Microservices development

November, 11-12 2017

Sergey Morenets, 2017

Event sourcing

- ✓ Captures all changes to an application state as a sequence of events

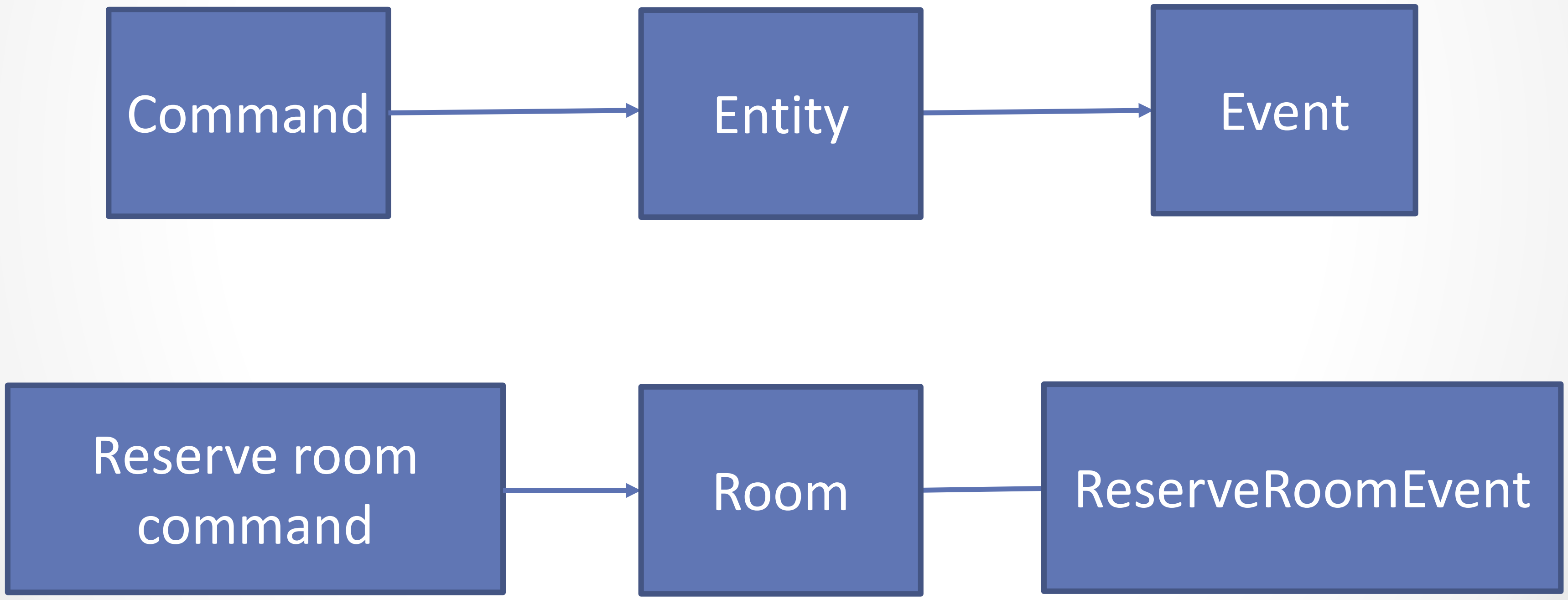
Event sourcing

- ✓ Enterprise design pattern
- ✓ Allows to see state changes (at some period of time)
- ✓ Every state change is stored in an event object
- ✓ Each name has a meaningful name
- ✓ All changes to the domain object are initiated by the event objects
- ✓ Past events are immutable

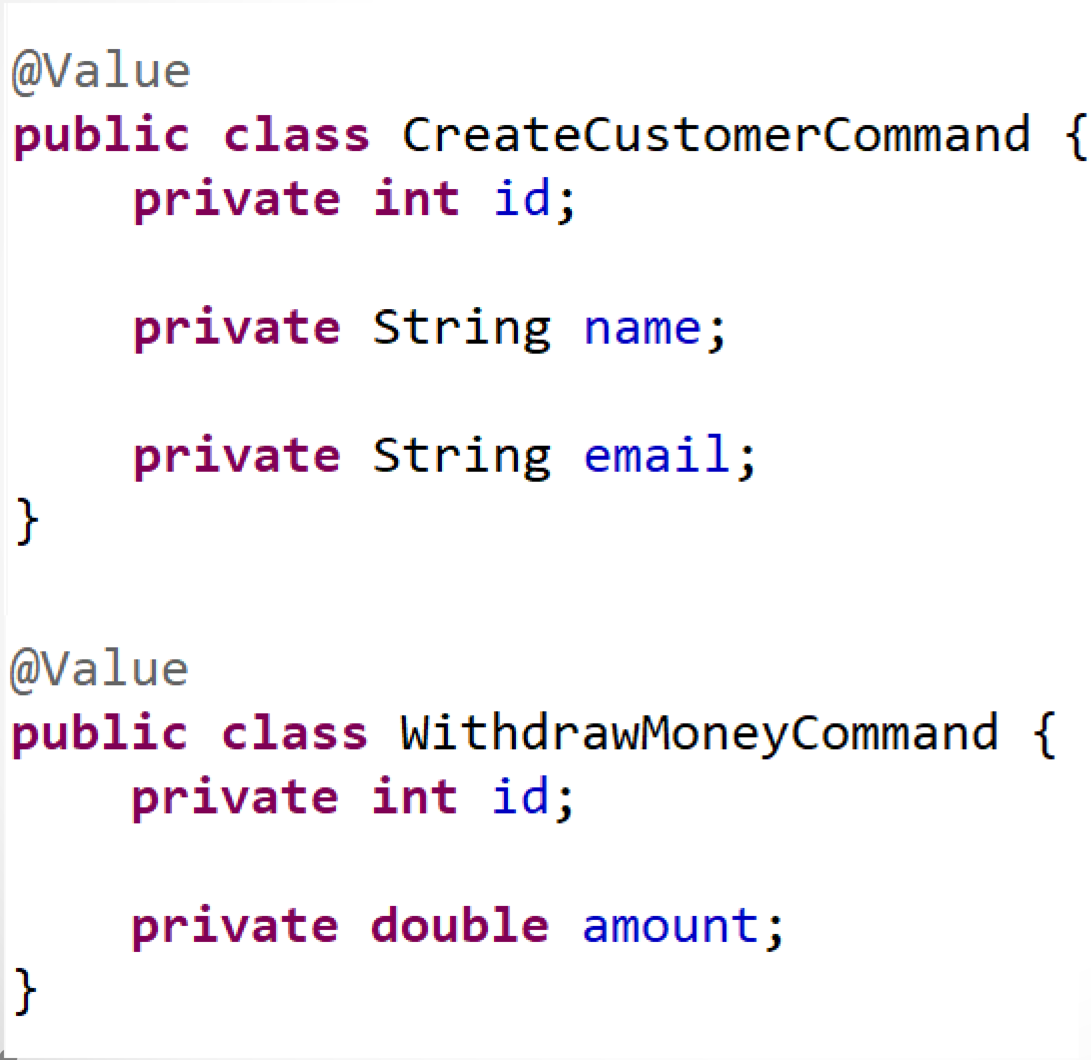
Event sourcing

- ✓ You can cache temporal application state in memory
- ✓ UI requires projects of all the event log
- ✓ Simplifies data consistency
- ✓ You cannot remove event types

Event sourcing



Commands



```java
@Value
public class CreateCustomerCommand {
    private int id;

    private String name;

    private String email;
}

@Value
public class WithdrawMoneyCommand {
    private int id;

    private double amount;
}
```

Domain entity. Commands

```
@Data
public class Customer {
    private int id;

    private String name;

    private String email;

    private double balance;

    public List<BaseEvent> process(CreateCustomerCommand cmd) {
        return Arrays.asList(new CustomerCreatedEvent(cmd.getId(),
                cmd.getName(), cmd.getEmail()));
    }
}
```

Domain entity. Commands

```
public List<BaseEvent> process(WithdrawMoneyCommand cmd) {
    if (cmd.getAmount() > balance) {
        return Arrays.asList(
                new WithdrawMoneyEvent(cmd.getId(),
                        cmd.getAmount()));
    } else {
        return Arrays.asList(
                new WithdrawMoneyFailureEvent(
                        cmd.getId(), "Insufficient funds"));
    }
}
```

Domain entity. Events

```
public void apply(CustomerCreatedEvent event) {
    this.id = event.getId();
    this.name = event.getName();
    this.email = event.getEmail();
}

public void apply(WithdrawMoneyEvent event) {
    this.balance -= event.getAmount();
}
```

Task 10. Event sourcing

1. Create command classes that will start any change in the application event.
2. Update controllers/services to use these commands
3. Update **Order** entity to react to the specified commands, update state and generate events
4. Create entity **EventLog** that will store all the generated events (event type, event payload, creation date)

