Microservices development

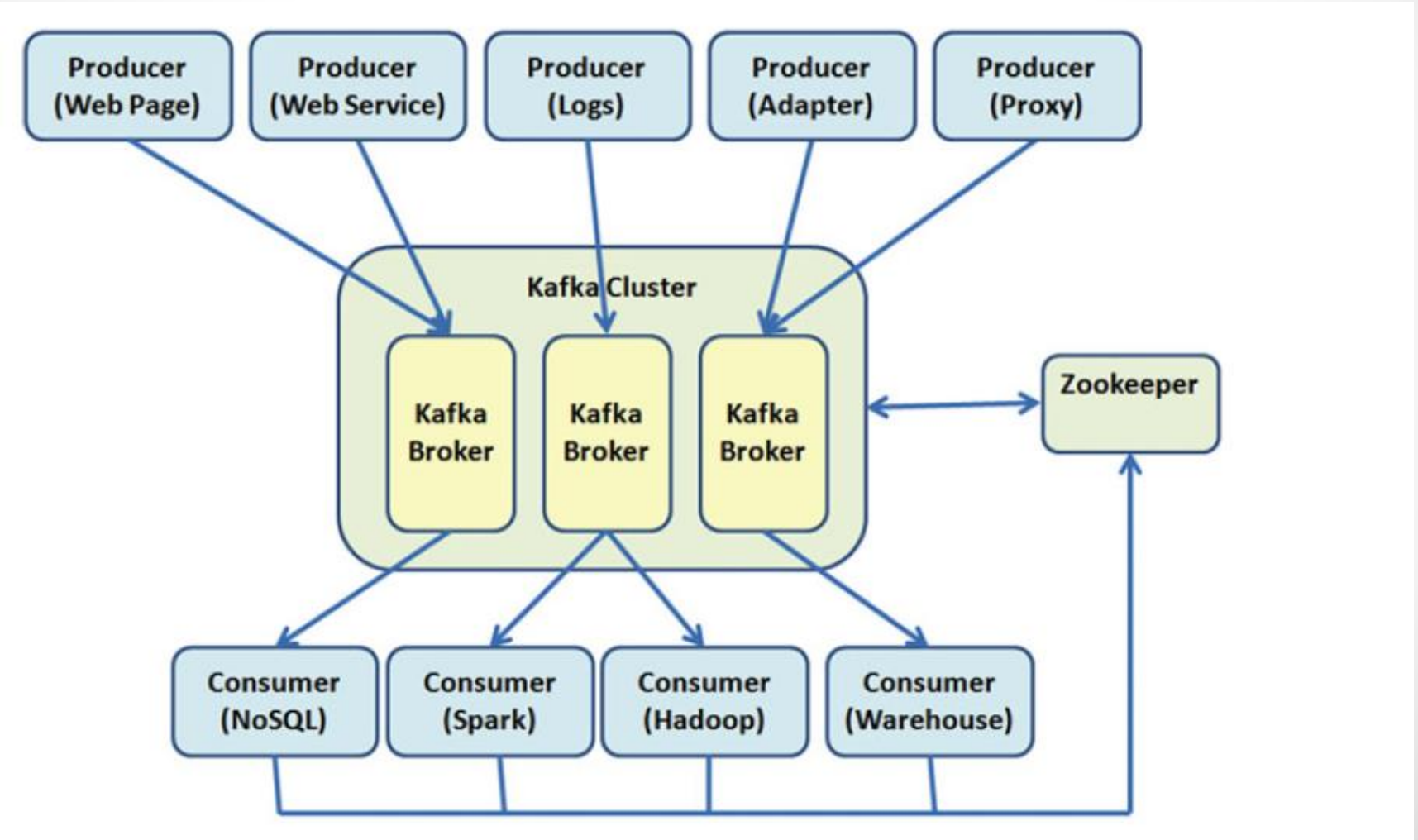
July, 20-21 2019

- Sergey Morenets, 2019

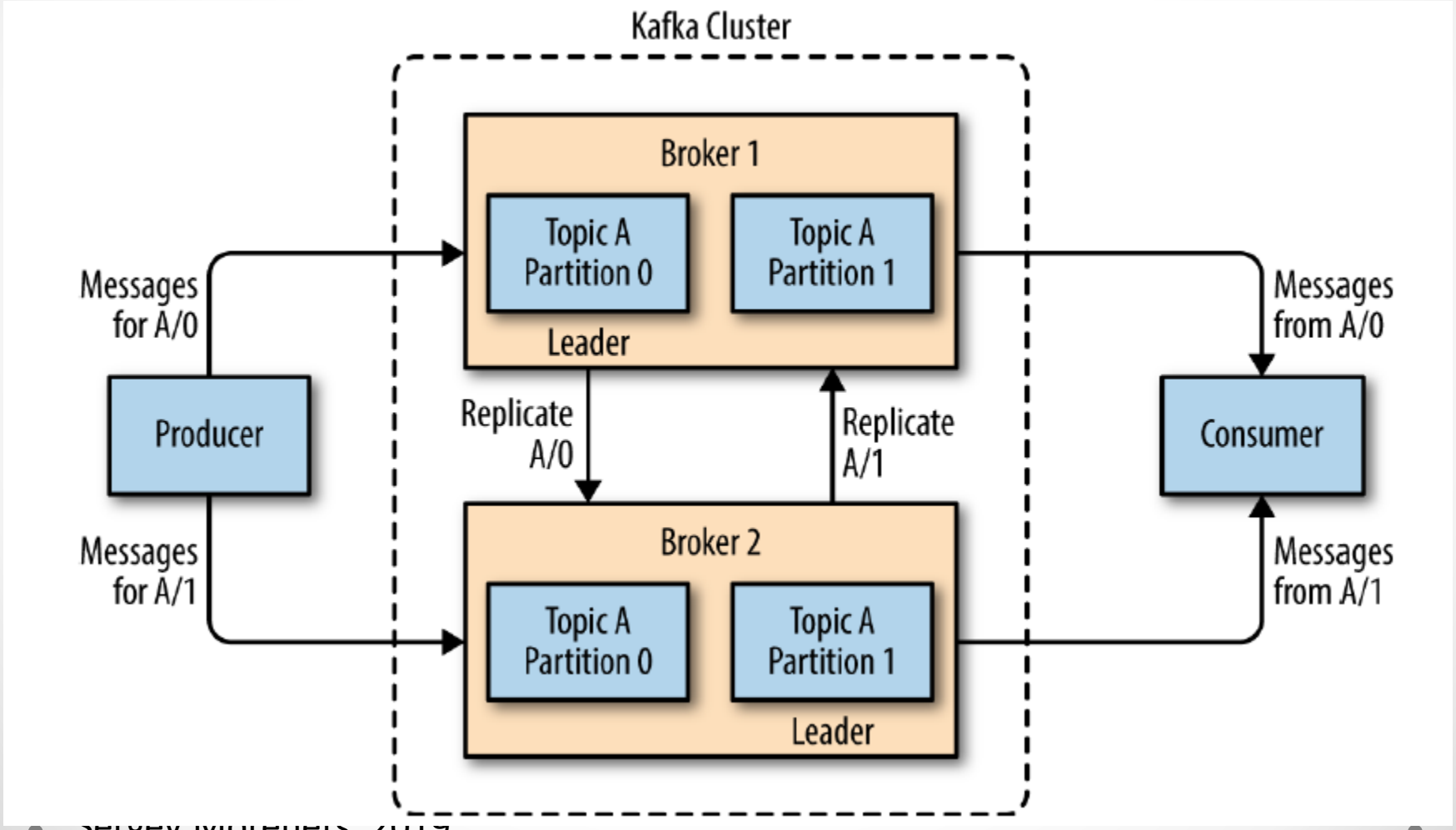
Kafka. Brokers

- Usually Kafka works inside clusters which contains servers (brokers)
- Each broker has numeric identifier
- Recommended 3 brokers per cluster for start
- Clients connects to a broker and other brokers get available for it
- Brokers are useful if topics have replication factor > 1 (recommended 2 or 3)

Kafka clustering



Apache Kafka. Replication



Kafka. Consistency and availability

- Messages will be appended to topic partition in the order they were sent
- Single consumer will read the messages in the order they are put in the log
- If producer doesn't wait for ack(the fastest approach) then it can lead to data loss. If producer waits for ack from leader then we random data loss can happen. If producer waits for ack from leader/replices then no data loss can happen
- Message cannot be lost if at least one replica is alive
- For efficiency, messages are written and consumed in batches and also compressed

Kafka. Message delivery

- Kafka has three modes to confirm delivery
- **At most once** mode persists consumer offset once message is received. If an error occurs during message processing then message will be lost
- **At least once** mode persists consumer offset once message has been processed. If an error occurs during message processing then message will be read again and in that case you will have to provide custom logic to avoid duplication
- **Exactly once** mode is hard achieve

Kafka. Message delivery

- Sergey Morenets, 2019

Kafka & Zookeeper

- Zookeeper stands for broker registry
- Broker management
- Assists in leader election for partitions
- Sends notification to Kafka (new topic, topic deletion, broker terminates)

Kafka stability

- ✓ Windows is not supported platform
- ✓ Broker may contain up to 4000 partitions and cluster up to 200000 partitions

Kafka / KAFKA-6188

Broker fails with FATAL Shutdown - log dirs have failed

Details

Type:	Bug	Status:	RESOLVED
Priority:	Blocker	Resolution:	Fixed
Affects Version/s:	1.0.0, 1.0.1	Fix Version/s:	None
Component/s:	clients, log		
Labels:	windows		
Environment:	Windows 10		

- Sergey Morenets, 2019

Kafka security

- Security is optional
- You can enable authentication between brokers and clients or between Zookeeper and brokers (SSL or SASL)
- Data encryption is supported
- Authorization for read/write operations

Kafka in production

- ✓ Used by New Relic, Uber, Square, LinkedIn, Netflix, AirBnB
- ✓ Cluster at New Relic processes 15 million of messages per second with aggregate rate 1 TBps
- ✓ Latency less than 10ms
- ✓ Can scale up to 100 brokers
- ✓ Currently processes 1.1 trillion messages per day by 1400 brokers

