Deep diving into micro-services

May, 20-21 2017

- Sergey Morenets, 2017

DEVELOPER 12 YEARS

TRAINER 4 YEARS

WRITER 3 BOOKS

- Sergey Morenets, 2017

FOUNDER

SPEAKER

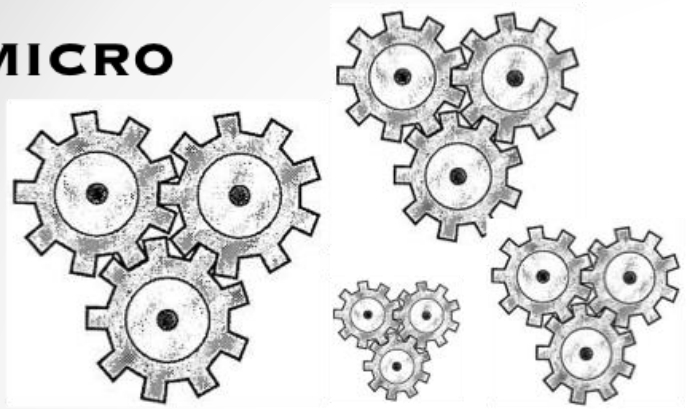
- Sergey Morenets, 2

Agenda

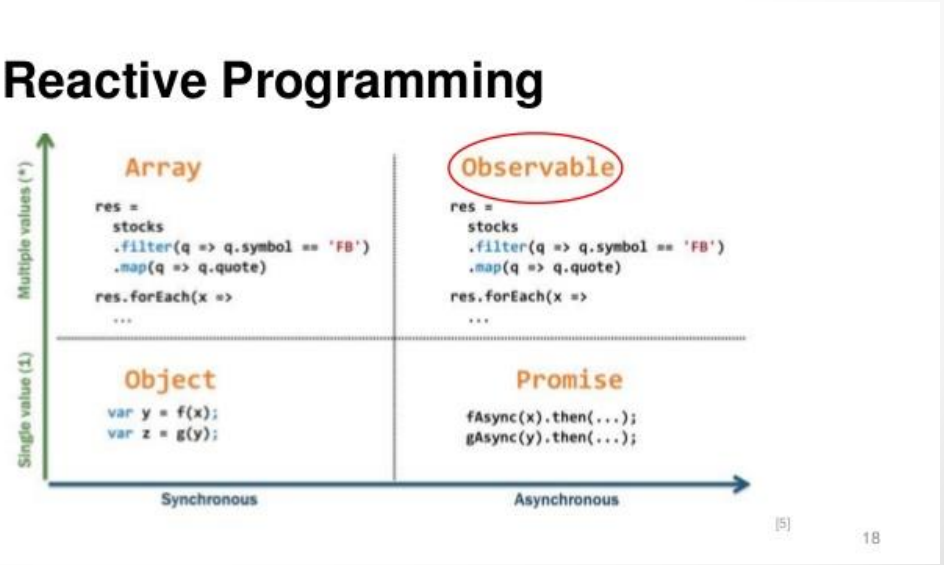
- ✓ Spring Framework infrastructure
- ✓ Complexity of monolith applications
- ✓ Micro-service architecture. Pro and cons
- ✓ Spring Cloud
- ✓ Load balancing
- ✓ Service monitoring
- ✓ Partitioning into micro-services
- ✓ MongoDB
- ✓ Apache Kafka

- Sergey Morenets, 2017

MICRO



SERVICES

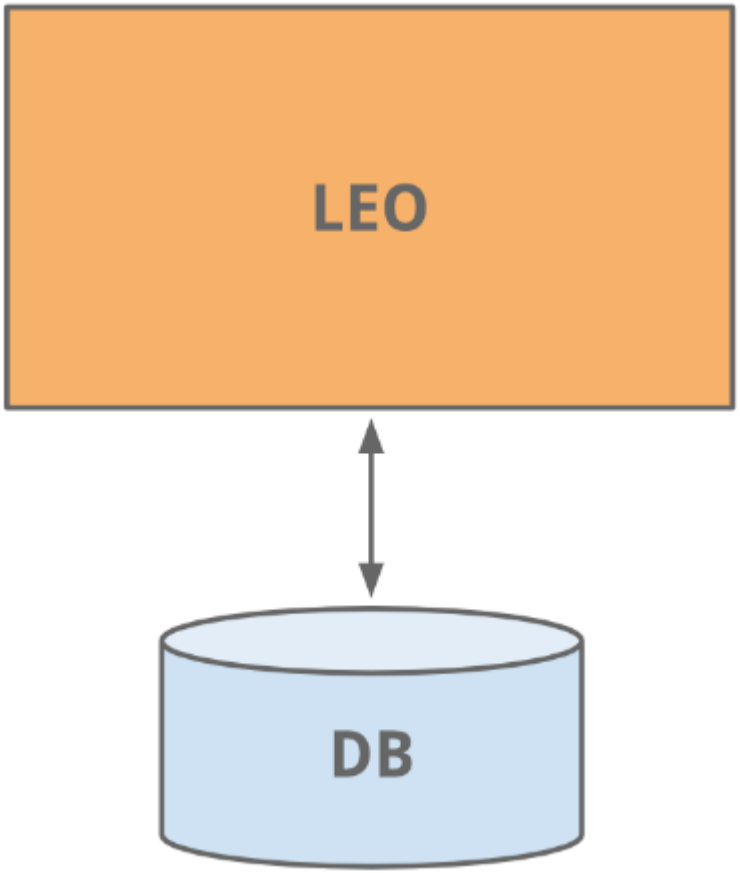


- Sergey Morenets, 2017

Story with happy-end

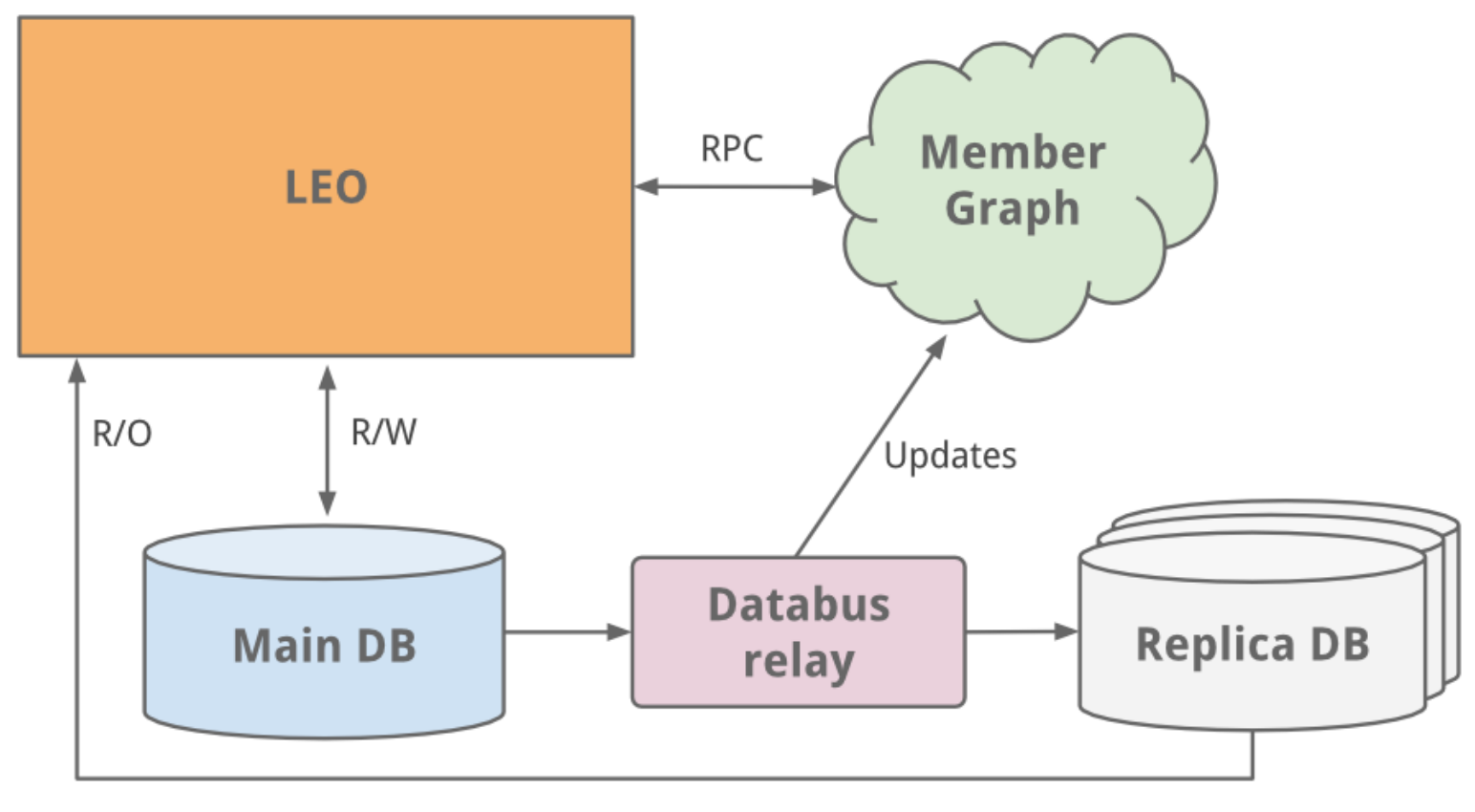
- Sergey Morenets, 2017

LinkedIn. Long ago ...



- Sergey Morenets, 2017

LinkedIn. Some time in the past

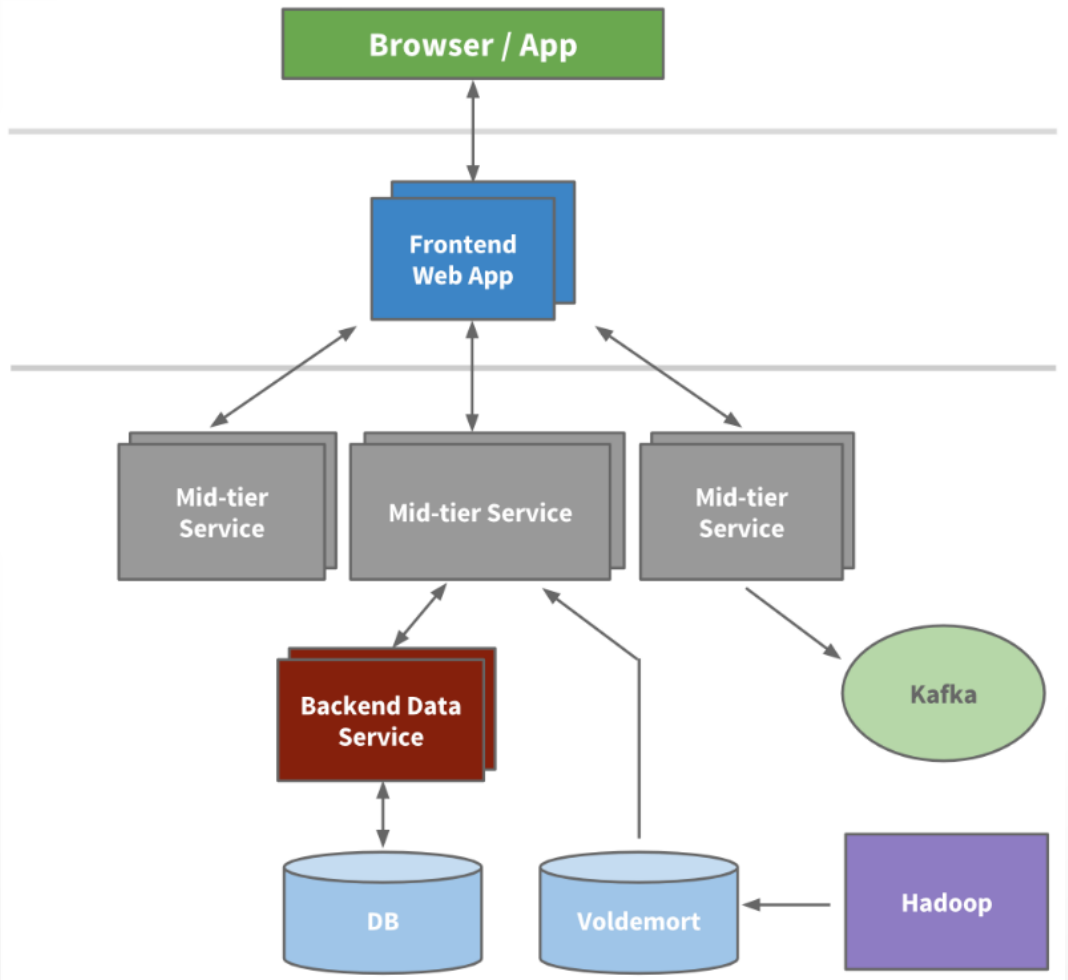


- Sergey Morenets, 2017

LinkedIn. Some time in the past

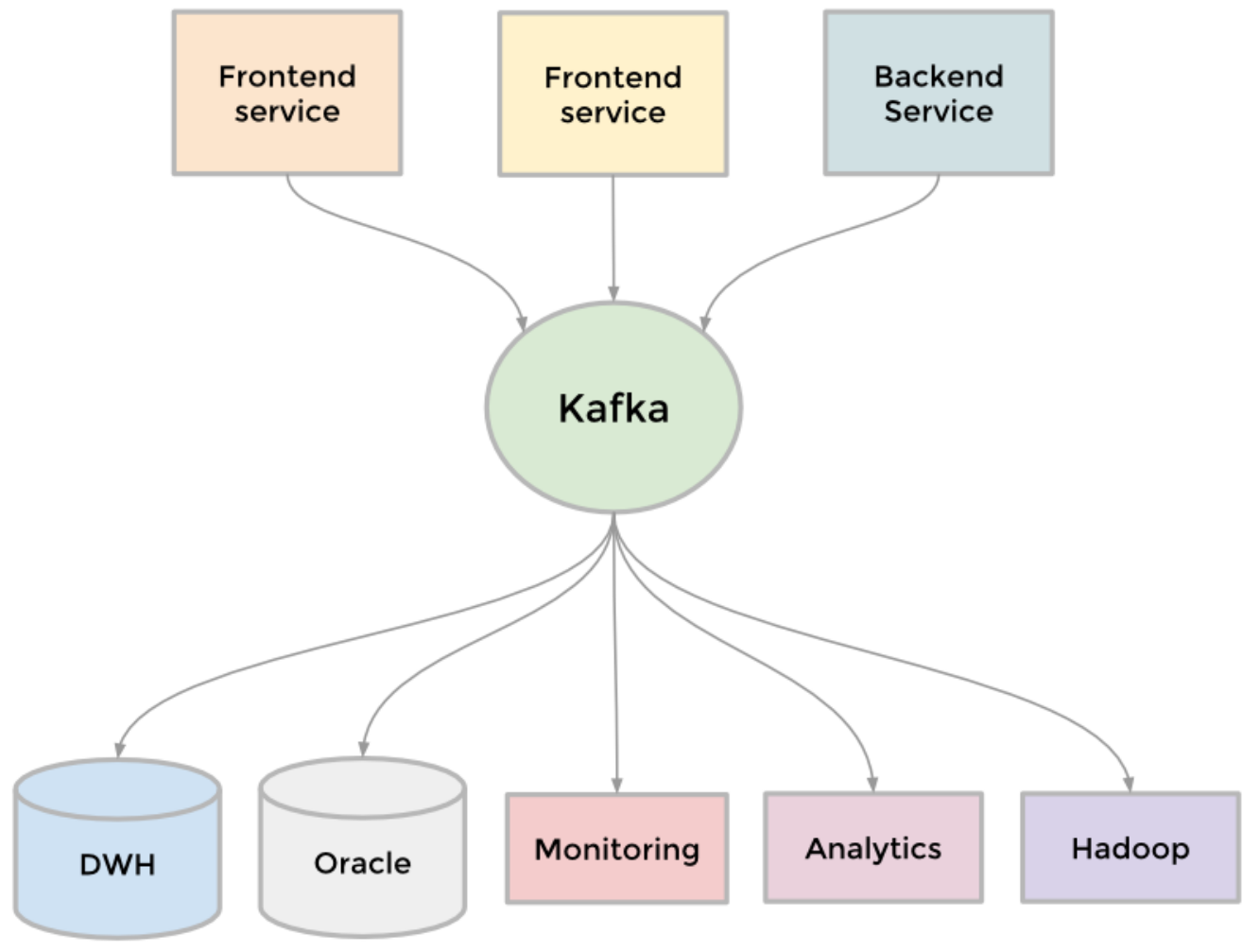
- Sergey Morenets, 2017

LinkedIn. Nowadays



- Sergey Morenets, 2017

LinkedIn. Nowadays



- Sergey Morenets, 2017

LinkedIn. Statistics (2015)

- 500 000 000 000 requests per day
- 500 000 000 users
- More than 900 services

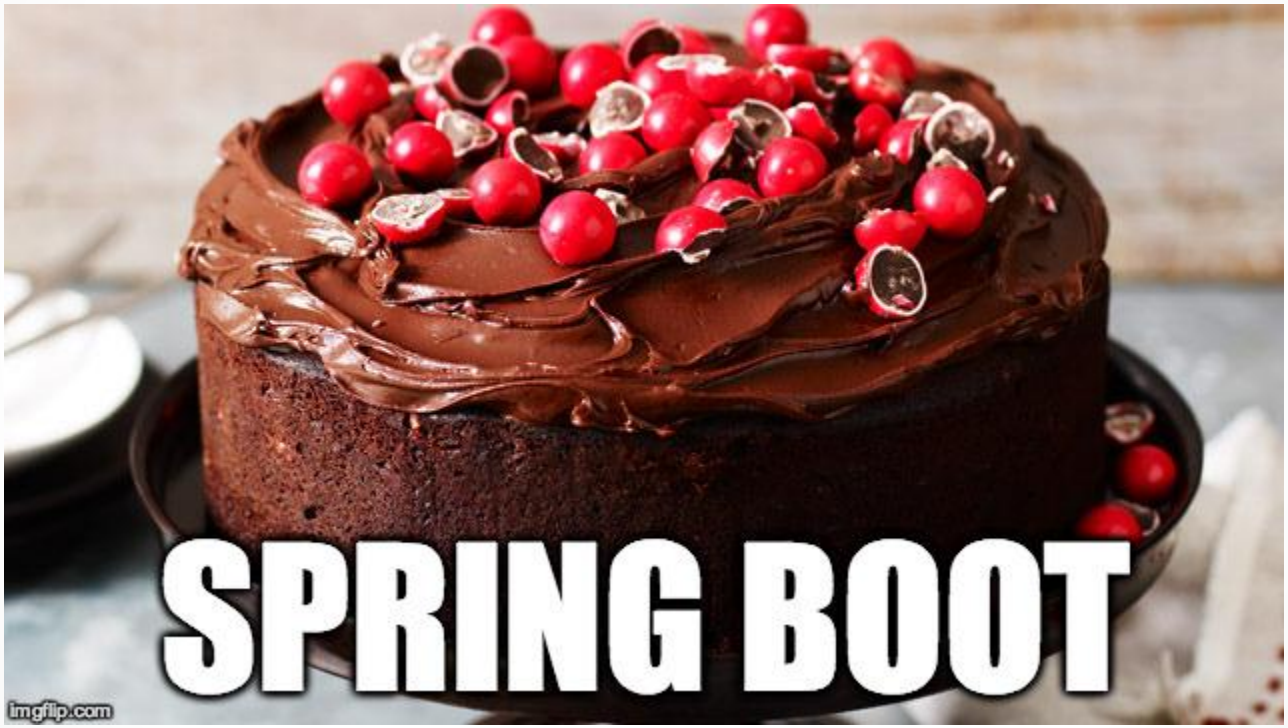
YouTube. Statistics (2017)

- 1 300 000 000 users
- 300 hours of video are uploaded every minute
- 5 000 000 000 videos are watched every single day
- 1 000 000 000 videos are watched in mobile every single day
- 30 000 000 visitors per day

Spring Framework

- Based on **DI**(IoC) conversion
- Integrated with most popular frameworks
- Contains over **30** sub-projects

- Sergey Morenets, 2017



- Sergey Morenets, 2017

Spring Boot

- Stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly
- Automatically Spring configuration
- Convention-over-configuration
- Absolutely no code generation and no requirement for XML configuration
- Focus on business features and less on infrastructure

Build management

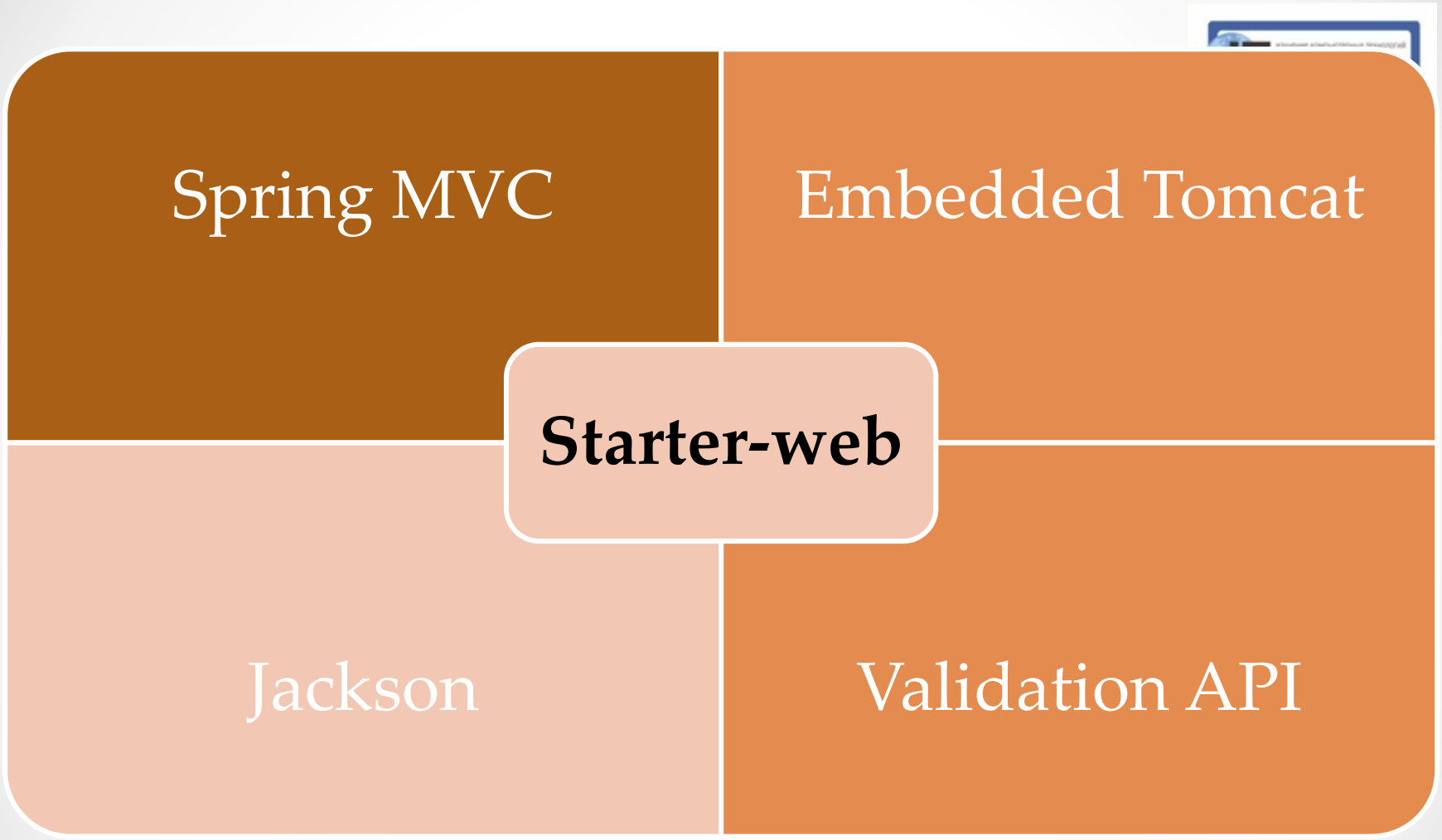
- Sergey Morenets, 2017

Maven. Web starter


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

Startup code


```
@SpringBootApplication
public class RestApplication {
    public static void main(String[] args) {
        SpringApplication.run(
                RestApplication.class, args);
    }
}
```


