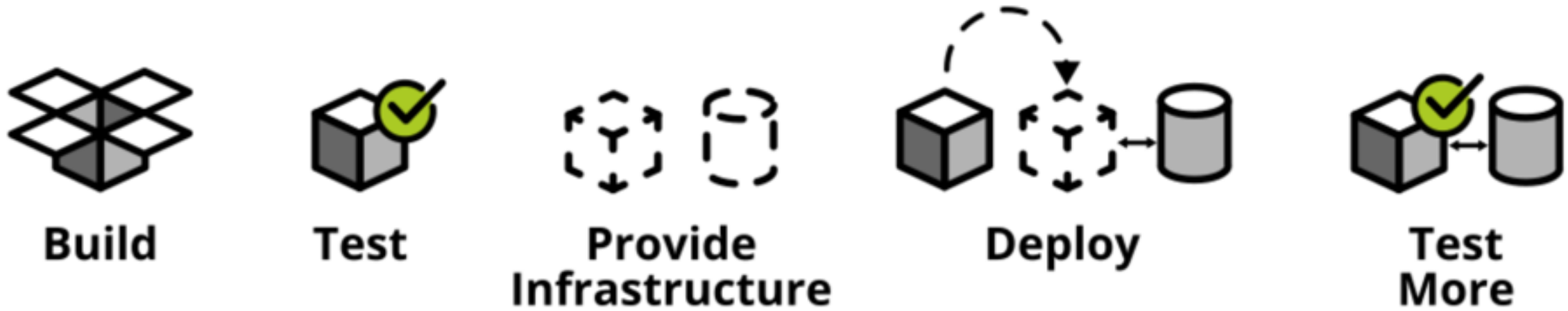
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Microservices and DevOps

