

Java Framework





Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata



Facebook somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc
@somkiat.cc

Home Posts Videos Photos

Liked Following Share ... + Add a Button

Help people take action on this Page. X



Agenda

- RESTful API
- Java frameworks
- Building RESTful API with Spring Boot
- Spring Boot with Spring Data (JDBC/JPA)
- Spring Boot with Spring Data (MongoDB)
- Testing with Spring Boot
- Q/A



REST



REST

REpresentation **S**tate **T**ransfer

The style of software architecture behind
RESTful services

Defined in 2000 by Roy Fielding

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm



Goals

Scalability
Generality of interfaces
Independent deployment of components



RESTful service



REST Request Messages

RESTful request is typically in form of
Uniform Resource Identifiers (URI)



REST Request Messages

RESTful request is typically in form of
Uniform Resource Identifiers (URI)

Structure of URI depend on specific service



REST Request Messages

RESTful request is typically in form of
Uniform Resource Identifiers (URI)

Structure of URI depend on specific service

Request can include parameter and data in
body of request as XML, JSON etc.



REST Request & Response

Request Method

GET, POST, PUT, DELETE

Client

Server



Response Format

XML or JSON



HTTP Methods meaning

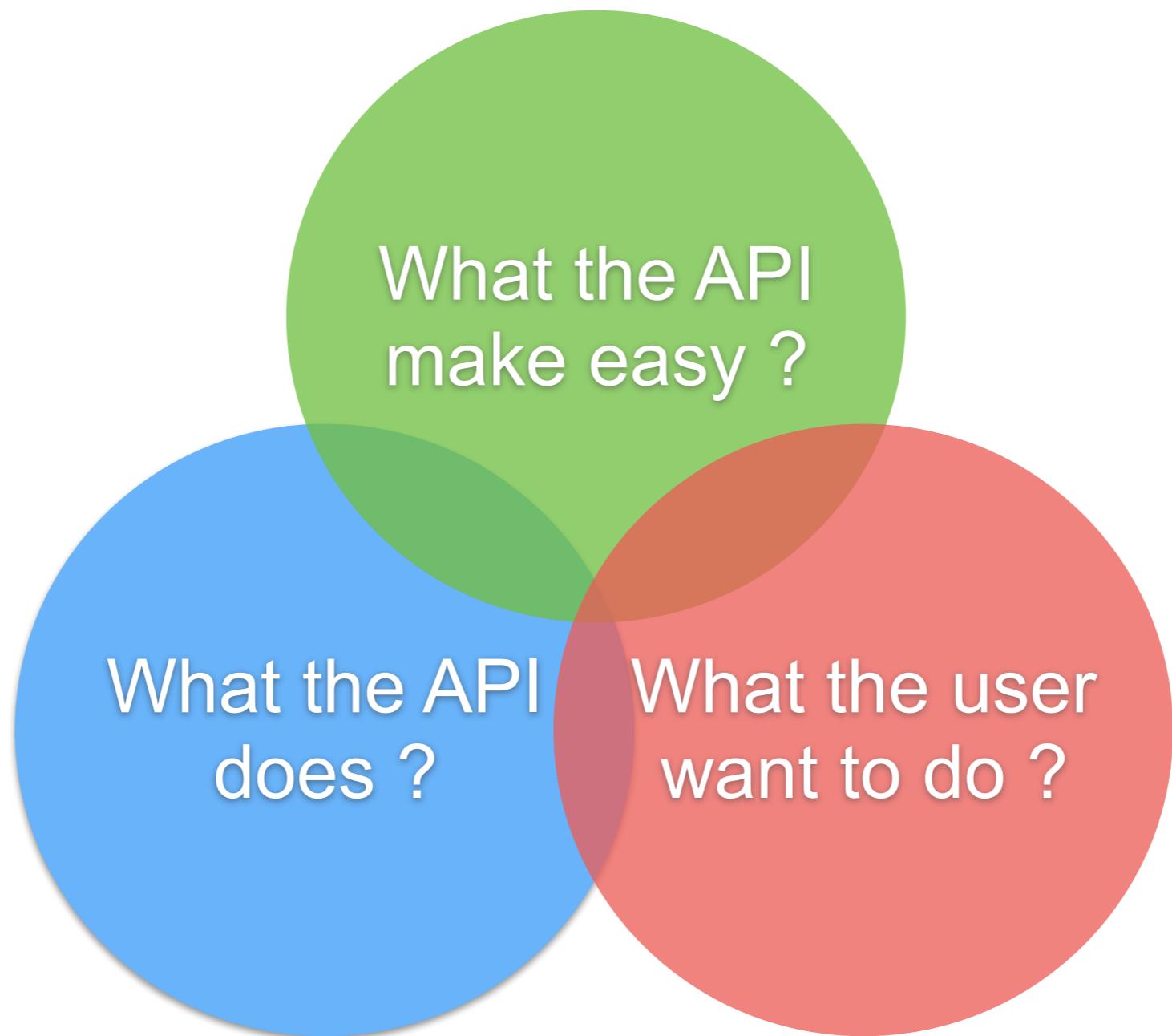
Method	Meaning
GET	Read data
POST	Create/Insert new data
PUT/PATCH	Update data or insert if a new id
DELETE	Delete data



Response format ?



Good APIs ?



Java Framework for RESTful



unirest

Lightweight HTTP Request Client Libraries



Dropwizard



Restlet



ACT.framework

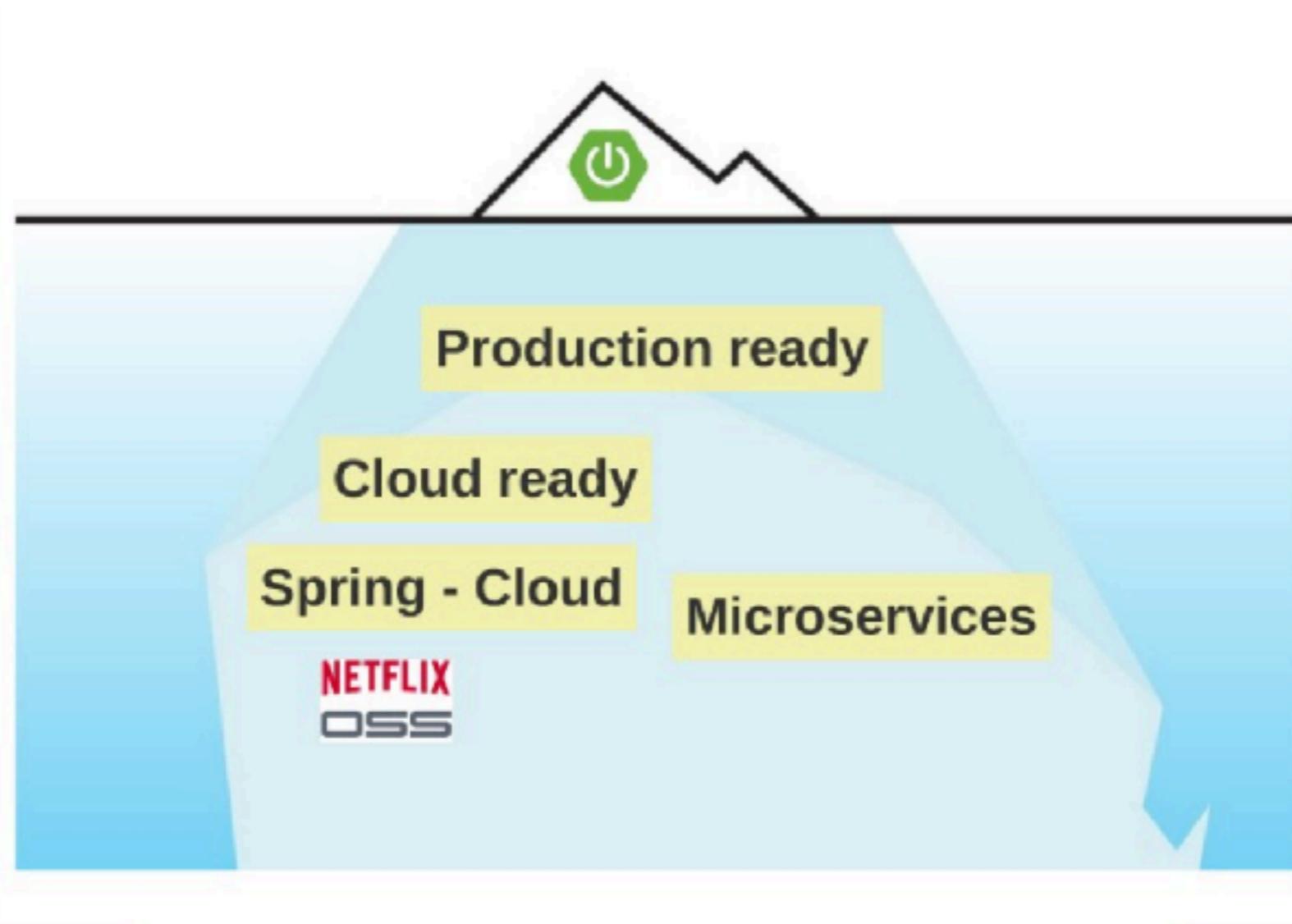


Hello Spring Boot 2.x



Why ?

Application skeleton generator
Reduce effort to add new technologies



What ?

Embedded application server

Integration with tools/technologies (starter)

Production tools (monitoring, health check)