Spring MVC
Embedded Tomcat
Starter-web
Jackson
Validation API

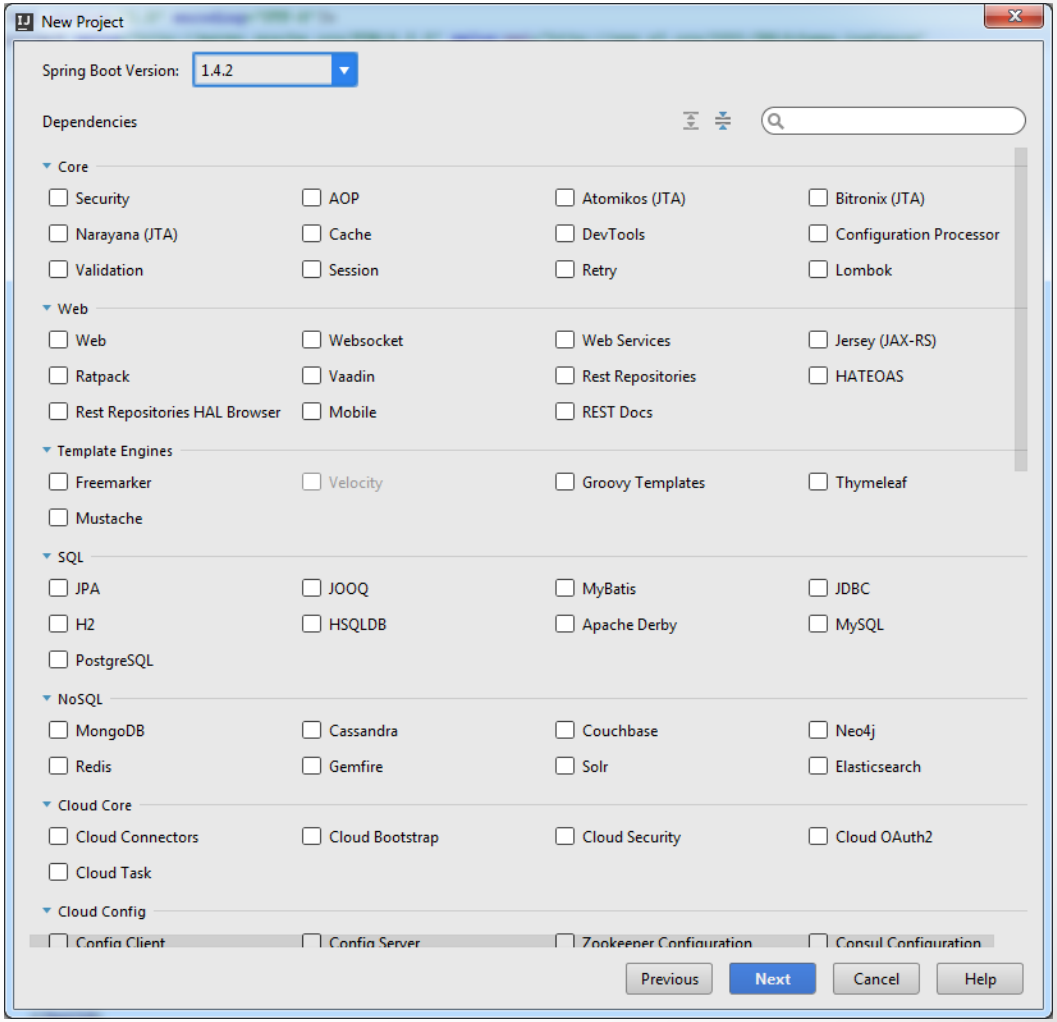
REST service


```
@RestController
@RequestMapping("book")
public class BookController {

    @GetMapping("/{id}")
    public Book getBook(@PathVariable int id) {
        return bookRepository.findBookById(id);
    }
```

IDE

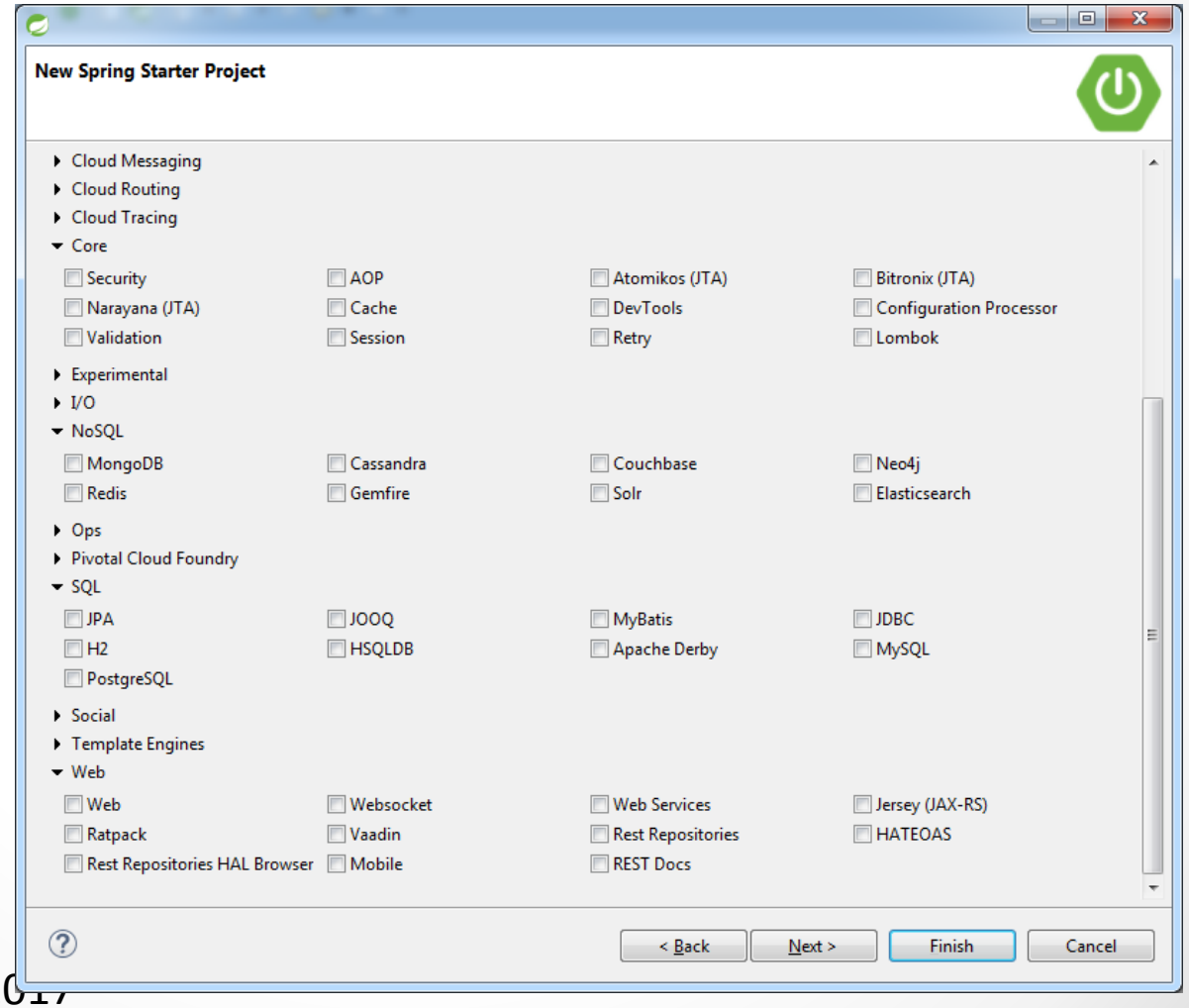
- Sergey Morenets, 2017

Spring Initializr



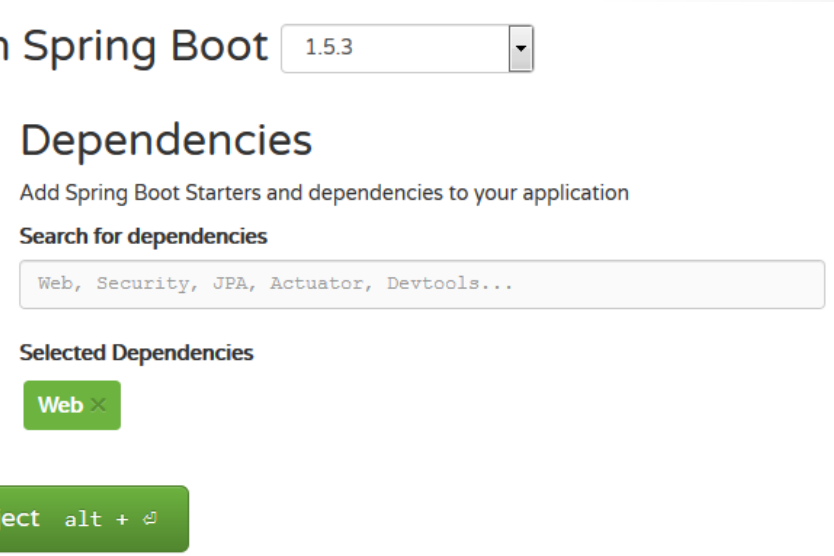
- Sergey Morenets, 2017

Spring Boot Starter



- Sergey Morenets, 2017

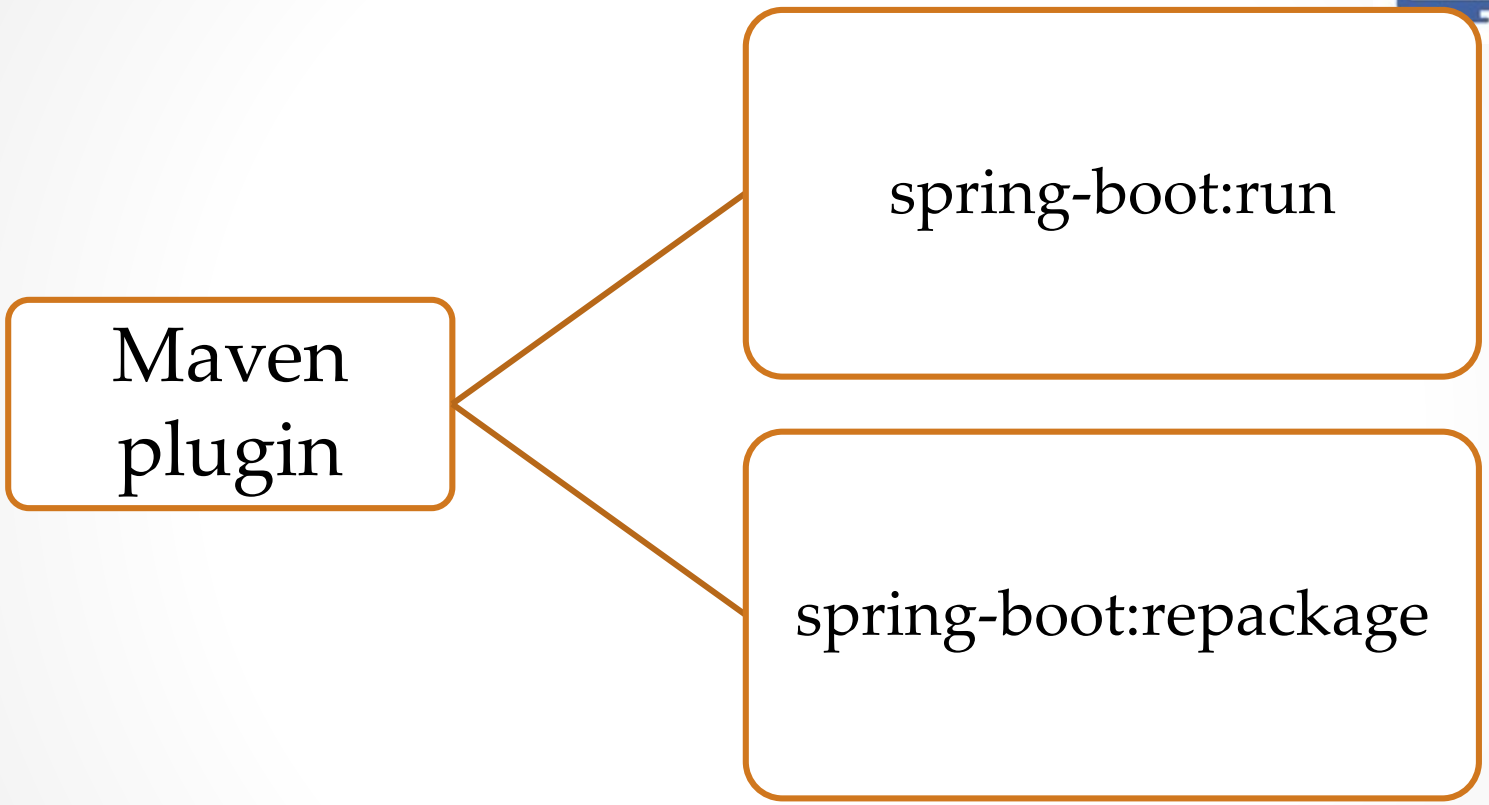
start.spring.io



- Sergey Morenets, 2017

Spring Boot. Maven plugin


```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>${spring.boot.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>repackage</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```


Maven plugin
spring-boot:run
spring-boot:repackage

Spring Boot. Dev tools

- ✓ Automatic restart when file(s) on a classpath changes
- ✓ **LiveReload** server support
- ✓ Remote application support
- ✓ Dev customization by default

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <version>${spring.boot.version}</version>
    <optional>true</optional>
</dependency>
```

Task 1. Spring Boot and REST services

1. Create Spring Boot project using **your IDE**
2. Create Spring Boot project using http://start.spring.io and open it in IDE
3. **Review** project configuration/contents
4. Write simplest REST service(GET and POST methods)

What is monolith application?

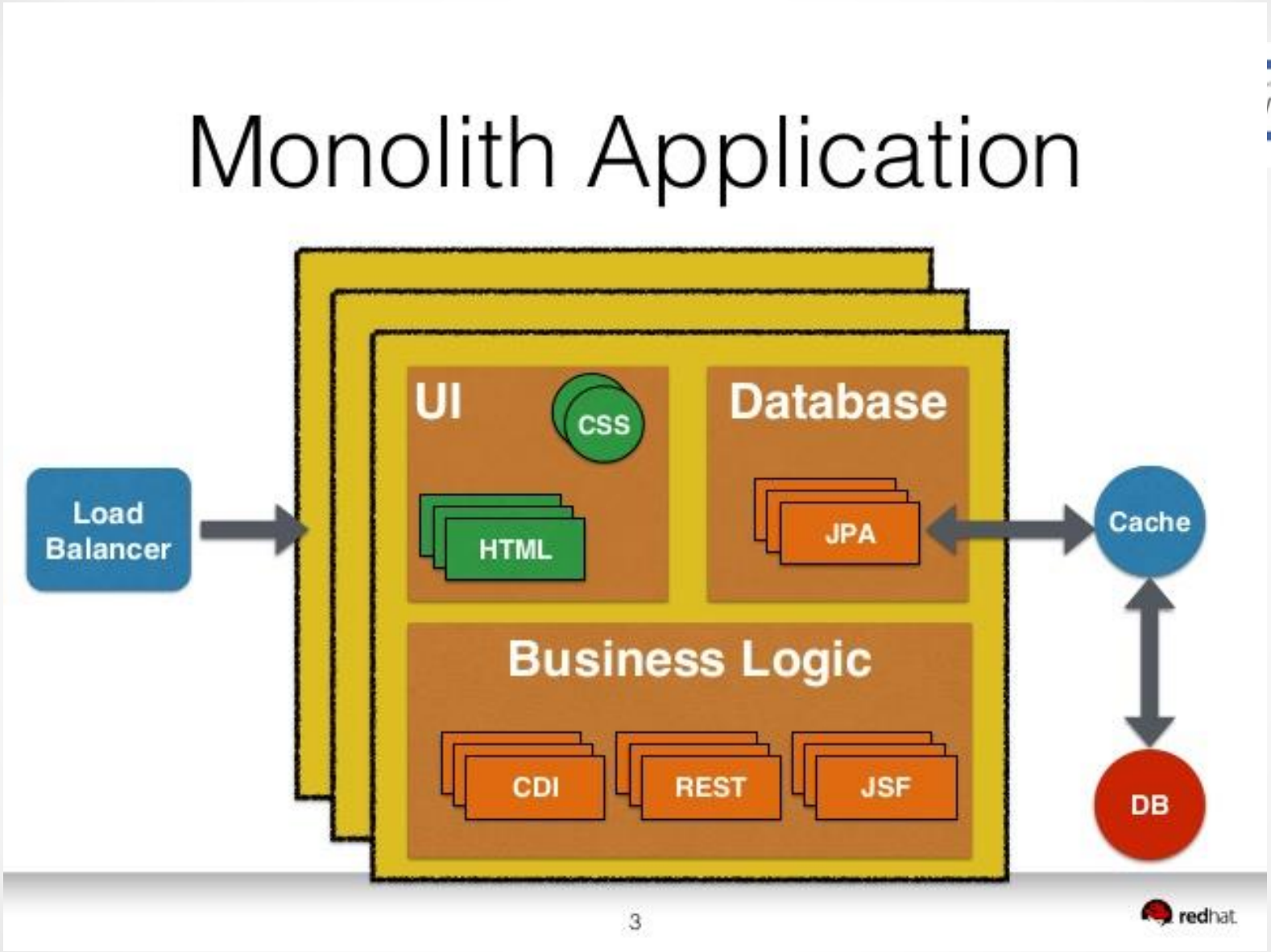
- Sergey Morenets, 2017

Monolith application

- ✓ Single **deployment** unit(WAR, EAR)
- ✓ Single codebase
- ✓ No restrictions on the **project size**
- ✓ Single **database** (RDBMS)
- ✓ Single **language**
- ✓ Long development **iterations**
- ✓ Fixed technology **stack**(JEE, Spring, Hibernate)
- ✓ **ACID** principle
- ✓ One or few **teams**

Monolith application

- ✓ **Tight coupling** between modules
- ✓ **Failures** could affect the whole application
- ✓ **Good** for small/average applications



- Sergey Morenets, 2017

Issues

- Sergey Morenets, 2017

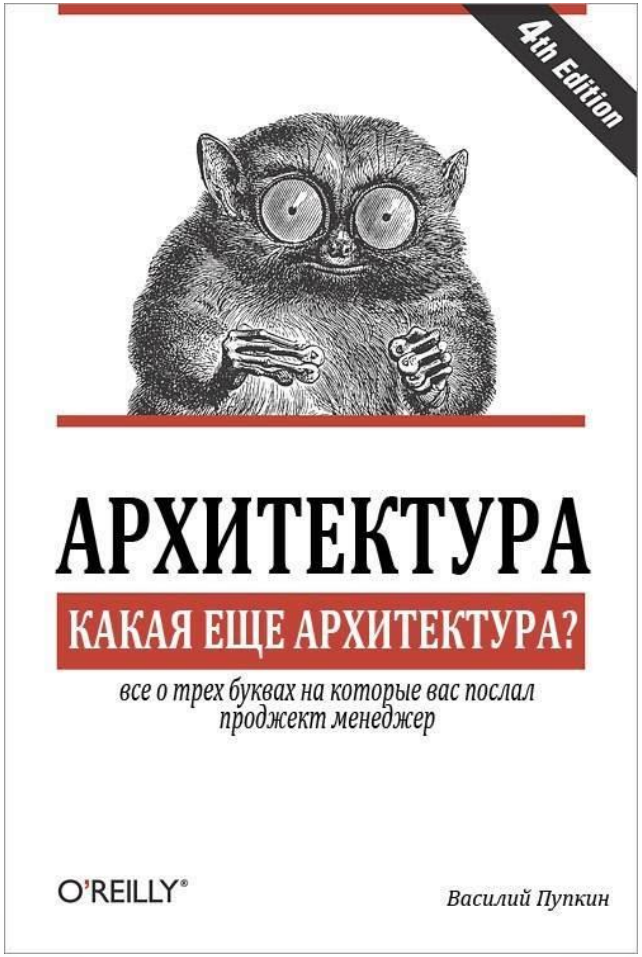
UI

- Sergey Morenets, 2017

Issues

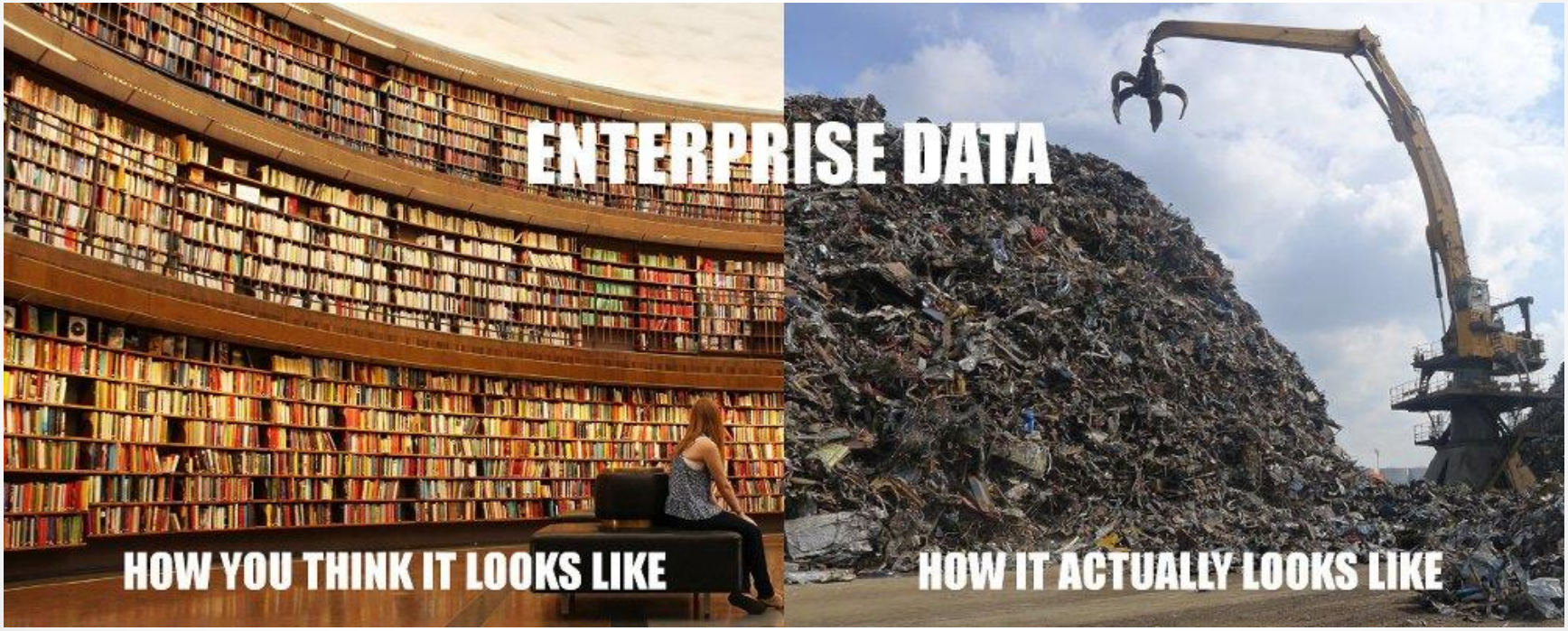
- ✓ Hard to maintain
- ✓ Hard to add new features (**fast**)
- ✓ Hard to scale (specific components)
- ✓ Hard to deploy
- ✓ Slower to start
- ✓ Slower to work in IDE
- ✓ Cannot deploy single module
- ✓ Cannot learn the whole project

Issues



- Sergey Morenets, 2017

Issues



- Sergey Morenets, 2017

Issues

- Sergey Morenets, 2017

Task 2. Monolith application

1. Import **monolith** project into your IDE
2. Review project functionality.
3. Update **BookController** class and add necessary **Spring** annotations.
4. Run application as **Spring Boot** project and observe its behavior.
5. Identify issues related by the monolith architecture and possible solutions for them.

