In28 Minutes

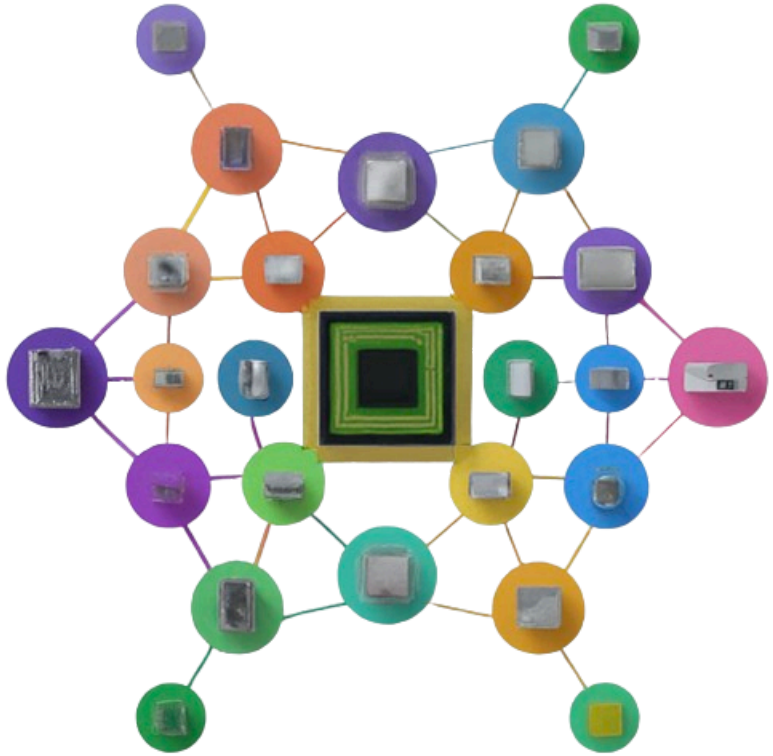
Master Microservices with Spring Boot and Spring Cloud

Getting Started

- **Microservices**: Most popular architecture
- **Top choices** for building microservices:
  - Spring Boot
  - Spring Cloud
  - Docker
  - Kubernetes

First Steps Are Challenging

- **Lots of Concepts**: REST API, Service Discovery, API Gateway, Naming Server, Circuit Breakers, Load Balancing, Security...
- **Diverse Deployment Options**: Containers, Orchestration, Cloud-native environments
- **Needs knowledge of tools and platforms**: Spring Boot, Spring Cloud, Docker, Kubernetes, Cloud, and more



Simple Path - Learn REST & Microservices & ...

In28 Minutes

- **Simple**: We've created a simple path focusing on the fundamentals
- **Hands-on**: You will build multiple microservice projects with all features
- **Beginner Friendly**: This course is *designed* for absolute beginners to Microservices
- **Our Goal** : Help you start your journey with Microservices and master it!

How do you put your best foot forward?

In28 Minutes

- **Tricky Path**: Learning Microservices can be tricky:
  - Lots of new terminology, tools and frameworks
- **We Forget Things**: With time, we forget things
- **Improve Your Chances**: How do you remember things for long time?
  - **Active learning** - think & make notes
  - **Review the presentation** once in a while

Our Approach

- **1: Understand The Fundamentals**: Beautiful Video Presentations
- **2: Hands-on Learning**: Videos where we build amazing projects
- **3: Regular Review**: Quizzes
- **My Recommendations**:
  - **Take your time**: Replay videos as needed
  - **Have Fun**: Enjoy the learning experience

Course Overview - 10,000 Feet

In28 Minutes

- **Agenda**: Step by Step Learning
  - REST API (Spring Boot)
  - Microservices (Spring Cloud)
  - Docker
  - Kubernetes
  - Regular Updates
    - Latest version of Java and Spring Boot
    - Latest version of IDE
    - Evolution

Thanks for the Amazing Feedback

In28 Minutes

- **250K+ Learners**: Amazing Reviews
  - "**Great** learning experience"
  - "Clear and **Simple**"
  - "**Fun** to learn"
  - "It's great that Ranga is **constantly adding** to it"
  - "**Exceeded** my expectations in every way"
- **Recommendations**: Great Experience
  - Little bit of patience for the first 1 hour
  - Go Hands-on
  - Go Step By Step!

FASTEST ROADMAPS

in28minutes.com

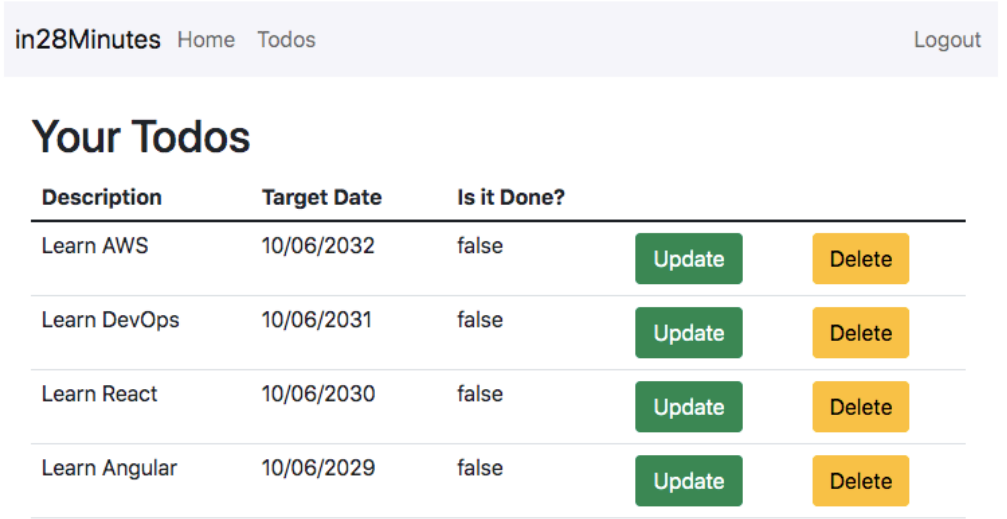
In28 Minutes

- Google Cloud Certifications
- Azure Certifications
- AWS Certifications
- DevOps
- Java Full Stack
- Java Microservices

Getting Started with Web Services

Is This a Web Service? (Try 1)

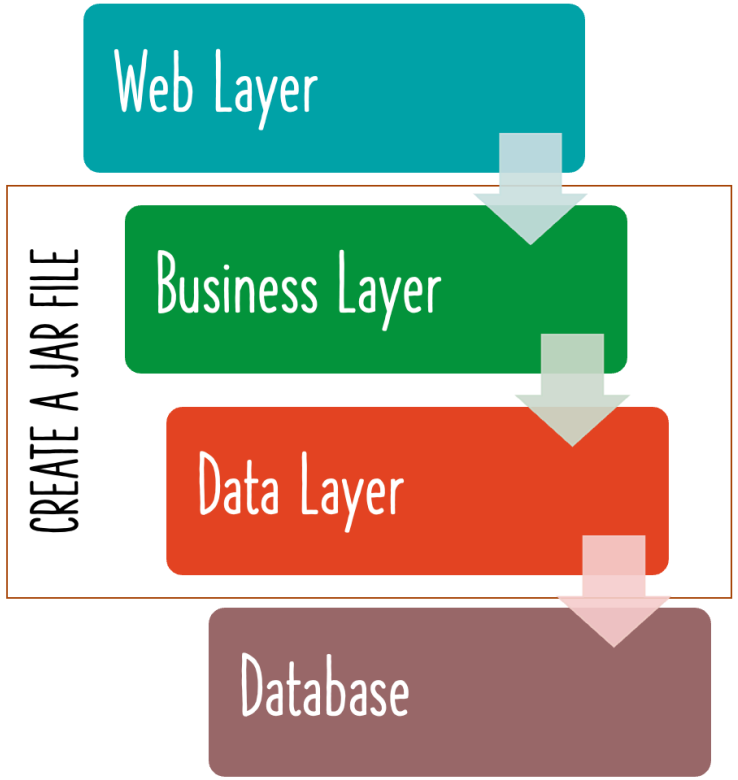
- **Definition**: Service delivered over the web?
- **Web Application**: Is the Todo Web Application a Web Service?
  - **Delivers HTML output**: Not consumable by other applications
- **NOT a Web Service**: Todo Web App is NOT a Web Service!
  - **Definition** is NOT just a "Service delivered over the web"



Is This a Web Service? (Try 2)

In28 Minutes

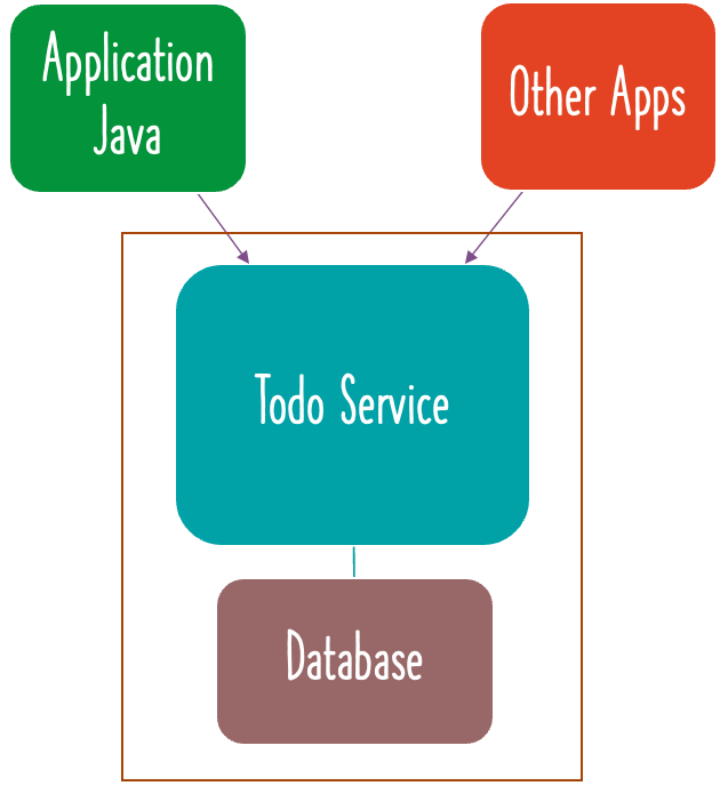
- **Question**: What if I create and share a JAR?
  - **Needs Other Dependencies**: Database, Queues, ..
  - **Communication of Changes**: Needs a process to handle future updates to code
  - **Not Platform independent**: What if the consumer is using .Net or Python or JS?
- **NOT a Web Service**: Sharing a JAR is NOT a web service approach!



Is This a Web Service? (Try 3)

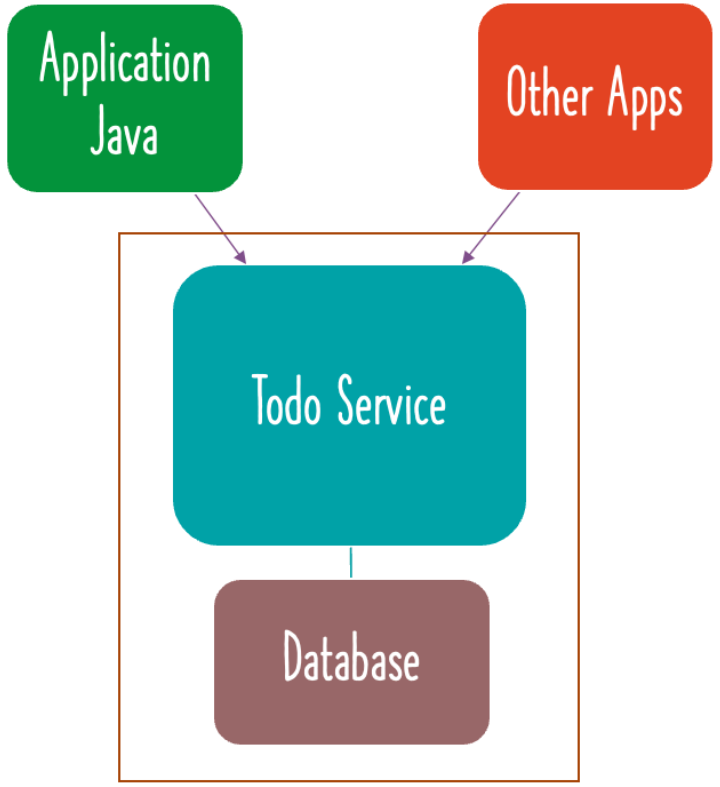
In28 Minutes

- **Question**: What if the Todo application is provide an output in a format that is consumable by other application?
  - **Good Question**: That's where we get into the concept of a web service!



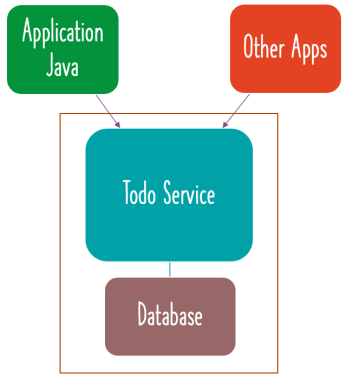
What is a Web Service?

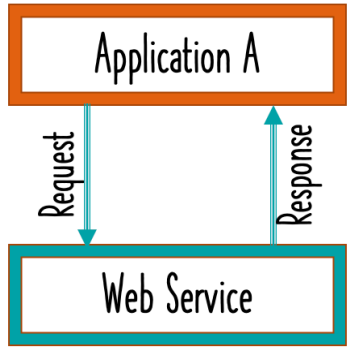
- **W3C definition**: Software system designed to support interoperable machine-to-machine interaction over a network
- **3 Keys**: All three are important
  - Designed for machine-to-machine (or application-to-application) interaction
  - Should be interoperable (Not platform dependent)
  - Should allow communication over a network



Web Services: How Question 1

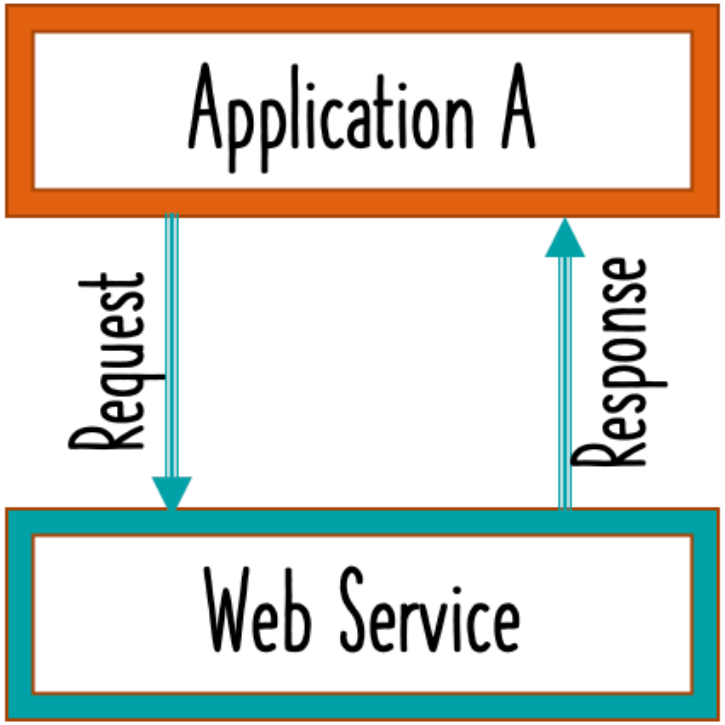
- **Question 1**: How does data exchange with a web service happen?
  - **How**:
    - **1**: Application/Service sends an input to a web service
    - **2**: Web service responds with an output
  - **Request**: Input to a web service
  - **Response**: Output from a web service





Web Services: How Question 2

- **Question 2**: How can we make web services platform independent?
  - How can we make applications built in different languages and platforms talk to each other?
  - **Key**: Communication should be platform independent
    - Request should be platform independent
    - Response should be platform independent



Understanding Web Services: Communication Format

In 28 Minutes

```
<getTodoDetailsRequest>
    <id>Ranga</id>
</getTodoDetailsRequest>
```

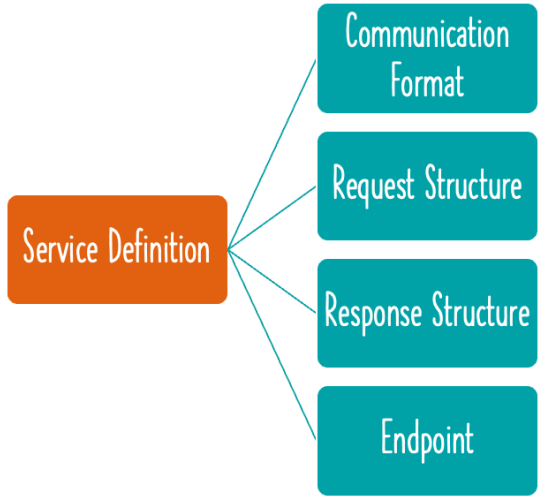
- **XML**:eXtensible Markup Language
- **JSON**: JavaScript Object Notation

```
[
  {
      "id": 1,
      "name": "Ranga",
      "description": "Learn Microservices"
  },
  {
      "id": 2,
      "name": "Ranga",
      "description": "Learn Google Cloud"
  }
]
```

- **Key**: XML and JSON are platform independent

Web Services: How Question 3

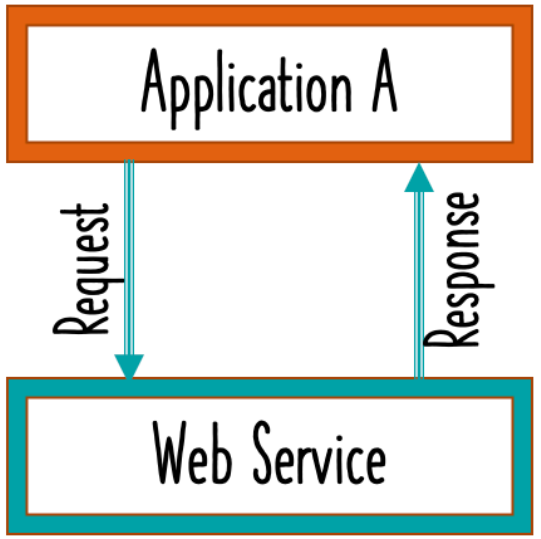
- **Question 3**: How does the Application know the format of Request and Response of a Web Service?
- **Answer**: Service Definition
  - **Request/Response Format**: XML or JSON or ..
  - **Request Structure**: What is the structure of the request?
  - **Response Structure**: What is the structure of the response?
  - **Endpoint**: How to call the service? What URL to use?



Web Services: Key Terminology - 1

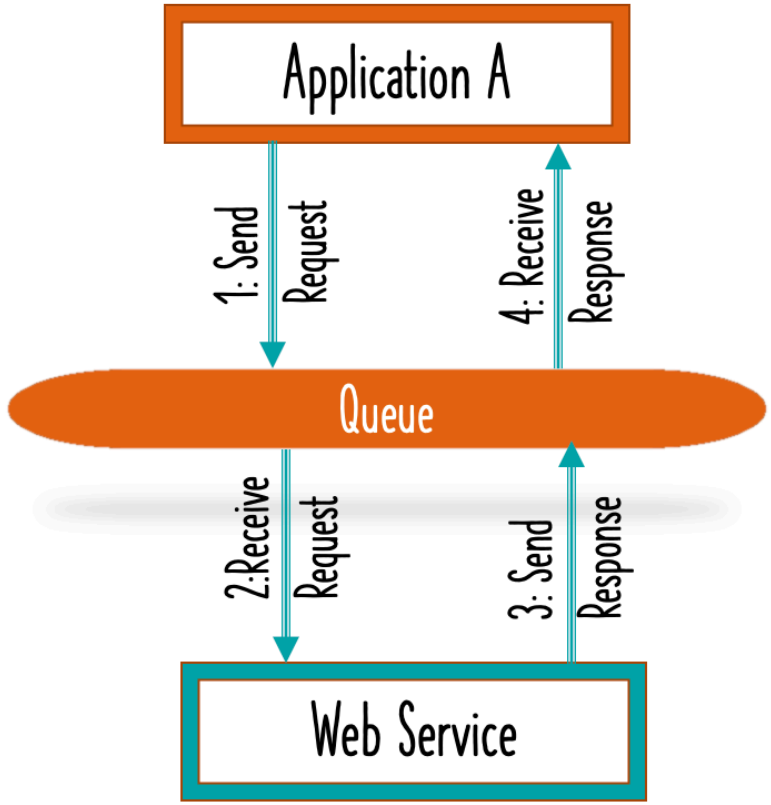
In28 Minutes

- **Request**: Input to a Web Service
- **Response**: Output from a Web Service
- **Message Exchange Format**: What format is used for communication?
  - XML or JSON or ..
- **Service Provider or Server**: Who is hosting/providing the Web Service?
- **Service Consumer or Client**: Who is using/consuming the webservice?