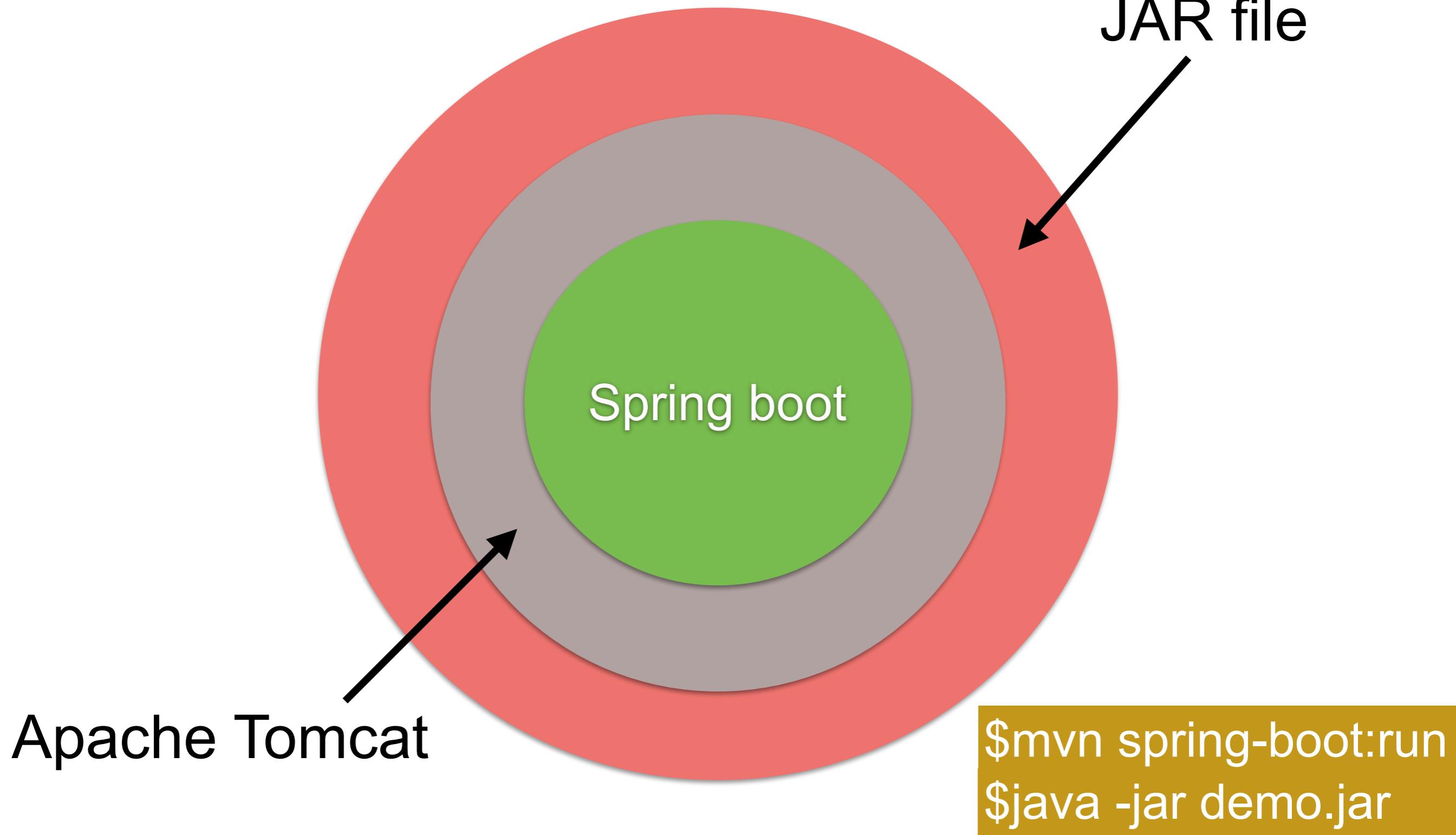
Configuration management

Dev tools

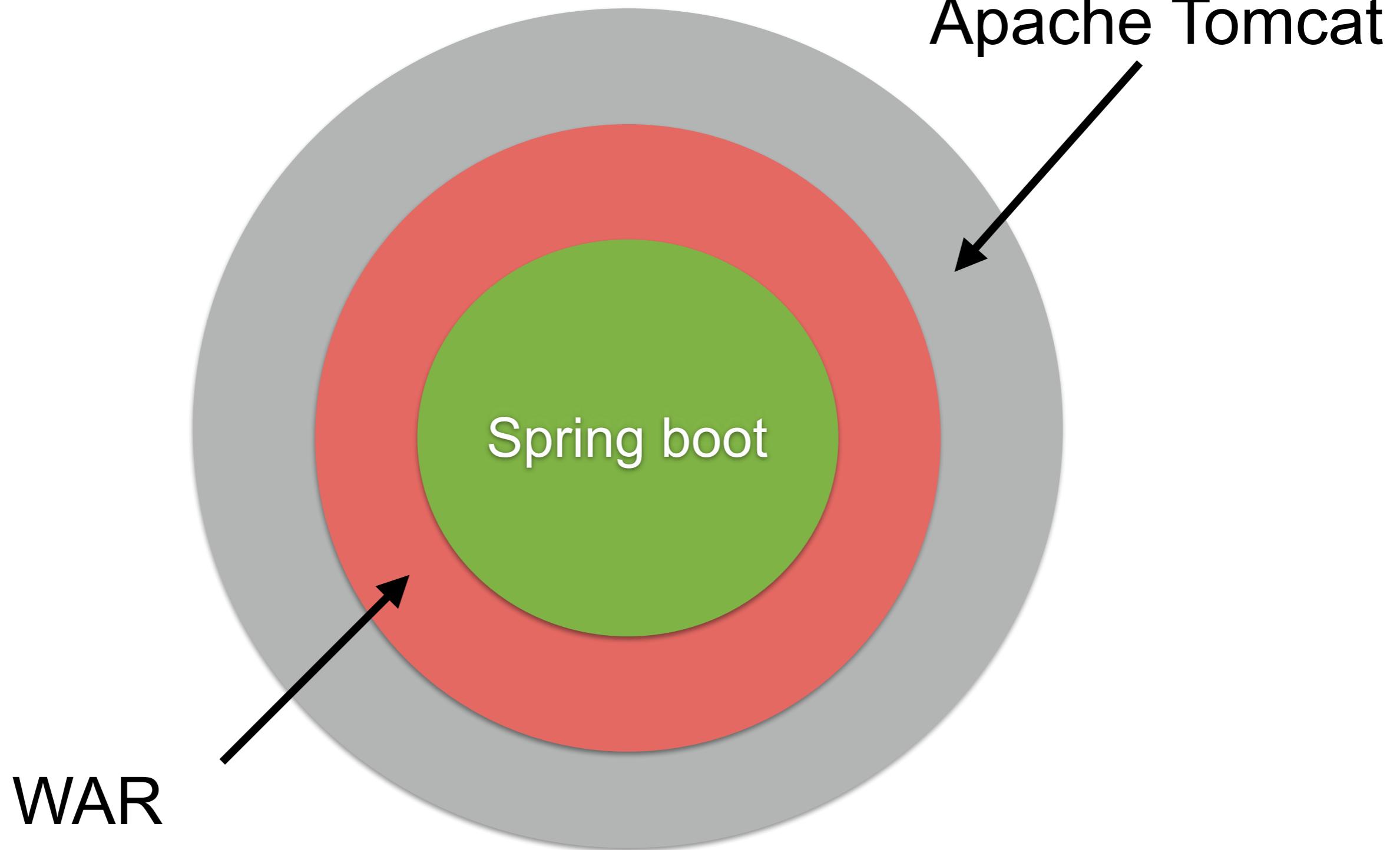
No source code generation, no XML



How ?



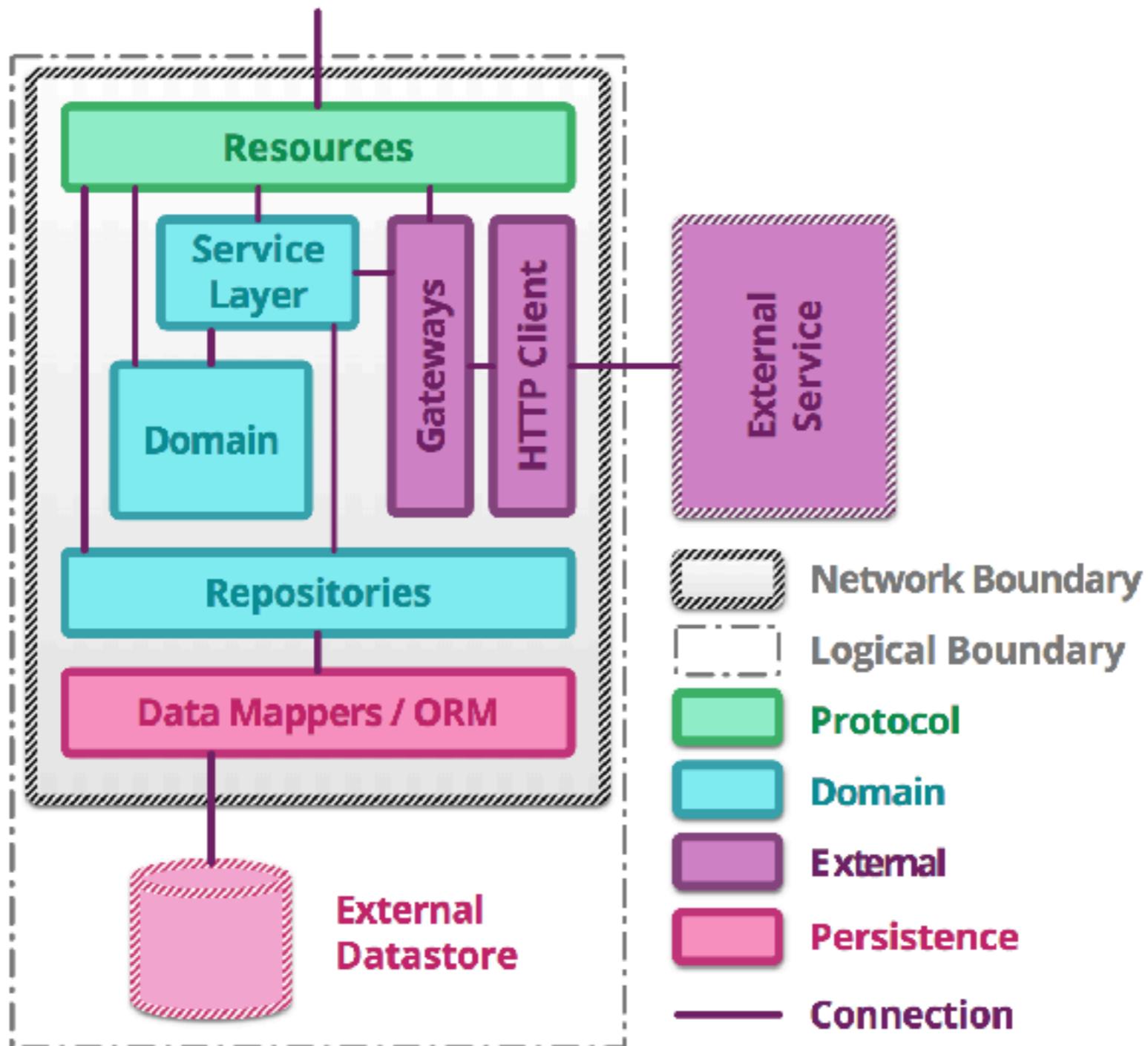
How ?



