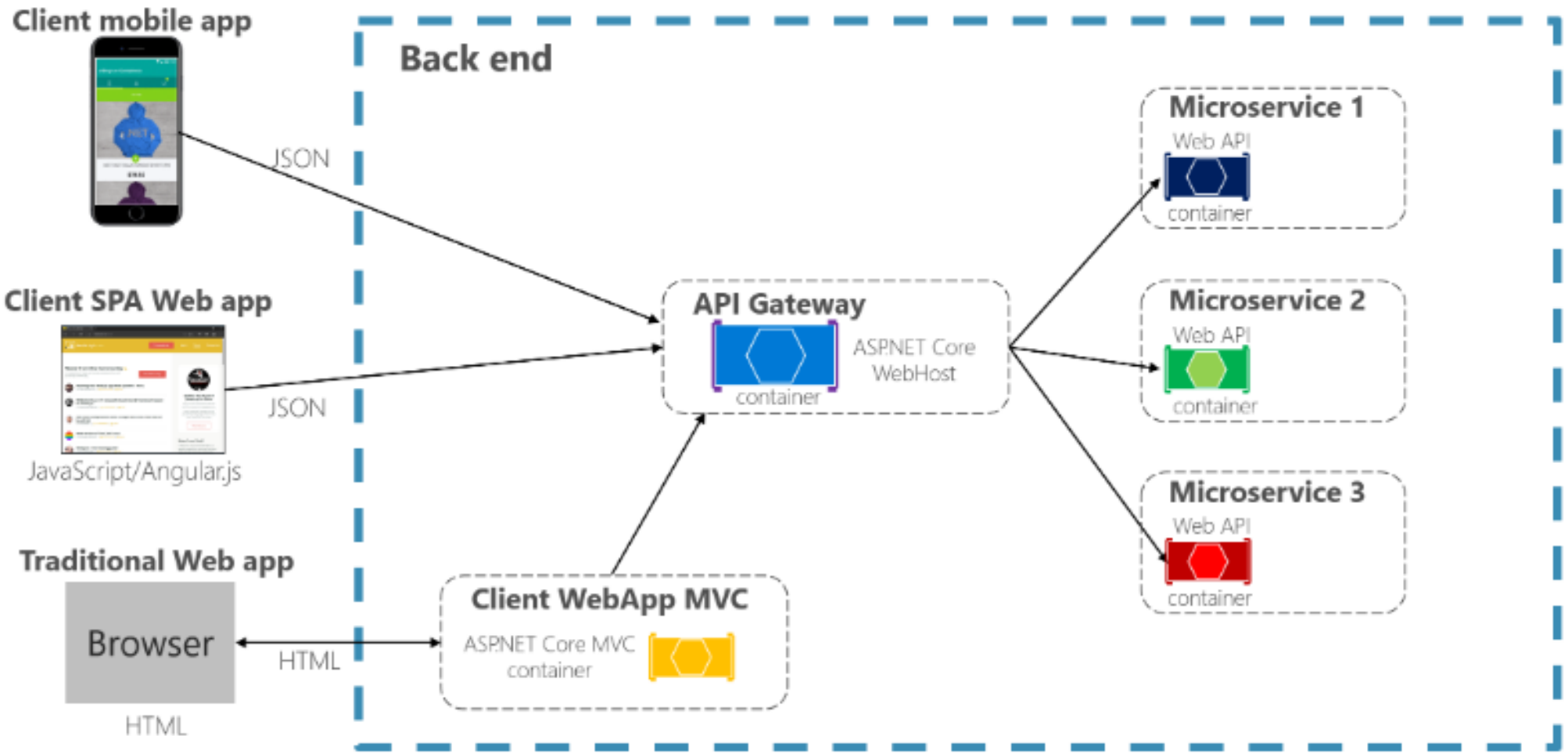
INTEGRATION: AGGREGATOR

Two commonly used approaches for aggregation of service data to be returned to a consumer

- **A composite microservice will make calls to all the required microservices, consolidate the data, and transform the data before sending back.**
- **An API Gateway can also partition the request to multiple microservices and aggregate the data before sending it to the consumer.**

© 2020 by Innovation In Software Corporation

INTEGRATION: PROXY



© 2020 by Innovation In Software Corporation

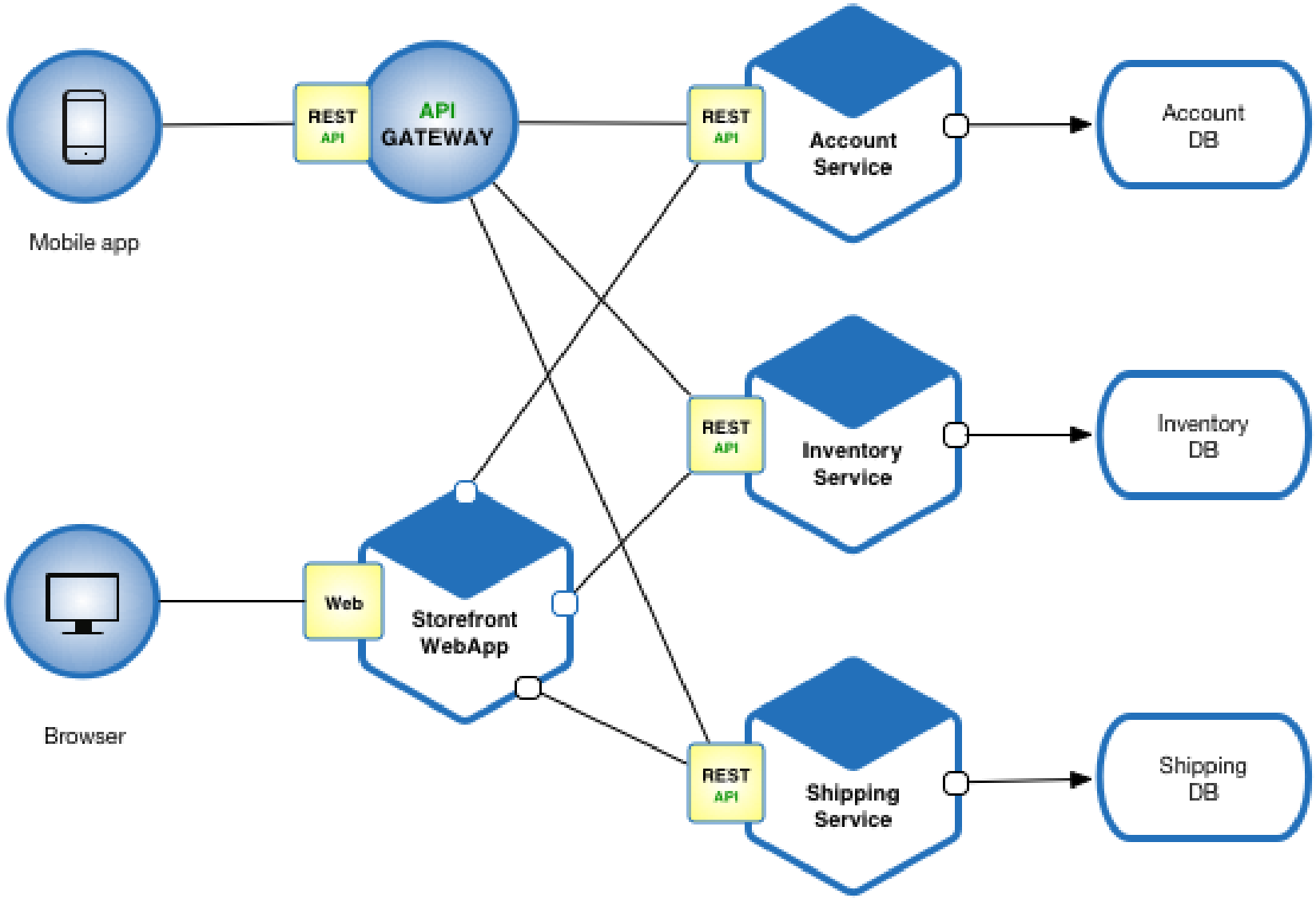
INTEGRATION: API GATEWAY

An API Gateway is the single point of entry for any microservice call

- **Works as a proxy service to route a request to the concerned microservice**
- **Aggregate the results to send back to the consumer**
- **Create a fine-grained API for each specific type of client**
- **Convert the protocol request and respond**
- **Offload the authentication/authorization responsibility of the microservice**

© 2020 by Innovation In Software Corporation

INTEGRATION: API GATEWAY



© 2020 by Innovation In Software Corporation

INTEGRATION: GATEWAY ROUTING

An API gateway or reverse proxying server like NGINX

- Implements some API operations by routing requests to the corresponding services
- Receives a request and consults a routing map that specifies which service to route the request
- A routing map might, for example, map an HTTP method and path to the HTTP URL of service

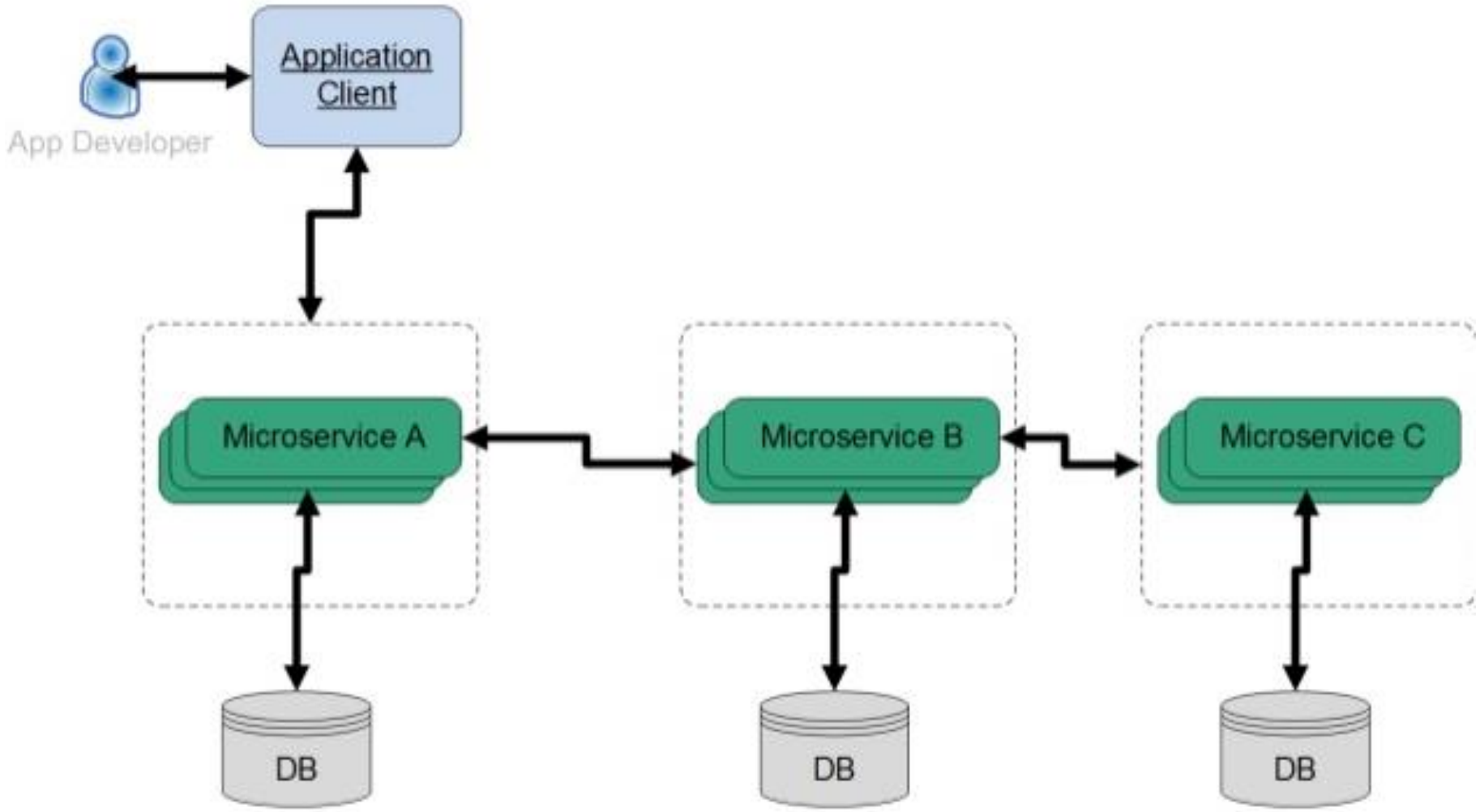
© 2020 by Innovation In Software Corporation

INTEGRATION: CHAINING

- Chained microservice design pattern provides the consolidated outcome/response to the consumer request
- The request received by a microservice-1, which is then communicating with microservice-2 and it may be communicating with microservice-3
- All these services are typically using synchronous calls

© 2020 by Innovation In Software Corporation

INTEGRATION: CHAINING



© 2020 by Innovation In Software Corporation

INTEGRATION: BRANCH

- Branch is used when a microservice may need to get the data from multiple sources including other microservices
- Branch microservice pattern is a mix of Aggregator & Chain design patterns and allows simultaneous request/response processing from two or more microservices
- Invoked microservice can be chains of microservices
- Branch integration pattern can also be used to invoke different chains of microservices, or a single chain, based your business capability being enabled

© 2020 by Innovation In Software Corporation

INTEGRATION: CLIENT-SIDE COMPOSITION

- UI designed as a skeleton with multiple sections/regions of the screen/page
- Each section will make a call to an individual backend microservice to pull the data
- Frameworks like AngularJS and ReactJS help to do this client-side composition more easily
- Screens are known as Single Page Applications (SPA) or Micro-UI
- Each team develops a client-side UI component, such an AngularJS directive, that implements the region of the page/screen for their service. A UI team is responsible for implementing the page skeletons that build pages/screens by composing multiple, service-specific UI components

© 2020 by Innovation In Software Corporation

DATA MANAGEMENT: CONSIDERATIONS

- UI designed as a skeleton with multiple sections/regions of the Services must be loosely coupled. They can be developed, deployed, and scaled independently
- Business transactions may enforce invariants that span multiple services
- Some business transactions need to query data that is owned by multiple services
- Backing services must sometimes be replicated and shared in order to scale
- Different services have different data storage, persistence and lifecycle requirements

© 2020 by Innovation In Software Corporation

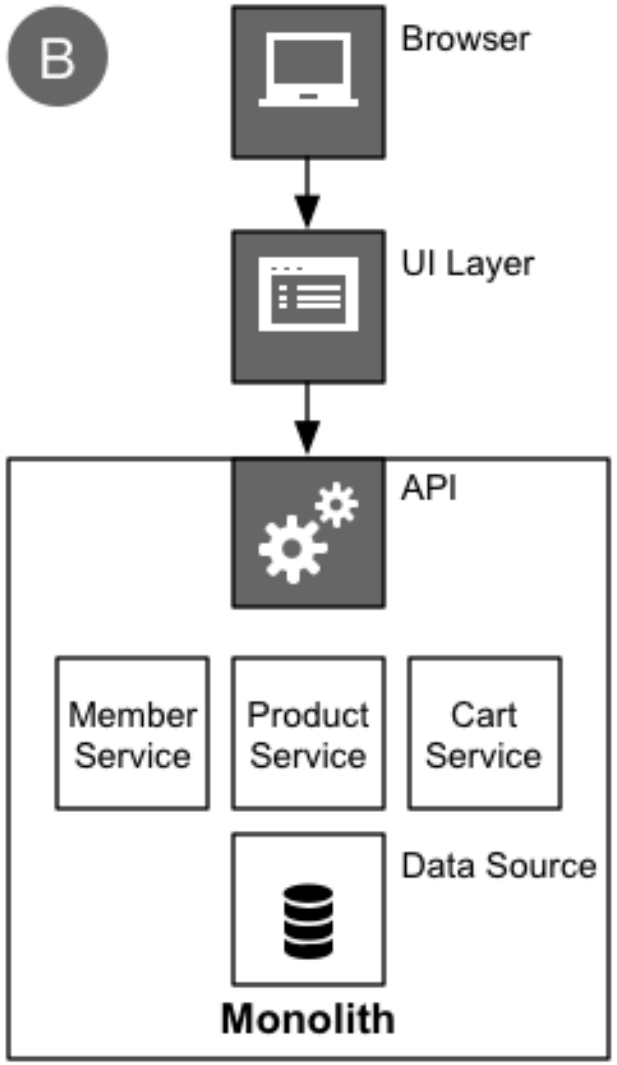
DATA MANAGEMENT: BACKING STORE PER SERVICE

- UI designed as a skeleton with multiple sections/regions
- One database per microservice
- Backing service private to that service only
- Accessed by the microservice API only
- Data cannot be accessed by other services directly
- Options for relational databases
  - private-tables-per-service
  - schema-per-service
  - database-server-per-service
- Options for messaging data management
  - cluster-per-service
  - queue-per-service
  - topic-per-service

© 2020 by Innovation In Software Corporation

DATA MANAGEMENT: BROWNFIELD SHARED DATA

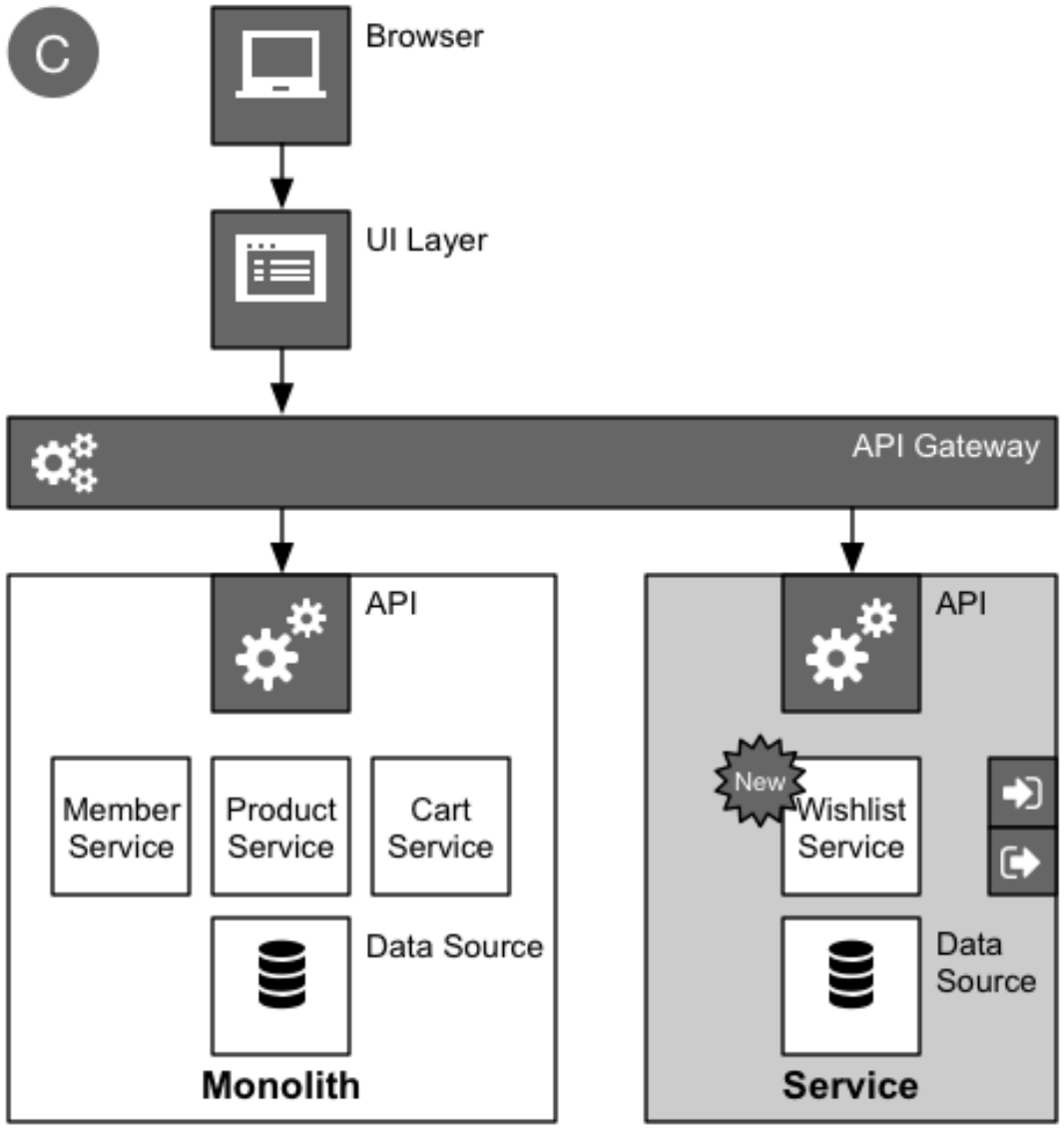
- In migration work it may be useful to consider initially microservices with a shared data management plane
- This allows for separation via MVC and a service layer, façade, or mesh without the initial complexity of a complete rewrite
- Retains investment in normalized schema with relations, stored procedures, …



© 2020 by Innovation In Software Corporation

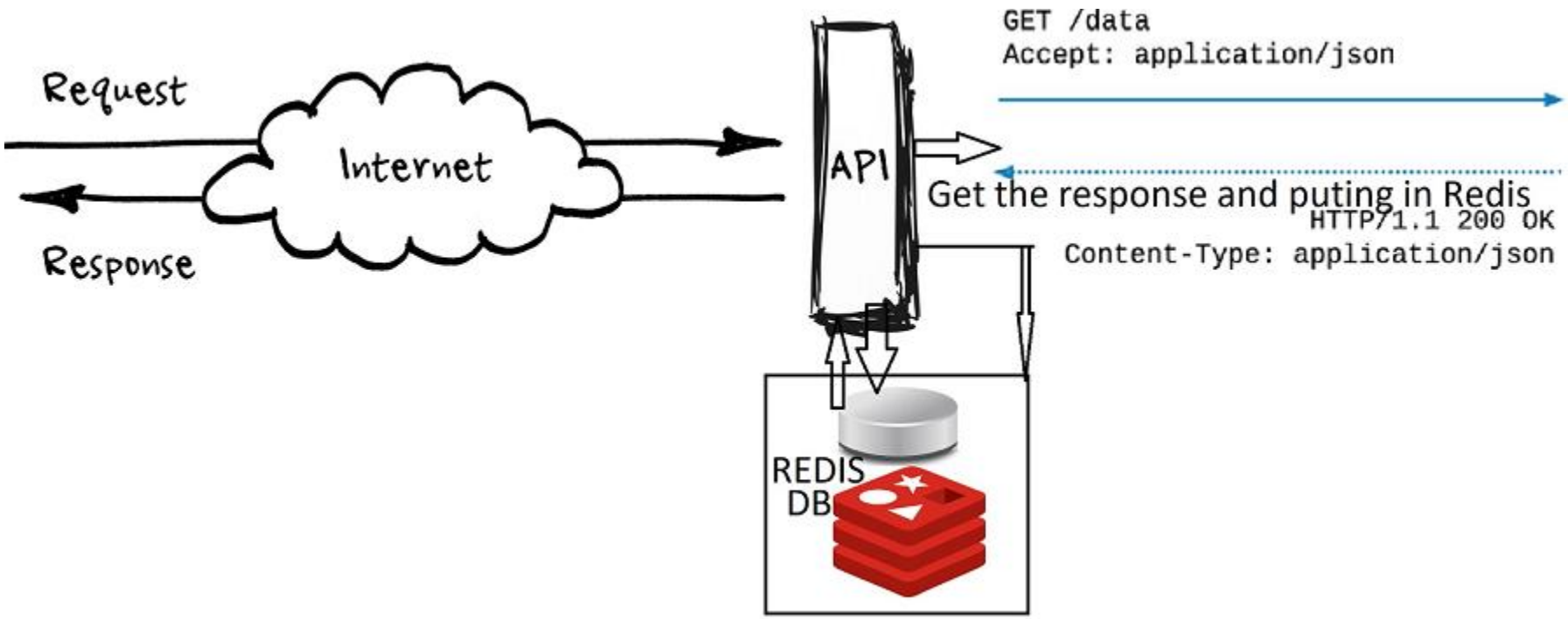
DATA MANAGEMENT: BROWNFIELD SERVICE SEPARATION

- New features added to a monolith have separated services, but may continue to rely on services tied to the shared database
- New feature uses isolated data management for the service(s)
- This gradual reduction allows for teams to learn the microservices environment and patterns while keeping the lights on for the existing monolith



© 2020 by Innovation In Software Corporation

DATA MANAGEMENT: DATA CACHING



- **Data that is more frequently read than updated/written can be cached in Redis or Elasticsearch allowing for significant performance improvement for your microservices architecture**

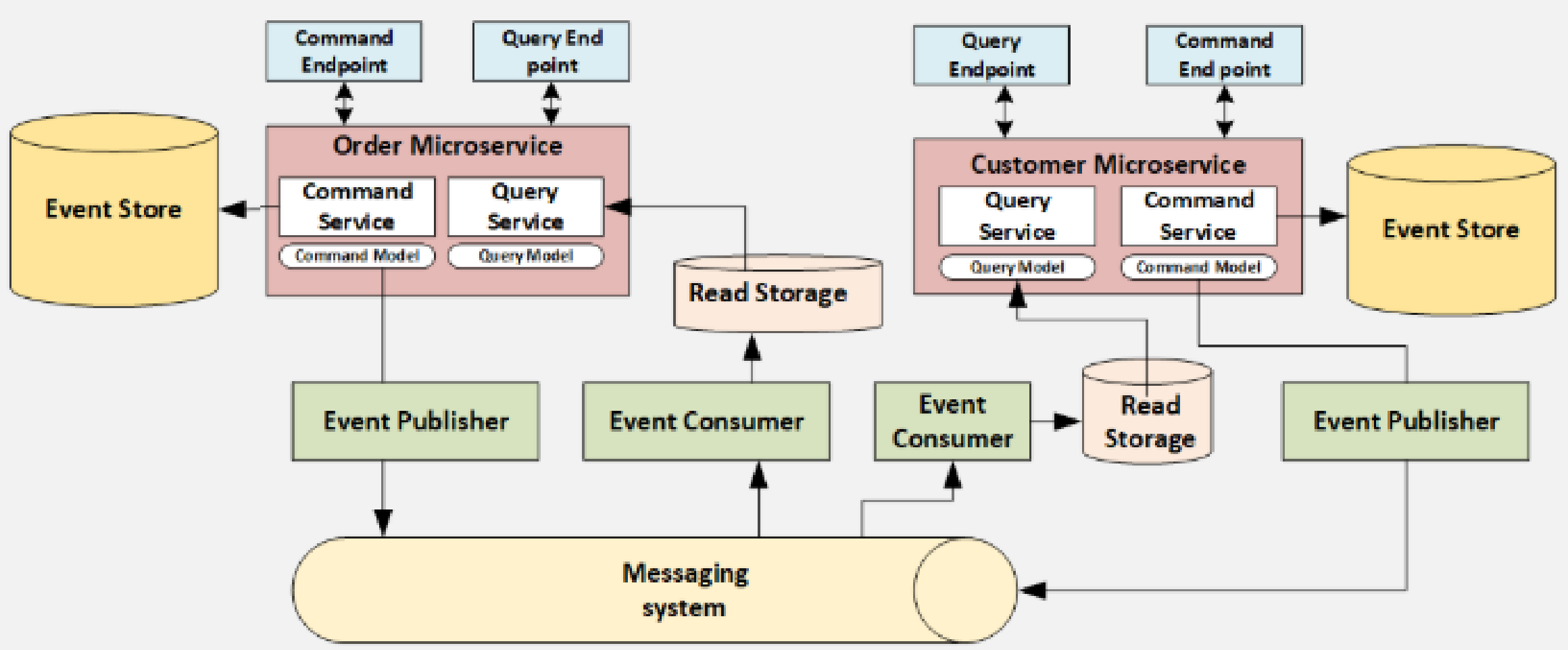
© 2020 by Innovation In Software Corporation

DATA MANAGEMENT: COMMAND QUERY RESPONSIBILITY SEGREGATION (CQRS)

- CQRS suggests splitting the application into two parts — the command side and the query side.
  - The command side handles the Create, Update, and Delete requests
  - The query side handles the query part by using the materialized views
- The event sourcing pattern is generally used along with it to create events for any data change.
- Materialized views are kept updated by subscribing to the stream of events.