Web Services: Key Terminology - 2

- **Transport**: What is the communication channel?
  - Is it over the internet/web (HTTP)? Is it over the queue?
- **Service Definition**: A contract between Service Provider and Service Consumer
  - What is the format of the request/response?
  - What is the structure of the request/response?
  - How can consumer call the service?
  - What is the transport?



Web Services: SOAP (Simple Object Access Protocol)

In28 Minutes

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCourseDetailsRequest xmlns:ns2="http://in28minutes.com/courses">
            <ns2:course>
                <ns2:id>Course1</ns2:id>
            </ns2:course>
        </ns2:getCourseDetailsRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- A protocol with strict standards
- **Constraint 1**: Uses SOAP-XML for message format
- **Constraint 2**: Uses WSDL for service definition
- SOAP does NOT care about Transport: HTTP (internet) or Queue (MQ)

SOAP Constraint 1: SOAP XML Request & Response

In28 Minutes

```
<!-- SOAP Request Example -->
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCourseDetailsRequest xmlns:ns2="http://in28minutes.com/courses">
            <ns2:course>
                <ns2:id>Course1</ns2:id>
            </ns2:course>
        </ns2:getCourseDetailsRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Each Request & Response contain:
  - **SOAP Envelope**: The wrapper
  - **SOAP Header**(Optional): The header
    - Meta information: Authentication/Authorization/Signatures
  - **SOAP Body**: The content of request/response

SOAP Response Example

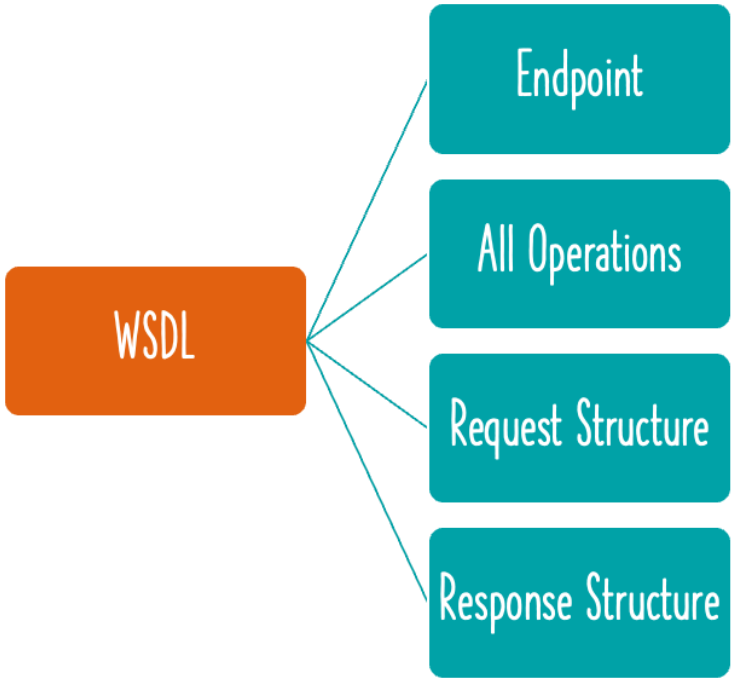
In28 Minutes

```
<!-- SOAP Response Example -->
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCourseDetailsResponse xmlns:ns2="http://in28minutes.com/courses">
            <ns2:course>
                <ns2:id>Course1</ns2:id>
                <ns2:name>AWS</ns2:name>
                <ns2:description>AWS in 10 Steps</ns2:description>
            </ns2:course>
        </ns2:getCourseDetailsResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- SOAP XML Response also contains:
  - **SOAP Envelope**: The wrapper
  - **SOAP Header**(Optional): The header
  - **SOAP Body**: The content of response

SOAP Constraint 2: Define a WSDL

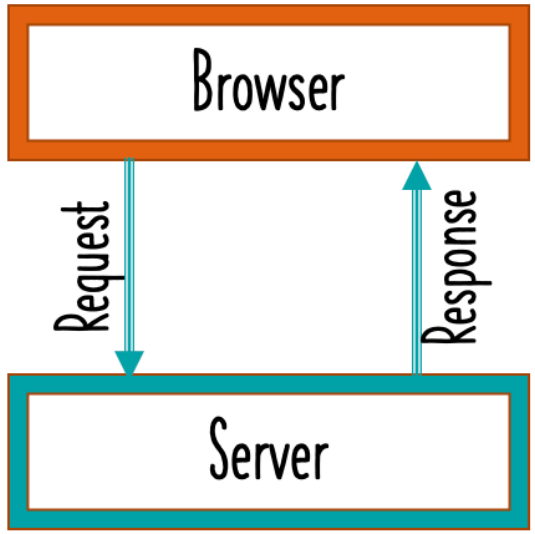
- A **WSDL (WebService Definition Language)** file defines:
  - **All Operations**: What operations are supported?
    - deleteCourse, updateCourse, createCourse, ...
  - **Endpoint**: How to call the service? What URL to use?
  - **Request/Response Format**: Always SOAP XML
  - **Request Structure**: What is the structure of the request?
  - **Response Structure**: What is the structure of the response?



Getting Started with REST & HTTP

In 28 Minutes

- **REST (REpresentational State Transfer)**: Coined by Roy Fielding
- **HTTP (HyperText Transfer Protocol)**: Foundation of the World Wide Web (or Internet, as we call it)
  - Browser sends a (HTTP) Request
  - Browser receives a (HTTP) Response



Exploring HTTP (HyperText Transfer Protocol)

In28 Minutes

- Request and Response Formats: Defined by HTTP
- **Request Format**: Request contains
  - **Request Method**: GET/POST/...
  - **Request Path**: /todos/1
  - **Request Headers** (Optional)
  - **Request Body** (Depending on Request Method)
- **Response Format**: Response contains
  - **Response Status**: 200/404/...
  - **Response Headers** (Optional)
  - **Response Body** (Depending on Request Method)

Crux of REST: Make Best of HTTP

- **Roy Fielding**: Why do we need to reinvent the wheel?
  - What if we can build REST on top of HTTP?
  - What if we can make the BEST use of HTTP to build our REST API?
- **Example**: Build a REST API for a Social Media Application
  - **One Key Resource**: User
  - **Key Details**:
    - User: id, name, birthDate

Request Methods for REST API

In 28 Minutes

- **GET** - Retrieve details of a resource
- **POST** - Create a new resource
- **PUT** - Update an existing resource
- **PATCH** - Update part of a resource
- **DELETE** - Delete a resource

localhost:8080/users


```
[
  {
    "id": 1,
    "name": "Adam",
    "birthDate": "2022-08-16"
  },
  {
    "id": 2,
    "name": "Eve",
    "birthDate": "2022-08-16"
  },
  {
    "id": 3,
    "name": "Jack",
    "birthDate": "2022-08-16"
  }
]
```

Social Media Application - Resources & Methods

- Retrieve all Users
  - **GET /users**
- Create a User
  - **POST /users**
- Retrieve a User with id 10
  - **GET /users/10**
- Delete a User with id 19
  - **DELETE /users/19**

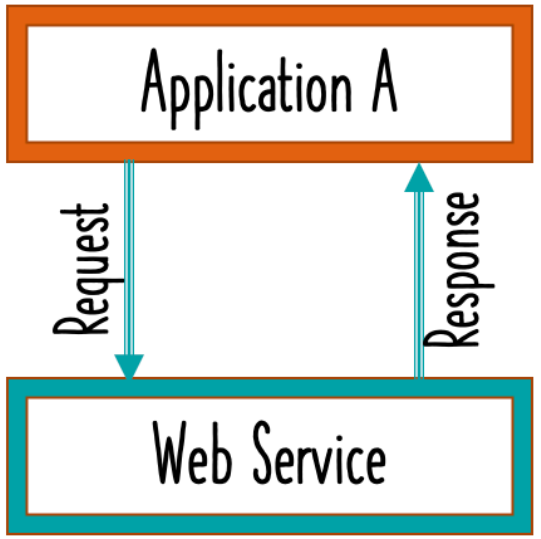
localhost:8080/users


```
[
  {
    "id": 1,
    "name": "Adam",
    "birthDate": "2022-08-16"
  },
  {
    "id": 2,
    "name": "Eve",
    "birthDate": "2022-08-16"
  },
  {
    "id": 3,
    "name": "Jack",
    "birthDate": "2022-08-16"
  }
]
```

REST vs SOAP - Quick Comparison - 1

In28 Minutes

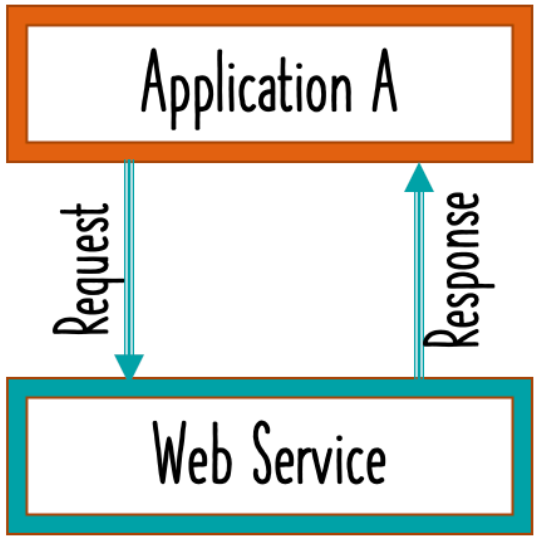
- 1: Restrictions vs Architectural Approach
  - **REST (REpresentational State Transfer)**: An Architectural approach
  - **SOAP (Simple Object Access Protocol)**: Puts constraints on the message exchange format of the web service
- 2: Data Exchange Format
  - **REST**: No restrictions (Popular JSON)
  - **SOAP**: XML only



REST vs SOAP - Quick Comparison - 2

In28 Minutes

- **3**: Service Definition
  - **REST**: No restrictions (Popular OpenAPI Specification)
  - **SOAP**: WSDL only
- **4**: Transport
  - **REST**: HTTP only
  - **SOAP**: No restrictions (HTTP/MQ/..)
- **5**: Ease of implementation
  - **REST**: Typically considered easier to implement
  - **SOAP**: Little bit more complex



Spring Boot
in 10(ish) Steps

Getting Started with Spring Boot

In 28 Minutes

- **WHY** Spring Boot?
  - You can build web apps & REST API WITHOUT Spring Boot
  - What is the need for Spring Boot?
- **WHAT** are the goals of Spring Boot?
- **HOW** does Spring Boot work?
- **COMPARE** Spring Boot vs Spring MVC vs Spring

Getting Started with Spring Boot - Approach

In28 Minutes

- **1:** Understand the world before Spring Boot (10000 Feet)
- **2:** Create a Spring Boot Project
- **3:** Build a simple REST API using Spring Boot
- **4:** Understand the MAGIC of Spring Boot
  - Spring Initializr
  - Starter Projects
  - Auto Configuration
  - Developer Tools
  - Actuator
  - ...

World Before Spring Boot!

In28 Minutes

- Setting up Spring Projects **before Spring Boot was NOT easy**!
- We needed to configure a **lot of things** before we have a **production-ready** application

World Before Spring Boot - 1 - Dependency Management

In28 Minutes

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>6.2.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- Manage frameworks and versions
  - **REST API** - Spring framework, Spring MVC framework, JSON binding framework, ..
  - **Unit Tests** - Spring Test, Mockito, JUnit, ...

World Before Spring Boot - 2 - web.xml

In28 Minutes

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/todo-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```

- **Example**: Configure **DispatcherServlet** for Spring MVC

World Before Spring Boot - 3 - Spring Configuration

In28 Minutes

```
<context:component-scan base-package="com.in28minutes" />

<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
        <value>/WEB-INF/views/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

- Define your **Spring Configuration**
  - Component Scan
  - View Resolver
  - ....

World Before Spring Boot - 4 - NFRs

In28 Minutes

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <path>/</path>
        <contextReloadable>true</contextReloadable>
    </configuration>
</plugin>

<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- Logging
- Error Handling
- Monitoring

World Before Spring Boot!

In28 Minutes

- Setting up Spring Projects **before Spring Boot was NOT easy**!
  - 1: Dependency Management (**pom.xml**)
  - 2: Define Web App Configuration (**web.xml**)
  - 3: Manage Spring Beans (**context.xml**)
  - 4: Implement Non Functional Requirements (NFRs)
- AND repeat this for every new project!
- Typically takes a **few days** to setup for each project (and countless hours to maintain)

Understanding Power of Spring Boot

In28 Minutes