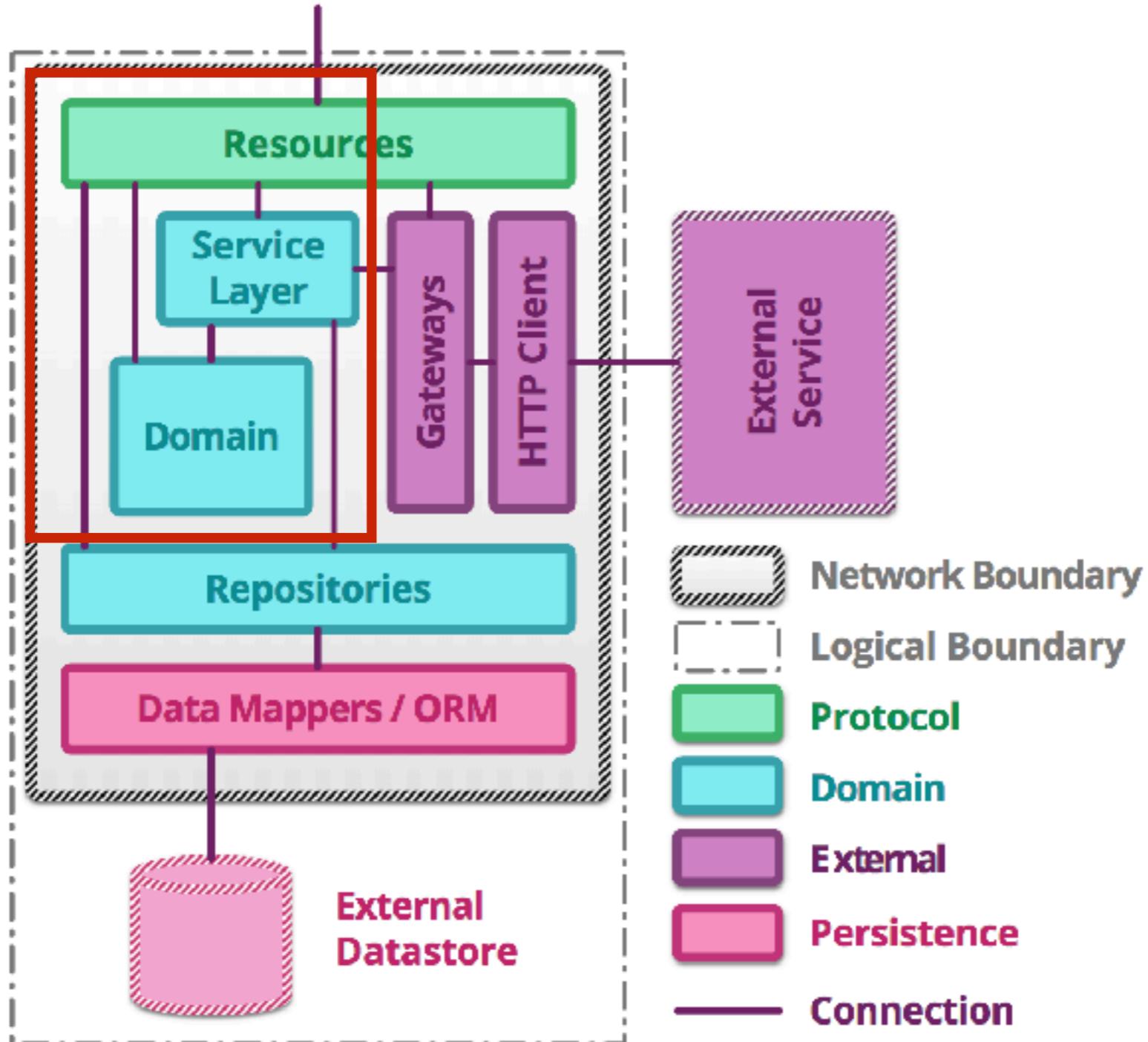
Building RESTful API with Spring Boot



Service Structure



Service Structure



Create project with Spring Initializr



Spring Initializr

<https://start.spring.io/>

SPRING INITIALIZR bootstrap your application now

Generate a with and Spring Boot

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

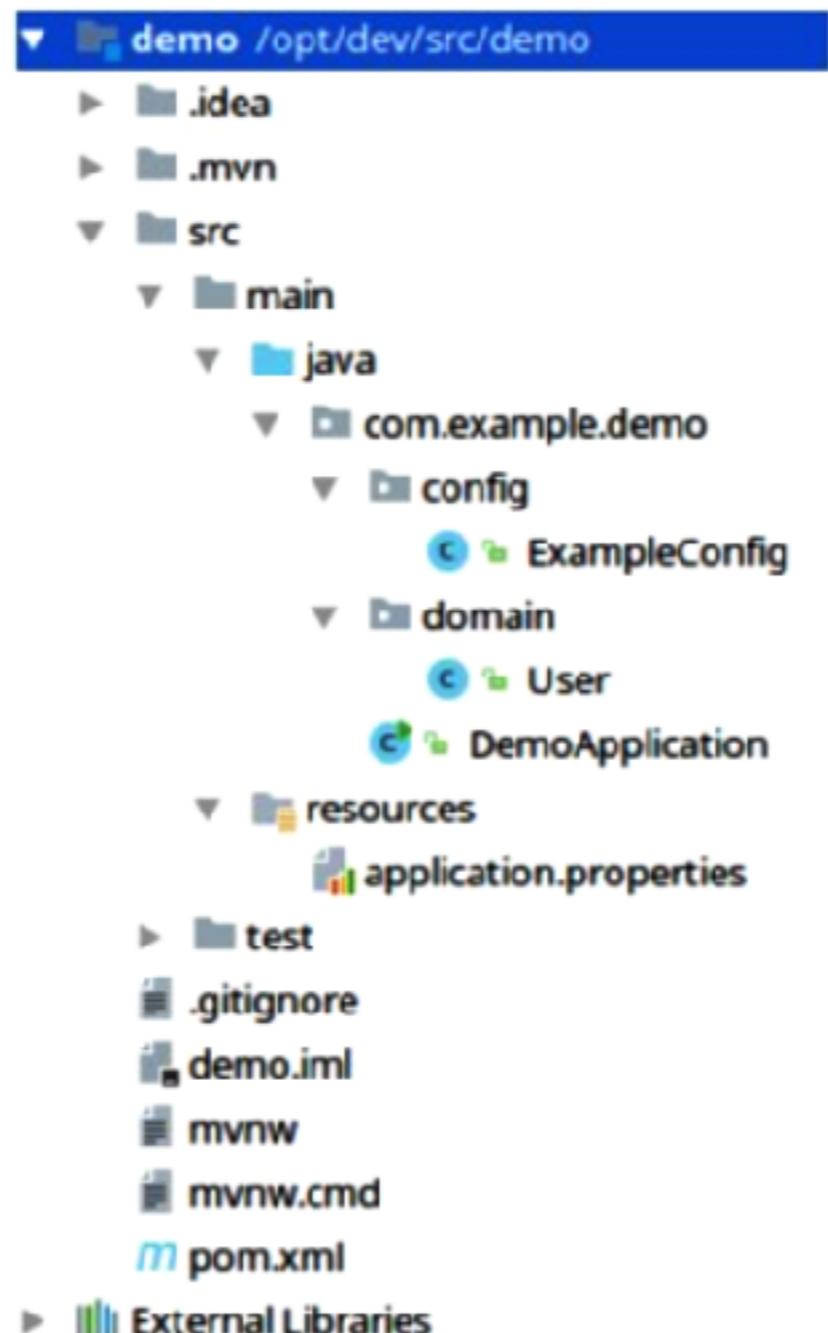
Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

start.spring.io is powered by [Spring Initializr](#) and [Pivotal Web Services](#)



Structure of Spring Boot



Spring Boot main class

```
package com.example.demo;

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

@SpringBootApplication
public class DemoApplication {

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



Run project (Dev mode)

./mvnw spring-boot:run

```
2018-06-07 13:03:30.412  INFO 12828 --- [  
oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
oApplication           : No active profile set, fall
```



Run project (production mode)

```
./mvnw package  
$java -jar target/<file name>.jar
```

```
2018-06-07 13:03:30.412  INFO 12828 --- [  
oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
oApplication           : No active profile set, fall
```



Custom Banner



Custom banner

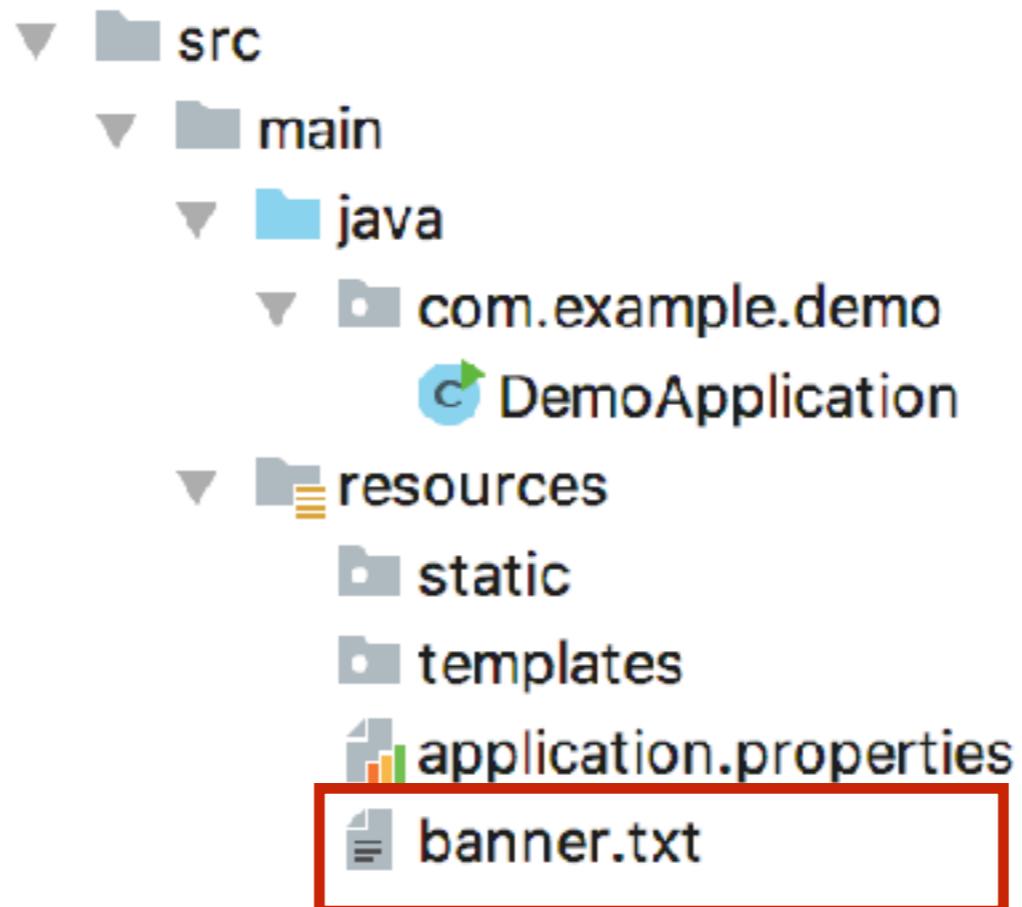
The Spring Boot logo is a complex arrangement of various characters including slashes, parentheses, brackets, and underscores, all connected by a network of lines to form a circular, organic shape.

```
2018-06-07 13:03:30.412  INFO 12828 --- [  
  oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
  oApplication           : No active profile set, fall
```



Custom banner (1)

Create file banner.txt or banner.png in resources folder



Custom banner (2)

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        ImageBanner banner = new ImageBanner(
            new ClassPathResource("try.png"));

        new SpringApplicationBuilder()
            .sources(DemoApplication.class)
            .banner(banner)
            .run();
    }
}
```



Disable banner

```
@SpringBootApplication
public class DemoApplication {

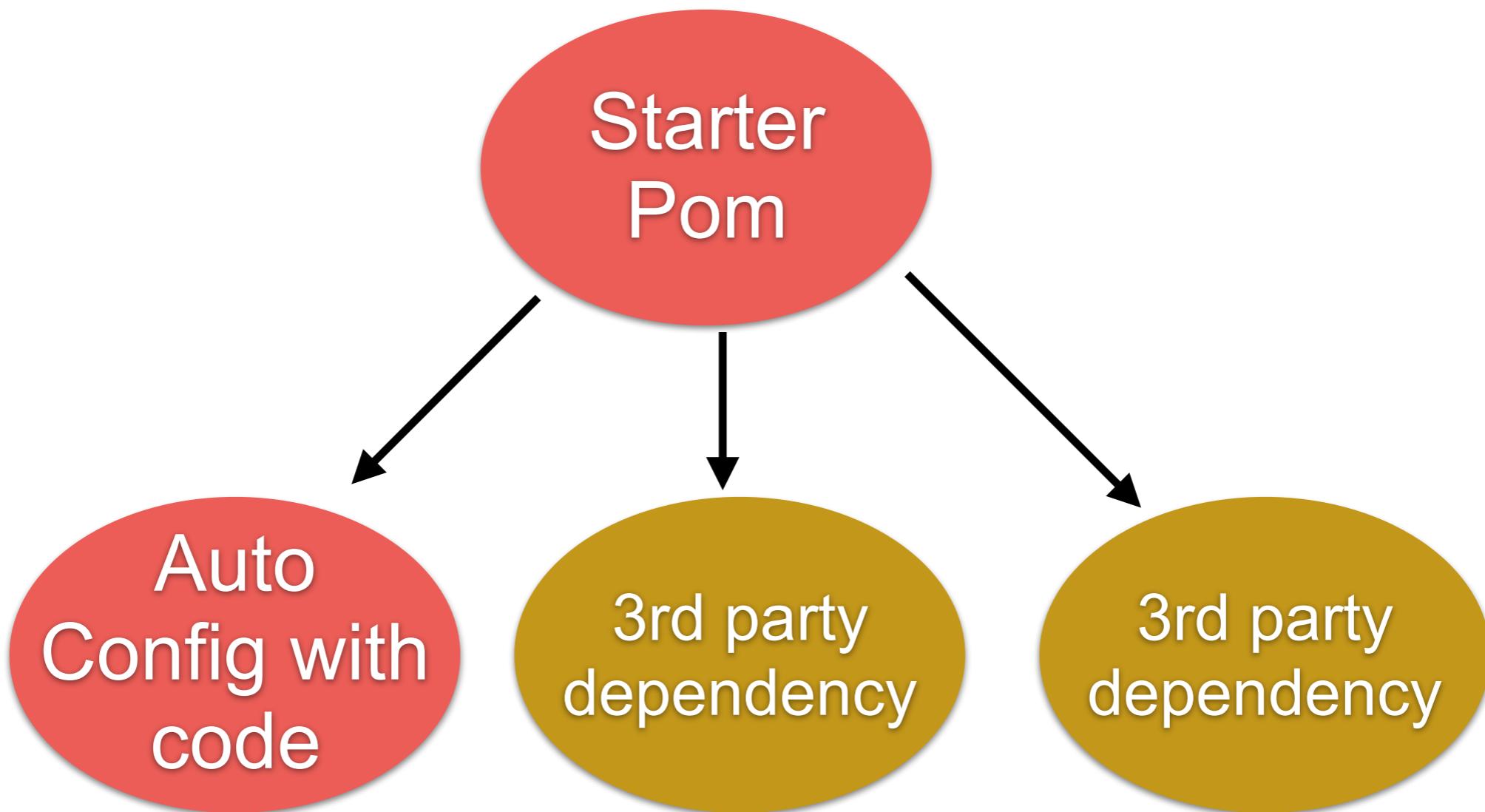
    public static void main(String[] args) {
        new SpringApplicationBuilder()
            .sources(DemoApplication.class)
            .bannerMode(Banner.Mode.OFF)
            .run(args);
    }
}
```



Anatomy of Starter



Anatomy of Starter



Configuration management

Properties/XML/YAML

Config server (App, Spring cloud config, git)

17 ways !!!