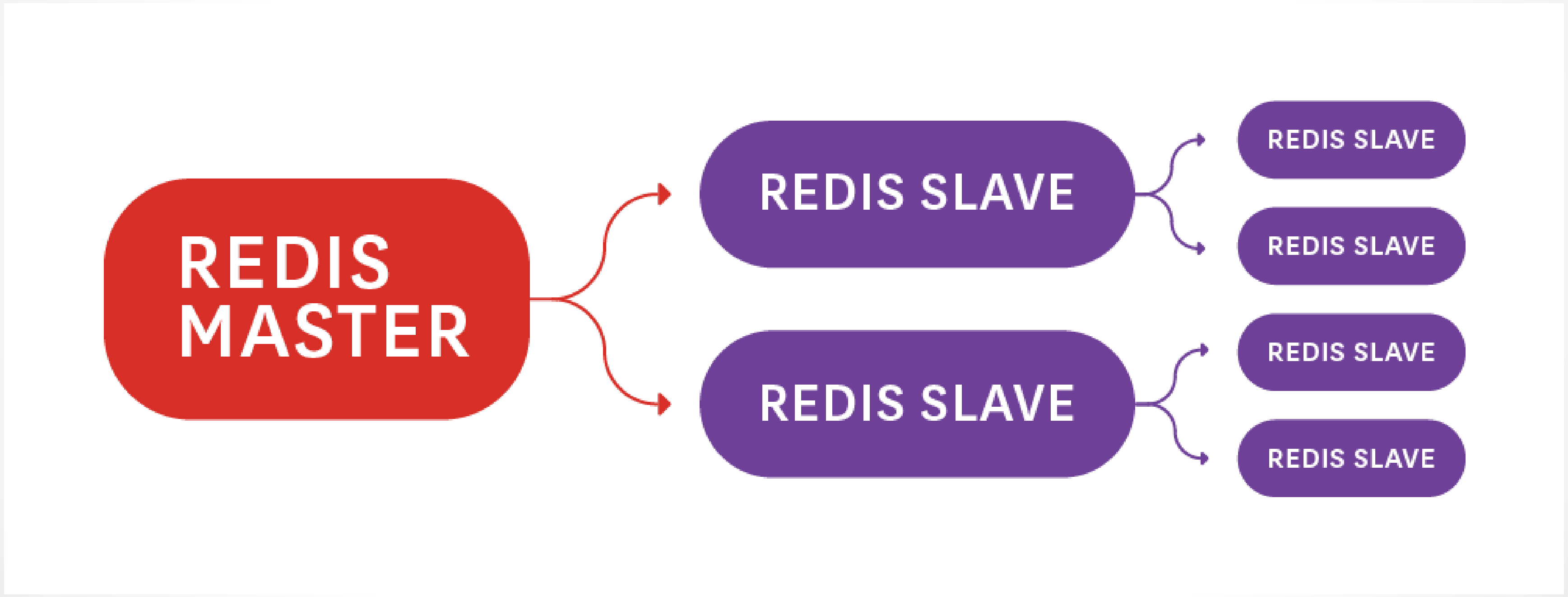
Redis

- ✓ Created in 2009 by Salvatore Sanfillipo as **Re**mote **Di**sctionary **S**erver
- ✓ Open-source
- ✓ Super-lightweight and easy to use
- ✓ In-memory **data structure** store
- ✓ 6 data types, 180+ commands
- ✓ Optional durability with snapshots or journals
- ✓ Provides automatic partitioning with **Redis Cluster**
- ✓ Supports monitoring and metrics
- ✓ Atomic, isolated and consistent

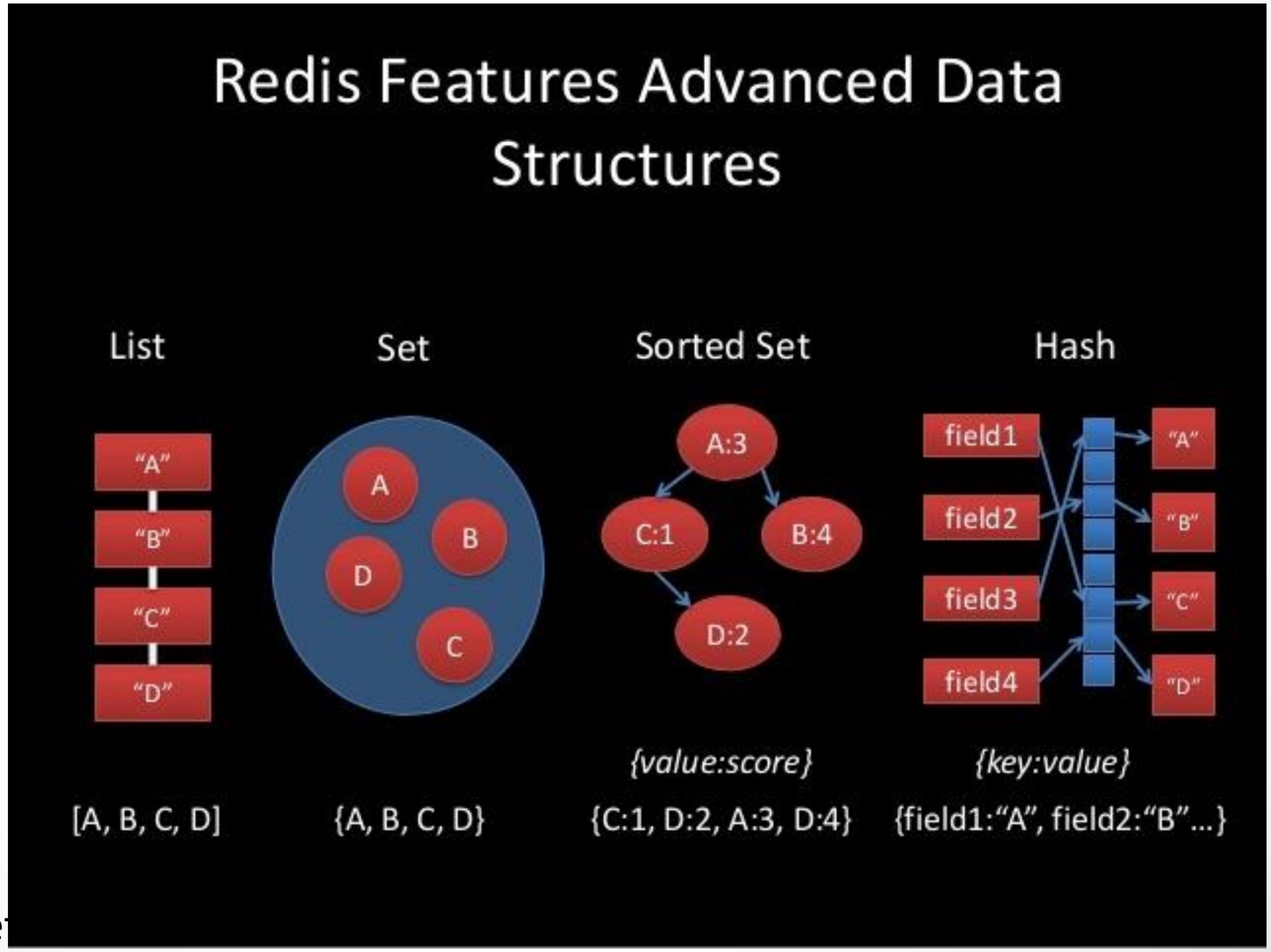
Redis. Popularity

- ✓ Competes with **Memcached**(multi-threaded vs single-threaded)
- ✓ Sometimes called “**Memcached on steroids**”
- ✓ Supports up to **512M** for key/value size (Memcached supports 250 bytes)
- ✓ #1 key-value database
- ✓ #4 NoSQL database

Redis replication



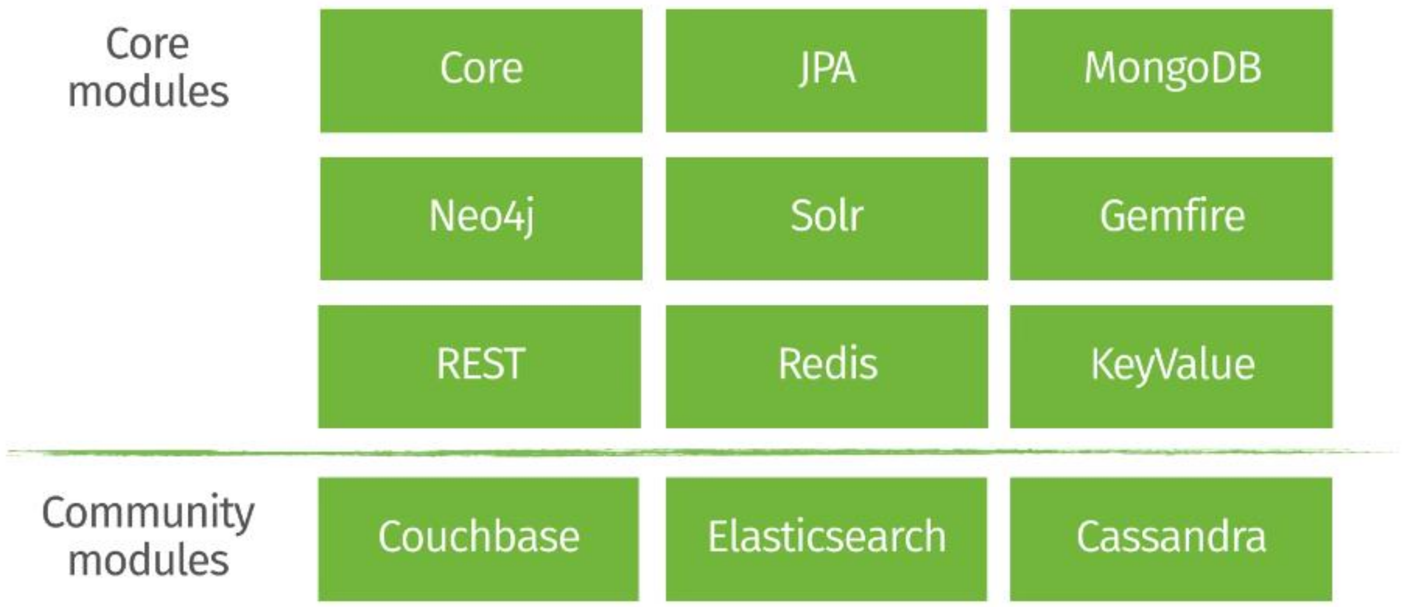
Redis data types



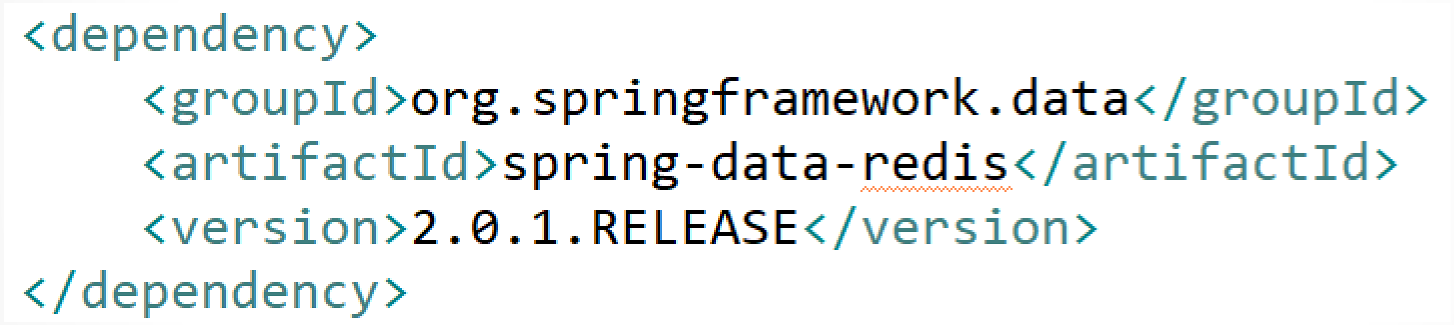
Task 11. Redis configuration

1. Download and install Redis.
2. Review **redis.conf** configuration file (or **redis.windows.conf** on Windows platform).
3. Start Redis command-line using **redis-cli** command.
4. Starts another Redis console
5. Try to print all the configuration settings
6. Run **redis-cli --stat** command to view active connections to **Redis** server.