What is micro-service?

- Sergey Morenets, 2017

What is micro-service?

- ✓ **100** lines of code
- ✓ **1 week** of coding
- ✓ **1 day** of documenting
- ✓ **Single** package
- ✓ Application packaged into **container**
- ✓ Single **framework/language**
- ✓ Work for **one man/team**
- ✓ **Single** functionality

Micro-service

- ✓ Loosely coupled service oriented architecture with bounded contexts

- Sergey Morenets, 2017

Micro-service

- ✓ Small autonomous service

- Sergey Morenets, 2017

Micro-services

- ✓ **Separately** written, deployed, scaled and maintained
- ✓ **Independently** upgraded
- ✓ **Easy** to understand/document
- ✓ Provides **business** features
- ✓ **Fast** deployment
- ✓ Use **cutting-edge** deployment
- ✓ Resolve **resource conflicts**(CPU, memory)
- ✓ Communication via **lightweight** protocols/formats

Micro-services

- ✓ **Smaller** and simpler applications
- ✓ Fewer **dependencies** between components
- ✓ Scale and develop independently
- ✓ Easy to introduce new **technologies**
- ✓ Cost, size and risk of changes reduced
- ✓ Easy to **test** single functionality
- ✓ Easy to introduce **versioning**
- ✓ **Cross-functional** distributed teams
- ✓ Improved security due to multiple data-sources
- ✓ Increased uptime

Micro-services design principles

- ✓ High cohesion (**SOLID**)
- ✓ Autonomous (from other services)
- ✓ Business-domain centric (business function)
- ✓ Resilience (respond to **failures**)
- ✓ Observable (system **health**, monitoring, logging)
- ✓ Automated (**reduce** time to setup environment & test)

Resilience

- ✓ Failure of downstream services
- ✓ Default functionality on failure detection
- ✓ Fail fast
- ✓ Use **timeouts** (thresholds)
- ✓ Network latency & bandwidth
- ✓ Monitoring

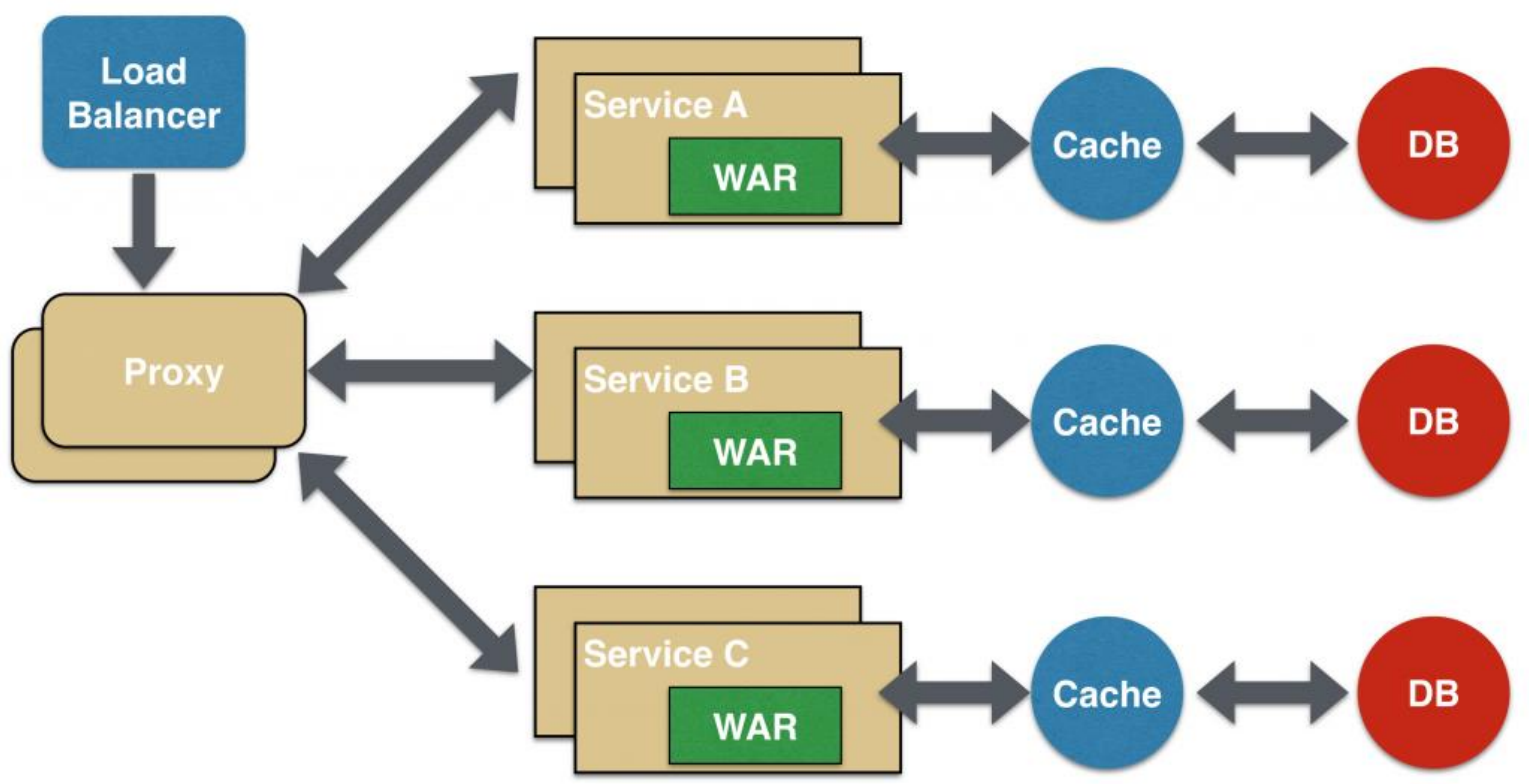
Micro-services

- ✓ No enterprise data model
- ✓ No transactions
- ✓ Micro-services don’t resist changes in other services

Drawbacks

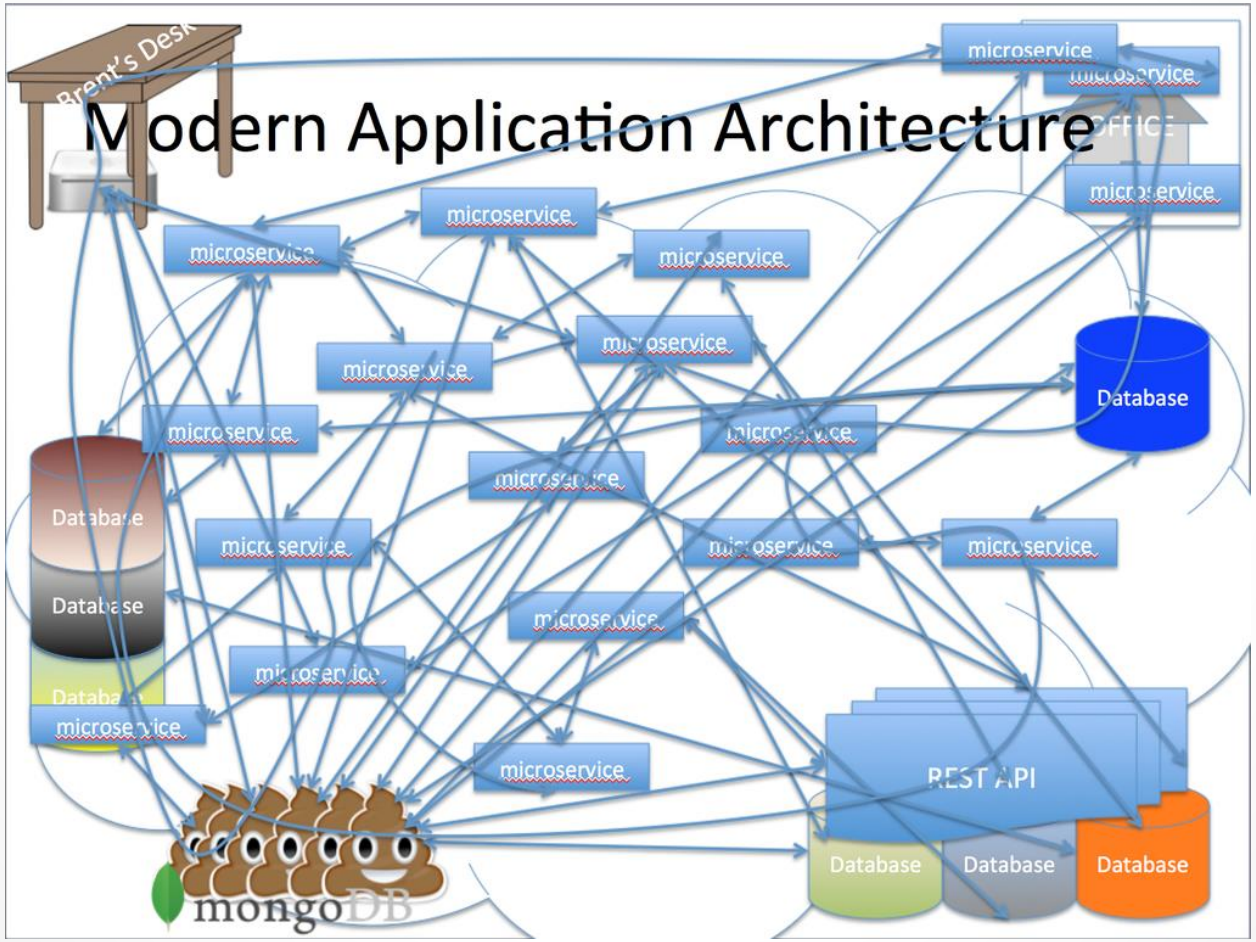
- ✓ Higher level of complexity
- ✓ Transaction management
- ✓ Testing of distributed application
- ✓ Deployment and management
- ✓ Cost of remote calls

Drawbacks



- Sergey Morenets, 2017

Drawbacks



- Sergey Morenets, 2017

Challenges

- ✓ Services unavailability
- ✓ Advanced monitoring
- ✓ Cost of remote calls
- ✓ Eventual consistency (instead of ACID)
- ✓ Single feature is moved into few services
- ✓ Version management
- ✓ Dependency management
- ✓ Multiple data sources(databases)

SOA vs micro-services

- Sergey Morenets, 2017

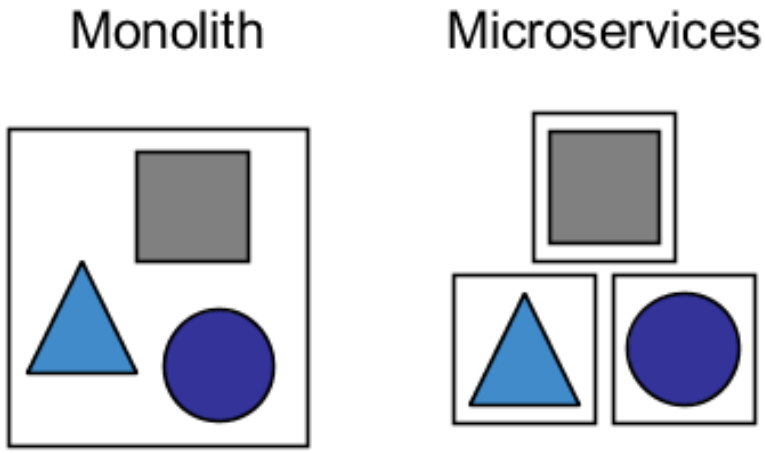
Transition

- Sergey Morenets, 2017

Transition

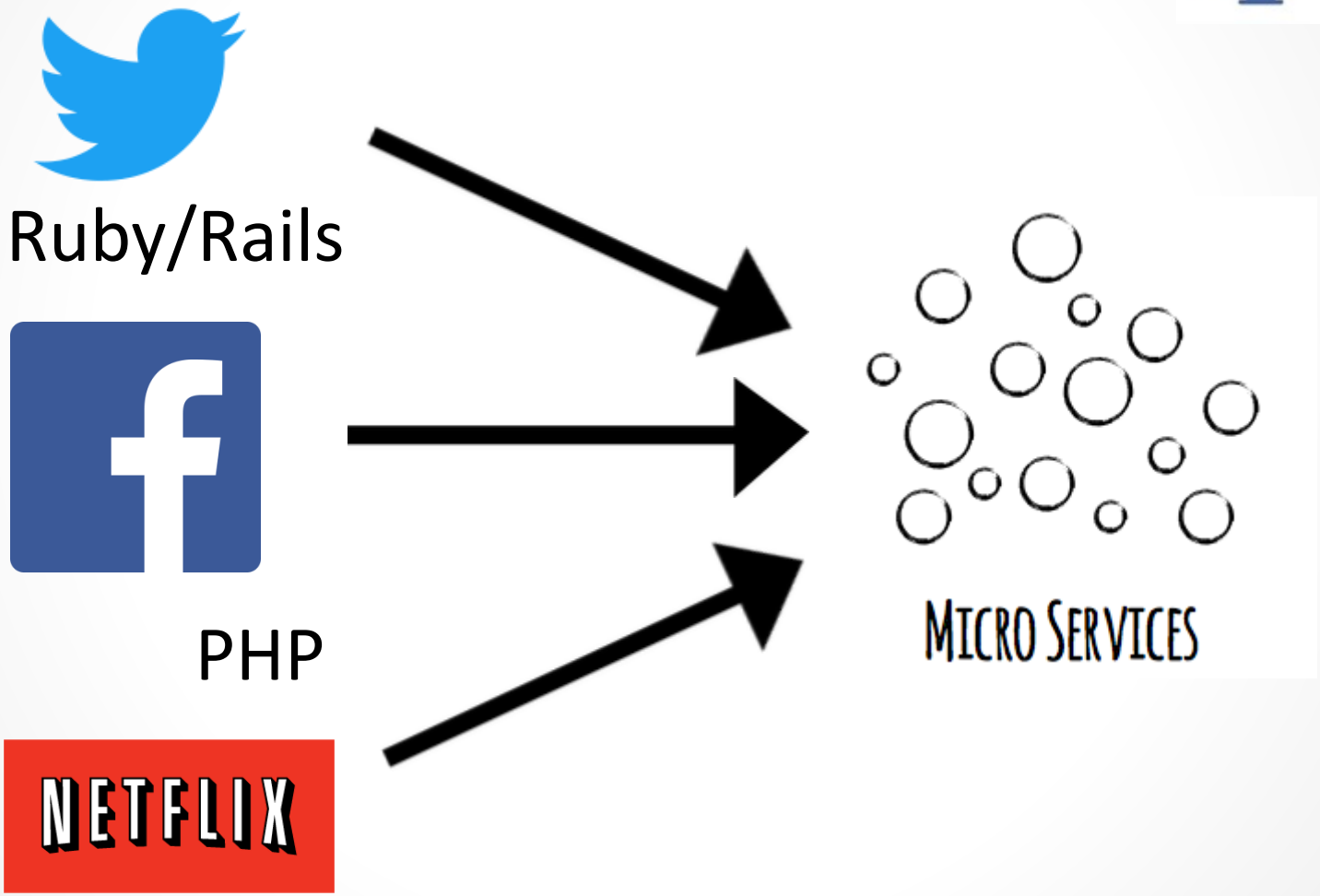
- Sergey Morenets, 2017

Transition

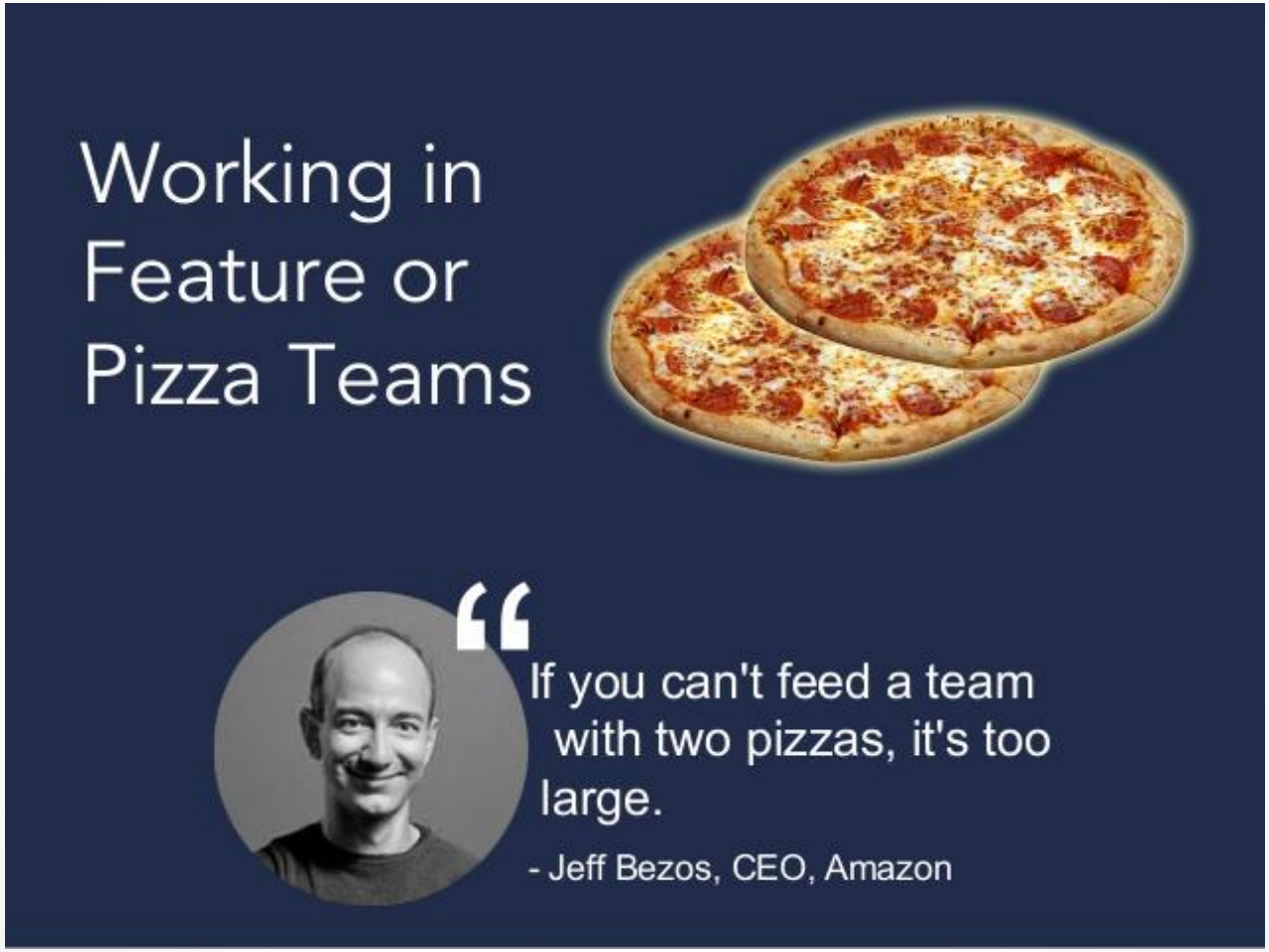


- Sergey Morenets, 2017

Transition



Transition



- Sergey Morenets, 2017

Transition

- ✓ Decomposition of an application
- ✓ Single responsibility principle. Single micro-service = single business feature
- ✓ Based on business functionality
- ✓ Bounded context