© 2020 by Innovation In Software Corporation

DATA MANAGEMENT: CQRS & EVENT SOURCING



© 2020 by Innovation In Software Corporation

Chaos Engineering

Concepts

© 2020 by Innovation In Software Corporation

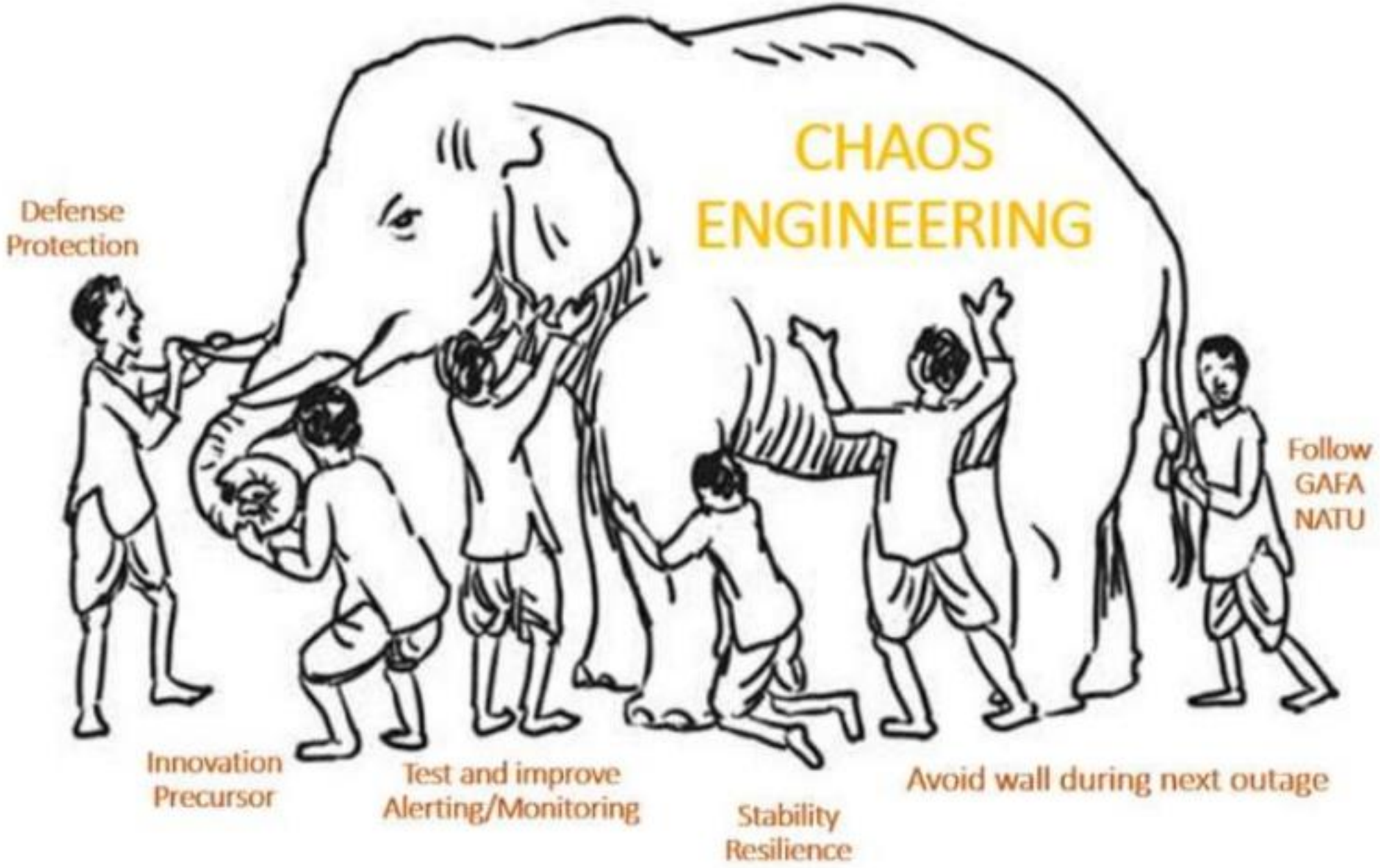
© 2020 by Innovation In Software Corporation

WHAT IS CHAOS ENGINEERING MORE SIMPLY?

Experiment to ensure that the impact of failure is mitigated

WHY DO CHAOS IN PRODUCTION?

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.



© 2020 by Innovation In Software Corporation

MICROSERVICE ELEMENTS OF THE CHAOS TOOLKIT

```
   "version": "1.0.0",
    "title": "System is resilient to provider's
failures",
    "description": "Can our consumer survive gracefully
a provider's failure?",
    "tags": [
        "service",
        "kubernetes",
        "spring"
    ],
    "steady-state-hypothesis": {
        "title": "Services are all available and
healthy",
        "probes": [
            {
                "type": "probe",
                "name": "all-services-are-healthy",
                "tolerance": true,
                "provider": {
                    "type": "python",
                    "module": "chaosk8s.probes",
                    "func": "all_microservices_healthy"
```

The key concepts of the Chaos Toolkit are `Experiments`, `Steady State Hypothesis` and the experiment's `Method`. The Method contains a combination of `Probes` and `Actions`.

```
    "method": [
        {
            "type": "action",
            "name": "stop-provider-service",
            "provider": {
                "type": "python",
                "module": "chaosk8s.actions",
                "func": "kill_microservice",
                "arguments": {
                    "name": "my-provider-service"
                }
            },
            "pauses": {
                "after": 10 } },
        {   "ref": "all-services-are-healthy"
},
        {   "type": "probe",
            "name": "consumer-service-must-still-
respond",
            "provider": {
                "type": "http",
                "url":
"http://192.168.42.58:31018/invokeConsumedService"
            }
        }
    ],
    "rollbacks": []
```

The key concepts of the Chaos Toolkit are `Experiments`, `Steady State Hypothesis` and the experiment's `Method`. The Method contains a combination of `Probes` and `Actions`.

DEFINING STEADY STATE PROBES

```
{
    "type": "probe",
    "name": "the-sunset-service-must-be-running",
    "tolerance": true,
    "provider": {
        "type": "python",
        "module": "os.path",
        "func": "exists",
        "arguments": {
            "path": "sunset.pid"
        }
    }
}
```

Here we define a Probe element, which we've added to our steady state Probes list that checks for a file on the filesystem.

Microservice SRE

Toolchains

© 2020 by Innovation In Software Corporation

© 2020 by Innovation In Software Corporation

PLATFORM TESTING TOOLS – KUBE MONKEY

An implementation of Netflix's Chaos Monkey for Kubernetes clusters

github.com/asobti/kube-monkey

© 2020 by Innovation In Software Corporation

PLATFORM TESTING TOOLS – POWERFULSEAL

A powerful testing tool for OpenStack Kubernetes clusters

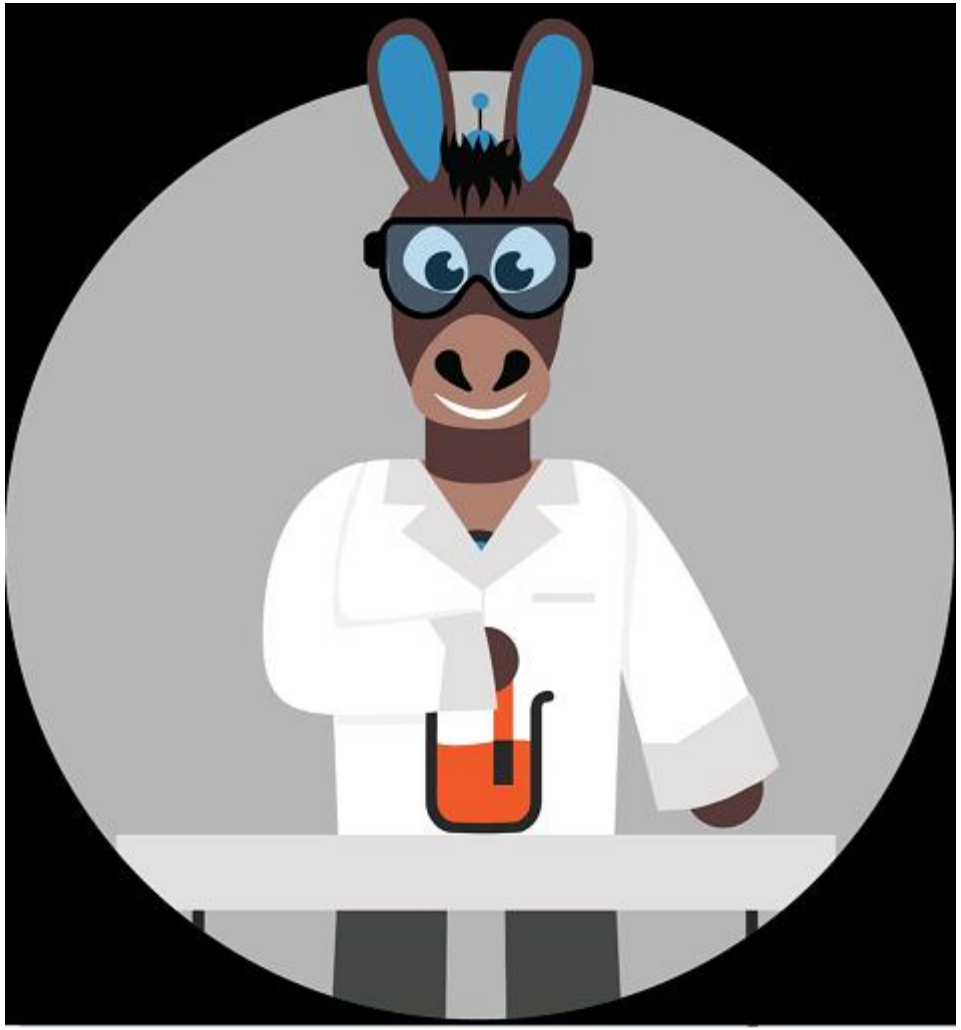
github.com/bloomberg/powerfulseal

© 2020 by Innovation In Software Corporation

PLATFORM TESTING TOOLS – LITMUS

Chaos engineering tool for stateful workloads on Kubernetes, hopefully without learning curves.

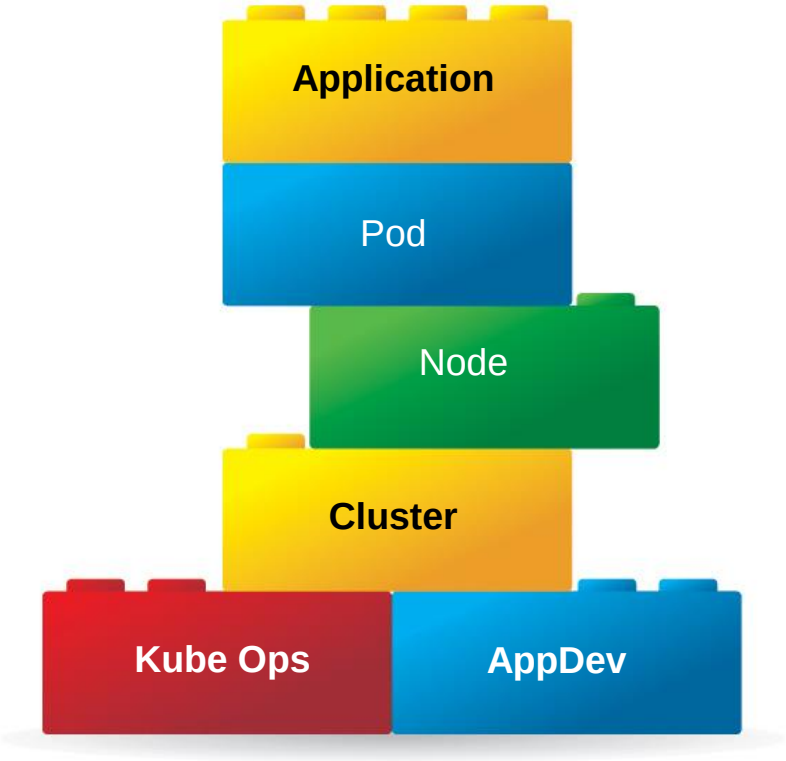
github.com/openebs/litmus



© 2020 by Innovation In Software Corporation

HOW TO FOCUS ON THE POD RESILIENCY TO BUILD CHAOS ENGINEERING EXPERIMENTS

In many system contexts there is a dividing line between those who are responsible for managing a Kubernetes cluster (including all the realcosts associated with these resources), let's call those the Kube Admin DevOps, and those trying to deploy and manage applications upon the cluster, the Software Engineering DevOps.



Our First Chaos Experiment

© 2020 by Innovation In Software Corporation

Our First Chaos Experiment

In this scenario you will learn how to write a simple Chaos Toolkit experiment and then use the chaos run command to execute your experiment.

This experiment demonstrates the good and the bad of pre-built environments, labs, and materials.

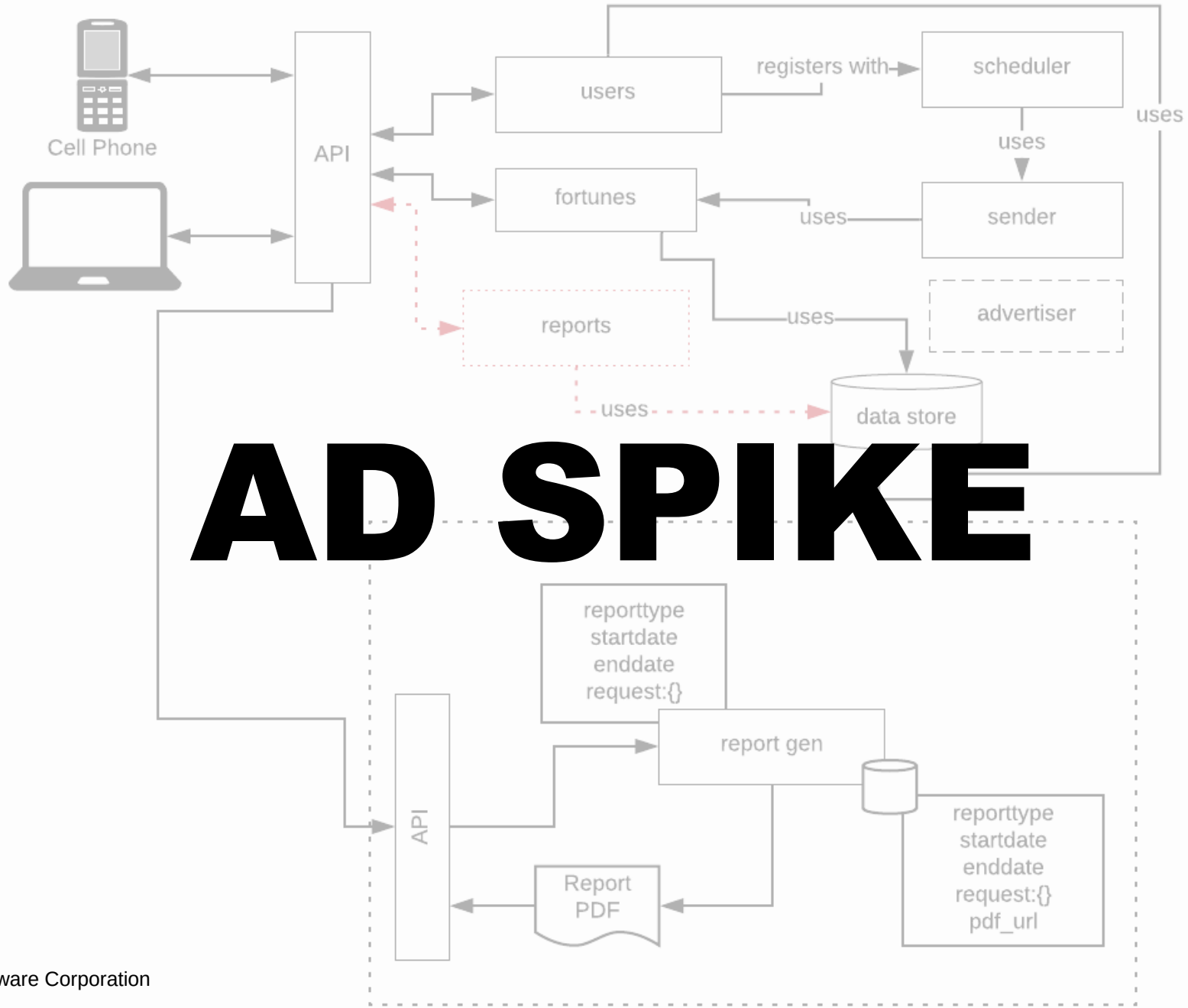
You'll note an error in the beginning of this experiment that requires us to execute the following prior to the "**chaos run**" command.

```
export LC_ALL=C.UTF-8
export LANG=C.UTF-8
```

https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/simple-experiment

© 2020 by Innovation In Software Corporation

AD SPIKE



© 2020 by Innovation In Software Corporation

Microservice Spike Session

Designing and Implementing the `advertiser` Microservices

We'll divide ourselves into groups by random selection. Each group will design and possibly implement the `advertiser` feature of Fortune Cookies as a microservice(s) using the concepts, ideas and techniques presented over the last two days.

Each group will do a 10 minute presentation of its work starting an hour before the end of class. Do as much as you can. It's about progress, not perfection. The objective is to experience the thought processes and creative challenges that are intrinsic to designing and implementing a microservice architecture.

This is a difficult exercise. The instructor is available at any time to provide additional instruction, guidance or clarification.

© 2020 by Innovation In Software Corporation

INDIVIDUAL/GROUP EXERCISE

Fortune Cookie

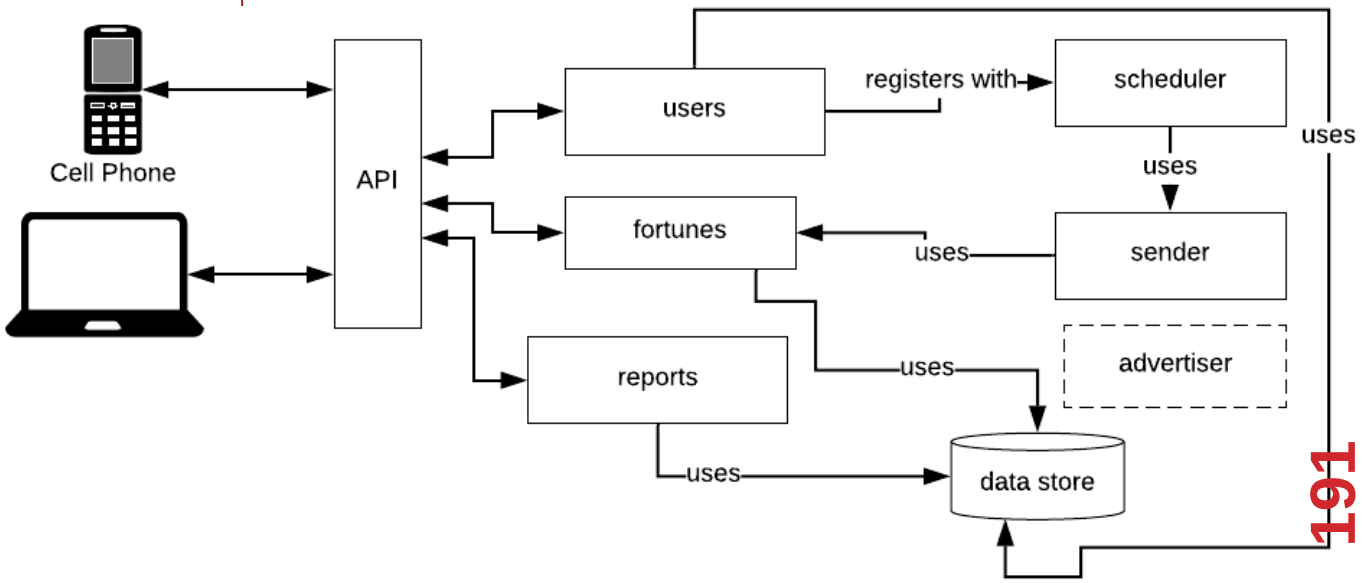
A marketing company wants your group to design an application that will deliver a “fortune cookie” to willing recipients. All that’s required is that the user register with the app using either Facebook, Google or LinkedIn credentials. Or the user can custom register by providing standard profile information. The fortune cookie text is provided by the Fortunes service. Each fortune cookie is accompanied by an advertisement from “Ad Auction” service. However, users need to be provided with with the option to not receive advertisements. Also, a feature needs to be implemented supporting a Fortune Cookie presence on Twitter by sending Fortune Cookie tweets out at configurable intervals. Fortune Cookie needs to work on iPhone, Android, standard web pages, Alexa and SMS. In addition, Fortune Cookie needs to provide usage and performance metrics on demand to administrative personnel. Fortune cookie is a free application.

Objective

Design and possibly implement `advertiser` as a microservice(s).

Ad Auction (`advertiser`)

- Keeps an inventory of ads for sale
- Conducts the auctioning of ads to the highest bidder
- Delivers the ad to the auction winner



© 2020 by Innovation In Software Corporation

INDIVIDUAL/GROUP EXERCISE HELPFUL HINTS

- Define the semantic definition of your microservice. In others words, what is the concern of the service? At a logical level, what is it supposed to do?
- Define the service boundary
- Determine if your microservice will be synchronous, asynchronous, or both?
- Define how your microservice will be be exposed to consumers.
- Describe the structure/format of the data the microservice will consume.
- Describe the structure/format of the data the microservice will emit.
- Describe the guarantee of the accuracy of your data

Ad Auction

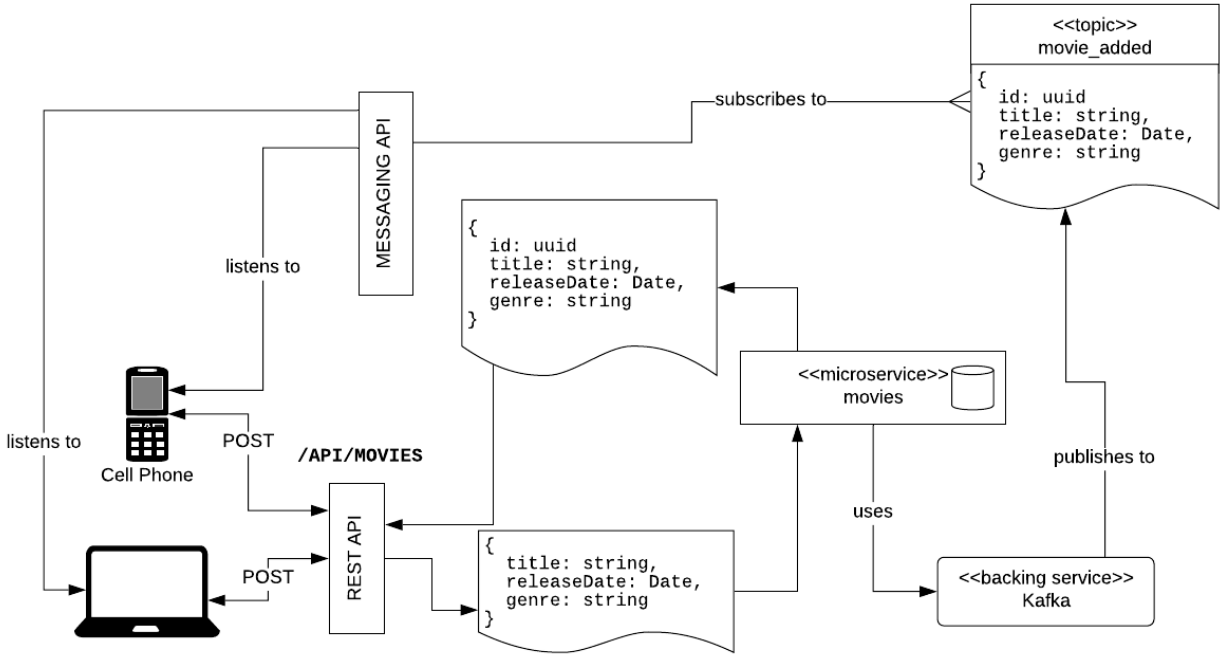
- Keeps an inventory of ads for sale
- Conducts the auctioning of ads to the highest bidder
- Delivers the ad to the auction winner

Objective

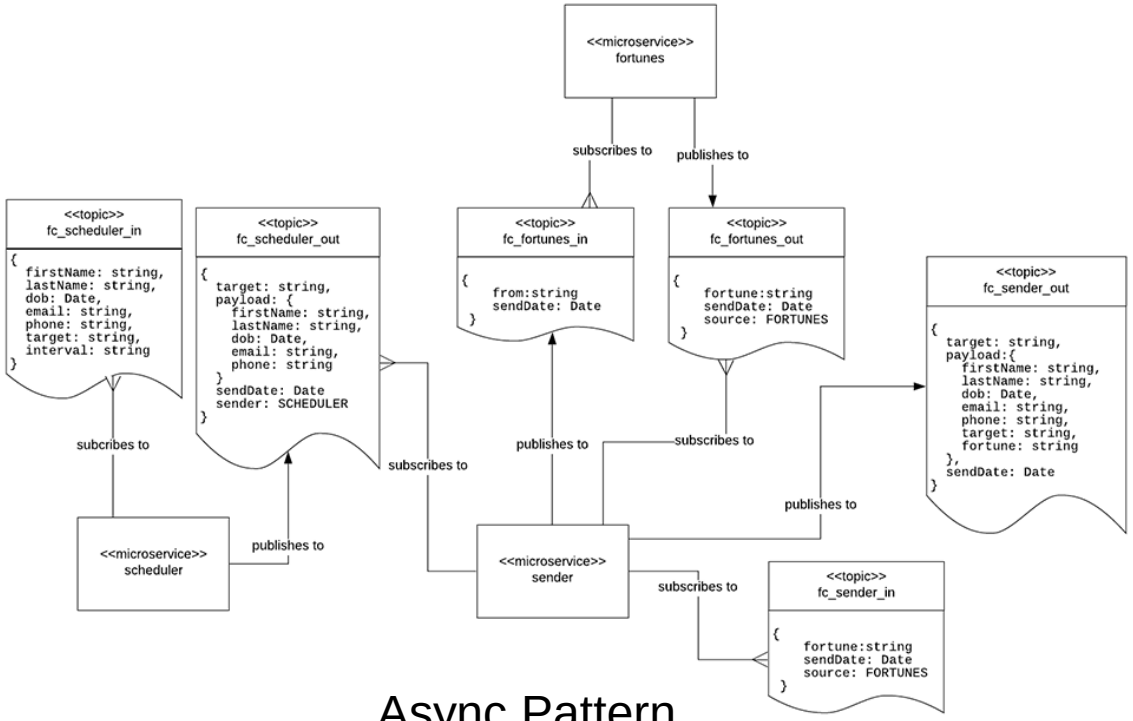
Design and possibly implement `advertiser` as a microservice(s).

© 2020 by Innovation In Software Corporation

INDIVIDUAL/GROUP EXERCISE EXAMPLES OF CONCEPTUAL/LOGICAL DELIVERABLE



Lambda Pattern



Async Pattern

Ad Auction

- Keeps an inventory of ads for sale
- Conducts the auctioning of ads to the highest bidder
- Delivers the ad to the auction winner

Objective

Design and possibly implement `advertiser` as a synchronous microservices.

© 2020 by Innovation In Software Corporation

WRAP UP

© 2020 by Innovation In Software Corporation

BRINGING IT ALL TOGETHER

- Consider a 4 phase approach to microservice architecture: Concept, Logic, Structure, Concrete Implementation
- The basic purpose of a micoservice architecture is to release high quality software as fast as possible
- Containers are the essential building blocks of modern microservices
- The more microservices you have, the more complex the system becomes.
- Don’t forget the CAP Theorem
- Context counts!
- When it comes to event driven microservice architectures, pay attention to messaging logic and message transformation mechanisms
- Separating service representation from service behavior will help you scale more easily
- Kubernetes is not the only microservice orchestration technology, but it is becoming the most popular
- If given the choice, it’s best to have a microservice carry its own data.
- Security is an art and a science. There is rarely on one size fits all solution. Still, treat everything outside your deployment unit as hostile.
- Monitor, monitor, monitor!
- Nothing is static. Hence, at some point you will need some sort of service discovery mechanism.
- Service meshes are fast becoming the dominant means by which to management very big microservice deployments at scale
- Design your systems to be flexible enough to accommodate change, yet stable enough to be predictable.