Spring Data Modules

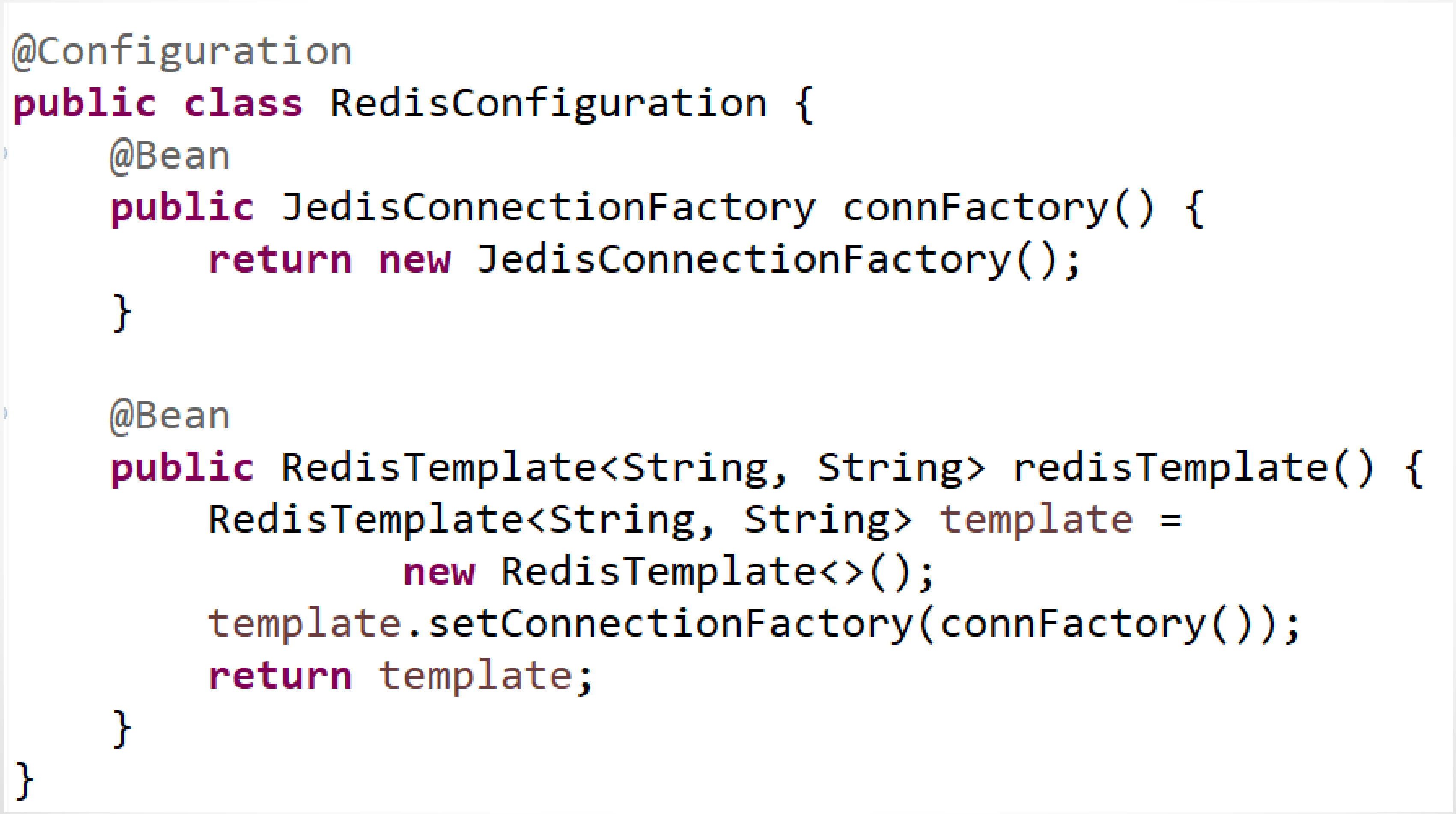


Spring Data Redis. Maven



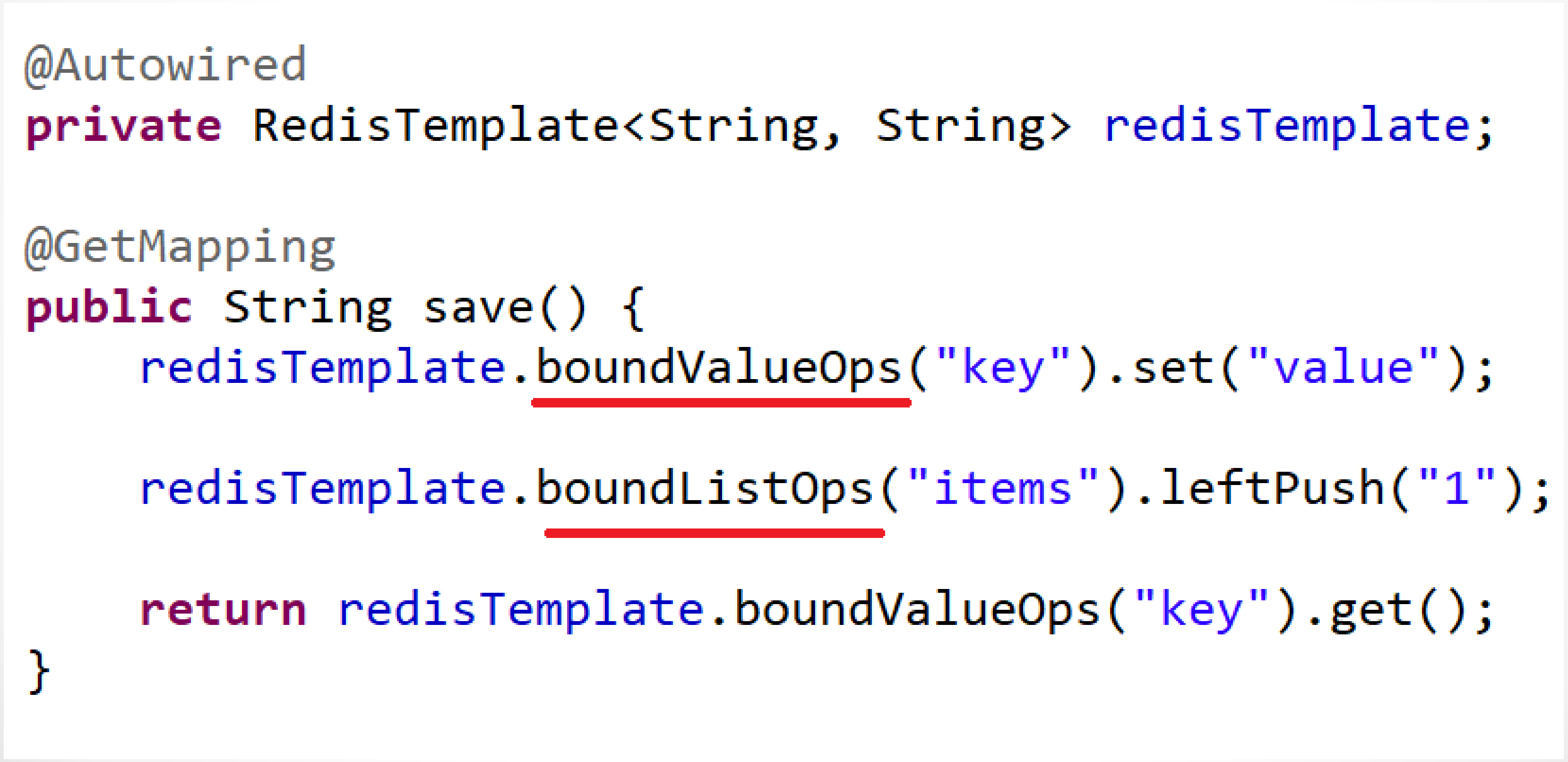
```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-redis</artifactId>
    <version>2.0.1.RELEASE</version>
</dependency>
```

Spring Data Redis. Configuration



```
@Configuration
public class RedisConfiguration {
    @Bean
    public JedisConnectionFactory connFactory() {
        return new JedisConnectionFactory();
    }

    @Bean
    public RedisTemplate<String, String> redisTemplate() {
        RedisTemplate<String, String> template =
                new RedisTemplate<>();
        template.setConnectionFactory(connFactory());
        return template;
    }
}
```

RedisTemplate



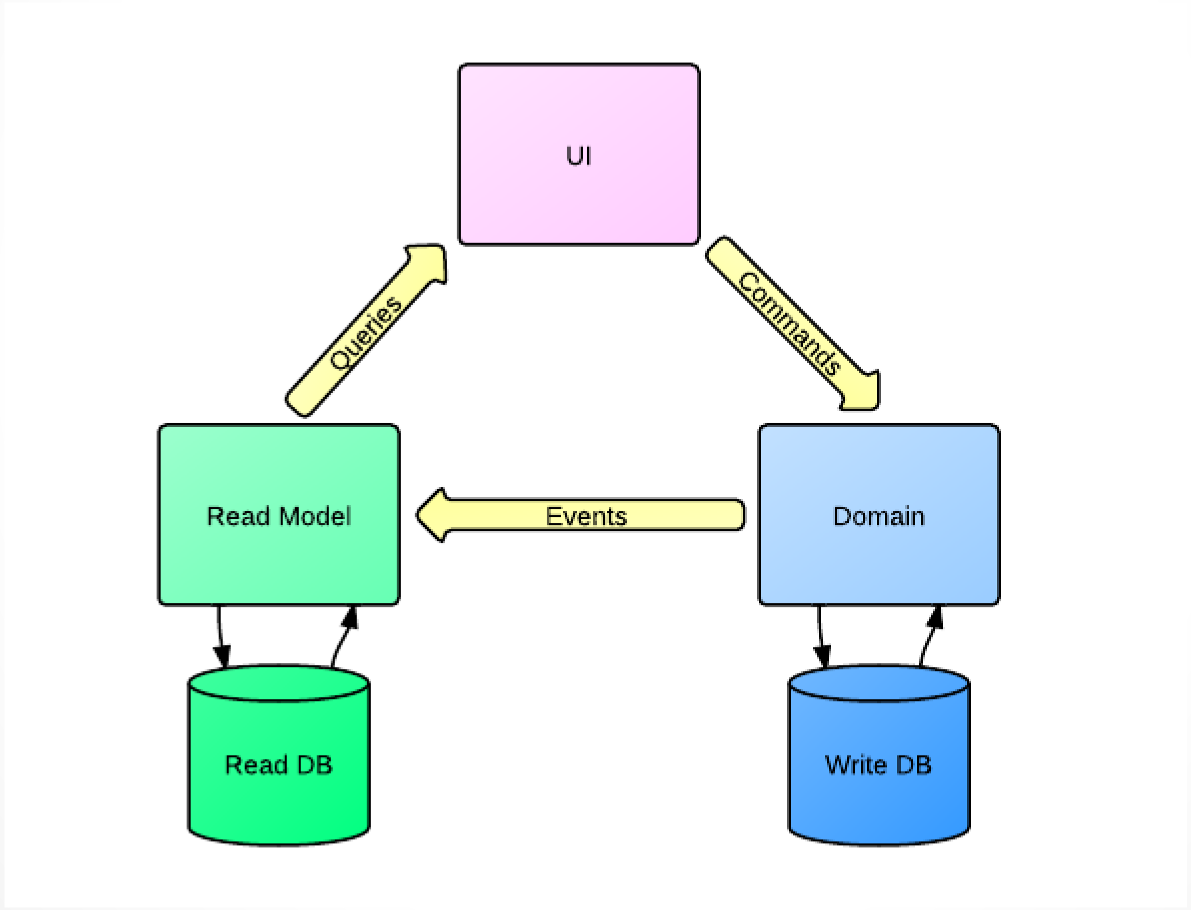
```java
@Autowired
private RedisTemplate<String, String> redisTemplate;

@GetMapping
public String save() {
    redisTemplate.boundValueOps("key").set("value");

    redisTemplate.boundListOps("items").leftPush("1");

    return redisTemplate.boundValueOps("key").get();
}
```