Polyglot persistence

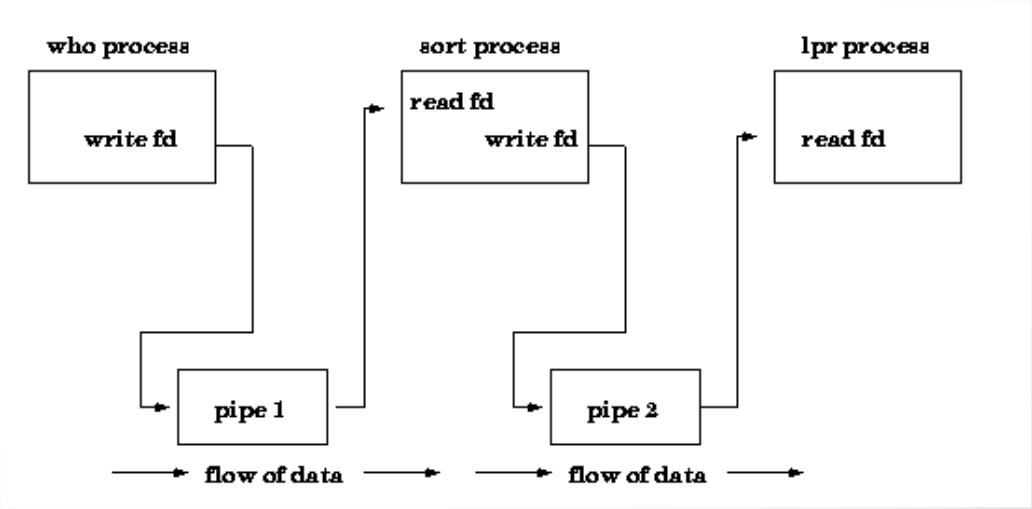
- Sergey Morenets, 2017

Transition

- ✓ Split domain model/**business logic**
- ✓ Split data model/persistence layer/**DB**
- ✓ Split/introduce **technologies**

Partitioning strategies

- By entity(Customer – Product – Order)
- By use case(Book – Buy – Search)
- Single Responsibility Principle
- Bounded context
- **Unix utilities**



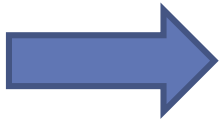
Enterprise model. Client

- ✓ Identifier
- ✓ Name
- ✓ Address
- ✓ Email/Phone
- ✓ Credit card number
- ✓ Expiration date
- ✓ Discount
- ✓ Purchases

Splitting data model

Client. **Purchase service**

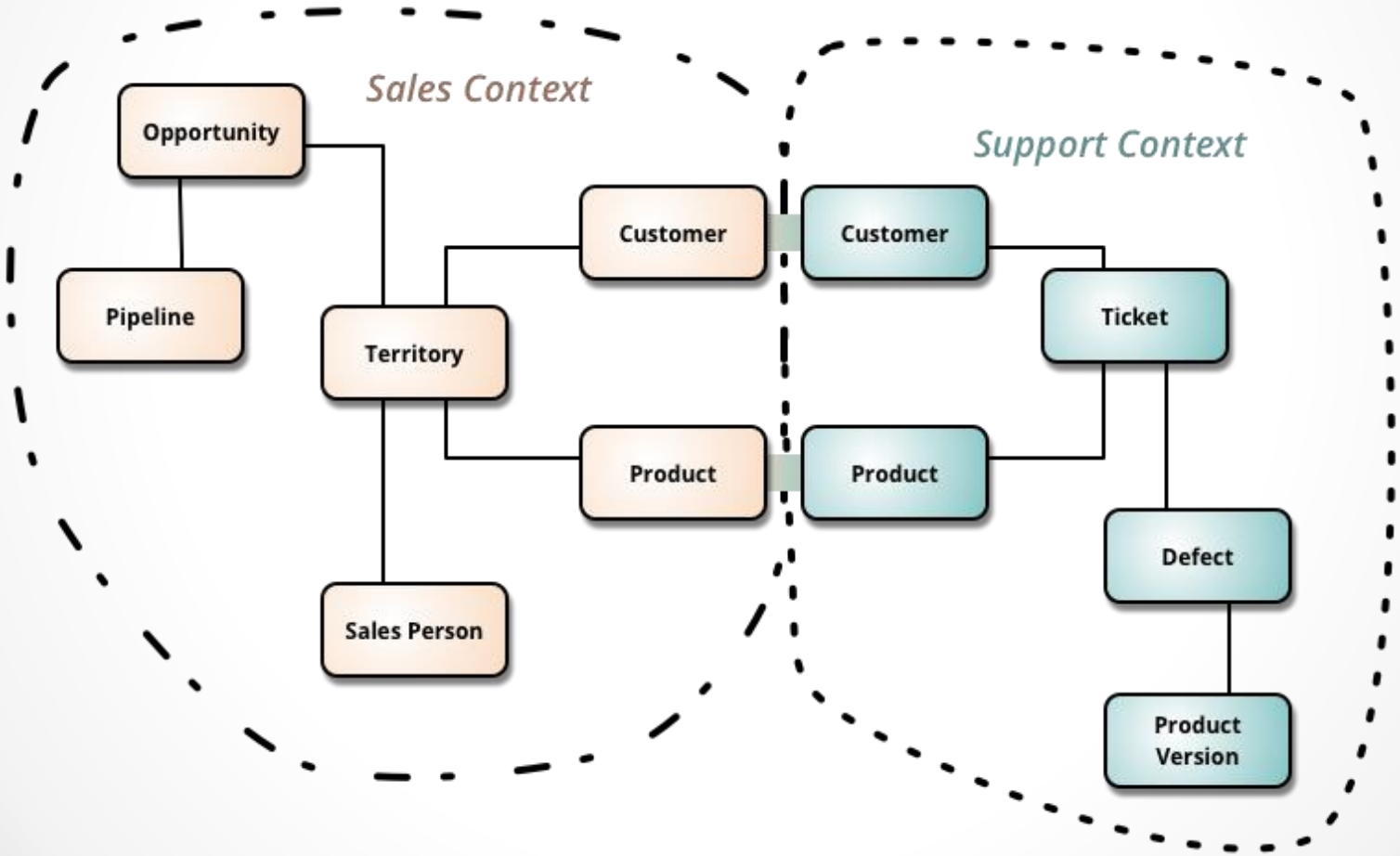
- ✓ Identifier
- ✓ Credit card number
- ✓ Expiration date
- ✓ Discount
- ✓ Purchases



Client. **Delivery service**

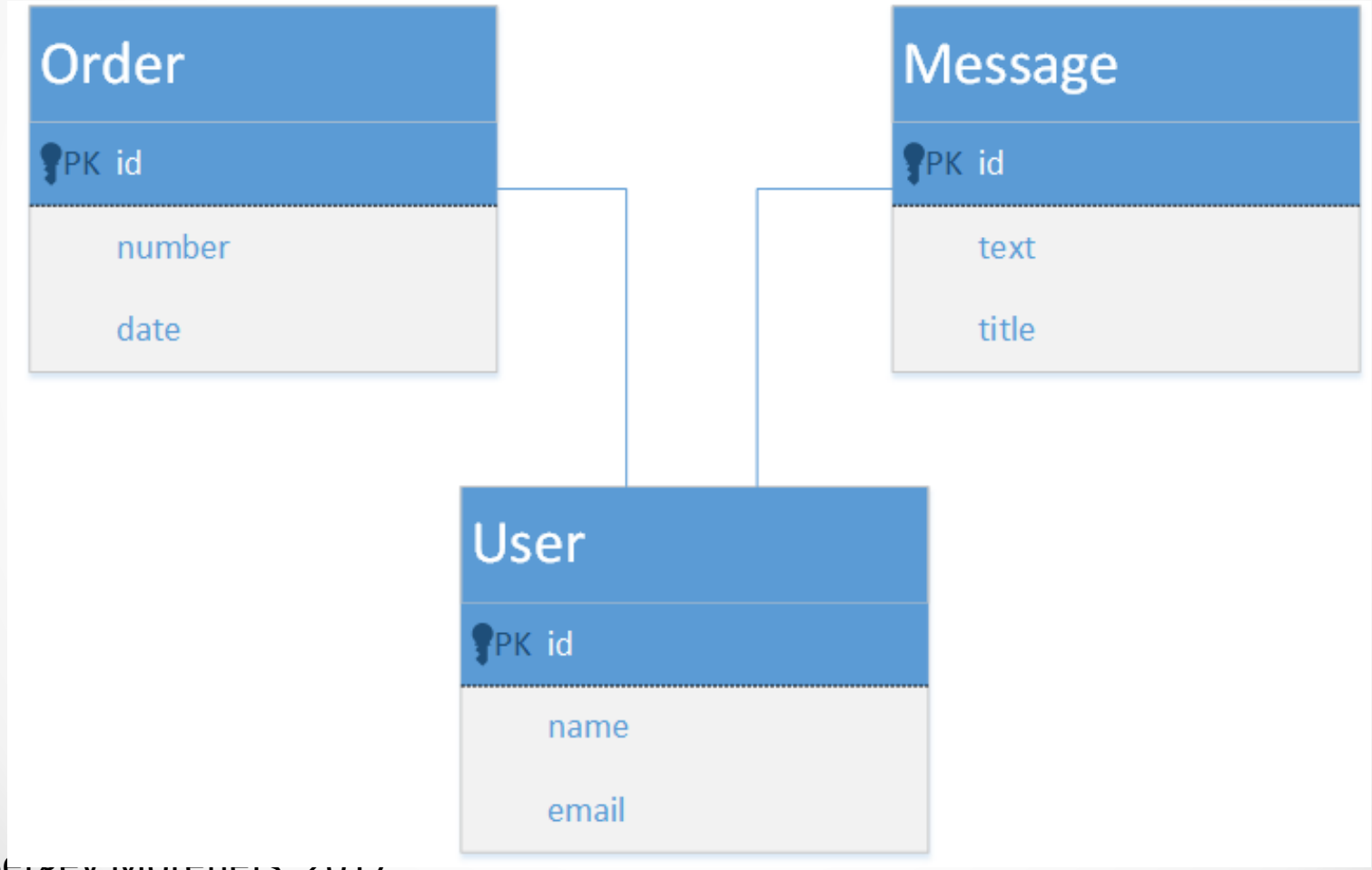
- ✓ Identifier
- ✓ Name
- ✓ Address
- ✓ Email/Phone

Bounded context



- Sergey Morenets, 2017

Splitting data model



Task 3. Splitting monolith

1. Review monolith application again. Try to convert it to micro-service architecture gradually.
2. Extract **functionality** that goes to the separate projects
3. Split **domain model**
4. Split **DAO layer(repository)**

MongoDB

- Open-source **NoSQL** document database
- Uses **JSON** documents(in binary-encoded format) with schemas
- Indexing
- Replication
- Load balancing through sharding
- Aggregation (MapReduce)
- Server-side JavaScript execution
- Current version 3.4.4

MongoDB. JSON document


```
{
    '_id' : 1,
    'name' : { 'first' : 'John', 'last' : 'Backus' },
    'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
    'awards' : [
        {
            'award' : 'W.W. McDowell Award',
            'year' : 1967,
            'by' : 'IEEE Computer Society'
        }, {
            'award' : 'Draper Prize',
            'year' : 1993,
            'by' : 'National Academy of Engineering'
        }
    ]
}
```

- Sergey Morenets, 2017

MongoDB. Shell methods

Method	Description
db.collection.find({ “name” : “test” })	Executes query to return collection
db.collection.findOne()	Returns single document
db.collection.drop()	Remove collection
db.collection.insert({ “name” : “Microservices”})	Inserts new document
db.collection.update({ name : “Phone”}, { name: “PC” })	Modifies document in the collection
db.collection.count()	Calculates number of the documents in the collection
db.collection.remove()	Clears collection
db.collection.renameCollection()	Changes name of the collection

Task 4. Install & configure MongoDB

1. Download and install **MongoDB** as a service(version 3.4.x and later is recommended).
2. Run **mongo** utility from c:/Mongo/bin folder.
3. Type command: **use sample**
4. Type command : *db.books.insert({"id" : "1", "title" : "Microservices" });*
5. Type command: *db.books.find();* What fields are returned from the database?

Spring Data Modules

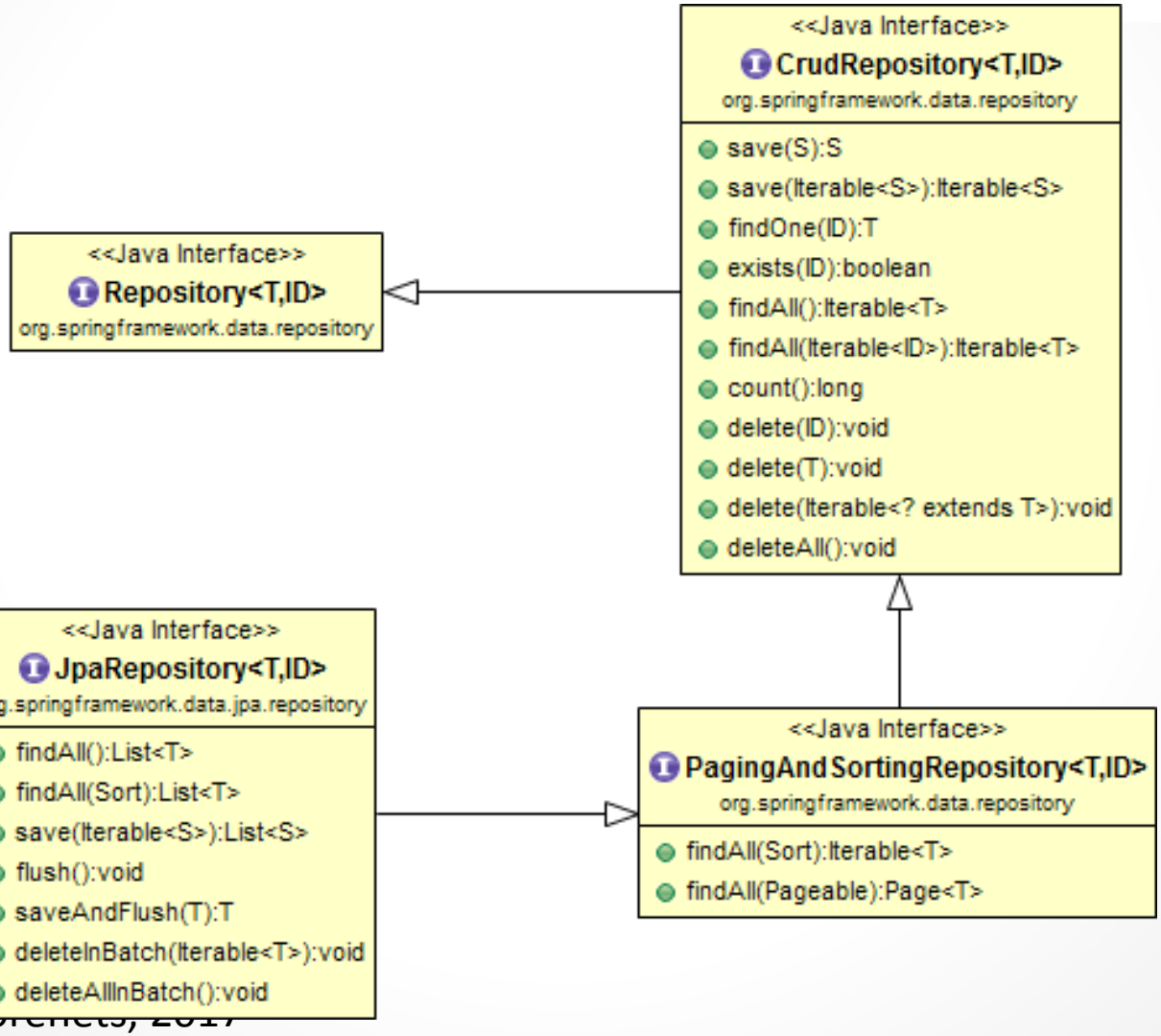
Spring Data MongoDB

- ✓ Spring Data sub-project
- ✓ MongoTemplate for operations
- ✓ Java-based Query/Criteria
- ✓ Automatic repository implementation
- ✓ Cross-store persistence
- ✓ Map-Reduce integration
- ✓ Uses MongoDB Java driver

Spring Data MongoDB. Maven


```
<dependencies>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-mongodb</artifactId>
        <version>1.10.3.RELEASE</version>
    </dependency>
</dependencies>
```

Spring Data. Hierarchy



Spring Data MongoDB. Repositories


```
public class Product {
    private int id;

    private String name;

    private double price;
```

```
public interface ProductRepository extends
    MongoRepository<Product, Integer>{
}
```

Spring Data MongoDB. Repositories


```java
@RestController
@RequestMapping("product")
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @GetMapping
    public List<Product> findProducts() {
        return productRepository.findAll();
    }
```

- Sergey Morenets, 2017

Spring Data MongoDB. Data Model


```
@Document(collection="items")
public class Product {
    @Id
    private int id;

    private String name;

    private double price;
```

CrudRepository

Method	Description
save	Saves given entities/entity
findOne	Retrieves an entity by its id
findAll	Returns all instances of the type
count	Returns the number of entities available
delete	Deletes the entity with the given id
exists	Returns whether an entity with the given id exists
deleteAll	Deletes all entities managed by the repository

Spring Data MongoDB. Properties


```
spring.data.mongodb.database=warehouse
spring.data.mongodb.host=localhost
spring.data.mongodb.password=pwd
spring.data.mongodb.port=27017
spring.data.mongodb.username=user
```

src/main/properties/application.properties

Task 5. Spring Data MongoDB

1. Add **Spring Data MongoDB** dependency to your project:
2. Add **application.properties** to **src/main/properties** folder
3. Create new **HitRepository** interface that extends **MongoReposiory** interface.
4. Create new **HitController** class and implement **findAll/save** REST services. Auto-wire **HitRepository** interface.
5. Run application and try to save **Hit** objects. Open **Mongo client** and verify that documents were saved in the database.

Questions

- ✓ Service discovery
- ✓ Service lookup
- ✓ Load balancing
- ✓ Auto-scaling
- ✓ Protocols and data formats
- ✓ Security
- ✓ API management
- ✓ Testing
- ✓ Data access

Questions

- ✓ How to deploy
- ✓ How service communicate with each other
- ✓ How client and service communicate
- ✓ How to split monolith into services
- ✓ How to handle failures