© 2020 by Innovation In Software Corporation

THANK YOU FOR ATTENDING

Please do the course evaluation provided by the session coach:

http://www.metricsthatmatter.com/student/evaluation.asp?XXXXXXX

© 2020 by Innovation In Software Corporation

MICROSERVICE APPLICATION ARCHITECTURE

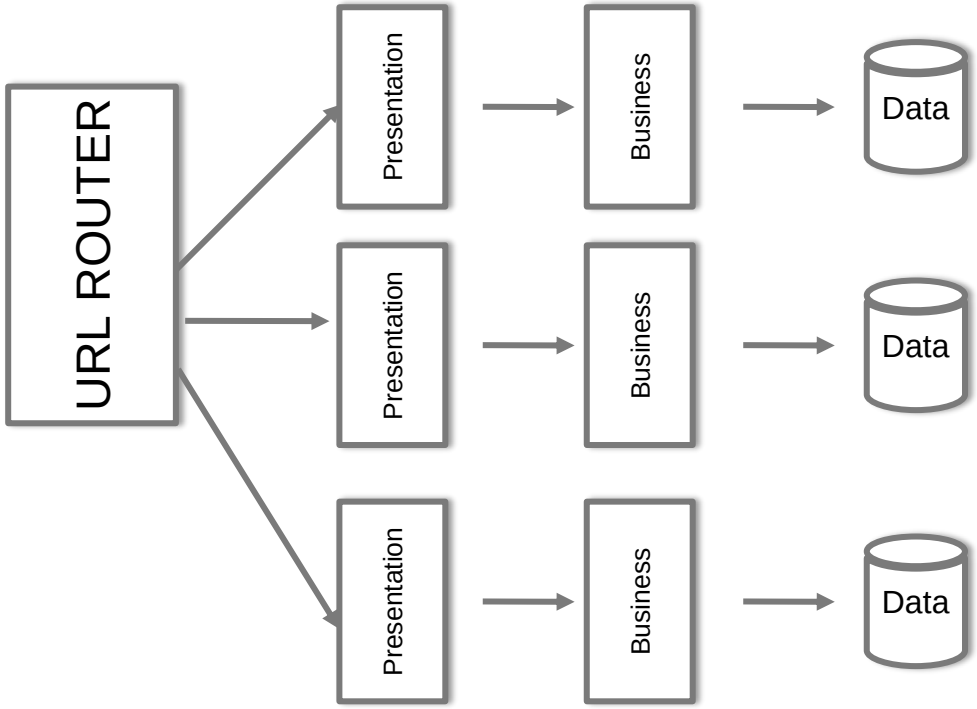
APPENDICES

© 2020 by Innovation In Software Corporation

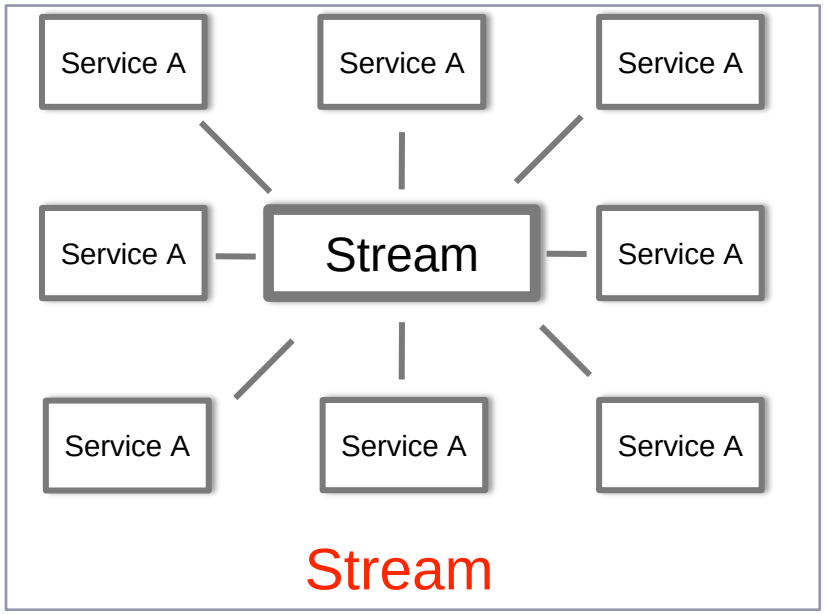
APPENDIX A

4 TYPES OF DISTRIBUTED ARCHITECTURES

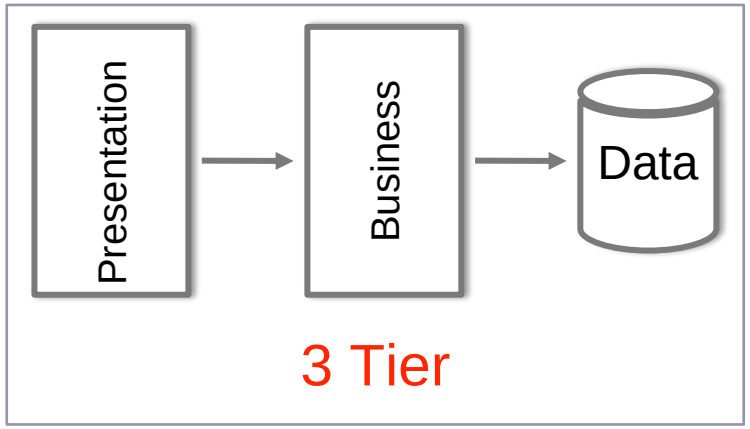
© 2020 by Innovation In Software Corporation



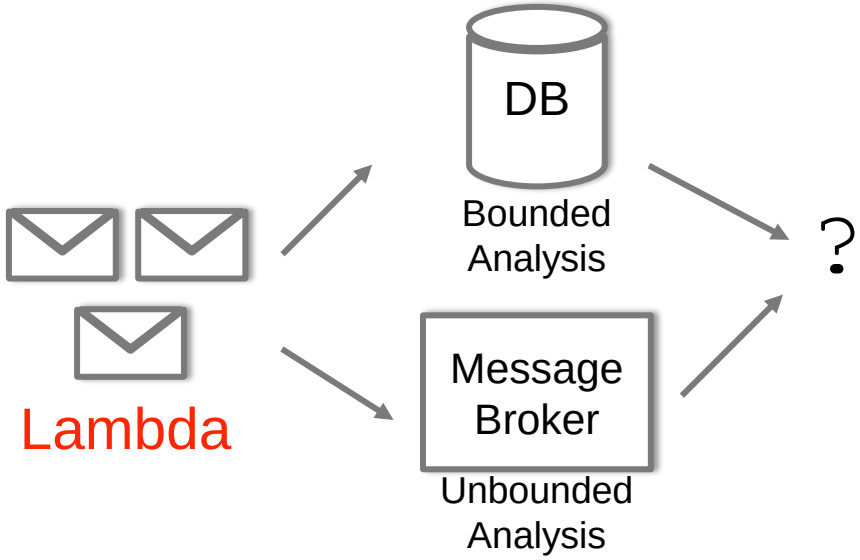
Sharded



Stream



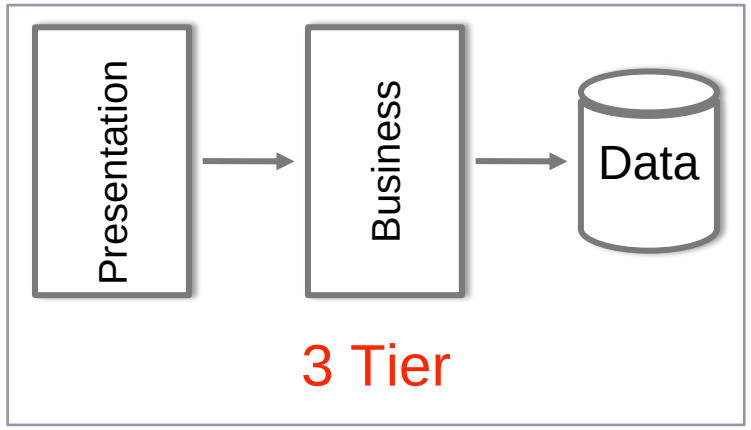
3 Tier



Lambda

4 TYPES OF DISTRIBUTED ARCHITECTURAL PATTERNS

© 2020 by Innovation In Software Corporation

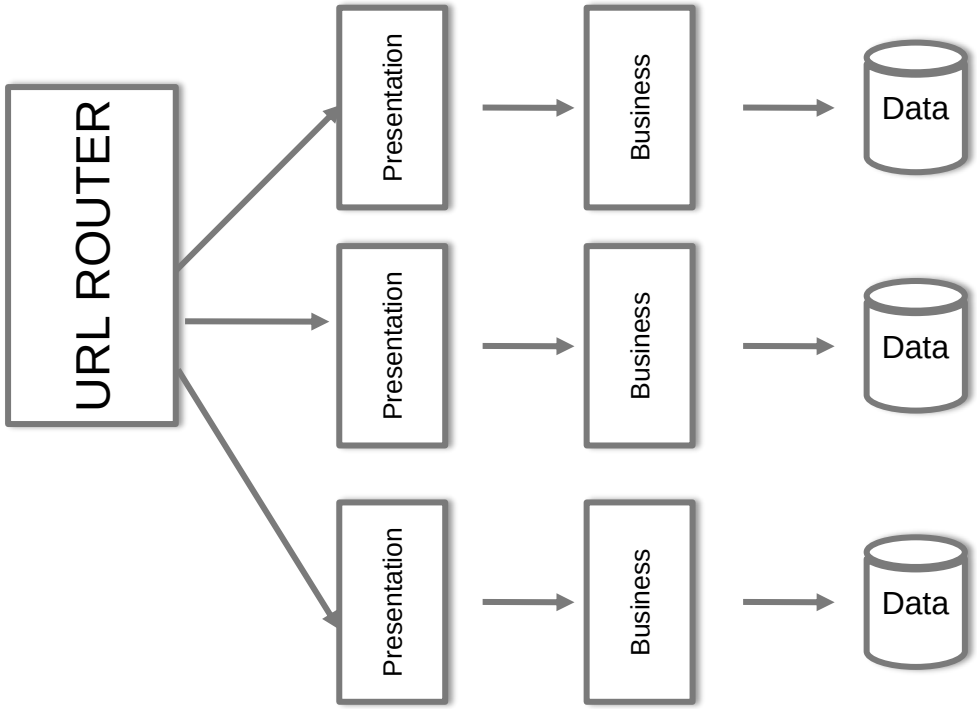


- Lots of great UI libraries
- You can scale tiers independently
- You need to be careful holding state in mid-tier
- Tend to be framework specific

Scalability	☆	☆	☆	☆
Coolness	☆	☆		
Difficulty	☆	☆		
Flexiblity	☆	☆	☆	☆

3 TIER

© 2020 by Innovation In Software Corporation



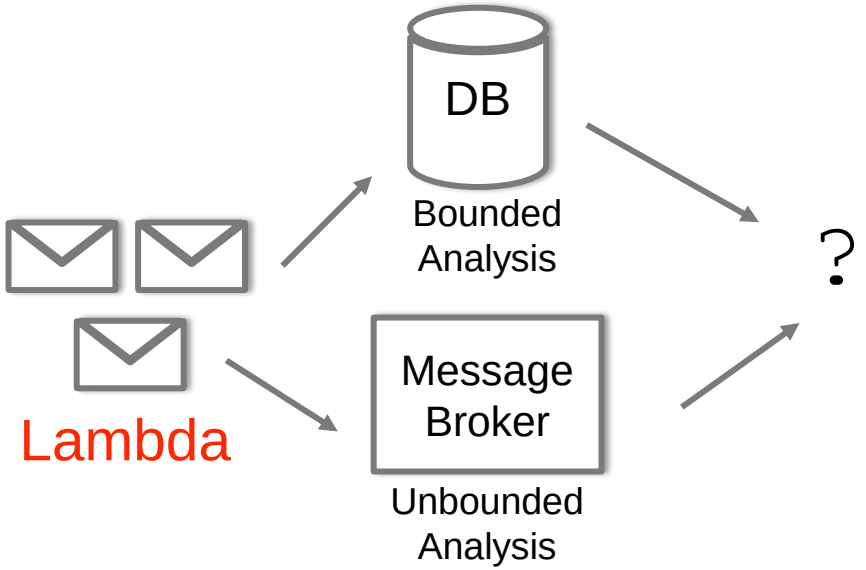
Sharded

- Client isolation easy
- No reinventing the wheel
- Over time shards will take on a life of their own
- Things can get complex
- Updates can be a challenge
- Shards can become oversized

Scalability	☆	☆	☆	
Coolness	☆			
Difficulty	☆	☆		
Flexiblity	☆	☆		

SHARDED

© 2020 by Innovation In Software Corporation

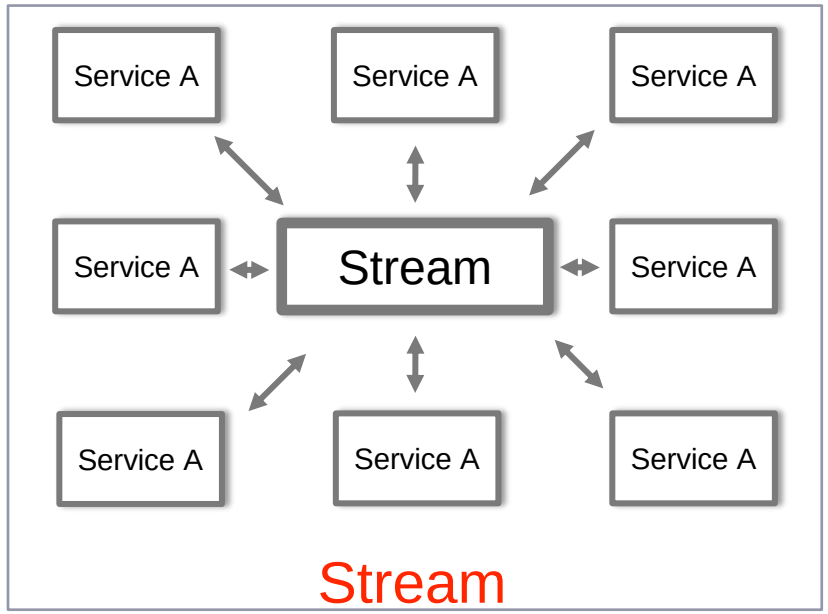


Scalability	☆	☆	☆	☆
Coolness	☆			
Difficulty	☆	☆	☆	☆
Flexiblity	☆	☆		

- Not general purpose
- Great for unbounded data
- Assumes events are immutable
- Things can get complex
- You have the best of both worlds
- You have the worst of both worlds

LAMBDA

© 2020 by Innovation In Software Corporation



Everything is an event

- DB Writes
- DB Updates
- Log Writes
- Errors & Exceptions
- App State Changes

Storage becomes multi-faceted

STREAM

Scalability	☆	☆	☆	☆
Coolness	☆	☆	☆	☆
Difficulty	☆	☆	☆	☆
Flexiblity	☆	☆	☆	☆

© 2020 by Innovation In Software Corporation

APPENDIX B

DOCKER COMPOSE

© 2020 by Innovation In Software Corporation

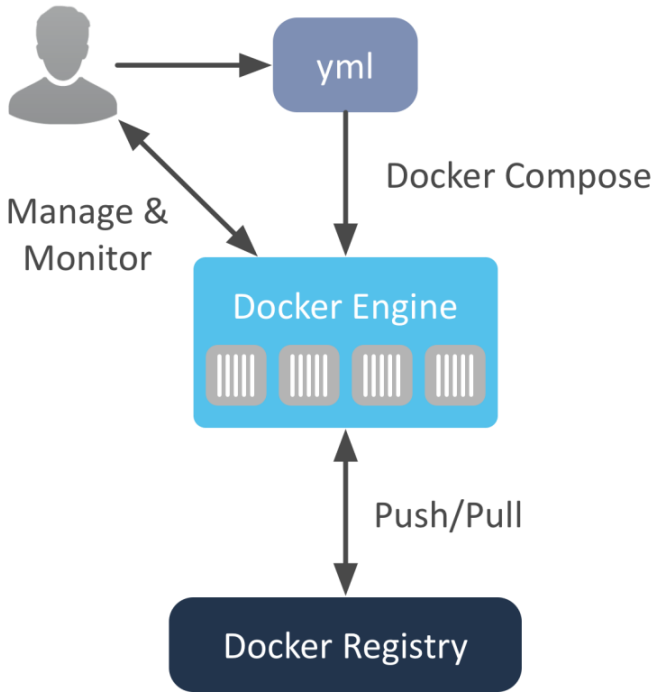
DOCKER COMPOSE

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Tool to create and manage multi-container applications

- **Applications defined in a single file:**

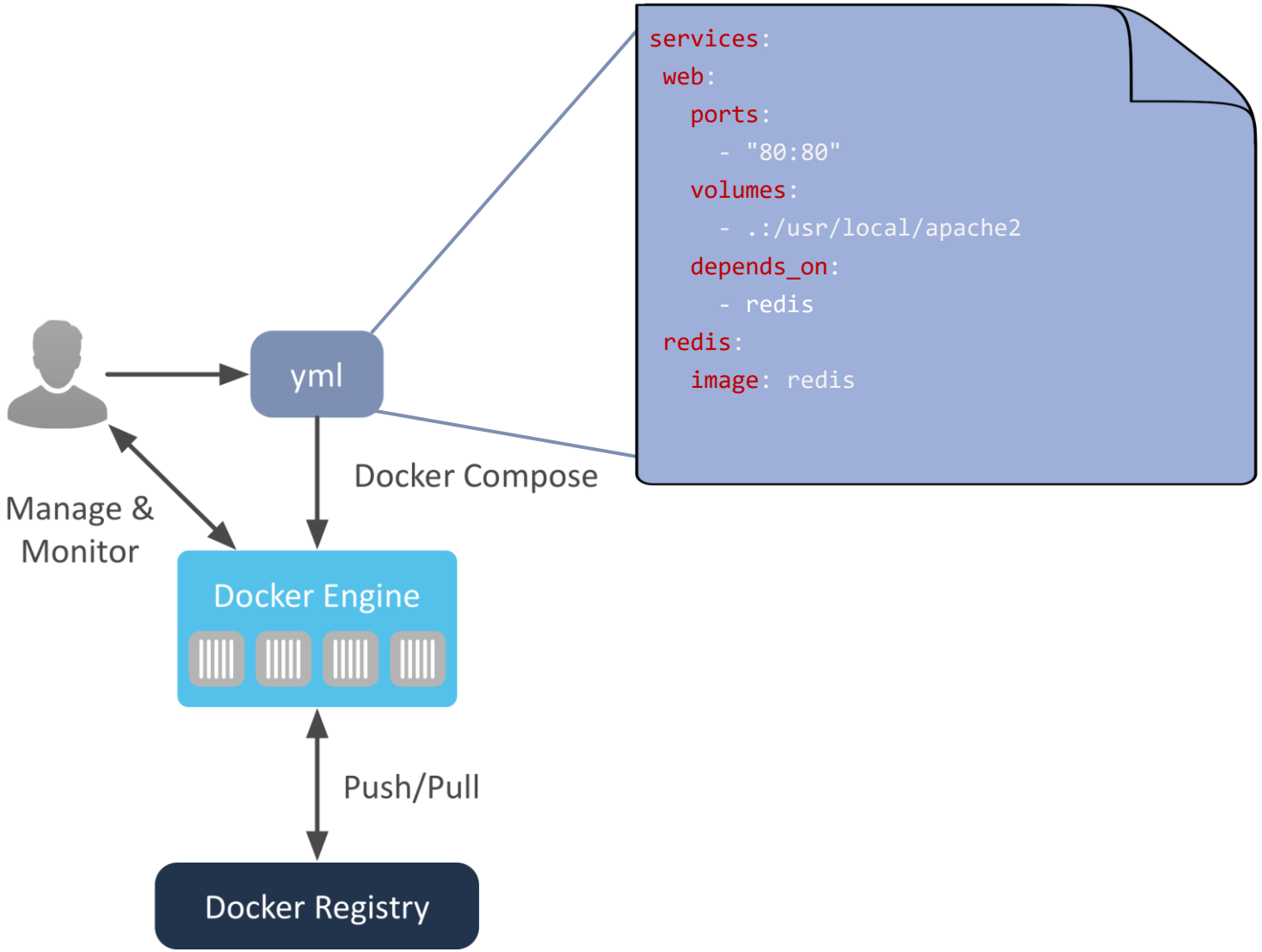
  docker-compose.yml
- **Transforms applications into individual containers that are linked together**
- **Compose will start all containers in a single command**



© 2020 by Innovation In Software Corporation

DOCKER COMPOSE

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm



© 2020 by Innovation In Software Corporation

LAB:
DOCKER COMPOSE

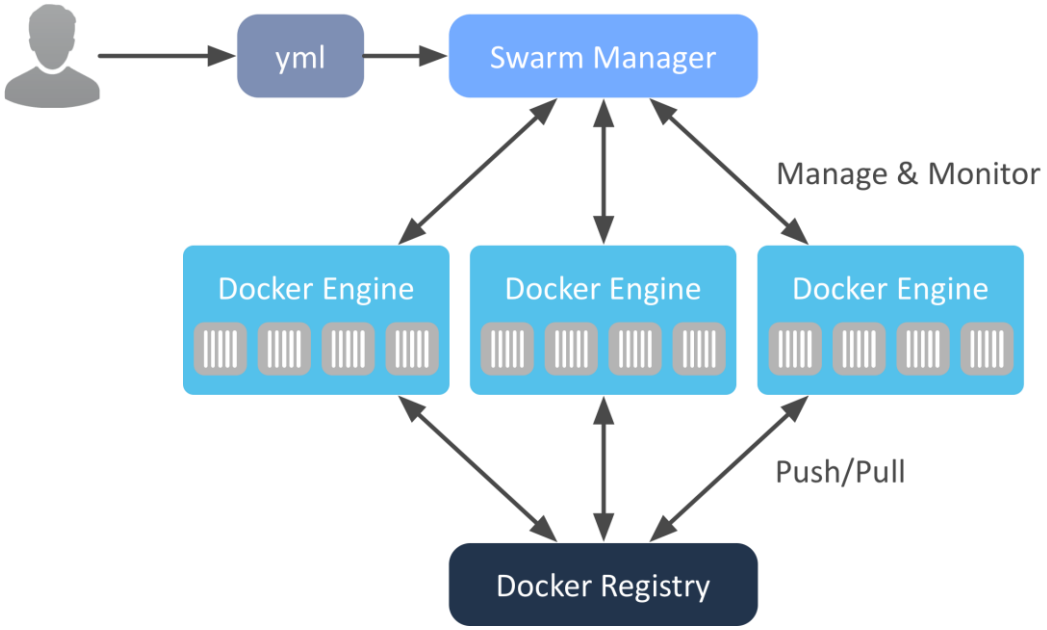
© 2020 by Innovation In Software Corporation

DOCKER SWARM

Docker Engine | *Docker Registry* | Docker Compose | Docker Swarm

Clusters Docker hosts and schedules containers

- **Native Clustering for Docker**
  - Turn a pool of Docker hosts into a single, virtual host
  - Serves the standard Docker API



© 2020 by Innovation In Software Corporation

COREOS RKT (ROCKET)

A security-minded, standards-based container engine.

© 2020 by Innovation In Software Corporation

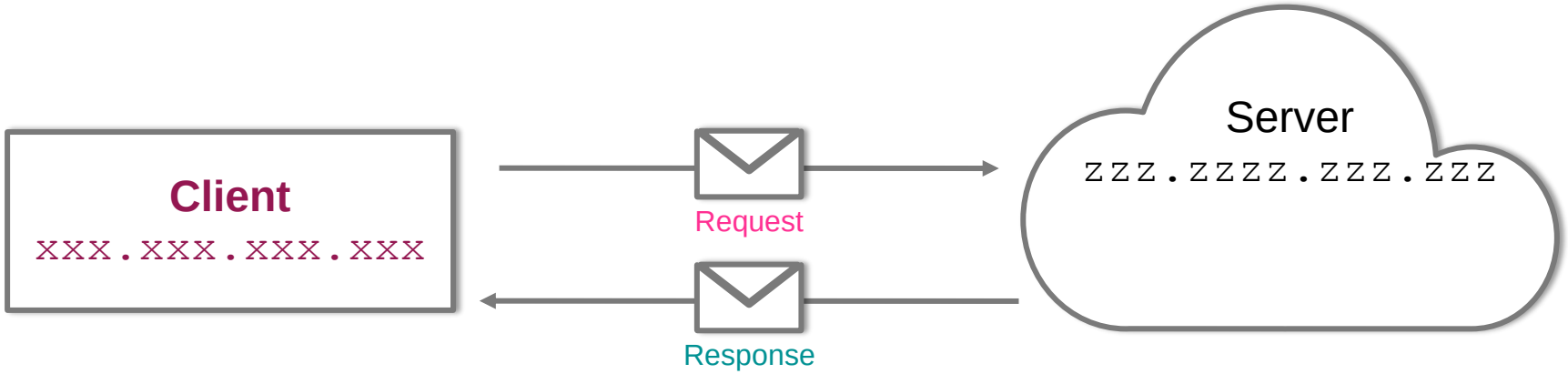
OPEN CONTAINER INITIATIVE (OCI)

The Open Container Initiative (OCI) is a lightweight, open governance structure (project), to create an open industry standards around container formats and runtime.

© 2020 by Innovation In Software Corporation

APPENDIX C
MESSAGING

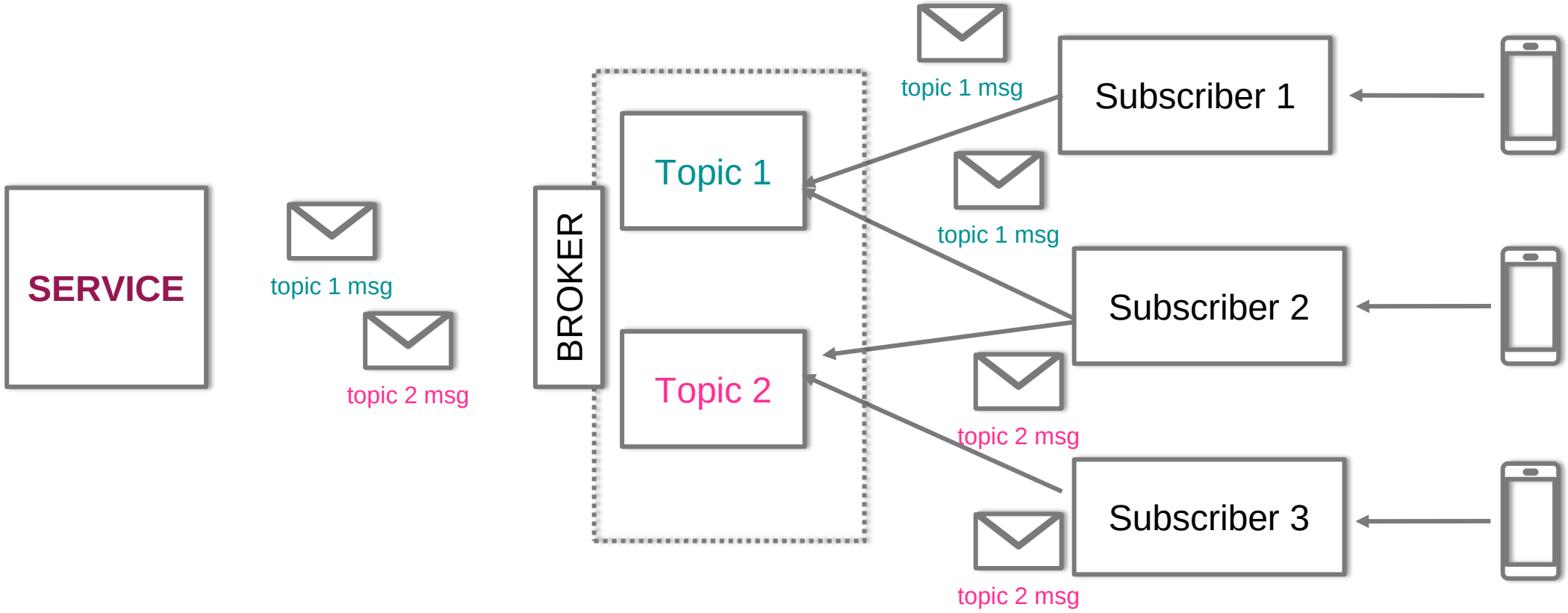
© 2020 by Innovation In Software Corporation



USEFUL MESSAGING PATTERNS

Point to Point

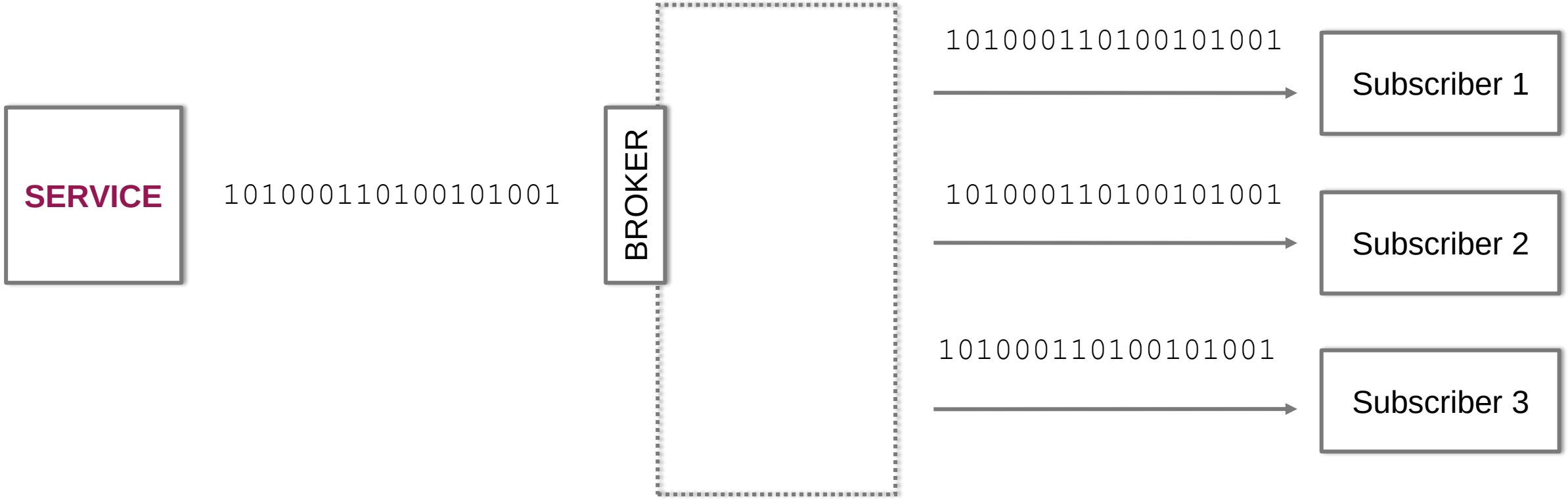
© 2020 by Innovation In Software Corporation



USEFUL MESSAGING PATTERNS

Pub-Sub

© 2020 by Innovation In Software Corporation



USEFUL MESSAGING PATTERNS Streams

© 2020 by Innovation In Software Corporation

MESSAGING PATTERNS

Streaming

- Streaming transmits data as a continuous flow, which allows the clients to utilize the data almost immediately.