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html>



Look at POM.xml



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>
```



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>

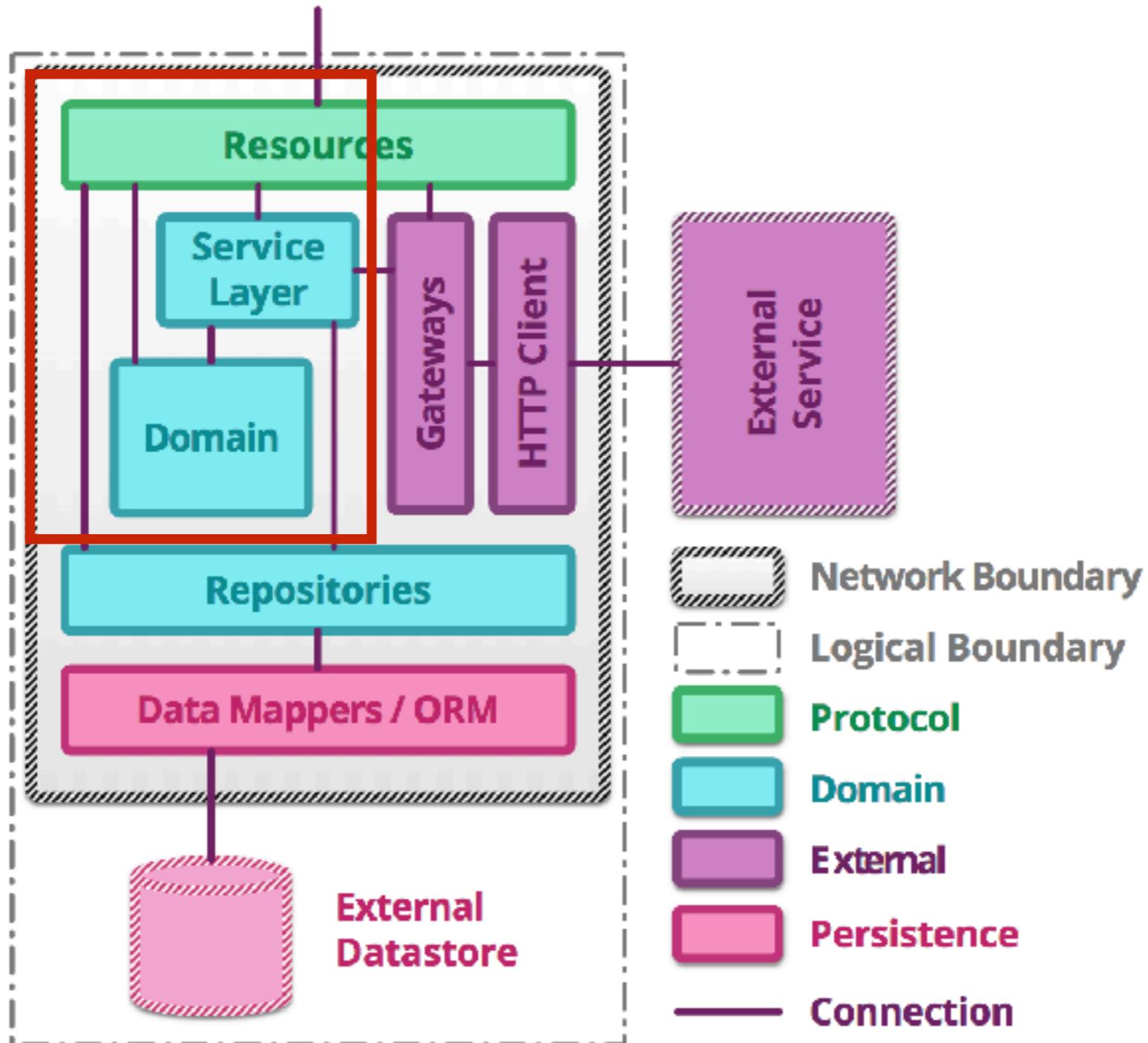
<build>
    <finalName>hello</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```



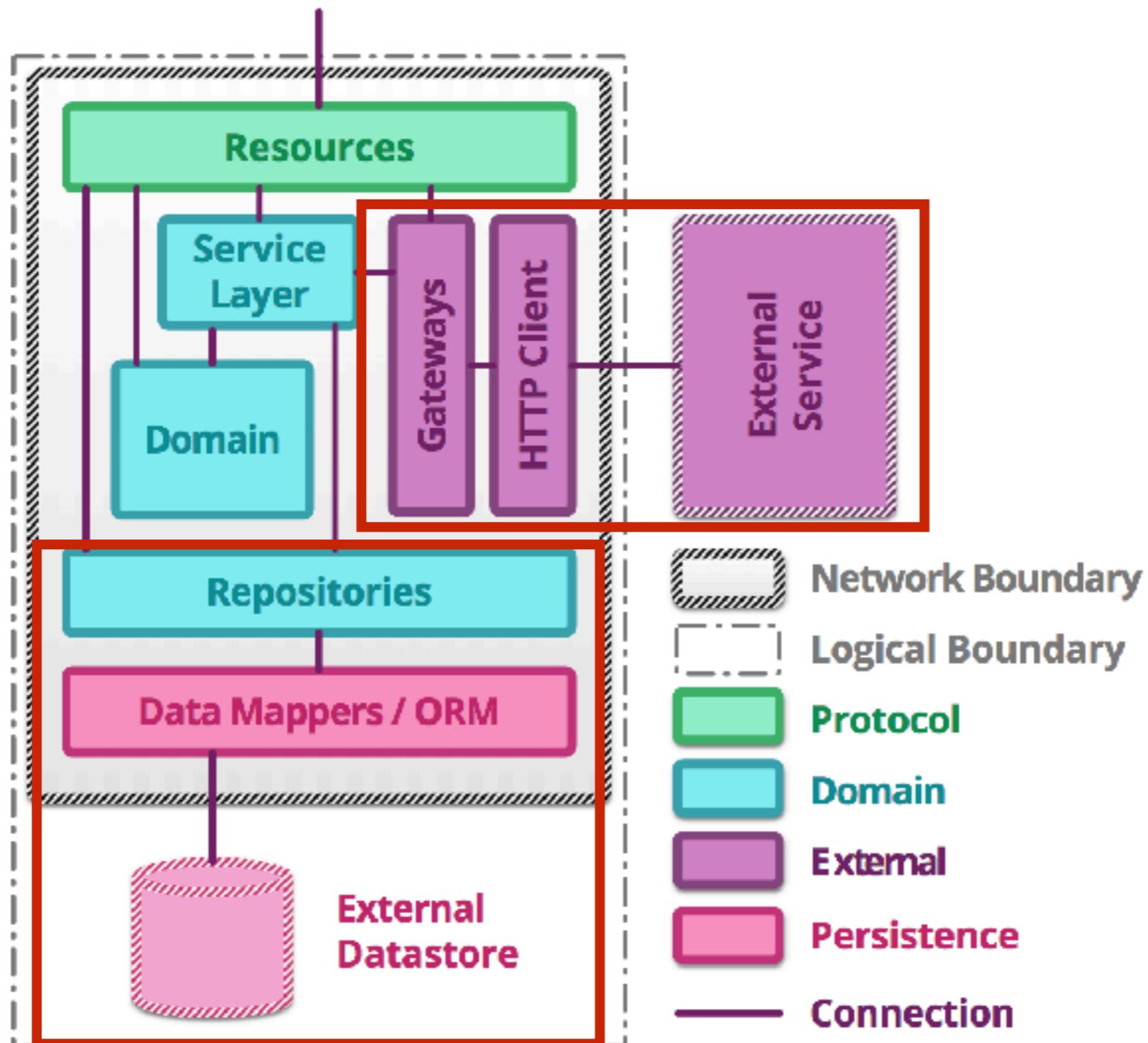
Basic structure of Spring Boot



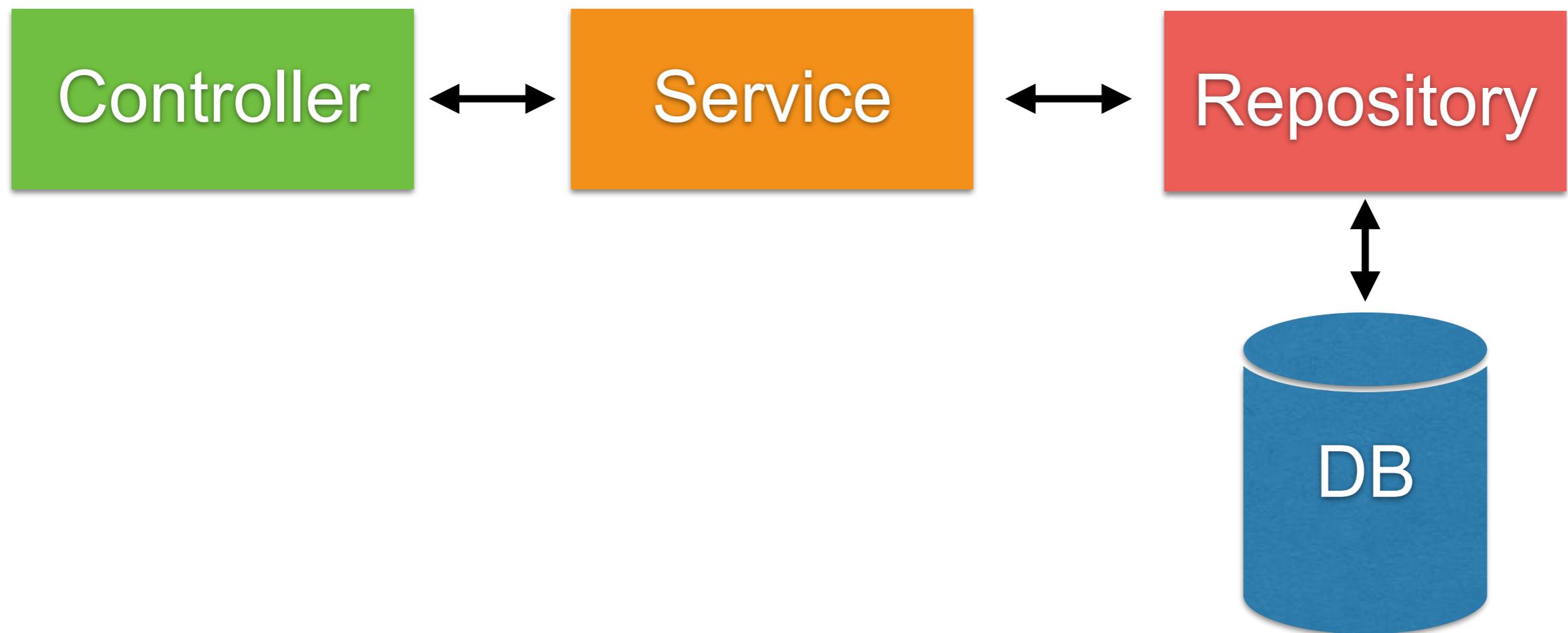
Service Structure



Service Structure



Basic structure of Spring Boot



Controller

Request and Response
Validation input

Delegate to others classes such as service and repository



Service

Control the flow of main business logic
Manage database transaction
Don't reuse service



Repository

Manage data in data store such as RDBMS and