```
// http://localhost:8080/courses
[
  {
    "id": 1,
    "name": "Learn AWS",
    "author": "in28minutes"
  }
]
```

- **1:** Create a Spring Boot Project
- **2:** Build a simple REST API using Spring Boot

What's the Most Important Goal of Spring Boot?

In 28 Minutes

- Help you build **PRODUCTION-READY** apps **QUICKLY**
  - Build **QUICKLY**
    - Spring Initializr
    - Spring Boot Starter Projects
    - Spring Boot Auto Configuration
    - Spring Boot DevTools
  - Be **PRODUCTION-READY**
    - Logging
    - Different Configuration for Different Environments
      - Profiles, ConfigurationProperties
    - Monitoring (Spring Boot Actuator)
    - ...

In28 Minutes

Spring Boot
BUILD QUICKLY

Exploring Spring Boot Starter Projects

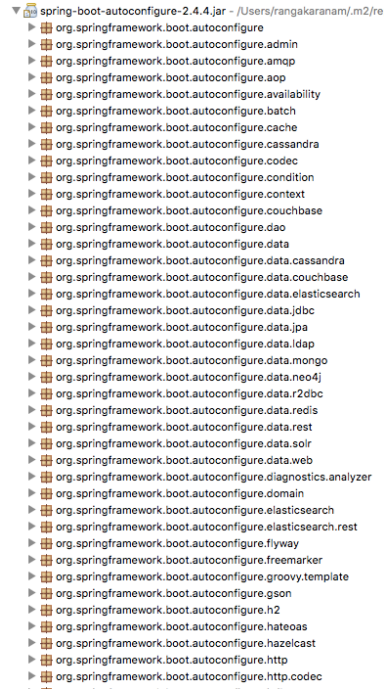
In28 Minutes

- I need a lot of frameworks to build application features:
  - **Build a REST API**: I need Spring, Spring MVC, Tomcat, JSON conversion...
  - **Write Unit Tests**: I need Spring Test, JUnit, Mockito, ...
- How can I group them and make it easy to build applications?
  - **Starters**: Convenient **dependency descriptors** for diff. features
- **Spring Boot** provides variety of starter projects:
  - **Web Application & REST API** - Spring Boot Starter Web (spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json)
  - **Unit Tests** - Spring Boot Starter Test
  - **Talk to database using JPA** - Spring Boot Starter Data JPA
  - **Talk to database using JDBC** - Spring Boot Starter JDBC
  - **Secure your web application or REST API** - Spring Boot Starter Security

Exploring Spring Boot Auto Configuration

In 28 Minutes

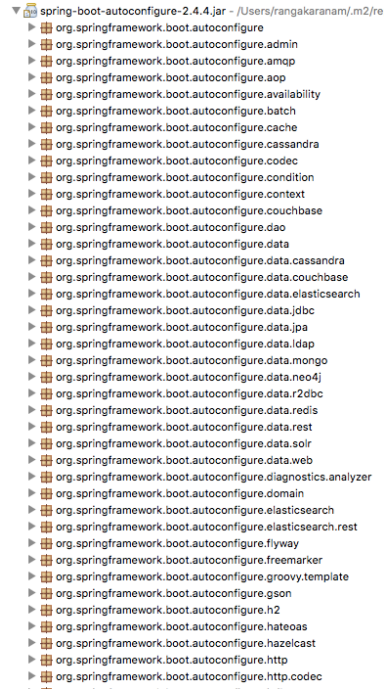
- I need **lot of configuration** to build Spring app:
  - Component Scan, DispatcherServlet, Data Sources, JSON Conversion, ...
- How can I simplify this?
  - **Auto Configuration**: **Automated configuration** for your app
    - **Decided** based on:
      - Which frameworks are in the Class Path?
      - What is the existing configuration (Annotations etc)?
- **Example**: Spring Boot Starter Web
  - Dispatcher Servlet (`DispatcherServletAutoConfiguration`)
  - Embedded Servlet Container - Tomcat is the default (`EmbeddedWebServerFactoryCustomizerAutoConfiguration`)
  - Default Error Pages (`ErrorMvcAutoConfiguration`)
  - Bean<->JSON (`JacksonHttpMessageConvertersConfiguration`)



Understanding the Glue - @SpringBootApplication

In 28 Minutes

- Questions:
  - Who is launching the Spring Context?
  - Who is triggering the component scan?
  - Who is enabling auto configuration?
- Answer: **@SpringBootApplication**
  - 1: **@SpringBootConfiguration**: Indicates that a class provides Spring Boot application @Configuration.
  - 2: **@EnableAutoConfiguration**: Enable auto-configuration of the Spring Application Context,
  - 3: **@ComponentScan**: Enable component scan (for current package, by default)



Build Faster with Spring Boot DevTools

In28 Minutes

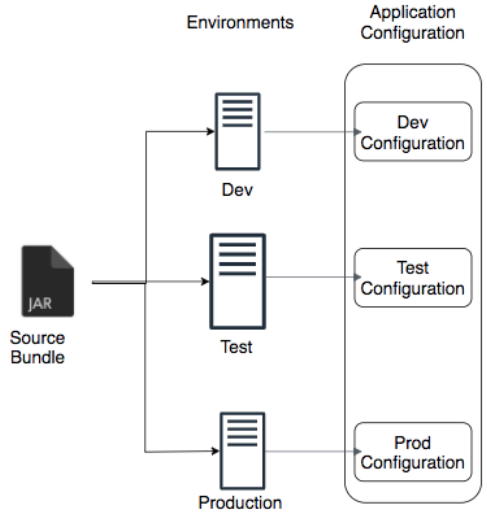
- Increase developer productivity
- Why do you need to restart the server **manually** for every code change?
- **Remember**: For pom.xml dependency changes, you will need to restart server **manually**

Spring Boot
PRODUCTION-READY

Managing App. Configuration using Profiles

In28 Minutes

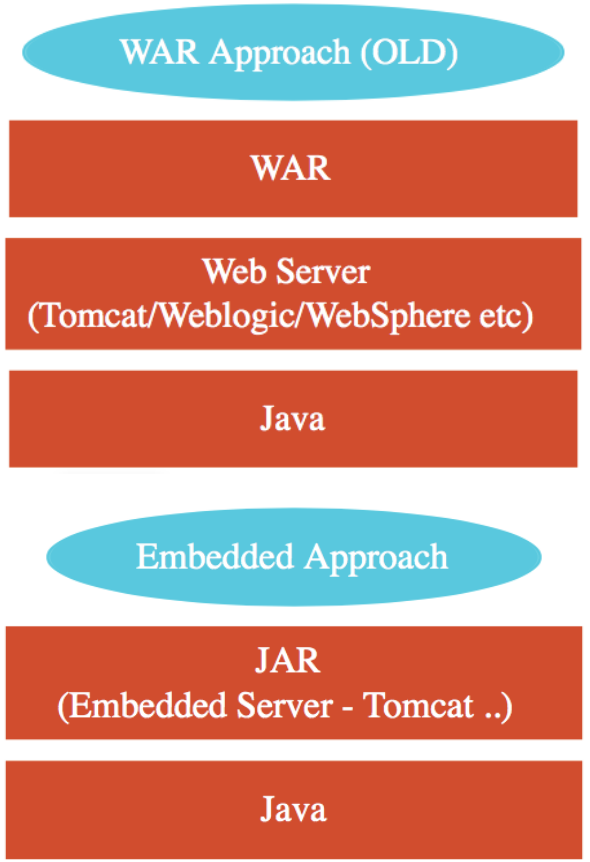
- Applications have different environments: **Dev, QA, Stage, Prod**, ...
- Different environments need **different configuration**:
  - Different Databases
  - Different Web Services
- How can you provide different configuration for different environments?
  - **Profiles**: Environment specific configuration
- How can you define externalized configuration for your application?
  - **ConfigurationProperites**: Define externalized configuration



Simplify Deployment with Spring Boot Embedded Servers

In 28 Minutes

- How do you deploy your application?
  - Step 1 : Install Java
  - Step 2 : Install Web/Application Server
    - Tomcat/WebSphere/WebLogic etc
  - Step 3 : Deploy the application WAR (Web ARchive)
    - This is the OLD **WAR** Approach
    - Complex to setup!
- **Embedded Server** - Simpler alternative
  - Step 1 : Install Java
  - Step 2 : Run **JAR** file
  - **Make JAR not WAR** (Credit: Josh Long!)
  - Embedded Server **Examples**:
    - spring-boot-starter-tomcat
    - spring-boot-starter-jetty
    - spring-boot-starter-undertow



Monitor Applications using Spring Boot Actuator

In 28 Minutes

- Monitor and manage your application in your production
- Provides a number of endpoints:
  - **beans** - Complete list of Spring beans in your app
  - **health** - Application health information
  - **metrics** - Application metrics
  - **mappings** - Details around Request Mappings

Understanding Spring Boot vs Spring MVC vs Spring

In28 Minutes

- **Spring Boot vs Spring MVC vs Spring**: What's in it?
  - **Spring Framework**: Dependency Injection
    - @Component, @Autowired, Component Scan etc..
    - Just Dependency Injection is NOT sufficient (You need other frameworks to build apps)
      - **Spring Modules and Spring Projects**: Extend Spring Eco System
        - Provide good integration with other frameworks (Hibernate/JPA, JUnit & Mockito for Unit Testing)
  - **Spring MVC** (Spring Module): Simplify building web apps and REST API
    - Building web applications with Struts was very complex
    - @Controller, @RestController, @RequestMapping("/courses")
  - **Spring Boot** (Spring Project): Build **PRODUCTION-READY** apps **QUICKLY**
    - **Starter Projects** - Make it easy to build variety of applications
    - **Auto configuration** - Eliminate configuration to setup Spring, Spring MVC and other frameworks!
    - Enable non functional requirements (NFRs):
      - **Actuator**: Enables Advanced Monitoring of applications
      - **Embedded Server**: No need for separate application servers!
      - Logging and Error Handling
      - Profiles and ConfigurationProperties

Spring Boot - Review

In 28 Minutes

- **Goal**: 10,000 Feet overview of Spring Boot
  - Help you understand the terminology!
    - Starter Projects
    - Auto Configuration
    - Actuator
    - DevTools
- **Advantages**: Get started quickly with production ready features!

In28 Minutes

Building REST API with Spring Boot

Building REST API with Spring Boot - Goals

In 28 Minutes

- **WHY** Spring Boot?
  - You can build REST API WITHOUT Spring Boot
  - What is the need for Spring Boot?
- **HOW** to build a great REST API?
  - Identifying Resources (/users, /users/{id}/posts)
  - Identifying Actions (GET, POST, PUT, DELETE, ...)
  - Defining Request and Response structures
  - Using appropriate Response Status (200, 404, 500, ..)
  - Understanding REST API Best Practices
    - Thinking from the perspective of your consumer
    - Validation, Internationalization - i18n, Exception Handling, HATEOAS, Versioning, Documentation, Content Negotiation and a lot more!

Building REST API with Spring Boot - Approach

In28 Minutes

- **1:** Build 3 Simple Hello World REST API
  - Understand the magic of Spring Boot
  - Understand fundamentals of building REST API with Spring Boot
    - @RestController, @RequestMapping, @PathVariable, JSON conversion
- **2:** Build a REST API for a Social Media Application
  - Design and Build a Great REST API
    - Choosing the right URI for resources (/users, /users/{id}, /users/{id}/posts)
    - Choosing the right request method for actions (GET, POST, PUT, DELETE, ..)
    - Designing Request and Response structures
    - Implementing Security, Validation and Exception Handling
  - Build Advanced REST API Features
    - Internationalization, HATEOAS, Versioning, Documentation, Content Negotiation, ...
- **3:** Connect your REST API to a Database
  - Fundamentals of JPA and Hibernate
  - Use H2 and MySQL as databases

What's Happening in the Background?

In28 Minutes

- Let's explore some **Spring Boot Magic**: Enable Debug Logging
  - **WARNING**: Log change frequently!
- **1:** How are our requests handled?
  - **DispatcherServlet** - Front Controller Pattern
    - `Mapping servlets: dispatcherServlet urls=[/]`
    - **Auto Configuration** (`DispatcherServletAutoConfiguration`)
- **2:** How does **HelloWorldBean** object get converted to JSON?
  - @ResponseBody + JacksonHttpMessageConverters
    - **Auto Configuration** (`JacksonHttpMessageConvertersConfiguration`)
- **3:** Who is configuring error mapping?
  - **Auto Configuration** (`ErrorMvcAutoConfiguration`)
- **4:** How are all jars available(Spring, Spring MVC, Jackson, Tomcat)?
  - **Starter Projects** - Spring Boot Starter Web (spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json)

Social Media Application REST API

In 28 Minutes

- Build a REST API for a Social Media Application
- **Key Resources:**
  - Users
  - Posts
- **Key Details:**
  - User: id, name, birthDate
  - Post: id, description

Request Methods for REST API

In 28 Minutes

- **GET** - Retrieve details of a resource
- **POST** - Create a new resource
- **PUT** - Update an existing resource
- **PATCH** - Update part of a resource
- **DELETE** - Delete a resource

localhost:8080/users


```
[
  {
    "id": 1,
    "name": "Adam",
    "birthDate": "2022-08-16"
  },
  {
    "id": 2,
    "name": "Eve",
    "birthDate": "2022-08-16"
  },
  {
    "id": 3,
    "name": "Jack",
    "birthDate": "2022-08-16"
  }
]
```

Social Media Application - Resources & Methods

In28 Minutes

- **Users REST API**
  - Retrieve all Users
    - **GET /users**
  - Create a User
    - **POST /users**
  - Retrieve one User
    - **GET /users/{id} -> /users/1**
  - Delete a User
    - **DELETE /users/{id} -> /users/1**
  - **Posts REST API**
    - Retrieve all posts for a User
      - **GET /users/{id}/posts**
    - Create a post for a User
      - **POST /users/{id}/posts**
    - Retrieve details of a post
      - **GET /users/{id}/posts/{post_id}**

Response Status for REST API

In 28 Minutes

- Return the **correct response status**
  - Resource is not found => 404
  - Server exception => 500
  - Validation error => 400
- **Important Response Statuses**
  - **200** — Success
  - **201** — Created
  - **204** — No Content
  - **401** — Unauthorized (when authorization fails)
  - **400** — Bad Request (such as validation error)
  - **404** — Resource Not Found
  - **500** — Server Error

localhost:8080/users

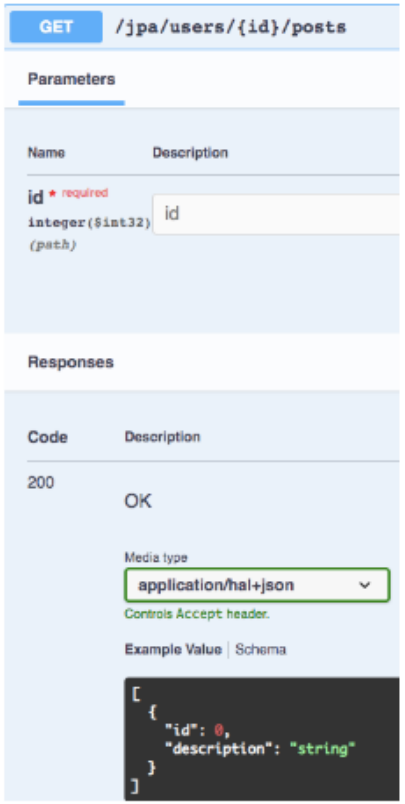

```
[
  {
    "id": 1,
    "name": "Adam",
    "birthDate": "2022-08-16"
  },
  {
    "id": 2,
    "name": "Eve",
    "birthDate": "2022-08-16"
  },
  {
    "id": 3,
    "name": "Jack",
    "birthDate": "2022-08-16"
  }
]
```

Advanced REST API Features

- Documentation
- Content Negotiation
- Internationalization - i18n
- Versioning
- HATEOAS
- Static Filtering
- Dynamic Filtering
- Monitoring
- ....

REST API Documentation

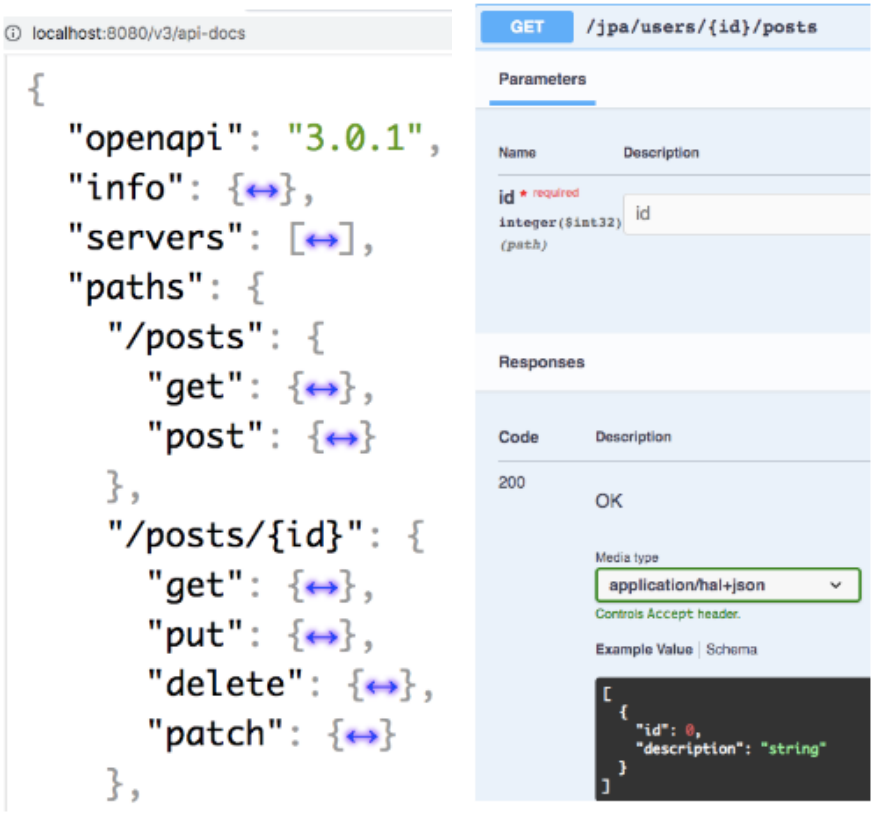
- Your REST API consumers need to understand your REST API:
  - Resources
  - Actions
  - Request/Response Structure (Constraints/Validations)
- **Challenges:**
  - Accuracy: How do you ensure that your documentation is upto date and correct?
  - Consistency: You might have 100s of REST API in an enterprise. How do you ensure consistency?
- **Options:**
  - **1**: Manually Maintain Documentation
    - Additional effort to keep it in sync with code
  - **2**: Generate from code



REST API Documentation - Swagger and Open API

In 28 Minutes

- **Quick overview:**
  - **2011:** Swagger Specification and Swagger Tools were introduced
  - **2016:** Open API Specification created based on Swagger Spec.
    - Swagger Tools (ex:Swagger UI) continue to exist
  - **OpenAPI Specification**: Standard, language-agnostic interface
    - Discover and understand REST API
    - Earlier called Swagger Specification
  - **Swagger UI**: Visualize and interact with your REST API
    - Can be generated from your OpenAPI Specification

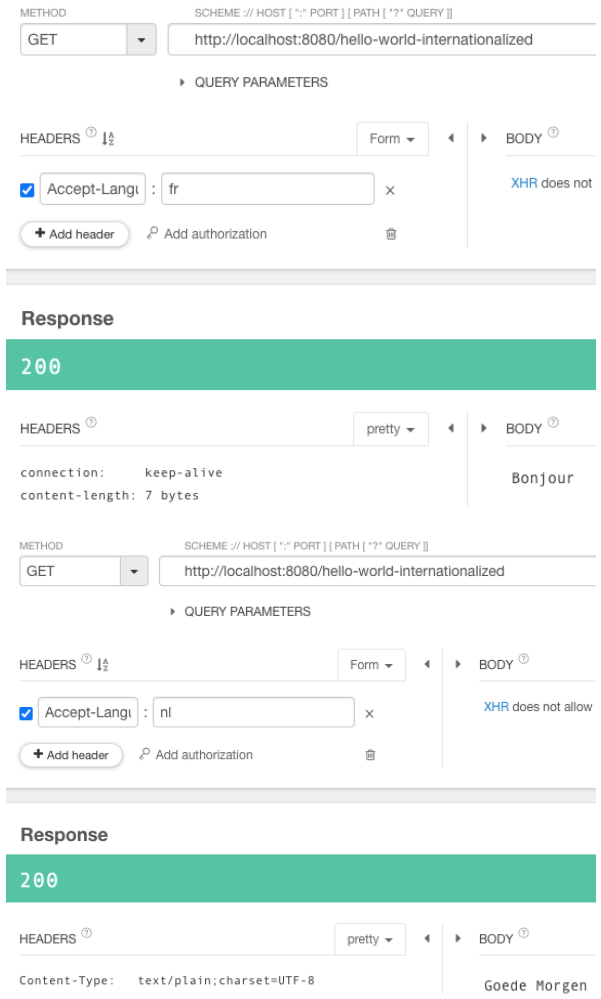


Content Negotiation

- **Same Resource** - Same URI
  - HOWEVER **Different Representations** are possible
    - Example: Different Content Type - XML or JSON or ..
    - Example: Different Language - English or Dutch or ..
- How can a consumer tell the REST API provider what they want?
  - **Content Negotiation**
- Example: Accept header (MIME types - application/xml, application/json, ..)
- Example: Accept-Language header (en, nl, fr, ..)

Internationalization - i18n

- Your REST API might have consumers from around the world
- How do you customize it to users around the world?
  - **Internationalization - i18n**
- Typically **HTTP Request Header - `Accept-Language`** is used
  - `Accept-Language` - indicates natural language and locale that the consumer prefers
  - Example: `en` - English (Good Morning)
  - Example: `nl` - Dutch (Goedemorgen)
  - Example: `fr` - French (Bonjour)



Versioning REST API

In 28 Minutes

- You have built an amazing REST API
  - You have 100s of consumers
  - You need to implement a breaking change
    - Example: Split name into firstName and lastName
- **SOLUTION**: Versioning REST API
  - **Variety of options**
    - URL
    - Request Parameter
    - Header
    - Media Type
  - **No Clear Winner!**

localhost:8080/v1/person

```
{
  "name": "Bob Charlie"
}
```

localhost:8080/v2/person

```
{
  "name": {
    "firstName": "Bob",
    "lastName": "Charlie"
  }
}
```

Versioning REST API - Options

- **URI Versioning** - Twitter
  - ***http://localhost:8080/v1/person***
  - ***http://localhost:8080/v2/person***
- **Request Parameter versioning** - Amazon
  - ***http://localhost:8080/person?version=1***
  - ***http://localhost:8080/person?version=2***
- **(Custom) headers versioning** - Microsoft
  - SAME-URL headers=[X-API-VERSION=1]
  - SAME-URL headers=[X-API-VERSION=2]
- **Media type versioning** (a.k.a "content negotiation" or "accept header") - GitHub
  - SAME-URL produces=application/vnd.company.app-v1+json
  - SAME-URL produces=application/vnd.company.app-v2+json

Versioning REST API - Factors

In28 Minutes

- **Factors to consider**
  - URI Pollution
  - Misuse of HTTP Headers
  - Caching
  - Can we execute the request on the browser?
  - API Documentation
  - **Summary**: No Perfect Solution
- **My Recommendations**
  - Think about versioning even before you need it!
  - One Enterprise - One Versioning Approach

URI Versioning - Twitter
- *http://localhost:8080/v1/person*
- *http://localhost:8080/v2/person*

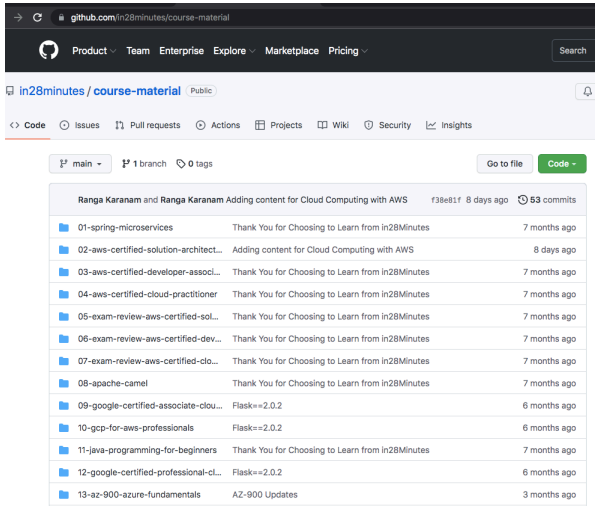
Request Parameter versioning - Amazon
- *http://localhost:8080/person?version=1*
- *http://localhost:8080/person?version=2*

(Custom) headers versioning - Microsoft
- SAME-URL headers=[X-API-VERSION=1]
- SAME-URL headers=[X-API-VERSION=2]

Media type versioning - GitHub
- SAME-URL produces=application/vnd.company.app-v1+json
- SAME-URL produces=application/vnd.company.app-v2+json

HATEOAS

- **Hypermedia as the Engine of Application State (HATEOAS)**
- Websites allow you to:
  - See **Data** AND Perform **Actions** (using links)
- How about enhancing your REST API to tell consumers how to perform subsequent actions?
  - **HATEOAS**
- **Implementation Options**:
  - **1:** Custom Format and Implementation
    - Difficult to maintain
  - **2:** Use Standard Implementation
    - **HAL (JSON Hypertext Application Language)**: Simple format that gives a consistent and easy way to hyperlink between resources in your API
    - **Spring HATEOAS**: Generate HAL responses with hyperlinks to resources