© 2020 by Innovation In Software Corporation

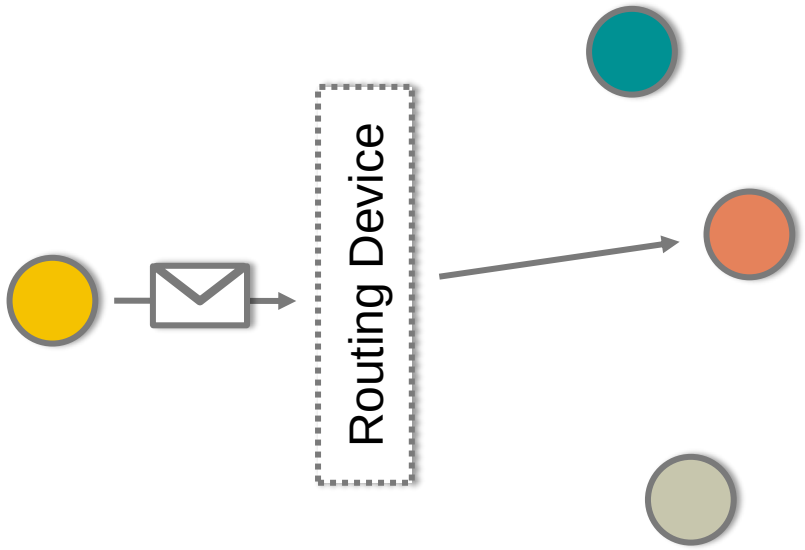
TYPE	ASSOCIATIONS	SCOPE	EXAMPLE
Unicast	1 to 1	Whole network	HTTP
Broadcast	1 to Many	Subnet	ARP
Multicast	One/Many to Many	Defined horizon	IPTV
Anycast	Many to Few	Whole network	6to4

PHYSICAL MESSAGING PATTERNS

© 2020 by Innovation In Software Corporation

MESSAGING PATTERNS

Unicast

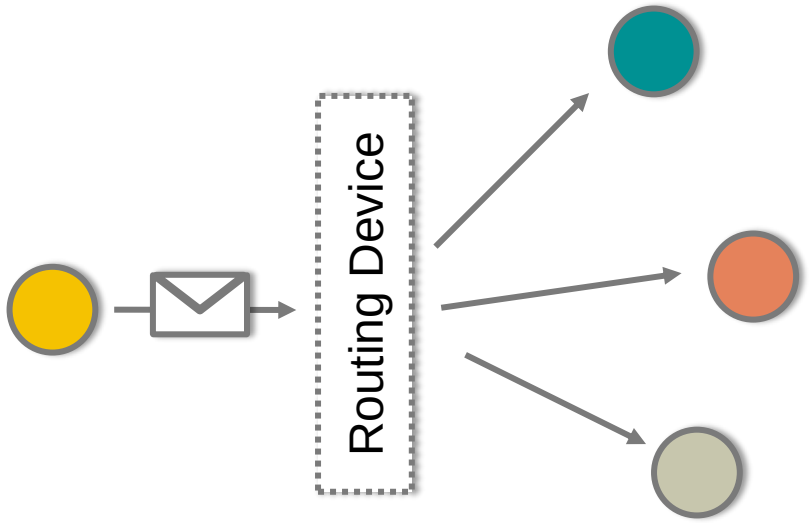


- Method of routing data that allows a single sender to one receiver.
- Supports one-to-one routing, in which a single device sends data to an other device.
- e.g., Request-response of HTTP

© 2020 by Innovation In Software Corporation

MESSAGING PATTERNS

Broadcast

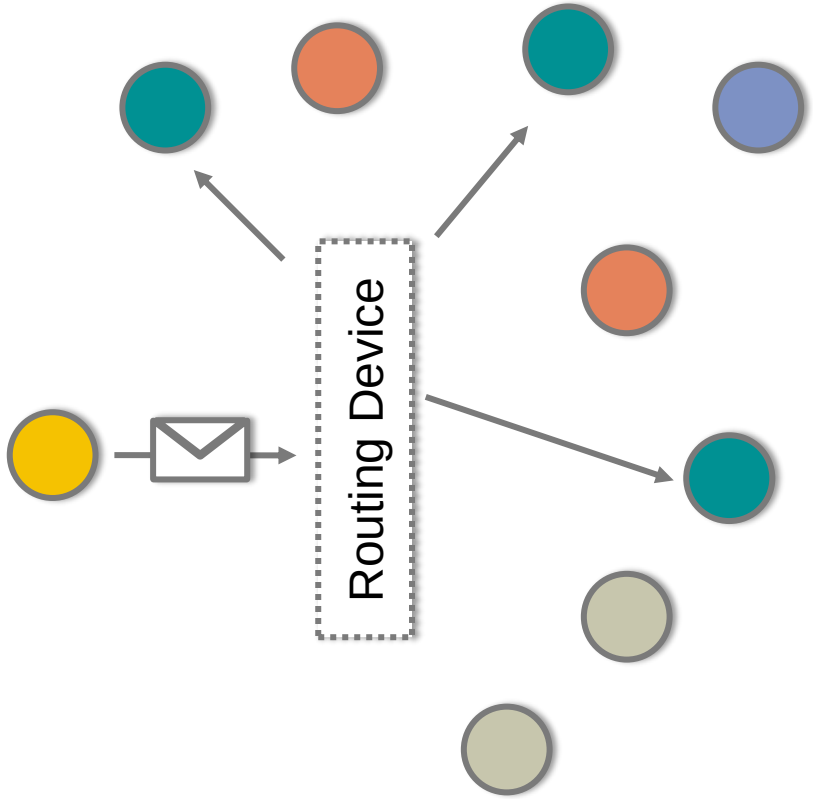


- A single sender sends data once. Data is delivered to all connected devices.
- Supports one-to-one routing, in which a single device sends data to an other device.
- e.g., Used by Address Resolution Protocol (ARP) for a source machine to find the MAC address of a machine according to IP Address

© 2020 by Innovation In Software Corporation

MESSAGING PATTERNS

Multicast

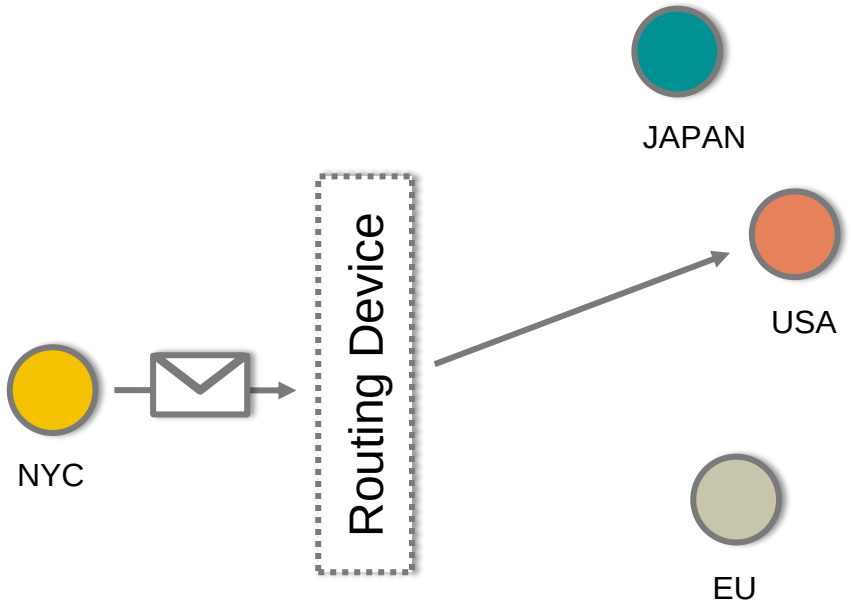


- Method of routing data that allows a single sender, or a group of senders, to communicate efficiently with a group of receivers.
- Supports one-to-many routing, in which a single device sends data to a group of devices.
- e.g., Used in Internet Protocol Television (IPTV) to send data to multiple televisions via the internet

© 2020 by Innovation In Software Corporation

MESSAGING PATTERNS

Anycast



- Method for routing Internet traffic.
- Specially-programmed router can dynamically determine the best destination for data packets.
- When data is requested, traffic is routed to different servers depending on a set of determining factors.
- Regardless of which server is chosen as the destination, the client receives the same data.
- e.g., Anycast is used to interact with a server which is near to a specific geolocation as in the case of a CDN.

© 2020 by Innovation In Software Corporation

MESSAGING COMPARISON

MESSAGING

- **Framed message based protocol**
- **Per event or operation**
  - Customer order fulfillment
  - Profile Update

STREAMING

- **Unframed data (bytes) stream based protocol**
- **Complex analytics:**
  - Microservice logging
  - Stock pricing acitiviy

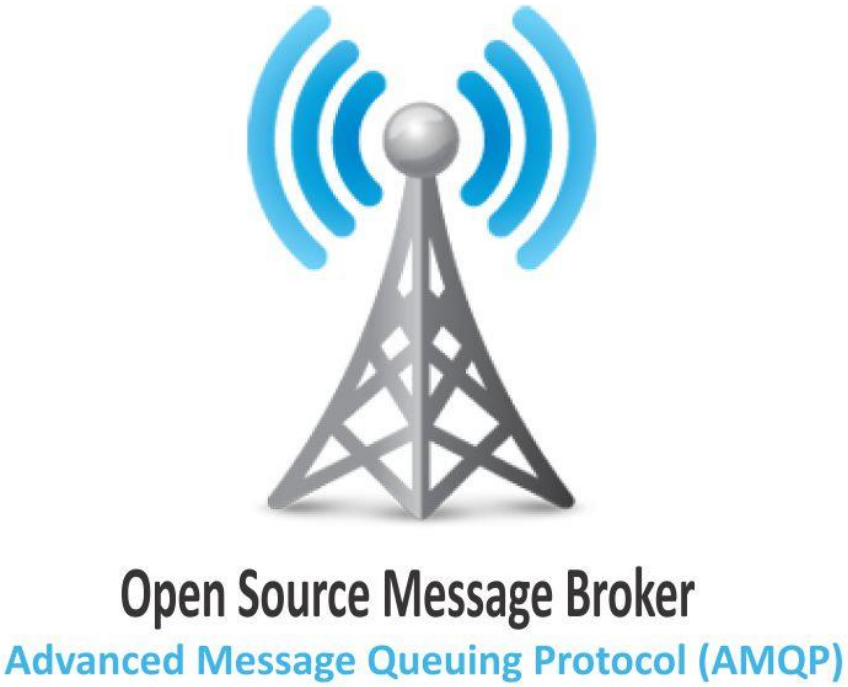
© 2020 by Innovation In Software Corporation

RABBITMQ

- Specifically designed for lightweight web messaging
- Many language supported
- Supports many messaging protocols:
  - SMTP, STOMP and HTTP
- Doesn’t limit clients to one time messages
- Consumers can retrieve different message at one time

© 2020 by Innovation In Software Corporation

ADVANCED MESSAGE QUEUING PROTOCOL (AMQP)



- **Created a functional method of accord between middle-ware and client**
- **Distribution protocol model**
- **Includes various messaging systems, such as:**
  - point-to-point, publish, fan-out, subscribe and request-response

© 2020 by Innovation In Software Corporation

ZEROMQ \ZERO-EM-QUEUE\:

- Connect your code in any language, on any platform.
- Carries messages across inproc, IPC, TCP, TIPC, multicast.
- Smart patterns like pub-sub, push-pull, and router-dealer.
- High-speed asynchronous I/O engines, in a tiny library.
- Backed by a large and active open source community.
- Supports every modern language and platform.
- Build any architecture: centralized, distributed, small, or large.

© 2020 by Innovation In Software Corporation

APACHE ACTIVEMQ

Apache ActiveMQ ™ is the a popular and powerful open source messaging and Integration Patterns server.

© 2020 by Innovation In Software Corporation

APACHE ACTIVEMQ

Supports a variety of Cross Language Clients and Protocols including:

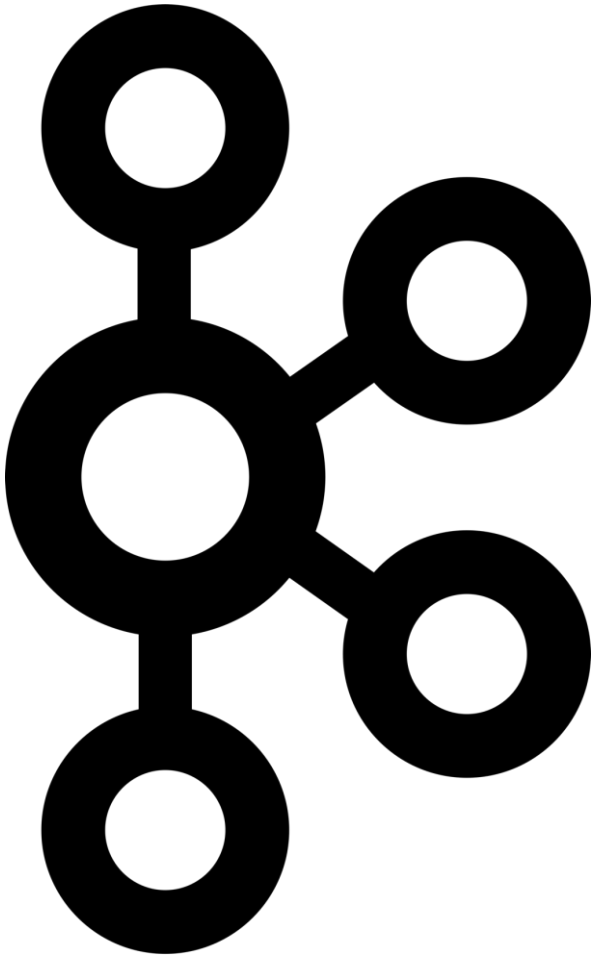
- OpenWire, Stomp, AMQP, MQTT

Full support for the Enterprise Integration Patterns

- JMS client and the Message Broker

© 2020 by Innovation In Software Corporation

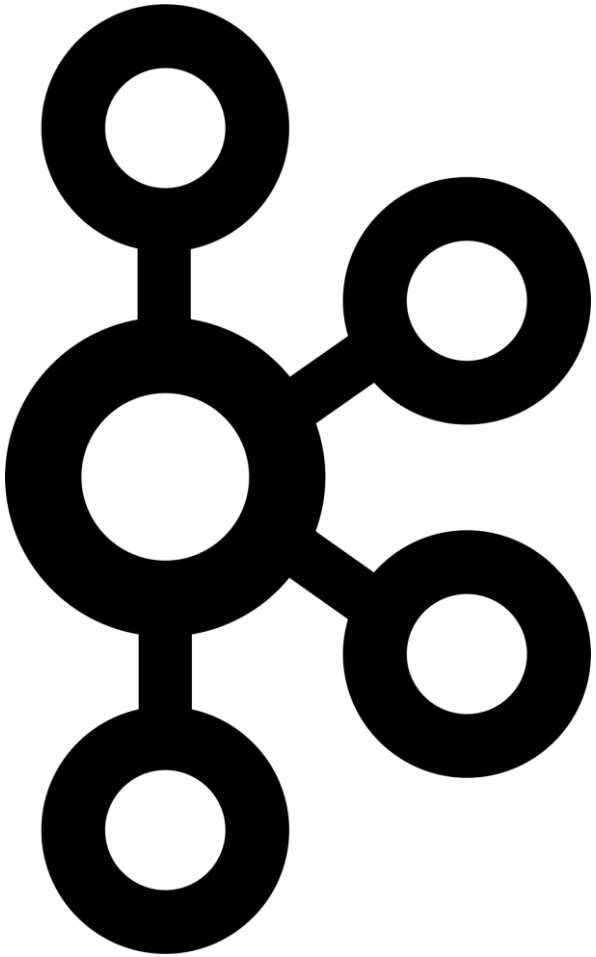
APACHE KAFKA



Kafka® is used for building real-time data pipelines and streaming apps.

© 2020 by Innovation In Software Corporation

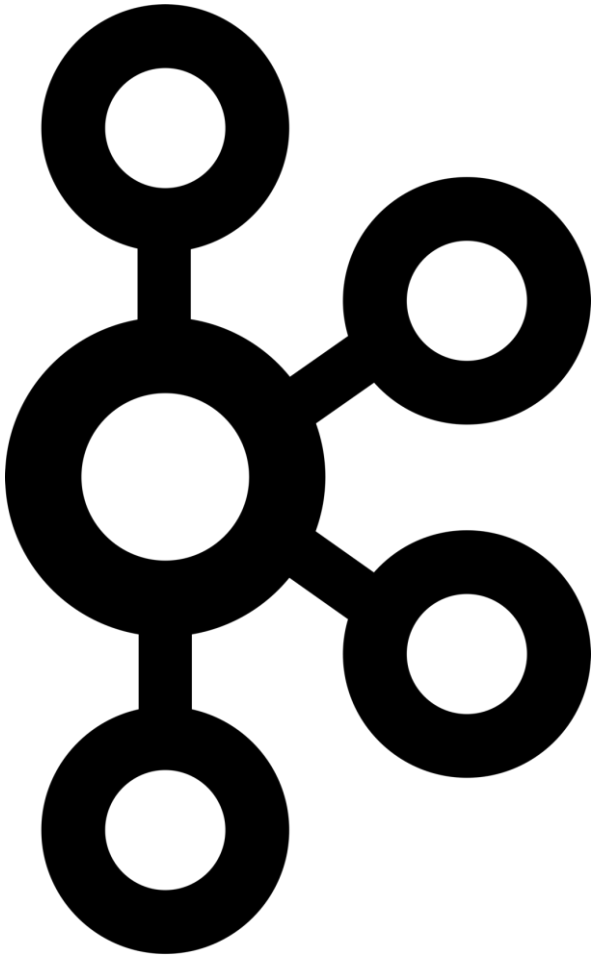
APACHE KAFKA



It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.

© 2020 by Innovation In Software Corporation

APACHE KAFKA



PUBLISH & SUBSCRIBE

- Read and write streams of data like a messaging system.

PROCESS

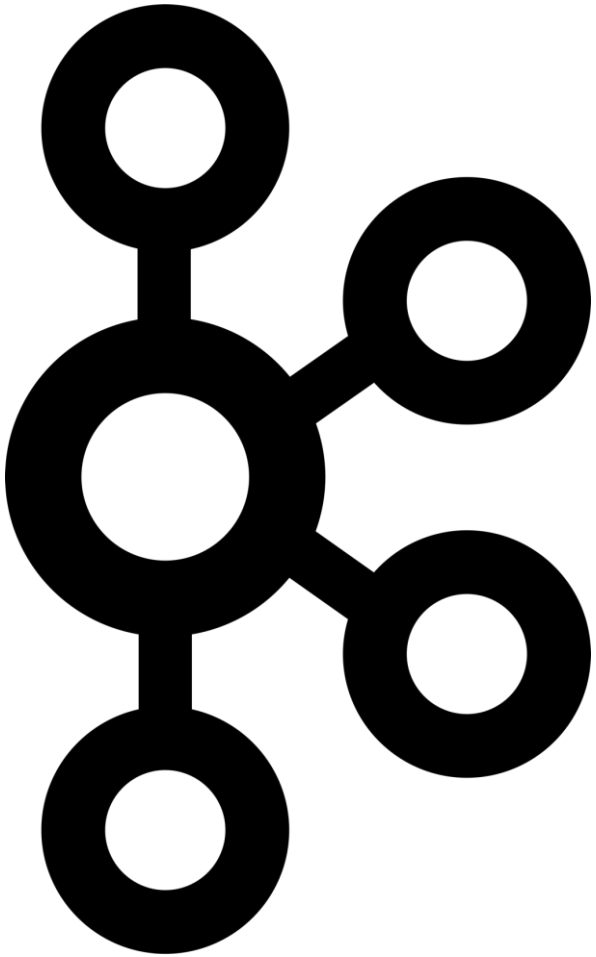
- Write scalable stream processing applications that react to events in real-time.

STORE

- Store streams of data safely in a distributed, replicated, fault-tolerant cluster.

© 2020 by Innovation In Software Corporation

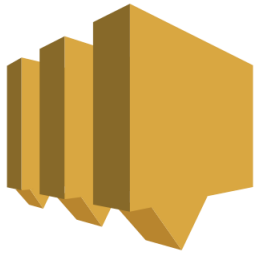
APACHE KAFKA



- Messages are stored in flat files
- Consumers able to ask messages that are based on an offset
- Routing capable through Zookeeper
- Rules are not available
- Stats available through JMX
- Great performance and low resource usage

© 2020 by Innovation In Software Corporation

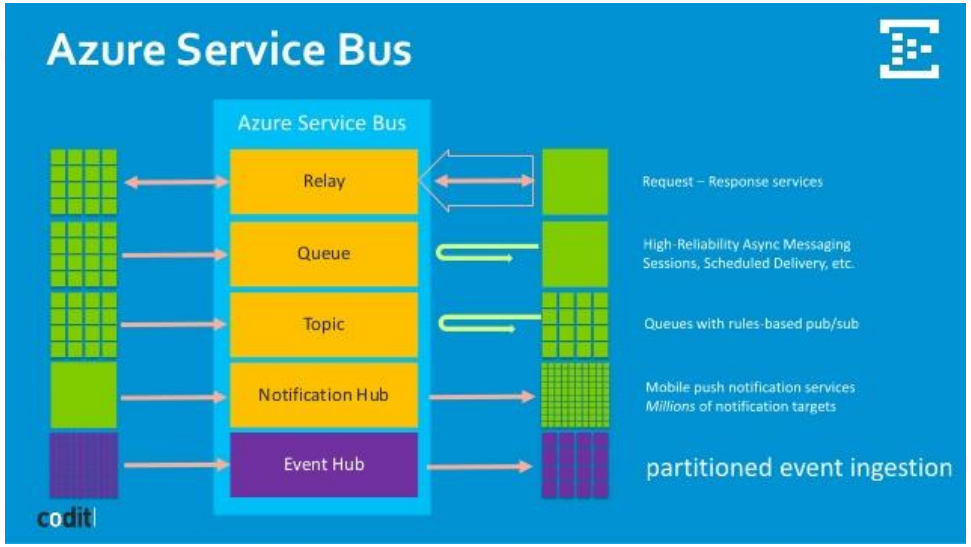
CLOUD PROVIDERS



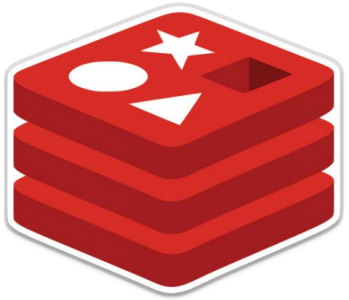
Amazon SNS

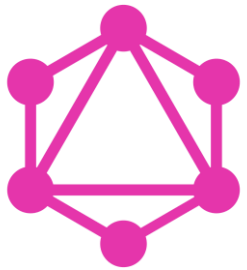
Amazon SQS

Google Cloud Pub/Sub



RELEVANT TECH





Apache Camel

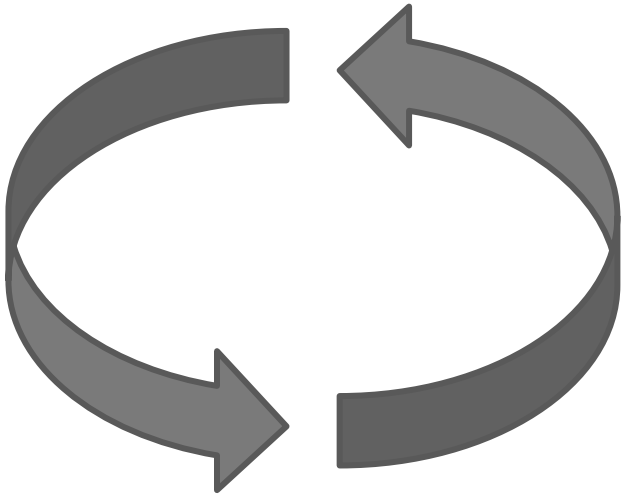
© 2020 by Innovation In Software Corporation

APPENDIX D
MONITORING

© 2020 by Innovation In Software Corporation

MONITORING

Quality Assurance



Continuous Improvement

© 2020 by Innovation In Software Corporation

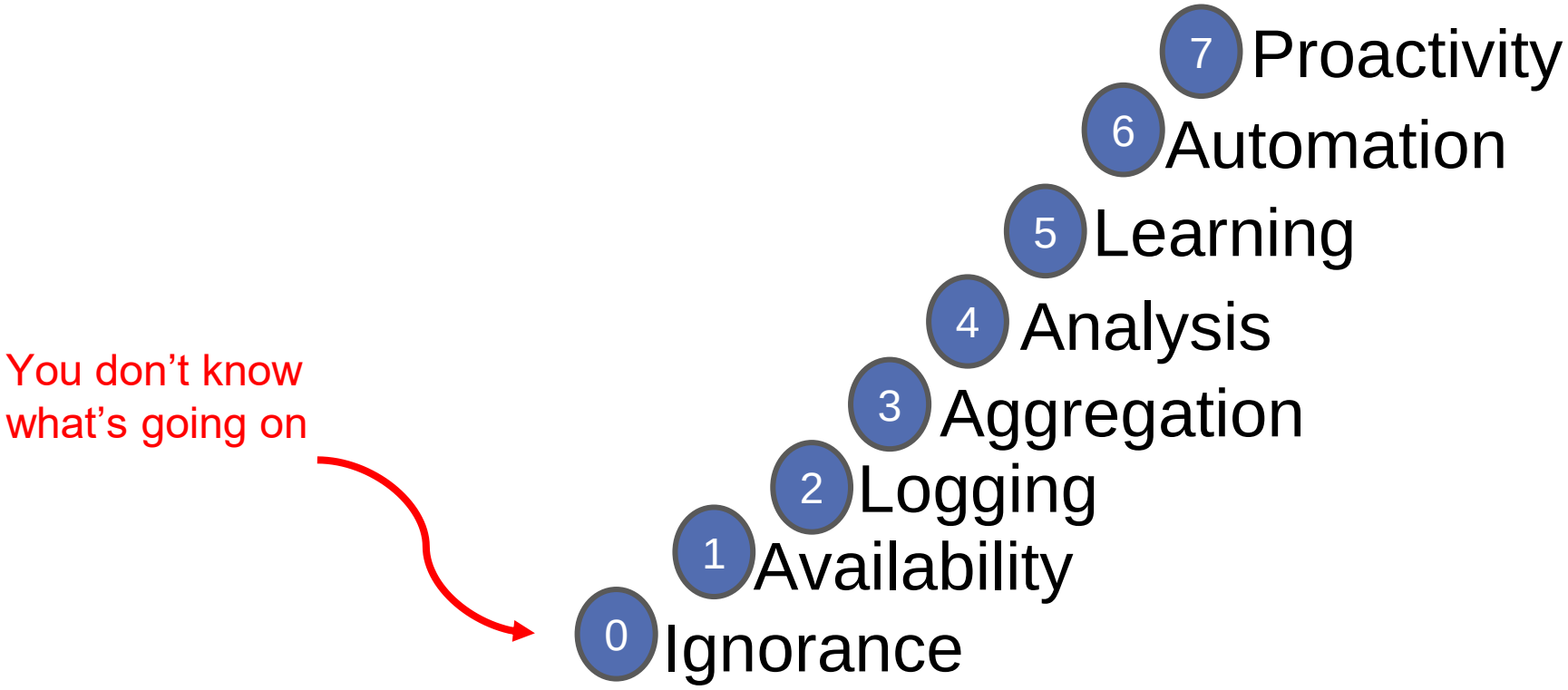
MONITORING

Why monitor?