Task 12. Spring Data Redis

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

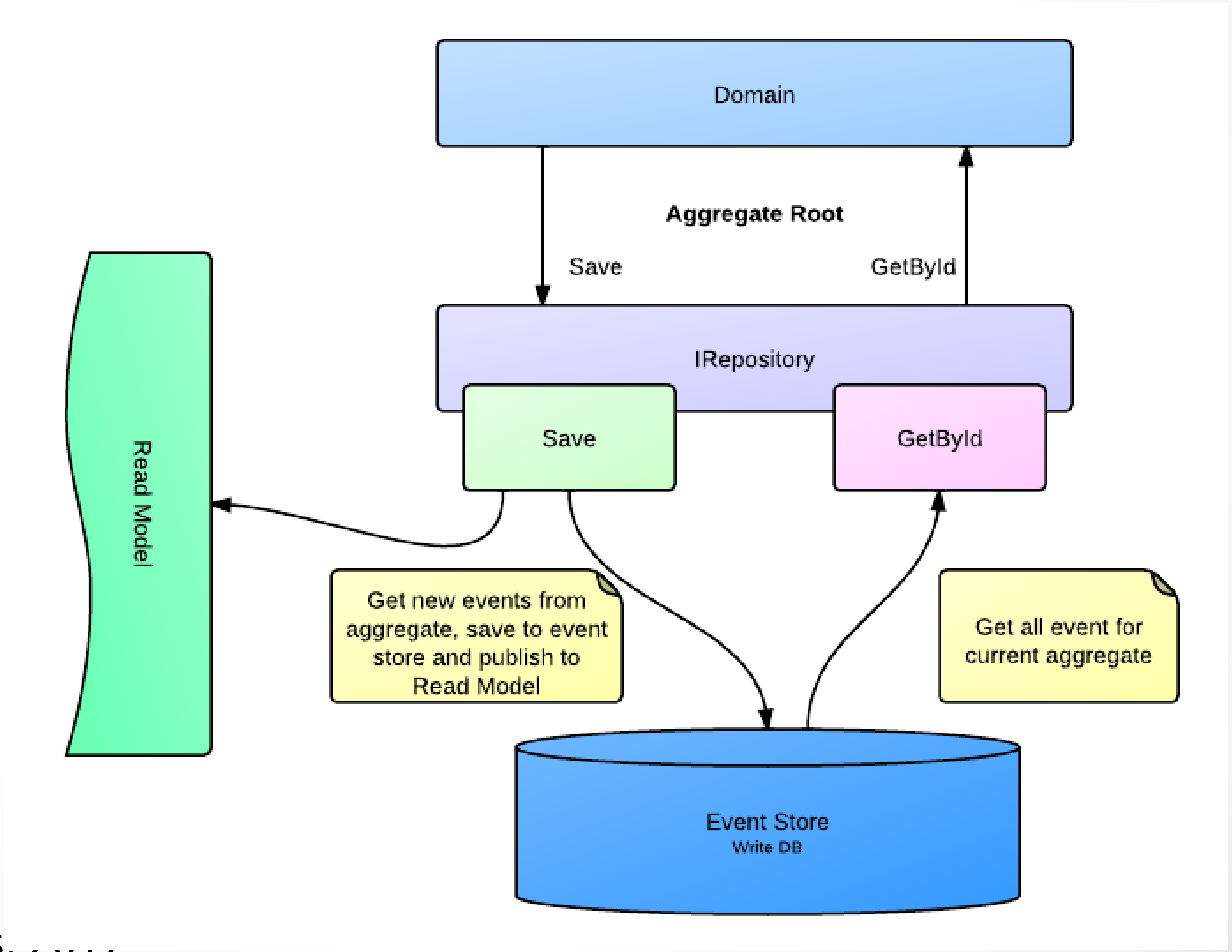
   were saved in the database.

CQRS + Event sourcing



Sergey Morenets, 2017

CQRS + Event sourcing


Domain
Aggregate Root
Save
GetById
IRepository
Save
GetById
Read Model
Get new events from aggregate, save to event store and publish to Read Model
Get all event for current aggregate
Event Store
Write DB

Query engines

- ✓ MongoDB
- ✓ Redis
- ✓ ElasticSearch
- ✓ Neo4J
- ✓ AWS DynamoDB
- ✓ AWS Cloud Search

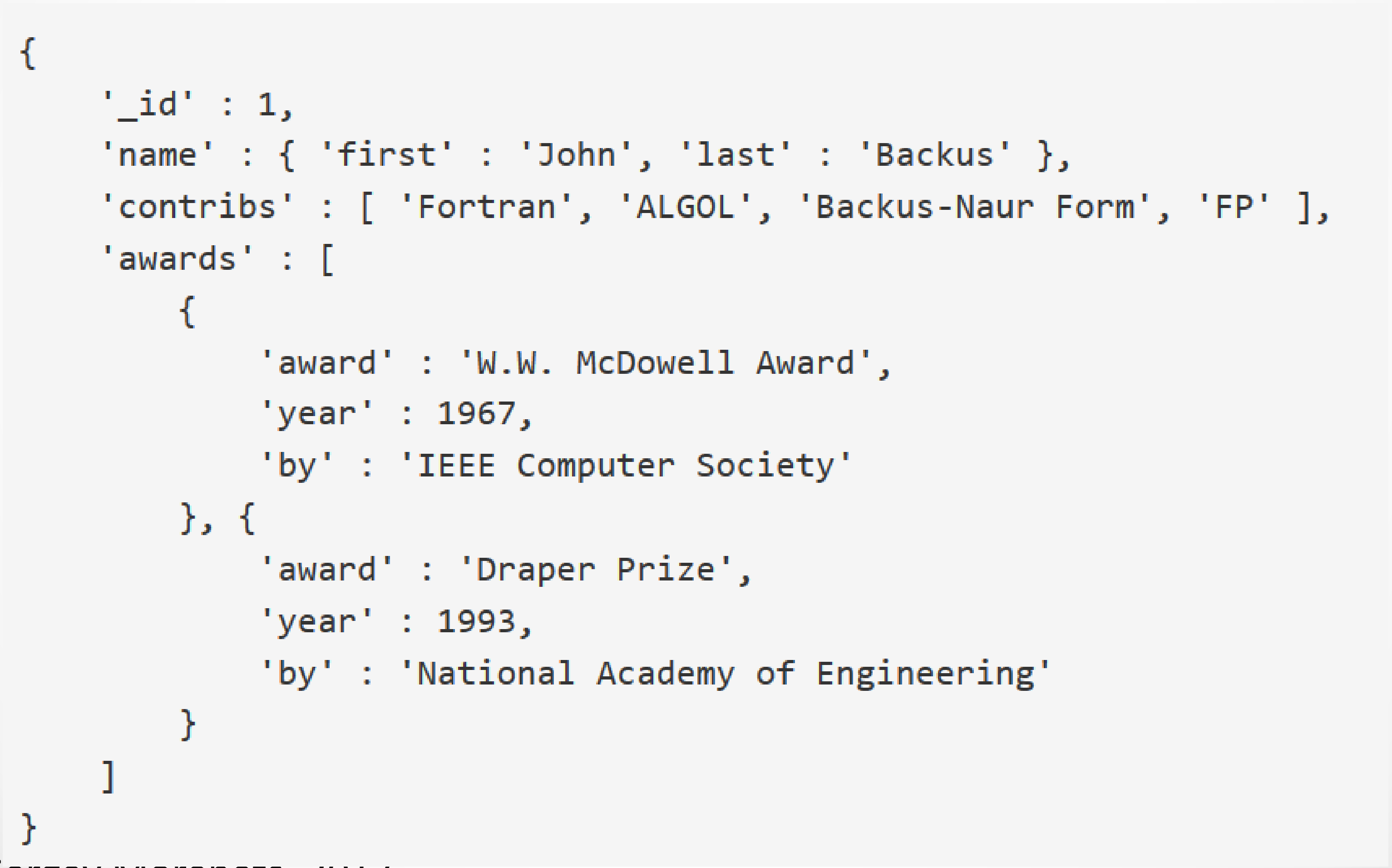
Task 13. Event sourcing and CQRS

1. Update Order application so that it will store read-only copy of the orders in Redis database.
2. Each time the order entity is updated you should store new copy of the order in Redis.
3. Change **OrderController** so that it fetch orders from Redis (operations findById, findAll).
4. Update automated tests.