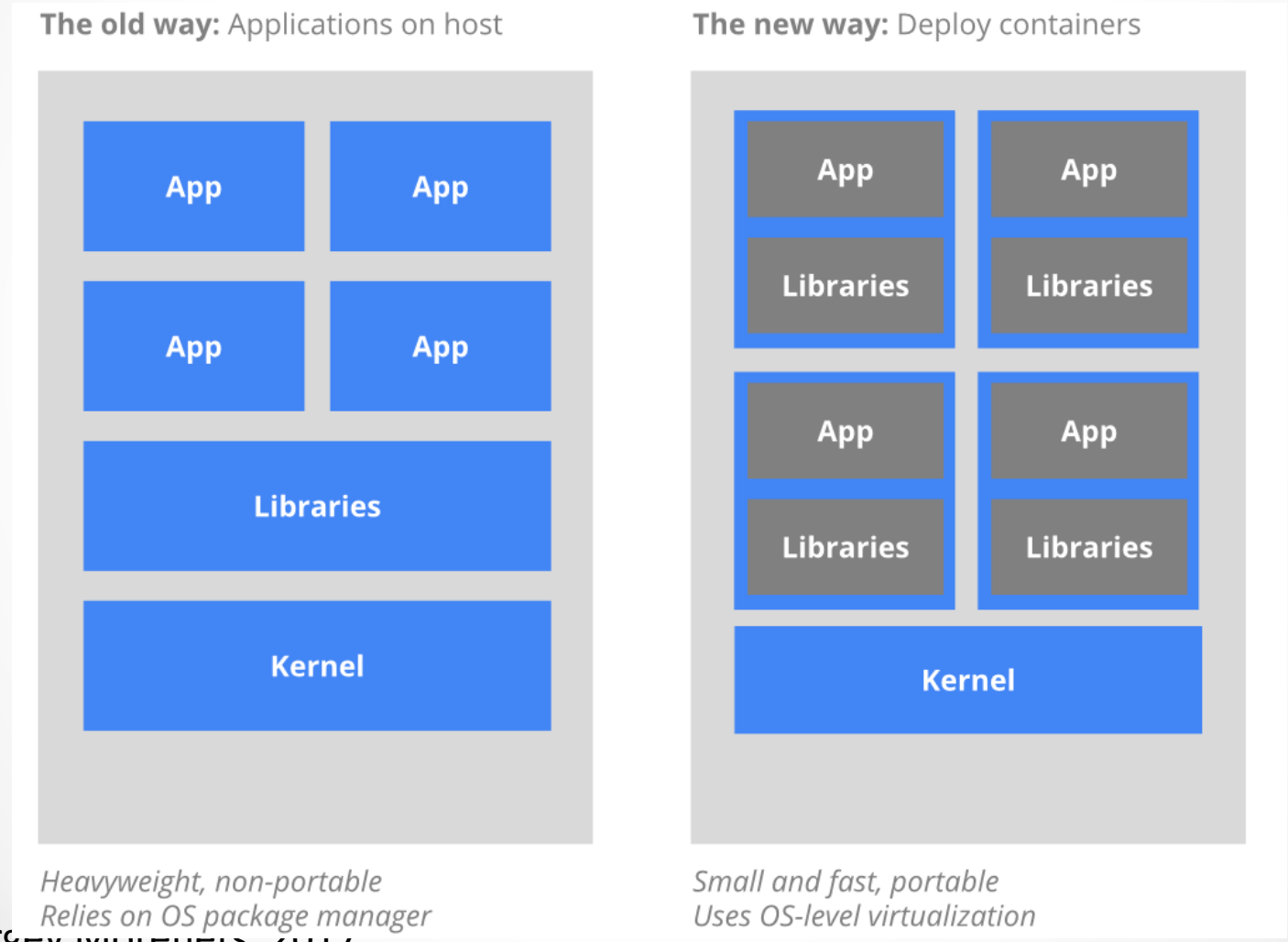
Communication

- ✓ TCP, HTTP, Messaging
- ✓ XML, JSON, BSON, Protobuf

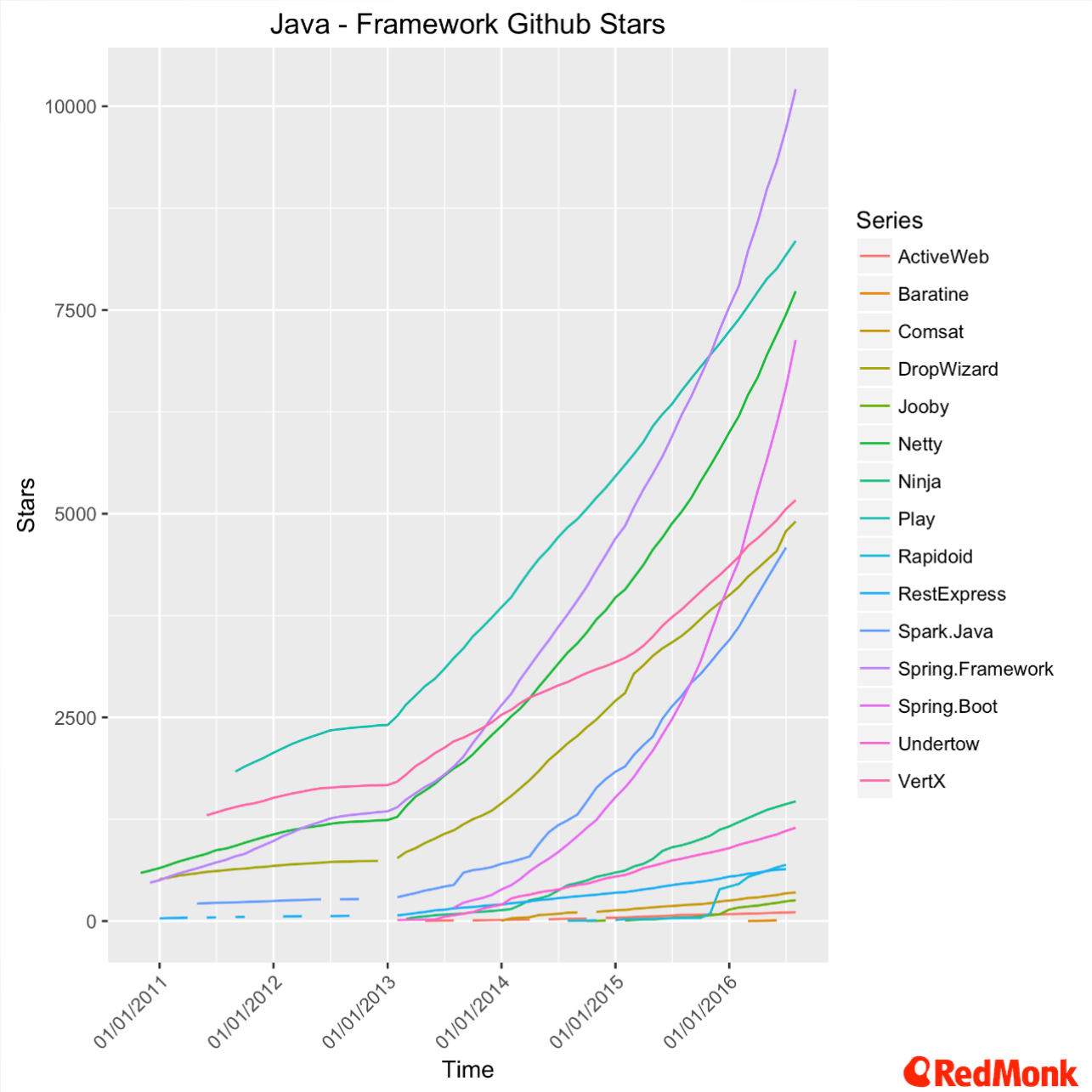
Infrastructure

- Sergey Morenets, 2017

Infrastructure



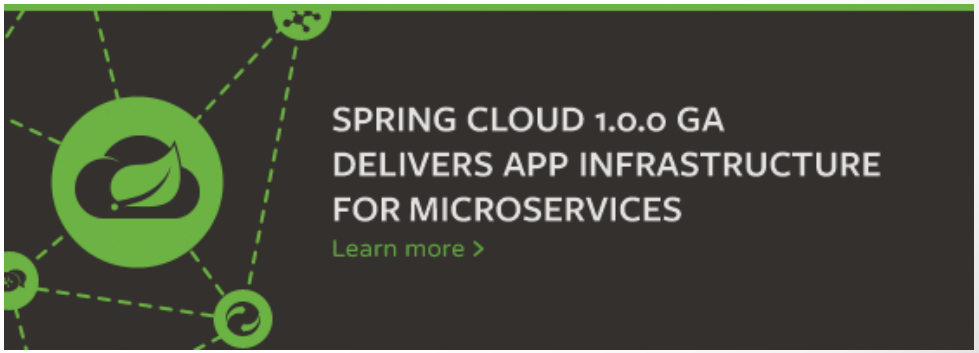
- Sergey Morenets, 2017



- Sergey Morenets, 2017

Spring Cloud

- ✓ Common distributed patterns
- ✓ Centralized configuration management
- ✓ Service registration and discovery
- ✓ Load balancing
- ✓ Service-to-service calls
- ✓ Routing
- ✓ Based on Spring Boot



- Sergey Morenets, 2017

Spring Cloud

- ✓ Spring Cloud Config
- ✓ Spring Cloud Netflix
- ✓ Spring Cloud Bus

Spring Cloud

Operations Component	Netflix, Spring, ELK
Service Discovery server	Netflix Eureka
Dynamic Routing and Load Balancer	Netflix Ribbon
Circuit Breaker	Netflix Hystrix
Monitoring	Netflix Hystrix dashboard and Turbine
Edge Server	Netflix Zuul
Central Configuration server	Spring Cloud Config Server
OAuth 2.0 protected API's	Spring Cloud + Spring Security OAuth2
Centralised log analyses	Logstash, Elasticsearch, Kibana (ELK)

Application configuration

- ✓ Resource management
- ✓ Resource settings(DB, network) storage

Configuration options

- ✓ Application-specific(requires restart)
- ✓ File system(unsupported for cloud)
- ✓ Environment variables
- ✓ Cloud-specific

Solution

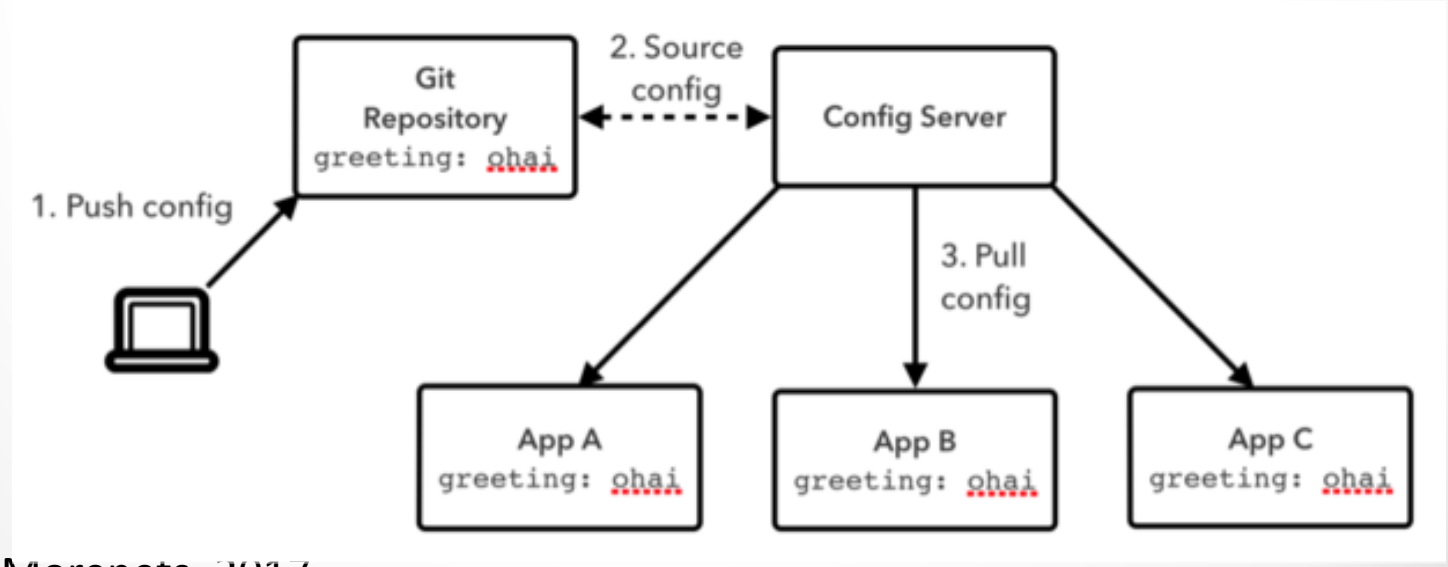
- ✓ Cloud-independent
- ✓ Centralized
- ✓ Doesn’t require app restart
- ✓ Self-management

Solution

- ✓ Spring Cloud Config
- ✓ Spring Cloud Bus
- ✓ Spring Cloud Netflix Eureka

Spring Cloud Config

- ✓ Standalone server that stores configuration settings
- ✓ Clients communicate with server via HTTP and obtain global configuration settings



Spring Cloud Config


```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-parent</artifactId>
            <version>${spring.cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
</dependencies>
```

Spring Cloud Config


```
@SpringBootApplication
@EnableConfigServer
public class MainApplication {
    public static void main(String[] args) {
        SpringApplication.run(MainApplication.class, args);
    }
}
```

- Sergey Morenets, 2017

Repository storage

- ✓ Git
- ✓ Local files
- ✓ User-defined

Git storage


```
spring.cloud.config.server.git.uri=https://github.com/
spring.cloud.config.server.git.search-paths=config
```

application.properties

```
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/
          searchPaths: config
```

application.yml

YAML


```yaml
---
key: value
map:
     key1: "foo:bar"
     key2: value2
list:
   - element1
   - element2
# This is a comment
listOfMaps:
   - key1: value1a
     key2: value1b
   - key1: value2a
     key2: value2b
---
```

```
some
other
listing
```

- Sergey Morenets, 2017

Task 6. Configuration server

1. Import **microservices** project into your IDE
2. Update ConfigServerApplication class
3. Update **application.properties** file.
4. Make executable jar file using command: **mvn install spring-boot:repackage** and run it as ordinary java file.

Client configuration


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Client configuration

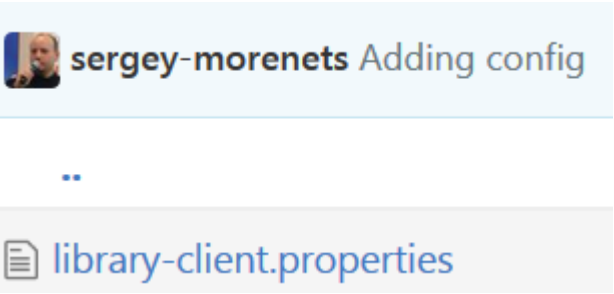

```
spring.application.name=library-client
spring.cloud.config.uri=http://localhost:8000
```

bootstrap.properties

```
spring:
  application:
    name: library-client
  cloud:
    config:
      uri: http://localhost:8000
```

bootstrap.yml

Client configuration




```
company1 = Crunchify
company2 = Google
Crunchify_Address = NYC,US
Google_Address = Mountain View,CA,US
```

Task 7. Client configuration

1. Add new dependency to the book application:
2. Add **bootstrap.properties** to the book application, populate application name (**application**) and cloud configuration URI.
3. Populate **libraryName** field in BookController from Spring configuration (it should be taken from GitHub repository).
4. Make executable jar file using command: **mvn install spring-boot:repackage** and run it as ordinary java file.

Local files


```
#spring.cloud.config.server.git.uri=https://github.com/
#spring.cloud.config.server.git.search-paths=config

spring.profiles.active=native
```

application.properties

```
company1 = Crunchify
company2 = Google
Crunchify_Address = NYC,US
Google_Address = Mountain View,CA,US
```

{spring.application.name}.properties

Profiles


```
spring.application.name=library-client
spring.cloud.config.uri=http://localhost:8000
spring.profiles.active=dev
```

bootstrap.properties

```
company1 = Crunchify
company2 = Google
Crunchify_Address = NYC,US
Google_Address = Mountain View,CA,US
```

{spring.application.name}-dev.properties

Task 8. Local configuration

1. Update **application.properties** in the config-server application to switch to local files storage.
2. Copy properties file from GitHub repository to the **src/main/properties** folder and rename it to **library-client.properties**. Rename book application name:
3. Restart **config-server** and book application
4. Copy properties file into new file with "**dev**" suffix. Update **bootstrap.properties** in the client application and set **Spring** profile to "dev"

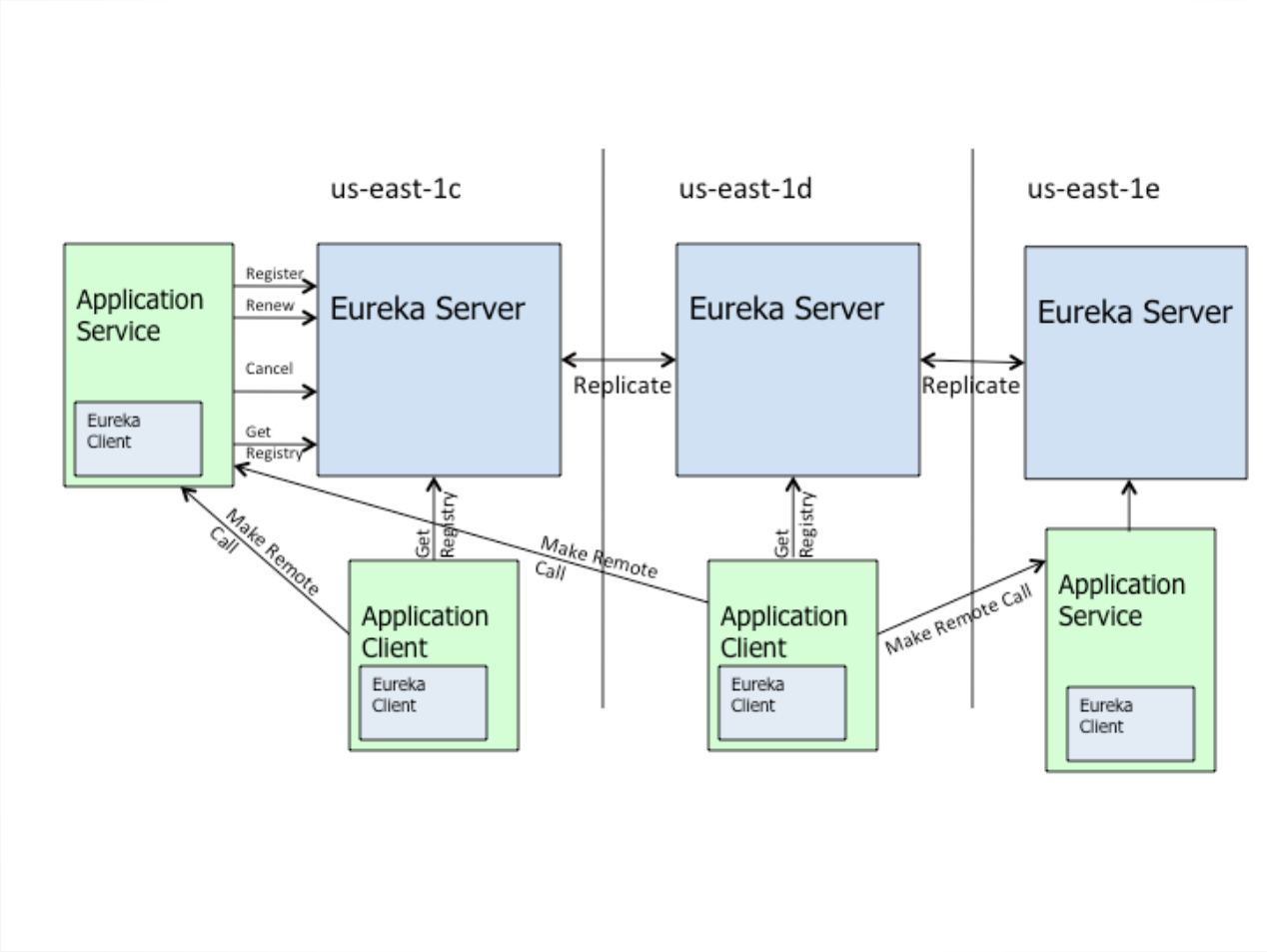
Service discovery

- ✓ Client should register itself and obtain information about other clients
- ✓ Single lookup service
- ✓ Solutions: Eureka, Consul, Zookeeper

Eureka

- Part of Spring Cloud Netflix
- Service discovery client/server
- Has two component: **Eureka Client** and **Eureka Server**
- **Application client** uses Eureka Client to communicate with Eureka Server
- Clients registers with Eureka, provides metadata and sends heart-beats
- High availability achieved by running multiple servers in different zones/regions sharing their state
- Client registrations are stored in memory

Eureka availability zones



- Sergey Morenets, 2017

Eureka regions&availability zones

Region

Availability zones

```
eureka.us-east-1.availabilityZones=us-east-1c,us-east-1d,us-east-1e

eureka.serviceUrl.us-east-1c=http://ec2-5.amazonaws.com:7001/eureka/v2/
eureka.serviceUrl.us-east-1d=http://ec2-2.amazonaws.com:7001/eureka/v2/
eureka.serviceUrl.us-east-1e=http://ec2-0.amazonaws.com:7001/eureka/v2/
```

Eureka server. eureka-client.properties

Eureka server configuration


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class,
                args);
    }
}
```

Eureka server configuration


```
server.port=8090
eureka.client.serviceUrl.defaultZone=http://localhost:8090/eureka
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Eureka server configuration

Setting	Description	Default
eureka.client.eurekaServiceUrlPollIntervalSeconds	Time to check for changes in client registry URL	5 min
eureka.instance.leaseRenewalIntervalInSeconds	Interval time to send heartbeats (from client to server)	30 sec
eureka.instance.leaseExpirationDurationInSeconds	Time to remove client from instance after not receiving heartbeat	90 sec
eureka.client.instanceInfoReplicationIntervalSeconds	Interval time to send instance information from client to server	30 sec
eureka.client.registryFetchIntervalSeconds	Interval time to query Eureka server registry	30 sec

Eureka server configuration

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP
THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

```yaml
eureka:
  server:
    enableSelfPreservation: false
```

Eureka server. application.yml

Eureka server endpoints

Endpoint	Description
/eureka/apps	Returns all the registered instances
/eureka/apps/<app.id>	Returns information about all <application.name> instances
/eureka/apps/<app.id>/<instanceId>	Returns information about specific instance
/eureka/instances/<instanceId>	Returns information about specific instance
PUT /eureka/apps/<app.id>/<instanceId>	Send client heartbeat
POST /eureka/apps/<app.id>	Register new application instance

Eureka client configuration


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

```
@EnableDiscoveryClient
@SpringBootApplication
public class ClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class,
                args);
    }
}
```

```
eureka.client.serviceUrl.defaultZone=http://localhost:8090/eureka/
```

Eureka client configuration


```
eureka:
  client:
    registerWithEureka: true
    fetchRegistry: true
    useDnsForFetchingServiceUrls: true
    eureka-server-d-n-s-name: eureka.local
    eurekaServerPort: 8761
    eureka-server-u-r-l-context: eureka
    region: us-east-1
    availabilityZones:
      us-east-1: us-east-1a,us-east-1b
    preferSameZoneEureka: true
```

Eureka client configuration


```
@Autowired
private DiscoveryClient discoveryClient;

public List<ServiceInstance> getInstances(
        String serviceId) {
    return discoveryClient.getInstances(serviceId);
}
```

Configuration

Name	Description
EurekaServerConfig	
EurekaClientConfig	
EurekaInstanceConfig	

Task 9. Eureka server

1. Add @EnableEurekaServer annotation to the EurekaApplication class.
2. Populate **application.properties** file for eureka project.
3. Populate **bootstrap.properties** with eureka address.
4. Start client project and observe in the eureka/client logs that client registers itself on the Eureka server.

Eureka/Config Server integration


```
spring.application.name=config-service
```

Configuration server. application.properties

```
spring.cloud.config.discovery.serviceId=config-service
spring.cloud.config.discovery.enabled=true
eureka.client.serviceUrl.defaultZone=http://localhost:8090/

spring.cloud.config.retry.max-attempts=20
```

Client application. bootstrap.properties

Task 10. Eureka & Config server

1. Add **eureka-server** dependency for the config-server project.
2. Add **@EnableDiscoveryClient** annotation for the **ConfigServerApplication** class.
3. Add properties for config-server project.
4. Update **bootstrap.properties** of the client applications to add new properties:
5. Start **Eureka/config-server/client applications** and make they work properly together.