Kafka. Best practices

- ✓ receive.buffer.bytes stores socket buffer size (64 kb). For data-intensive operations it should be increased
- ✓ Use wait for acknowledgment mode for producers not to lose messages
- ✓ Adjust number of producer **retries** (3 by default) for data-intensive operations
- ✓ *buffer.memory* and *batch.size* should be increased for data-intensive operations
- ✓ Configure Kafka logging to filter only important events
- ✓ Cleanup unused topics
- ✓ Use random partitioning unless if you have other project requirements

Task 7. Kafka. Clustering and fault-tolerance

1. Run container with Kafka/Zookeeper
2. Open new terminal and run this command: *docker exec -it spotify_kafka bash*
3. Navigate to /opt/ kafka_2.11-0.10.1.0 folder
4. Copy config/server.properties into server2.properties. Change **broker.id** to 1, also update log.dirs property. Add new line

port=9093

Spring Kafka

- Wrapper over Apache Kafka Client
- Introduced template abstraction (**KafkaTemplate**) for messaging operations
- Supports message-driven approach with @KafkaListener operation
- Transactions support since client 0.11
- Synchronous and asynchronous message delivery
- Requires Spring Framework 5, Spring Integration 3.0 and Kafka client 0.11

Spring Kafka. Dependencies


```
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
  <version>2.0.1</version>
  <scope>compile</scope>
</dependency>
```

Pure Java libary

Spring integration

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

Spring Kafka. Listeners


```java
@Configuration
@EnableKafka
public class KafkaConfiguration {

    @KafkaListener(topics = "orders")
    public void listen(ConsumerRecord<?, ?> record) {
        System.out.println(record.offset());
        System.out.println(record.partition());
        System.out.println(record.topic());
        System.out.println(record.toString());
    }
```

Requires to listen to the messages

application.yml

```yaml
spring:
  kafka:
    consumer:
      group-id: app
      auto-offset-reset: earliest
```

Which offset to use if initial offset doesn't exist

Also exception, latest

Spring Kafka. Listeners & partitions


```
@KafkaListener(id = "myGroupId",
        topicPartitions = { @TopicPartition(
                topic = "data1", partitions = { "0", "2" }),
        @TopicPartition(topic = "data2",
        partitions = "1",
        partitionOffsets = @PartitionOffset(
                partition = "1", initialOffset = "100")) })
public void listen(ConsumerRecord<?, ?> record) {
```

Spring Kafka. Metadata


```
@KafkaListener(topics = "orders")
public void listen(@Payload String item,
        @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) Integer key,
        @Header(KafkaHeaders.RECEIVED_PARTITION_ID) int partition,
        @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
        @Header(KafkaHeaders.RECEIVED_TIMESTAMP) long ts,
        Acknowledgment ack) {
    try {

    } finally {
        ack.acknowledge();
    }
}
```

Manual message acknowledgment, all the previous messages in the partition assumed to be processed

Spring Kafka. Batch processing


```
@Bean
public KafkaListenerContainerFactory<?> batchFactory() {
    ConcurrentKafkaListenerContainerFactory<Integer, String>
            factory = new ConcurrentKafkaListenerContainerFactory<>();
    factory.setConsumerFactory(consumerFactory());
    factory.setBatchListener(true);
    return factory;
}
```

```
@KafkaListener(topics = "orders",
        containerFactory = "batchFactory")
public void listen(List<Message<?>> messages,
        Acknowledgment ack) {
}
```

- Sergey Morenets, 2019

KafkaTemplate. Producers


```
@Autowired
private KafkaTemplate<String, String> template;

@PostMapping
public void send() {
    template.send("orders", "message");
    template.send("orders", "key", "message_with_key");
    template.send("orders2", 0, "key2", "message2_with_key");

    template.flush();
}
```

Topic name

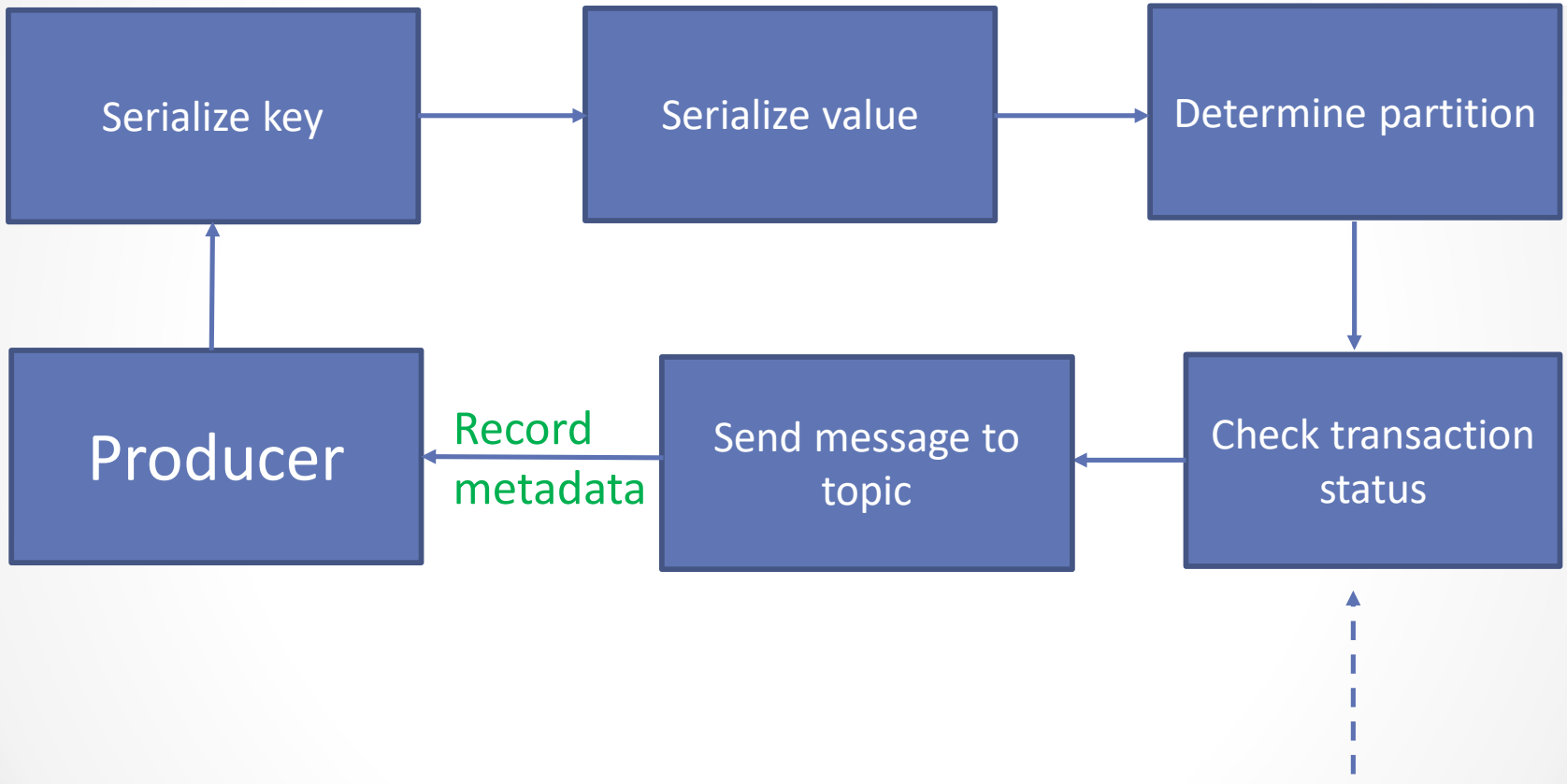
Async sending

Partition

```
var future = template.send("orders", 1,
        "key3", "message3_with_key");
future.addCallback(result -> System.out.println(result),
        ex -> Log.error(ex.getMessage(), ex));
```

Success callback

Sending message flow



- Sergey Morenets, 2019

Partitioning


```
public class RandomPartitioner implements Partitioner{

    @Override
    public int partition(String topic, Object key,
            byte[] keyBytes, Object value, byte[] valueBytes,
            Cluster cluster) {
        if(key == null) {
            return 0;
        } else {
            int numPartitions = cluster.
                    partitionCountForTopic(topic);
            return key.hashCode() % numPartitions;
        }
    }
```

Kafka. Producer configuration


```
@Configuration
public class KafkaConfiguration {
    @Bean
    public ProducerFactory<String, String> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(
          ProducerConfig.PARTITIONER_CLASS_CONFIG,
          RandomPartitioner.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

org.apache.kafka.common.config.ConfigException: Missing required configuration "key.serializer" which has no default value.

Kafka. Producer configuration


```
Map<String, Object> configProps = new HashMap<>();
configProps.put(
  ProducerConfig.PARTITIONER_CLASS_CONFIG,
  RandomPartitioner.class);
configProps.put(
        ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
        StringSerializer.class);
configProps.put(
        ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
        StringSerializer.class);
configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
        "localhost:9092");
return new DefaultKafkaProducerFactory<>(configProps);
```

Kafka. Producer configuration


```
@Bean
public ProducerFactory<String, String> producerFactory() {
    Map<String, Object> configProps = new HashMap<>();
    configProps.put(
      ProducerConfig.PARTITIONER_CLASS_CONFIG,
      RandomPartitioner.class);
    configProps.put(
            ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
            StringSerializer.class);
    configProps.put(
            ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
            StringSerializer.class);
    return new DefaultKafkaProducerFactory<>(configProps);
}
```

org.apache.kafka.common.config.ConfigException: No resolvable bootstrap urls given in bootstrap.servers

Kafka. Producer & custom value

```
var future = template.send("orders", 0,
        "key3", new Book());
```

java.lang.ClassCastException: it.discovery.microservice.book.Book cannot be cast to java.base/java.lang.String

```
Map<String, Object> configProps = new HashMap<>();
configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
        StringSerializer.class);
configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
        JsonSerializer.class);
```

Uses Jackson serializer

Message delivery

application.yml

```
spring:
  kafka:
    producer:
      retries: 3
      acks: 1
```

Controls acknowledgment of the message:

0 – no confirmation from leader
1 – confirmation from leader
all – confirmation after successful replication

Testing Kafka applications

- Start embedded Kafka/Zookeeper servers on random port before each tests

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka-test</artifactId>
    <scope>test</scope>
</dependency>
```

Testing Kafka applications


```
@ExtendWith(SpringExtension.class)
@EmbeddedKafka(topics = { "topic1", "topic2" },
               partitions = 1)
public class KafkaTest {
    @Autowired
    private EmbeddedKafkaBroker embeddedKafka;
```

Starts embedded Kafka/Zookeeper

Stream processing

- Stream processor is software component that receives and sends stream of data
- Apache Storm, Apache Flink or Kafka Streams

Task 8. Spring Kafka

1. Add new dependency to your project:
2. Add Kafka configuration class; put **@EnableKafka** annotation on it.
3. Update (or create) *src/main/resources/application.yml* file and put group-id and auto-offset-reset properties.
4. Create class **KafkaListener** that will listens for messages in **library** topic and print it to the console. Put **@KafkaListener** annotation on it.