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 3.4.10

MongoDB. JSON document



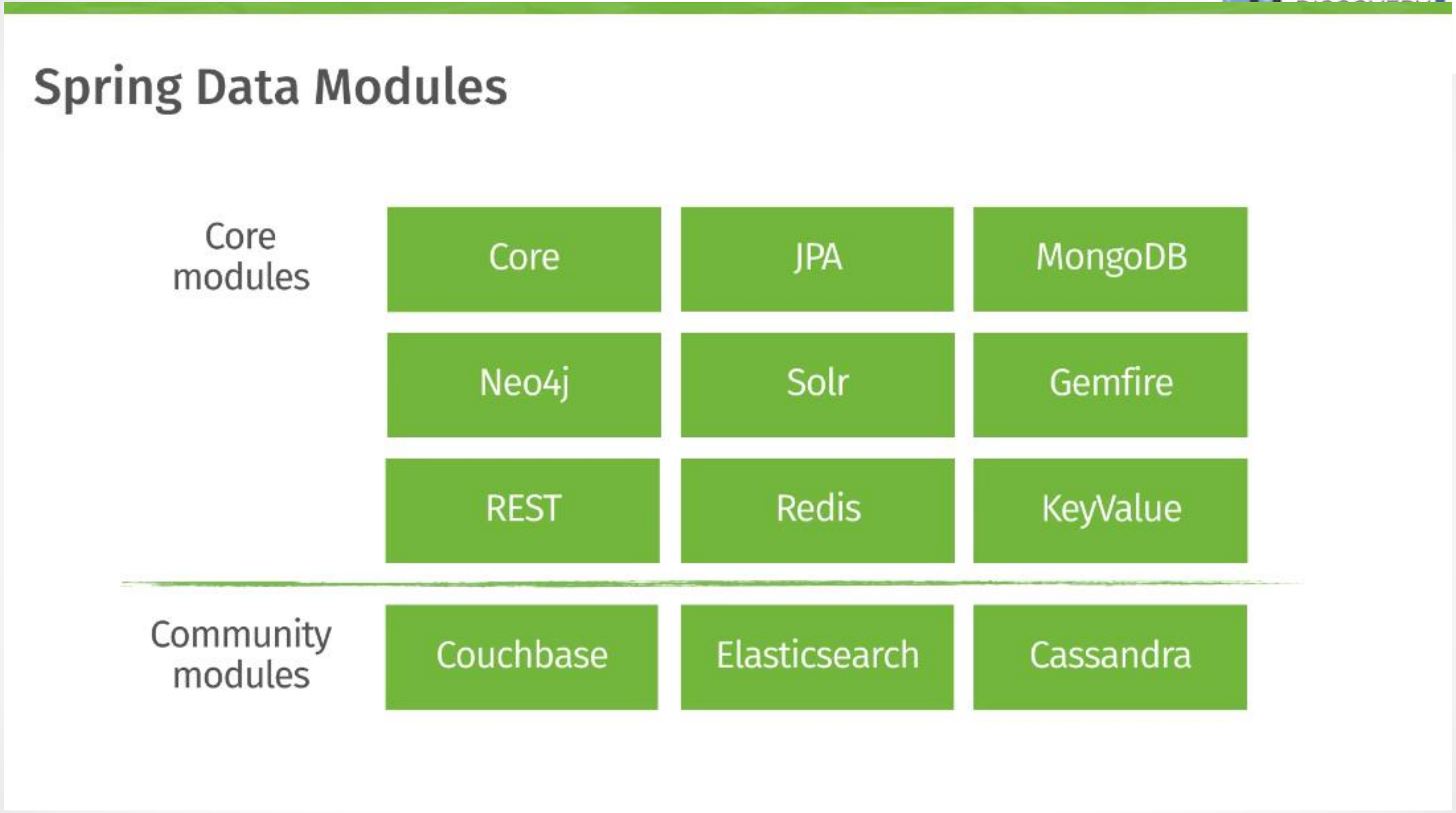
```
{
    '_id' : 1,
    'name' : { 'first' : 'John', 'last' : 'Backus' },
    'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
    'awards' : [
        {
            'award' : 'W.W. McDowell Award',
            'year' : 1967,
            'by' : 'IEEE Computer Society'
        }, {
   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

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 14. 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

Core modules	Core	JPA	MongoDB
	Neo4j	Solr	Gemfire
	REST	Redis	KeyValue
Community modules	Couchbase	Elasticsearch	Cassandra

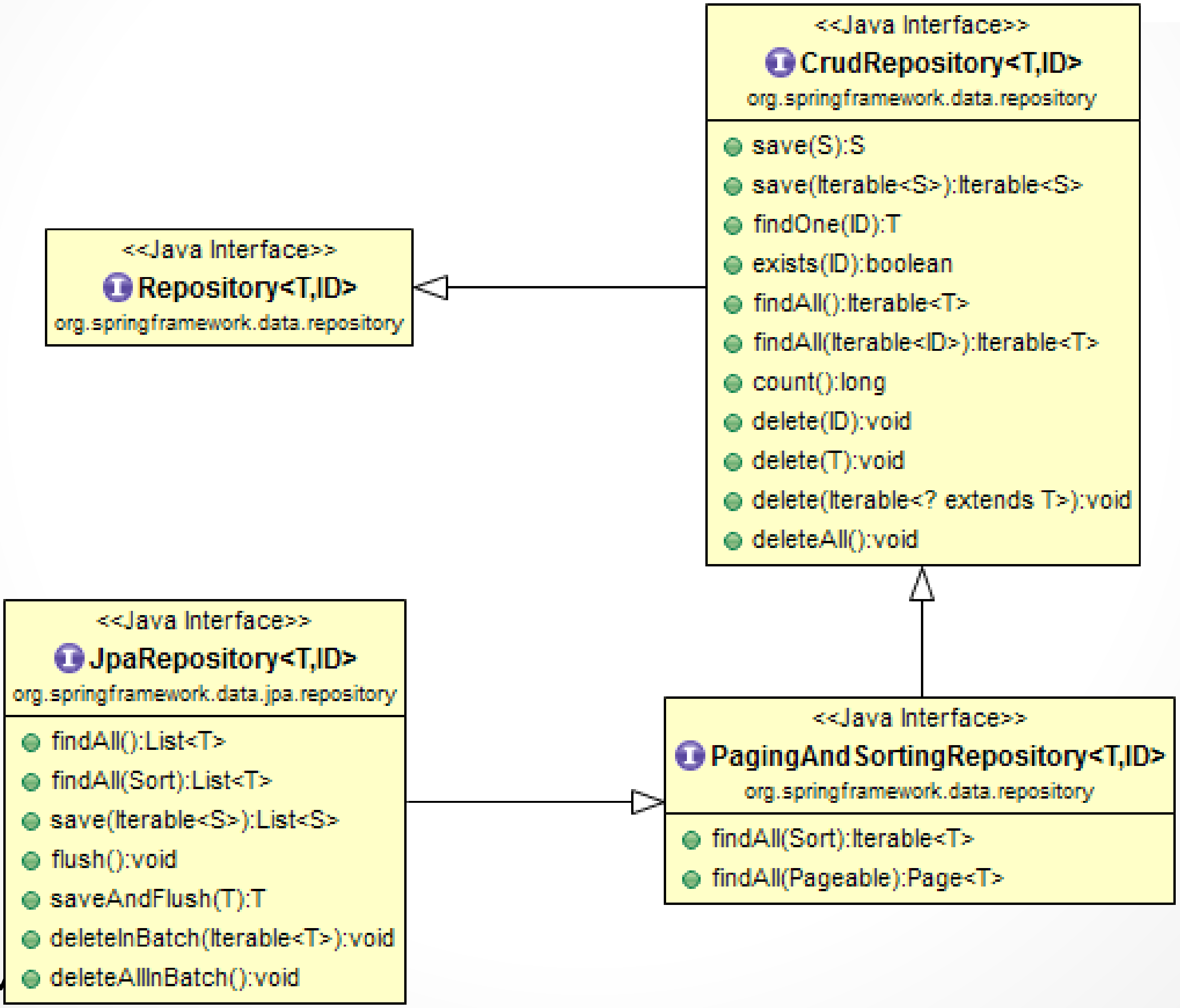
Spring Data MongoDB

- ✓ 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

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

Spring Data. Hierarchy



<<Java Interface>>
Repository<T,ID>
org.springframework.data.repository
<<Java Interface>>
CrudRepository<T,ID>
org.springframework.data.repository
save(S):S
save(Iterable<S>):Iterable<S>
findOne(ID):T
exists(ID):boolean
findAll():Iterable<T>
findAll(Iterable<ID>):Iterable<T>
count():long
delete(ID):void
delete(T):void
delete(Iterable<? extends T>):void
deleteAll():void
<<Java Interface>>
JpaRepository<T,ID>
org.springframework.data.jpa.repository
findAll():List<T>
findAll(Sort):List<T>
save(Iterable<S>):List<S>
flush():void
saveAndFlush(T):T
deleteInBatch(Iterable<T>):void
deleteAllInBatch():void
<<Java Interface>>
PagingAndSortingRepository<T,ID>
org.springframework.data.repository
findAll(Sort):Iterable<T>
findAll(Pageable):Page<T>

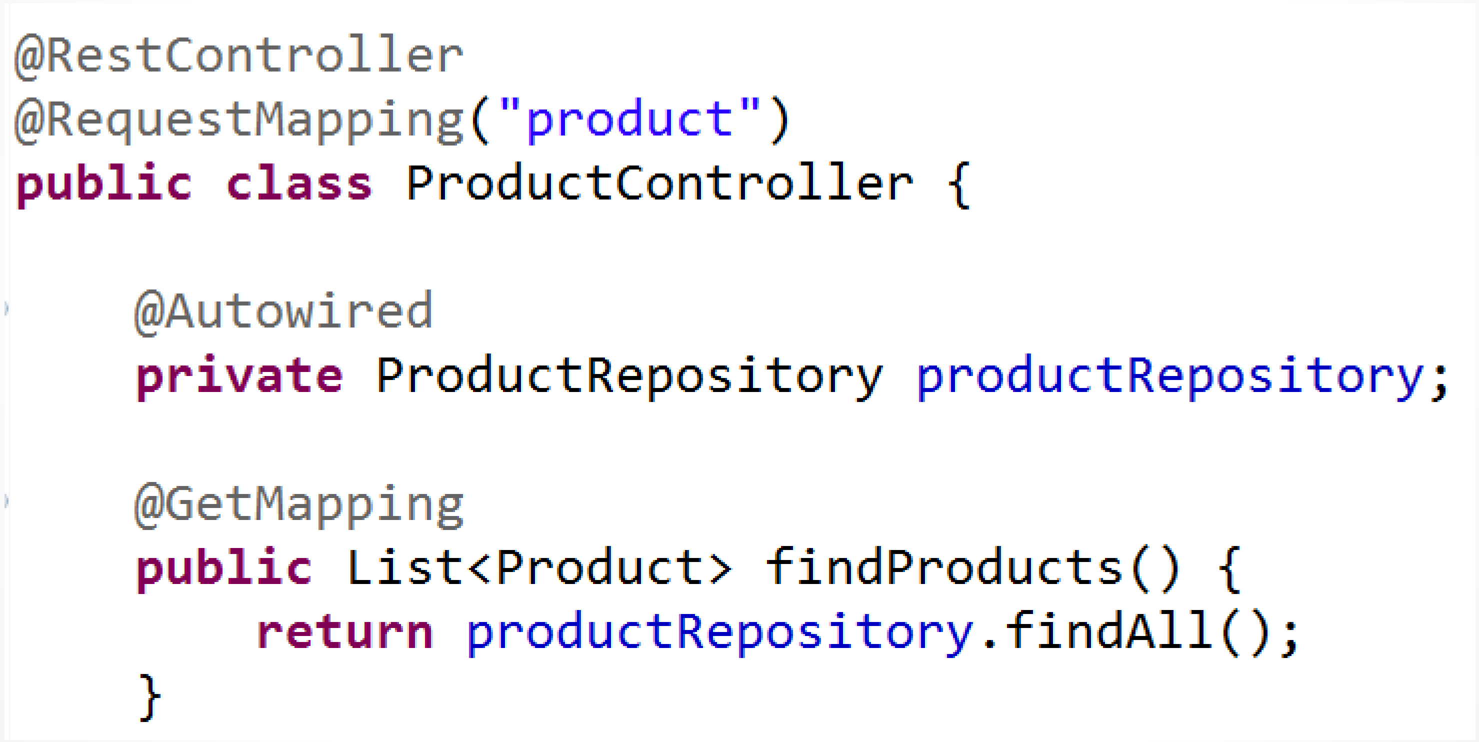
Spring Data MongoDB. Repositories