- **Know when things go wrong**
- **Debug and learn from issues**
- **Historical view/Trends**
- **Drive technical/business decisions**
- **Feed into other systems/processes (QA, Security, automation)**

© 2020 by Innovation In Software Corporation

MONITORING



© 2020 by Innovation In Software Corporation

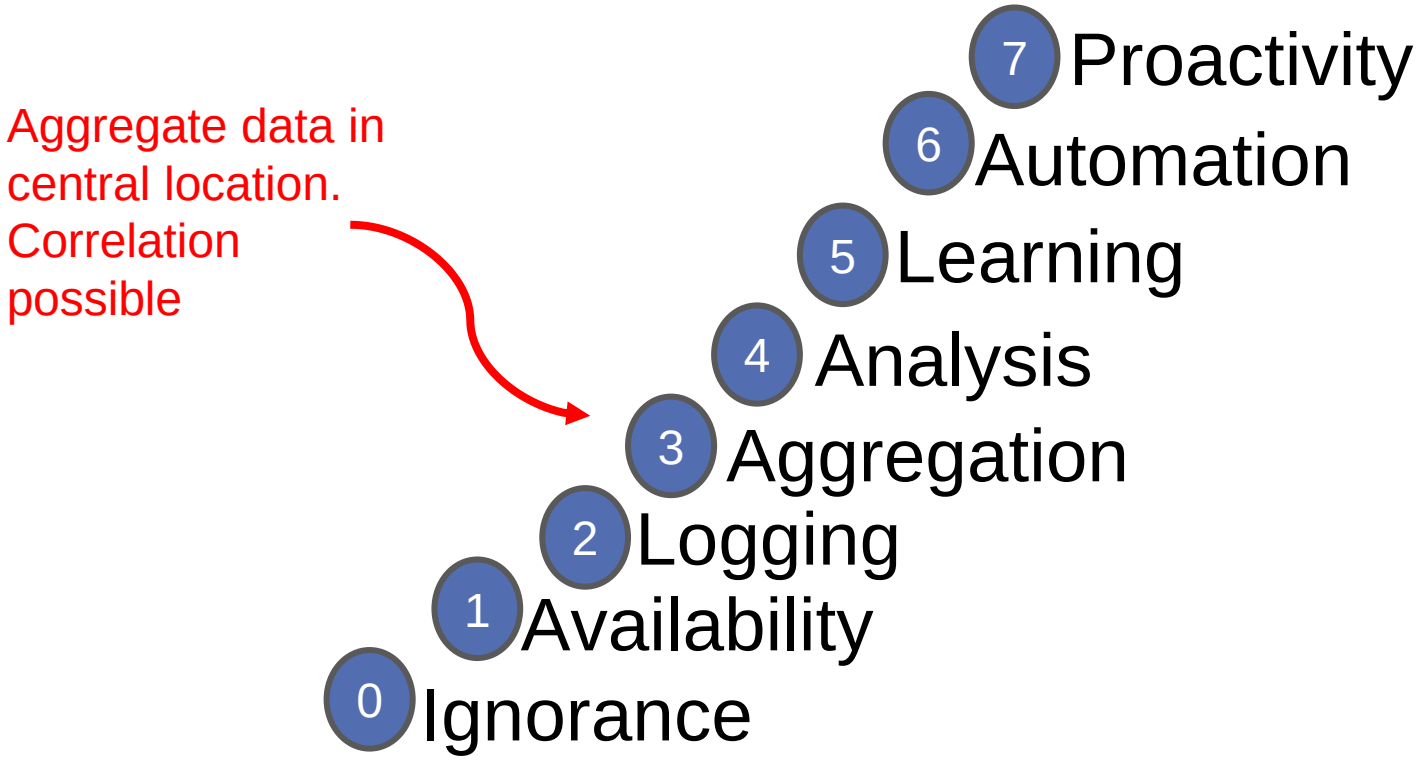
MONITORING

© 2020 by Innovation In Software Corporation

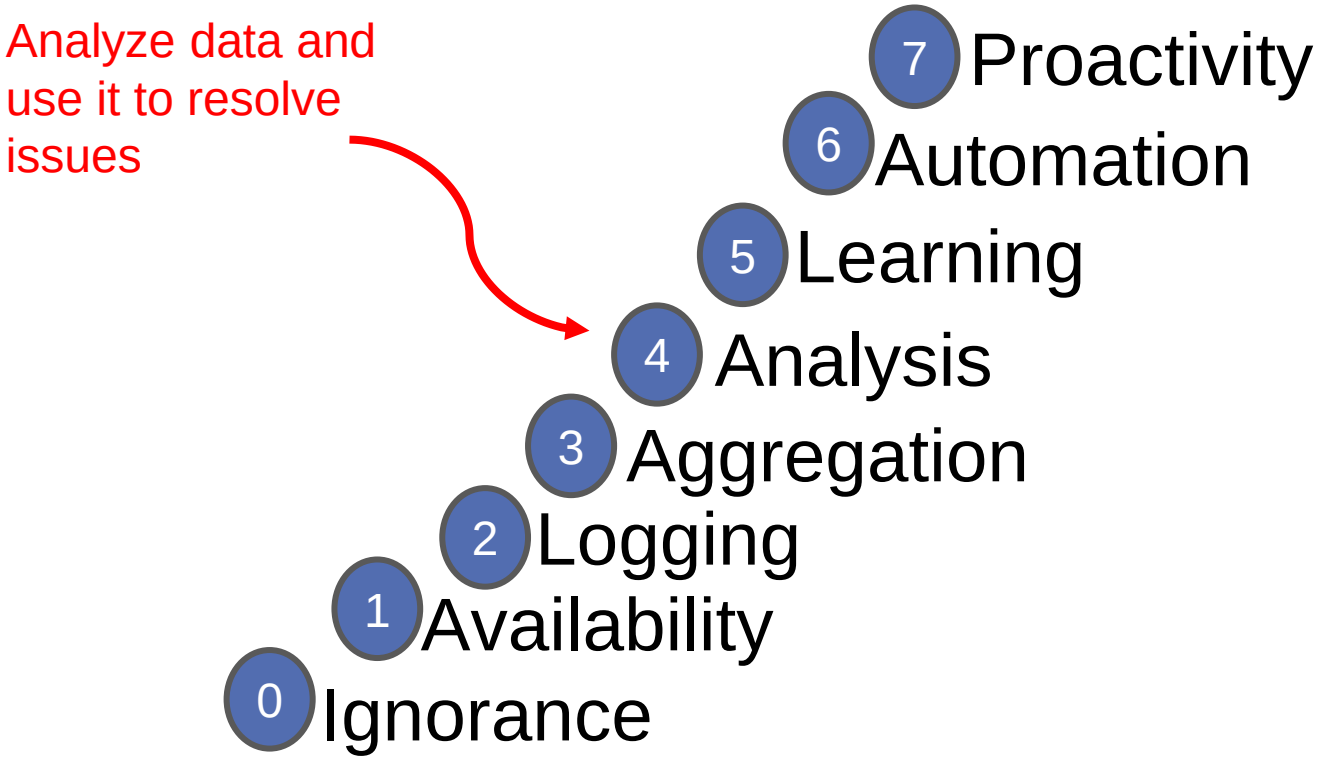
MONITORING

© 2020 by Innovation In Software Corporation

MONITORING

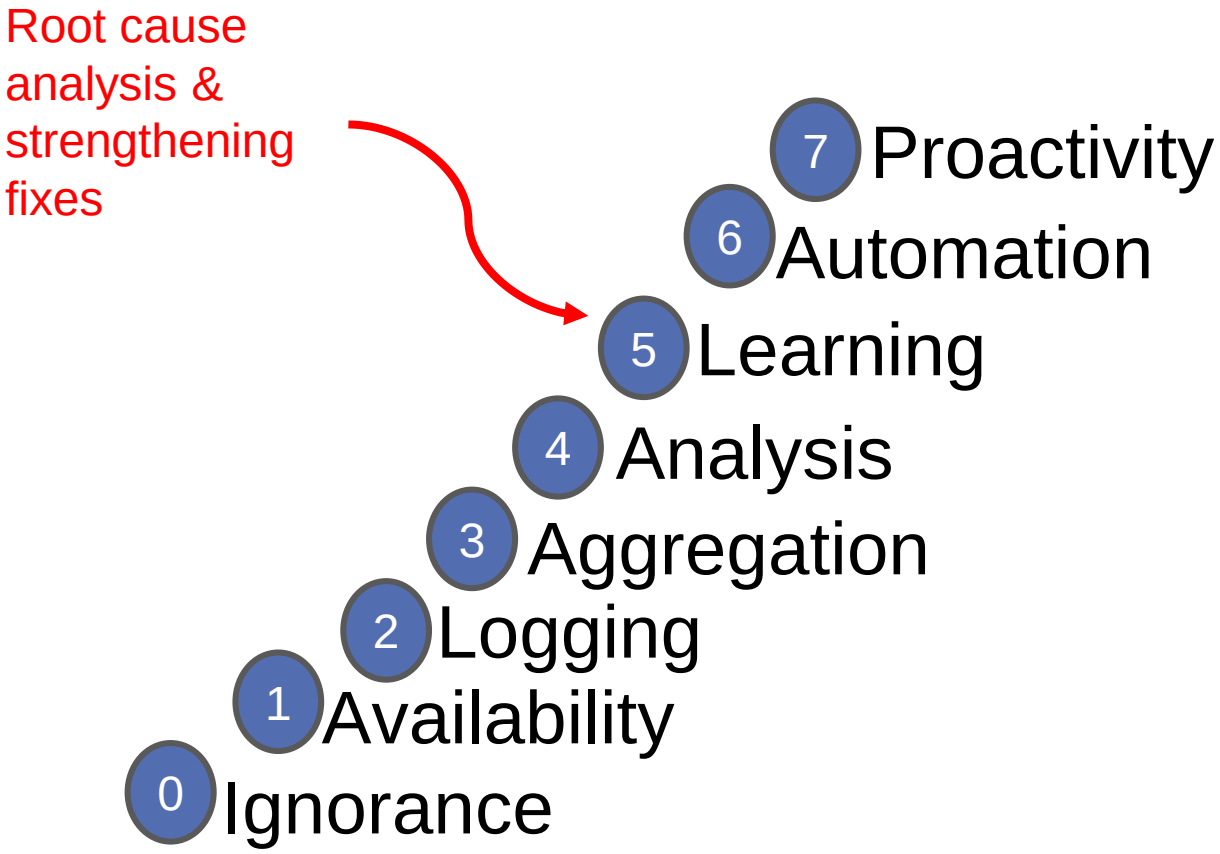

Aggregate data in central location. Correlation possible
7 Proactivity
6 Automation
5 Learning
4 Analysis
3 Aggregation
2 Logging
1 Availability
0 Ignorance

MONITORING



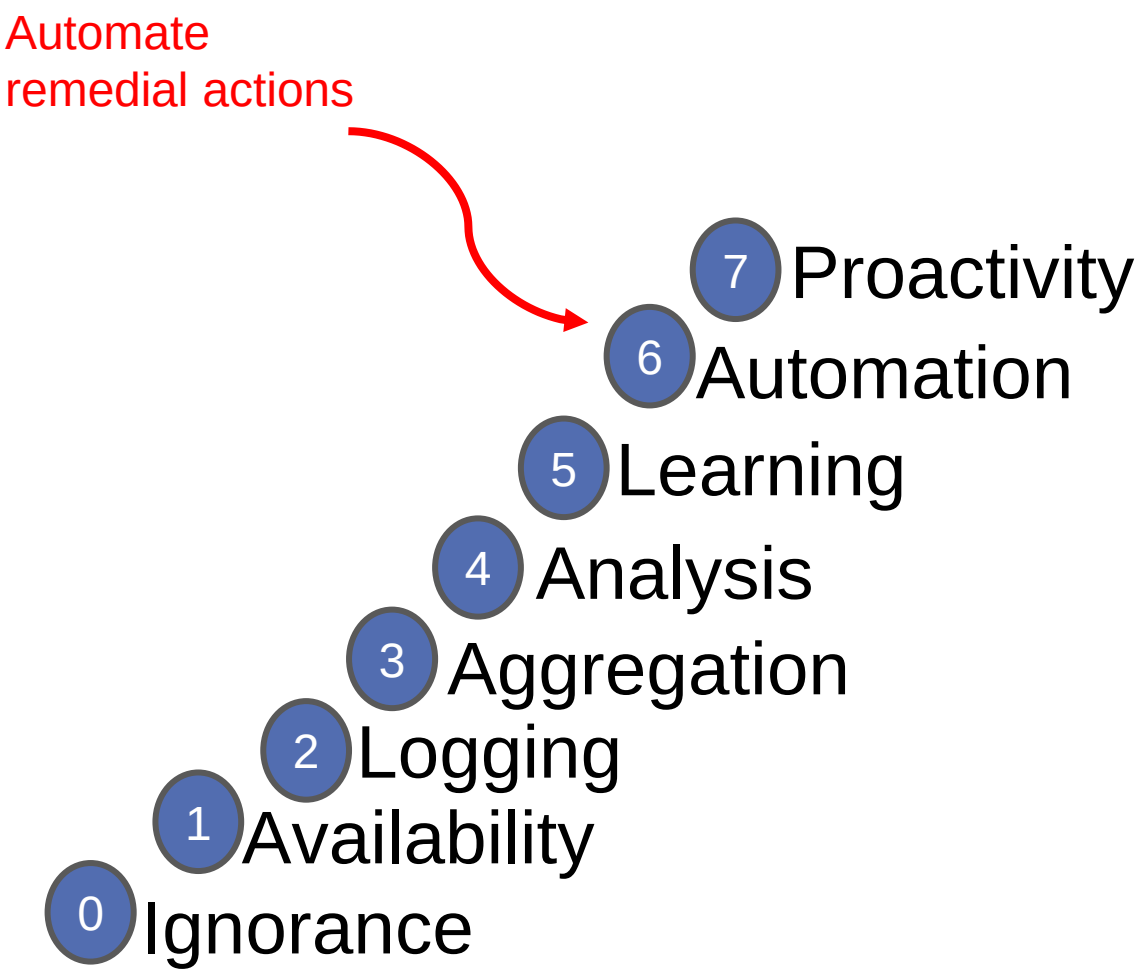
© 2020 by Innovation In Software Corporation

MONITORING



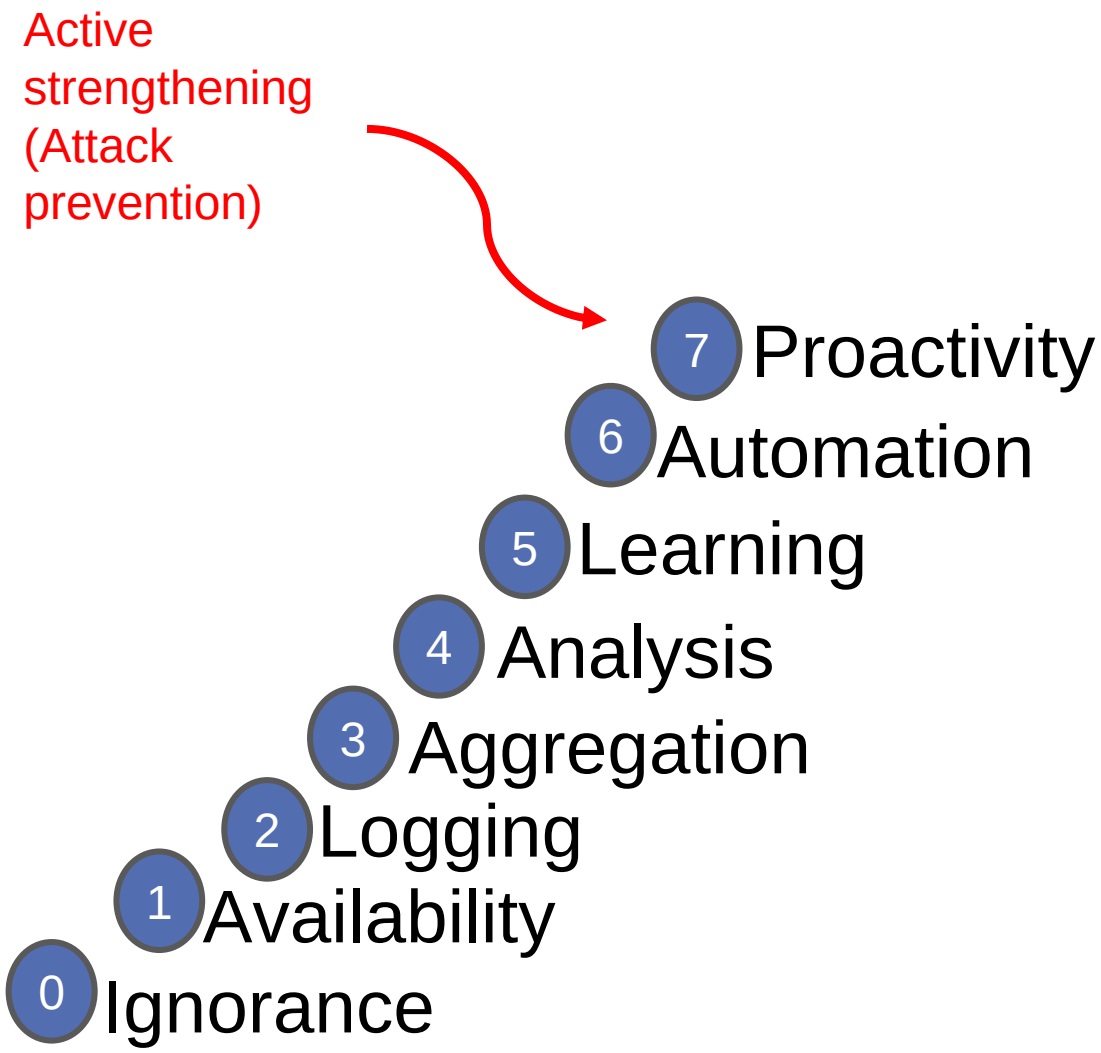
© 2020 by Innovation In Software Corporation

MONITORING



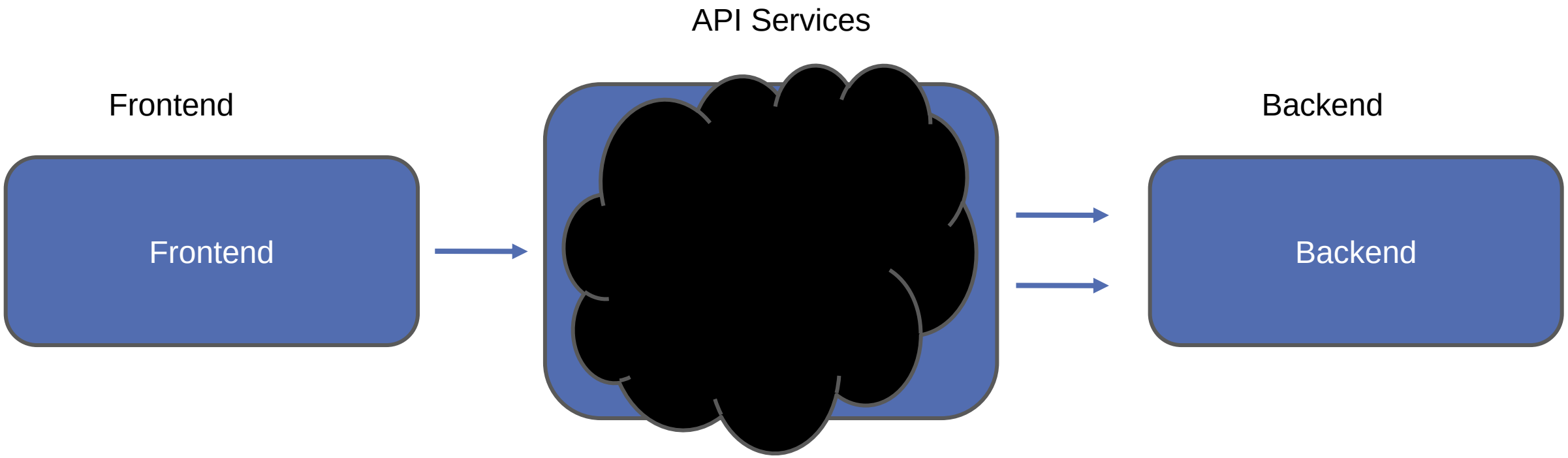
© 2020 by Innovation In Software Corporation

MONITORING



© 2020 by Innovation In Software Corporation

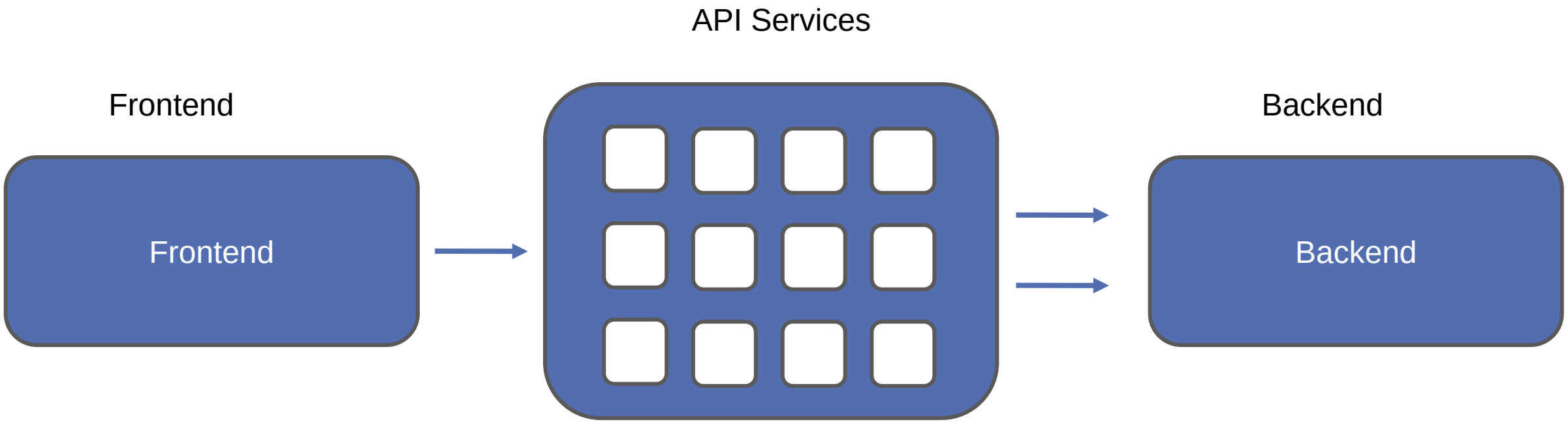
MONITORING



Limited visibility

© 2020 by Innovation In Software Corporation

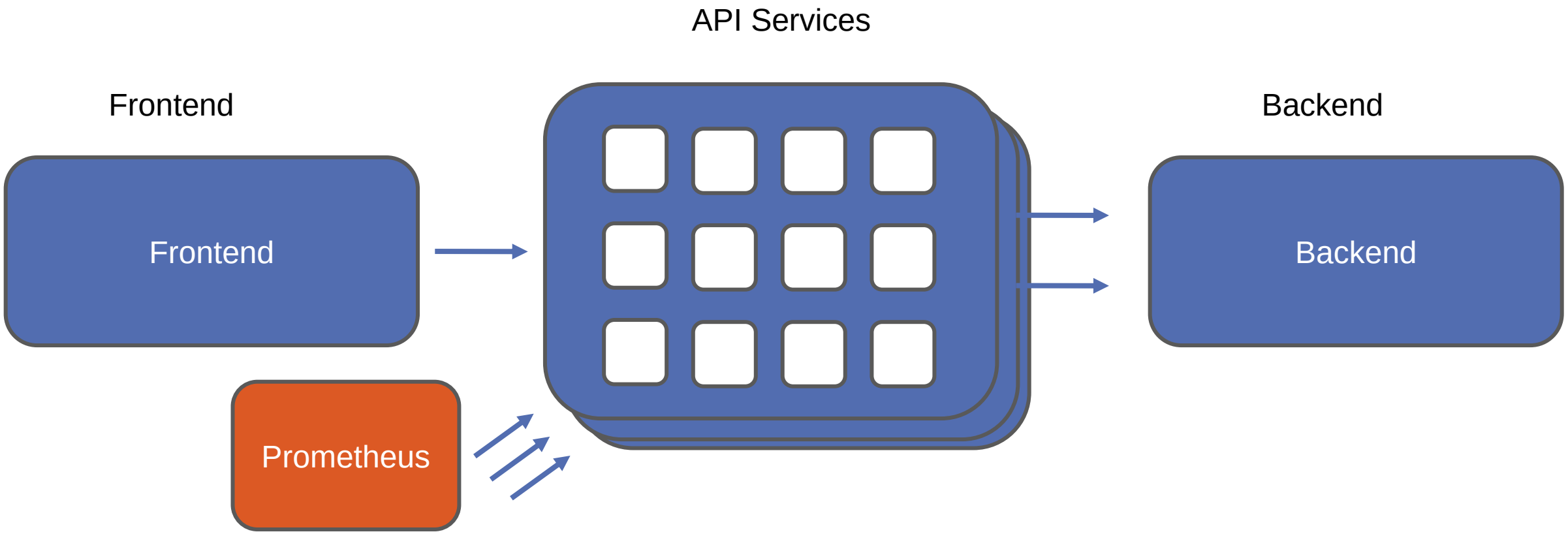
MONITORING



Monitor Internals

© 2020 by Innovation In Software Corporation

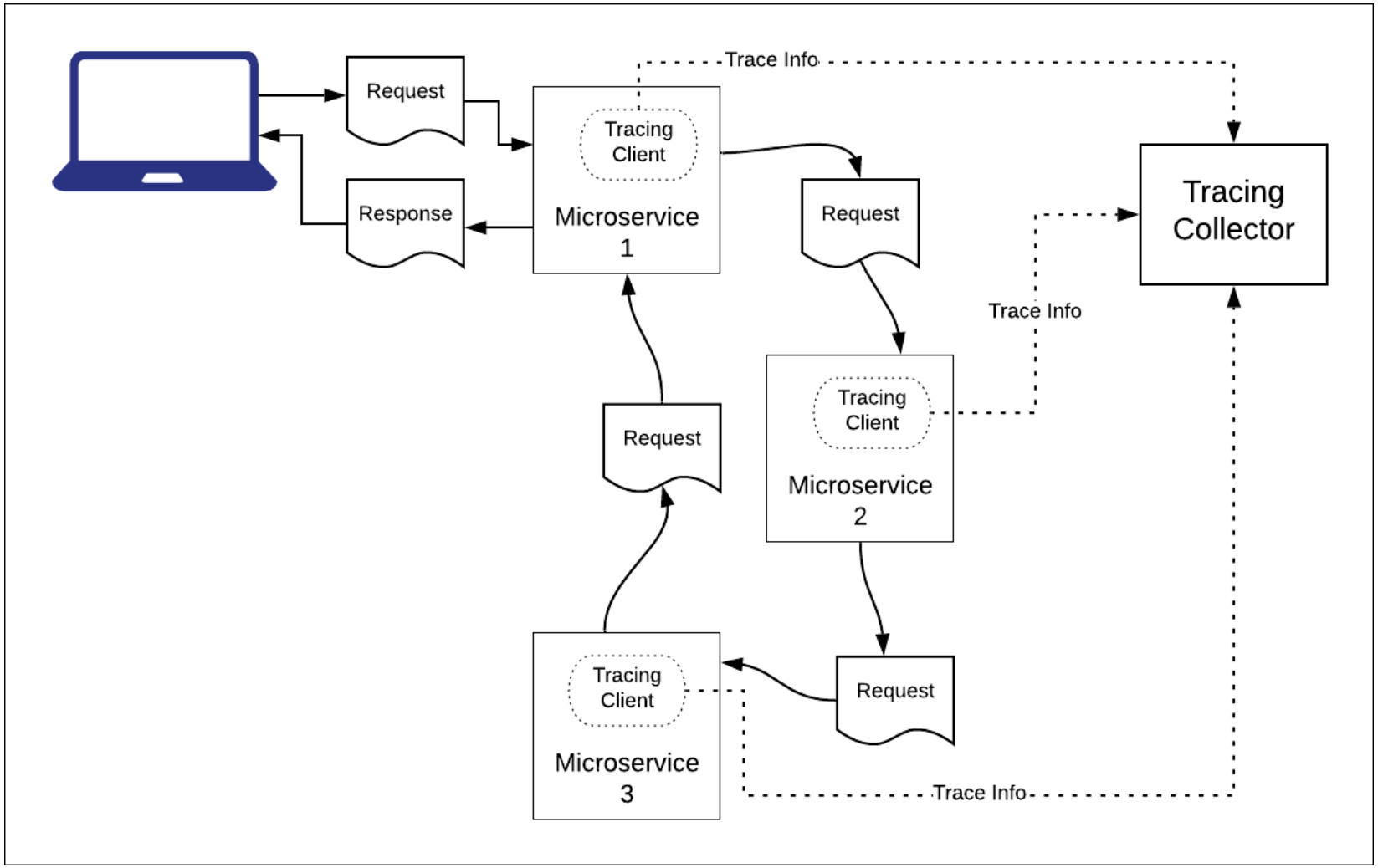
MONITORING



Monitor Services

© 2020 by Innovation In Software Corporation

MONITORING



Distributed Tracing

© 2020 by Innovation In Software Corporation

MONITORING PRODUCTS

Prometheus

ZIPKIN

Jaeger

Grafana

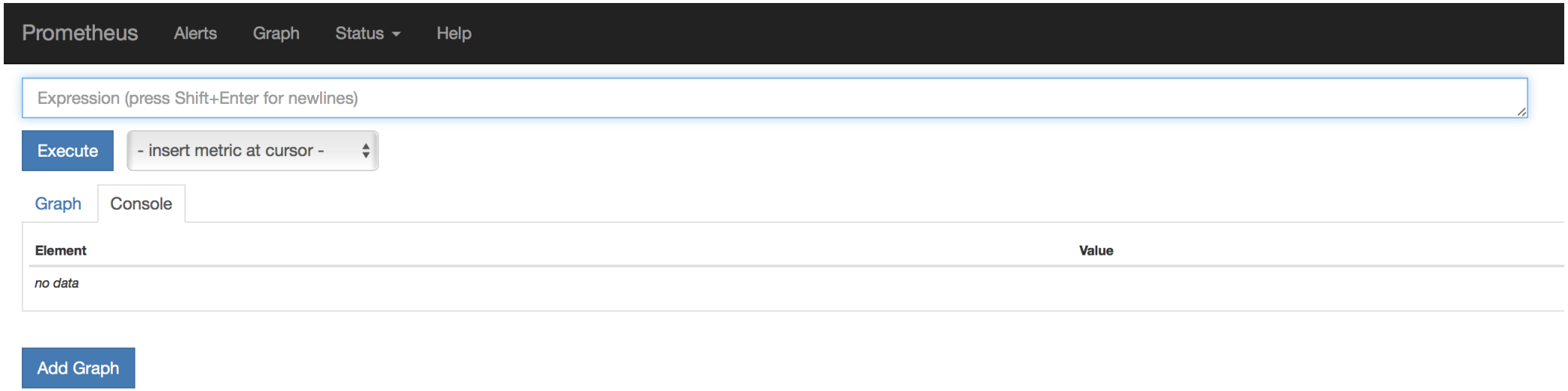
dynatrace

Datadog

© 2020 by Innovation In Software Corporation

MONITORING

Prometheus



© 2020 by Innovation In Software Corporation

MONITORING

Prometheus

Prometheus Alerts Graph Status Help

Targets

kubernetes-apiservers (1/1 up)

Endpoint	State	Labels	Last Scrape	Error
https://192.168.64.2:8443/metrics	UP	instance="192.168.64.2:8443"	49.262s ago	

kubernetes-cadvisor (1/1 up)

Endpoint	State	Labels	Last Scrape	Error
https://kubernetes.default.svc:443/api/v1/nodes/minikube/proxy/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="minikube" kubernetes_io_hostname="minikube"	46.023s ago	

kubernetes-nodes (1/1 up)

Endpoint	State	Labels	Last Scrape	Error
https://kubernetes.default.svc:443/api/v1/nodes/minikube/proxy/metrics	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="minikube" kubernetes_io_hostname="minikube"	8.864s ago	