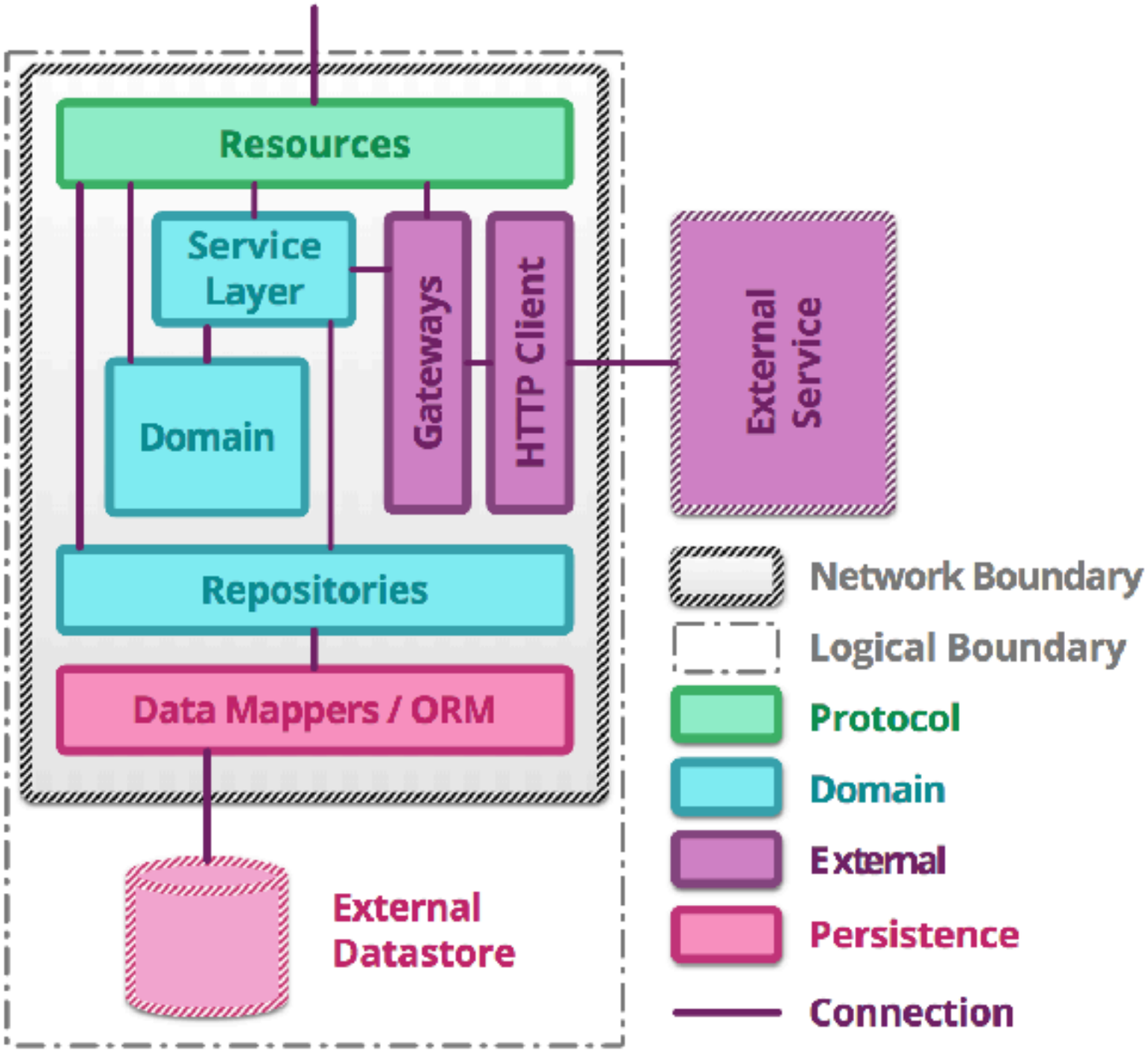
In Practices with Java Technology

Build pipeline



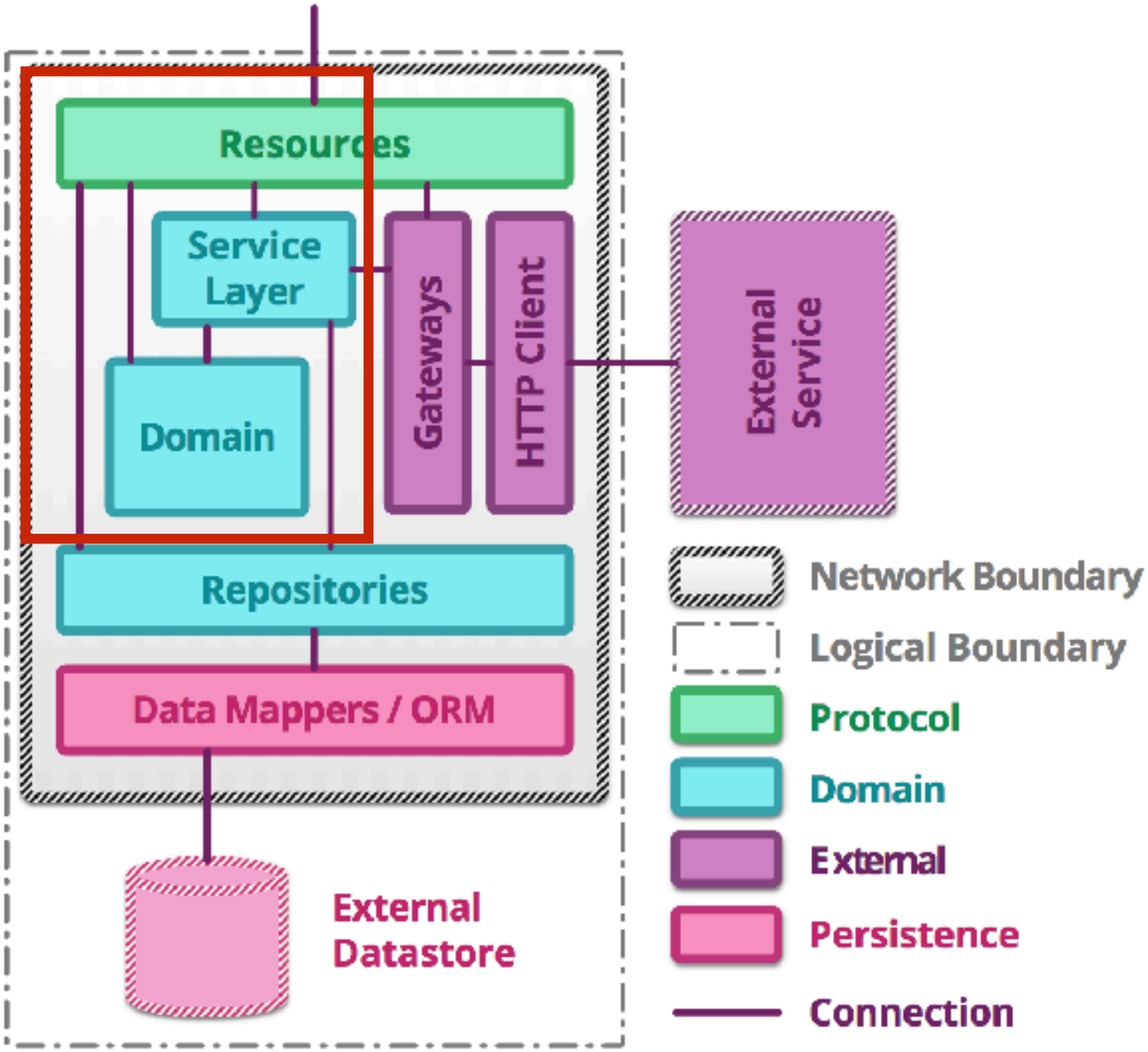
Develop Microservices with Spring Boot

Service Structure

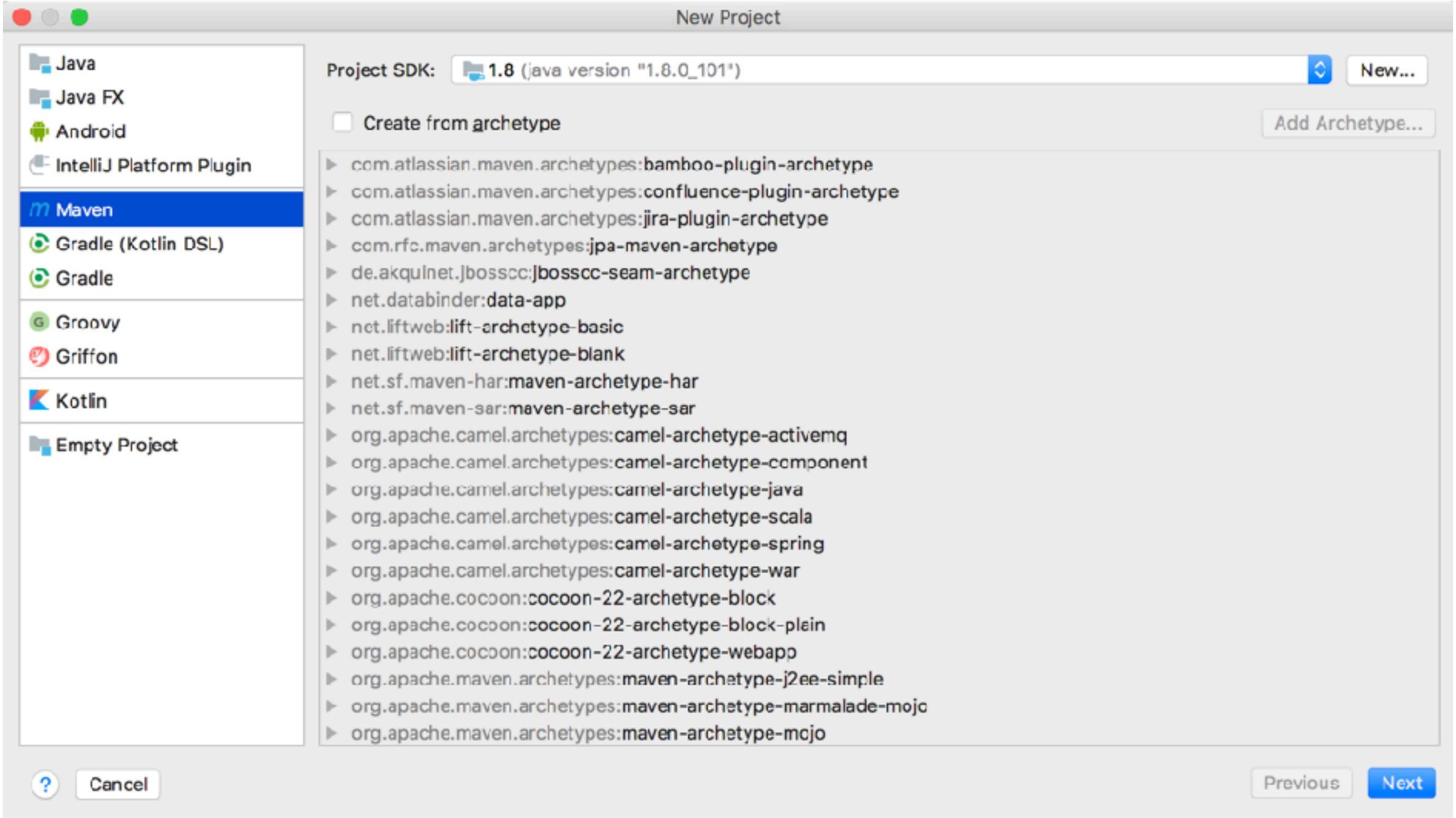


Hello Spring Boot 2.0

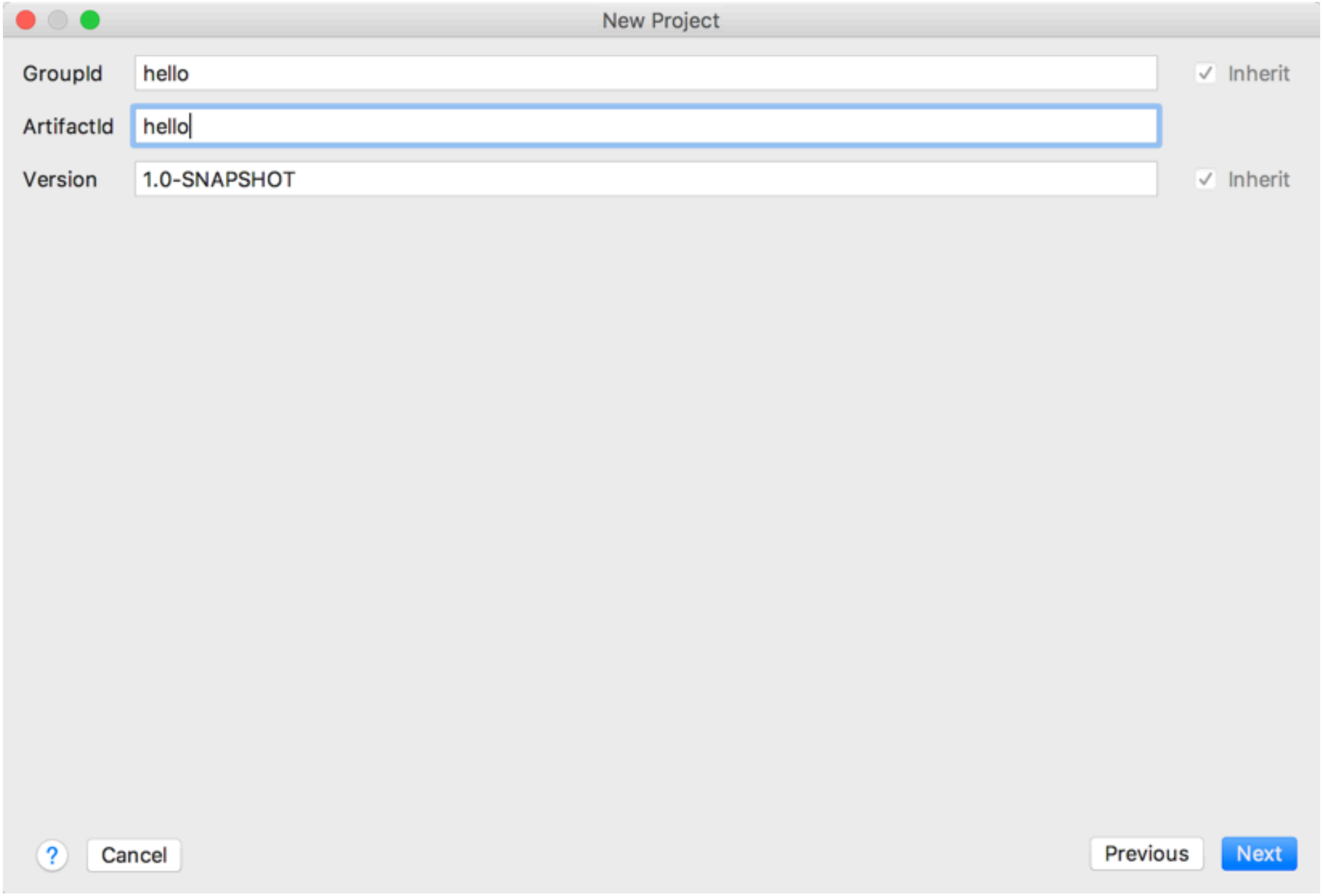
Service Structure



1. Create Maven Project



2. Project Name



3. Modify pom.xml (1)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.o
    <modelVersion>4.0.0</modelVersion>

    <groupId>hello</groupId>
    <artifactId>hello</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.0.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>
```

3. Modify pom.xml (2)

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <finalName>hello</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

4. Create String boot application

hello.HelloApplication.java

```
package hello;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloApplication.class, args);
    }

}
```

5. Create model class

hello.model.Hello.java

```java
package hello.domain;

public class Hello {

    private String message;

    public Hello(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

6. Create REST Controller

hello.controller.HelloController.java

```java
package hello.controller;

import hello.domain.Hello;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public Hello sayHi(@PathVariable String name) {
        return new Hello("Hello " + name);
    }

}
```