```java
public class Product {
    private int id;

    private String name;

    private double price;
```

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

Sergey Morenets, 2017

Spring Data MongoDB. Repositories



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

    @Autowired
    private ProductRepository productRepository;

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

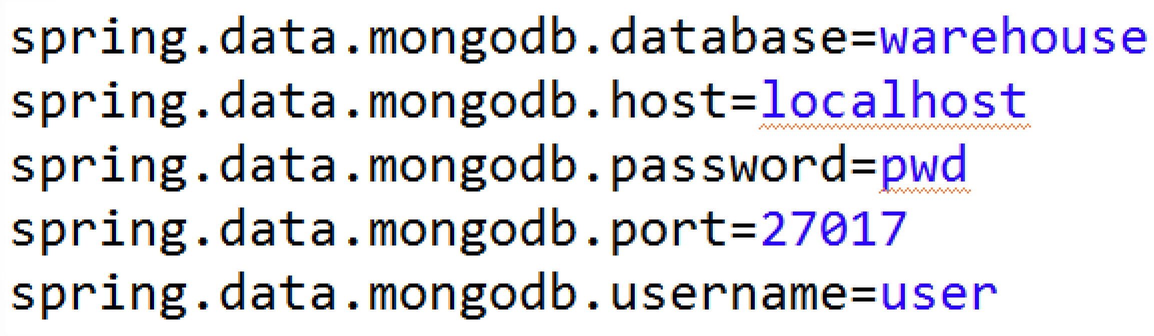
Spring Data MongoDB. Data Model

```
@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



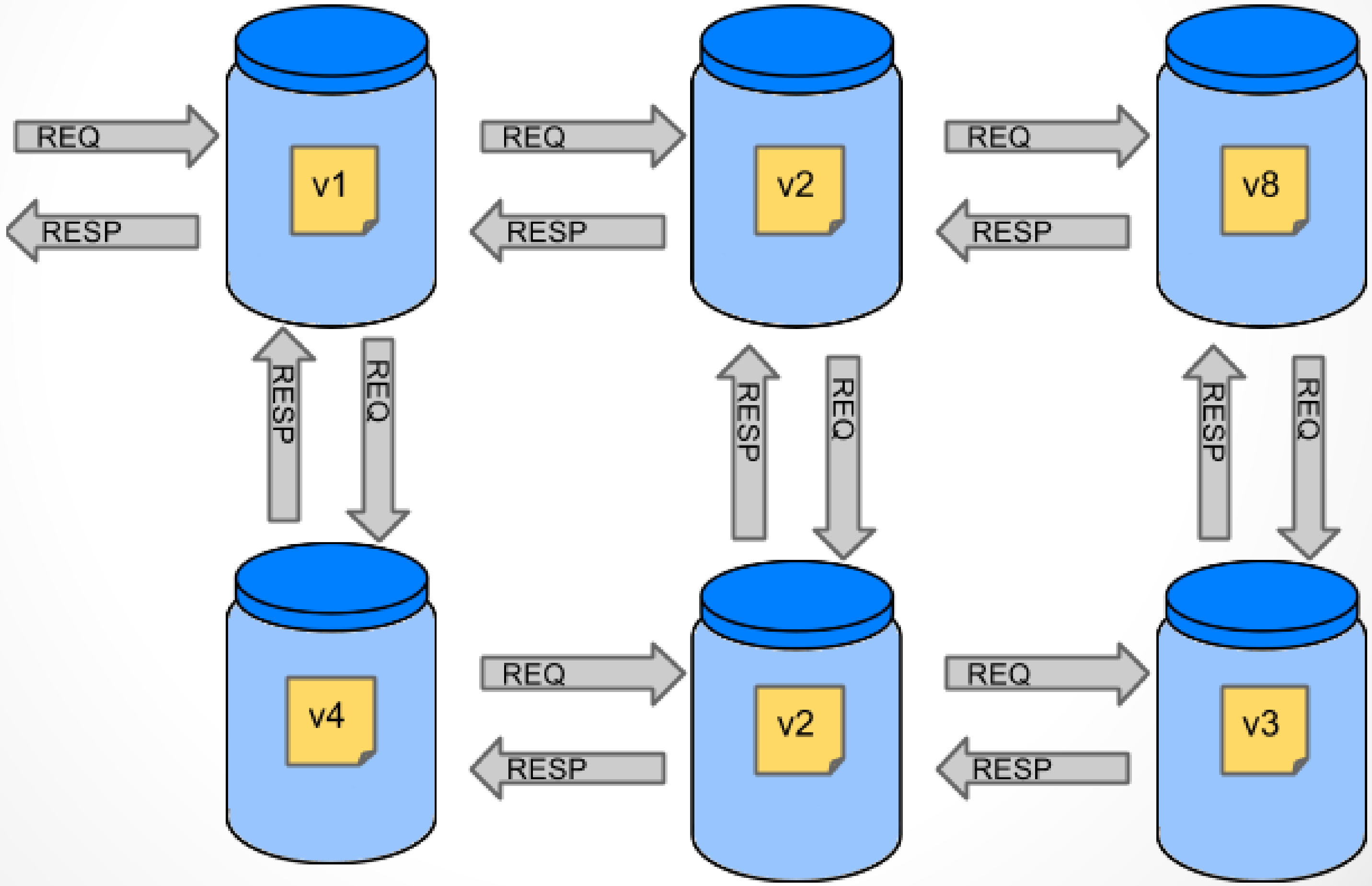
```
spring.data.mongodb.database=warehouse
spring.data.mongodb.host=localhost
spring.data.mongodb.password=pwd
spring.data.mongodb.port=27017
spring.data.mongodb.username=user
```

src/main/properties/application.properties

Task 15. Spring Data MongoDB

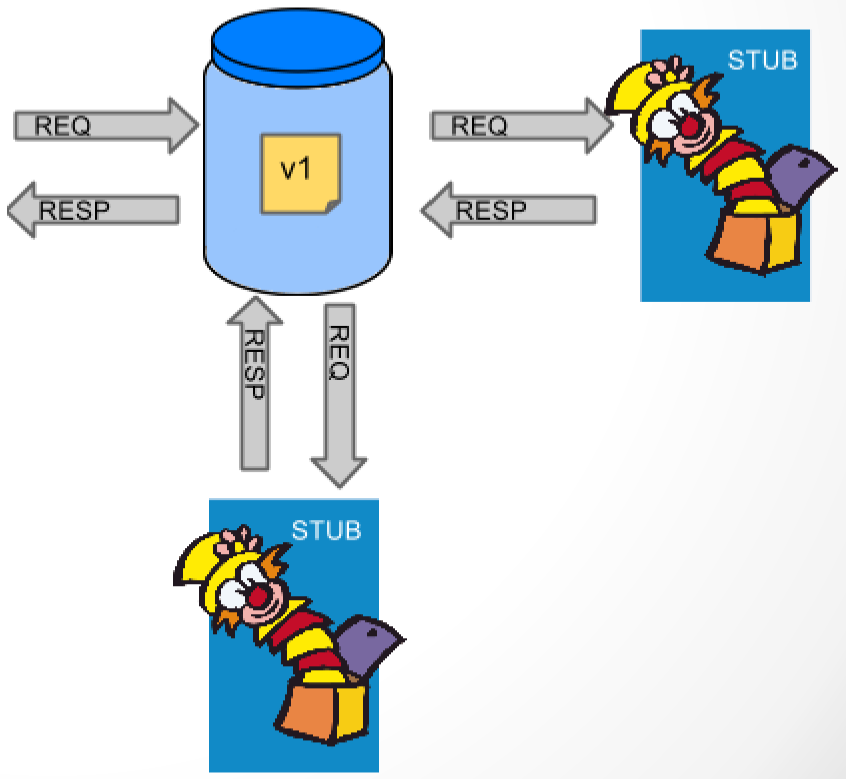
1. Add **Spring Data MongoDB** dependency to your project:
2. Add **application.properties** to **src/main/properties** folder
3. Let **NotificationRepository** extends **MongoReposiory** interface.
4. Create new **NotificationController** class, mark it with @RestController/GetMapping/@PostMapping annotation and implement **findAll/save** REST services. Auto-wire **NotificationRepository** interface.