```
{
  "name": "Adam",
  "birthDate": "2022-08-16",
  "_links": {
    "all-users": {
      "href": "http://localhost:8080/users"
    }
  }
}
```

Customizing REST API Responses - Filtering and more..

In 28 Minutes

- **Serialization**: Convert object to stream (example: JSON)
  - Most popular JSON Serialization in Java: Jackson
- How about customizing the REST API response returned by Jackson framework?
- **1:** Customize field names in response
  - @JSONProperty
- **2:** Return only selected fields
  - **Filtering**
  - Example: Filter out Passwords
  - **Two types**:
    - **Static Filtering**: Same filtering for a bean across different REST API
      - @JsonIgnoreProperties, @JsonIgnore
    - **Dynamic Filtering**: Customize filtering for a bean for specific REST API
      - @JsonFilter with FilterProvider

localhost:8080/filtering-list


```
[
  {
    "field2": "value2",
    "field3": "value3"
  },
  {
    "field2": "value5",
    "field3": "value6"
  }
]
```


localhost:8080/filtering


```
{
  "field1": "value1",
  "field3": "value3"
}
```

Get Production-ready with Spring Boot Actuator

In 28 Minutes

- **Spring Boot Actuator**: Provides Spring Boot's production-ready features
  - Monitor and manage your application in your production
- **Spring Boot Starter Actuator**: Starter to add Spring Boot Actuator to your application
  - **spring-boot-starter-actuator**
- Provides a number of endpoints:
  - **beans** - Complete list of Spring beans in your app
  - **health** - Application health information
  - **metrics** - Application metrics
  - **mappings** - Details around Request Mappings
  - and a lot more .....

Explore REST API using HAL Explorer

In28 Minutes

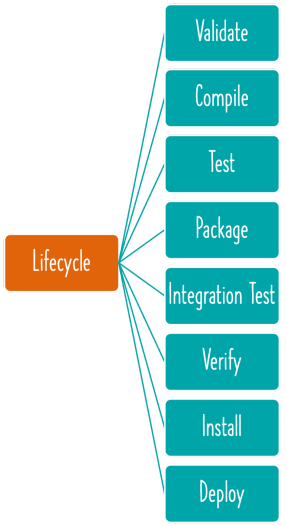
- 1: **HAL (JSON Hypertext Application Language)**
  - Simple format that gives a consistent and easy way to hyperlink between resources in your API
- 2: **HAL Explorer**
  - An API explorer for RESTful Hypermedia APIs using HAL
  - Enable your non-technical teams to play with APIs
- 3: **Spring Boot HAL Explorer**
  - Auto-configures HAL Explorer for Spring Boot Projects
  - spring-data-rest-hal-explorer

Maven

What is Maven?

In28 Minutes

- **Things you do** when writing code each day:
  - Create new projects
  - Manages **dependencies** and their versions
    - Spring, Spring MVC, Hibernate,...
    - Add/modify dependencies
  - **Build** a JAR file
  - Run your application locally in Tomcat or Jetty or ..
  - Run **unit tests**
  - Deploy to a test environment
  - and a lot more..
- Maven helps you do all these and more...



spring

Maven™

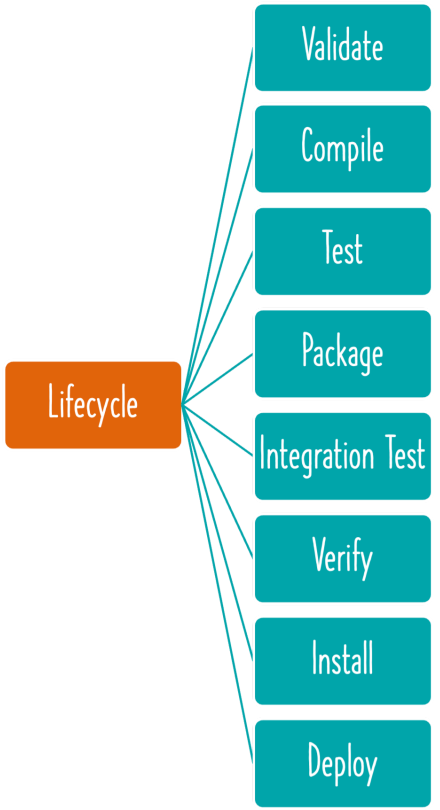
Exploring Project Object Model - pom.xml

In 28 Minutes

- Let's explore Project Object Model - pom.xml
  - **1: Maven dependencies**: Frameworks & libraries used in a project
    - Ex: `spring-boot-starter-web` and `spring-boot-starter-test`
    - Why are there so many dependencies in the classpath?
      - Answer: Transitive Dependencies
      - (REMEMBER) Spring dependencies are DIFFERENT
  - **2: Parent Pom**: spring-boot-starter-parent
    - Dependency Management: `spring-boot-dependencies`
    - Properties: `java.version`, plugins and configurations
  - **3: Name of our project**: groupId + artifactId
    - **1:groupId**: Similar to package name
    - **2:artifactId**: Similar to class name
    - **Why is it important?**
      - Think about this: How can other projects use our new project?
- **Activity**: help:effective-pom, dependency:tree & Eclipse UI
  - Let's add a new dependency: spring-boot-starter-web

Exploring Maven Build Life Cycle

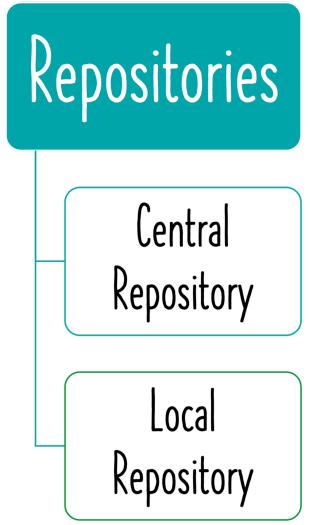
- When we run a maven command, maven build life cycle is used
- Build LifeCycle is a sequence of steps
  - Validate
  - Compile
  - Test
  - Package
  - Integration Test
  - Verify
  - Install
  - Deploy



How does Maven Work?

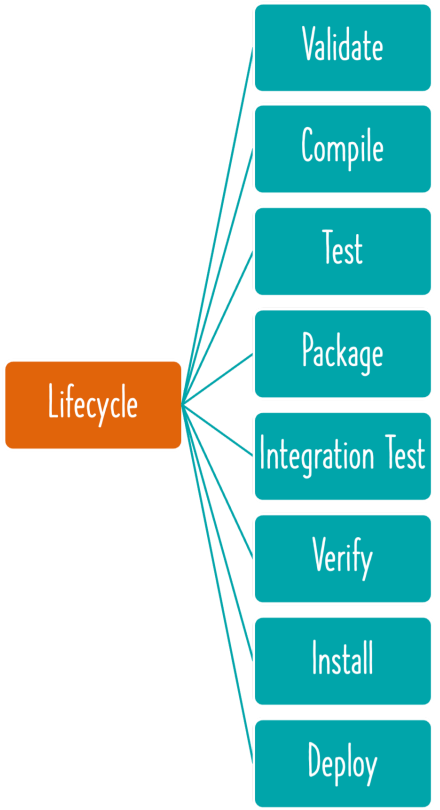
In28 Minutes

- Maven follows **Convention over Configuration**
  - Pre defined folder structure
  - Almost all Java projects follow **Maven structure** (Consistency)
- **Maven central repository** contains jars (and others) indexed by artifact id and group id
  - Stores all the versions of dependencies
  - repositories > repository
  - pluginRepositories > pluginRepository
- When a dependency is added to pom.xml, Maven tries to download the dependency
  - Downloaded dependencies are stored inside your maven local repository
  - **Local Repository** : a temp folder on your machine where maven stores the jar and dependency files that are downloaded from Maven Repository.



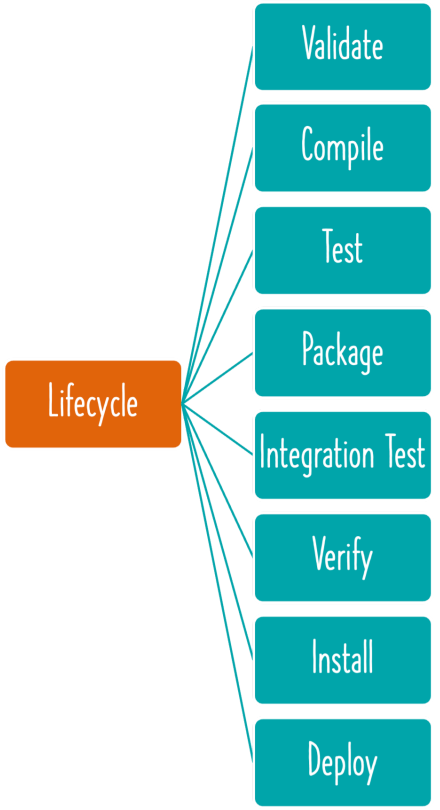
Important Maven Commands

- mvn --version
- mvn compile: Compile source files
- mvn test-compile: Compile test files
  - OBSERVCE CAREFULLY: This will also compile source files
- mvn clean: Delete target directory
- mvn test: Run unit tests
- mvn package: Create a jar
- mvn help:effective-pom
- mvn dependency:tree



Spring Boot Maven Plugin

- **Spring Boot Maven Plugin**: Provides Spring Boot support in Apache Maven
  - Example: Create executable jar package
  - Example: Run Spring Boot application
  - Example: Create a Container Image
- **Commands**:
  - mvn spring-boot:**repackage** (create jar or war)
    - Run package using `java –jar`
  - mvn spring-boot:**run** (Run application)
  - mvn spring-boot:**start** (Non-blocking - Ex:integration tests)
  - mvn spring-boot:**stop** (Stop application)
  - mvn spring-boot:**build-image** (Build a container image)



How are Spring Releases Versioned?

- **Version scheme** - MAJOR.MINOR.PATCH[-MODIFIER]
  - **MAJOR**: Significant amount of work to upgrade (10.0.0 to 11.0.0)
  - **MINOR**: Little to no work to upgrade (10.1.0 to 10.2.0)
  - **PATCH**: No work to upgrade (10.5.4 to 10.5.5)
  - **MODIFIER**: Optional modifier
    - **Milestones** - M1, M2, .. (10.3.0-M1,10.3.0-M2)
    - **Release candidates** - RC1, RC2, .. (10.3.0-RC1, 10.3.0-RC2)
    - **Snapshots** - SNAPSHOT
    - **Release** - Modifier will be ABSENT (10.0.0, 10.1.0)
- **Example versions in order:**
  - 10.0.0-SNAPSHOT, 10.0.0-M1, 10.0.0-M2, 10.0.0-RC1, 10.0.0-RC2, 10.0.0, ...
- **MY RECOMMENDATIONS:**
  - Avoid SNAPSHOTs
  - Use ONLY Released versions in PRODUCTION

Gradle

Gradle

In 28 Minutes

- **Goal**: Build, automate and deliver better software, faster
  - **Build Anything**: Cross-Platform Tool
    - Java, C/C++, JavaScript, Python, ...
  - **Automate Everything**: Completely Programmable
    - Complete flexibility
    - Uses a DSL
      - Supports Groovy and Kotlin
  - **Deliver Faster**: Blazing-fast builds
    - Compile avoidance to advanced caching
    - Can speed up Maven builds by up to 90%
      - **Incrementality** — Gradle runs only what is necessary
        - Example: Compiles only changed files
      - **Build Cache** — Reuses the build outputs of other Gradle builds with the same inputs
- Same project layout as Maven
- IDE support still evolving

Gradle Plugins

- Top 3 Java Plugins for Gradle:
  - **1: Java Plugin**: Java compilation + testing + bundling capabilities
    - **Default Layout**
      - src/main/java: Production Java source
      - src/main/resources: Production resources, such as XML and properties files
      - src/test/java: Test Java source
      - src/test/resources: Test resources
    - **Key Task**: build
  - **2: Dependency Management**: Maven-like dependency management
    - `group:'org.springframework', name:'spring-core', version:'10.0.3.RELEASE'` OR
    - Shortcut: `org.springframework:spring-core:10.0.3.RELEASE`
  - **3: Spring Boot Gradle Plugin**: Spring Boot support in Gradle
    - Package executable Spring Boot jar, Container Image (bootJar, bootBuildImage)
    - Use dependency management enabled by spring-boot-dependencies
      - No need to specify dependency version
        - Ex: `implementation('org.springframework.boot:spring-boot-starter')`

Maven vs Gradle - Which one to Use?

In28 Minutes

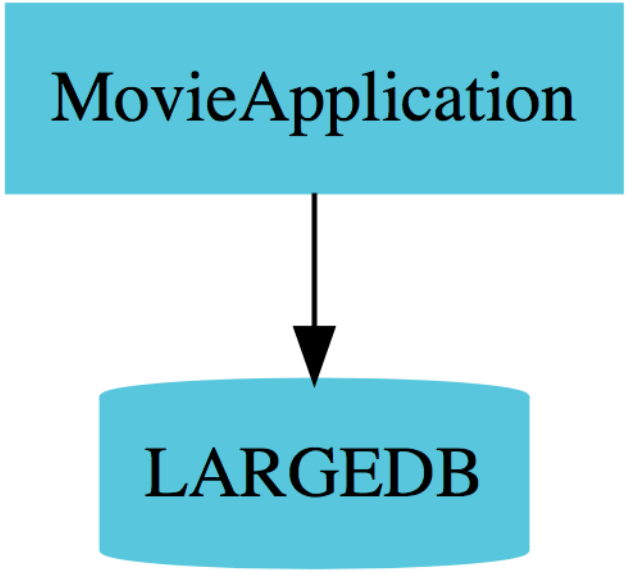
- Let's start with a few popular examples:
  - **Spring Framework** - Using Gradle since 2012 (Spring Framework v3.2.0)
  - **Spring Boot** - Using Gradle since 2020 (Spring Boot v2.3.0)
  - **Spring Cloud** - Continues to use Maven even today
    - Last update: Spring Cloud has no plans to switch
- **Top Maven Advantages**: Familiar, Simple and Restrictive
- **Top Gradle Advantages**: Faster build times and less verbose
- **What Do I Recommend**: I'm sitting on the fence for now
  - Choose whatever tool best meets your projects needs
  - If your builds are taking really long, go with Gradle
  - If your builds are simple, stick with Maven

Microservices

What is a Monolith?

In28 Minutes

- **Monolith**: A large application
  - Was a popular architecture for a few decades!
- **Challenges**: A lot of challenges
  - **Deployment complexity** - Minor updates need complete redeployment
  - **Tightly coupled components** - Changes in one part of can affect others => longer release cycles
  - **Scalability limitations** - Scaling can be resource-intensive. Entire app to be scaled even if only one part of it needs more capacity.
  - **Technology lock-in** - Entire app built on a single technology stack. Adopting new technologies is challenging.