Eureka server. Peer awarness

- ✓ Every server know about other peers and replicate data with them
- ✓ Client connects to first server in the list and send heartbeats
- ✓ If the first server becomes unavailable all clients migrate to the next server
- ✓ If the first server becomes available all clients migrate back to the first server
- ✓ If neighboring peer become offline the peer goes into **preservation mode** until nearby peer gets online

Eureka server. Peer awarness


```yaml
spring:
  profiles: peer1
server:
  port: 8090
eureka:
  instance:
    hostname: peer1
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://peer1:8090/eureka/,http://peer2:8091/eureka/
```

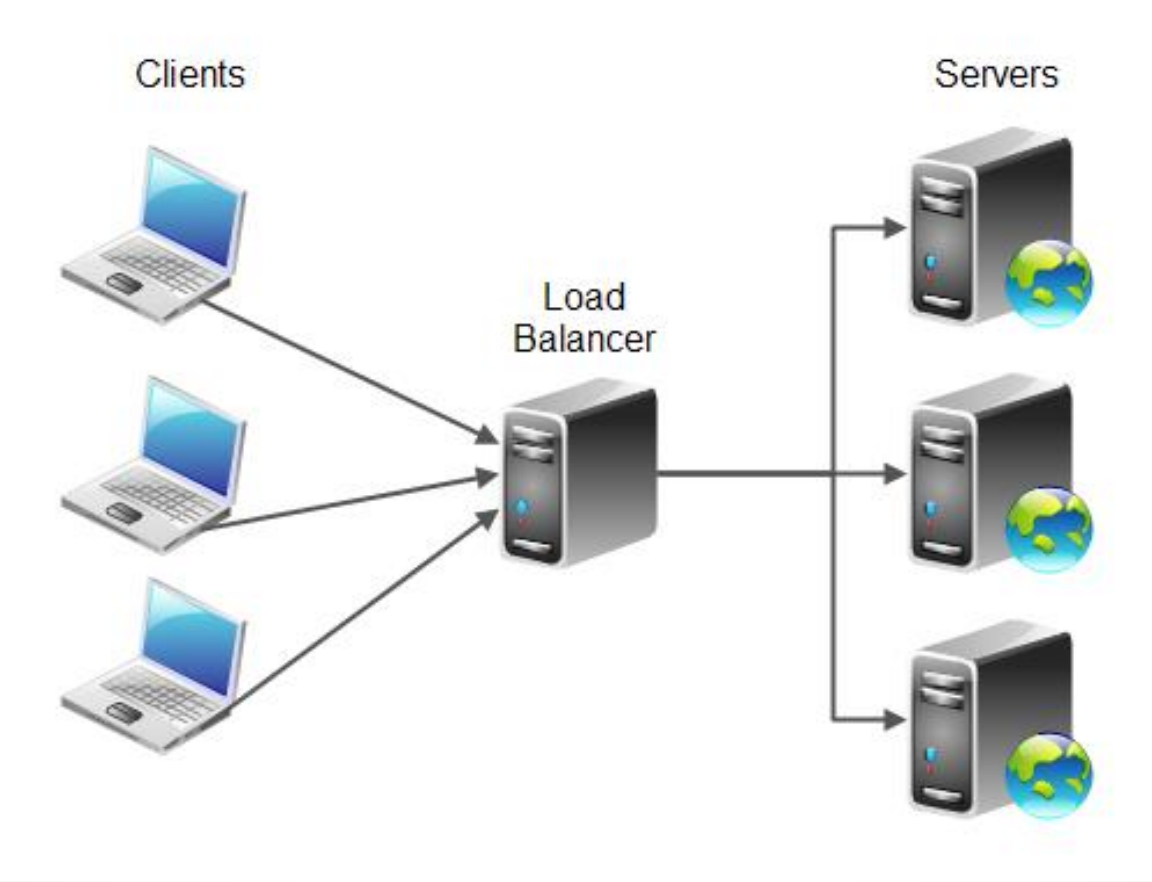
Task 11. Eureka peer-to-peer

1. Update **hosts** file
2. Update **application.yml** file for **eureka-server** project and include new configuration for two peers.
3. Start first **Eureka** peer.
4. Open Web UI http://localhost:8090/ and observer registered clients.
5. Start second **Eureka** peer.
6. Update configuration for one of the client
7. applications:

Load balancing

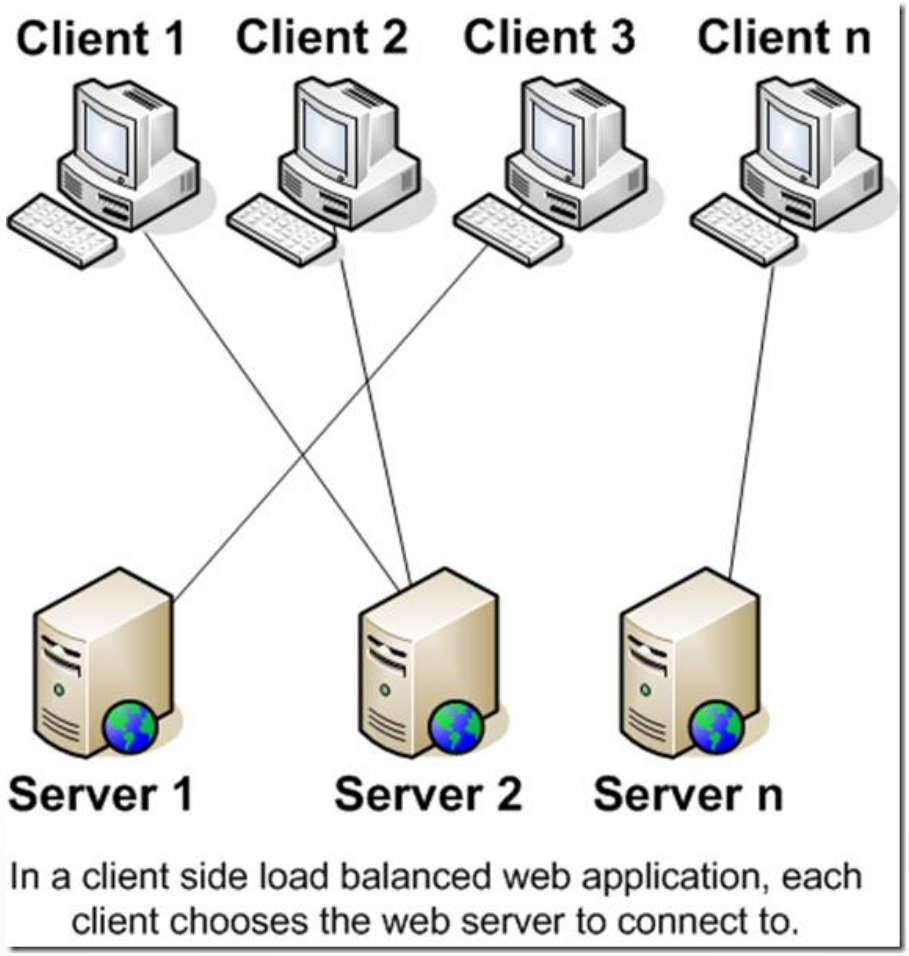
- Servers may go down and become unavailable
- Servers may be heavy loaded
- Servers may reside in the different zones

Load balancing(server)



- Sergey Morenets, 2017

Load balancing(client)



In a client side load balanced web application, each client chooses the web server to connect to.

- Sergey Morenets, 2017

Spring Cloud Netflix Ribbon

- ✓ Client-side load balancer which is grouped into application set by specific name
- ✓ Integrates with Spring Cloud Eureka
- ✓ Caching/batching
- ✓ Built-in failure resilience (skipping non-live servers)
- ✓ Implements Circuit Breaker pattern
- ✓ Supports TCP/UDP/HTTP

Ribbon API

- **Rule**

  Load balancing rule in the application
- **Ping**

  Algorithm to determine server availability at run-time
- **ServerList**

  Can be dynamic or static

Ribbon configuration


```
spring.application.name=hello
```

Server application

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
```

Client application

RestTemplate

High-level abstraction over Apache Http components library

HTTP method	RestTemplate method
DELETE	delete
GET	getForObject, getForEntity
POST	postForObject
PUT	put

RestTemplate


```
public class RestClient {

    private final RestTemplate restTemplate =
            new RestTemplate();

    public List<Book> findBooks() {
        return (List<Book>) restTemplate.getForObject(
                "http://localhost:8080/book", List.class);
    }
```

Ribbon(client)


```
@SpringBootApplication
@RibbonClient(name="hello")
public class ClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class,
                args);
    }

    @LoadBalanced
    @Bean
    RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

Ribbon(client)


```
@Autowired
RestTemplate restTemplate;

public String hi(String name) {
    return restTemplate.getForObject("http://hello/greeting/"
            + name, String.class);
}
```

Ribbon configuration


```
hello:
  ribbon:
    eureka:
      enabled: false
    listOfServers: localhost:8080,localhost:7000
    ServerListRefreshInterval: 3000
```

Disable Eureka

```
hello:
  ribbon:
    MaxAutoRetries: 2
    MaxAutoRetriesNextServer: 2
    ServerListRefreshInterval: 2000
    ConnectTimeout: 5000
    ReadTimeout: 90000
```

Connection settings

Task 12. Ribbon client

1. Adds new method to **HitRepository** that returns number of hits for specific book. Adds new **REST service** that calls this method and returns hit number. Verify service behavior.
2. Add Ribbon dependency to the book application
3. Adds **@RibbonClient** annotation and **RestTemplate bean** for the bootstrap class of the book application.
4. Add **hitCount** field to the book class. Update **GET REST service** in the

Task 13. Ribbon client without Eureka

1. Remove Eureka dependency from book application
2. Start multiple instances of the logging application
3. Update **bootstrap.properties** (or .yml) of book application and include list of available logging servers
4. Start book application and invoke new REST service multiple times. Which logging instances are used now?

Spring Cloud Feign

- Declarative REST client
- Allows to make service calls without single line of implementation
- Similar to RestTemplate
- Automatically integrates with Ribbon/Eureka

Spring Cloud Feign


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
```

```
@SpringBootApplication
@RibbonClient(name="hello")
@EnableFeignClients
public class ClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class,
                args);
    }
```

Spring Cloud Feign


```
@FeignClient("hello")
public interface HelloClient {

    @RequestMapping("/greeting/{name}")
    String handshake(@PathVariable("name") String name);
}
```

```
@Autowired
private HelloClient helloClient;

@RequestMapping("/handshake/{name}")
public String hi(@PathVariable(value = "name") String name) {
    return helloClient.handshake(name);
}
```

- Sergey Morenets, 2017

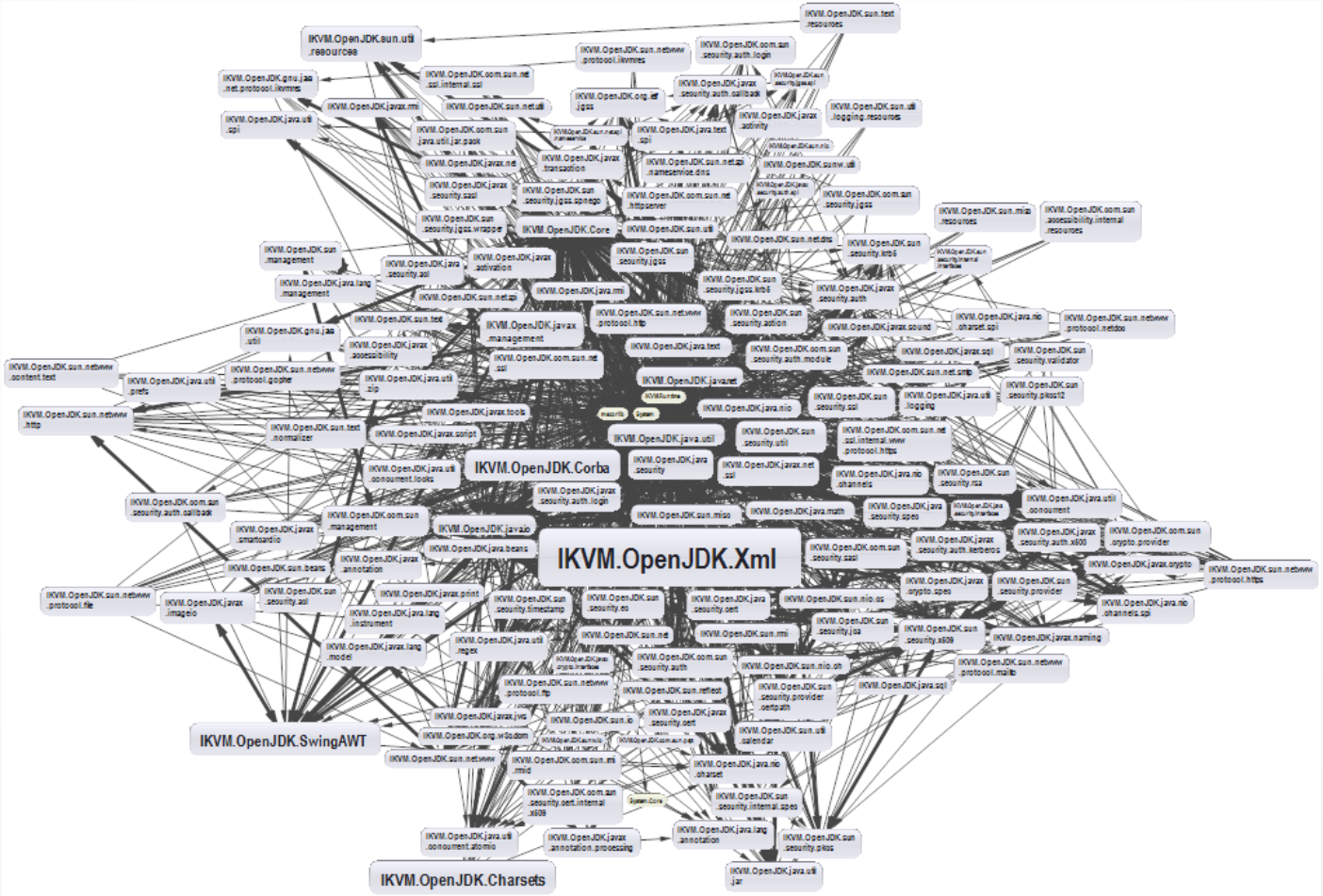
Task 14. Feign client

1. Add **Feign** dependency to the book application:
2. Update bootstrap class of book application and enable Feign clients using **@EnableFeignClients** annotation.
3. Create new Feign client **HitClient** interface that will provide communication with logging application.
4. Update your REST service that sends requests to the logging application and switch from **RestTemplate** to **HitClient**.

Cascading failure

- Sergey Morenets, 2017

Cascading failure



- Sergey Morenets, 2017

Circuit breaker pattern

- ✓ Prevents cascading failures
- ✓ When it discovers a failure it disconnects(opens) circuit
- ✓ When a problem is gone you can switch on(close) circuit

Spring Cloud Hystrix

- By default detects 20 failures in 5 seconds
- Provides fallback in case of service dependency failure
- Automatically closes circuit breaker after 5 seconds

Spring Cloud Hystrix


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableHystrix
public class ClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class,
                args);
    }
```

Spring Cloud Hystrix

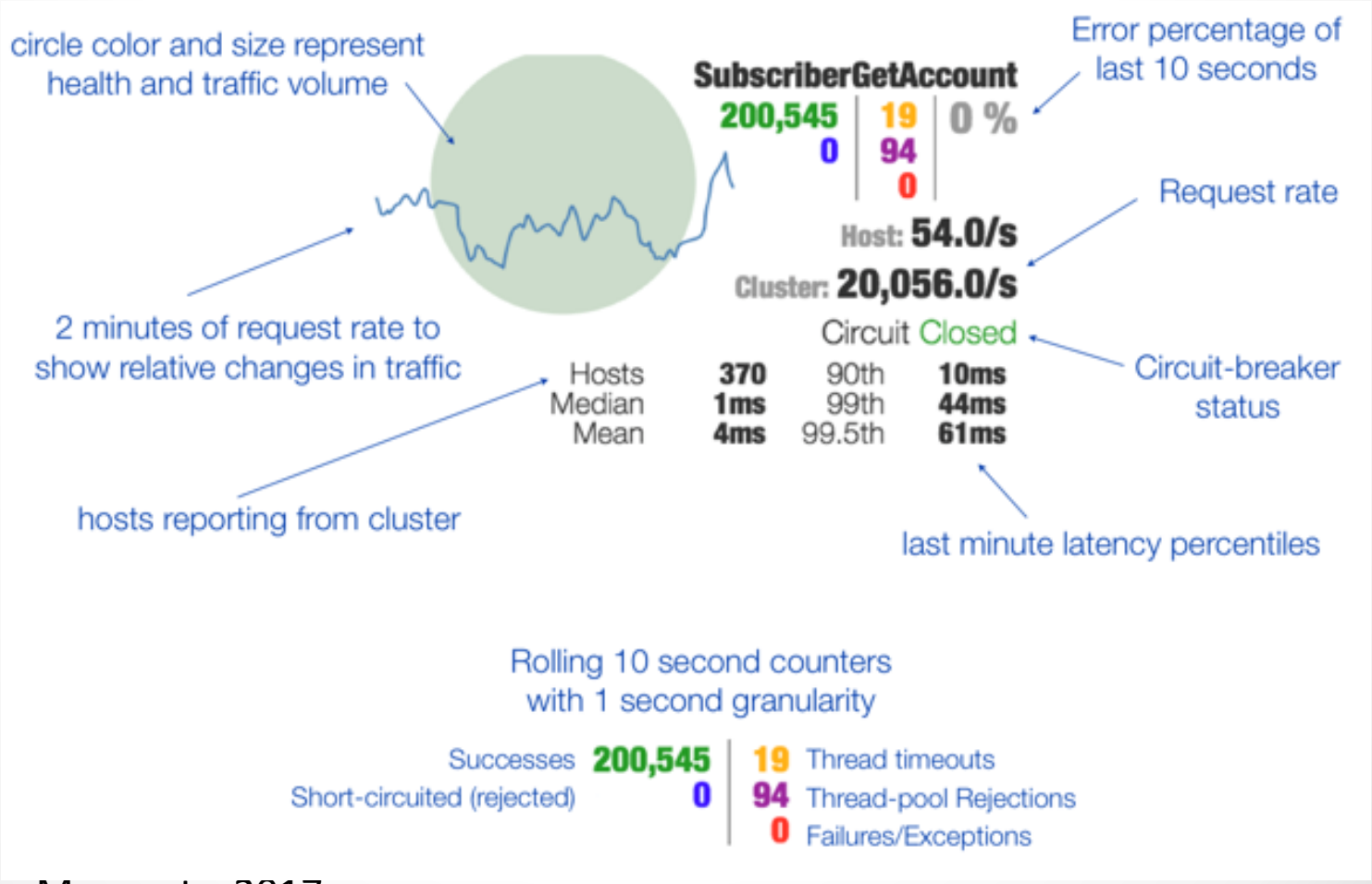

```
@HystrixCommand(fallbackMethod = "fallback")
public String hi(String name) {
    return helloClient.handshake(name);
}

public String fallback() {
    return "Service temporarily unavailable";
}
```

```
@HystrixCommand(fallbackMethod = "fallback")
public String hi(String name) {
    return helloClient.handshake(name);
}

public String fallback(String name) {
    return "Service temporarily unavailable";
}
```

Spring Cloud Hystrix

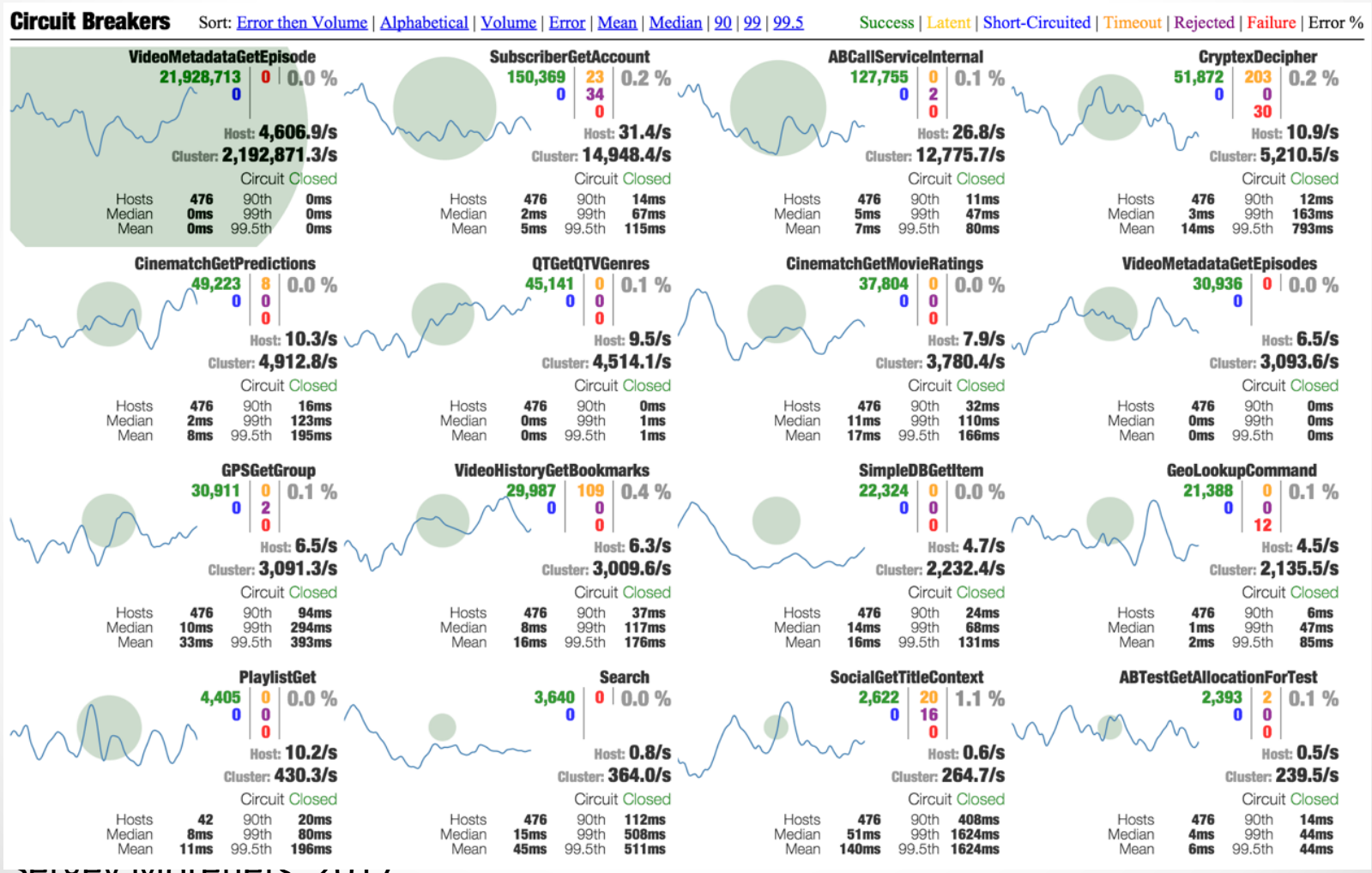

```
@RequestMapping("/handshake/{name}")
@HystrixCommand(fallbackMethod = "fallback", commandProperties = {
        @HystrixProperty(name =
                "circuitBreaker.errorThresholdPercentage", value = "1")
public String hi(String name) {
    return helloClient.handshake(name);
}

public String fallback(String name) {
    return "Service temporarily unavailable";
}
```