Contract verifier



Sergey Morenets, 2017

Contract verifier. Stubs



WireMock

- ✓ Technology for mocking your API
- ✓ Simulator(mock-server) for HTTP-based API
- ✓ Request matching
- ✓ Record and playback
- ✓ Allows to simulate faults
- ✓ Supports Android

Spring and WireMock

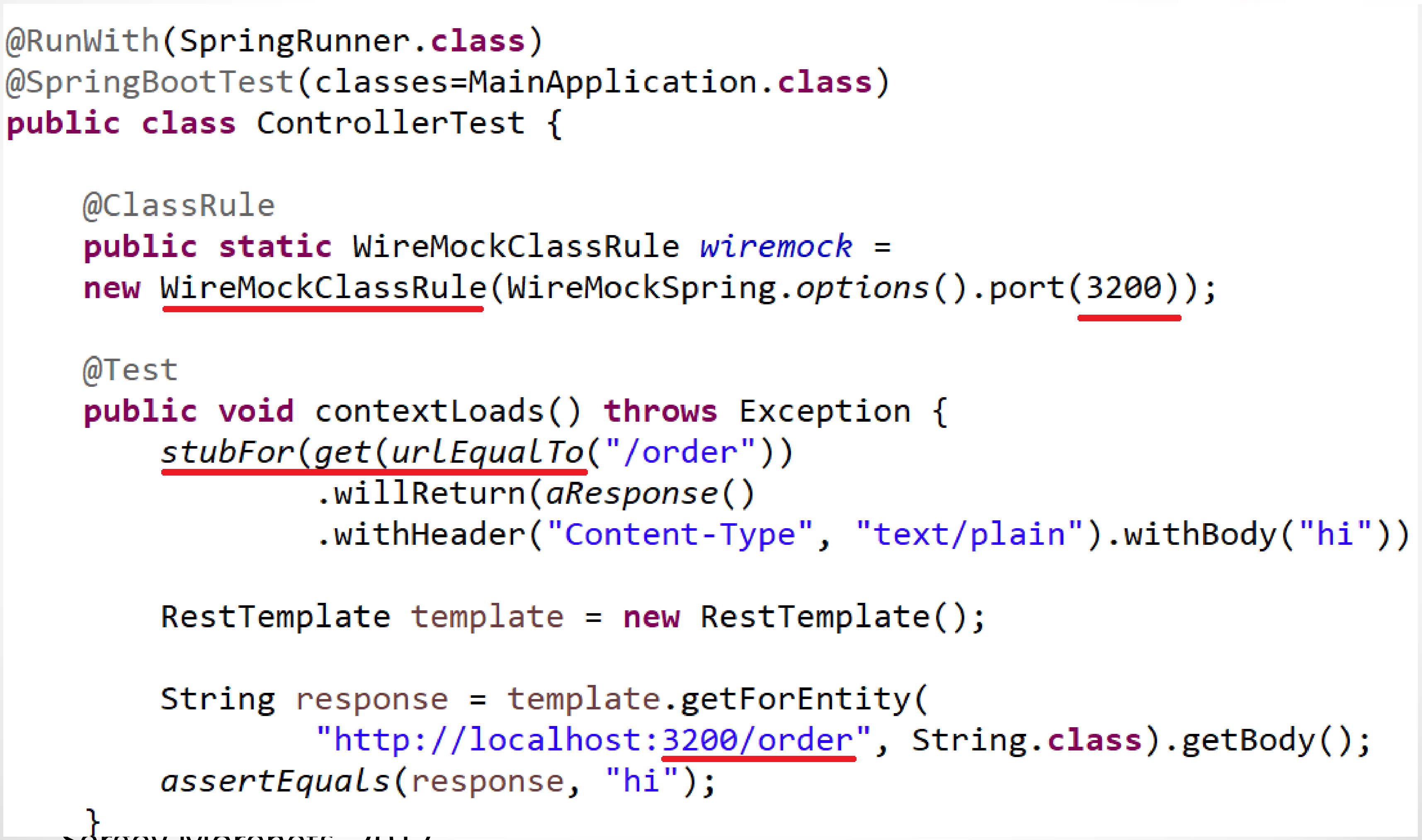
- ✓ Spring Cloud Contract WireMock allows to integrate Spring Boot and WireMock
- ✓ WireMock run as stub server
- ✓ Registering stubs using Java API or static JSON

Spring and WireMock. Dependency

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

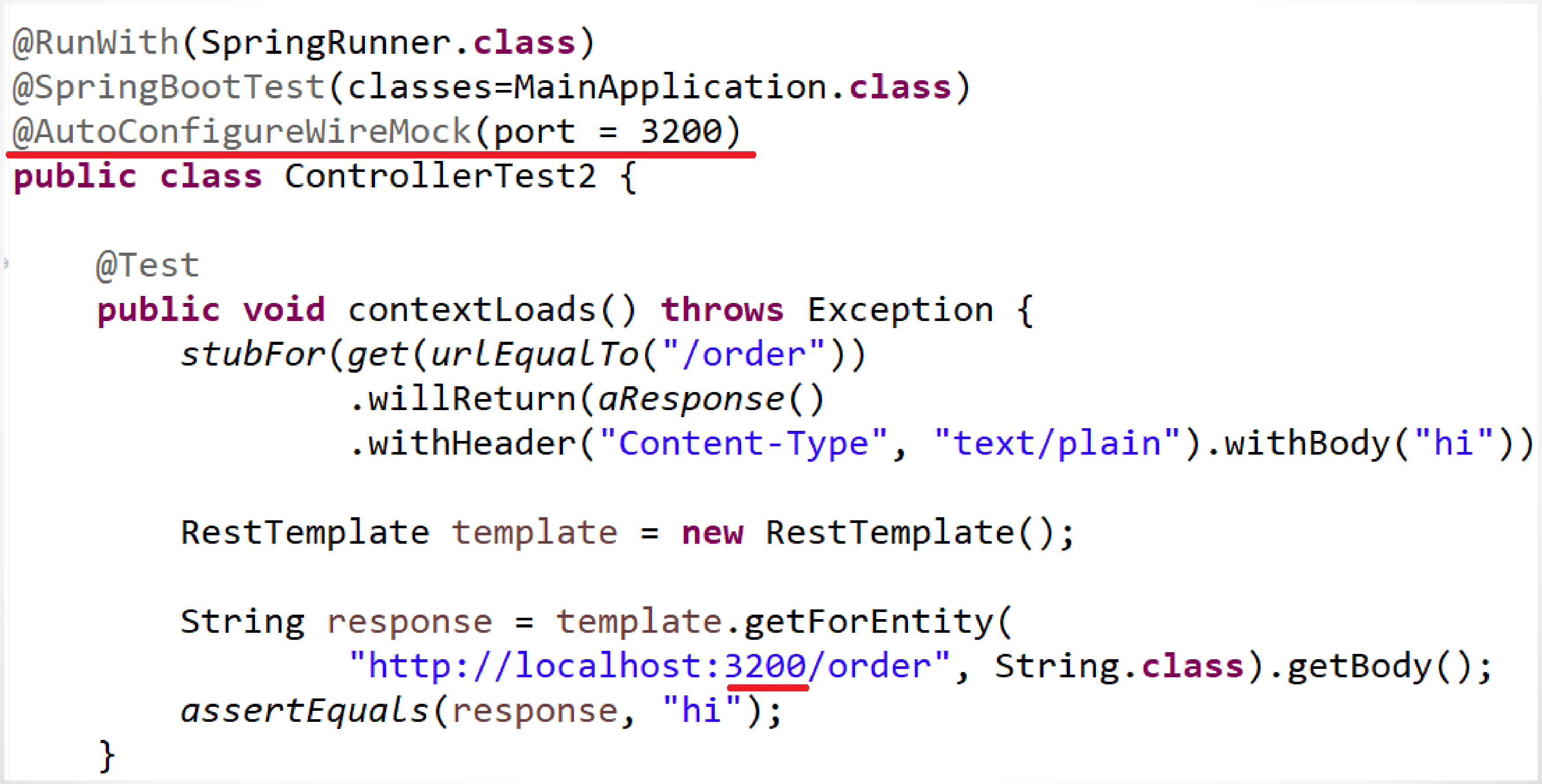
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-contract-wiremock</artifactId>
    <scope>test</scope>
</dependency>
```

WireMock. Class rules



```java
@RunWith(SpringRunner.class)
@SpringBootTest(classes=MainApplication.class)
public class ControllerTest {

    @ClassRule
    public static WireMockClassRule wiremock =
    new WireMockClassRule(WireMockSpring.options().port(3200));

    @Test
    public void contextLoads() throws Exception {
        stubFor(get(urlEqualTo("/order"))
                .willReturn(aResponse()
                .withHeader("Content-Type", "text/plain").withBody("hi"))

        RestTemplate template = new RestTemplate();

        String response = template.getForEntity(
                "http://localhost:3200/order", String.class).getBody();
        assertEquals(response, "hi");
    }
```

WireMock. Auto-configuration

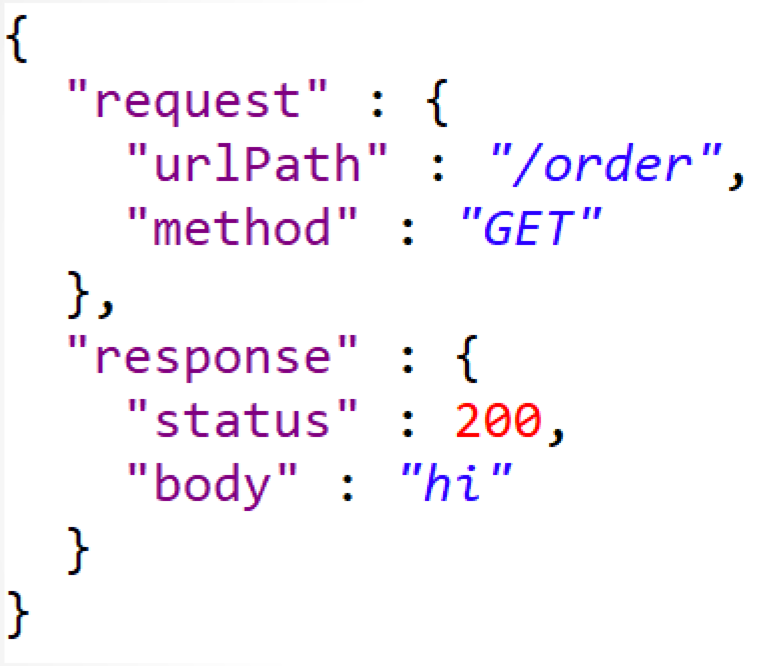


```
@RunWith(SpringRunner.class)
@SpringBootTest(classes=MainApplication.class)
@AutoConfigureWireMock(port = 3200)
public class ControllerTest2 {

    @Test
    public void contextLoads() throws Exception {
        stubFor(get(urlEqualTo("/order"))
                .willReturn(aResponse()
                .withHeader("Content-Type", "text/plain").withBody("hi"))

        RestTemplate template = new RestTemplate();

        String response = template.getForEntity(
                "http://localhost:3200/order", String.class).getBody();
        assertEquals(response, "hi");
    }
```

WireMock. Static responses

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = MainApplication.class)
public class ControllerTest3 {
    @Test
    public void testOrderRequest() throws Exception {
        RestTemplate template = new RestTemplate();
        MockRestServiceServer server = WireMockRestServiceServer
                .with(template).baseUrl("http://localhost:3200")
                .stubs("classpath:/responses/**/*.json").build();

        String response = template.getForEntity(
                "http://localhost:3200/order", String.class).getBody();
        assertEquals(response, "hi");
        server.verify();
    }
}
```