NoSQL



Gateway

Call external service via network such as
WebServices and RESTful APIs



Spring Boot Structure (1)

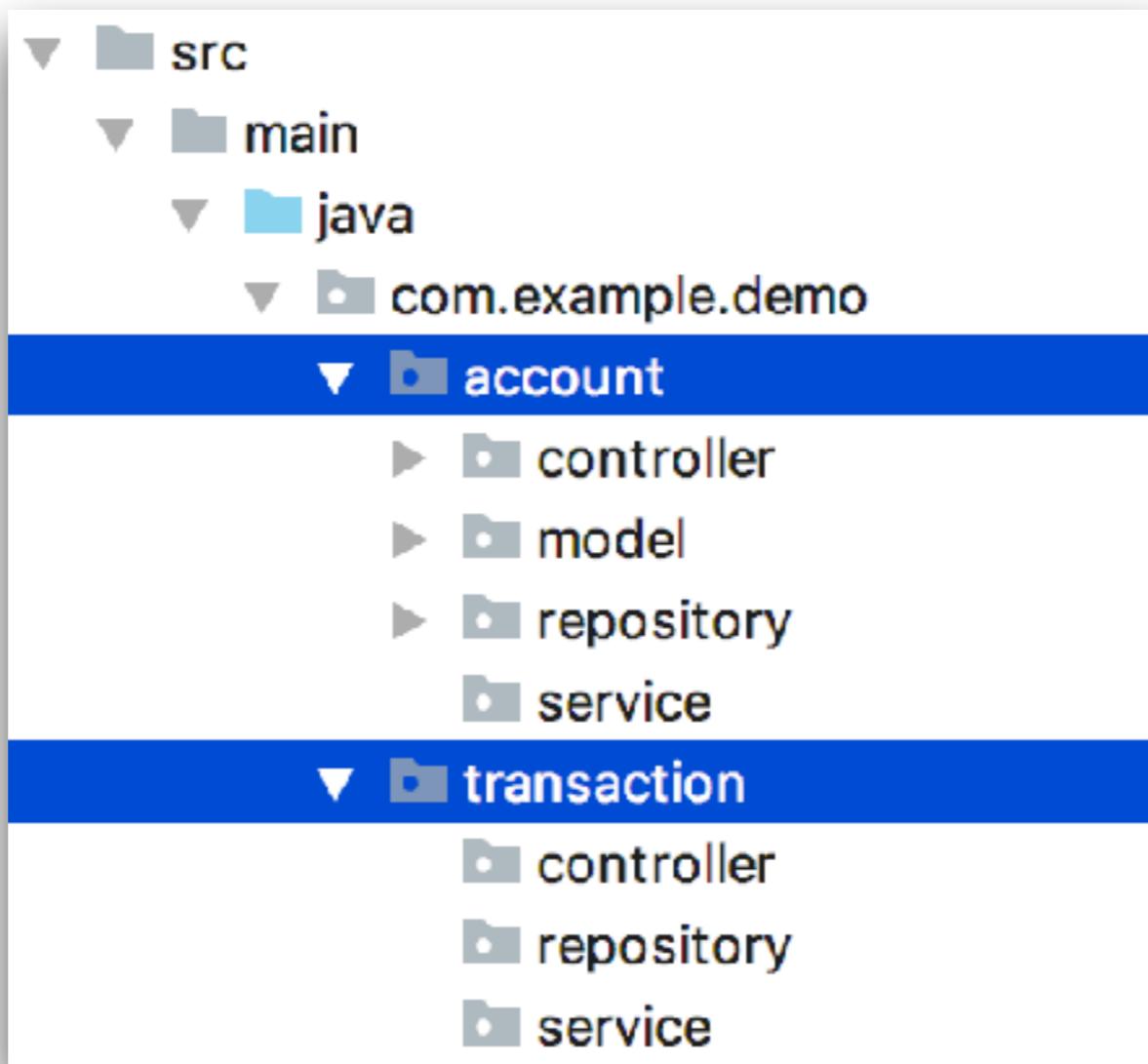
Separate by function/domain/feature

- feature1**
 - controller
 - service
 - repository
- feature2**
 - controller
 - service
 - repository



Spring Boot Structure (2)

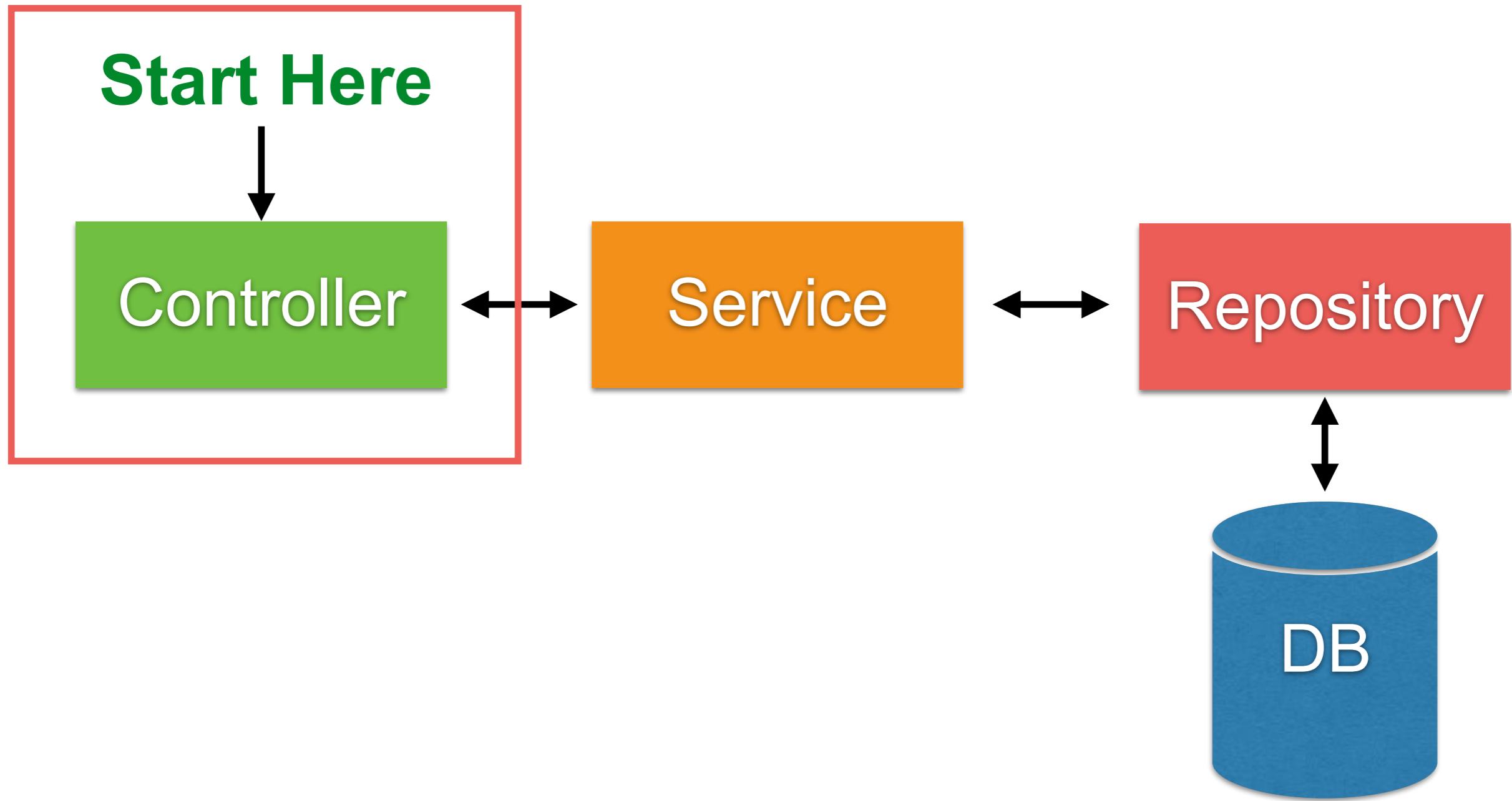
Separate by function/domain/feature



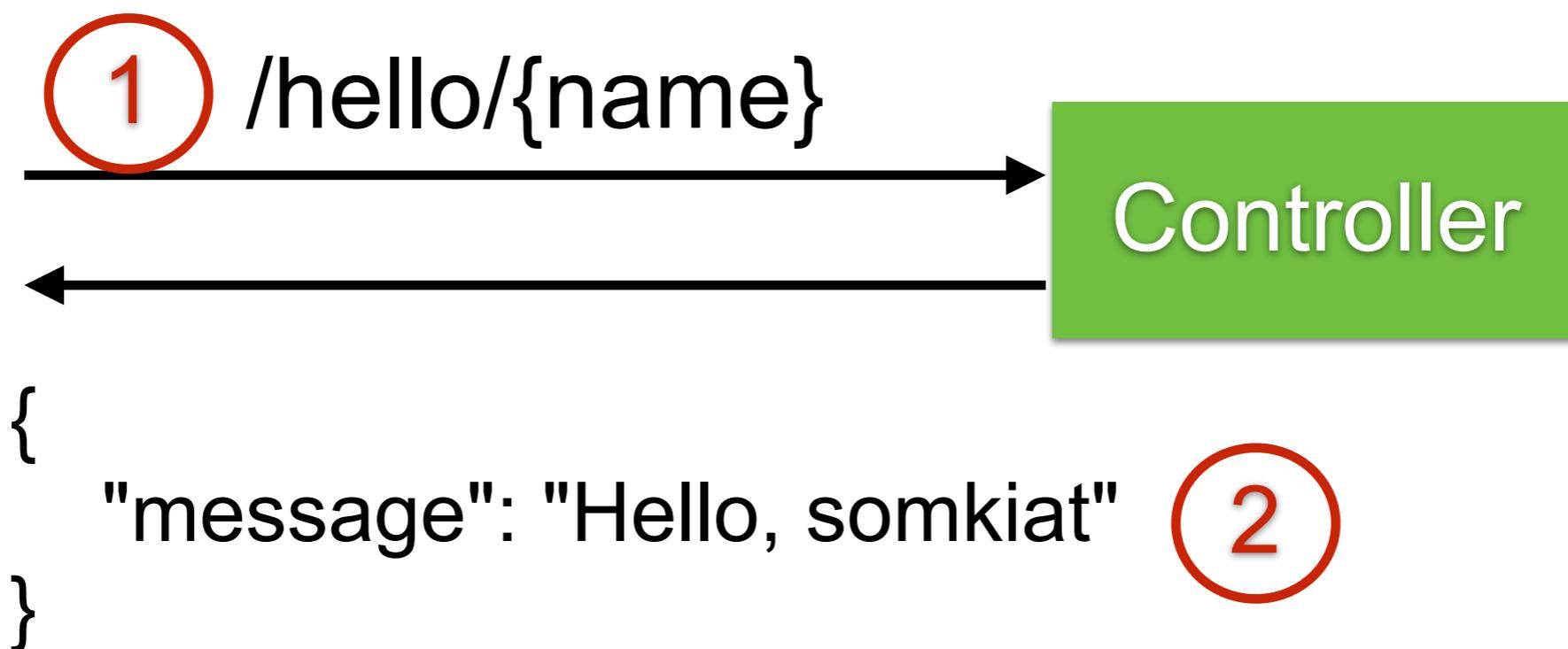
Create first RESTful API



Basic structure of Spring Boot



Hello API



1. Create REST Controller

HelloController.java

```
@RestController
public class HelloController {

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

}
```



2. Create model class

Hello.java

```
public class Hello {  
  
    private String message;  
  