7. Compile and Packaging

$mvn clean package

8. Run

$java -jar target/hello.jar

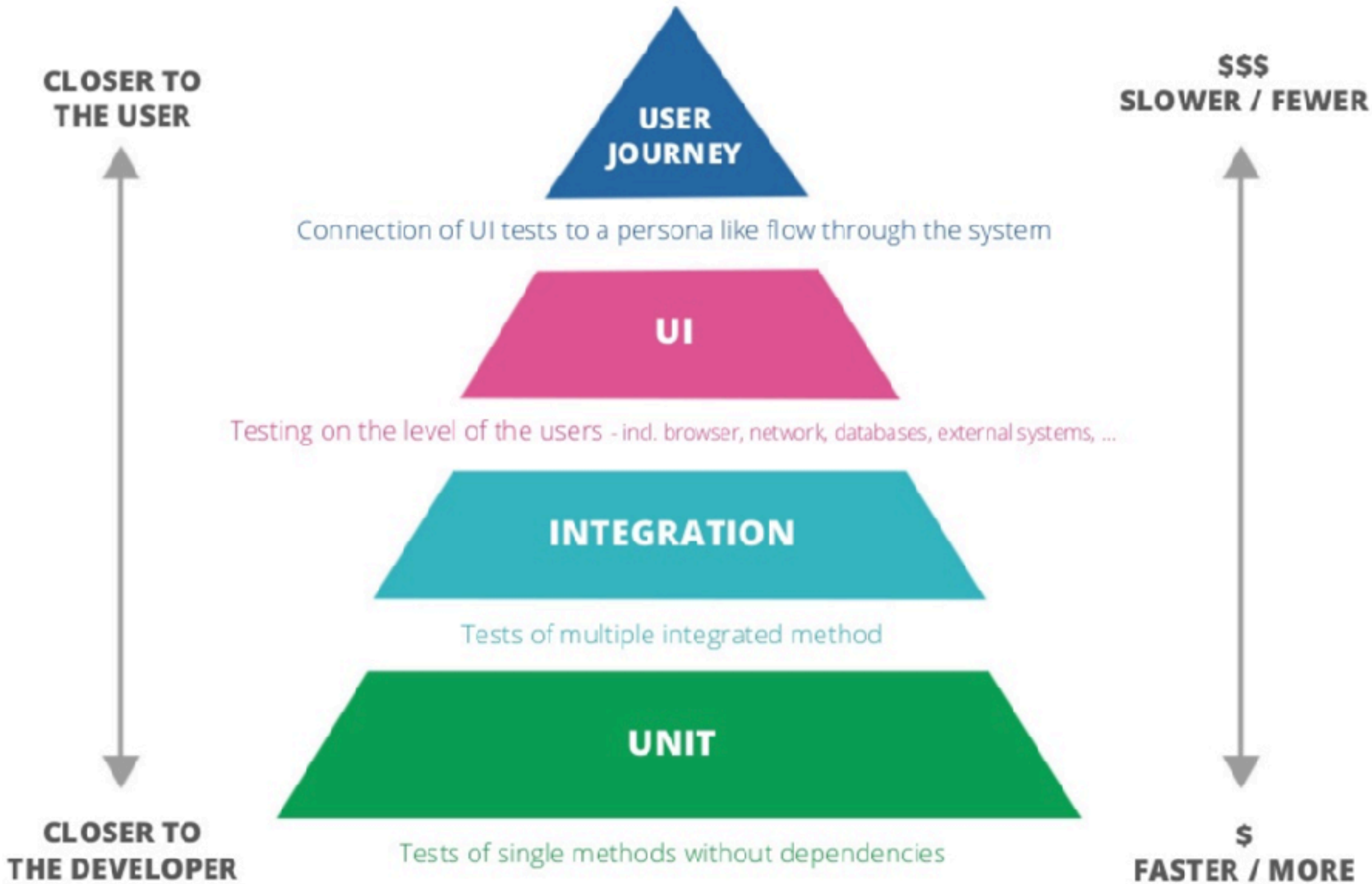
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

2018-03-05 23:37:18.018  INFO 30560 --- [                mai
                : Starting HelloApplication v1.0-SNAPSH
th PID 30560 (/Users/somkiat/data/slide/microservice/sl
op/course-microservice/slide/4days-workshop/workshop/he
 by somkiat in /Users/somkiat/data/slide/microservice/s
hop/course-microservice/slide/4days-workshop/workshop/h
2018-03-05 23:37:18.023  INFO 30560 --- [                mai
                : No active profile set, falling back t
2018-03-05 23:37:18.138  INFO 30560 --- [                mai
```

9. Open in browser

http://localhost:8080/hello/somkiat

How to test the Hello service ?



Unit tests

How to use model ?

```java
package hello.controller;

import hello.domain.Hello;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public Hello sayHi(@PathVariable String name) {
        return new Hello("Hello " + name);
    }

}
```

API/Controller tests

How to use controller ?

Spring boot provides MockMvc to test Controller

API/Controller tests

```
@RunWith(SpringRunner.class)
@WebMvcTest(controllers = HelloController.class)
public class HelloControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void shouldReturnHelloSomkiat() throws Exception {
        mockMvc.perform(get( urlTemplate: "/hello/somkiat"))
                .andExpect(
                        jsonPath( expression: "$.message")
                                .value( expectedValue: "Hello somkiat"))
                .andExpect(status().is2xxSuccessful());
    }

}
```

Compile with testing

$mvn clean package

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
```

% of Code/Test coverage

Add coverage to pom.xml (1)

```
<build>
    <finalName>hello</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>

        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.5.1</version>
            <configuration>
                <source>${java.version}</source>
                <target>${java.version}</target>
                <encoding>${project.build.sourceEncoding}</encoding>
            </configuration>
        </plugin>
```

Add coverage to pom.xml (2)

```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>cobertura-maven-plugin</artifactId>
    <version>2.7</version>
    <configuration>
        <formats>
            <format>html</format>
            <format>xml</format>
        </formats>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>cobertura</goal>
            </goals>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>org.ow2.asm</groupId>
            <artifactId>asm</artifactId>
            <version>5.0.3</version>
        </dependency>
    </dependencies>
</plugin>
```