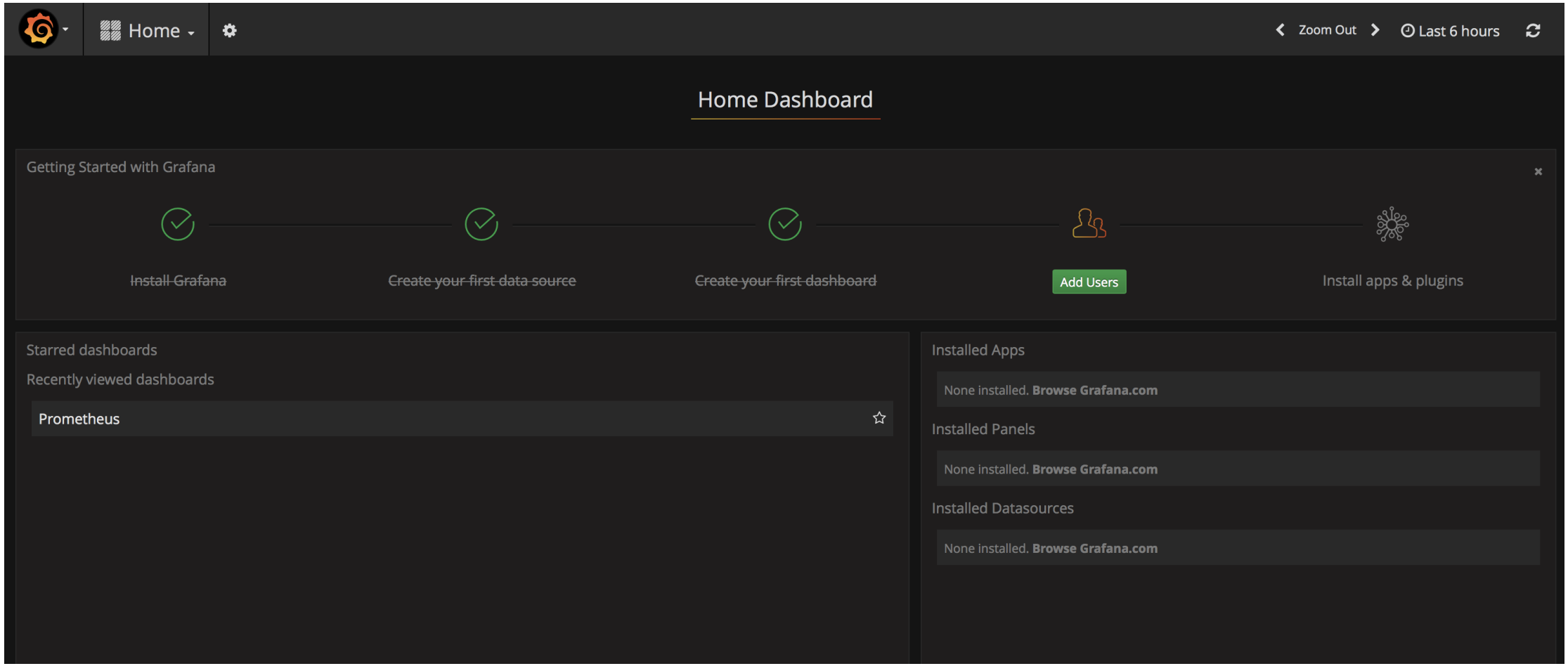
kubernetes-pods (1/1 up)

Endpoint	State	Labels	Last Scrape	Error
http://172.17.0.3:9090/metrics	UP	instance="172.17.0.3:9090" kubernetes_namespace="monitoring" kubernetes_pod_name="prometheus-5947dcc755-97qjd" name="prometheus" pod_template_hash="1503877311"	8.719s ago	

© 2020 by Innovation In Software Corporation

MONITORING

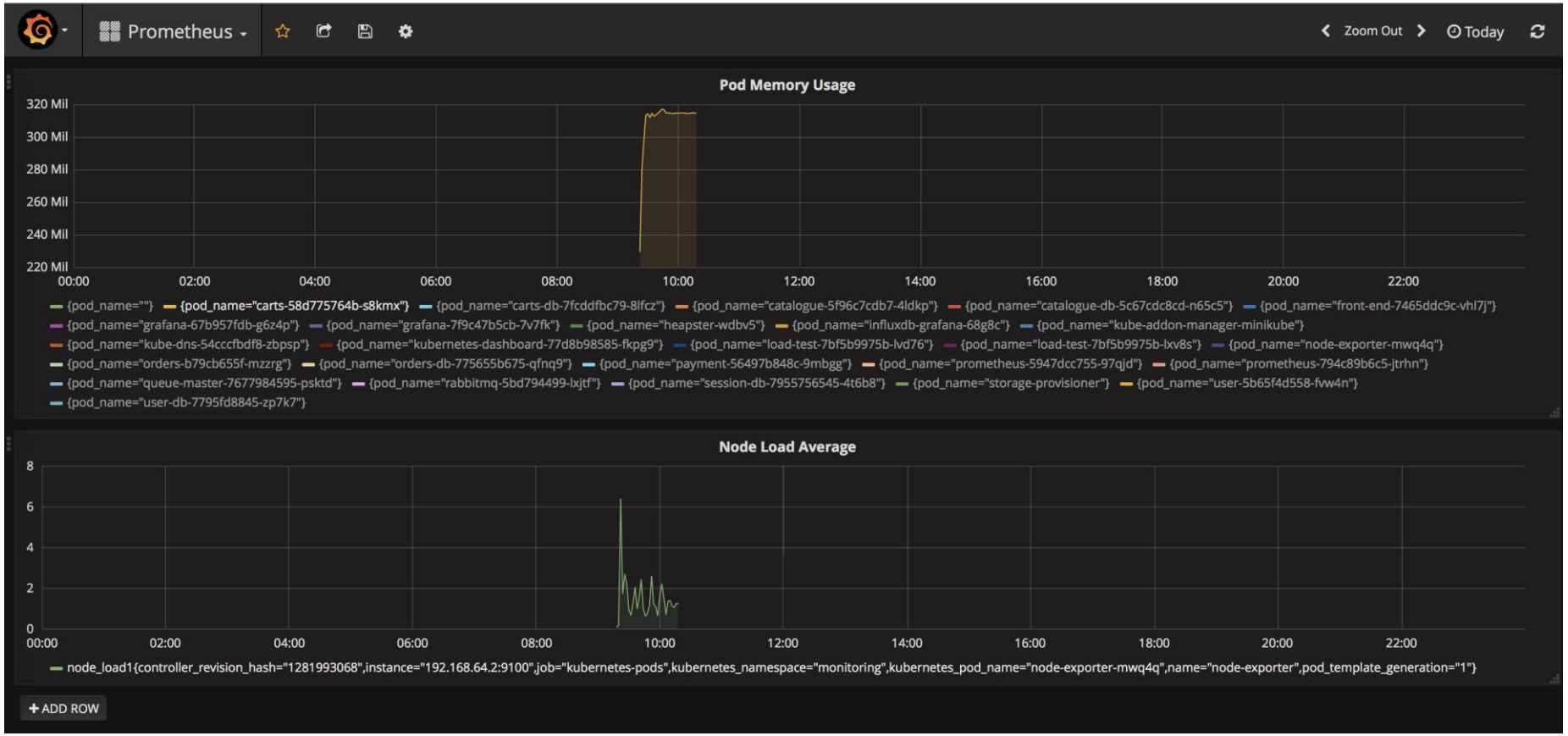
Grafana



© 2020 by Innovation In Software Corporation

MONITORING

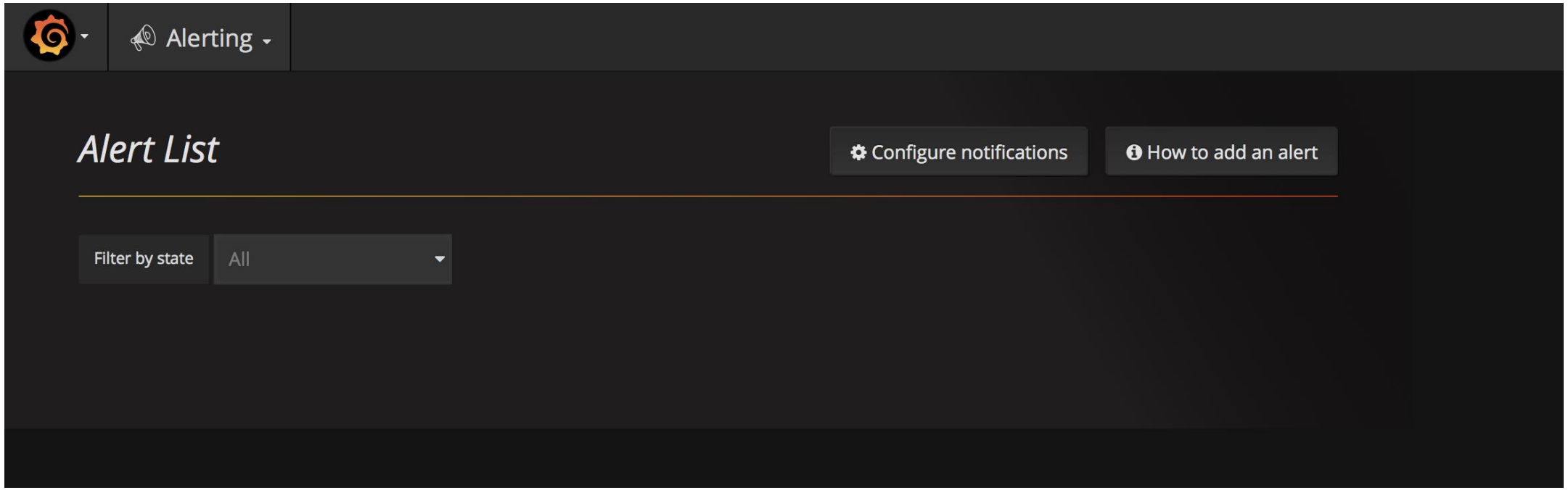
Grafana



© 2020 by Innovation In Software Corporation

MONITORING

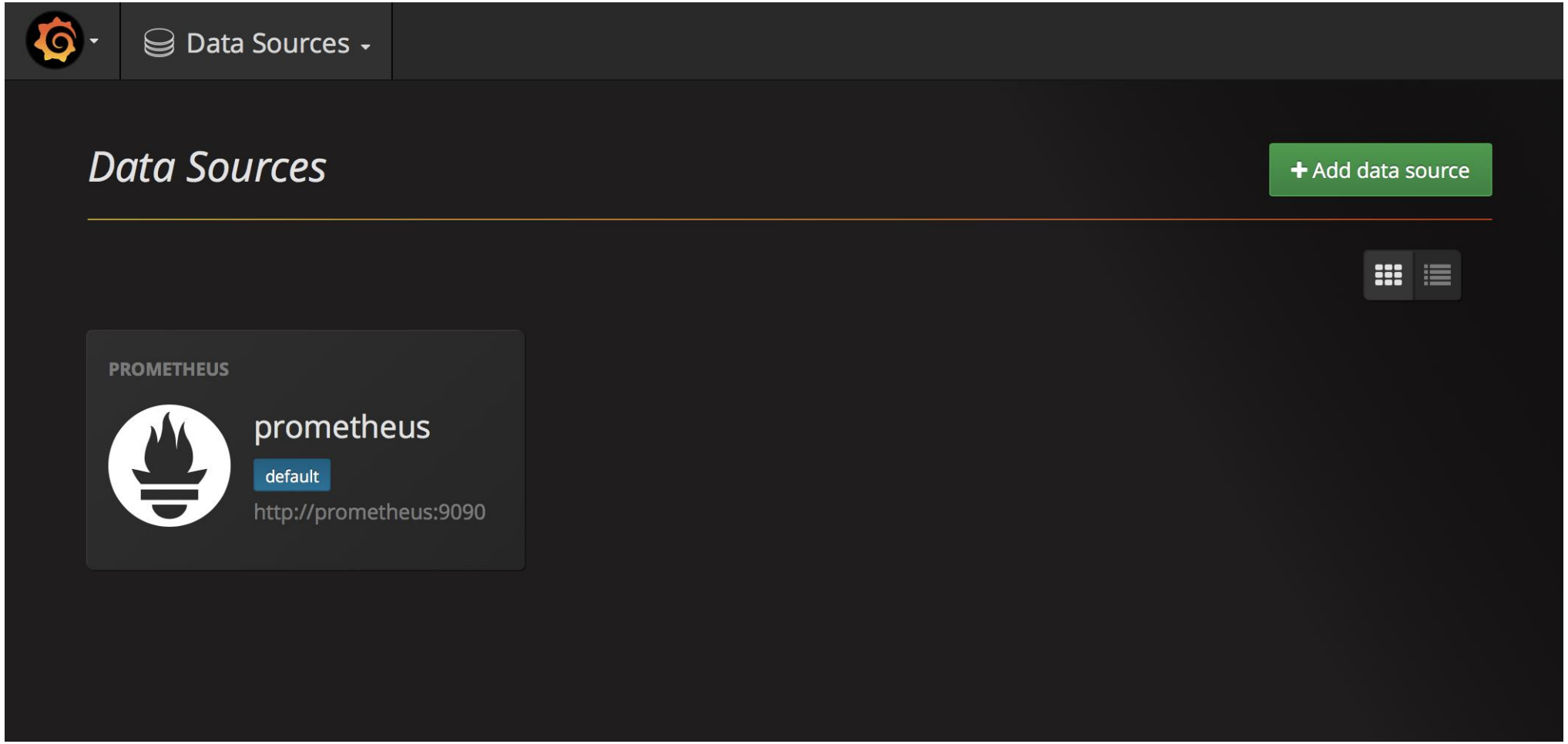
Grafana



© 2020 by Innovation In Software Corporation

MONITORING

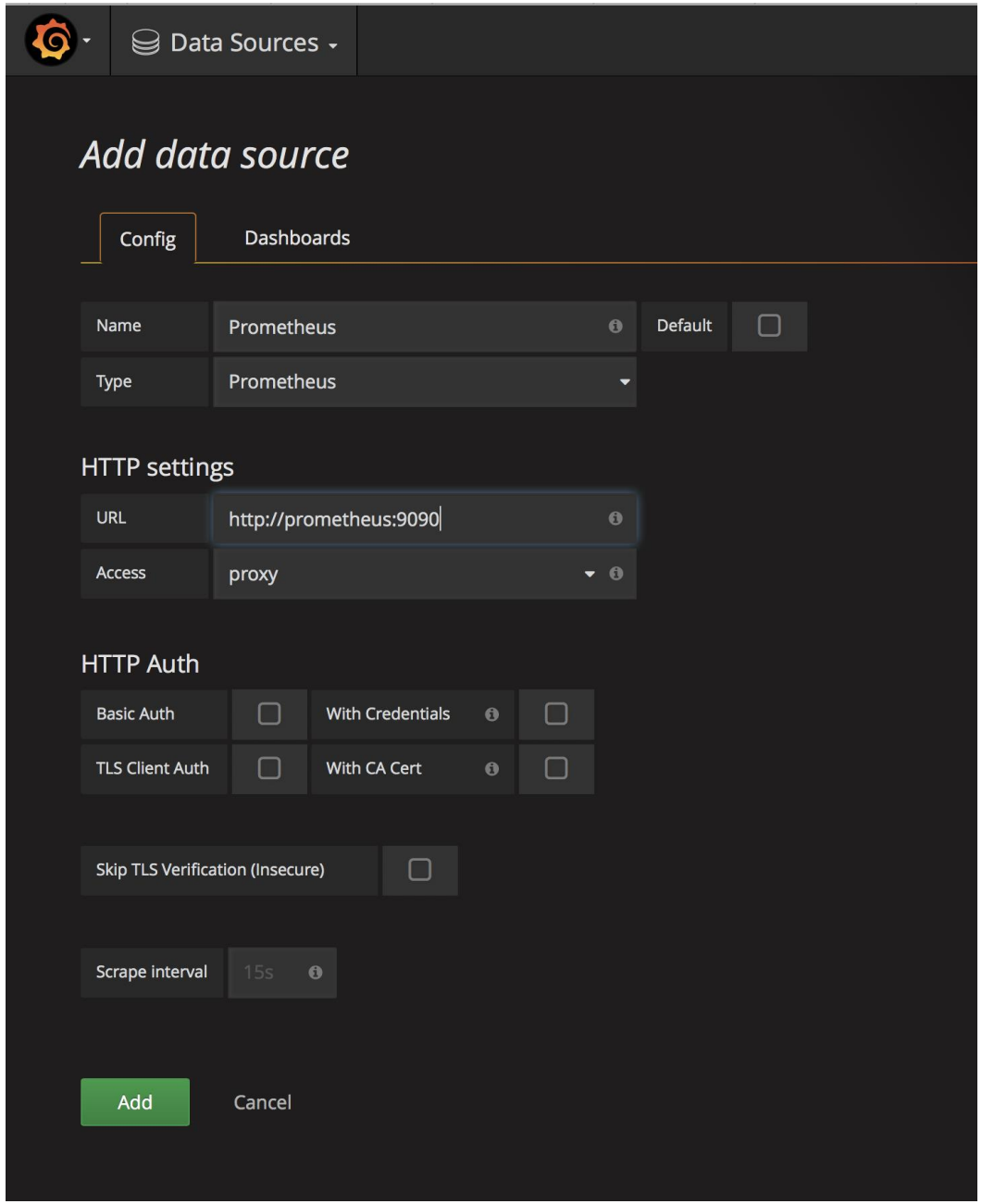
Grafana



© 2020 by Innovation In Software Corporation

MONITORING

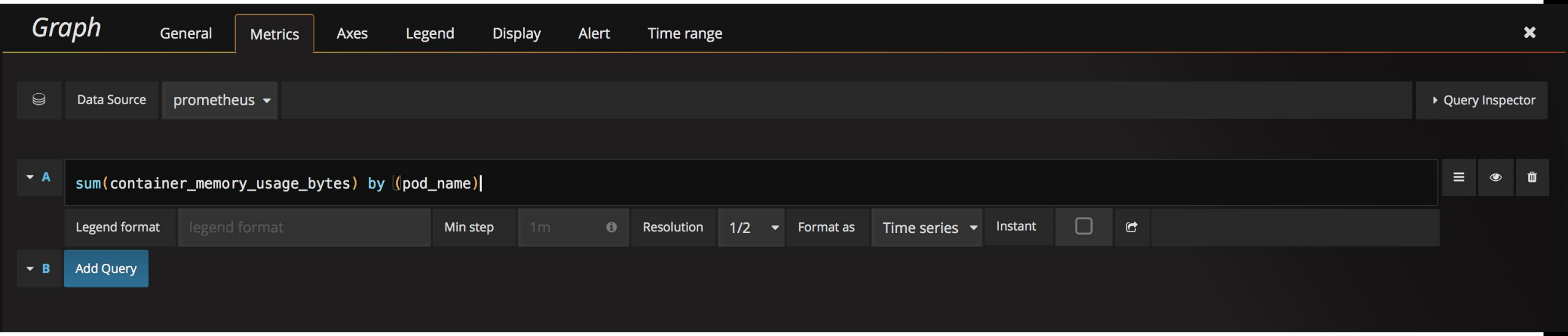
Grafana & Prometheus



© 2020 by Innovation In Software Corporation

MONITORING

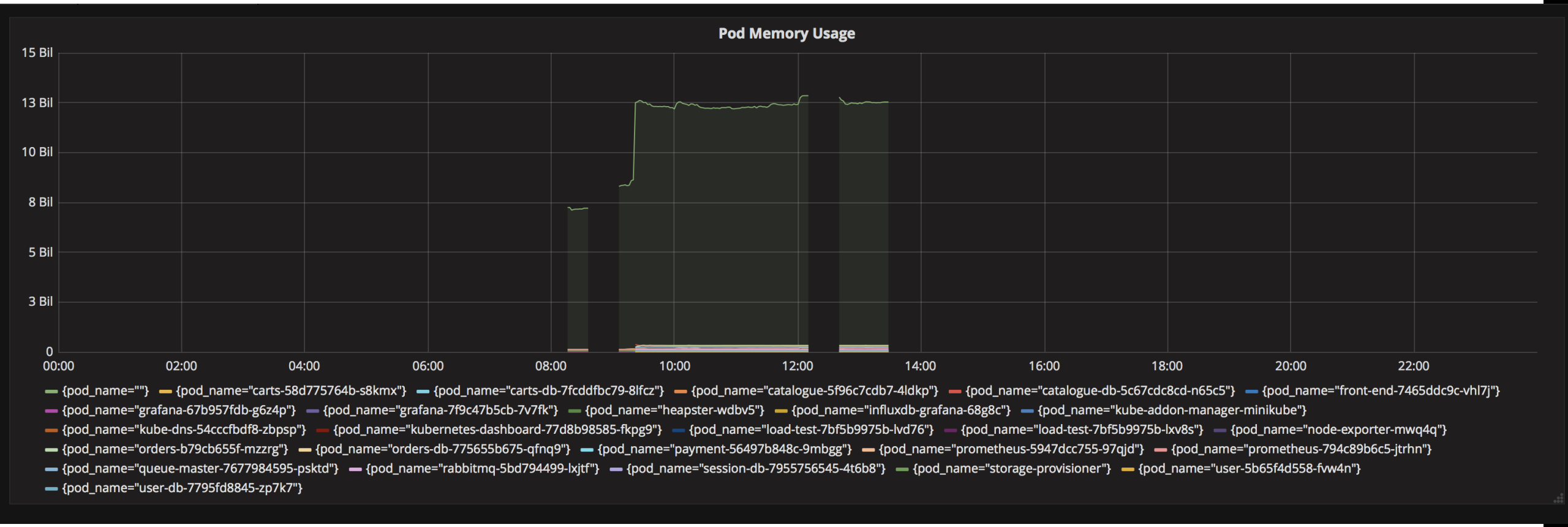
Add graph



© 2020 by Innovation In Software Corporation

MONITORING

Grafana graph



© 2020 by Innovation In Software Corporation

DEMO & LAB: GRAFANA

© 2020 by Innovation In Software Corporation

APPENDIX E

CONTEXT AND BOUNDARY

© 2020 by Innovation In Software Corporation

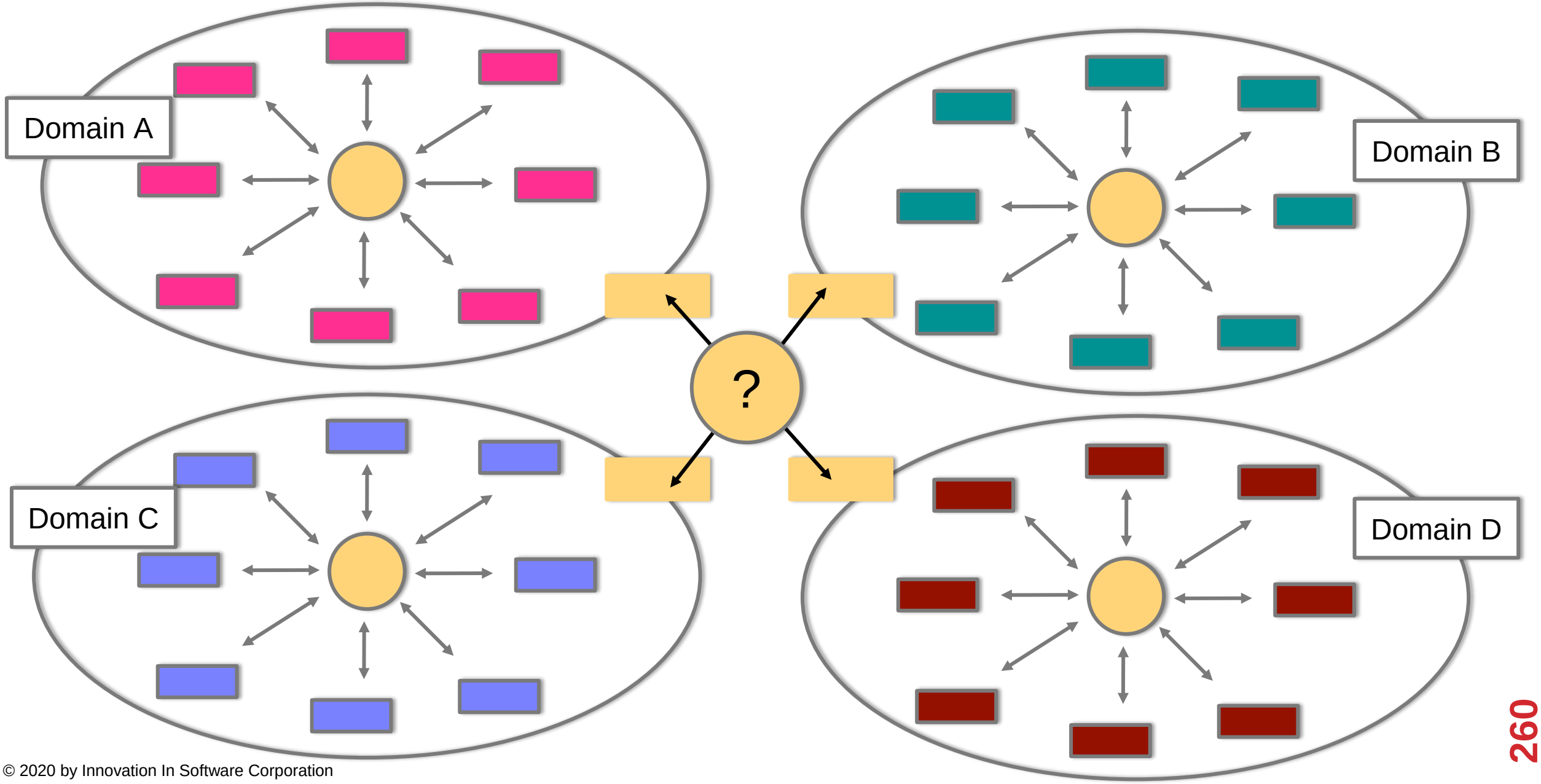
THE IMPORTANCE OF CONTEXT & BOUNDARY

Record: *noun*

Record: *verb*

© 2020 by Innovation In Software Corporation

THE IMPORTANCE OF CONTEXT: TRANSLATION



© 2020 by Innovation In Software Corporation

MICROSERVICE BOUNDARIES

As with most design decisions, many metrics are arbitrary

Single responsibility principle is the most important metric

A bounded context is a subdomain that is responsible for a particular function

- Strongly isolated, business function or capability

Several approaches can help with identifying bounded contexts

- Top-down domain decomposition
- Top-down use case decomposition
- Migration from existing, monolithic system

© 2020 by Innovation In Software Corporation

ESTABLISHING BOUNDARIES

Considerations in establishing boundaries

- Autonomous unit of functionality
- Size of deployable component
- High value unit when migrating from monolithic application
- Need for transactional integrity is a major consideration
- Scaling considerations
- Isolation considerations
- Volatility and evolutionary considerations

© 2020 by Innovation In Software Corporation

BOUNDARY CONSIDERATIONS

Autonomous unit of functionality

- Identify dependencies on external resources
- Work to minimize those dependencies
- If can eliminate dependencies, good boundary to consider

Size of deployable component

- Large deployable units present challenges for automated processes
- Constrict boundaries to support managing deployable units

© 2020 by Innovation In Software Corporation

BOUNDARY CONSIDERATIONS (CONT'D)

High value unit when migrating from monolithic application

- Look for functions with highest or most valuable usage

Need for transactional integrity is a major consideration

- Transaction boundaries should not cross service boundaries

© 2020 by Innovation In Software Corporation

BOUNDARY CONSIDERATIONS (CONT'D)

Scaling considerations

- A function may have different scaling needs than other, related functions
- Establishing a boundary based on scaling can make sense

Isolation considerations

- Function should be easily detachable from larger system
- Strive for high cohesion within the service and loose coupling between services
- View a microservice as a product by itself
    - Targeted consumers
    - Full cross-functional development team

© 2020 by Innovation In Software Corporation

BOUNDARY CONSIDERATIONS (CONT'D)

Volatility and evolutionary considerations

- New or experimental "products" may be a bounded context
- Develop Minimum Viable Products (MVPs)
- Allow system to evolve

© 2020 by Innovation In Software Corporation

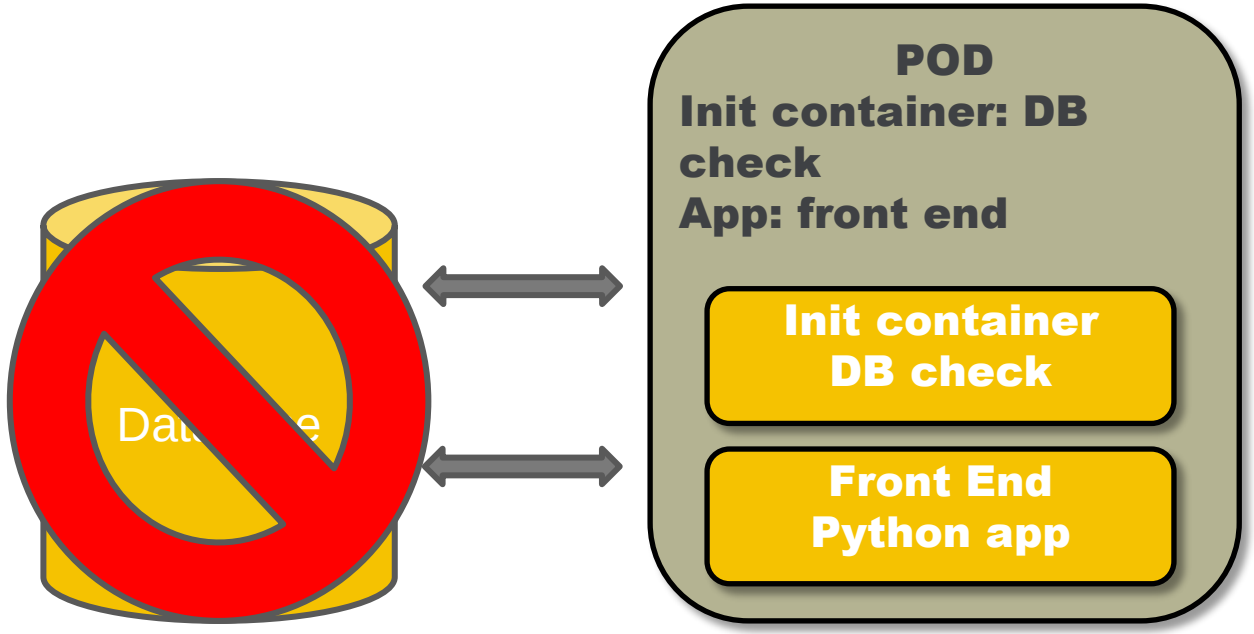
APPENDIX F
INIT
CONTAINERS

© 2020 by Innovation In Software Corporation

INIT CONTAINERS

Application orchestration

- **Init container runs first to completion**
  - If failure, POD is restarted until successful.