    Hello(String message) {  
        this.message = message;  
    }  
  
    public String getMessage() {  
        return message;  
    }  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
}
```



3. Compile and Packaging

\$mvnw clean package



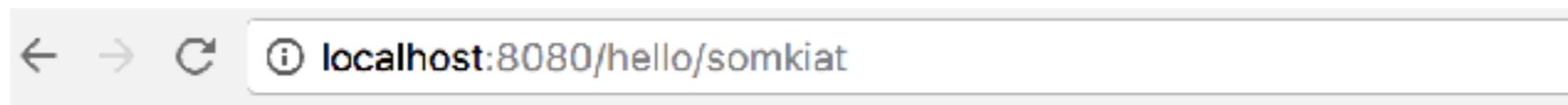
4. Run

```
$java -jar target/hello.jar
```



5. Open in browser

<http://localhost:8080/hello/somkiat>

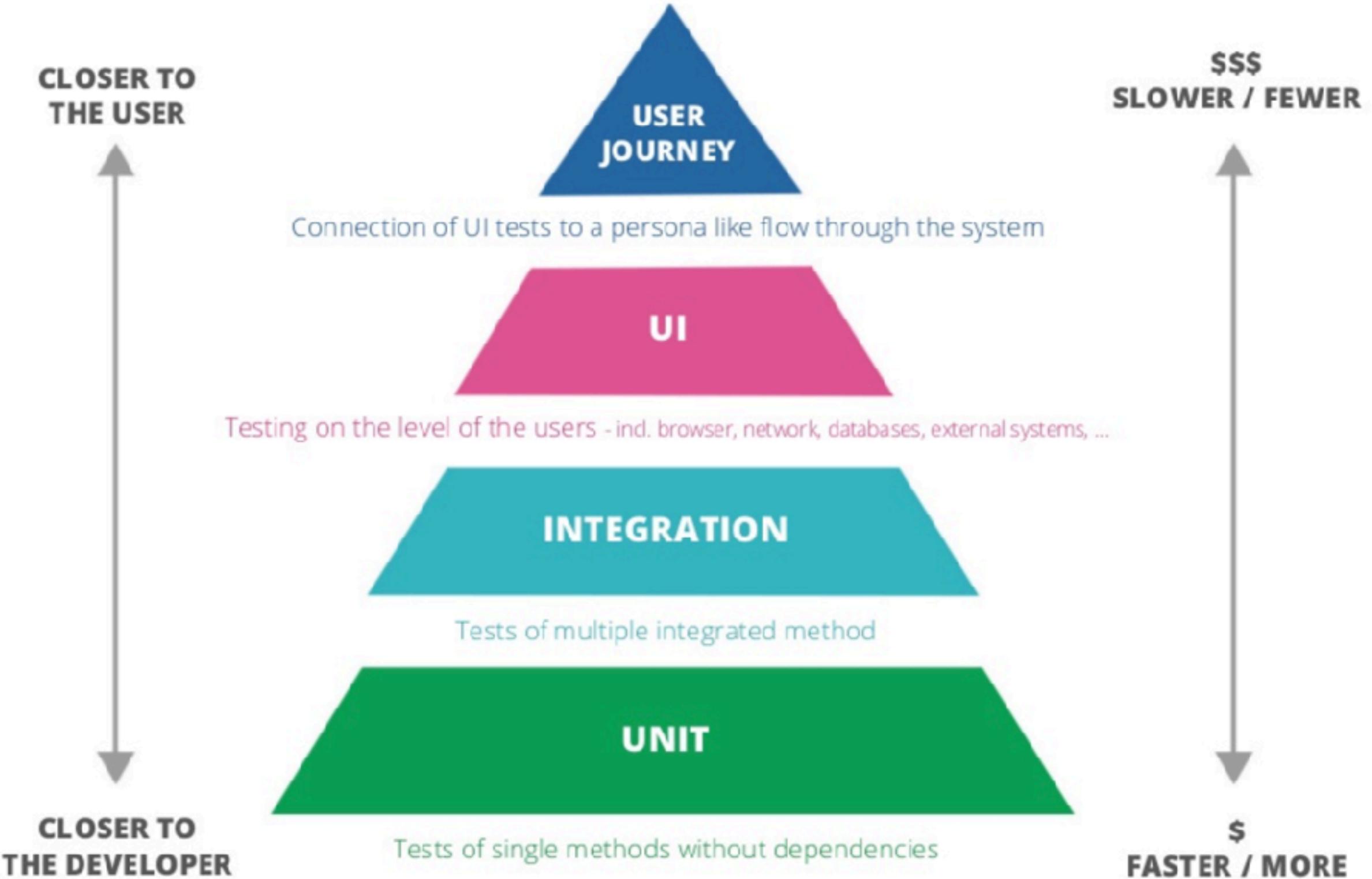


{ "message": "Hello somkiat" }



How to test the Hello service ?





Unit tests

How to use model ?

```
public class HelloTest {  
  
    @Test  
    public void success_to_create_model_with_constructor() {  
        Hello hello = new Hello("Somkiat");  
        assertEquals( expected: "Somkiat", hello.getMessage());  
    }  
  
}
```



Controller testing

How to testing with Spring Boot ?



Testing in Spring Boot

`@SpringBootTest`
`@WebMVCTest`
`@JsonTest`
`@DataJpaTest`
`@RestClientTest`



Testing in Spring Boot

@SpringBootTest

@WebMVC Test

@JsonTest

@DataJpaTest

@RestClientTest

Slice testing



SpringBootTest

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment
        = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    public void sayHi() {
        // Action :: Call controller
        Hello actualResult
                = testRestTemplate.getForObject("/hello/somkiat",
                                                Hello.class);

        // Assertion :: Check result with expected result
        assertEquals("Hello, somkiat", actualResult.getMessage());
    }
}
```



MockMvcTest #1

```
@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());
    }

}
```



MockMvcTest #2

```
@RunWith(SpringRunner.class)
@WebMvcTest/controllers = HelloController.class)
public class HelloController2Test {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void success_with_statuscode_200() throws Exception {
        MockHttpServletResponse response
            = mockMvc.perform(get("/hello/somkiat"))
                .andReturn().getResponse();

        assertEquals(HttpStatus.OK.value(), response.getStatus());
    }
}
```



MockMvcTest #2

Use **JacksonTester** to convert object to JSON message

```
private JacksonTester<Hello> jsonHello;

@Test
public void success_with_correct_json_data() throws Exception {
    JacksonTester.initFields(this, new ObjectMapper());

    MockHttpServletResponse response = mockMvc.perform(get("/hello/somkiat"))
        .andReturn().getResponse();

    String expected = jsonHello.write(new Hello("somkiat")).getJson();
    assertEquals(expected, response.getContentAsString());
}
```



Compile with testing

\$mvn clean test

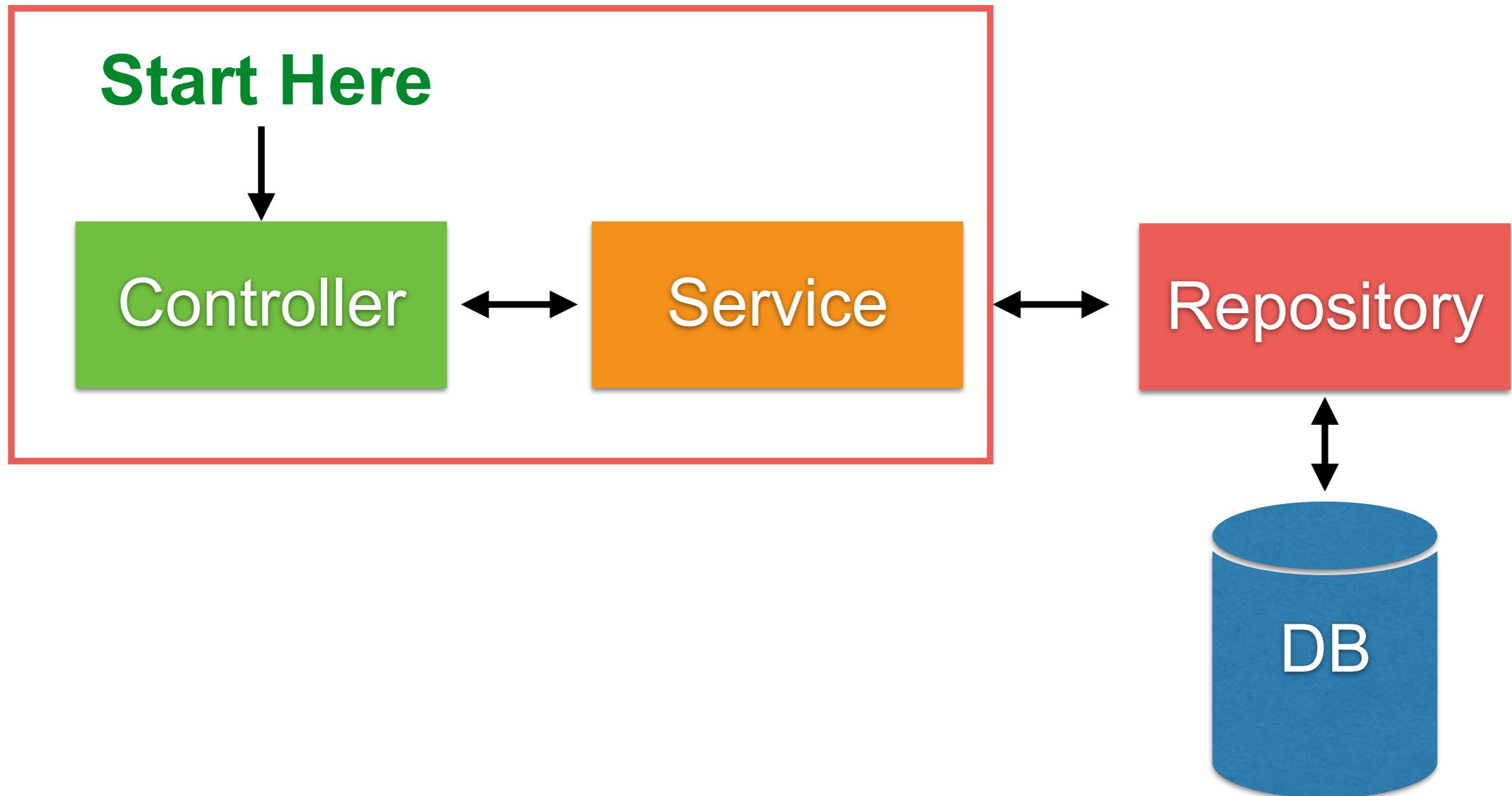
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
```



Move business logic to service

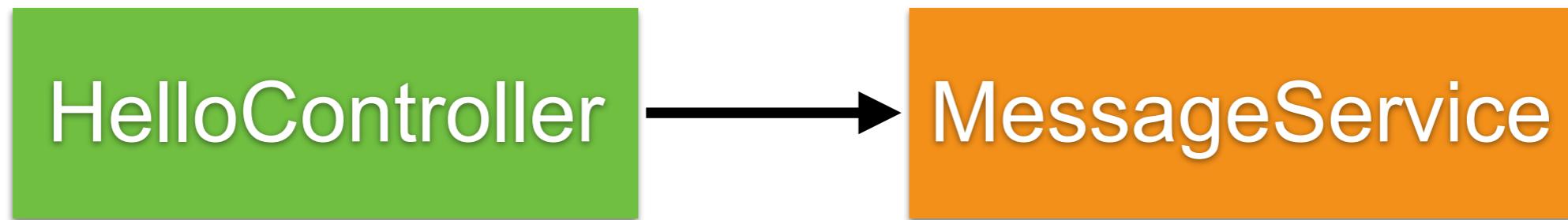


Working with service



Move business logic to service

MessageService.java



MessageService.java

```
@Service
public class MessageService {

    public String concat(String name) {
        return "Hello, " + name;
    }

}
```



Edit HelloController.java

Call method from service

```
@Autowired  
private MessageService messageService;  
  