Run test again

$mvn clean package

```
Cobertura Report generation was successful.
Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file
Cobertura: Loaded information on 3 classes.
time: 125ms

Cobertura Report generation was successful.
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Coverage report

open target/site/cobertura/index.html

Packages

All
hello
hello.controller
hello.domain

All Packages

Classes

Hello *(66%)*
HelloApplication *(33%)*
HelloController *(100%)*

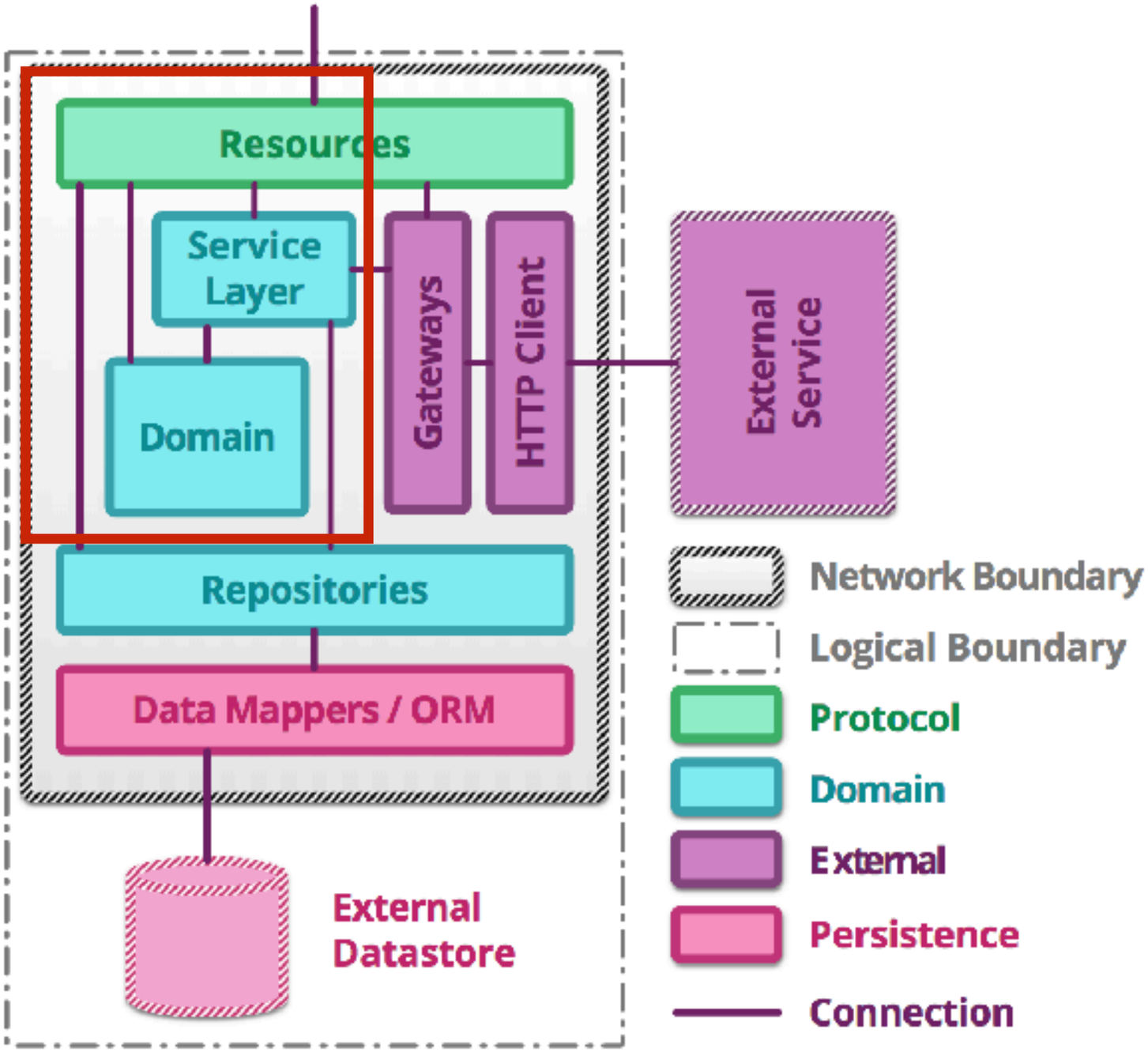
Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	3	63%	7/11	N/A	N/A	1
hello	1	33%	1/3	N/A	N/A	1
hello.controller	1	100%	2/2	N/A	N/A	1
hello.domain	1	66%	4/6	N/A	N/A	1

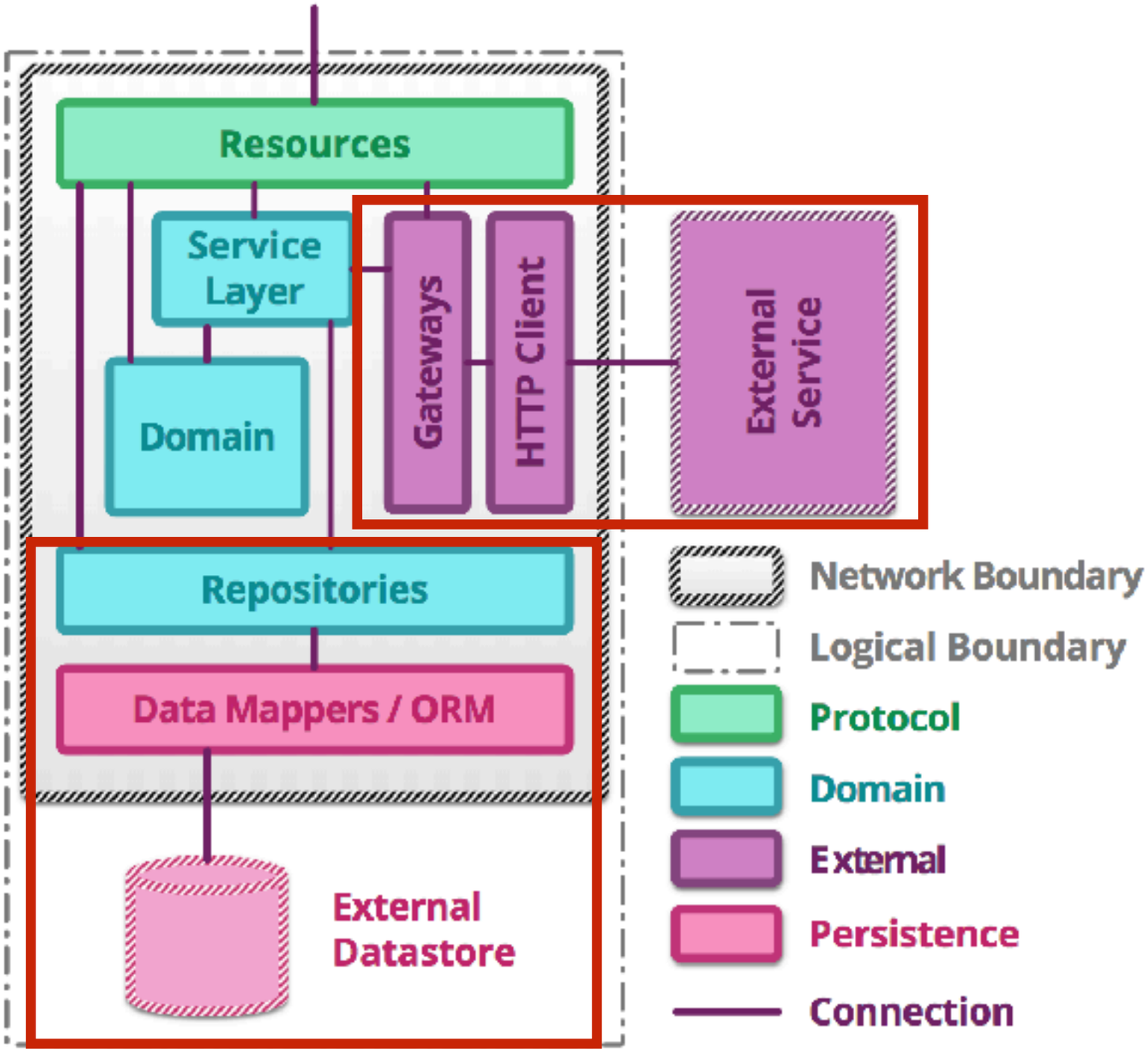
Report generated by Cobertura 2.1.1 on 3/6/18 12:40 AM.

How to improve % of coverage ?

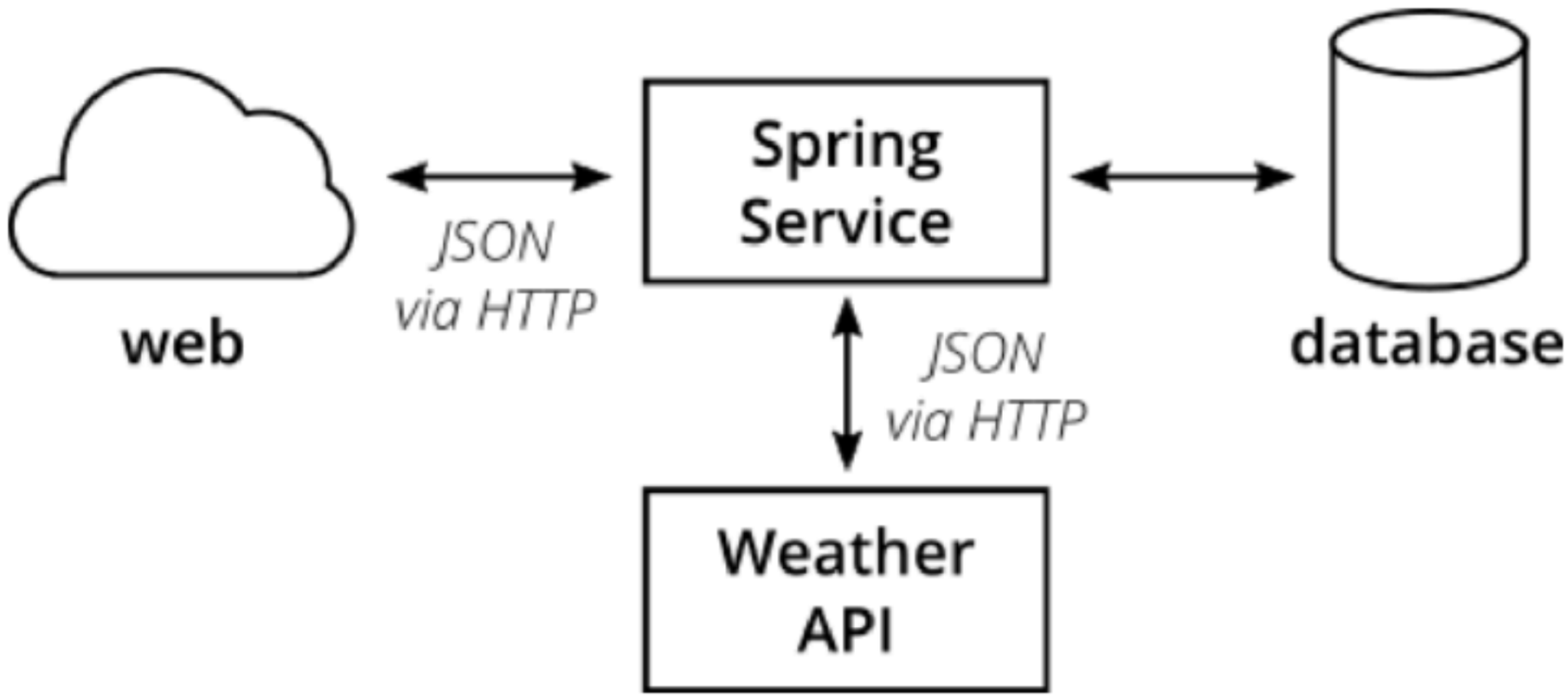
Service Structure



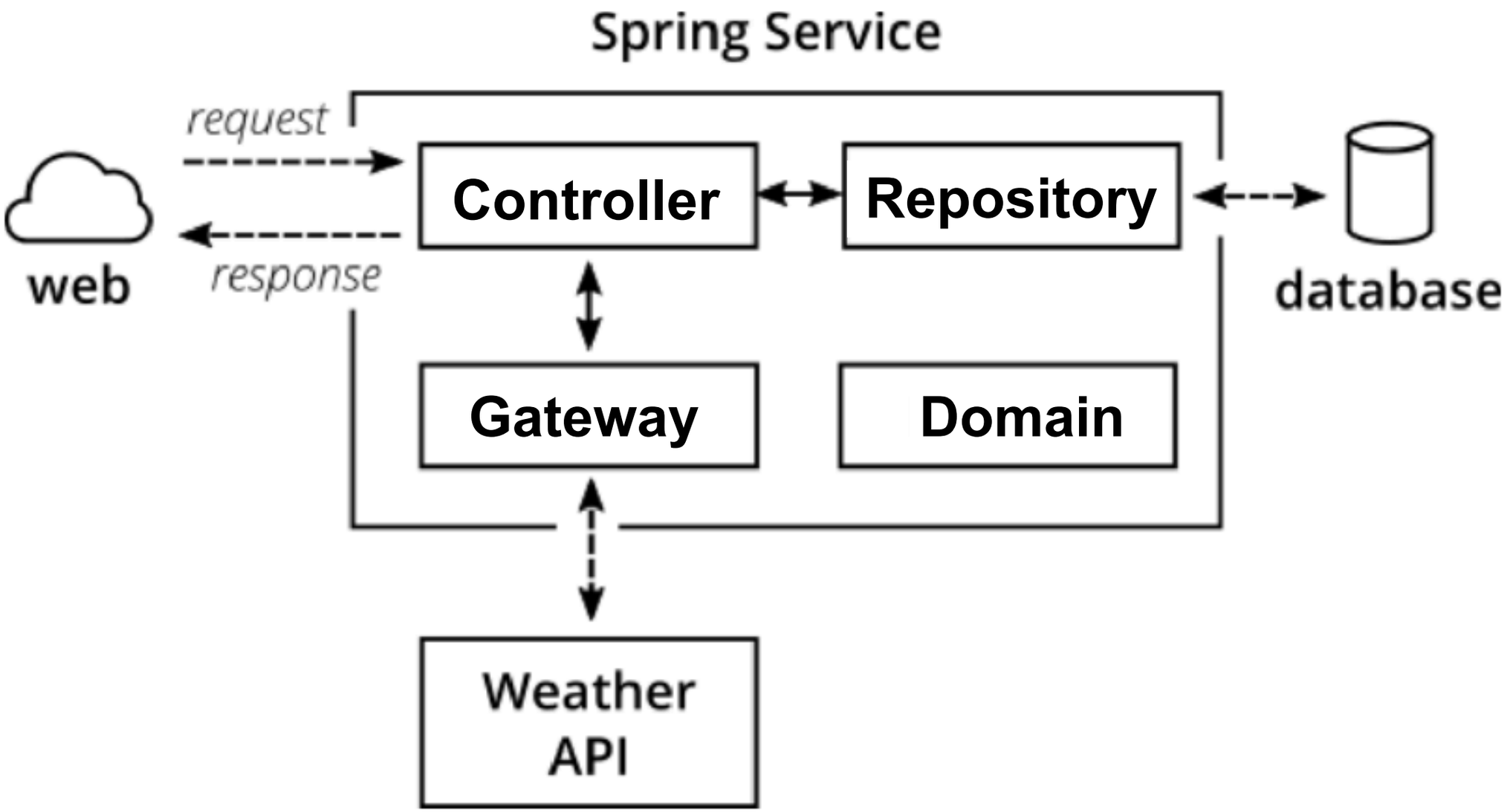
Service Structure