© 2020 by Innovation In Software Corporation

INIT CONTAINERS

Support same features as application containers.

- Resource limits
- Volumes
- Security settings
- Etc.

Do not support readiness probes since they must run to completion before POD can be “ready”

© 2020 by Innovation In Software Corporation

KUBERNETES 1.5 SYNTAX

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
  annotations:
    pod.beta.kubernetes.io/init-containers: '[
        {
            "name": "init-myservice",
            "image": "busybox",
            "command": ["sh", "-c", "until nslookup myservice; do echo waiting for myservice; sleep 2; done;"]
        },
        {
            "name": "init-mydb",
            "image": "busybox",
            "command": ["sh", "-c", "until nslookup mydb; do echo waiting for mydb; sleep 2; done;"]
        }
    ]'
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
```

© 2020 by Innovation In Software Corporation

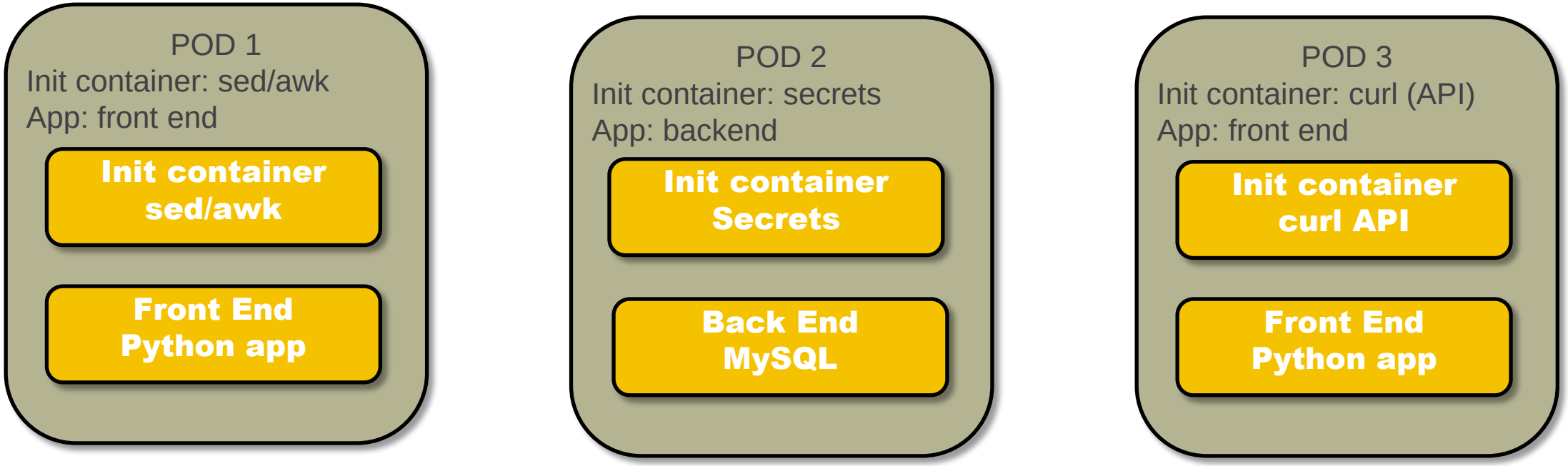
KUBERNETES 1.6+ SYNTAX

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb
    image: busybox
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

© 2020 by Innovation In Software Corporation

INIT CONTAINER USE-CASES

Lots of reasons



© 2020 by Innovation In Software Corporation

DEMO & LAB: K8S INIT CONTAINERS

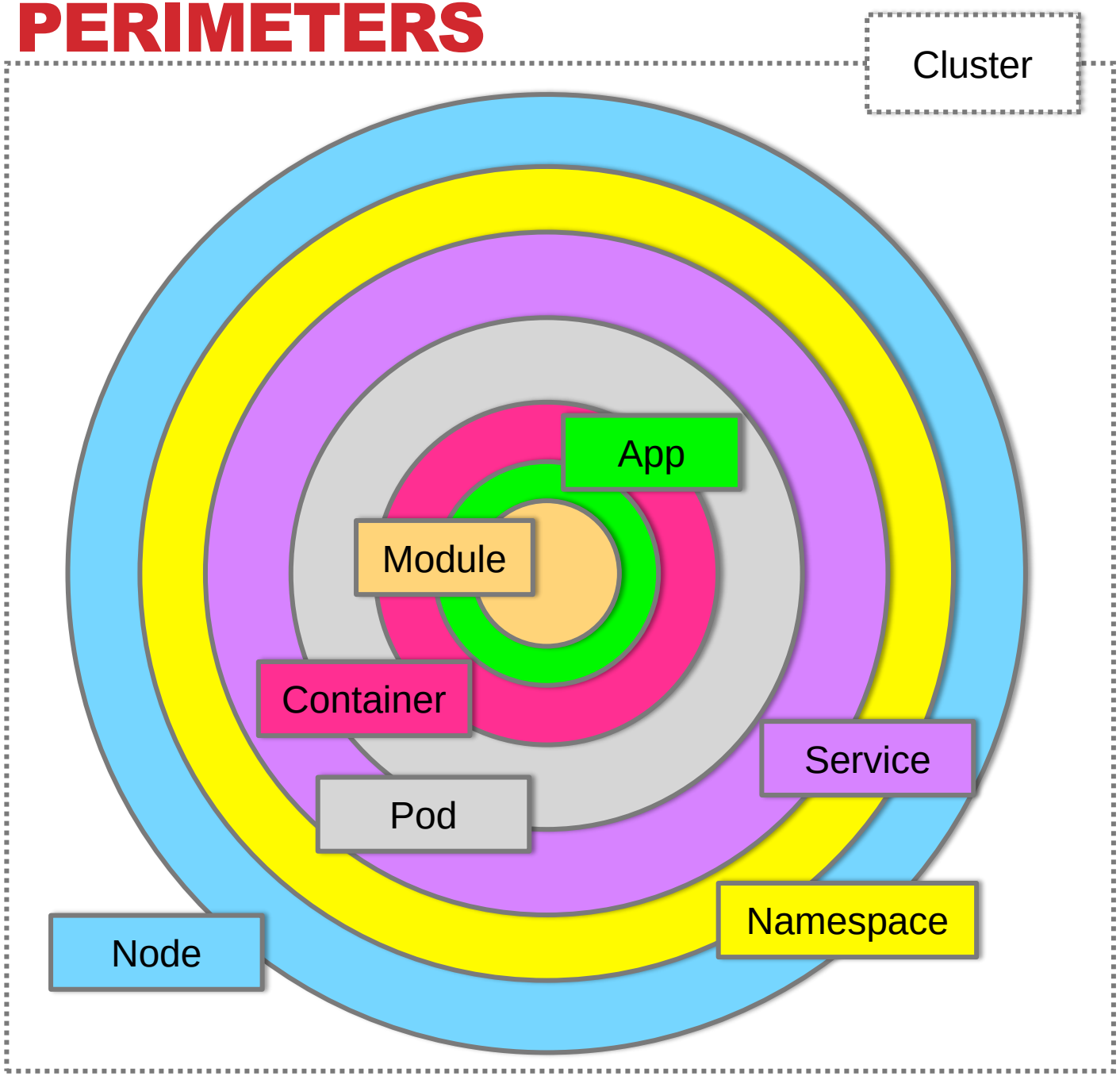
© 2020 by Innovation In Software Corporation

APPENDIX G

K8S SECURITY

© 2020 by Innovation In Software Corporation

SECURITY PERIMETERS



© 2020 by Innovation In Software Corporation

SECURITY ARTIFACTS

- Username/Password
- JWT Token
- Public/Private Key (TLS)

SECURITY TECHNIQUES

- OAuth
- Namespace Restriction
- Routing Rules
- Ingress
- Egress

© 2020 by Innovation In Software Corporation

STORING SECRETS INSIDE KUBERNETES

What secrets do applications have?

- Database credentials
- API credentials & endpoints (Twitter, Facebook etc.)
- Infrastructure API credentials (Google, Azure, AWS)
- Private keys (TLS, SSH)
- Many more!

© 2020 by Innovation In Software Corporation

STORING SECRETS INSIDE KUBERNETES

It is a bad idea to include these secrets in your code.

- Accidentally push up to GitHub with your code
- Push into your file storage and forget about
- Etc.

© 2020 by Innovation In Software Corporation

STORING SECRETS INSIDE KUBERNETES

There are bots crawling GitHub searching for secrets

Real life example:

- Dev put keys out on GitHub, woke up next morning with a ton of emails and missed calls from Amazon
- 140 instances running under Dev's account.
- $2,375 worth of Bitcoin mining.

© 2020 by Innovation In Software Corporation

CREATE SECRET

- **Designed to hold all kinds of sensitive information**
- **Can be used by Pods (filesystem & environment variables) and the underlying kubelet when pulling images**

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
Type: Opaque
data:
   password: mmyWfoidfIuL==
   username: NyhdOKwB
```

© 2020 by Innovation In Software Corporation

POD SECRET

```
kind: Pod
metadata:
   name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
    - name: SECRET_USERNAME
        valueFrom:
            secretKeyRef:
               name: mysecret
               key: username
```

© 2020 by Innovation In Software Corporation

VOLUME SECRET

```
spec:
  containers
  - name: mycontainer
    image: redis
    volumeMounts:
    - name: “secrets”
      mountPath: “/etc/my-secrets”
      readOnly: true
  volumes:
   - name: “secrets”
      secret:
        secretName: “mysecret”
```

© 2020 by Innovation In Software Corporation

DEMO: K8S SIMPLE SECRETS

© 2020 by Innovation In Software Corporation

DEMO: USING A K8S TLS SECURITY SECRET

© 2020 by Innovation In Software Corporation

APPENDIX H

MICROSERVICE PATTERNS

© 2020 by Innovation In Software Corporation

MICROSERVICES PATTERNS

- Aggregator
- Branch
- Proxy
- Chained
- Shared Resources (Data)
- Asynchronous Messaging
- Bulkhead Isolation

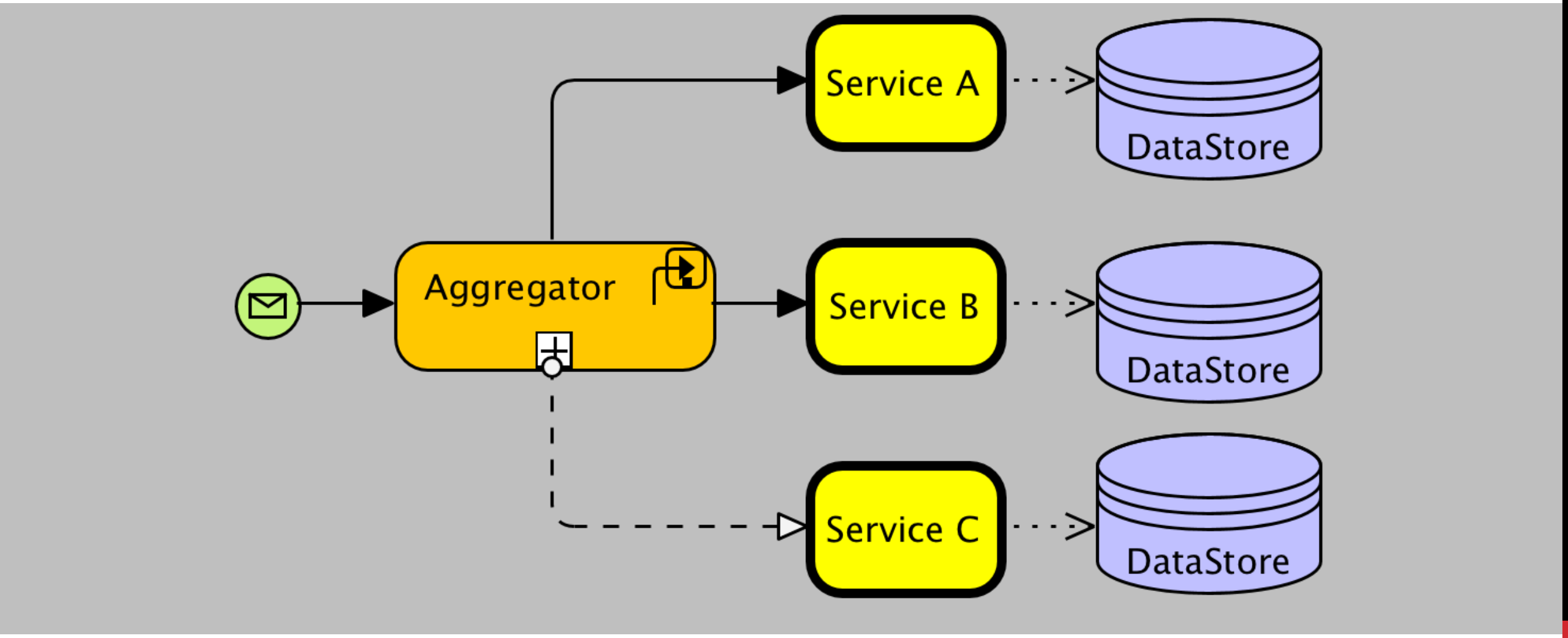
© 2020 by Innovation In Software Corporation

PATTERN: AGGREGATOR

Description:

- An Aggregator is a web page that invokes multiple services to achieve the required functionality
- An Aggregator could also be a composite service
- This design pattern follows the DRY principle
- An Aggregator is similar to the Façade Pattern

© 2020 by Innovation In Software Corporation



PATTERN: AGGREGATOR

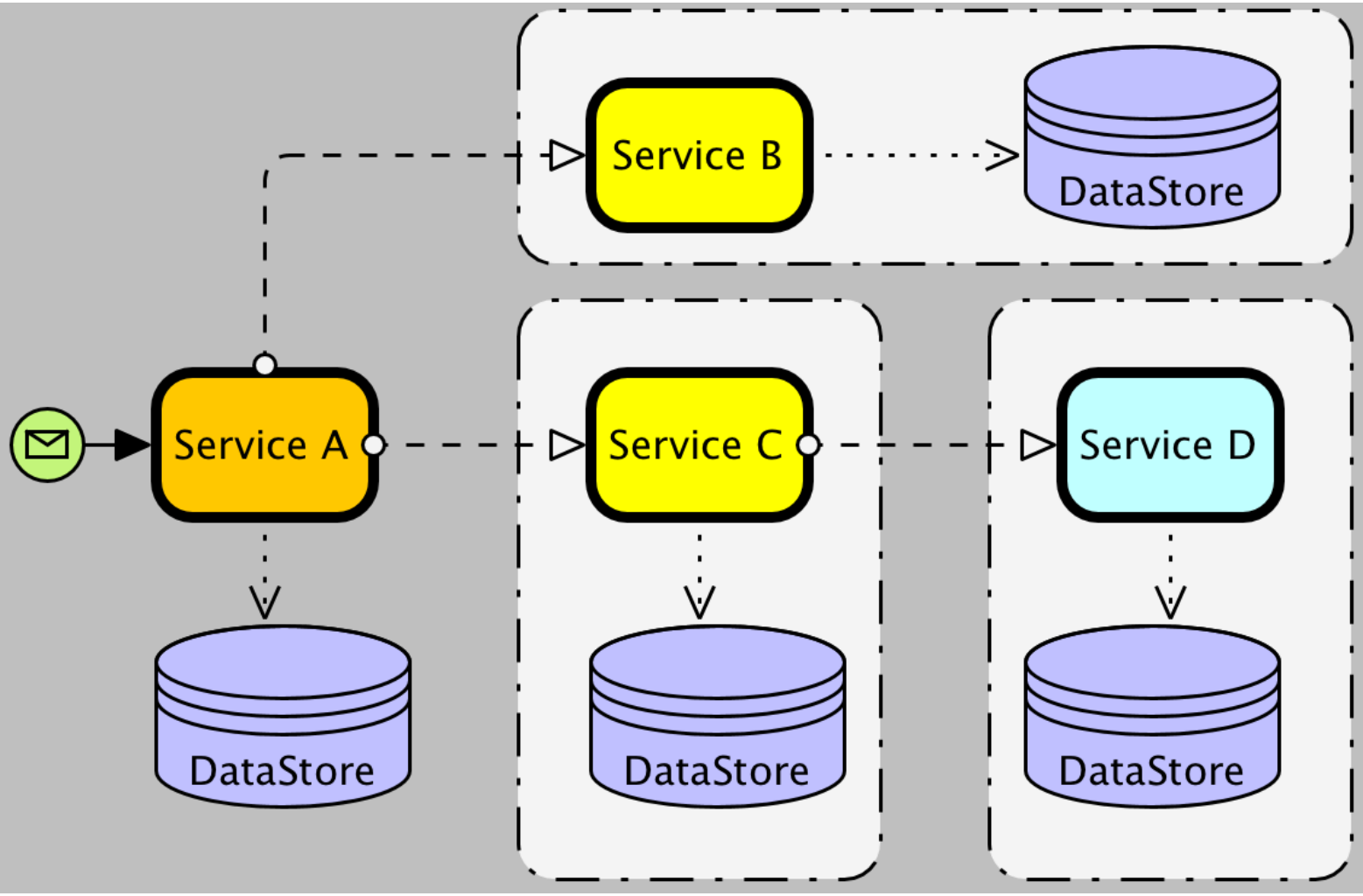
© 2020 by Innovation In Software Corporation

PATTERN: BRANCH

Description:

- The Branch design pattern extends the Aggregator design pattern
- Allows simultaneous response processing from two, likely mutually exclusive, chains of microservices
- This pattern can also be used to call different chains, or a single chain, based upon the business needs
- Service A, either a web page or a composite microservice, can invoke two different chains concurrently in which case this will resemble the Aggregator design pattern
- This may be configured using routing of Enterprise Integration Pattern (EIP) endpoints, and would need to be dynamically configurable

© 2020 by Innovation In Software Corporation



PATTERN: BRANCHING

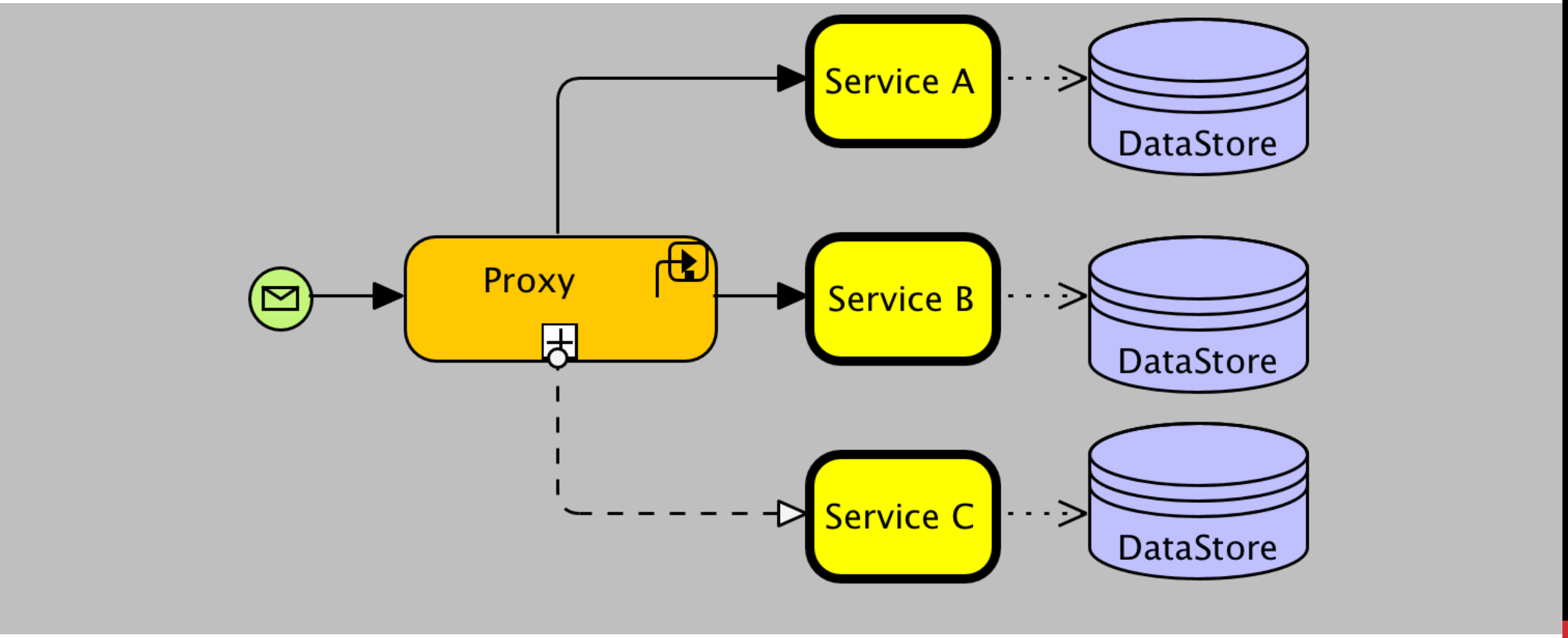
© 2020 by Innovation In Software Corporation

PATTERN: PROXY

Description:

- Proxy microservice design pattern is a variation of the Aggregator Pattern
- Different microservice may be invoked based upon the business needs
- Each Proxy can scale independently
- A proxy is a consumer facing interface in lieu of exposing individual services
- May be a dumb proxy that delegates requests to one of the services
- May be a Smart Proxy, where data is transformed before response returned to consumer

© 2020 by Innovation In Software Corporation



PATTERN: PROXY

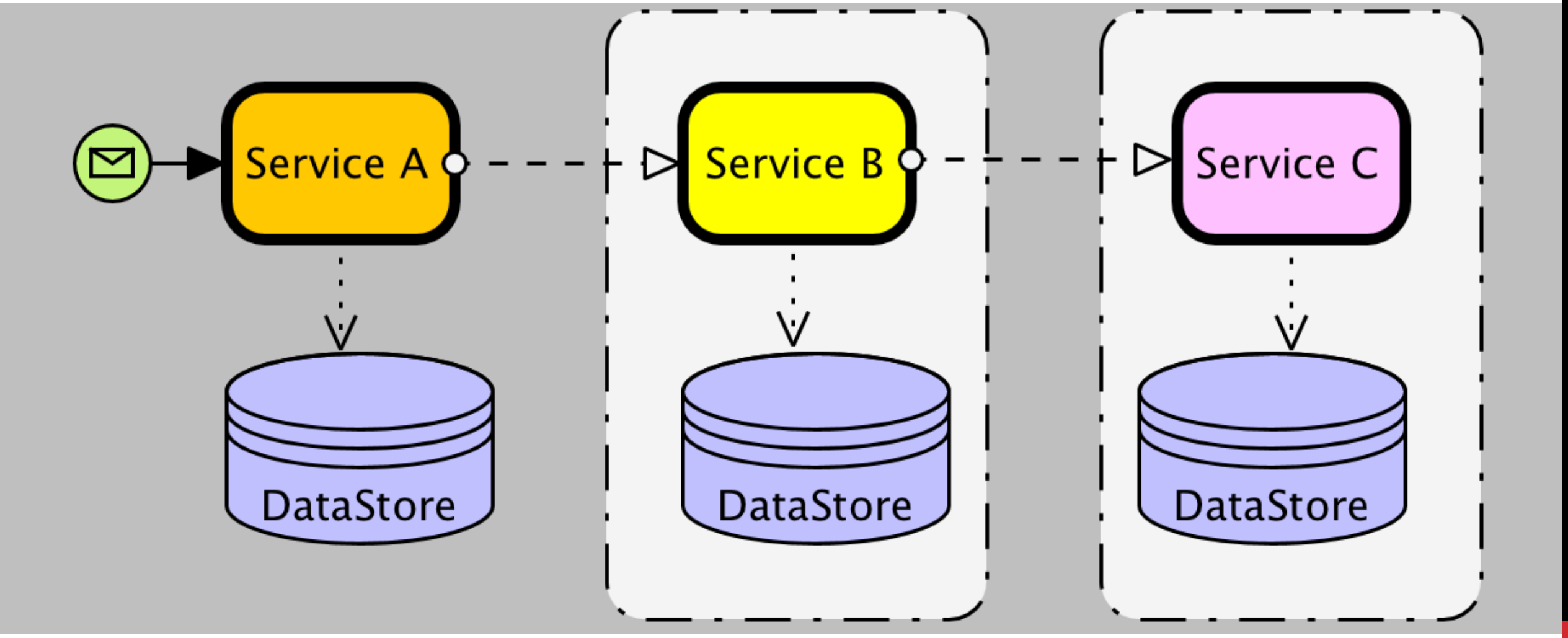
© 2020 by Innovation In Software Corporation

PATTERN: CHAIN

Description:

- Produces single consolidated response to request
- Request is received by Service A, which is chained to Service B, which in turn which is changed to Service C
- Services are likely using a synchronous HTTP request/response messaging
- Client is blocked until request/response chain is completed
- Important not to make chains too long
- A chain with a single microservice is called singleton chain

© 2020 by Innovation In Software Corporation



PATTERN: CHAIN

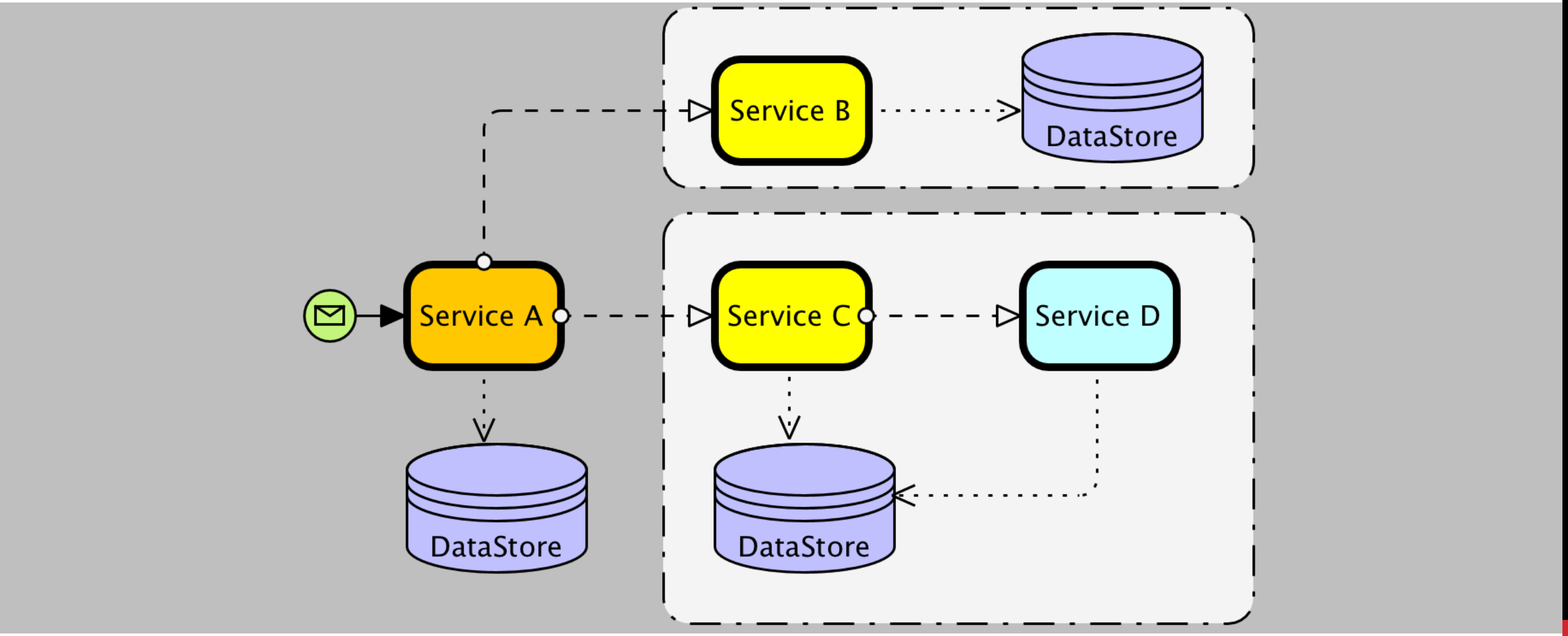
© 2020 by Innovation In Software Corporation

PATTERN: SHARED RESOURCES (DATA)

Description:

- A fundamental design principle of microservice is autonomy
- A service should be full-stack and have control of all the components
    - UI, middleware, persistence, transaction, etc.
- Service should be polyglot
    - Use the right tool for the right job
- Data normalization is typical issue when refactoring existing monolithic applications
- This pattern may benefit from shared caching and database stores
- This is a transitional pattern until service can be fully autonomous

© 2020 by Innovation In Software Corporation



PATTERN: SHARED RESOURCE (DATA)

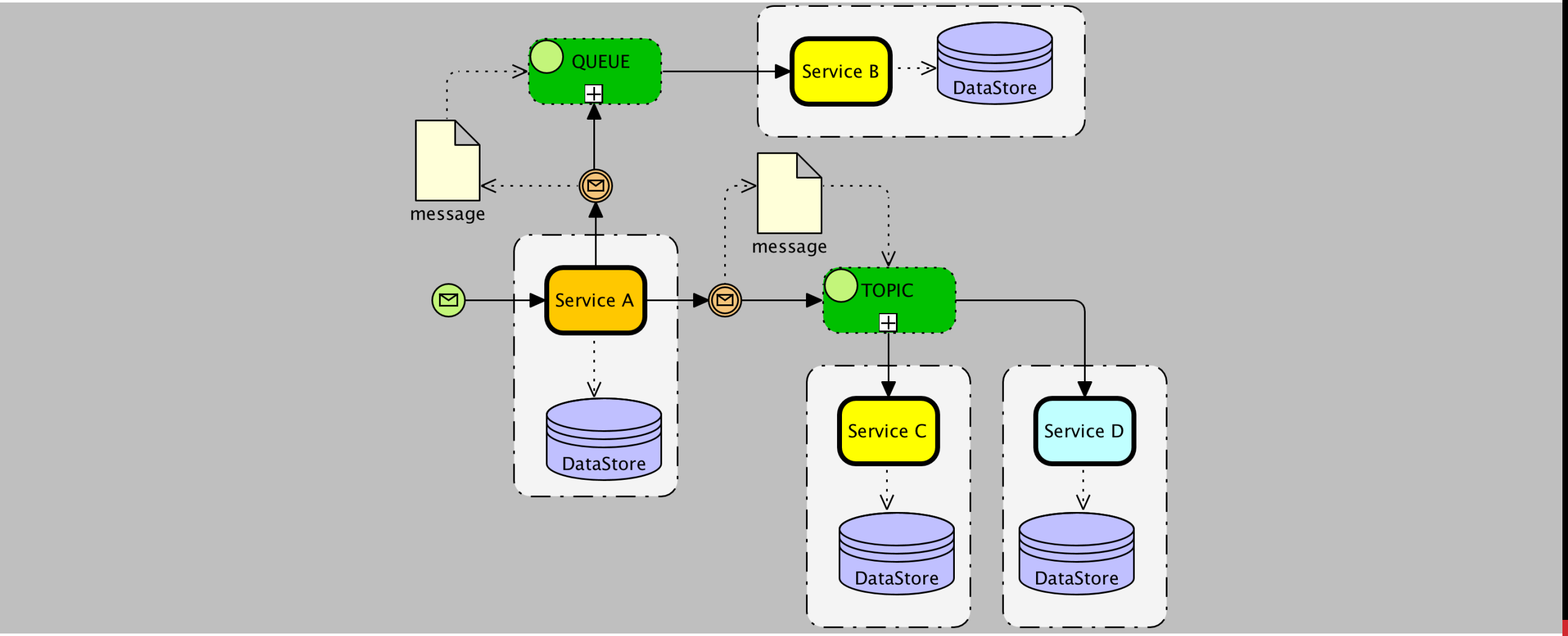
© 2020 by Innovation In Software Corporation

PATTERN: ASYNCHRONOUS MESSAGING

Description:

- RESTful services are inherently synchronous, and thus blocking
- Microservice architectures may use asynchronous messaging instead of RESTful request/response
- A service may call another service synchronously, which then communicates with other services asynchronously
- Could communicate asynchronously in many different forms:
    - WebSockets, Message Queues, Reactive Streams
- Can combine RESTful service and publish/subscribe messaging to accomplish the business needs

© 2020 by Innovation In Software Corporation



PATTERN: ASYNCHRONOUS MESSAGING

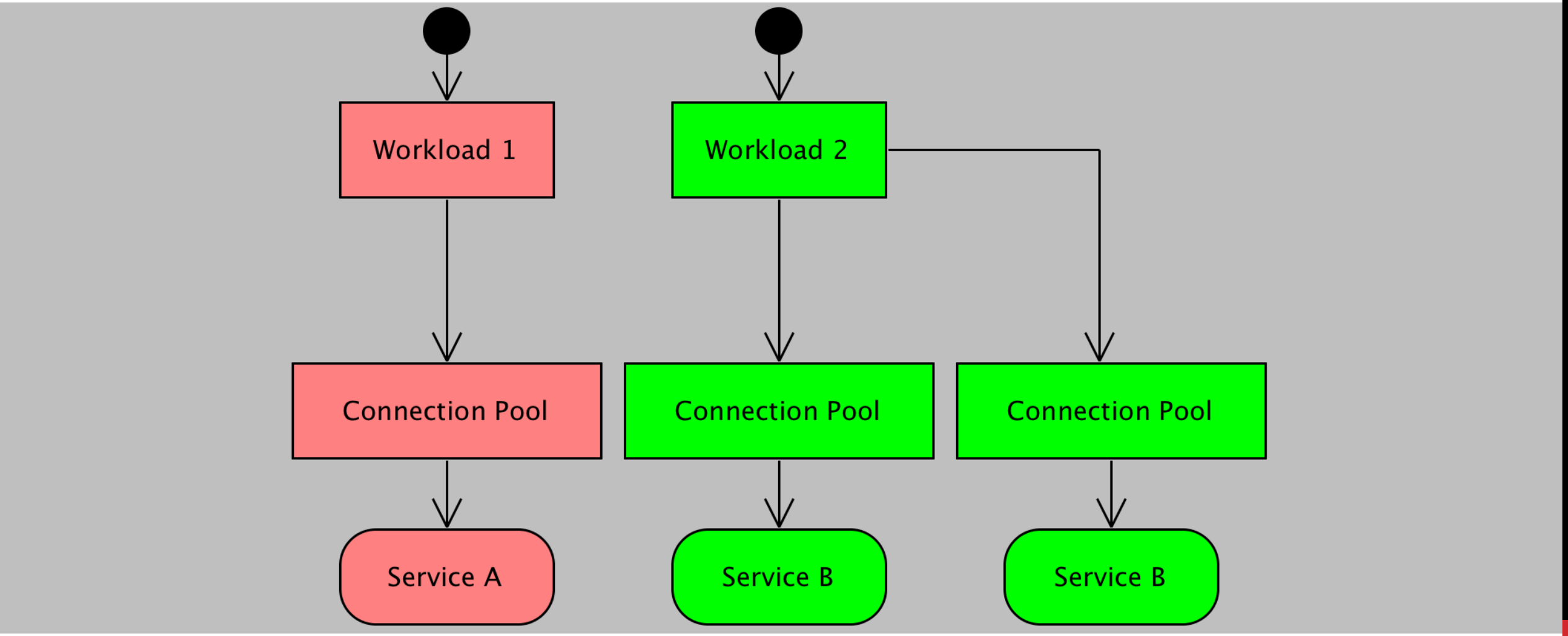
© 2020 by Innovation In Software Corporation

PATTERN: BULKHEAD ISOLATION

Description:

- Isolate elements of an application into pools so that if one fails, the others will continue to function
- This pattern resembles the sectioned partitions of a ship's hull
  - If ship hull is compromised, only the damaged section fills with water, which prevents the ship from sinking
- Partition service instances into different groups, based on consumer load and availability requirements
- Isolating failures, and allowing application to sustain service functionality for some consumers, even during a failure

© 2020 by Innovation In Software Corporation



PATTERN: BULKHEAD

© 2020 by Innovation In Software Corporation

PATTERN: BULKHEAD ISOLATION

Issues and considerations

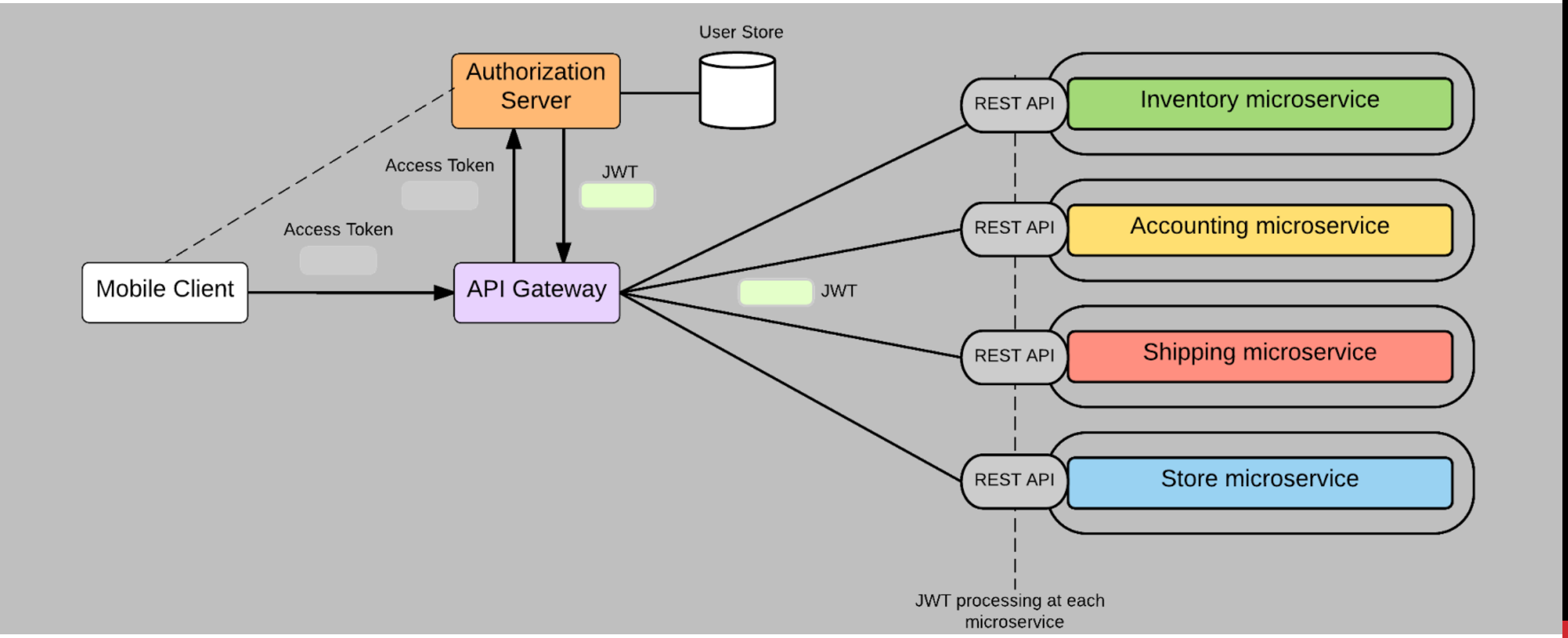
- Define partitions around the business and technical requirements of the application
- Consider level of isolation offered by the technology as well as the overhead in terms of cost, performance and manageability
- Consider combining bulkheads with retry, circuit breaker, and throttling patterns to provide more sophisticated fault handling
- Use processes, thread pools, and semaphores
- Use frameworks for creating consumer bulkheads
- Deploy into separate VM's, containers, or processes
- Isolate asynchronous service into different queues
- Monitor each partition's performance and SLA

© 2020 by Innovation In Software Corporation

APPENDIX I

API GATEWAY

© 2020 by Innovation In Software Corporation



API GATEWAY

© 2020 by Innovation In Software Corporation

API GATEWAYS

A single point of entry into the system.

The API is used to encapsulate the entire system and provided only the required API for the service to the client.

The nature of the API definition depends on which IPC mechanism you are using.

- If you are using messaging, the API consists of the message channels and the message types.
- If you are using HTTP, the API consists of the URLs and the request and response formats.

© 2020 by Innovation In Software Corporation

API GATEWAYS (CONT'D)

It might have other responsibilities such as:

- Authentication
- Monitoring
- Load balancing
- Caching
- Static response handling
- More…

© 2020 by Innovation In Software Corporation

API GATEWAY CONSIDERATIONS

Benefits

- A major benefit of using an API Gateway is that it encapsulates the internal structure of the application.

Drawbacks

- There is also a risk that the API Gateway becomes a development bottleneck.
- Developers must update the API Gateway in order to expose each microservice's endpoints.

© 2020 by Innovation In Software Corporation

API GATEWAY (AUTHENTICATION)

Authentication

API Gateways usually handle the authentication and authorization from the external callers to the microservice level.

© 2020 by Innovation In Software Corporation

API GATEWAY (AUTHENTICATION)

OAuth 2.0

OAuth delegated authorization with JSON Web Tokens (JWT) is an efficient and scalable solution for authentication and authorization for microservices.

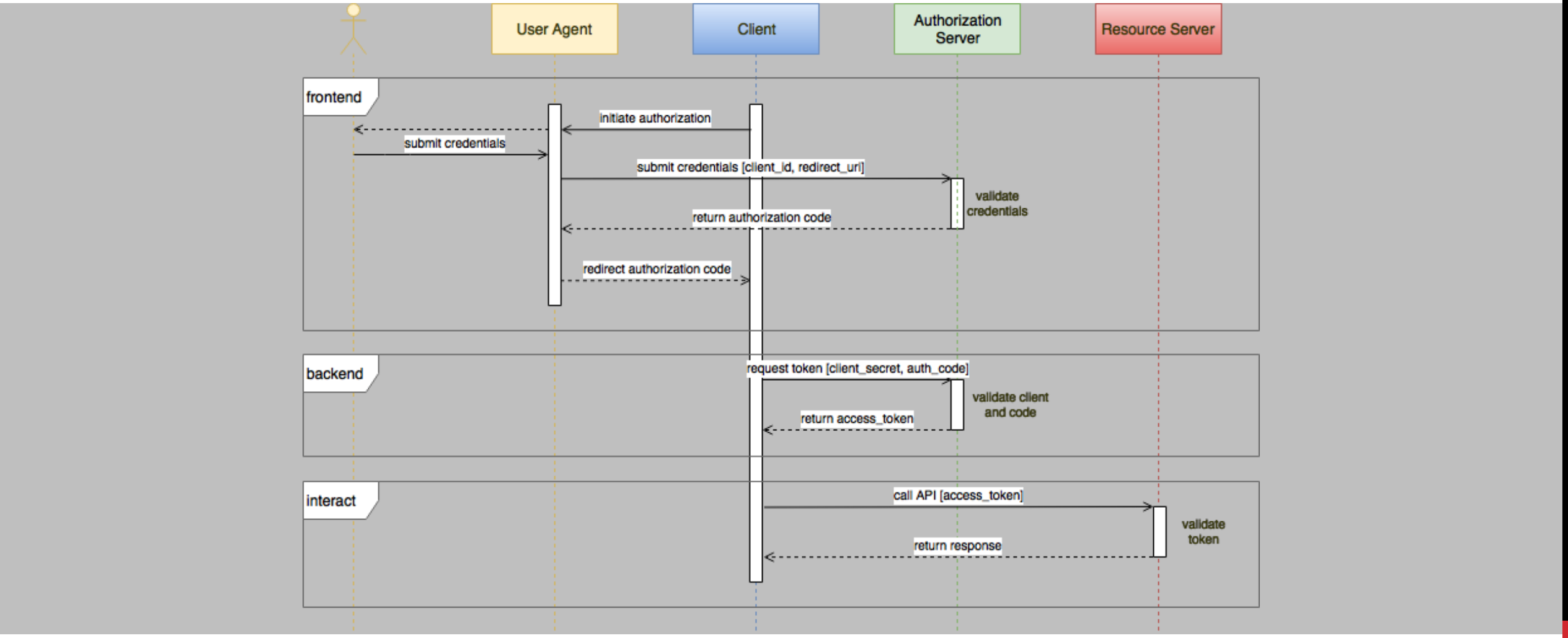
© 2020 by Innovation In Software Corporation

API GATEWAY (AUTHENTICATION)

JSON Web Token (JWT)

A JSON web token, or JWT is a standardized, optionally validated and/or encrypted container format that is used to securely transfer information between two parties.

© 2020 by Innovation In Software Corporation



API GATEWAY (OAUTH)

OAuth and JWT Authentication

© 2020 by Innovation In Software Corporation

API PROXYING

In simple terms you can think of the Proxy providing you a new endpoint for an existing API

Adds some lightweight security, monitoring, etc.

© 2020 by Innovation In Software Corporation

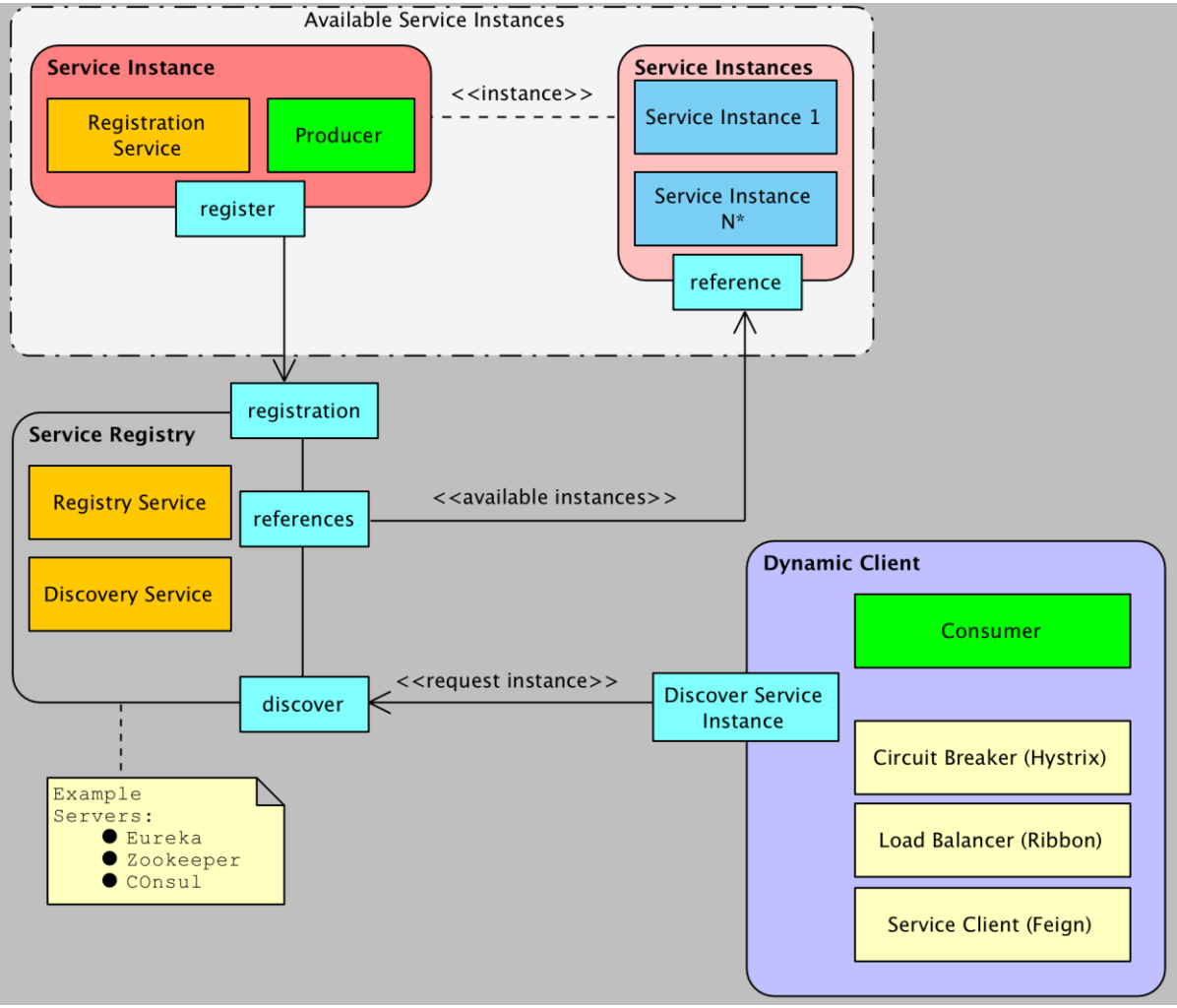
DEMO: IMPLEMENTING AN API GATEWAY ON AWS

© 2020 by Innovation In Software Corporation

APPENDIX J

SERVICE DISCOVERY

© 2020 by Innovation In Software Corporation



MICROSERVICE REGISTRY / DISCOVERY

© 2020 by Innovation In Software Corporation

MICROSERVICE REGISTRY / DISCOVERY

Context

Clients of a service use either Client-side discovery or Server-side discovery to determine the location of a service instance to which to send requests.

© 2020 by Innovation In Software Corporation

MICROSERVICE REGISTRY / DISCOVERY

Issue

How do clients of a service (in the case of Client-side discovery) and/or routers (in the case of Server-side discovery) know about the available instances of a service?

© 2020 by Innovation In Software Corporation

MICROSERVICE REGISTRY / DISCOVERY

Forces

- Each instance of a service exposes a remote API such as HTTP/REST at a particular location (host:port)
- Number of services instances and their locations changes dynamically.
- Virtual machines and containers are assigned a dynamic IP address.
- Can adjust the number of instances based on load.

© 2020 by Innovation In Software Corporation

MICROSERVICE REGISTRY / DISCOVERY

Solution

- Implement a service registry, which is a database of services, their instances and their locations.
- Service instances are registered with the service registry on startup and deregistered on shutdown.
- Client of the service and/or routers query the service registry to find the available instances of a service.
- A service registry might invoke a service instance's health check API to verify that it is able to handle requests

© 2020 by Innovation In Software Corporation

MICROSERVICE DISCOVERY (SERVER-SIDE)

Issue

- How does the client of a service - the API gateway or another service - discover the location of a service instance?

© 2020 by Innovation In Software Corporation

MICROSERVICE DISCOVERY (SERVER-SIDE)

Forces

- Each instance of a service exposes a remote API such as HTTP/REST at a particular location (host:port)
- Number of services instances and their locations changes dynamically.
- Virtual machines and containers are assigned a dynamic IP address.
- Can adjust the number of instances based on load.

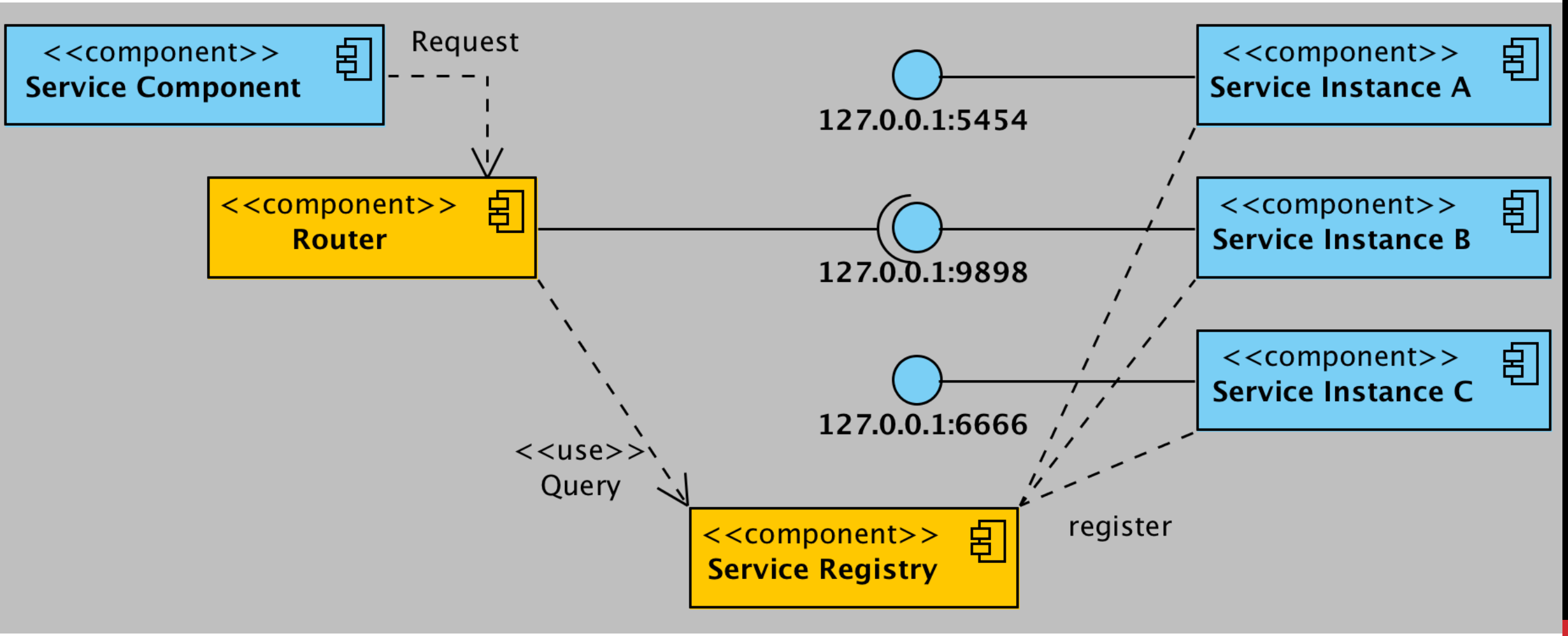
© 2020 by Innovation In Software Corporation

MICROSERVICE DISCOVERY (SERVER-SIDE)

Solution

- When making a request to a service, the client makes a request via a router (load balancer) that runs at a well known location.
- Router queries a service registry, which might be built into the router, and forwards the request to an available service instance.

© 2020 by Innovation In Software Corporation



MICROSERVICE DISCOVERY (SERVER-SIDE)

© 2020 by Innovation In Software Corporation

MICROSERVICE DISCOVERY (SERVER-SIDE)

Forces

- Each instance of a service exposes a remote API such as HTTP/REST at a particular location (host:port)
- Number of services instances and their locations changes dynamically.
- Virtual machines and containers are assigned a dynamic IP address.
- Can adjust the number of instances based on load.

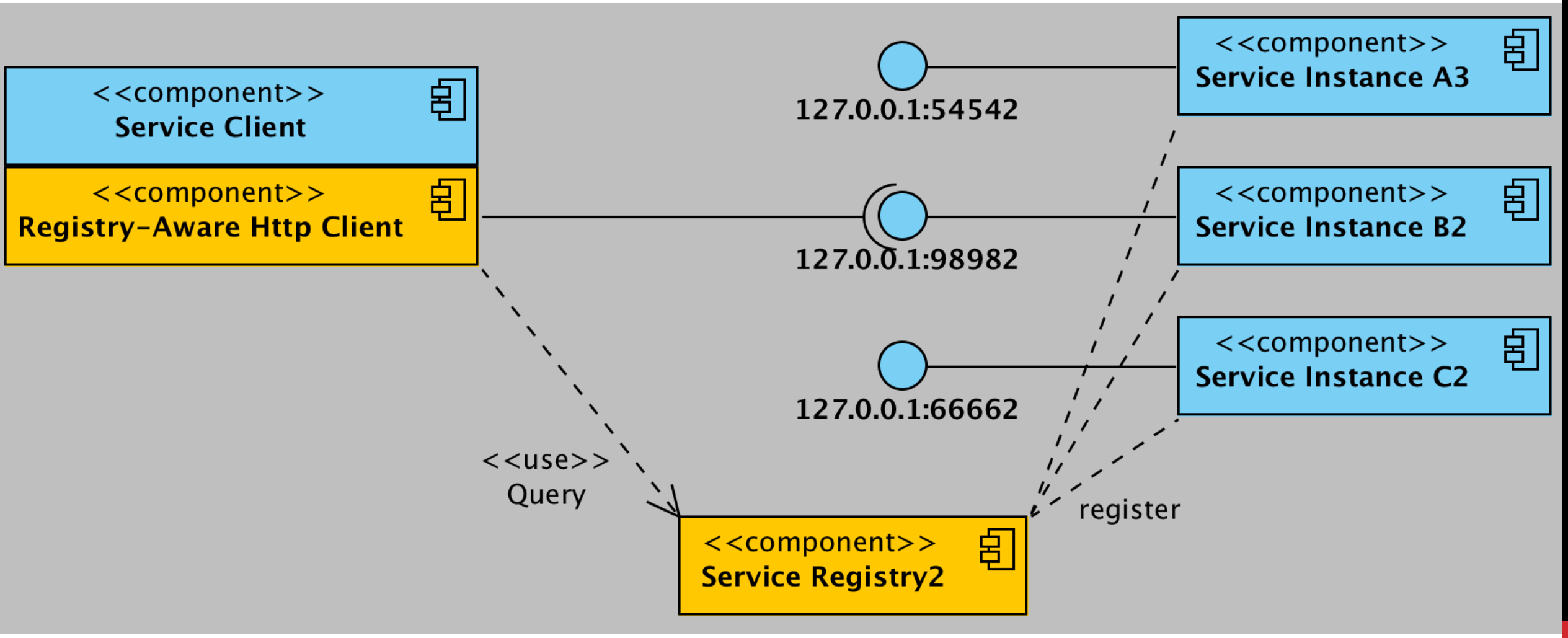
© 2020 by Innovation In Software Corporation

MICROSERVICE DISCOVERY (SERVER-SIDE)

Solution

- When making a request to a service, the client obtains the location of a service instance by querying a Service Registry, which knows the locations of all service instances.

© 2020 by Innovation In Software Corporation



MICROSERVICE DISCOVERY (CLIENT-SIDE)

© 2020 by Innovation In Software Corporation