Hystrix dashboard



- Sergey Morenets, 2017

Hystrix dashboard



- Sergey Morenets, 2017

Hystrix dashboard. Configuration


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
</dependency>
```

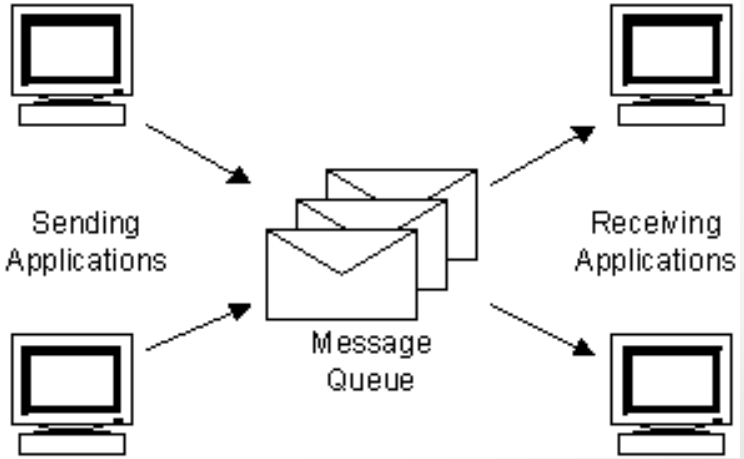
```
@SpringBootApplication
@EnableHystrix
@EnableHystrixDashboard
public class ClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClientApplication.class,
                args);
    }
```

Task 15. Hystrix

1. Add **Spring Cloud Hystrix** dependency:
2. Update ClientApplication and enable Hystrix support.
3. Update REST service that sends requests to the server and add support for **Hystrix fallback**.
4. Start **client/server** applications and make sure client correctly obtains responses from the server.

Message queue attributes

- ✓ Delivery
- ✓ Transactions (group messages together)
- ✓ Durability
- ✓ Sync/Async messaging
- ✓ High availability
- ✓ Load balancing

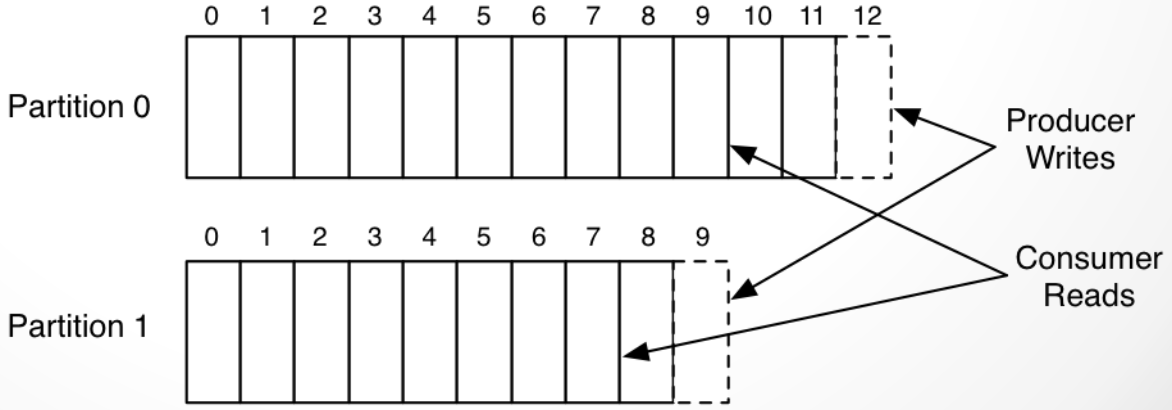


Apache Kafka

- ✓ Stream-processing low-latency high-thoughtput platform
- ✓ Developed by LinkedIn in Java/Scala
- ✓ Open-sourced in 2011
- ✓ Publisher/subscriber message queue (message broker)
- ✓ Uses **ZooKeeper** for cluster membership/ routing
- ✓ Event sourcing/ log aggregation
- ✓ Competes with RabbitMQ/ActiveMQ

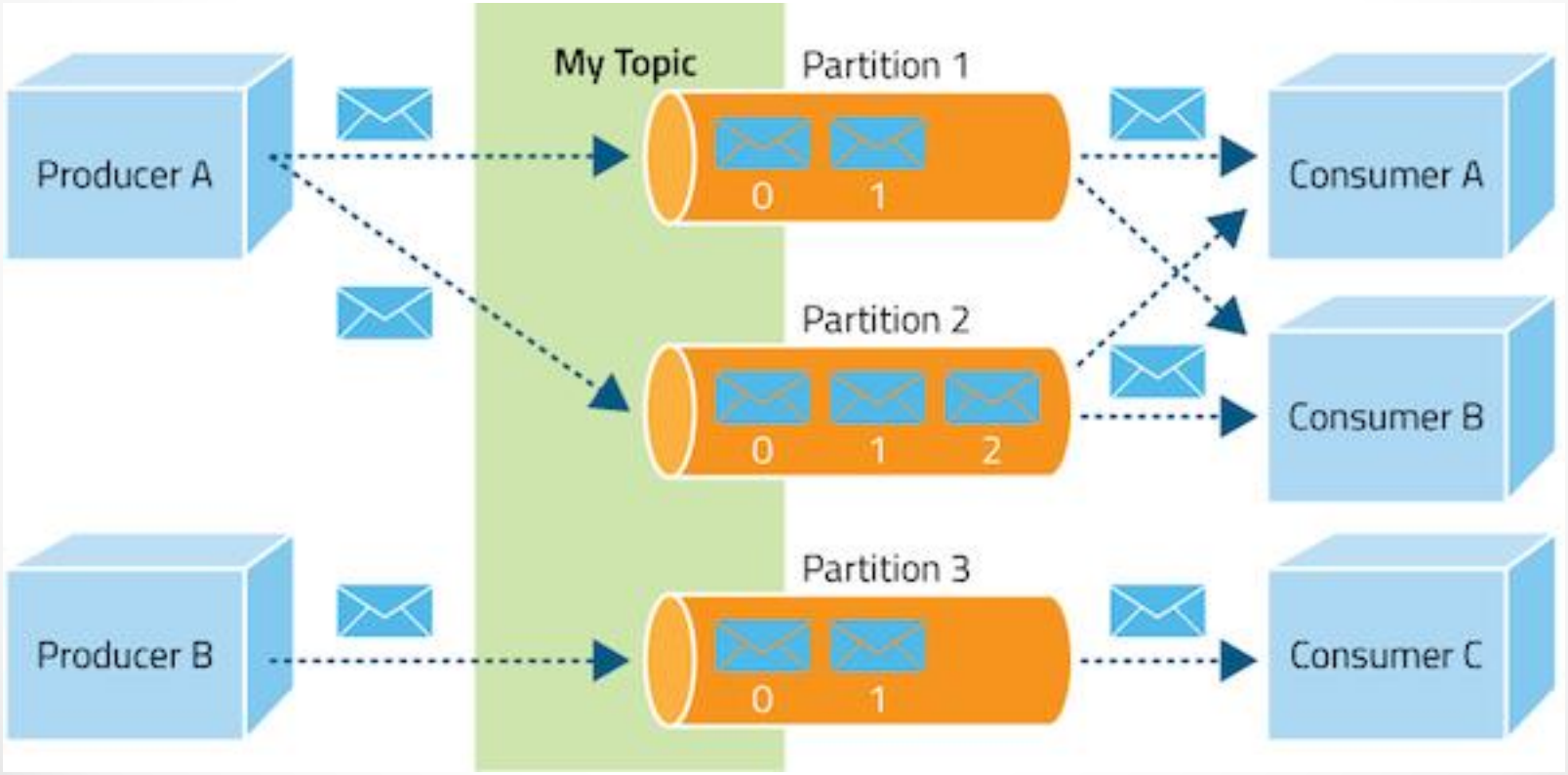
Apache Kafka. Messaging

- ✓ Producers publish messages to the topic
- ✓ Subscribers subscribe to one or more topic
- ✓ Topic is sharded write-ahead log
- ✓ Producers append messages to the logs
- ✓ Each message is key/value pair
- ✓ Key is used to determine partition



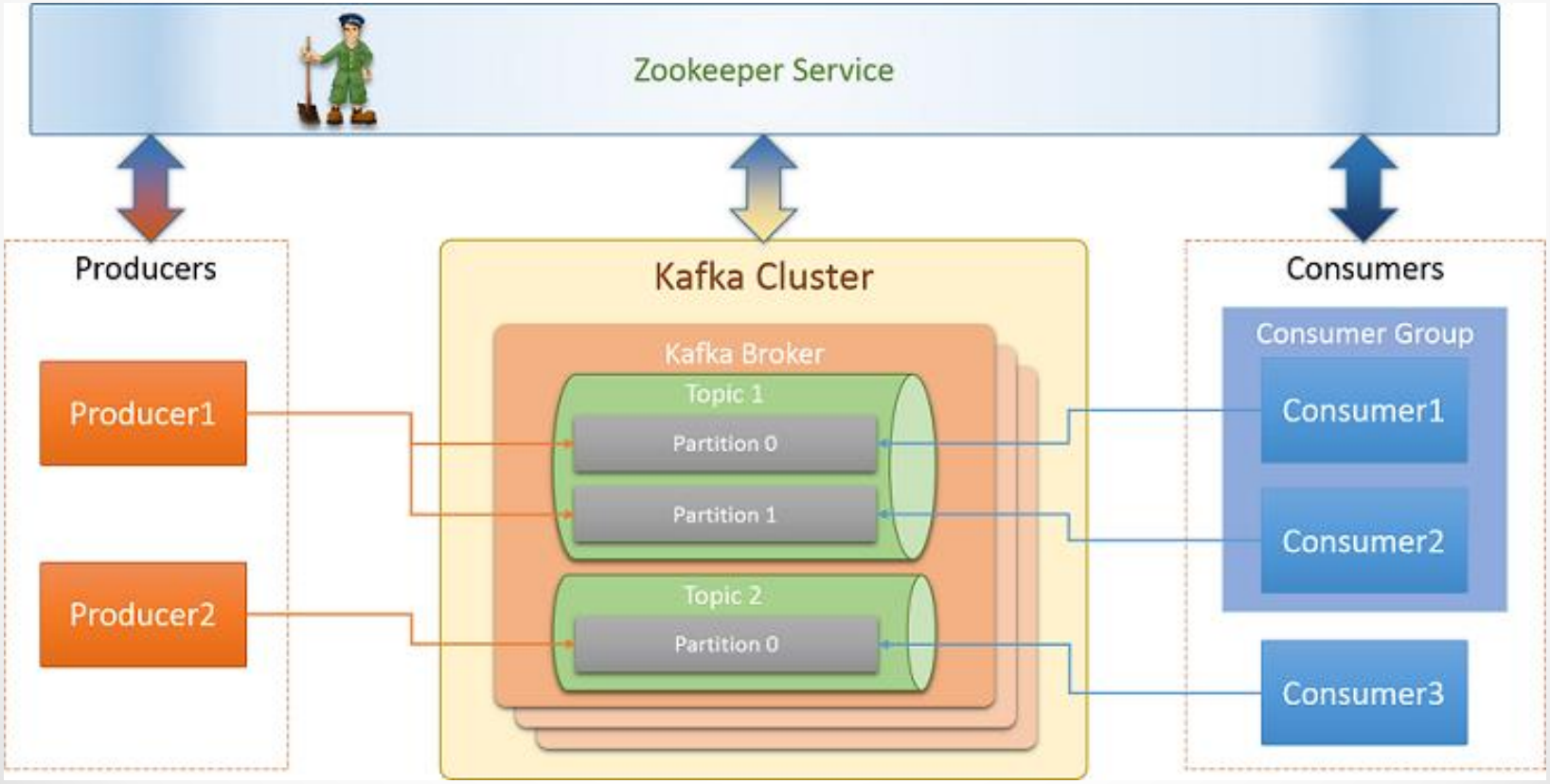
- Sergey Morenets, 2017

Apache Kafka. Messaging



- Sergey Morenets, 2017

Apache Kafka. Clustering



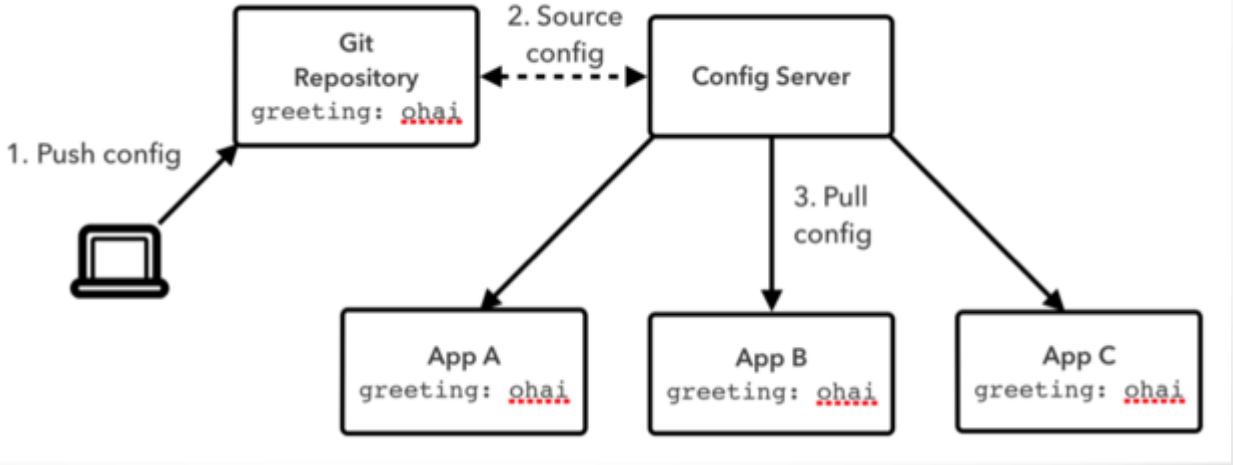
- Sergey Morenets, 2017

Task 16. Apache Kafka

1. Download and install **Apache Kafka**
2. Review **server.properties** and **zookeeper.properties** in c:/Kafka/config folder
3. Run Apache **Zookeeper.** Review console logs.
4. Run Apache **Kafka**
5. Run command *“kafka-topics.bat --list --zookeeper localhost:2181”* to view all existing topics.

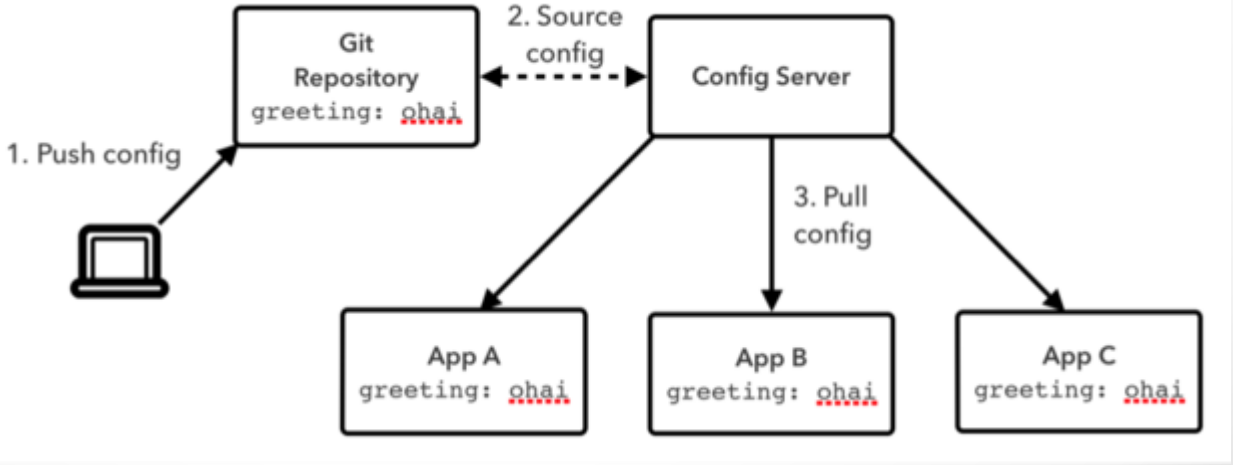
Static configuration

- ✓ Configuration server manages centralized storage of the application settings
- ✓ Clients connect to the configuration server **at startup-time** and fetch settings



Dynamic configuration

- ✓ Client periodically poll changes from server
- ✓ Server pushes changes to the clients



Spring Cloud. Refresh scope

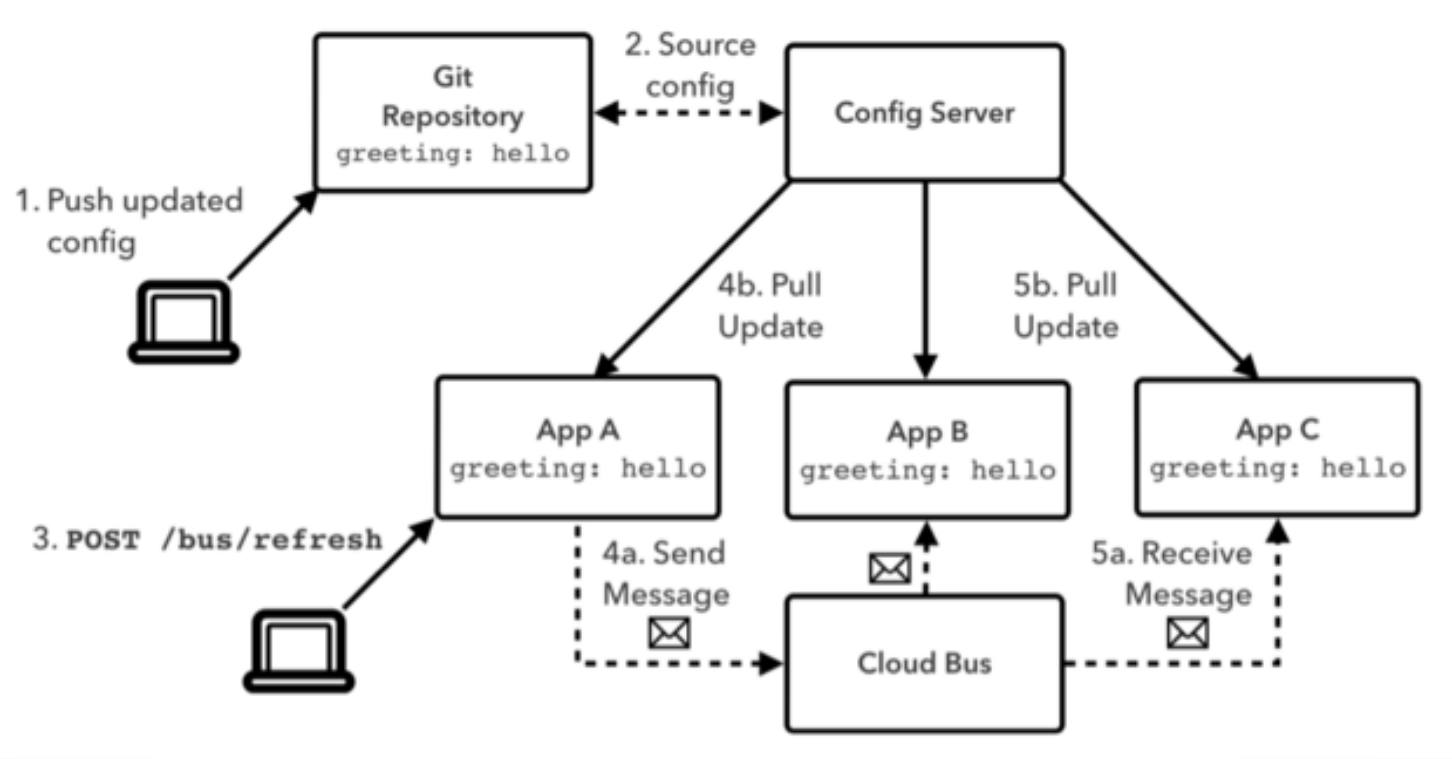
- ✓ **RefreshScope** beans are lazy and proxied
- ✓ Bean is reloaded after configuration changes

```
@RestController
@RefreshScope
public class ServerController {

    @Autowired
    private HelloClient helloClient;
```

POST /refresh

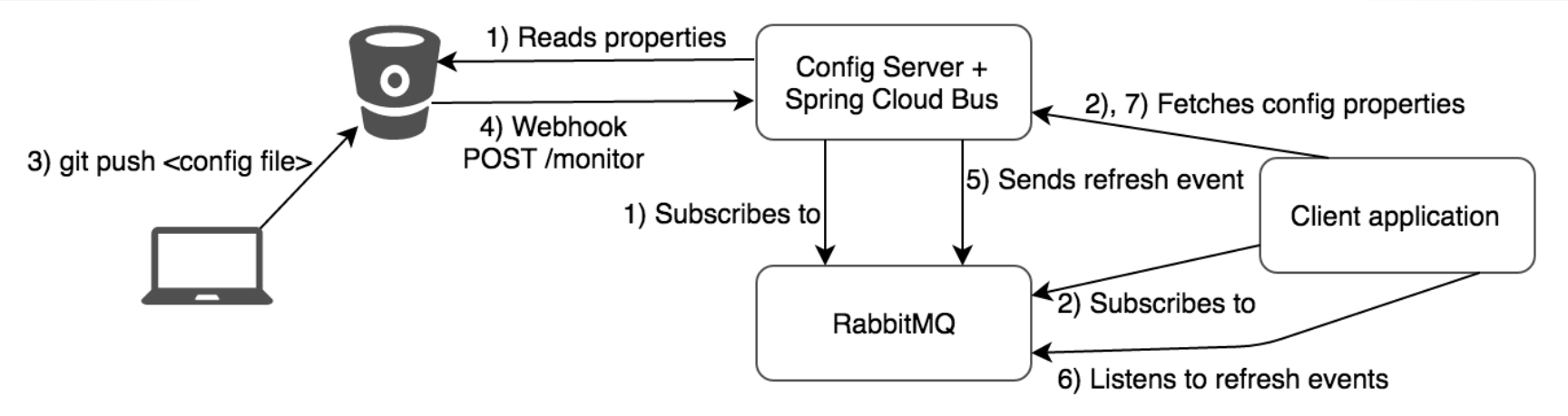
- Sergey Morenets, 2017

Spring Cloud. Refresh scope



- Sergey Morenets, 2017

Spring Cloud Bus



Spring Cloud Bus

- ✓ Relies on Spring Cloud Stream
- ✓ Uses messaging providers(Apache Kafka, Rabbit MQ)

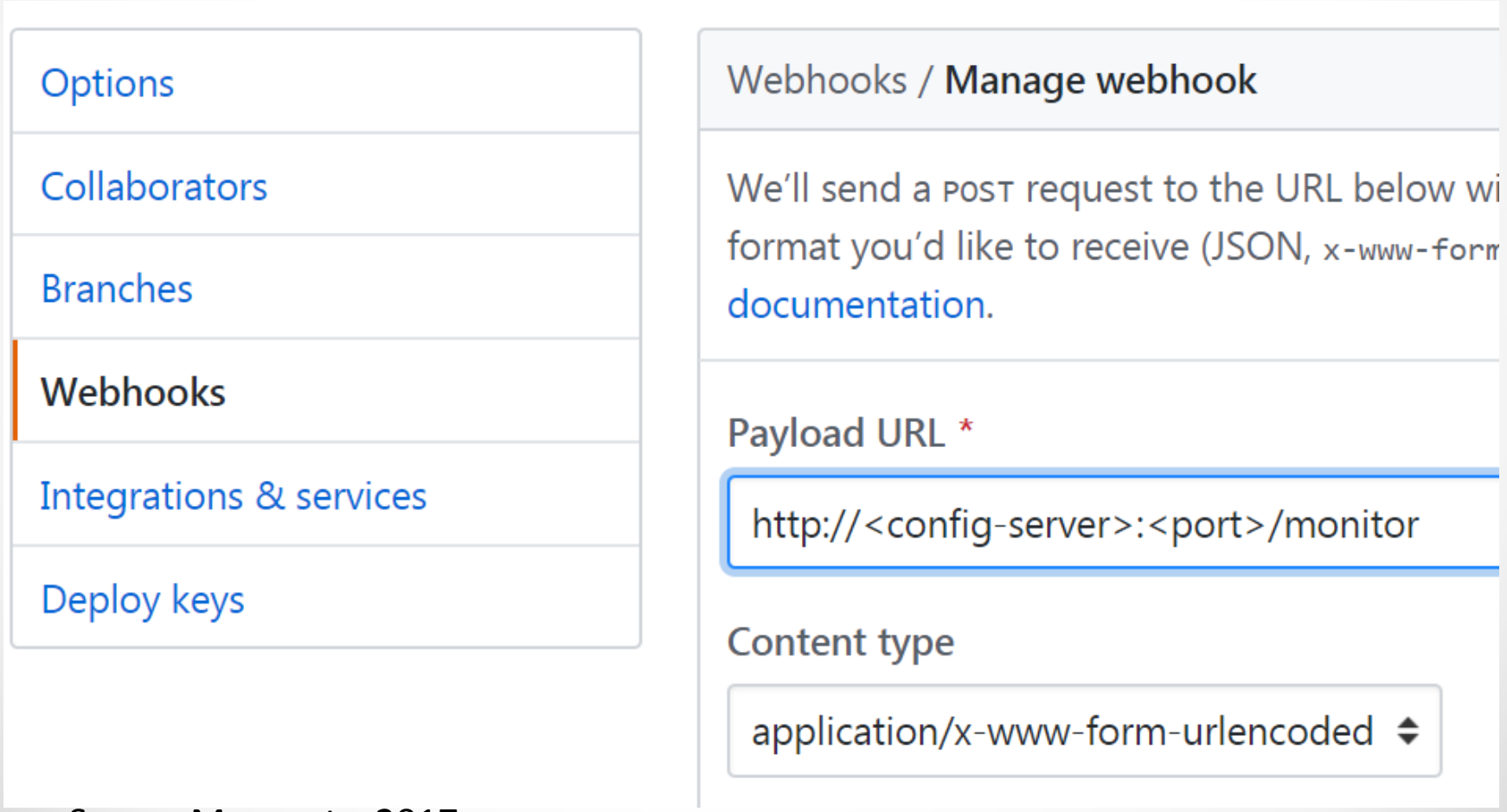
Spring Cloud Bus

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-bus-impq</artifactId>
</dependency>
```