Sample application

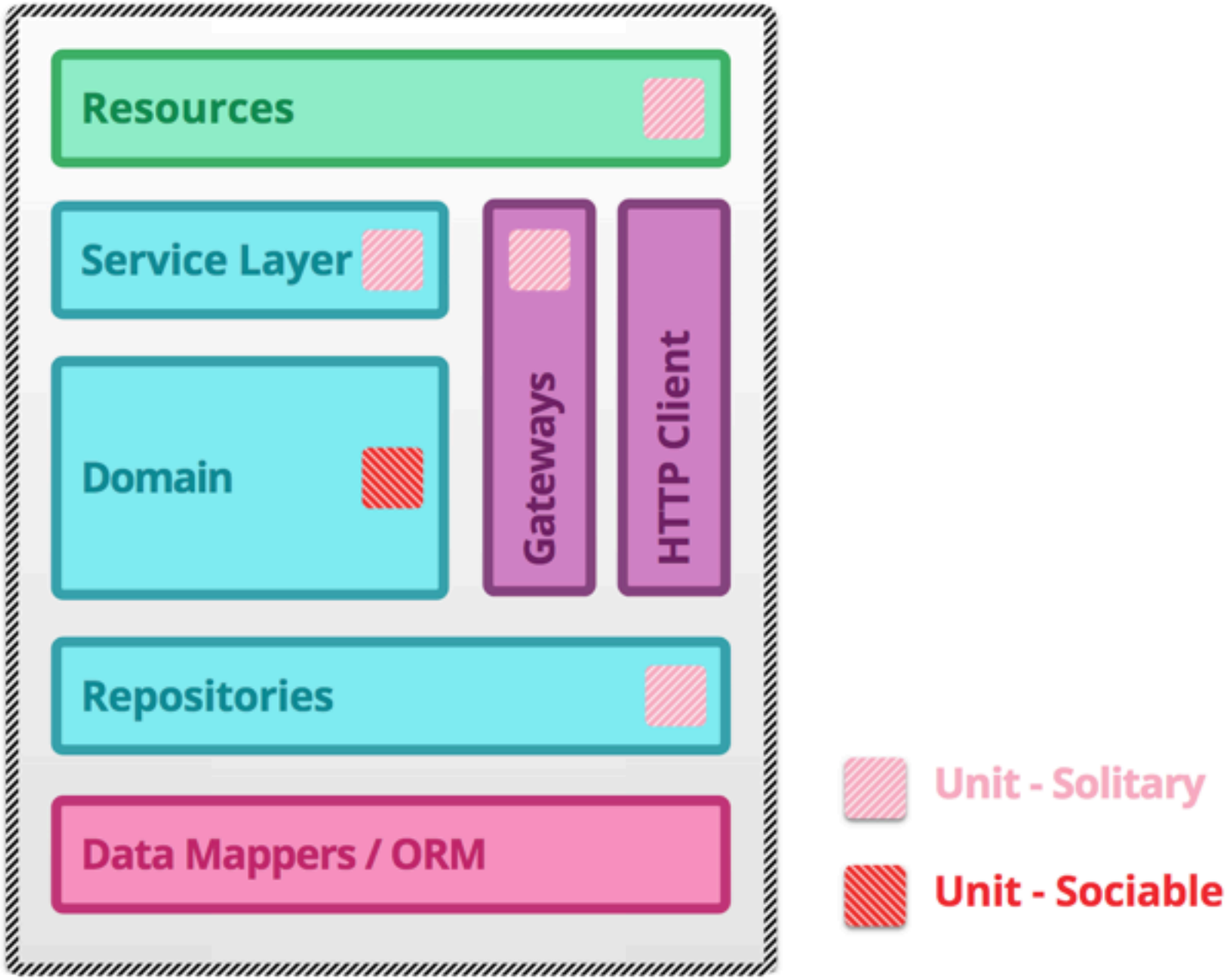


Project structure

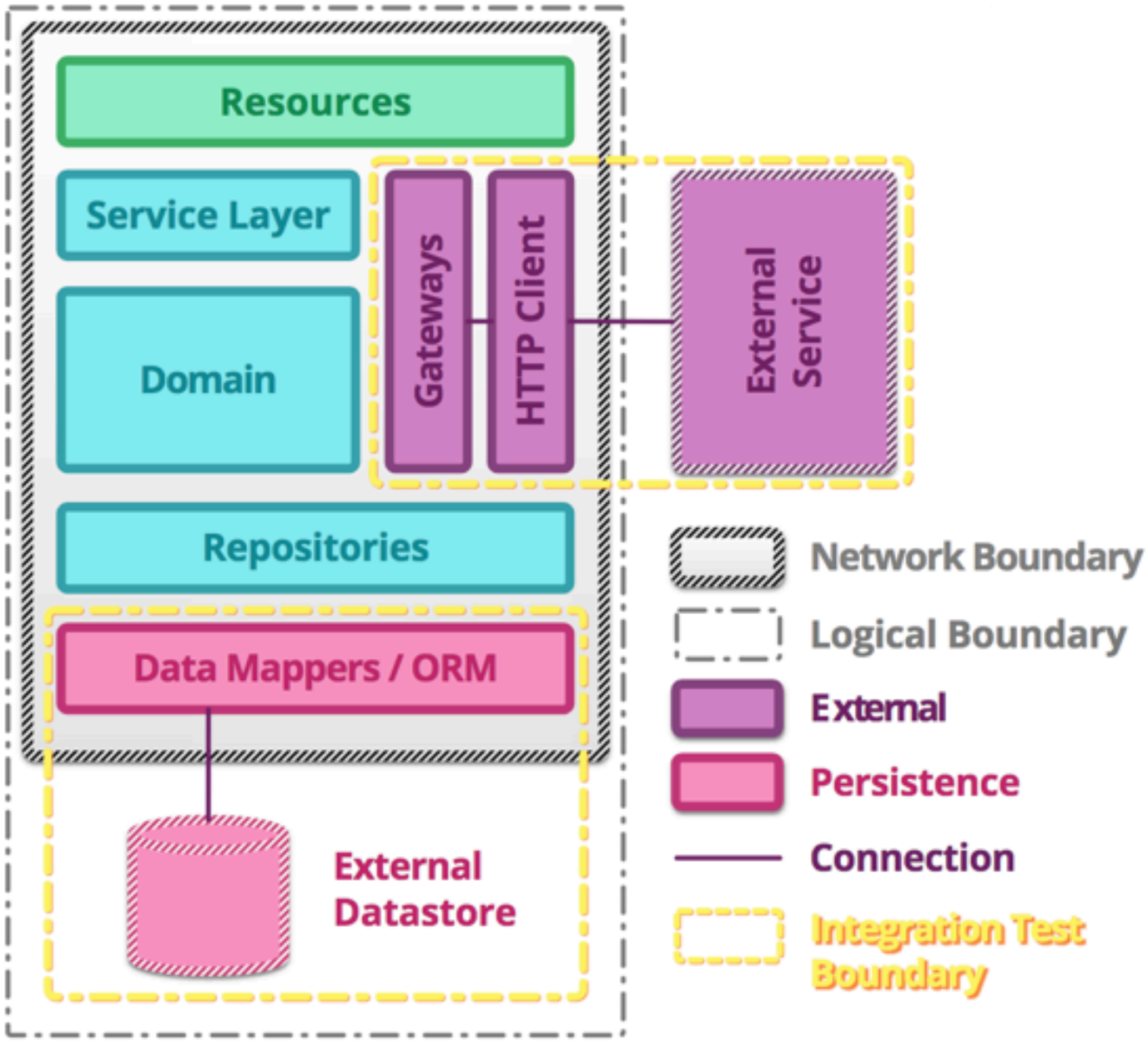


How to test ?

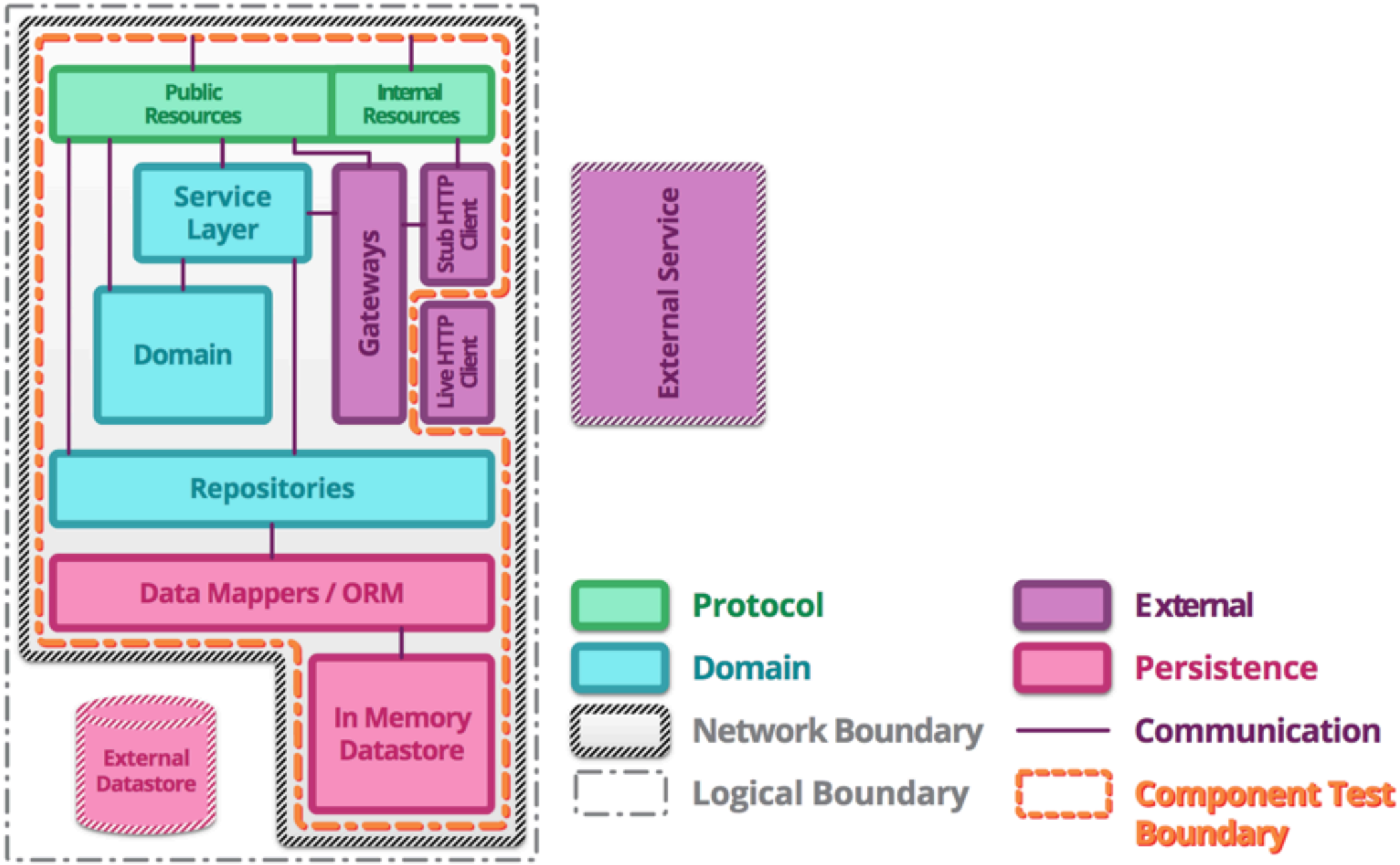
Unit testing



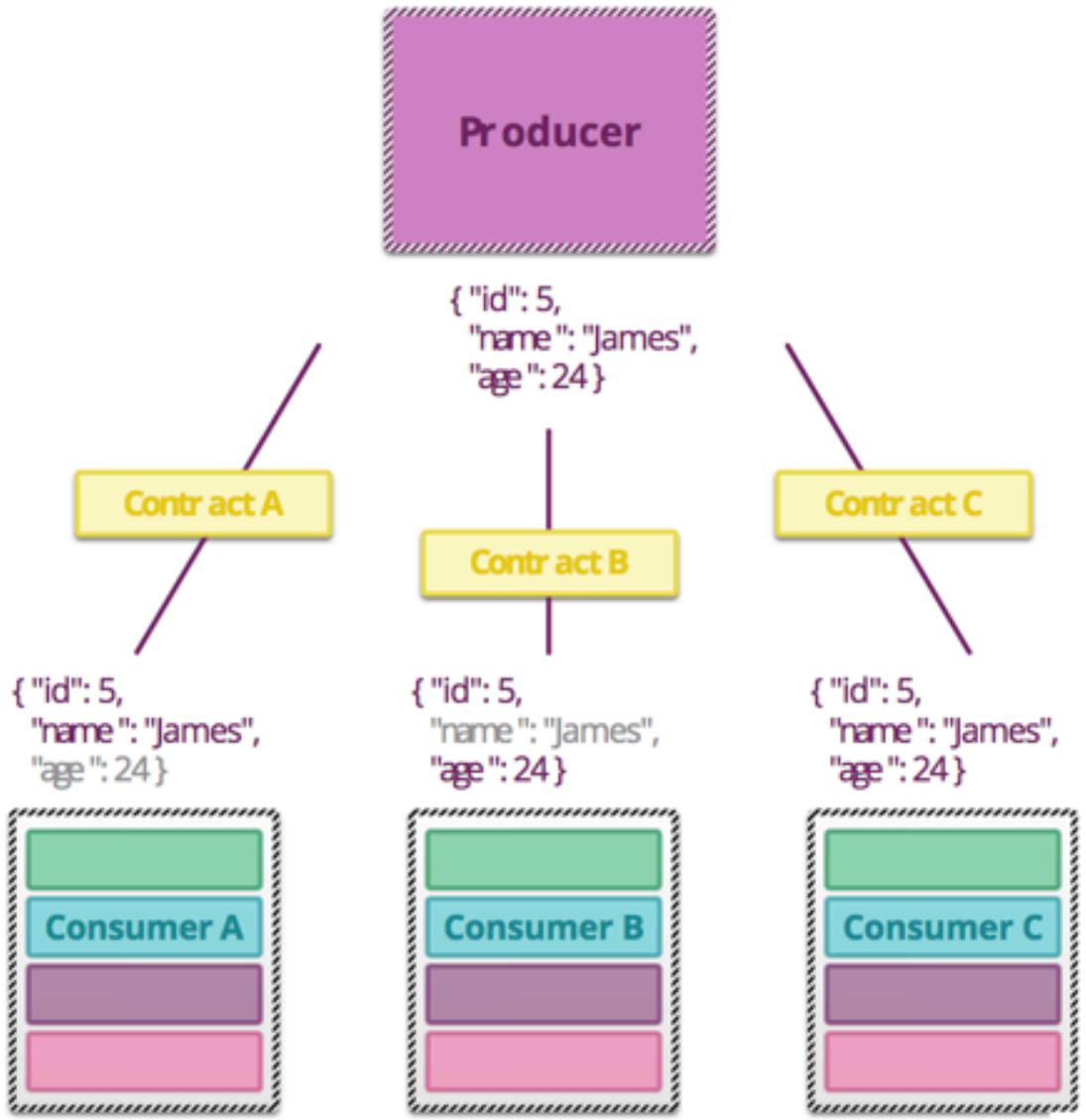
Integration testing



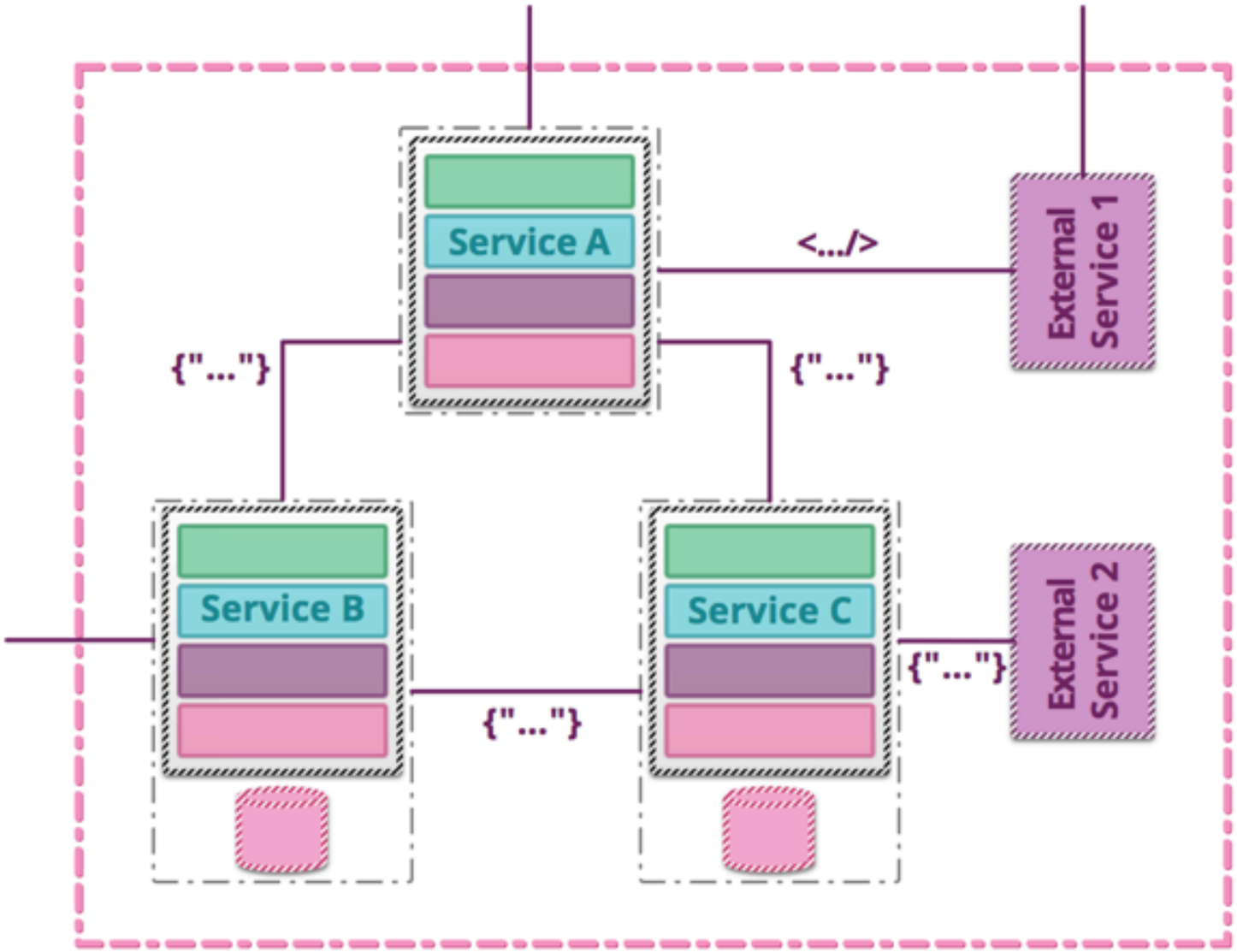
Component testing



Contract testing



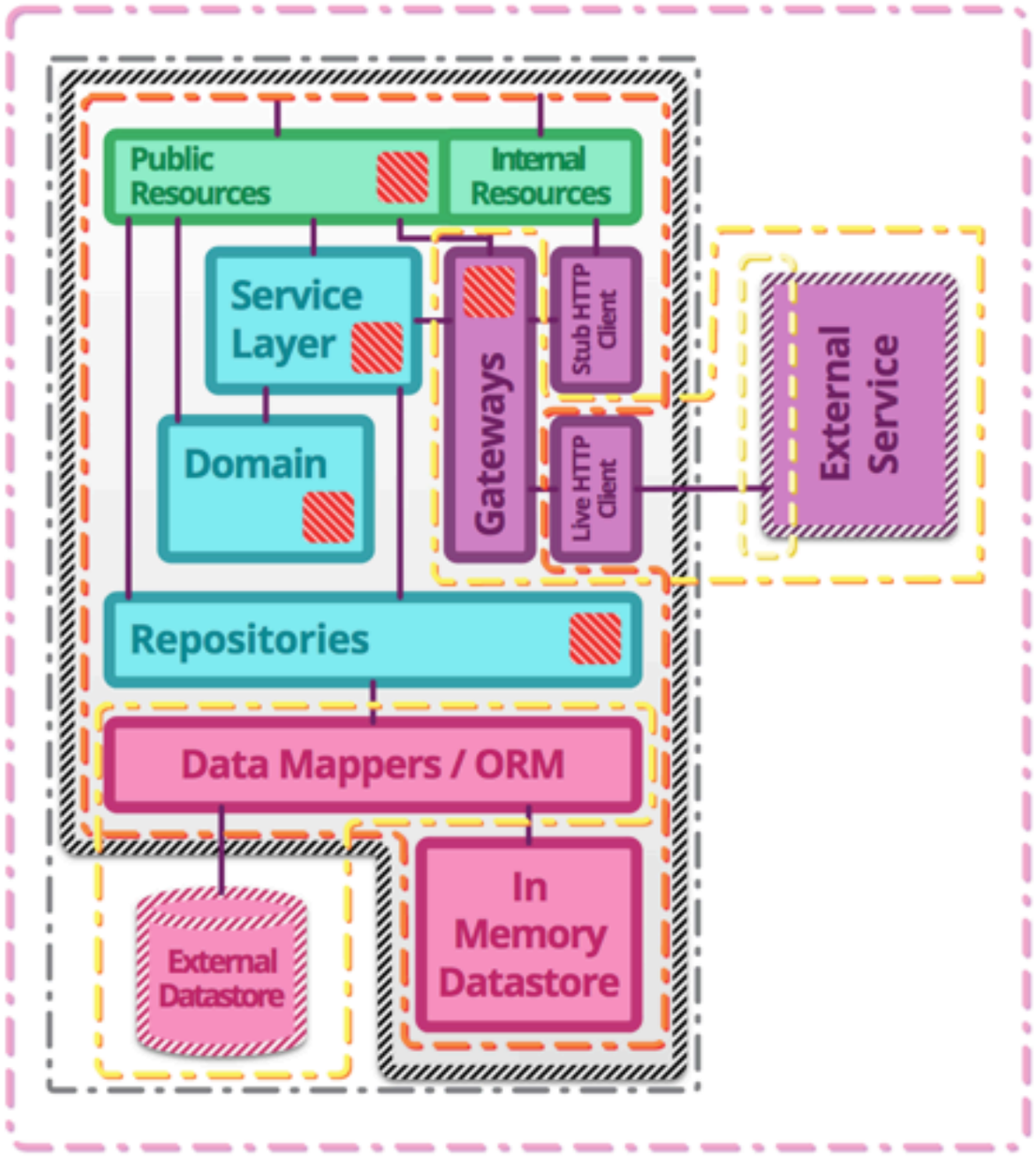
End-to-End testing



Summary

Unit tests : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.