Event sourcing

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

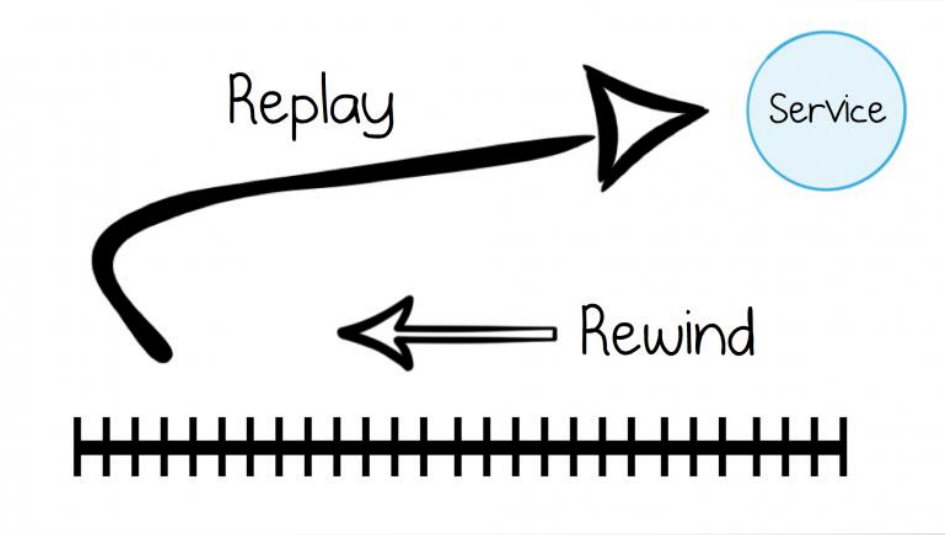
- Sergey Morenets, 2019

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

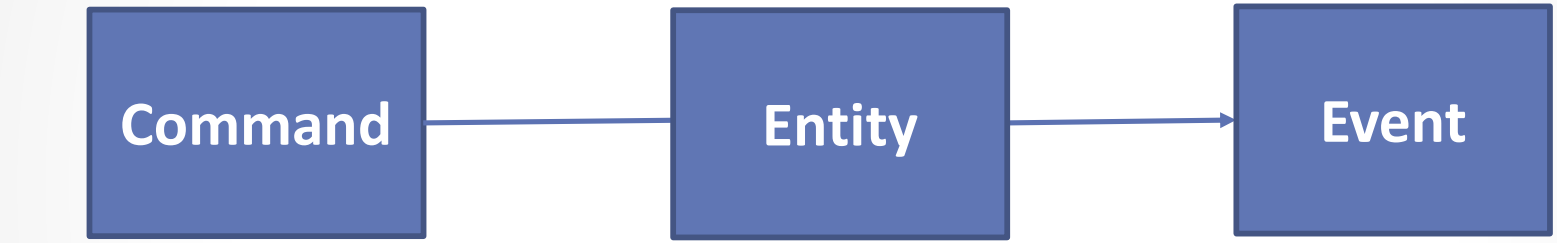
Event sourcing

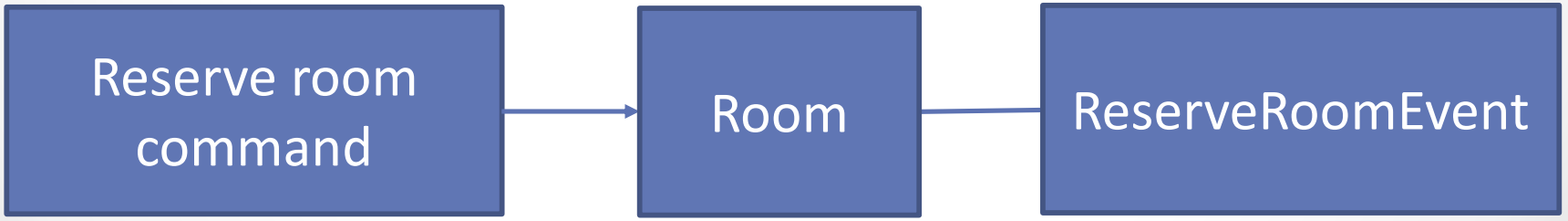
- ✓ You can cache temporal application state in memory
- ✓ UI requires projects of all the event log
- ✓ Simplifies data consistency
- ✓ You cannot remove event types
- ✓ You can replay events log from production and reproduce a bug



- Sergey Morenets, 2019

Event sourcing





- Sergey Morenets, 2019

Commands


```
@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

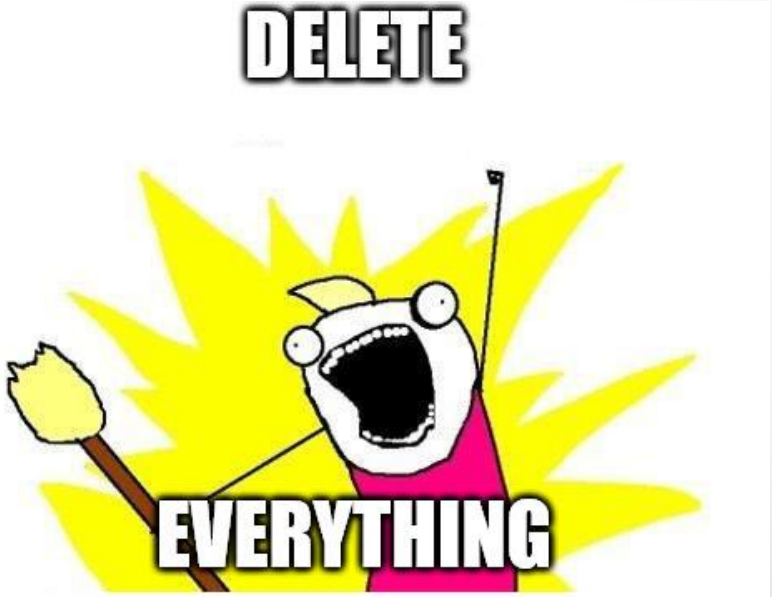

```
public List<BaseEvent> process(WithdrawMoneyCommand cmd) {
    if (cmd.getAmount() > balance) {
        return Arrays.asList(
                new WithdrawMoneyEvent(cmd.getId(),
                        cmd.getAmount()));
    } else {
        return Arrays.asList(
                new WithdrawMoneyFailureEvent(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

Event purge/compaction

- ✓ Storage is not free and not unlimited
- ✓ Security purposes
- ✓ GDPR rules
- ✓ Compaction means keeping most recent data



Task 9. 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

Session store

Caching

Geo indexing

Queues

Analytics

- Sergey Morenets, 2019

Redis

- Created in 2009 by Salvatore Sanfillipo as **Re**mote **D**isctionary **S**erver
- 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**(since v3).
- Supports monitoring and metrics
- High-availability, fault-tolerance and sharding
- Supports transactions with multi-commands