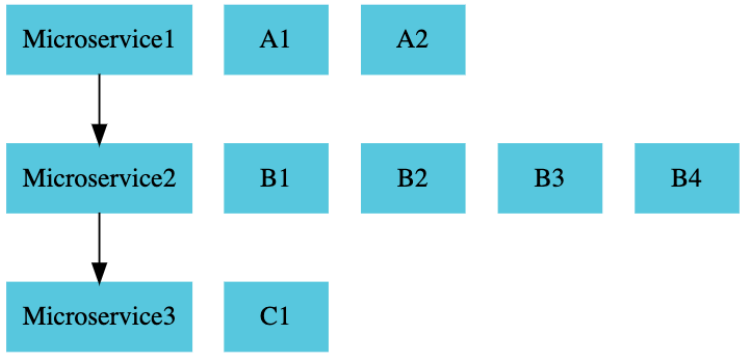
Getting Started with Microservices

- **Varied Definitions**: Let's review a few definitions of Microservices
  - **Simple**: "Small autonomous services that work together" - **Sam Newman**
  - **Complex**: "In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies." - **James Lewis and Martin Fowler**

Microservices - Keeping it simple!

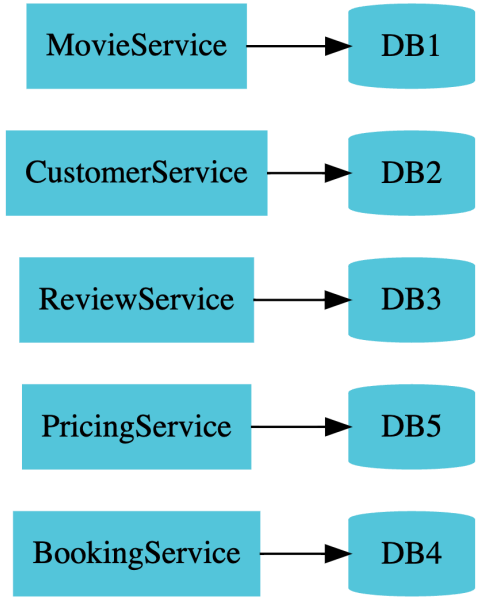
In28 Minutes

- **Let's keep it simple**: Three Critical Focus Areas
  - **1: REST**: Built following REST API Standards and Best Practices
  - **2: Small Well Chosen Deployable Units**: Independently deployable units of small services
  - **3: Cloud Enabled**: Possible to scale up and down independent of each other



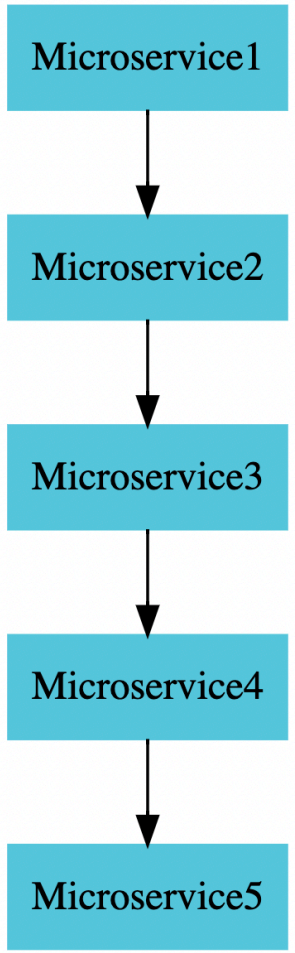
Movie Booking Application: Key Microservices

- **MovieService** - Central service managing movie details, showtimes, and availability
- **BookingService** - Handles ticket booking, seat selection, and booking management
- **PricingService** - Manages ticket pricing, discounts, and special offers
- **CustomerService** - Manages customer profiles, authentication, and customer support
- **ReviewService** - Allows users to submit and view reviews, ratings, and comments



Microservices - 3 Key Advantages

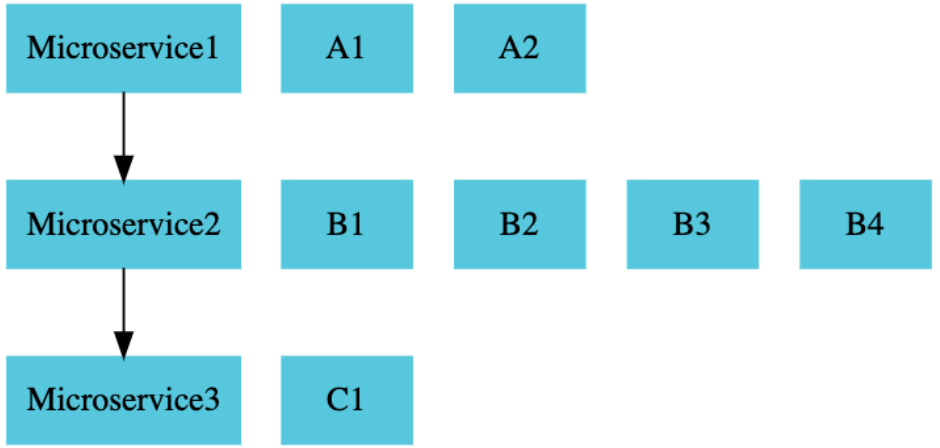
- **New Technology & Process Adoption** - Teams can adopt new technologies and processes for individual services
  - **Flexibility**: Choose best frameworks, and languages for each service
  - **Innovation**: Easier to experiment and use emerging technologies
- **Dynamic Scaling** - Enable scaling of individual components based on demand
  - **Efficiency**: Scale only the services that need it, reducing costs
- **Faster Release Cycles** - Smaller, independent services can be developed, tested, and deployed more quickly
  - **Agility**: Allows for more frequent updates and quicker response to market demands



Microservices - Key Challenges - 1

In28 Minutes

- **Bounded Context**: Defining clear boundaries for each service is crucial but challenging
- **Config Management**: Managing configurations for multiple services across environments can be complex
  - 100s of microservices **X** multiple instances **X** multiple environments
- **Dynamic Scaling**: Scaling services up or down in response to demand requires robust infrastructure

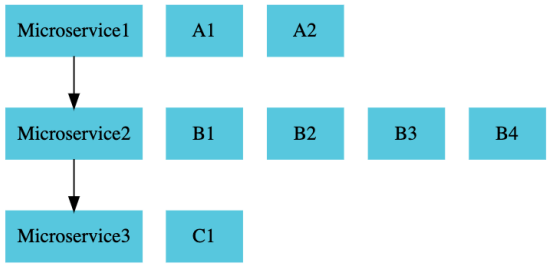


Microservices - Key Challenges - 2

- **Visibility**: Ensuring you have insights into service performance and health across the system is essential
- **Inter-dependencies**: Services can become a "pack of cards," where failure in one can affect others
  - How can you identify root-cause of failure?
- **Orchestration**: Coordinating multiple services to work together seamlessly is critical and challenging
  - Release management also becomes complex

Microservice - Solutions

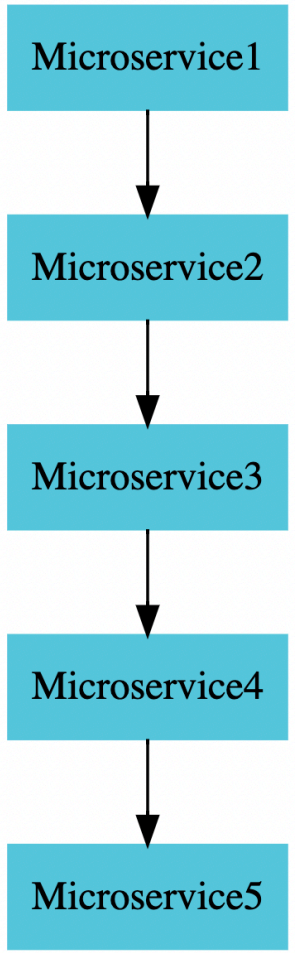
- **Spring Boot**: Faster API development
- **Spring Cloud**: Umbrella Project
  - **Configuration Management**: Spring Cloud Config Server
  - **Load Distribution**: Spring Cloud Load Balancer, Ribbon(OLD)
  - **Location Transparency**: Eureka Naming Server
  - **Tracing**: Zipkin
  - **API Gateway**: Spring Cloud Gateway, Zuul (OLD)
  - **Fault Tolerance**: Resilience4j, Hystrix (OLD)
- **Docker**: Language & Cloud Neutral deployable units
- **Kubernetes**: Orchestrate 1000s of Microservices



Microservices - Common Features

In28 Minutes

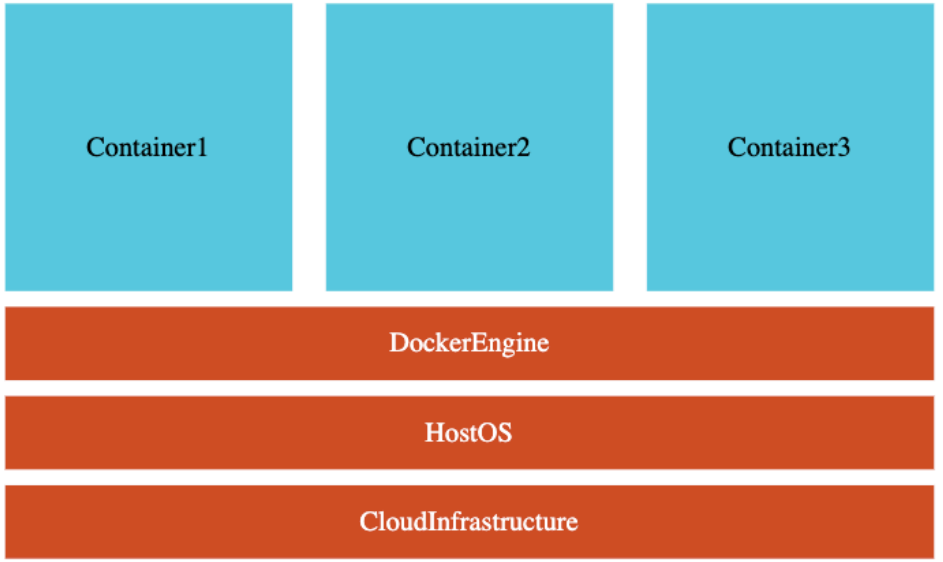
- **Small** - Each service is focused on a single business capability
- **Independent Deployment** - Services can be deployed independently without affecting others
- **Simple Communication** - Services communicate with each other using lightweight protocols like HTTP/REST
- **Stateless** - Services do not retain client state between requests (more easily scalable)
- **Automated Build and Deployment** - Continuous integration and delivery pipelines ensure quick and reliable deployments
- **Resilience** - Services are designed to handle failures gracefully, ensuring minimal impact on the overall system



Microservices - V2

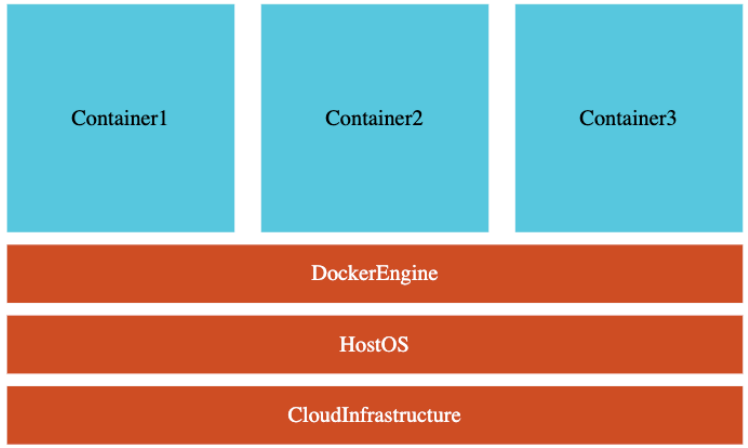
In28 Minutes

- **V2 (2.4+)** - Latest Releases of
  - Spring Boot
  - Spring Cloud
  - Docker and
  - Kubernetes
  - **Continue on to next lecture :)**
- **V1** - Old Versions - MOVED to APPENDIX
  - Spring Boot v2.3 and LOWER
  - **Go To Appendix :)**



Microservices - Evolution

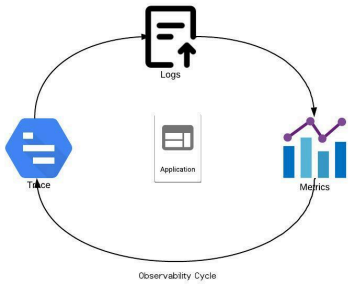
- **Goal**: Evolve with Microservices
  - **V1** - Spring Boot 2.0.0 to 2.3.x
  - **V2** - Spring Boot 2.4.0 to 3.0.0 to ...
    - Spring Cloud LoadBalancer (Ribbon)
    - Spring Cloud Gateway (Zuul)
    - Resilience4j (Hystrix)
    - NEW: Docker
    - NEW: Kubernetes
    - NEW: Observability
      - NEW: Micrometer (Spring Cloud Sleuth)
      - NEW: OpenTelemetry



Microservices - Spring Boot 2 vs Spring Boot 3

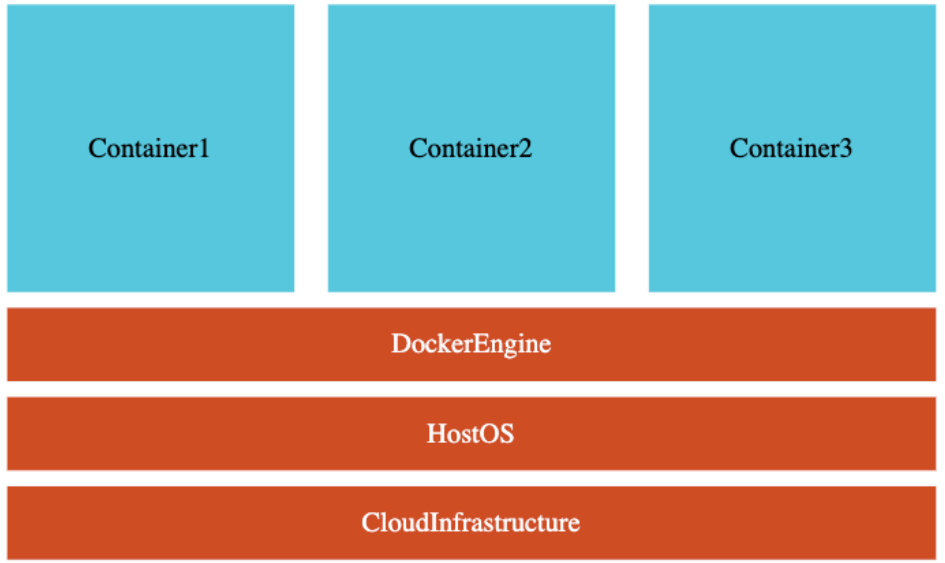
In28 Minutes

- **V1**(2.0.0 to 2.3.x)
- **V2** (2.4.x to 3.0.0 to ..)
- Spring Boot 2.4.0+
  - ***https://github.com/in28minutes/spring-microservices-v2***
- Spring Boot 3.0.0+
  - ***https://github.com/in28minutes/spring-microservices-v3***
  - Notes: v3-upgrade.md
  - Key Changes:
    - **Observability** - Ability of a system to measure its current state based on the generated data
      - Monitoring is reactive while Observability is proactive
      - **OpenTelemetry**: One Standard for Logs + Traces + Metrics



Microservices - V2

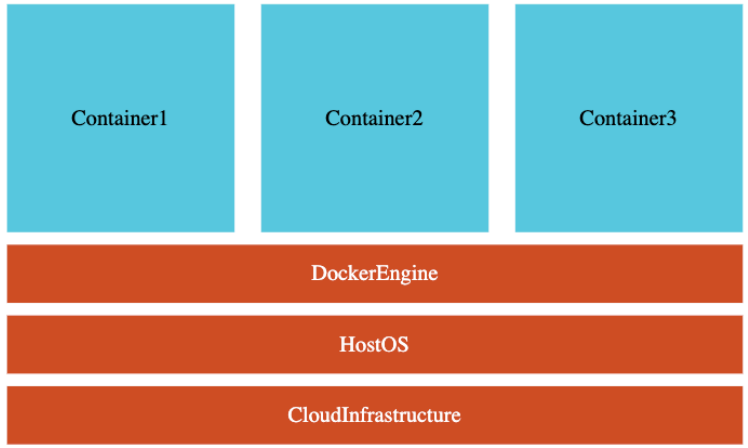
- You have **skipped V1**
  - Go to next lecture!
- You have **completed V1**
  - Option 1: **Start from Zero Again**:
    - Go to the next lecture!
  - Option 2: **Get a Quick Start**:
    - Jump to **"Step 21 - QuickStart by Importing Microservices"**
      - Same microservices as V1: **Currency Exchange** and **Currency Conversion**
      - Very little changes in **Eureka Naming Server**
      - **Step 21** helps you set these up and get started quickly!

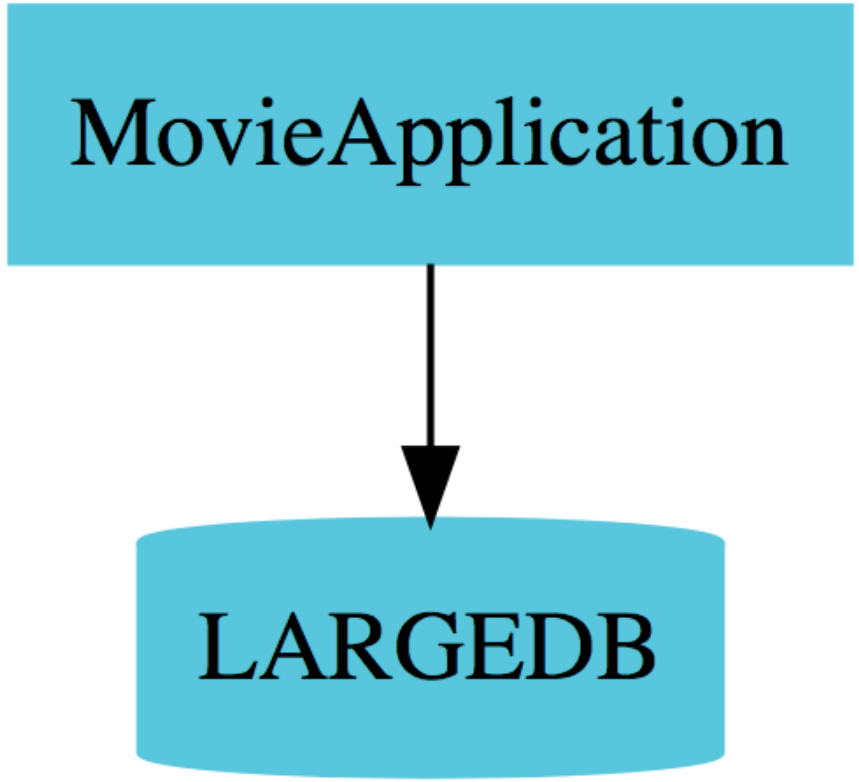


Microservices - V2 - What's New

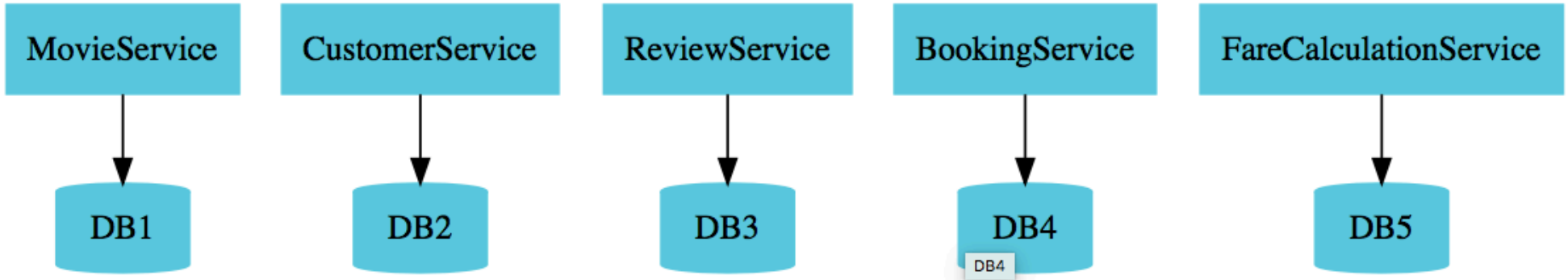
In28 Minutes

- Microservices Evolve Quickly
- **V2** (Spring Boot - 2.4.x to 3.0.0 to LATEST)
  - **Spring Cloud LoadBalancer** instead of Ribbon
  - **Spring Cloud Gateway** instead of Zuul
  - **Resilience4j** instead of Hystrix
  - **Docker**: Containerize Microservices
    - Run microservices using Docker and Docker Compose
  - **Kubernetes**: Orchestrate all your Microservices with Kubernetes
  - **OpenTelemetry**: One Standard - Logs, Traces & Metrics
  - **Micrometer** (Replaces Spring Cloud Sleuth)





Monolith



Microservices

What is a Microservice?

Small autonomous services that work together

Sam Newman

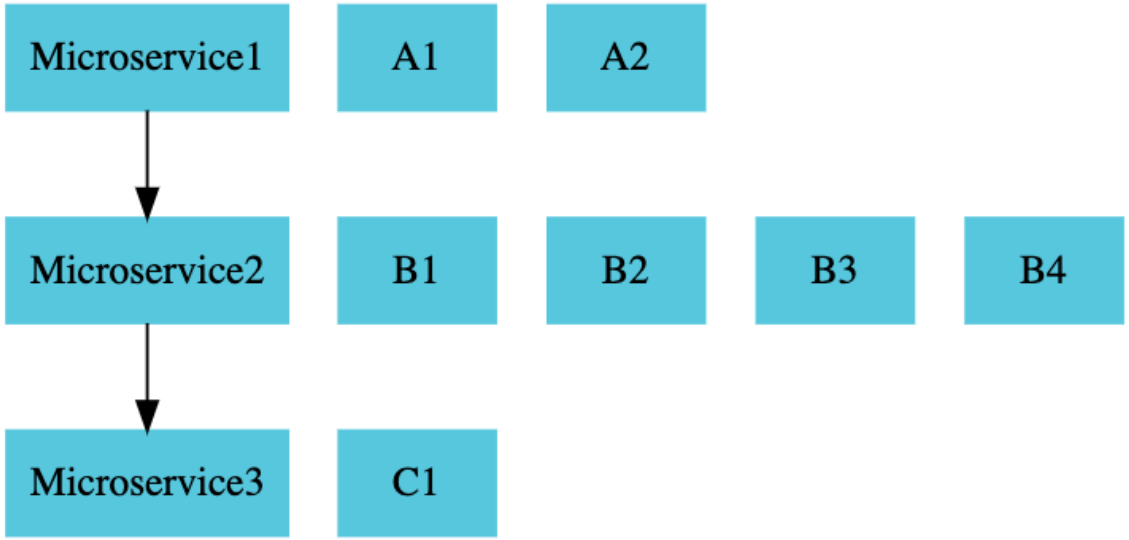
What is a Microservice?

Approach to developing a application as a suite of small services, each running in its own process and communicating with lightweight mechanisms often an HTTP resource API.
These services are built around business capabilities and independently deployable by fully automated deployment machinery.
There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

James Lewis and Martin Fowler

Microservices for me

- REST
- Small Well Chosen Deployable Units
- Cloud Enabled



Microservices - Challenges

- Bounded Context
- Configuration Management
- Dynamic Scale Up and Scale Down
- Visibility
- Pack of Cards
- Zero Downtime Deployments

Microservice - Solutions

- **Spring Cloud** Umbrella Projects
  - Centralized Configuration Management (Spring Cloud Config Server)
  - Location Transparency - Naming Server (Eureka)
  - Load Distribution (Ribbon, Spring Cloud Load Balancer)
  - Visibility and Monitoring (Zipkin)
  - API Gateway (Zuul, Spring Cloud Gateway)
  - Fault Tolerance (Hystrix, Resilience4j)
- **Docker**: Language Neutral, Cloud Neutral deployable units
- **Kubernetes**: Orchestrate Thousands of Microservices

Microservices - 3 Key Advantages

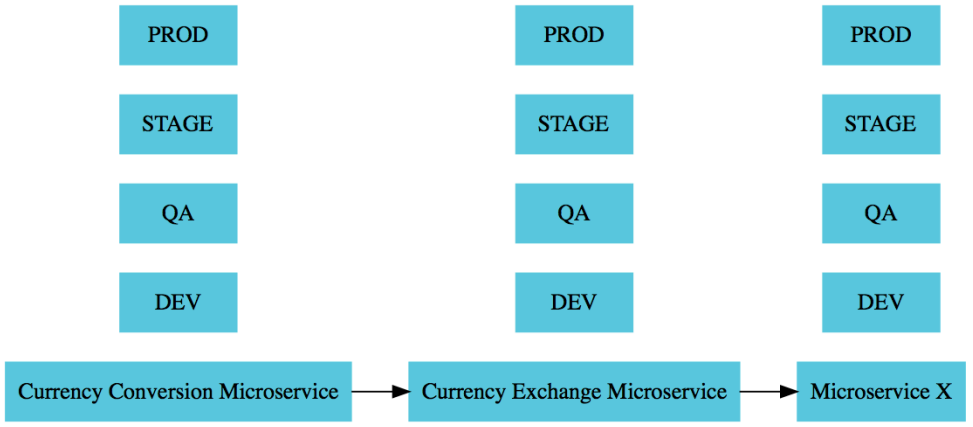
- New Technology & Process Adoption
- Dynamic Scaling
- Faster Release Cycles

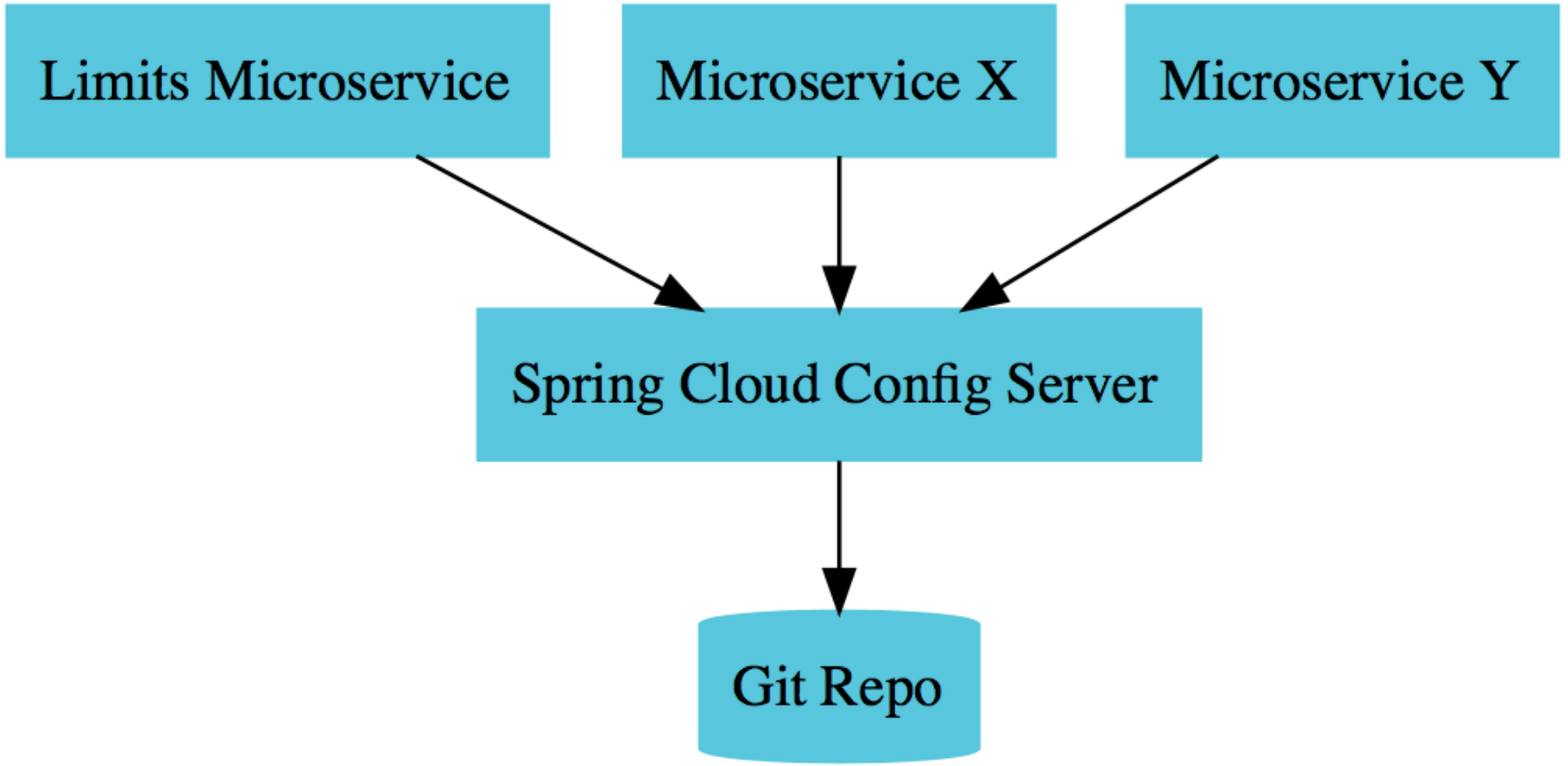
Ports Standardization

Application	Port
Limits Microservice	8080, 8081, ...
Spring Cloud Config Server	8888
Currency Exchange Microservice	8000, 8001, 8002, ..
Currency Conversion Microservice	8100, 8101, 8102, ...
Netflix Eureka Naming Server	8761
API Gateway	8765
Zipkin Distributed Tracing Server	9411

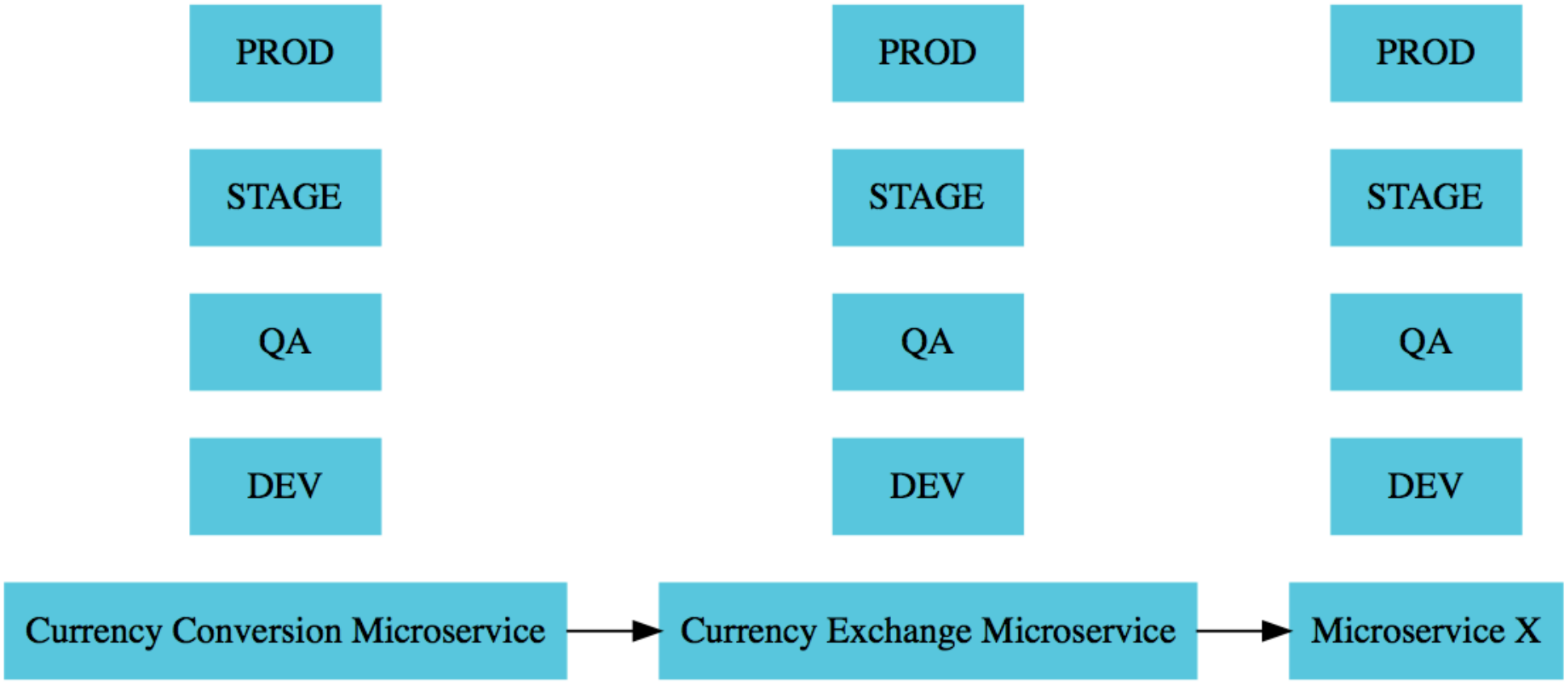
Need for Centralized Configuration

- Lot of configuration:
  - External Services
  - Database
  - Queue
  - Typical Application Configuration
- Configuration variations:
  - 1000s of Microservices
  - Multiple Environments
  - Multiple instances in each Environment
- How do you manage all this configuration?

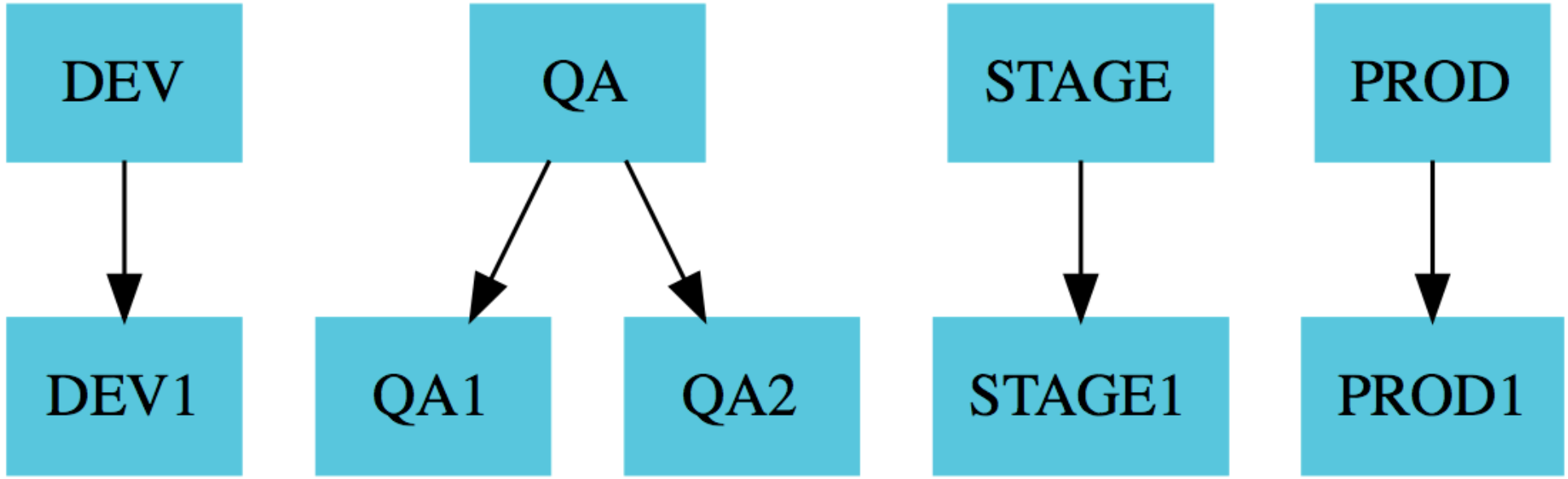




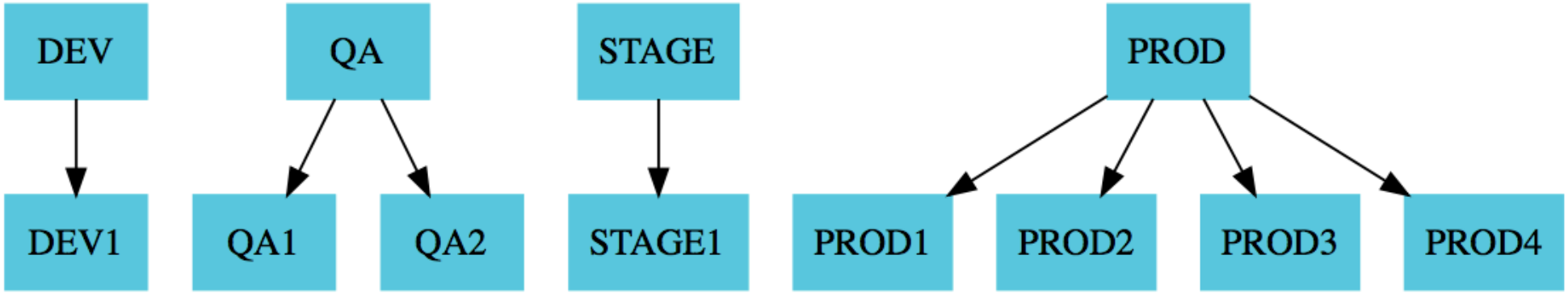
Config Server



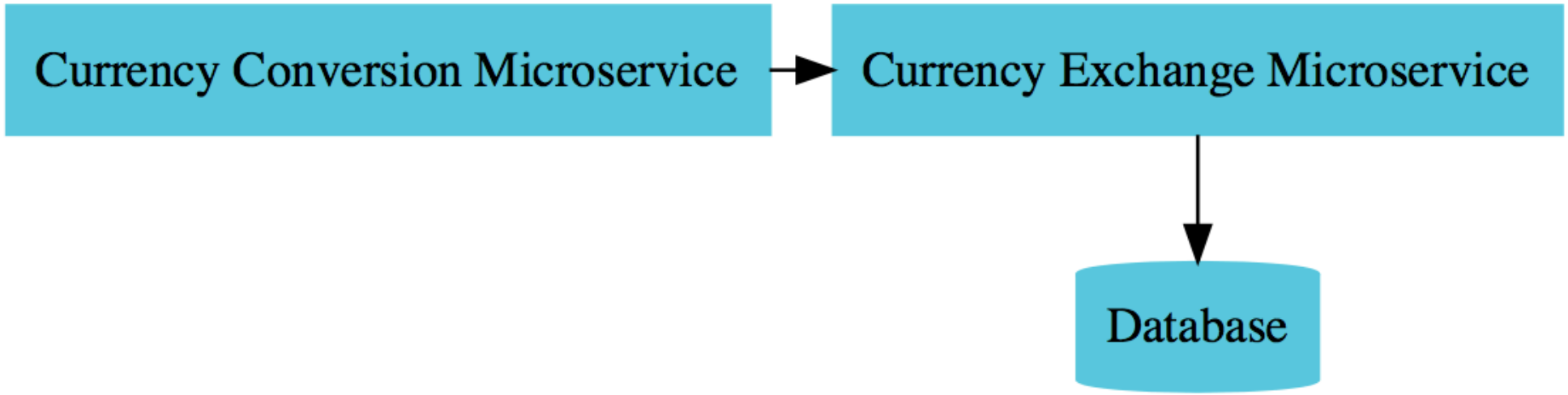
Environments



Environments



Environments



Microservices Overview

Currency Exchange Microservice

In28 Minutes

What is the exchange rate of one currency in another?