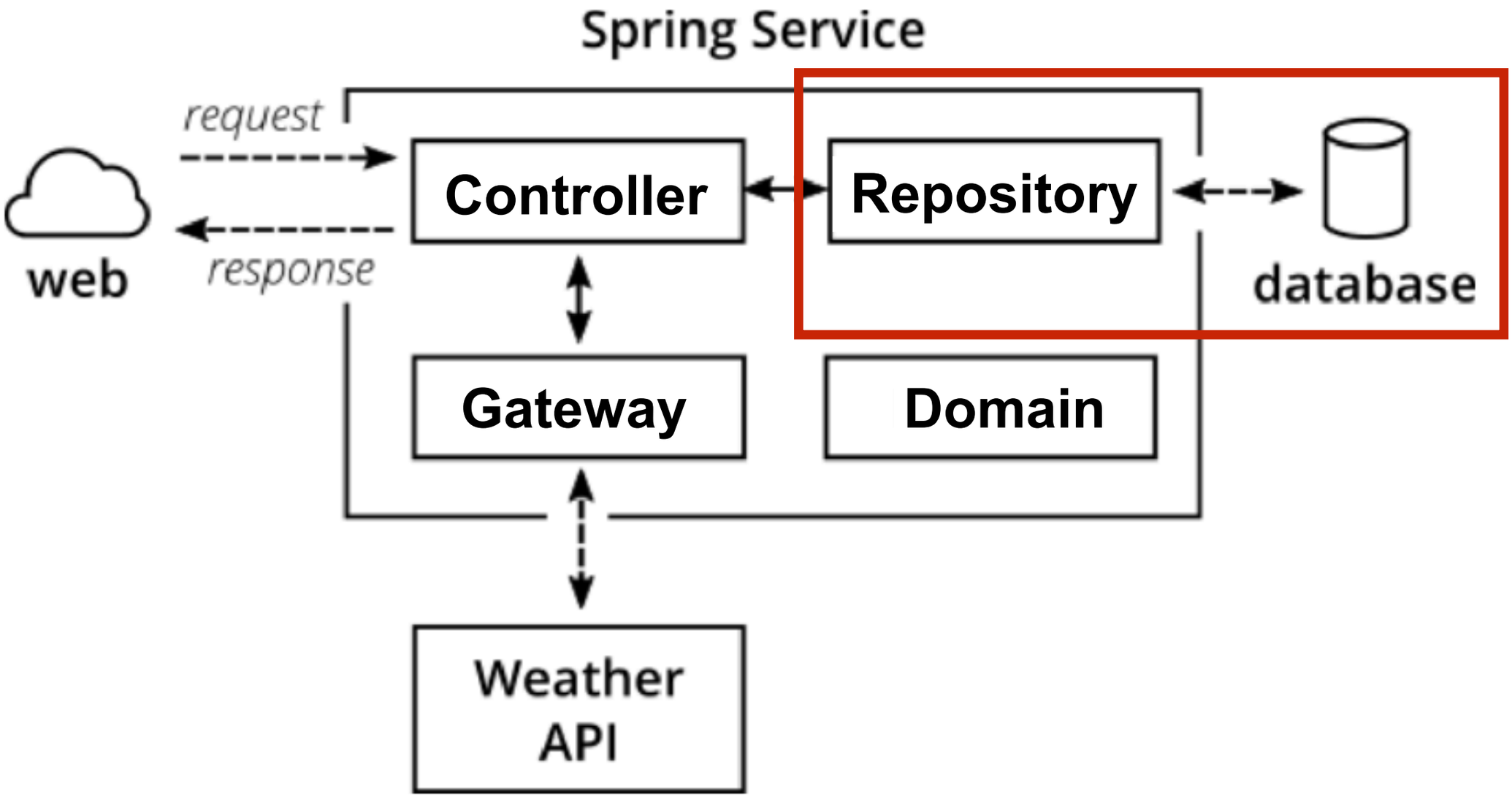
Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

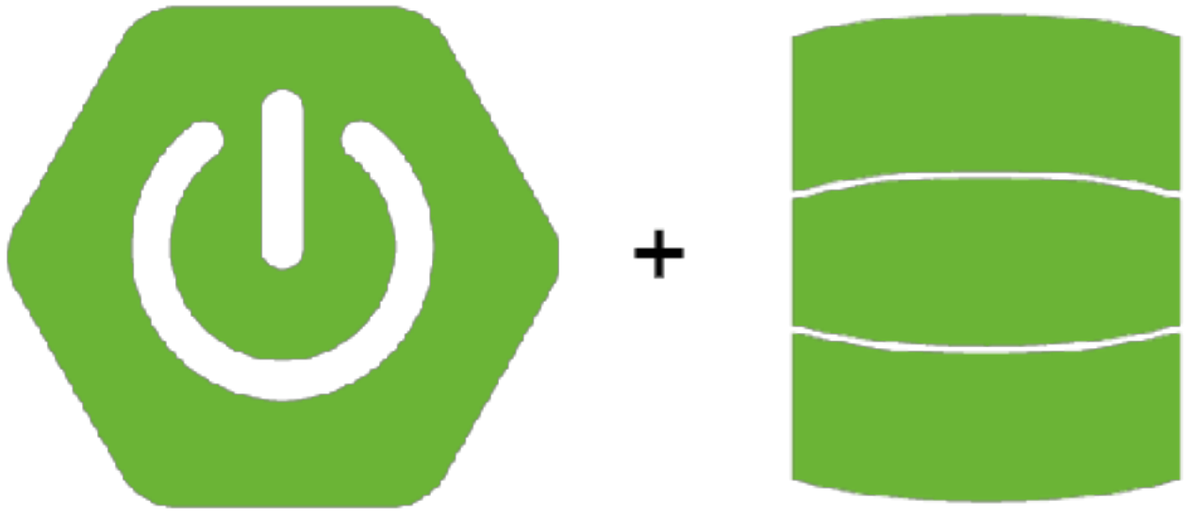
Let’s workshop

Working with repository



Working with repository

We’re using Spring Data



Modify pom.xml

Add library of Spring Data

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

Modify pom.xml

Add library of Persistence/data store

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.1.1</version>
</dependency>
```