redis

Redis 5.x features

- ✓ Redis streams
- ✓ Sorted sets
- ✓ Redis modules
- ✓ Improved performance

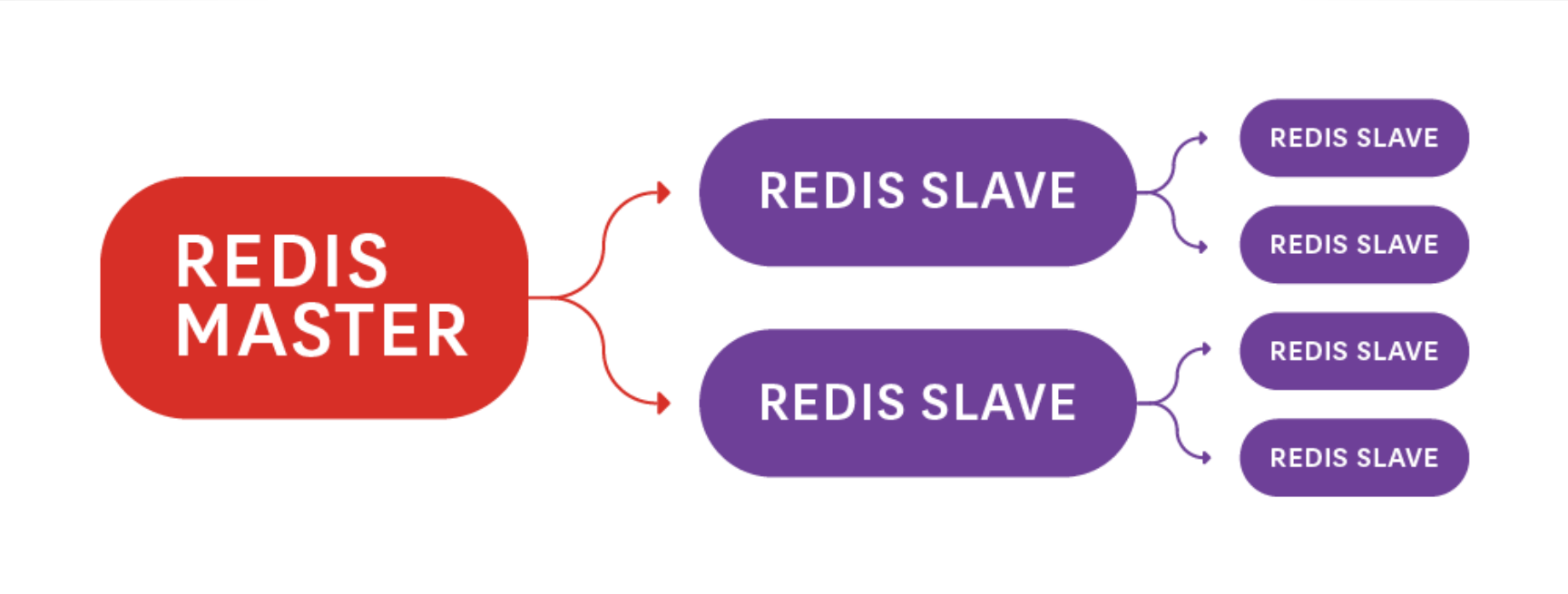
redis

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

Redis replication



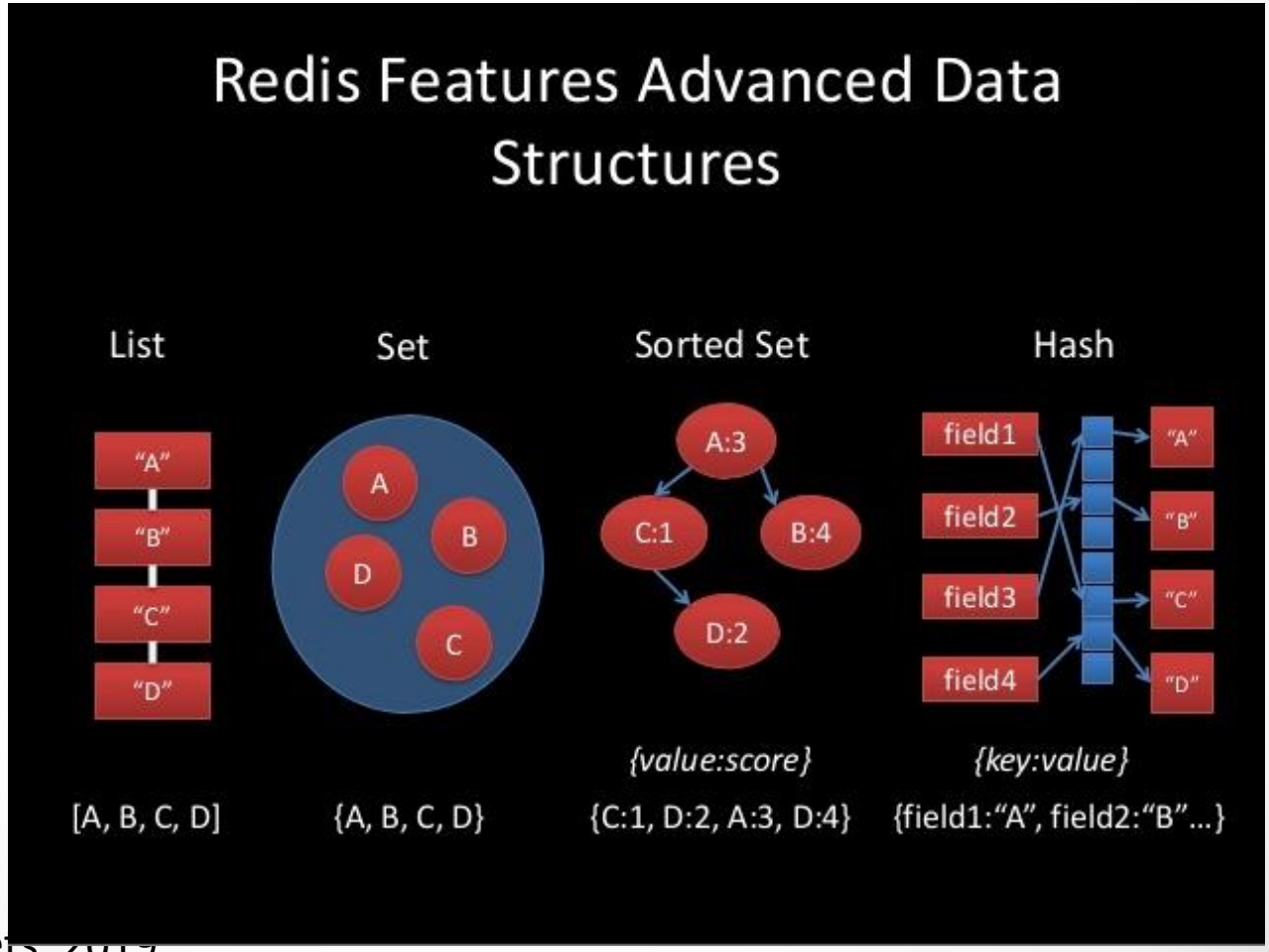
- Sergey Morenets, 2019

Redis replication

- Replication is asynchronous and non-blocking
- Master can have multiple slaves (replicas)
- Slaves are able to connect to other slaves
- Replication is used to add more scalability, data safety or high availability
- Slaves usually perform partial synchronization but also can do full synchronization

redis

Redis data types



- Sergey Morenets, 2019

Redis security

- Main configuration file is **redis.conf**
- If no **bind** instructions are present all the clients can connect
- **bind 127.0.0.1** allows only local connections
- Supports authentication using master password
- No encryption supported

Task 10. 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>
</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. Value operations


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

@GetMapping
public String save() {
    ValueOperations<String, Object> operations =
            redisTemplate.opsForValue();
    operations.set("key", "value");

    System.out.println(operations.get("key"));

    operations.increment("counter", 1);
    operations.set("key", "value", 1, TimeUnit.DAYS);

    return "success";
}
```

RedisTemplate. Hash operations


```
@Autowired
private RedisConnectionFactory connectionFactory;

@GetMapping
public void save() {
    HashOperations<String, String, String> operations =
            redisTemplate.opsForHash();
    operations.put("user1", "name", "John");

    operations.keys("user1").forEach(System.out::println);

    System.out.println(operations.get("user1", "name"));

    RedisAtomicLong counter = new
            RedisAtomicLong("counter2", connectionFactory, 0);
    System.out.println(counter.incrementAndGet());
```

RedisTemplate. Storing object


```
@Autowired
private RedisConnectionFactory connectionFactory;

@GetMapping
public void save() {
    Order order = new Order();

    ObjectHashMapper mapper = new ObjectHashMapper();
    Map<byte[], byte[]> hash = mapper.toHash(order);

    RedisConnection conn = connectionFactory.getConnection();
    conn.hMSet("order:1".getBytes(), hash);

    Order newOrder = (Order)mapper.fromHash(
            conn.hGetAll("order:1".getBytes()));
    System.out.println(newOrder);
```

Redis. String operations

```
127.0.0.1:6379> keys *
1) "\xac\xed\x00\x05t\x00\acounter"
2) "text"
3) "\xac\xed\x00\x05t\x00\x03key"
127.0.0.1:6379> exit
```

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

    StringRedisSerializer serializer = new StringRedisSerializer();
    template.setKeySerializer(serializer);
    template.setValueSerializer(serializer);
    template.setHashKeySerializer(serializer);
    template.setHashValueSerializer(serializer);
    return template;
}
```

Spring Data Redis. Security


```
@Configuration
public class RedisConfiguration {

    @Value("${spring.redis.password}")
    private String password;

    @Bean
    public JedisConnectionFactory connFactory() {
        RedisStandaloneConfiguration configuration =
                new RedisStandaloneConfiguration();
        configuration.setPassword(
                RedisPassword.of(password));
        return new JedisConnectionFactory(configuration);
    }
```

Redis. Spring Data Repositories


```java
@Data
@RedisHash("orders")
public class Order {
    private int id;

    private List<OrderItem> items;
```

```java
public interface OrderRepository
        extends CrudRepository<Order, Integer>{

}
```

- Sergey Morenets, 2019

Redis. Spring Data Repositories

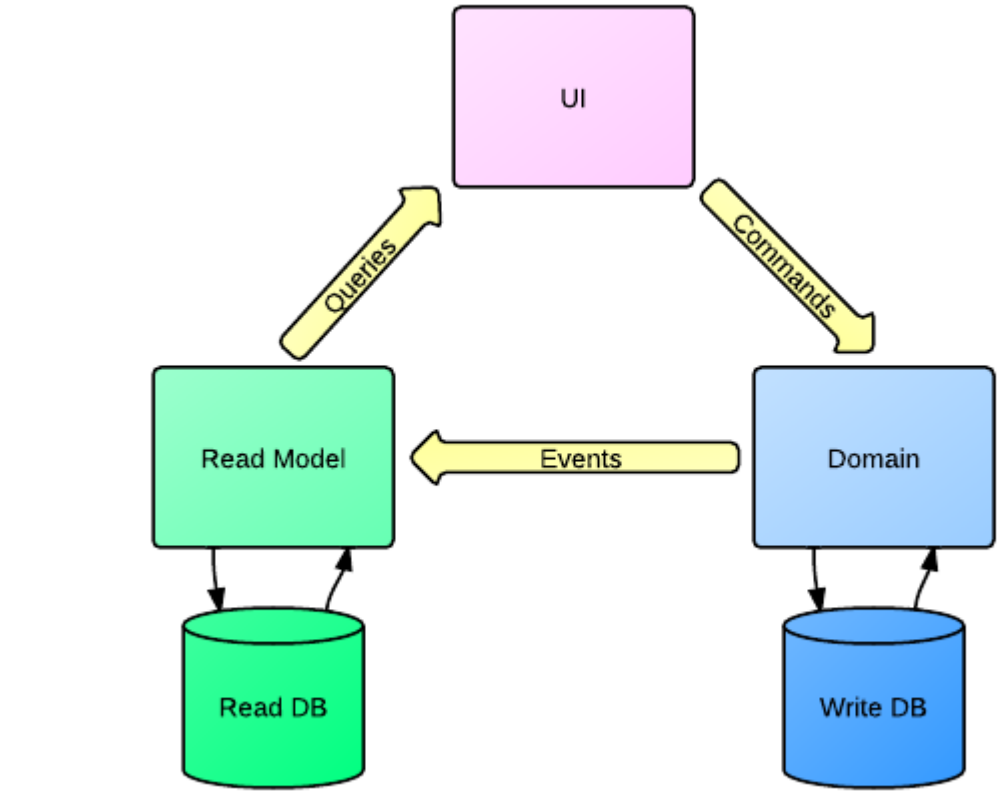

```java
@SpringBootApplication
@EnableRedisRepositories
public class MainApplication {
    public static void main(String[] args) {
        SpringApplication.run(MainApplication.class, args);
    }
}
```

```java
@Autowired
private OrderRepository orderRepository;

@GetMapping
public void save() {
    Order order = new Order();
    order.setId(1);
    orderRepository.save(order);
```