```
http://localhost:8000/currency-exchange/from/USD/to/INR

{
   "id":10001,
   "from":"USD",
   "to":"INR",
   "conversionMultiple":65.00,
   "environment":"8000 instance-id"
}
```

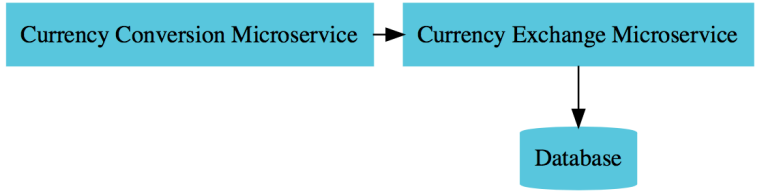
Currency Conversion Microservice

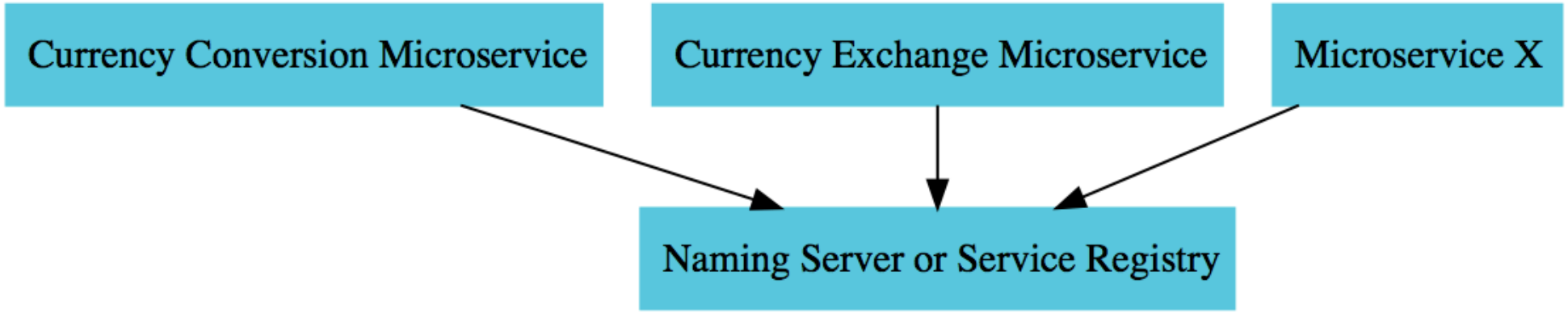
In28 Minutes

Convert 10 USD into INR

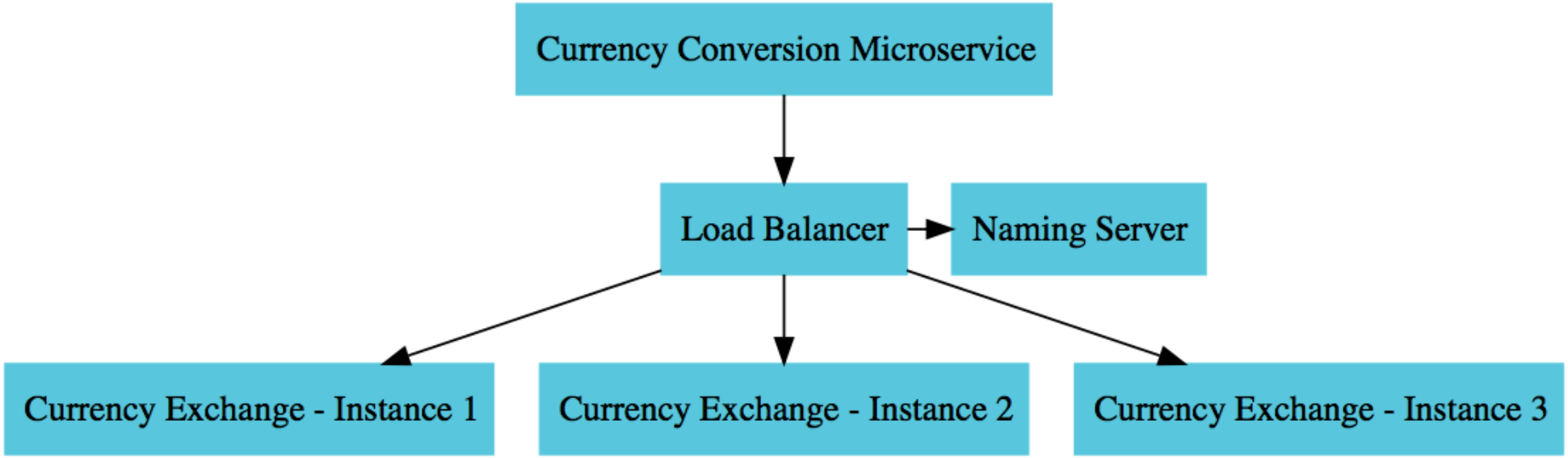
```
http://localhost:8100/currency-conversion/from/USD/to/INR/quantity/10