WireMock. Static responses



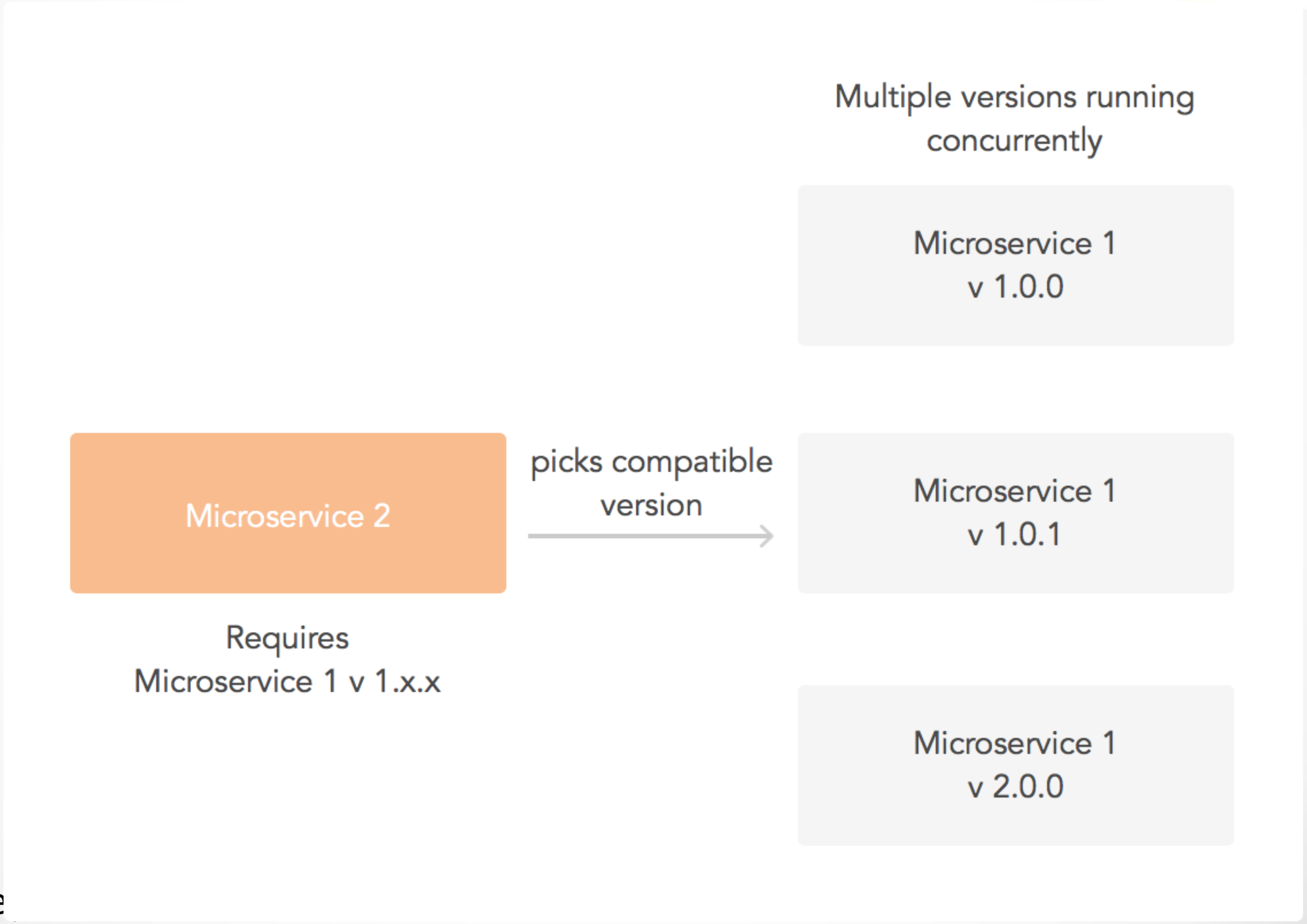
```
{
  "request" : {
    "urlPath" : "/order",
    "method" : "GET"
  },
  "response" : {
    "status" : 200,
    "body" : "hi"
  }
}
```

src/test/resources/responses/order.json

Task 16. WireMock

1. Add Spring Cloud WireMock dependency:
2. Write integration tests for **OrderController** using class rules and WireMock.
3. Write integration tests for **OrderController** using @AutoConfigureWireMock annotation.

Versioning


Multiple versions running concurrently
Microservice 1 v 1.0.0
Microservice 2
picks compatible version
Microservice 1 v 1.0.1
Requires Microservice 1 v 1.x.x
Microservice 1 v 2.0.0

Open-source PAAS

- ✓ OpenShift
- ✓ CloudFoundry
- ✓ DEIS

Centralized logging

- ✓ Elastic log
- ✓ Logstash
- ✓ Splunk
- ✓ Kibana
- ✓ Graphite

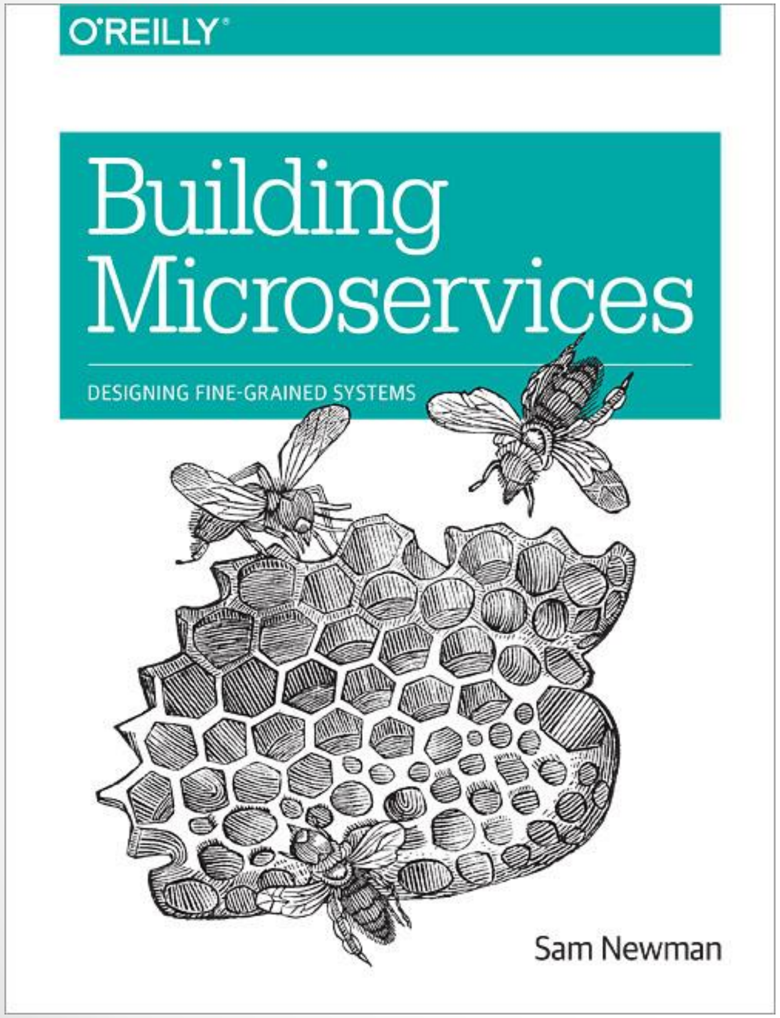
Hoverfly. Distributed testing

- ✓ Realistic API simulation (network failure, latency)
- ✓ Flexible customization with any programming language
- ✓ Export/share/edit API simulations
- ✓ Lightweight/ high-performance

Drawbacks

- ✓ Bad design solutions stored forever
- ✓ Complex queries & lookup

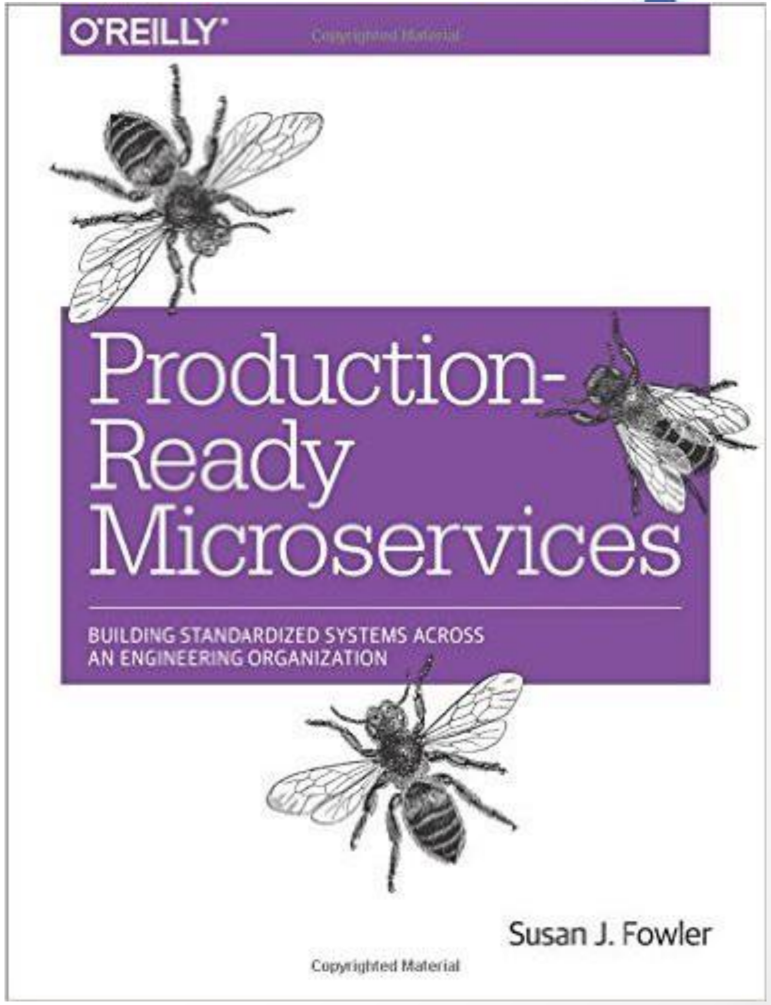
IT DISCOVERY


O'REILLY

Building Microservices

DESIGNING FINE-GRAINED SYSTEMS

Sam Newman


O'REILLY

Copyrighted Material

Production-Ready Microservices

BUILDING STANDARDIZED SYSTEMS ACROSS AN ENGINEERING ORGANIZATION

Susan J. Fowler

Copyrighted Material

That's all Folks!

- Sergey Morenets, sergey.morenets@gmail.com