Task 11. Spring Data Redis

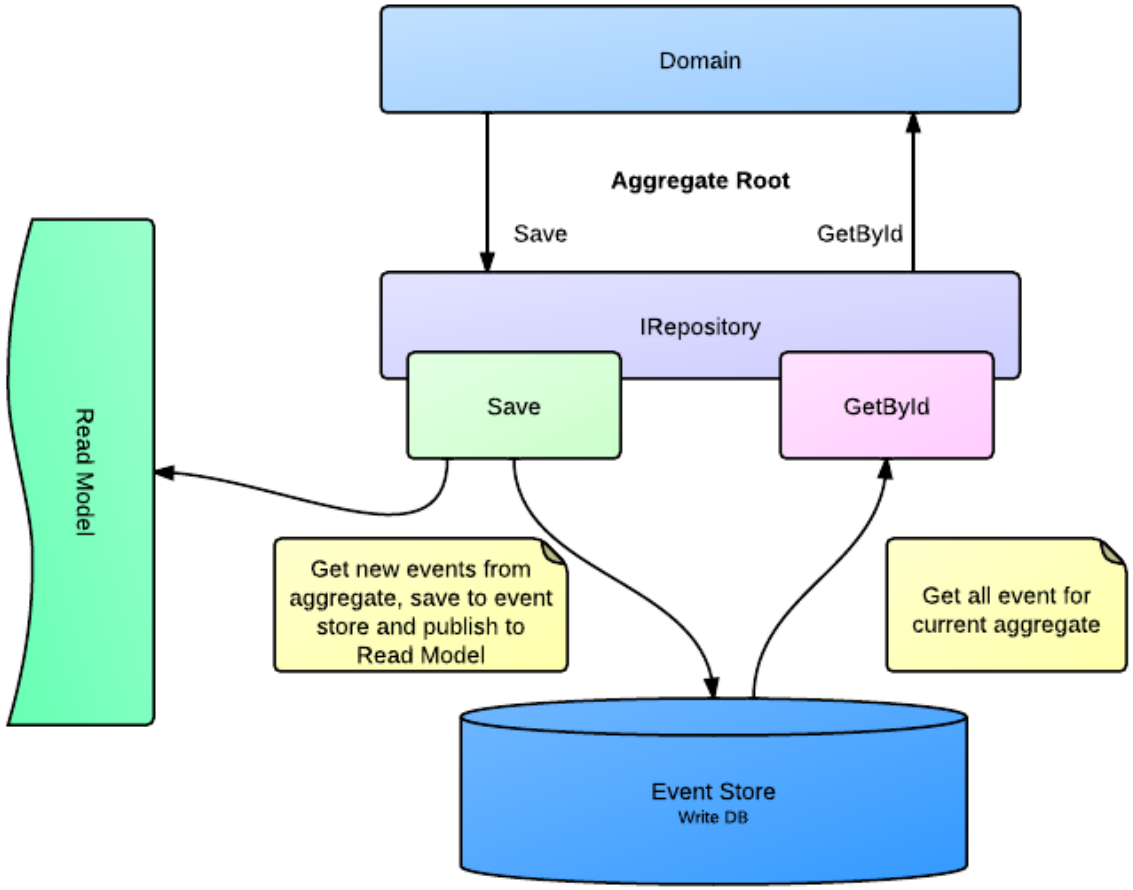
1. Add Spring Data Redis dependency to your project
2. Create Spring configuration for Redis and define connectionFactory and redisTemplate beans.
3. Try to persist key-value pairs of different types (string, list, set, hashmap) using RedisTemplate instance.

CQRS + Event sourcing



- Sergey Morenets, 2019

CQRS + Event sourcing



- Sergey Morenets, 2019

Query engines

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

Task 12. 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 4.0.10
- ✓ Supports transactions since 4.x

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, 2019

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** (version 4.0.x and later is recommended).
2. Switch to **sample** database: *use sample*.
3. Print all the existing databases: *db.adminCommand({ listDatabases: 1 });* What is the difference between this command and *show databases* command?

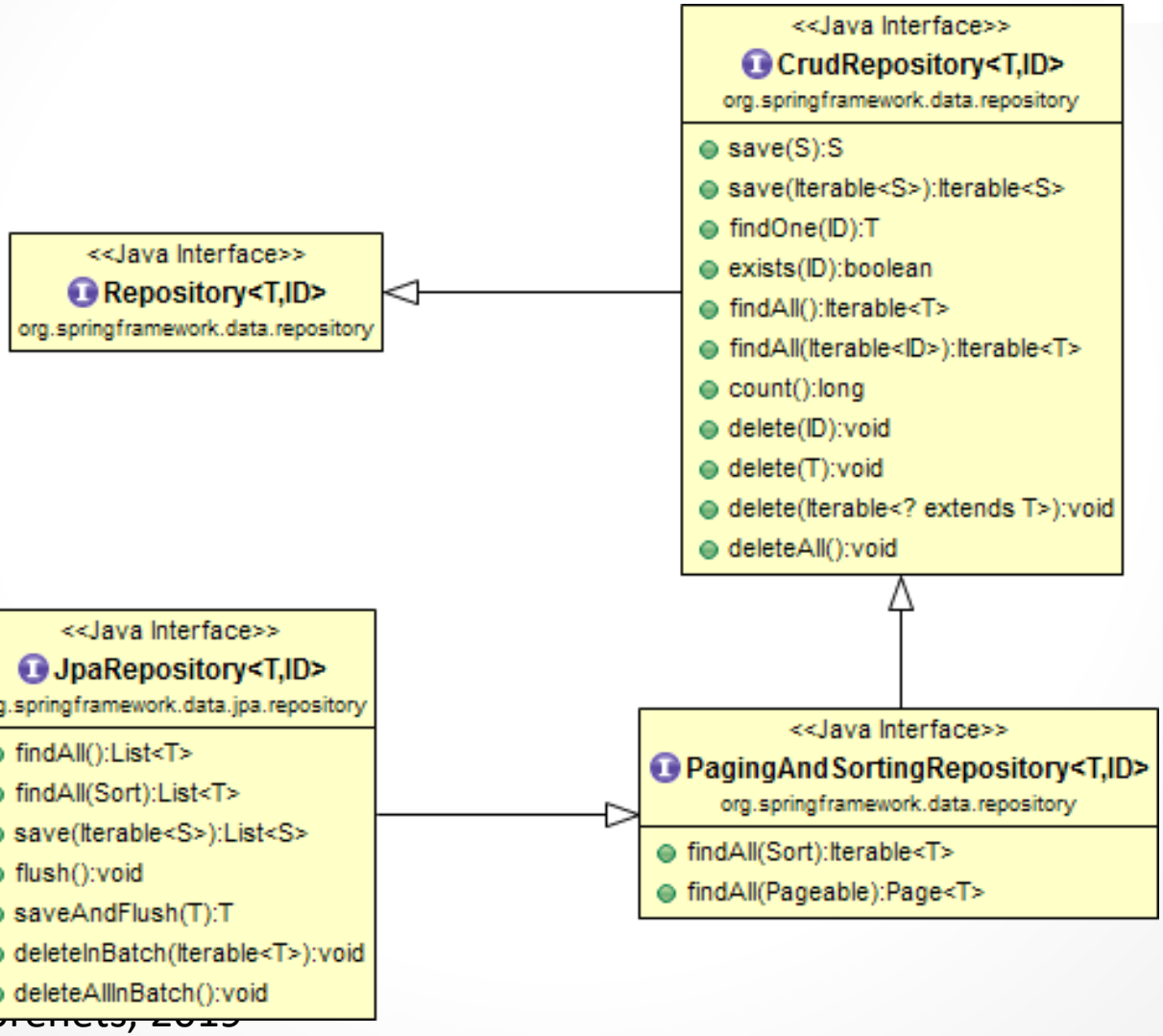
Spring Data MongoDB

- ✓ 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


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

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>{
}
```

```
@Autowired
private ProductRepository productRepository;
```

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
```

application.properties

Embedded MongoDB

- ✓ Open-source project
- ✓ In-memory Mongo database
- ✓ Easy to use in integration testing
- ✓ Easy to change version db version per test
- ✓ Supports Java 8.x and higher

```
<dependency>
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo</artifactId>
</dependency>
```

Spring Boot. Embedded MongoDB


```
@ConfigurationProperties(prefix = "spring.mongodb.embedded")
public class EmbeddedMongoProperties {

    /**
     * Version of Mongo to use.
     */
    private String version = "3.5.5";
```