{
  "id": 10001,
  "from": "USD",
  "to": "INR",
  "conversionMultiple": 65.00,
  "quantity": 10,
  "totalCalculatedAmount": 650.00,
  "environment": "8000 instance-id"
}
```





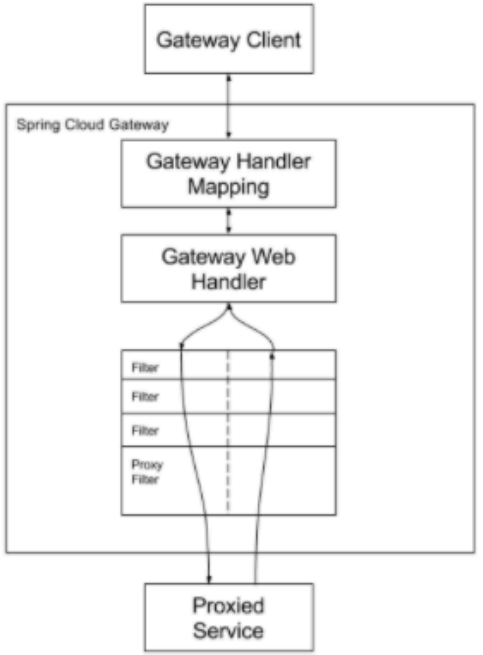
Naming Server



Load Balancing

Spring Cloud Gateway

- Simple, yet effective way to route to APIs
- Provide cross cutting concerns:
  - Security
  - Monitoring/metrics
- Built on top of Spring WebFlux (Reactive Approach)
- Features:
  - Match routes on any request attribute
  - Define Predicates and Filters
  - Integrates with Spring Cloud Discovery Client (Load Balancing)
  - Path Rewriting



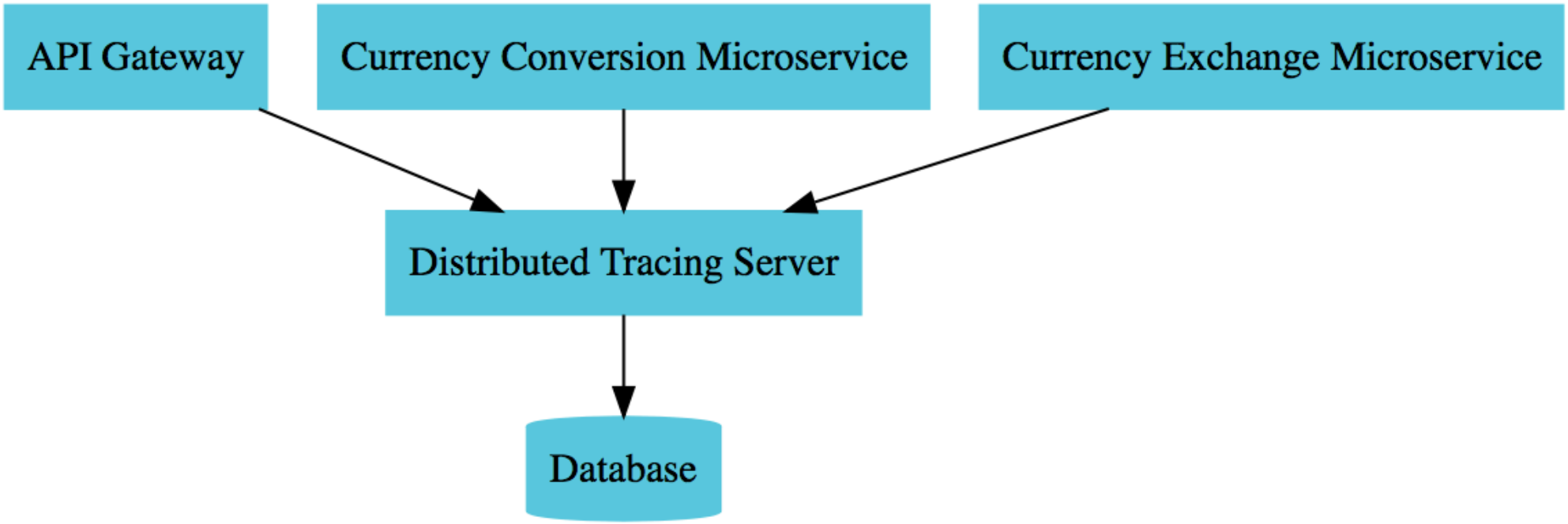
From *https://docs.spring.io*

Circuit Breaker

- What if one of the services is down or is slow?
  - Impacts entire chain!
- Questions:
  - Can we return a fallback response if a service is down?
  - Can we implement a Circuit Breaker pattern to reduce load?
  - Can we retry requests in case of temporary failures?
  - Can we implement rate limiting?
- Solution: Circuit Breaker Framework - Resilience4j

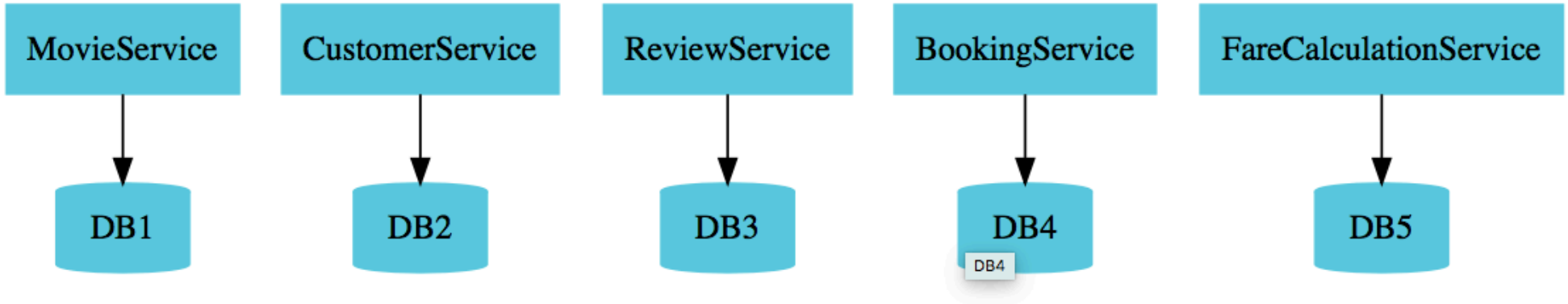
Distributed Tracing

- Complex call chain
- How do you debug problems?
- How do you trace requests across microservices?
- Enter Distributed Tracing



Distributed Tracing

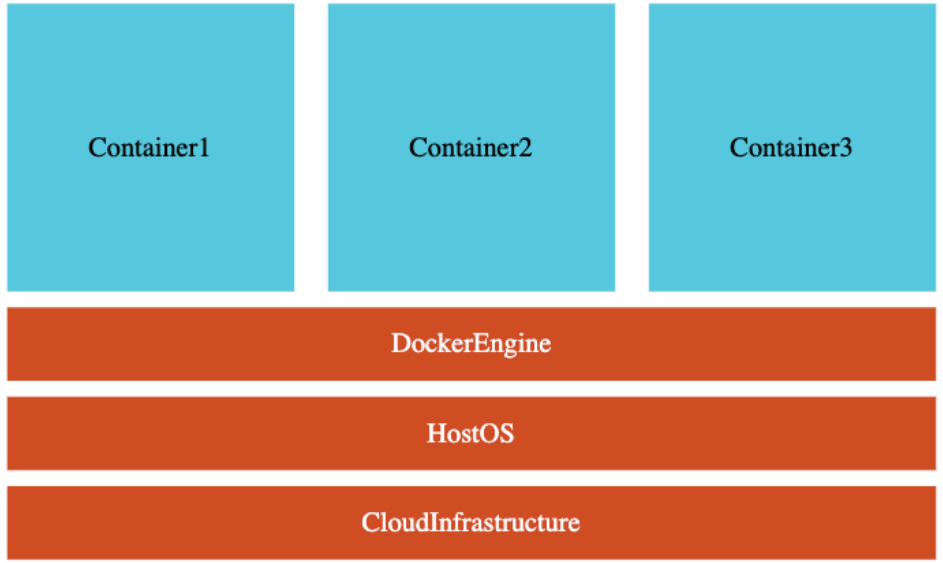
Microservices

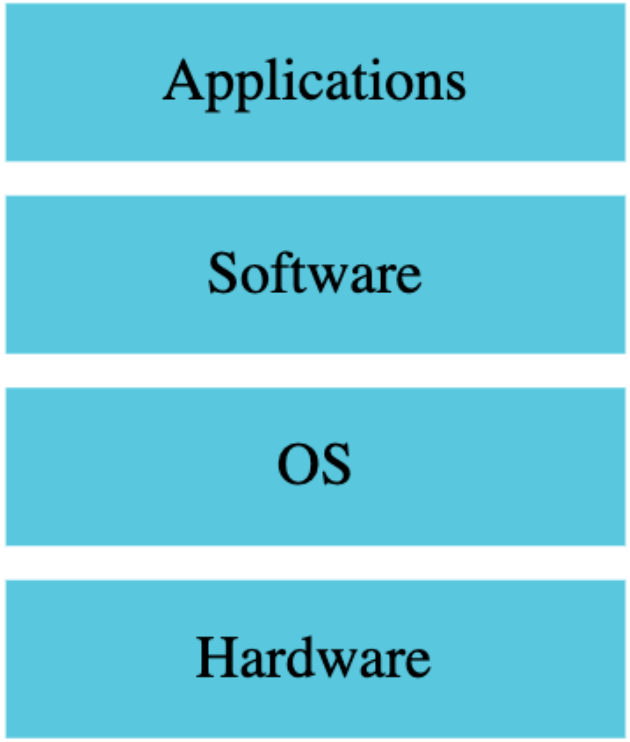


- Enterprises are heading towards microservices architectures
  - Build small focused microservices
  - **Flexibility to innovate** and build applications in different programming languages (Go, Java, Python, JavaScript, etc)
  - BUT **deployments become complex**!
  - How can we have **one way of deploying** Go, Java, Python or JavaScript .. microservices?
    - Enter **containers**!

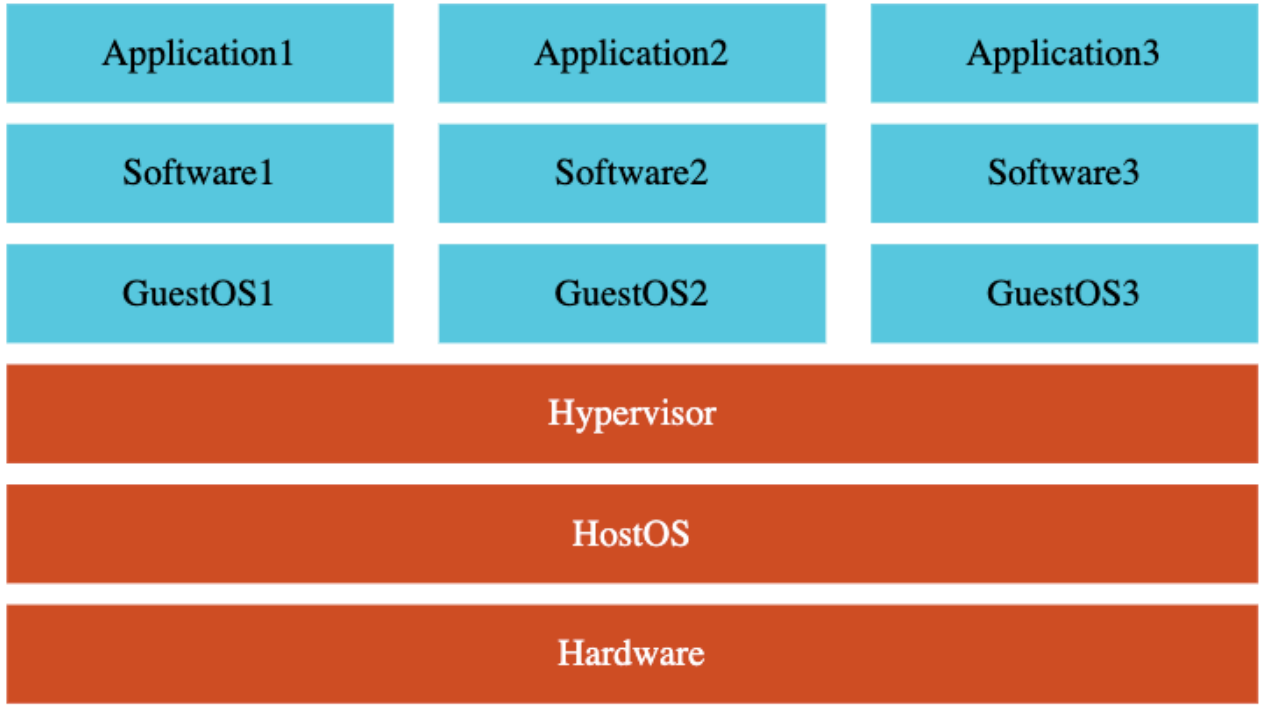
Docker

- Create **Docker images** for each microservice
- Docker image **contains everything a microservice needs** to run:
  - Application Runtime (JDK or Python or NodeJS)
  - Application code
  - Dependencies
- You can run these docker containers **the same way** on any infrastructure
  - Your local machine
  - Corporate data center