 @GetMapping("/hello/{name}")  
 public HelloResponse  
     sayHi(@PathVariable String name) {  
         String result = messageService.concat(name);  
         return new HelloResponse(result);  
    }
```



Run all tests !!

\$mvnw clean test



Fix fail test cases



Testing controller with service

Working with @WebMvcTest

```
@Autowired  
private MockMvc mockMvc;
```

```
@MockBean  
private MessageService messageService; 1
```

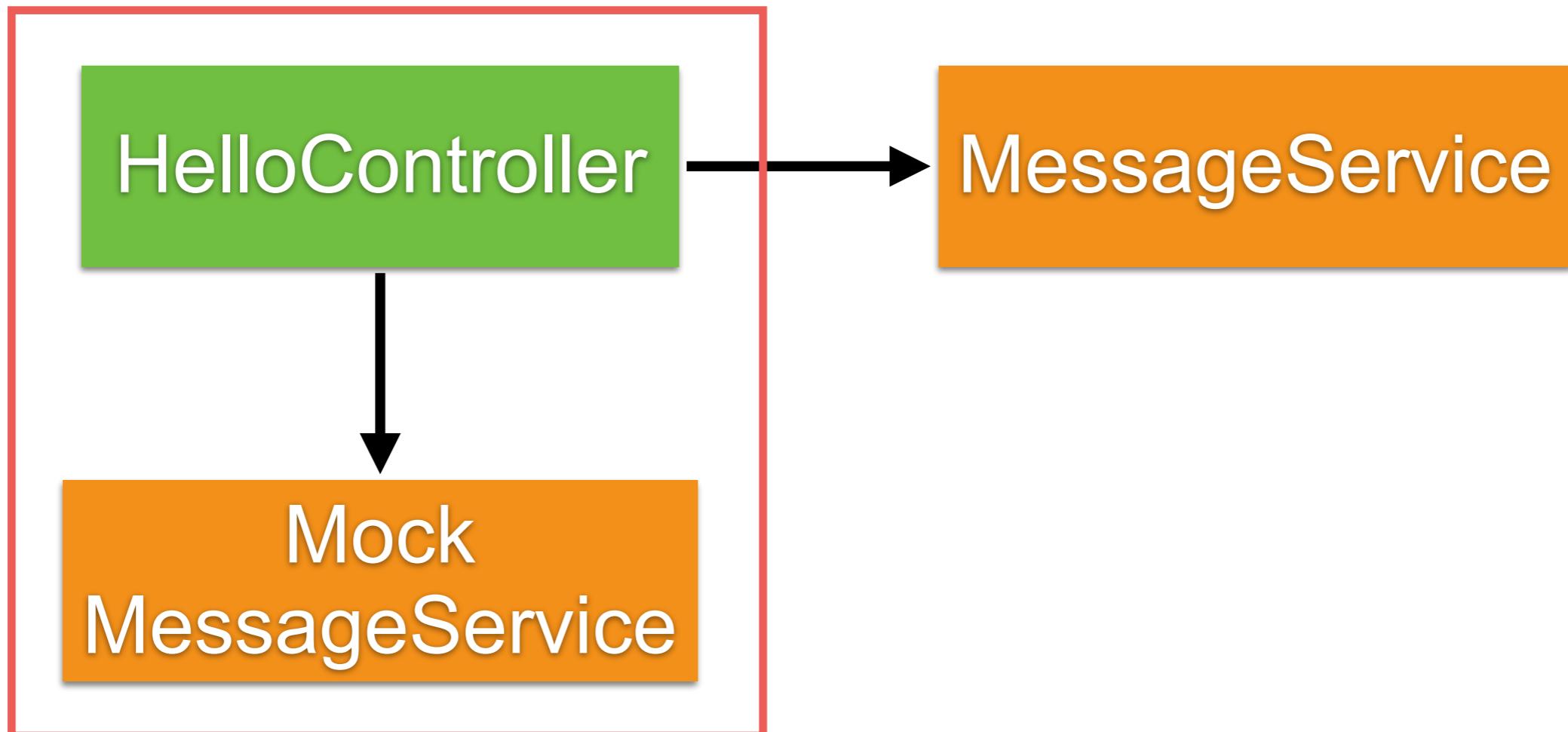
```
private JacksonTester<HelloResponse> tester;
```

```
@Before  
public void initial() {  
    JacksonTester.initFields(this, new ObjectMapper());  
    given(messageService.concat("somkiat"))  
        .willReturn("Hello, somkiat"); 2  
}
```



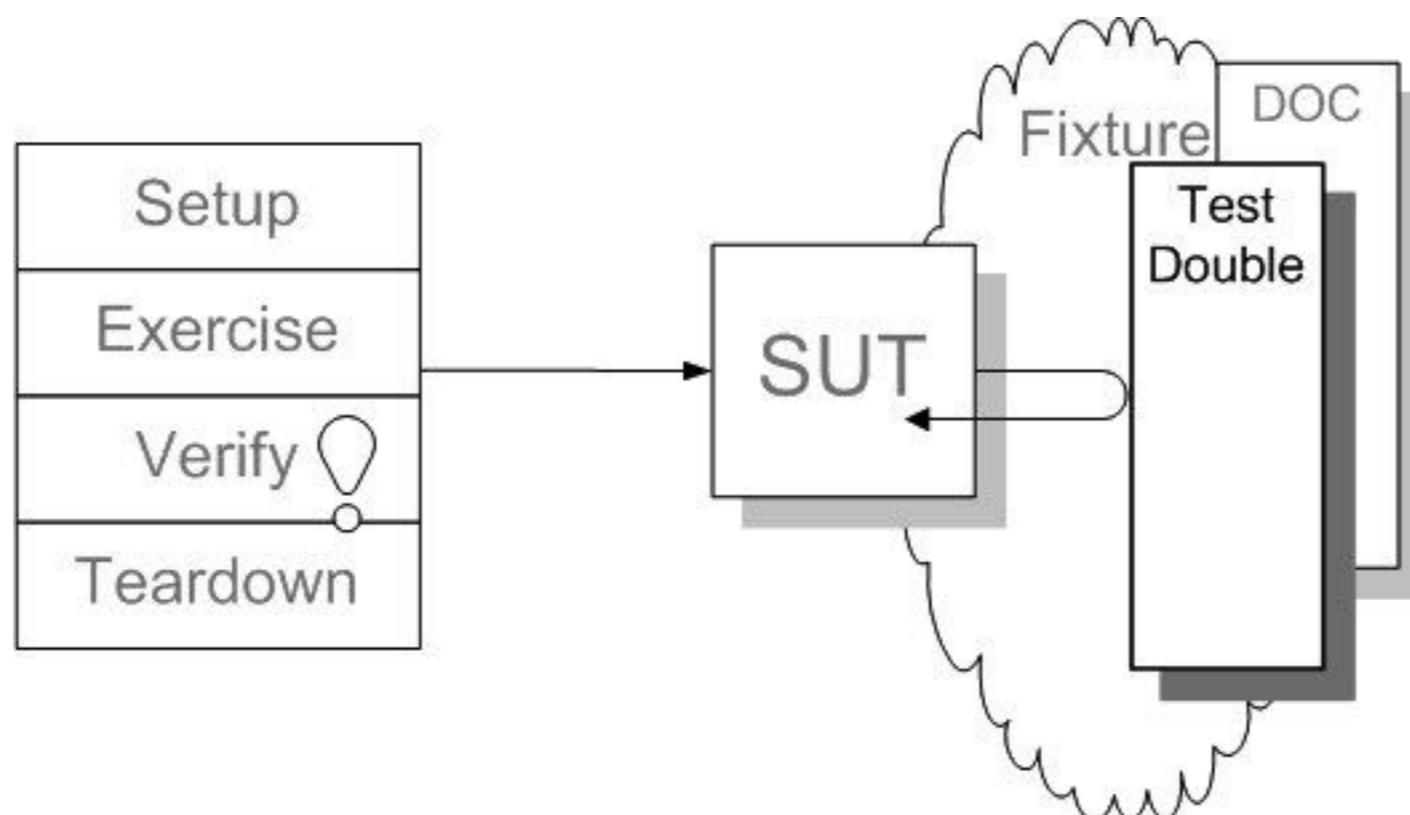
Testing controller with service

Try to mocking service with Mockito



Test Double

How can we verify logic independently ?
How can we avoid Slow tests ?



<http://xunitpatterns.com/Test%20Double.html>



Types of Test Double

Based on **how** and **why** we use it !!

Dummy
object

Test
stub

Test
spy

Mock
object

Fake
object



Run all tests !!

\$mvnw clean test

```
[INFO]
[INFO] Results:
[INFO]
[WARNING] Tests run: 10, Failures: 0, Errors: 0,
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.299 s
[INFO] Finished at: 2018-08-20T23:36:31+07:00
[INFO] -----
```

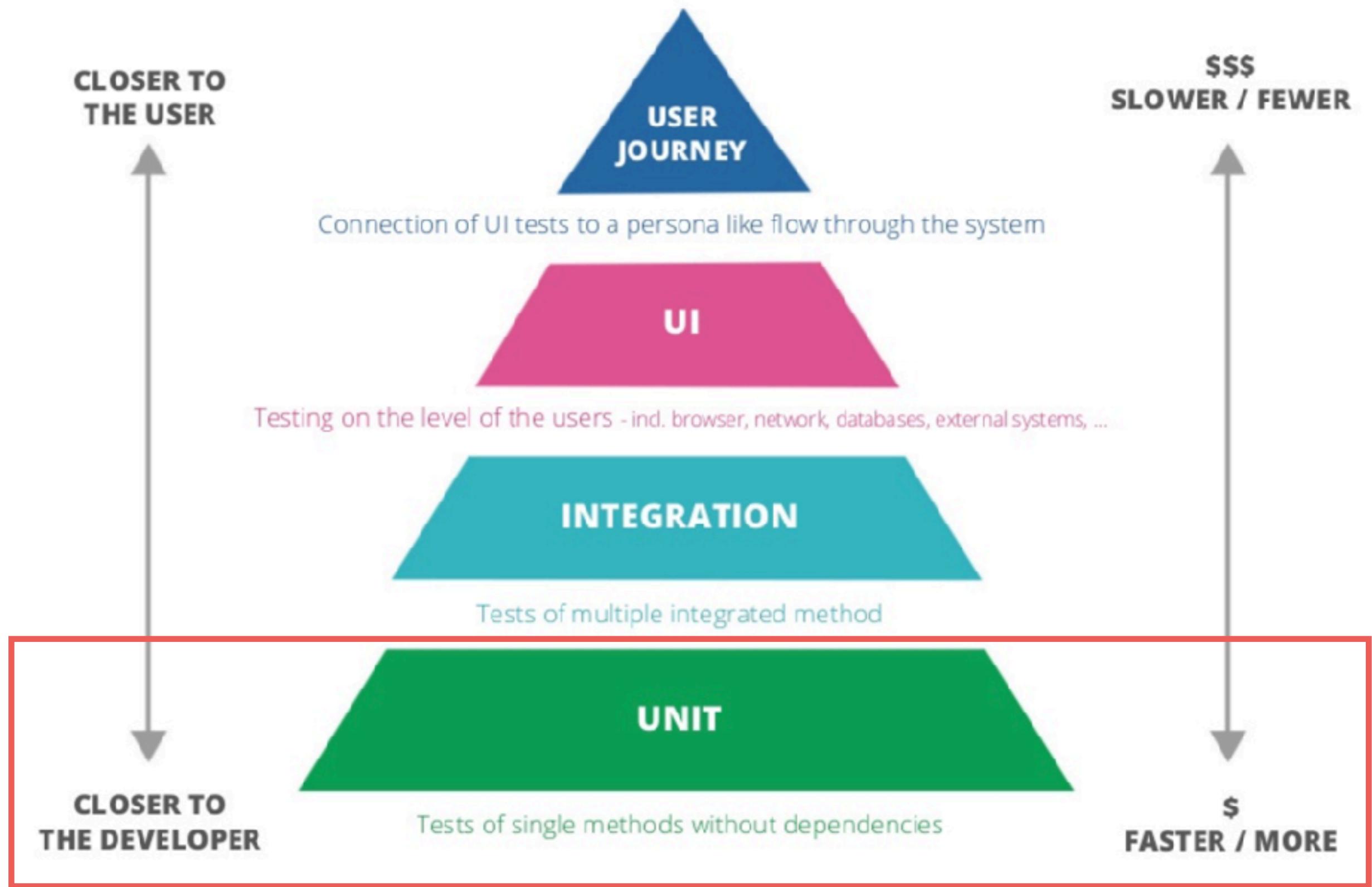


Slow Test ?



How to improve the speed of testing ?





Unit testing with Controller



Unit test

Use Test Double

In java, use Mockito library



<http://site.mockito.org/>



Unit test with Mockito (1)

Setup dependency and controller