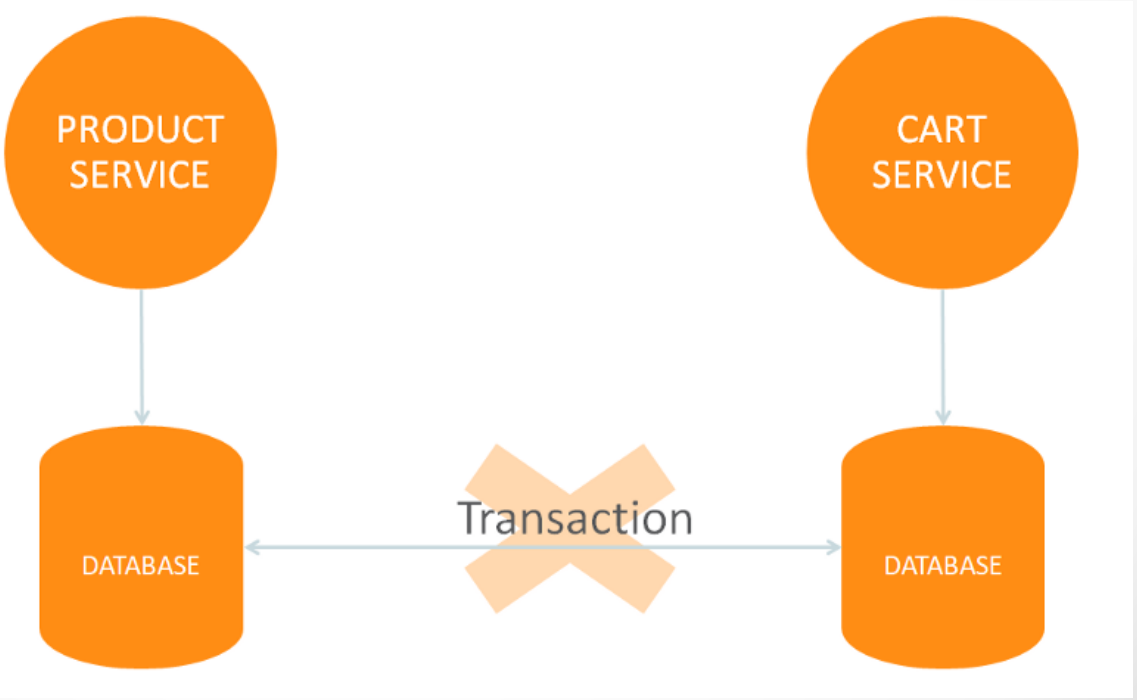
Default version to download/run

Task 14. Spring Data MongoDB

1. Add **Spring Data MongoDB** dependency to your project:
2. Implement event sourcing using Mongo database. Let **EventLogRepository** extends **MongoReposiory** interface.
3. Put **@EnableMongoRepositories** on the Spring Boot bootstrap class.
4. Run application and try to update or find **orders** objects. Open **Mongo client** and verify that documents were saved in the database.

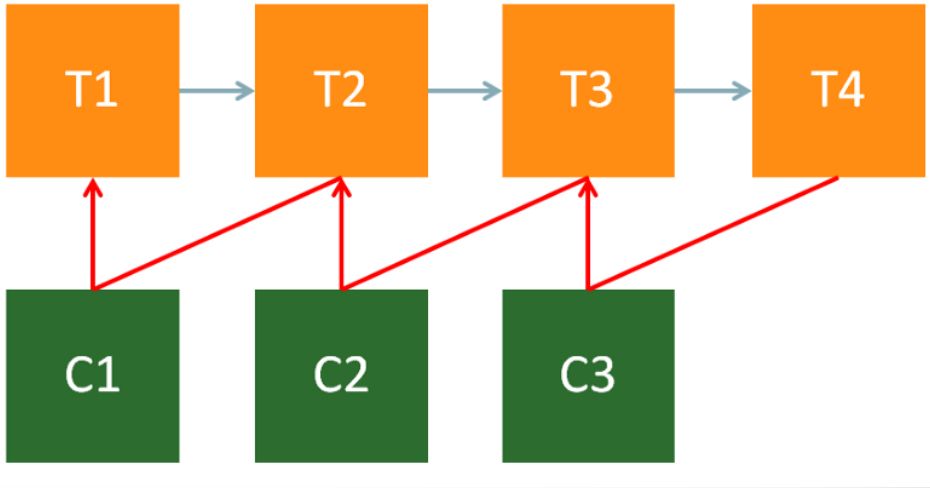
Distributed transactions

- It’s not possible to organize ACID transaction against multiple databases



Sagas

- Saga design pattern was invented by by Hector Garcia-Molina and Kenneth Salem in 1987
- If you have a transaction inside your microservice you should organize **compensation** transaction.
- For example, you put 50$ to your account. If you need to rollback, you create new transaction to deposit 50$



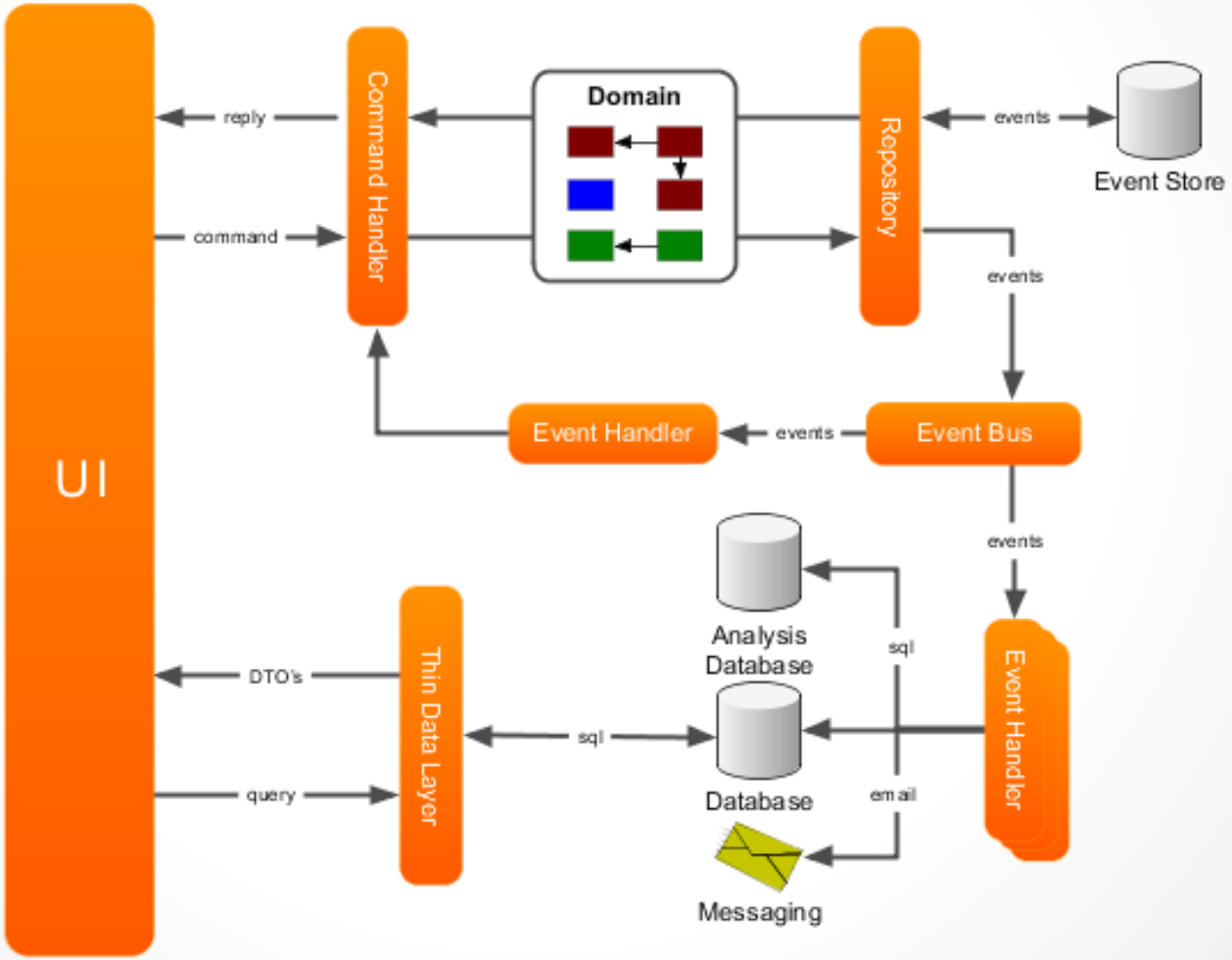
Sagas

- There are two types of compensation transactions
- Self-managed transactions are executed if service knows itself how to handle failures (involves coupling and increase complexity between services)
- Orchestrated transactions are executed by another service who organize service calls.

Axon

- Includes Axon Framework and Axon Server
- Axon Framework is Java technology to implement DDD, event sourcing and CQRS
- Supports command/event/query handlers, sagas, aggregates
- Axon Server supports built-in event storage, load balancing, messaging, basic security and monitoring

Axon application



- Sergey Morenets, 2019

Axon Framework

- Version 4.0 is going to release
- Integrates with Spring Boot
- Contains OSGI-compatible modules

```
<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-core</artifactId>
    <version>3.4.1</version>
</dependency>
```