  - Cloud

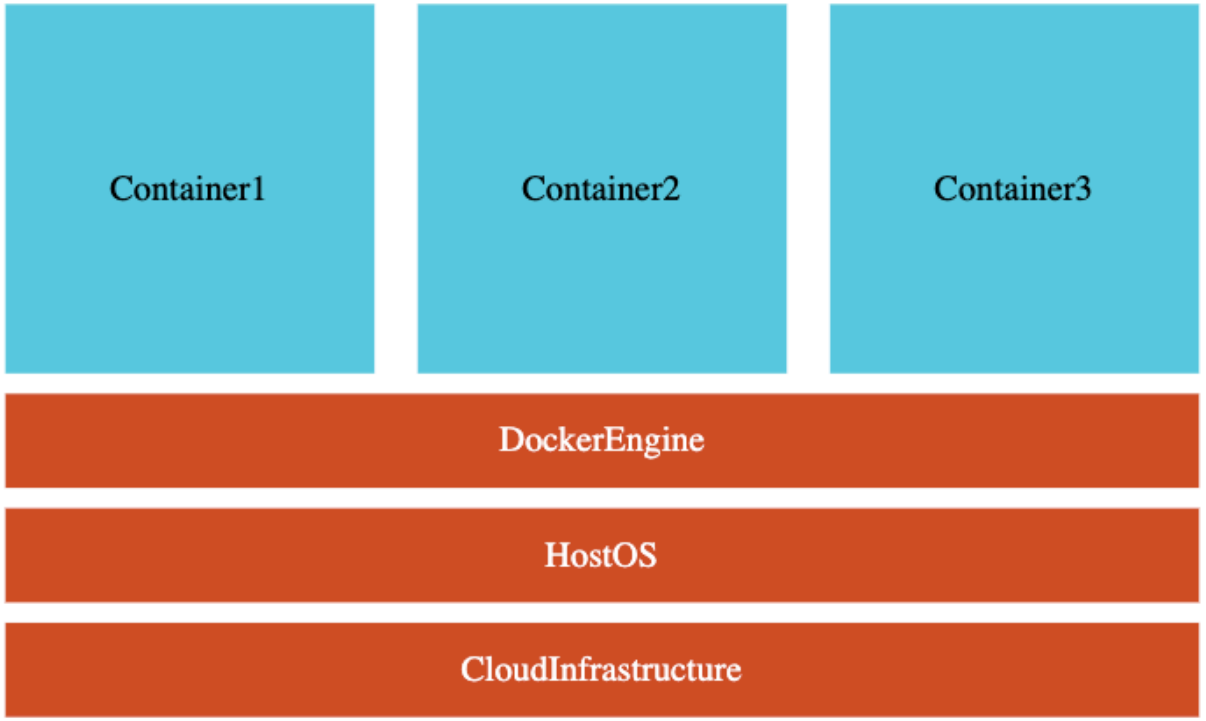




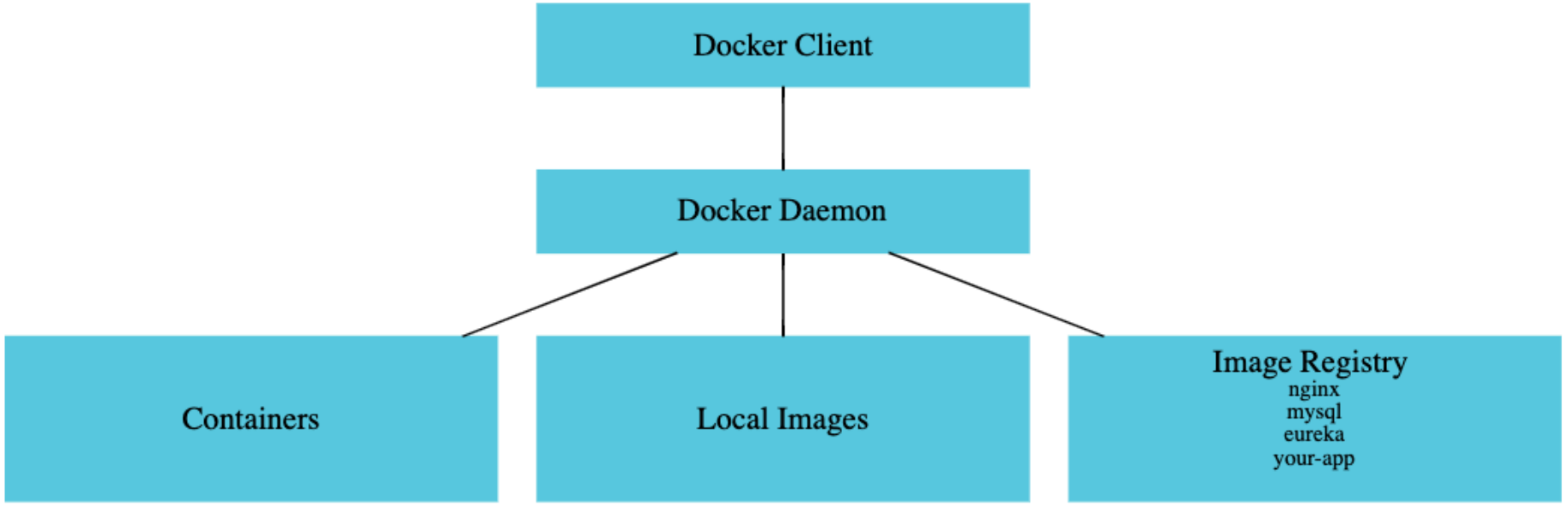
Traditional Deployment



Deployments using Virtual Machines



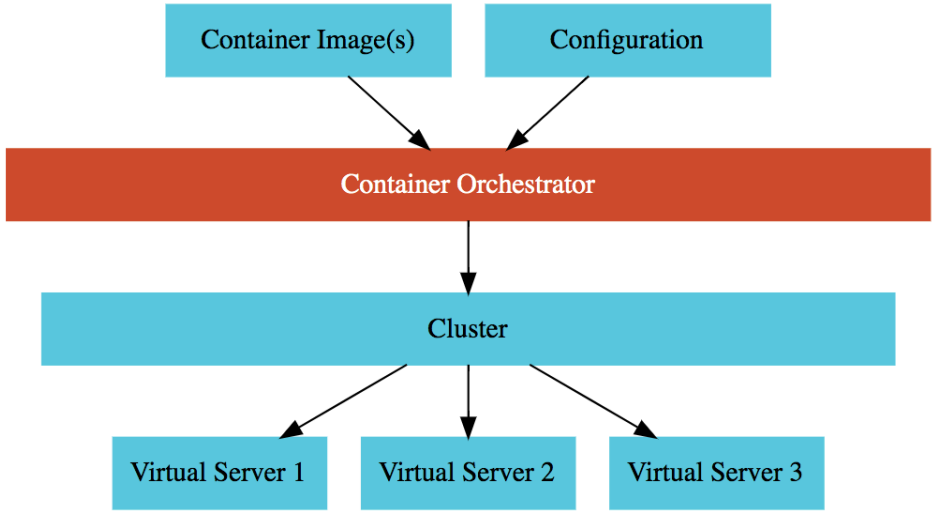
Deployments using Docker



Docker Architecture

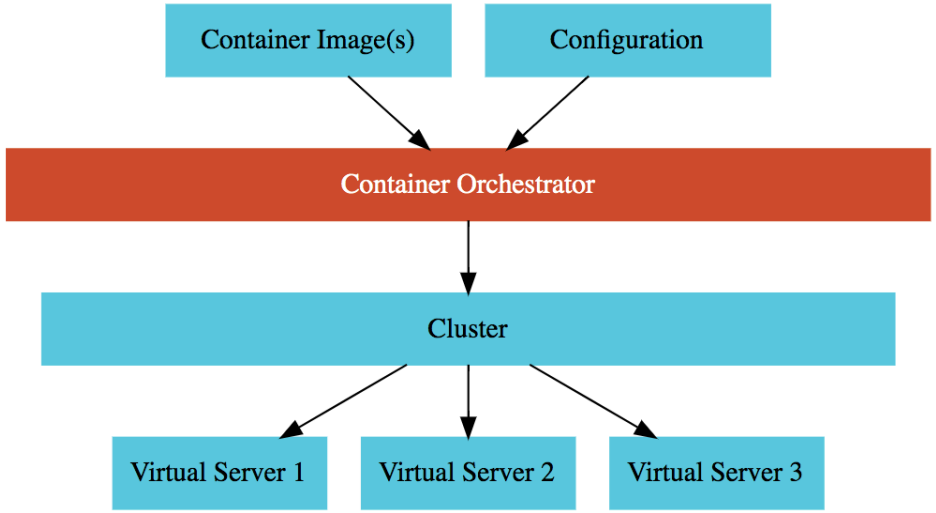
Container Orchestration

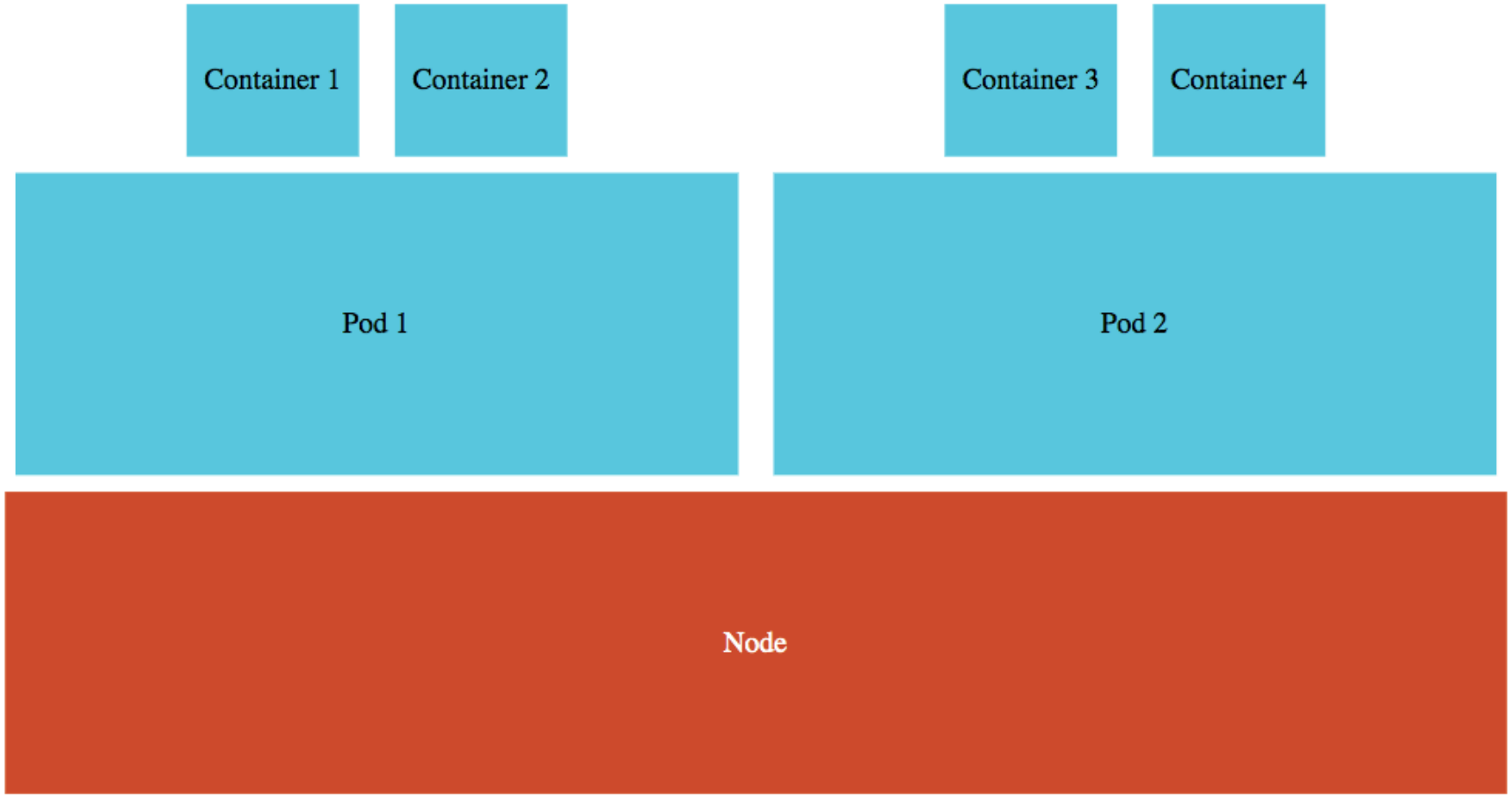
- **Requirement** : I want 10 instances of Microservice A container, 15 instances of Microservice B container and ....
- Typical Features:
  - **Auto Scaling** - Scale containers based on demand
  - **Service Discovery** - Help microservices find one another
  - **Load Balancer** - Distribute load among multiple instances of a microservice
  - **Self Healing** - Do health checks and replace failing instances
  - **Zero Downtime Deployments** - Release new versions without downtime

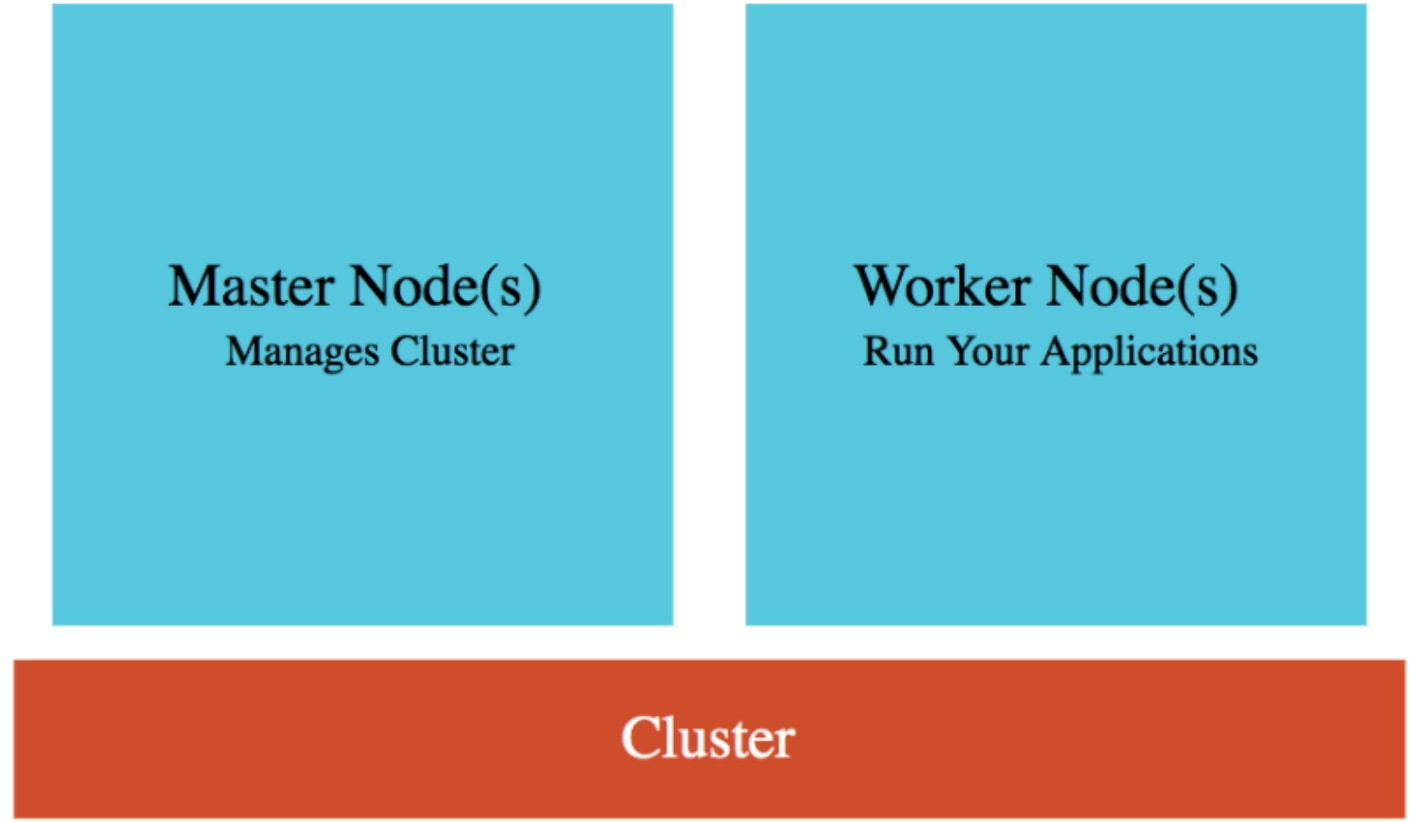


Container Orchestration Options

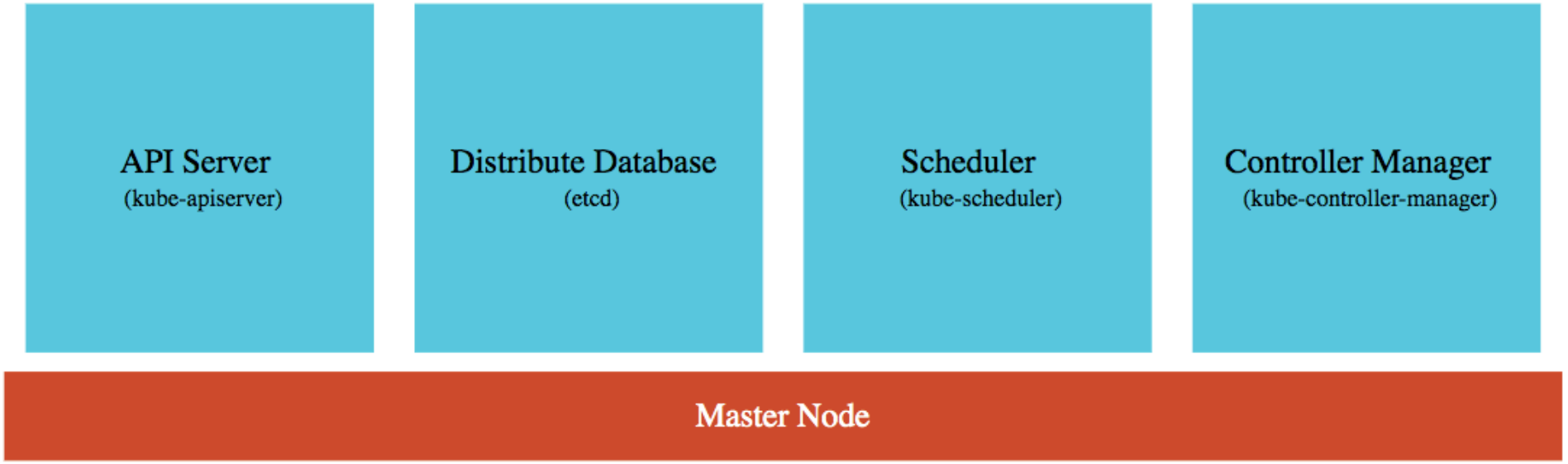
- **AWS Specific**
  - AWS Elastic Container Service (ECS)
  - AWS Fargate : Serverless version of AWS ECS
- **Cloud Neutral** - Kubernetes
  - AWS - Elastic Kubernetes Service (EKS)
  - Azure - Azure Kubernetes Service (AKS)
  - GCP - Google Kubernetes Engine (GKE)
  - EKS/AKS does not have a free tier!
    - We use GCP and GKE!



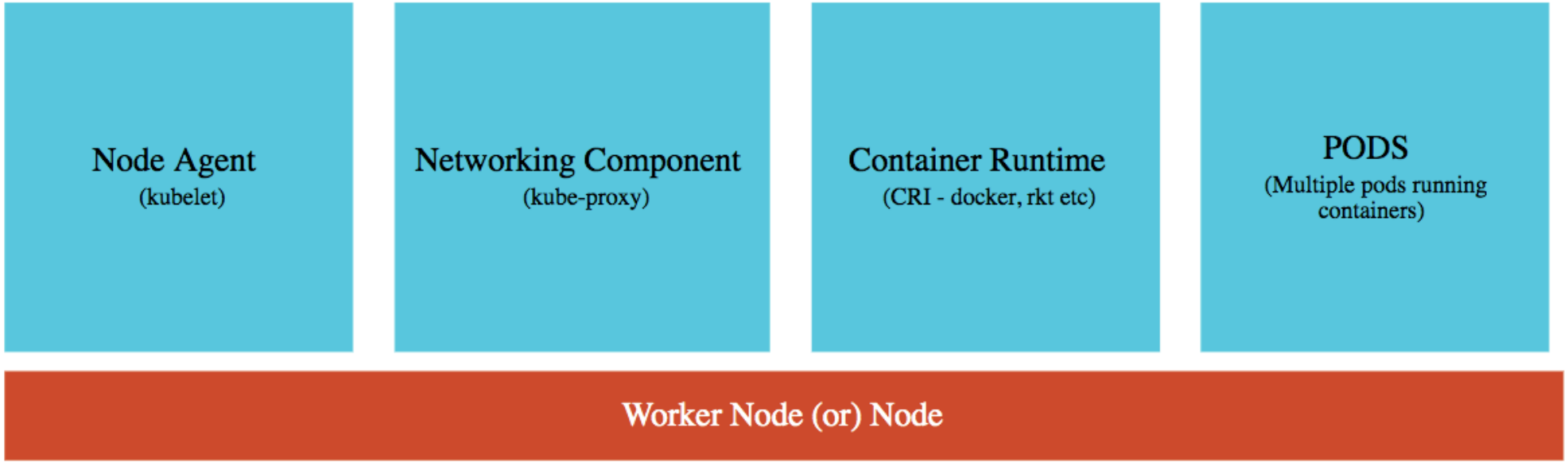

Container 1
Container 2
Container 3
Container 4
Pod 1
Pod 2
Node



Kubernetes Architecture



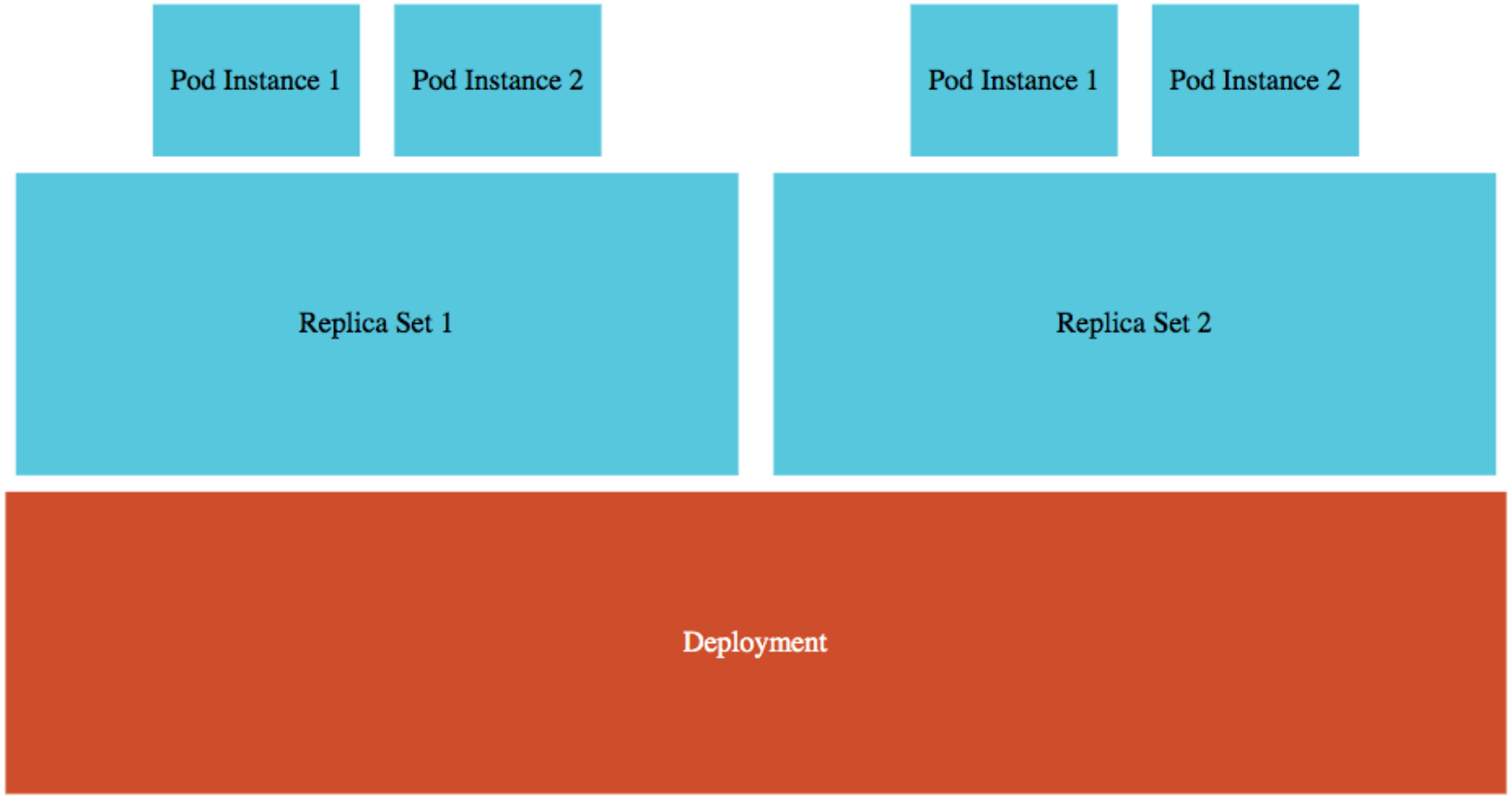
Kubernetes Architecture



Kubernetes Architecture

Kubernete Deployments

Kubernete Deployments



Kubernete Deployments

Kubernete Service

Kubernetes - Liveness and Readiness Probes

- Kubernetes uses probes to check the health of a microservice:
  - If readiness probe is not successful, no traffic is sent
  - If liveness probe is not successful, pod is restarted
- Spring Boot Actuator (>=2.3) provides inbuilt readiness and liveness probes:
  - /health/readiness
  - /health/liveness

What Next?

FASTEST ROADMAPS

in28minutes.com

In28 Minutes

- Google Cloud Certifications
- Azure Certifications
- AWS Certifications
- DevOps
- Java Full Stack
- Java Microservices