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-bus-kafka</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-monitor</artifactId>
</dependency>
```

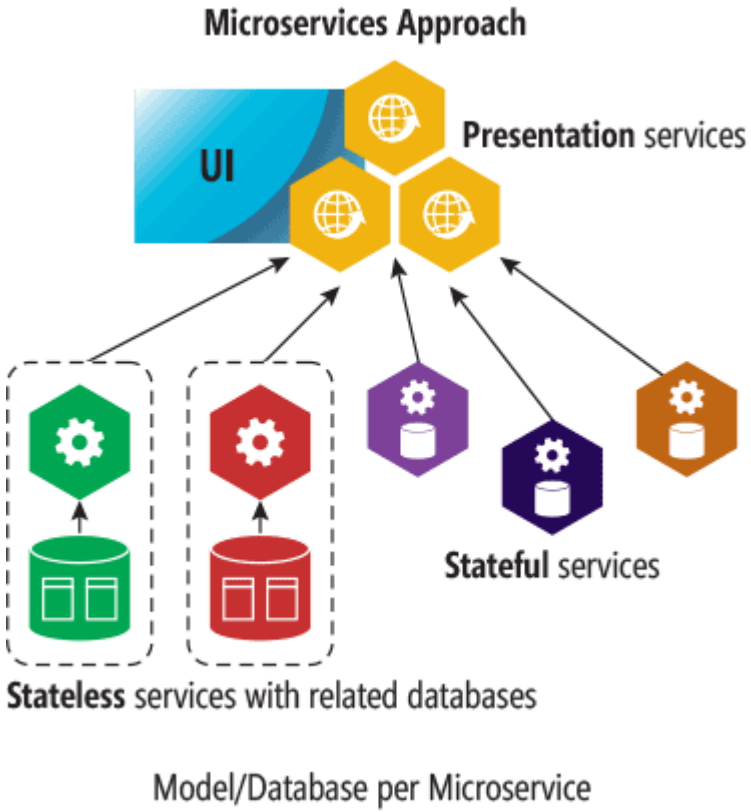
Spring Cloud Bus. Monitoring



Task 17. Spring Cloud Bus

- Add dependency to book/logging/config-server applications
- Add dependency to **config-server** application:
- Put **@RefreshScope** annotation on REST controllers in book/logging applications.
- Add **spring.cloud.bus.enabled=true** line to application.properties of config-server.
- Run applications. Verify that applications started using **Apache Kafka**.

API gateway

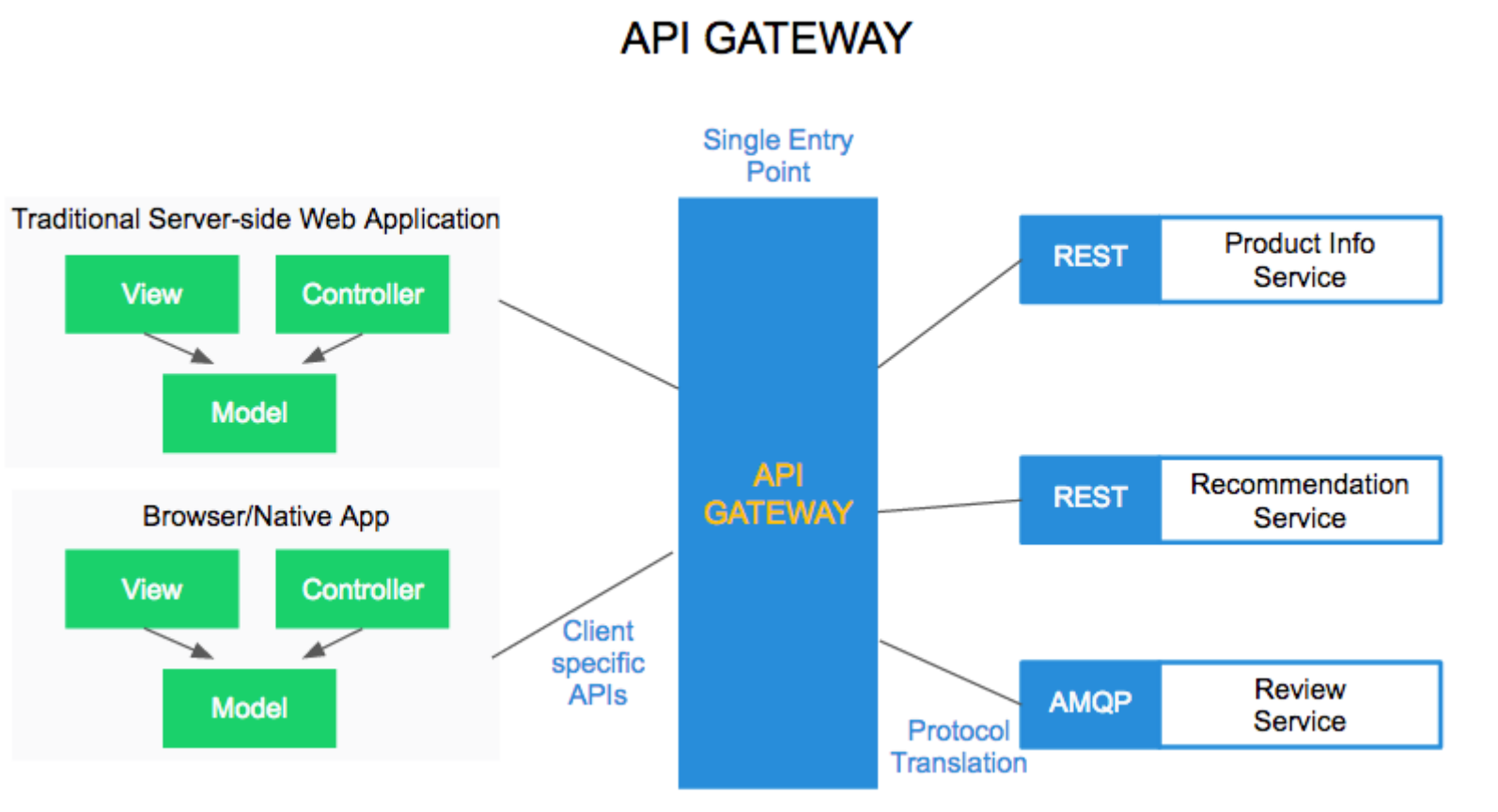


- Sergey Morenets, 2017

Concerns

- ✓ **Internal API** exposing
- ✓ Security(CORS)
- ✓ Multiple calls

API gateway



- Sergey Morenets, 2017

API gateway

- ✓ **Façade pattern** providing concrete client needs
- ✓ Decreases remote calls
- ✓ Security
- ✓ Caching
- ✓ Protocol translation

Spring Cloud Zuul

- ✓ **Router and server-side load balancer**
- ✓ Includes Ribbon/Hystrix
- ✓ Eureka service ID is URI
- ✓ Analogs are Kong, ApiGee and MuleSoft

Netflix and Zuul

- ✓ Authentication
- ✓ Insights
- ✓ Stress/canary testing
- ✓ Dynamic routing
- ✓ Service migration
- ✓ Static response testing

Zuul workflow

- ✓ There is a service with id “order”
- ✓ Zuul create local proxy for its id
- ✓ Proxy uses **Ribbon** to auto-discover service and forwards request
- ✓ All requests are executed inside **Hystrix** command
- ✓ Request results are available in Hystrix dashboard

http://localhost:8080/product → http://localhost:9000/

http://localhost:8080/order → http://localhost:9001/

http://localhost:8080/catalog → http://localhost:9002/

Spring Cloud Zuul. Configuration


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zuul</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableZuulProxy
public class ZuulApplication {
    public static void main(String[] args) {
        SpringApplication.run(ZuulApplication.class,
                args);
    }
}
```

Spring Cloud Zuul. Service mapping


```
zuul:
  prefix: /api
  routes:
    orders:
      path: /orders/**
      serviceId: orders-service
    messages:
      path: /messages/**
      url: http://localhost:8002
```



```
ribbon:
  eureka:
    enabled: false
```

Task 18. Zuul

1. Review **zuul** project.
2. Update **ZuulApplication** class to enable **Zuul** support.
3. Update **application.yml**
4. Start applications. Verify that API gateway works
5. Stop applications and disable Eureka support in **application.yml**:
6. Start applications again and make sure **API gateway** works properly.

Authentication

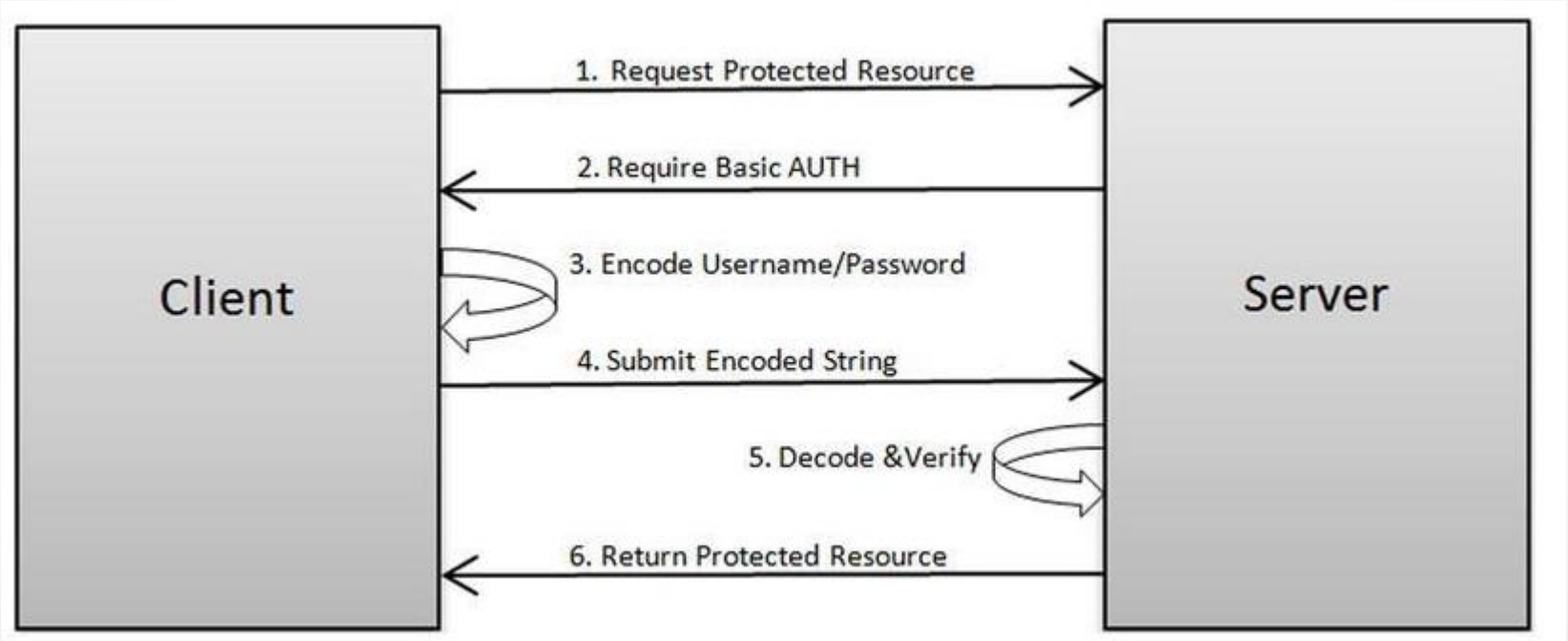
Из диалога двух программистов:
— Кажется, у нас дыра в безопасности!
— Слава Богу, хоть что-то у нас в безопасности...

- Sergey Morenets, 2017

Authentication

- ✓ Basic
- ✓ Digest
- ✓ Token
- ✓ Digital Signature
- ✓ Certificate
- ✓ OAuth 1.0/2.0

Basic authentication



- Sergey Morenets, 2017

Spring Security

- Formerly **Acegi Security**
- Support for authentication and authorization
- Integration with **Spring MVC(REST)**
- Attacks protections (CSRF, session fixation)
- Basic, Digest, CAS, OpenID, JAAS and LDAP authentication
- Integration testing
- Based on filter chain

Spring Security


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

Spring Security


```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
@EnableWebSecurity
public class SecurityConfiguration extends
    WebSecurityConfigurerAdapter {
```

- Sergey Morenets, 2017

Spring Security

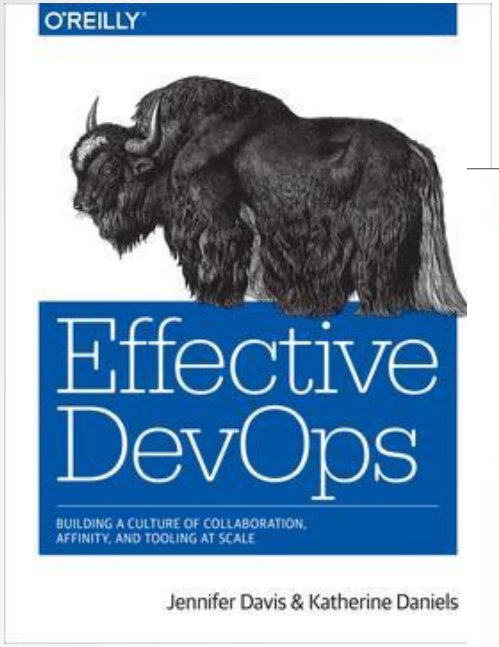

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
    auth.inMemoryAuthentication().
        withUser("user").password("user").roles("USER").
            and().withUser("admin")
            .password("admin").roles("USER", "ADMIN");
}

@Override
protected void configure(HttpSecurity http)
        throws Exception {
    http.authorizeRequests().anyRequest().
    fullyAuthenticated();
    http.httpBasic();
    http.csrf().disable();
}
```

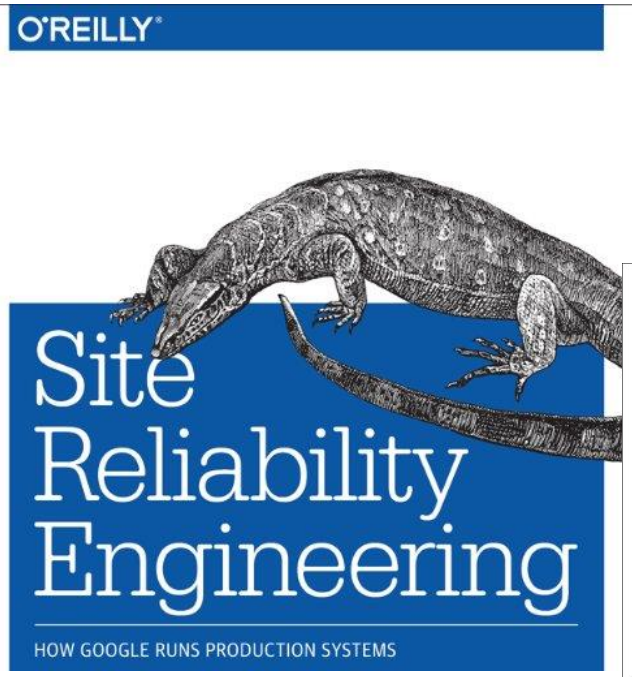
Important types

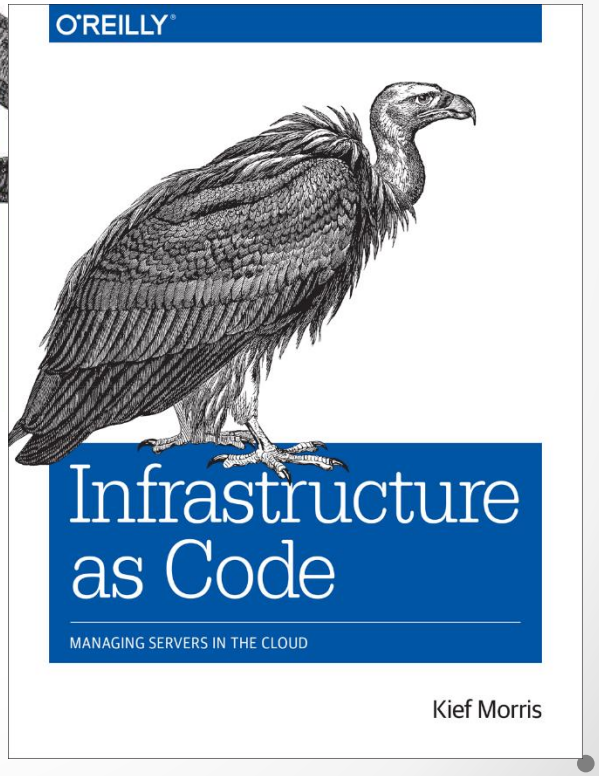
Name	Description
Principal	Any logged user with login id
Authentication	Token for authentication request, including authorities, credentials and principal
SecurityContext	Authentication information linked to current execution thread
SecurityContextHolder	Allows to get/update security context for current execution thread
AuthenticationProvider	Implementation that can authorize user based on current **Authentication** object

Task 19. Spring Security

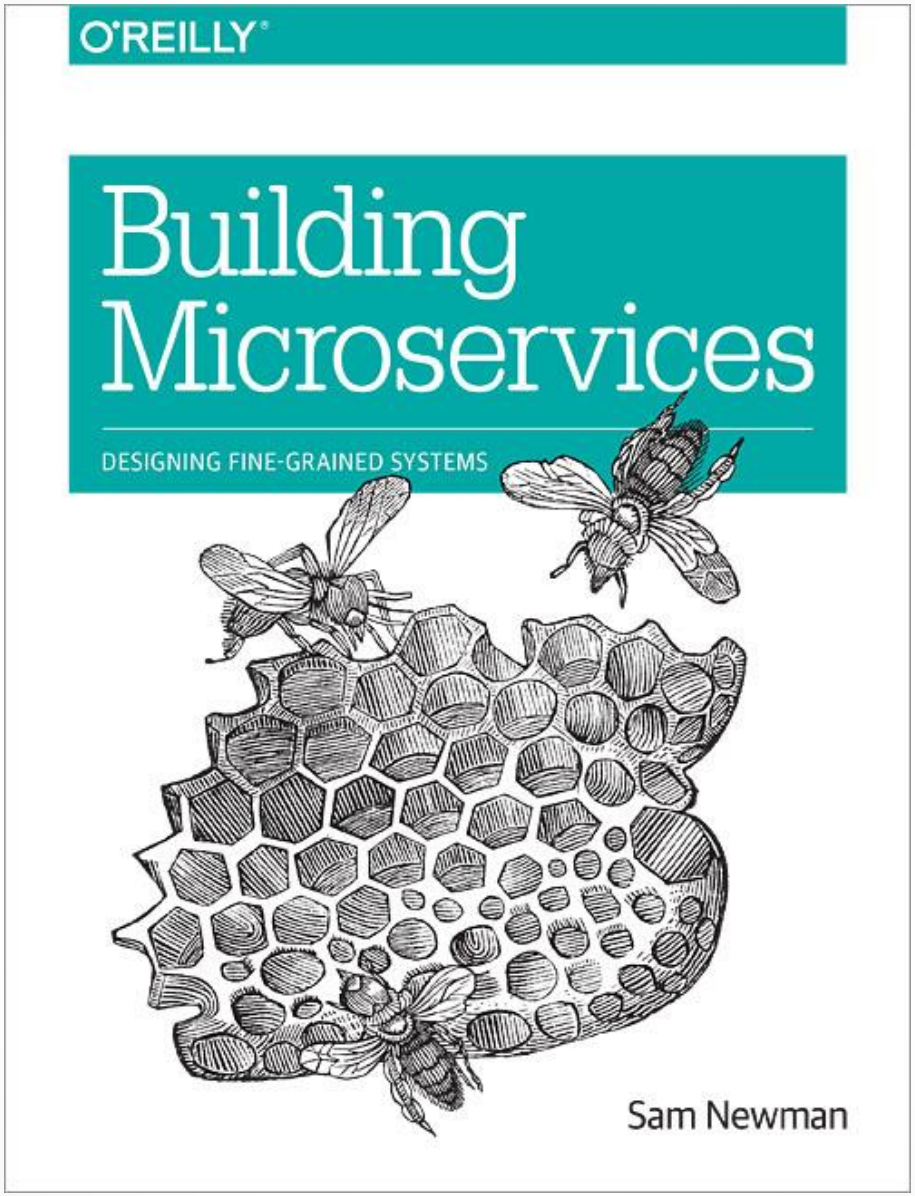
1. Add Spring Security dependency for **Zuul** application
2. Add **Spring Security** configuration:
    1. Add users with different roles
3. Start Zuul application and try invoke any service endpoints.







- Sergey Morenets, 2017



- Sergey Morenets, 2017

- Sergey Morenets, 2017

✓ Sergey Morenets, sergey.morenets@gmail.com