Axon. Event handlers

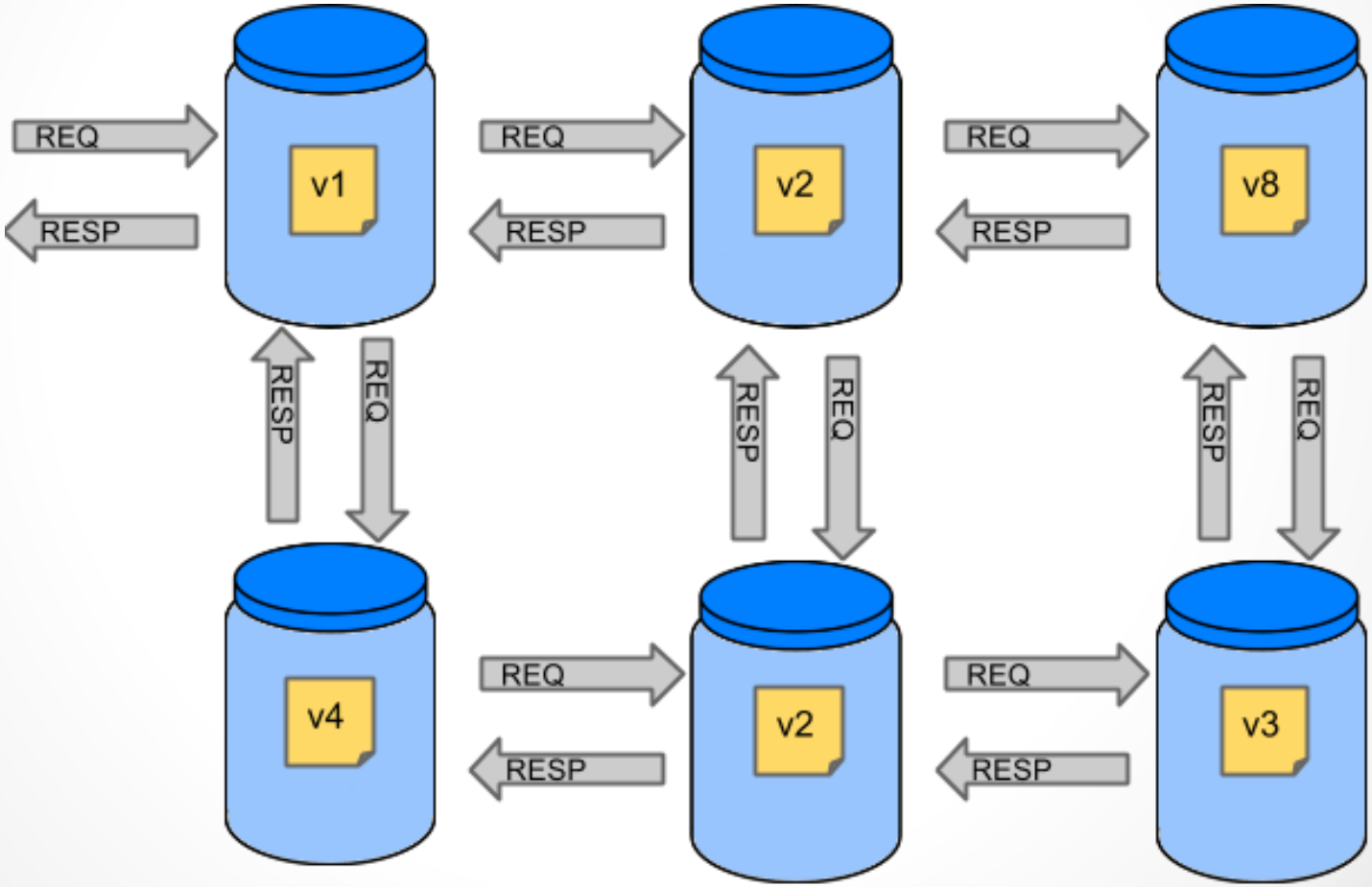

```
public class OrderListener {

    @EventHandler
    public void handle(OrderCreatedEvent event) {
    }

    @EventHandler
    public void handle(OrderCompletedEvent event) {
    }

}
```

Axon Server

- ✓ Version 4.0 released in October 2018
- ✓ Builds on top of Spring Boot
- ✓ Container event/query/command bus implementation
- ✓ Provides support for Docker/Kubernetes/Minikube
- ✓ Supports cluster node/load balancing
- ✓ Uses HTTP/gPRC for communication
- ✓ Use H2 database to store server configuration

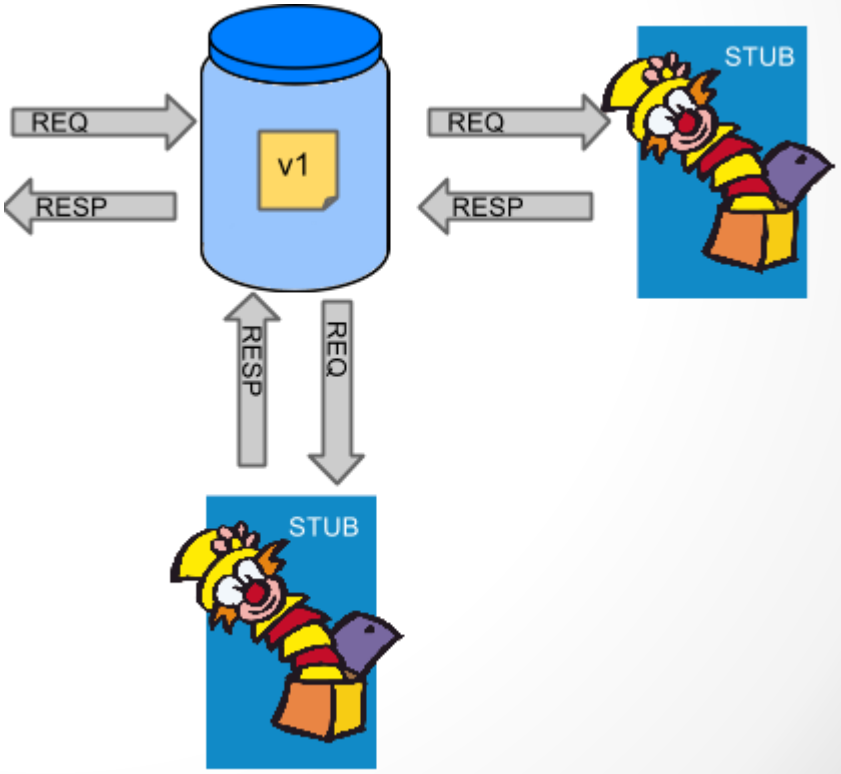
```
<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-spring-boot-starter</artifactId>
    <version>4.0</version>
</dependency>
```

Contract verifier



- Sergey Morenets, 2019

Contract verifier. Stubs



- Sergey Morenets, 2019

WireMock

- Technology for mocking your API
- Simulator(mock-server) for HTTP-based API
- Request matching
- Record and playback
- Allows to simulate faults
- Supports Android
- Configurable by Java API/JSON files

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>
```

- Sergey Morenets, 2019

WireMock. Class rules(JUnit 4)


```
@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");
    }
```

- Sergey Morenets, 2019

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

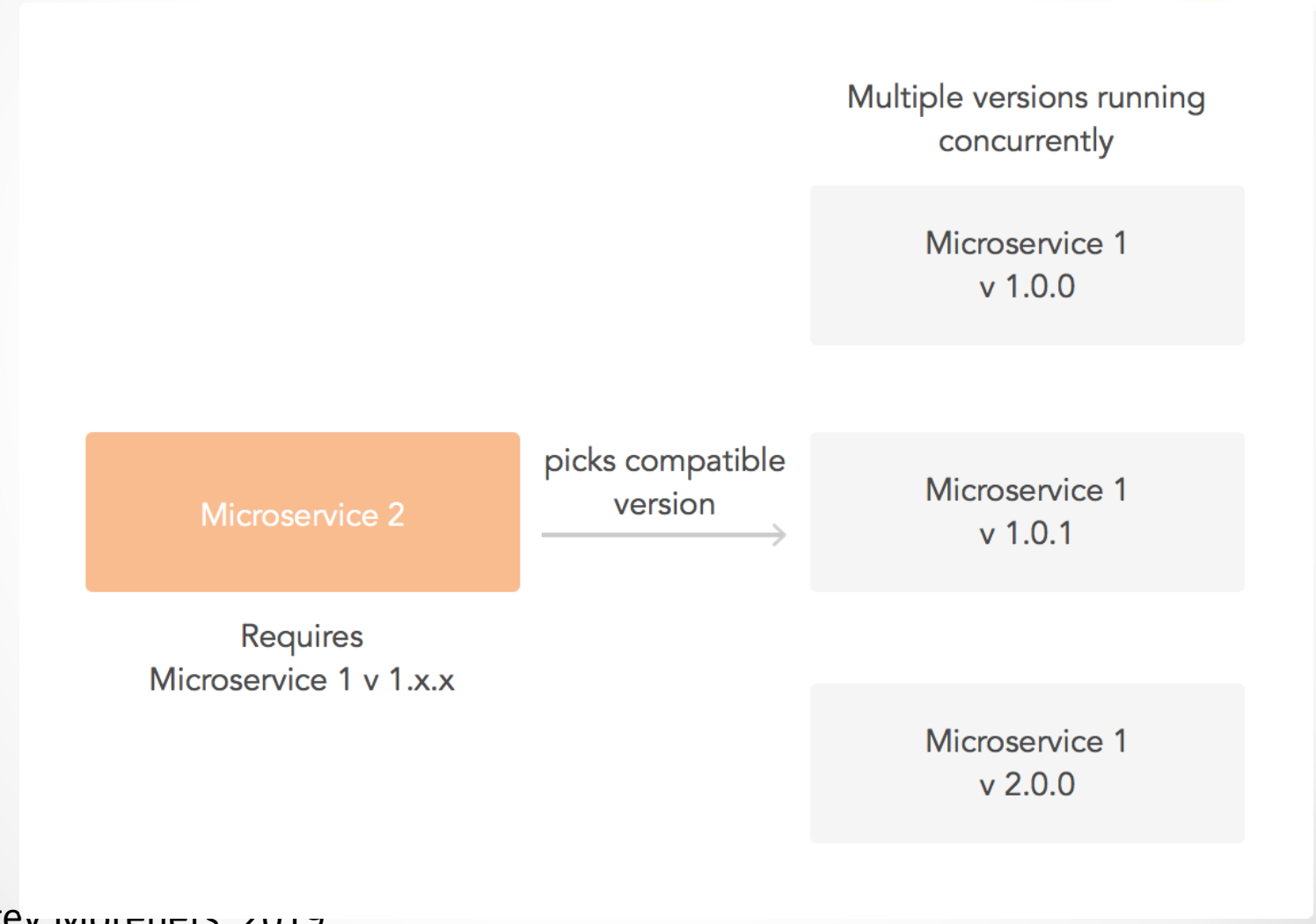