Create repository interface

hello.repository.PersonRepository.java

```
public interface PersonRepository
        extends CrudRepository<Person, String> {

    Optional<Person> findByFirstName(String name);

}
```

Create Entity class

hello.repository.Person.java

```
@Entity
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String firstName;
    private String lastName;

    public Person() {
    }

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

Create new controller

hello.repository.HelloControllerWithRepository.java

```
public class HelloControllerWithRepository {

    private final PersonRepository personRepository;

    @Autowired
    public HelloControllerWithRepository(PersonRepository personRepository) {
        this.personRepository = personRepository;
    }

    @GetMapping("/hello/data/{name}")
    public Hello sayHi(@PathVariable String name) {
        Optional<Person> foundPerson = personRepository.findByName(name);
        String result = foundPerson
                .map(person -> String.format("Hello %s", person.getFirstName()))
                .orElse( other: "Data not found");
        return new Hello(result);
    }

}
```

Run test and package

$mvn clean package

```
Cobertura Report generation was successful.
Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file
Cobertura: Loaded information on 3 classes.
time: 125ms

Cobertura Report generation was successful.
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Coverage report

Packages

All
hello
hello.controller
hello.domain
hello.repository

All Packages

Classes

Hello *(66%)*
HelloApplication *(33%)*
HelloController *(100%)*
HelloControllerWithRepository *(0%)*
Person *(0%)*
PersonRepository *(N/A)*

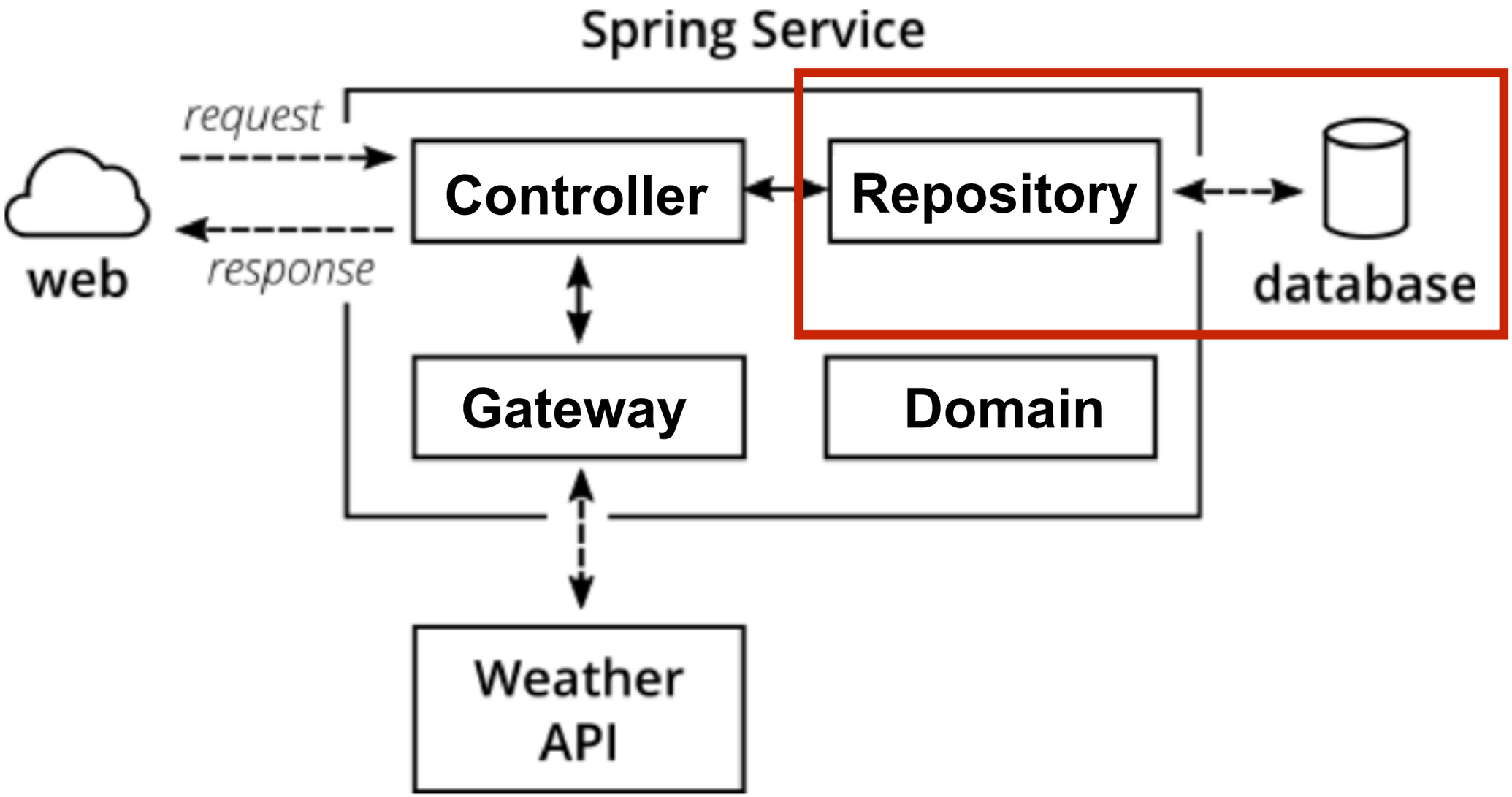
Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	6	25%	7/28	N/A	N/A	1
hello	1	33%	1/3	N/A	N/A	1
hello.controller	2	20%	2/10	N/A	N/A	1
hello.domain	1	66%	4/6	N/A	N/A	1
hello.repository	2	0%	0/9	N/A	N/A	1

Report generated by Cobertura 2.1.1 on 3/6/18 8:52 AM.

How to test ?

How to test with Repository ?



Spring boot provide DataJpaTest

should be add H2 library to pom.xml

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```

Repository Testing (1)

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class PersonRepositoryTest {

    @Autowired
    private PersonRepository personRepository;

    @After
    public void clearData() {
        personRepository.deleteAll();
    }
```

Repository Testing (2)

Add a test case

```
@Test
public void shouldSaveAndGetData() throws Exception {
    //Arrange
    Person somkiat = new Person("somkiat", "pui");
    personRepository.save(somkiat);

    Optional<Person> shouldSomkiat
            = personRepository.findByFirstName("somkiat");

    assertEquals( expected: "somkiat",
            shouldSomkiat.get().getFirstName());

}
```

Run test and package

$mvn clean package

```
Cobertura Report generation was successful.
Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file
Cobertura: Loaded information on 3 classes.
time: 125ms

Cobertura Report generation was successful.
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Coverage report

Packages

All
hello
hello.controller
hello.domain
hello.repository

All Packages

Classes

Hello *(66%)*
HelloApplication *(33%)*
HelloController *(100%)*
HelloControllerWithRepository *(0%*
Person *(88%)*
PersonRepository *(N/A)*

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	6	53%	15/28	N/A	N/A	1
hello	1	33%	1/3	N/A	N/A	1
hello.controller	2	20%	2/10	N/A	N/A	1
hello.domain	1	66%	4/6	N/A	N/A	1
hello.repository	2	88%	8/9	N/A	N/A	1

Report generated by Cobertura 2.1.1 on 3/6/18 9:32 AM.

Run your application

$java -jar target/hello.jar

```
org.postgresql.util.PSQLException: Connection to 127.0.0.1:15432 refused. Check that th
 the postmaster is accepting TCP/IP connections.
        at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFa
jar!/:42.1.1]
        at org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java:
        at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:194) ~[postgresql-
        at org.postgresql.Driver.makeConnection(Driver.java:450) ~[postgresql-42.1.1.ja
        at org.postgresql.Driver.connect(Driver.java:252) ~[postgresql-42.1.1.jar!/:42.
        at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:
        at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:
        at com.zaxxer.hikari.pool.PoolBase.newConnection(PoolBase.java:365) [HikariCP-2
        at com.zaxxer.hikari.pool.PoolBase.newPoolEntry(PoolBase.java:194) [HikariCP-2.
        at com.zaxxer.hikari.pool.HikariPool.createPoolEntry(HikariPool.java:460) [Hika
        at com.zaxxer.hikari.pool.HikariPool.checkFailFast(HikariPool.java:534) [Hikari
        at com.zaxxer.hikari.pool.HikariPool.<init>(HikariPool.java:115) [HikariCP-2.7.
        at com.zaxxer.hikari.HikariDataSource.getConnection(HikariDataSource.java:112)
```

Working with container

Configuration

Monitoring and Metric

Tracing

Service breaker