```
{
  "request" : {
    "urlPath" : "/order",
    "method" : "GET"
  },
  "response" : {
    "status" : 200,
    "body" : "hi"
  }
}
```

src/test/resources/responses/order.json

Task 15. 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.

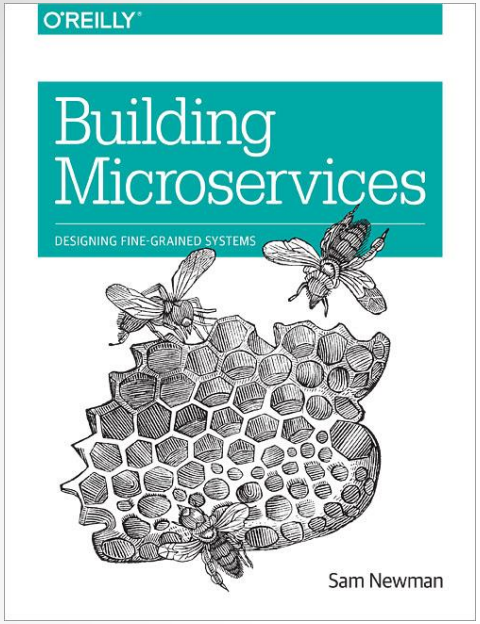
Versioning

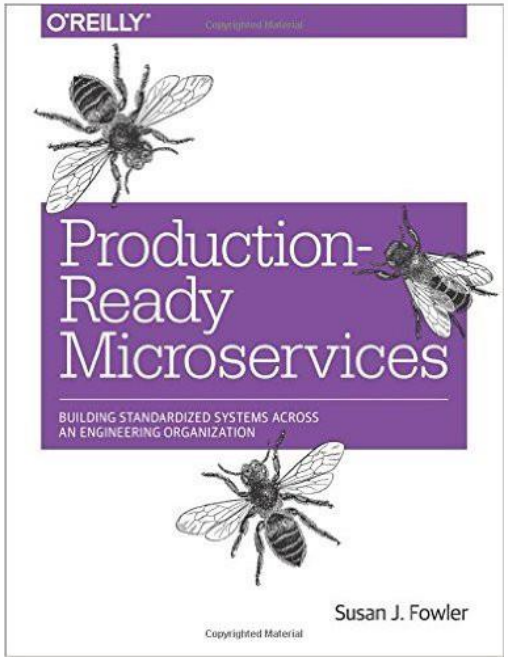


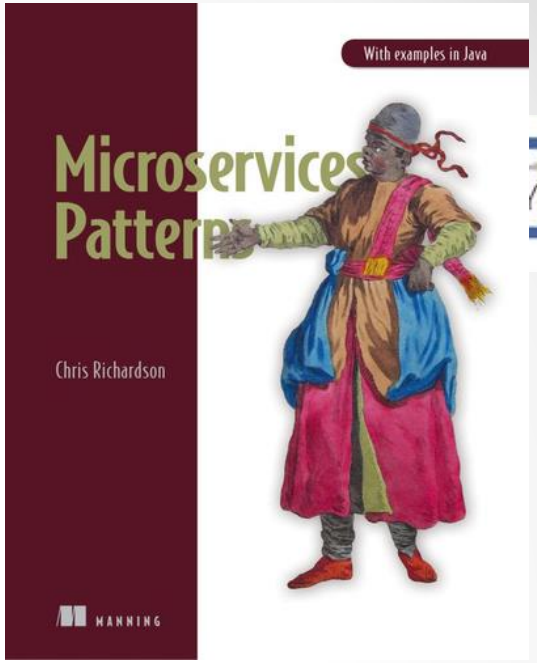
- Sergey Morenets, 2019

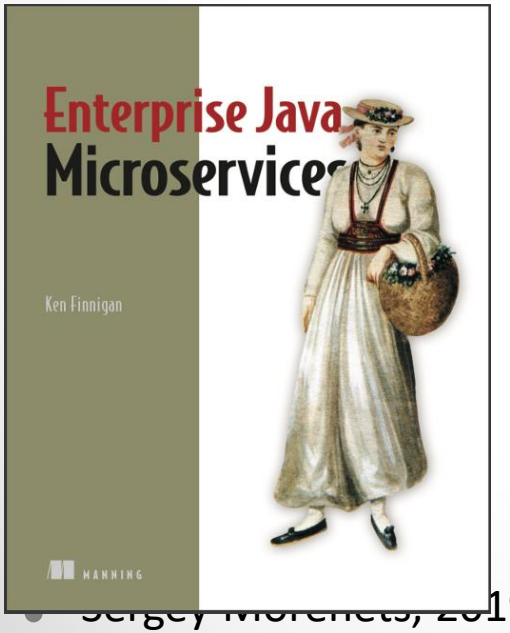
Drawbacks

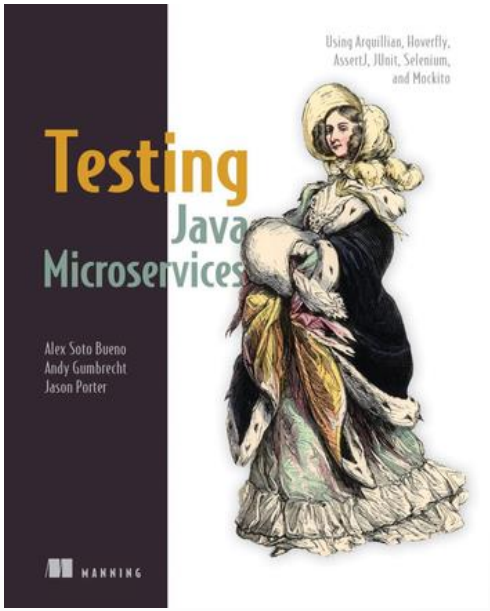
- ✓ Bad design solutions stored forever
- ✓ Complex queries & lookup

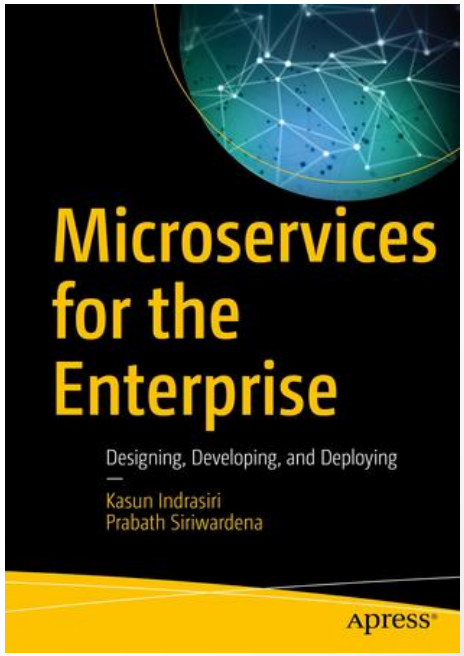

O'REILLY
Building Microservices
DESIGNING FINE-GRAINED SYSTEMS
Sam Newman


O'REILLY
Production-Ready Microservices
BUILDING STANDARDIZED SYSTEMS ACROSS AN ENGINEERING ORGANIZATION
Susan J. Fowler
Copyrighted Material

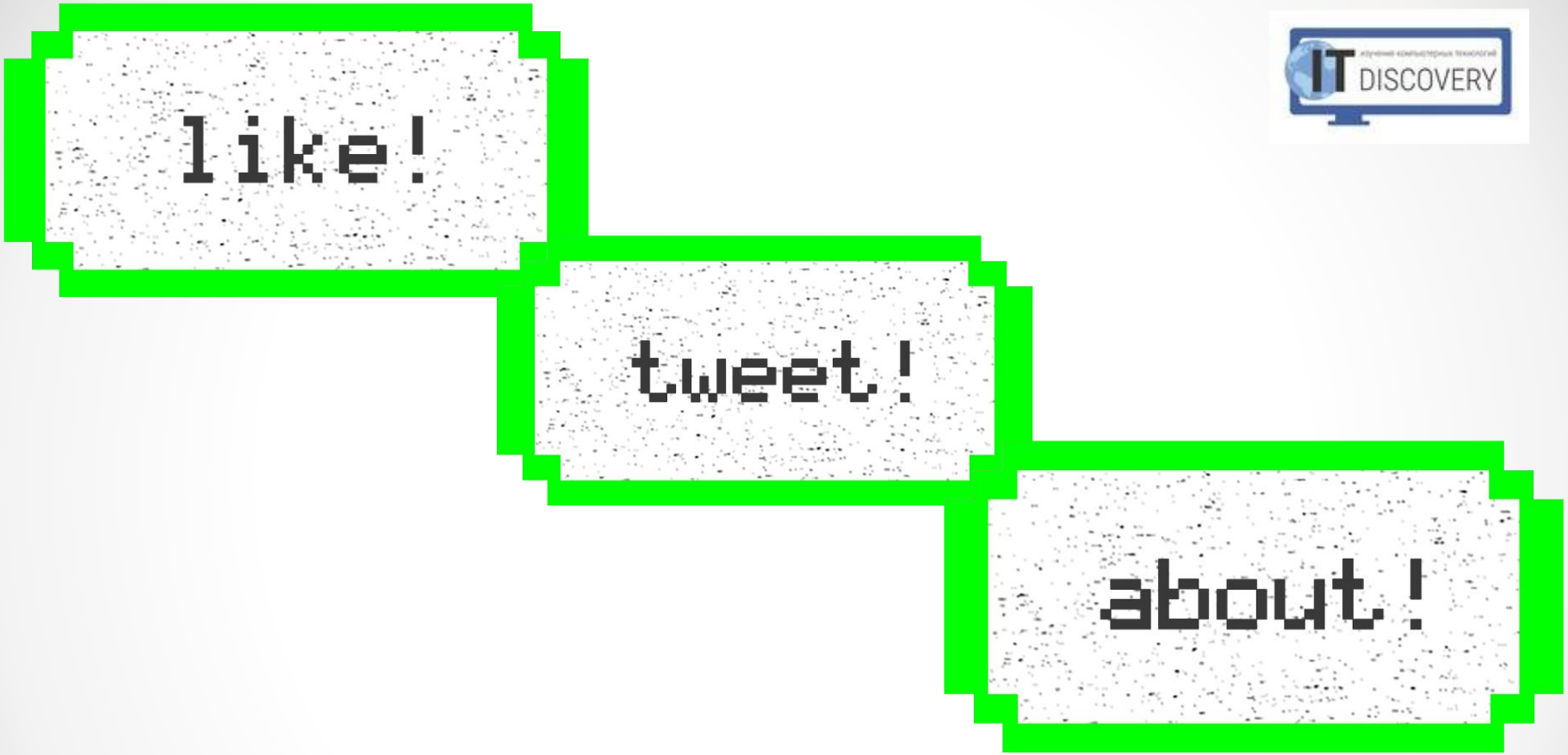

With examples in Java
Microservices Patterns
Chris Richardson
MANNING


Enterprise Java Microservices
Ken Finnigan
MANNING


Using Arquillian, Hoverfly, AssertJ, JUnit, Selenium, and Mockito
Testing Java Microservices
Alex Soto Bueno
Andy Gumbrecht
Jason Porter
MANNING


Microservices for the Enterprise
Designing, Developing, and Deploying
Kasun Indrasiri
Prabath Siriwardena
Apress

✓ Sergey Morenets, sergey.morenets@gmail.com



✓ Sergey Morenets, sergey.morenets@gmail.com