```
public class HelloControllerUnitTest {  
  
    private HelloController helloController;  
  
    @Mock  
    private MessageService messageService; ①  
  
    @Before  
    public void initial() {  
        initMocks(this);  
        helloController = new HelloController();  
        helloController.setMessageService(messageService); ②  
    }  
}
```



Unit test with Mockito (2)

Testing by call method from controller

```
@Test  
public void test() {  
    given(messageService.concat("somkiat"))  
        .willReturn("Hello, somkiat");  
  
    HelloResponse response  
        = helloController.sayHi("somkiat");  
  
    assertEquals("Hello, somkiat", response.getMessage());  
}
```



Code coverage



"Code coverage can show the high risk areas in a program, but never the risk-free."

Paul Reilly, 2018, Kotlin TDD with Code Coverage



Code coverage

A tool to measure how much of your code is covered by tests that break down into classes, methods and lines.



Code coverage

But 100% of code coverage **does not mean** that your code is 100% correct



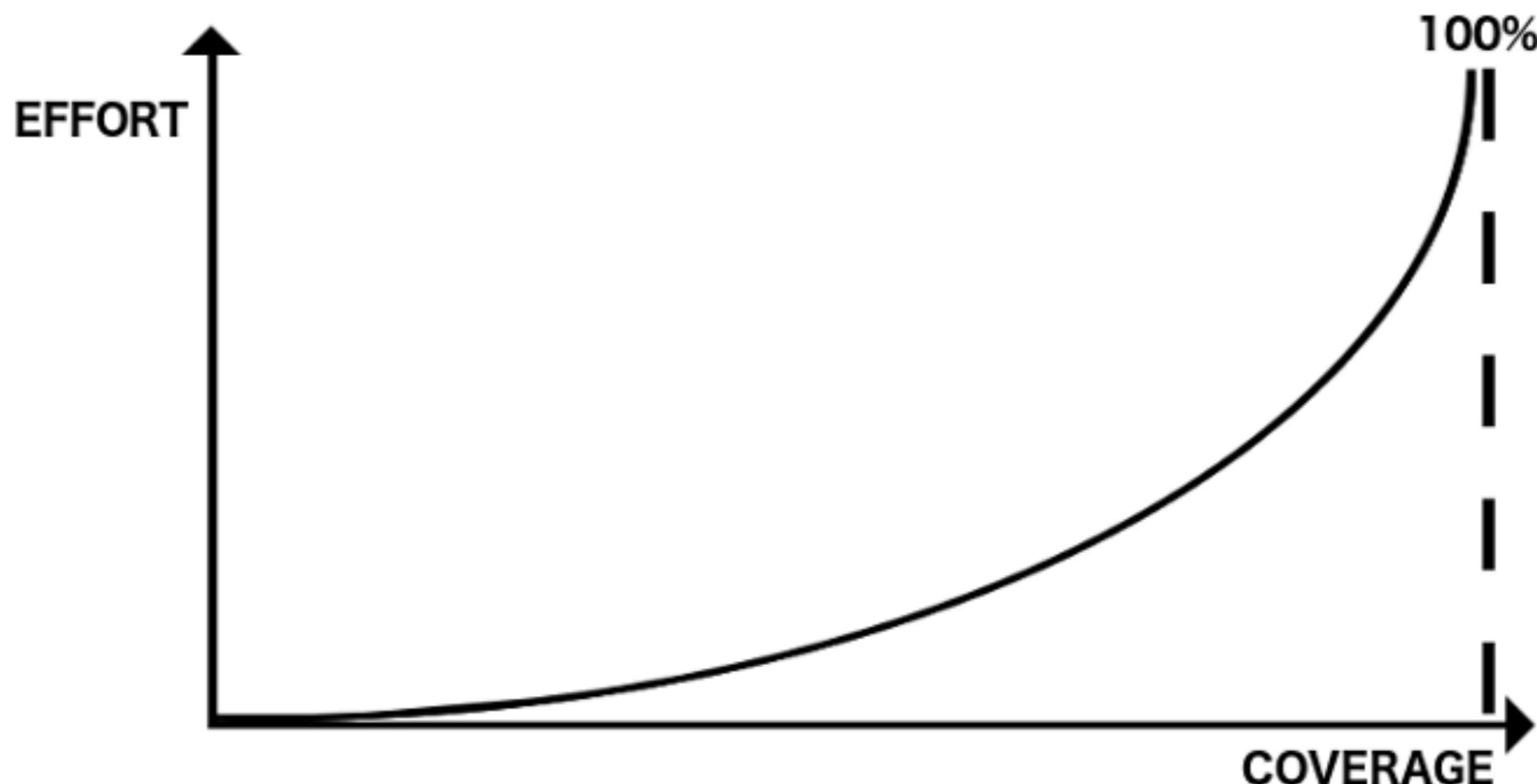
Code coverage

Powerful tool to improve the quality of your code

Code coverage != quality of tests



Code coverage 100% ?



% 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>
    </plugins>
</build>
```



Add coverage to pom.xml (2)

```
<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](#)

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.

All Packages

Classes

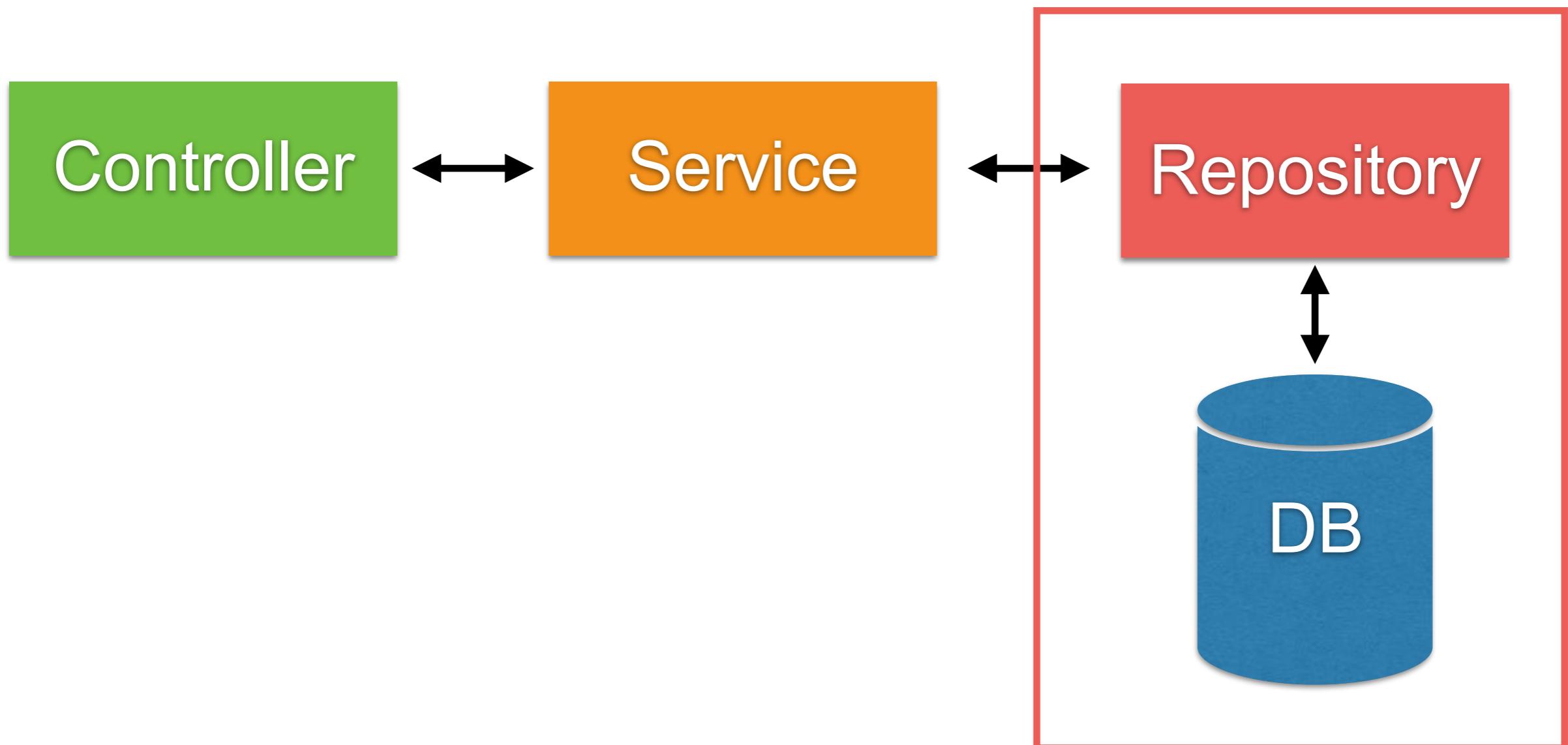
[Hello \(66%\)](#)
[HelloApplication \(33%\)](#)
[HelloController \(100%\)](#)



Working with Repository



Working with repository



Working with repository

We're using Spring Data



<http://projects.spring.io/spring-data/>



Modify pom.xml

Add library of Spring Data such as JPA

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



Modify pom.xml

Add library of data store such as PostgreSQL

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



Add data store config

In src/main/resources/application.properties

```
spring.datasource.url= jdbc:postgresql://127.0.0.1:15432/postgres
spring.datasource.username= user
spring.datasource.password= password
spring.datasource.platform= POSTGRESQL

spring.jpa.show-sql= true
spring.jpa.hibernate.ddl-auto= create-drop
spring.jpa.database-platform= org.hibernate.dialect.PostgreSQLDialect
```



Create repository interface

PersonRepository.java

```
public interface PersonRepository  
    extends CrudRepository<Person, String> {  
  
    Optional<Person> findByFirstName(String name);  
  
}
```



Create Entity class

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



Run your application

```
$java -jar target/hello.jar
```

```
org.postgresql.util.PSQLException: Connection to 127.0.0.1:15432 refused. Check that the postmaster is accepting TCP/IP connections.  
at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:142)  
at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:142)  
at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:142)  
at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:142)  
at org.postgresql.Driver.connect(Driver.java:252) ~[postgresql-42.1.1.jar!/:42.1.1]  
at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:103)  
at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:103)  
at com.zaxxer.hikari.pool.PoolBase.newConnection(PoolBase.java:365) [HikariCP-2.7.0]  
at com.zaxxer.hikari.pool.PoolBase.newPoolEntry(PoolBase.java:194) [HikariCP-2.7.0]  
at com.zaxxer.hikari.pool.HikariPool.createPoolEntry(HikariPool.java:460) [HikariCP-2.7.0]  
at com.zaxxer.hikari.pool.HikariPool.checkFailFast(HikariPool.java:534) [HikariCP-2.7.0]  
at com.zaxxer.hikari.pool.HikariPool.<init>(HikariPool.java:115) [HikariCP-2.7.0]  
at com.zaxxer.hikari.HikariDataSource.getConnection(HikariDataSource.java:112)  
at com.zaxxer.hikari.HikariDataSource.getConnection(HikariDataSource.java:112)
```

Can not connect to database !!!

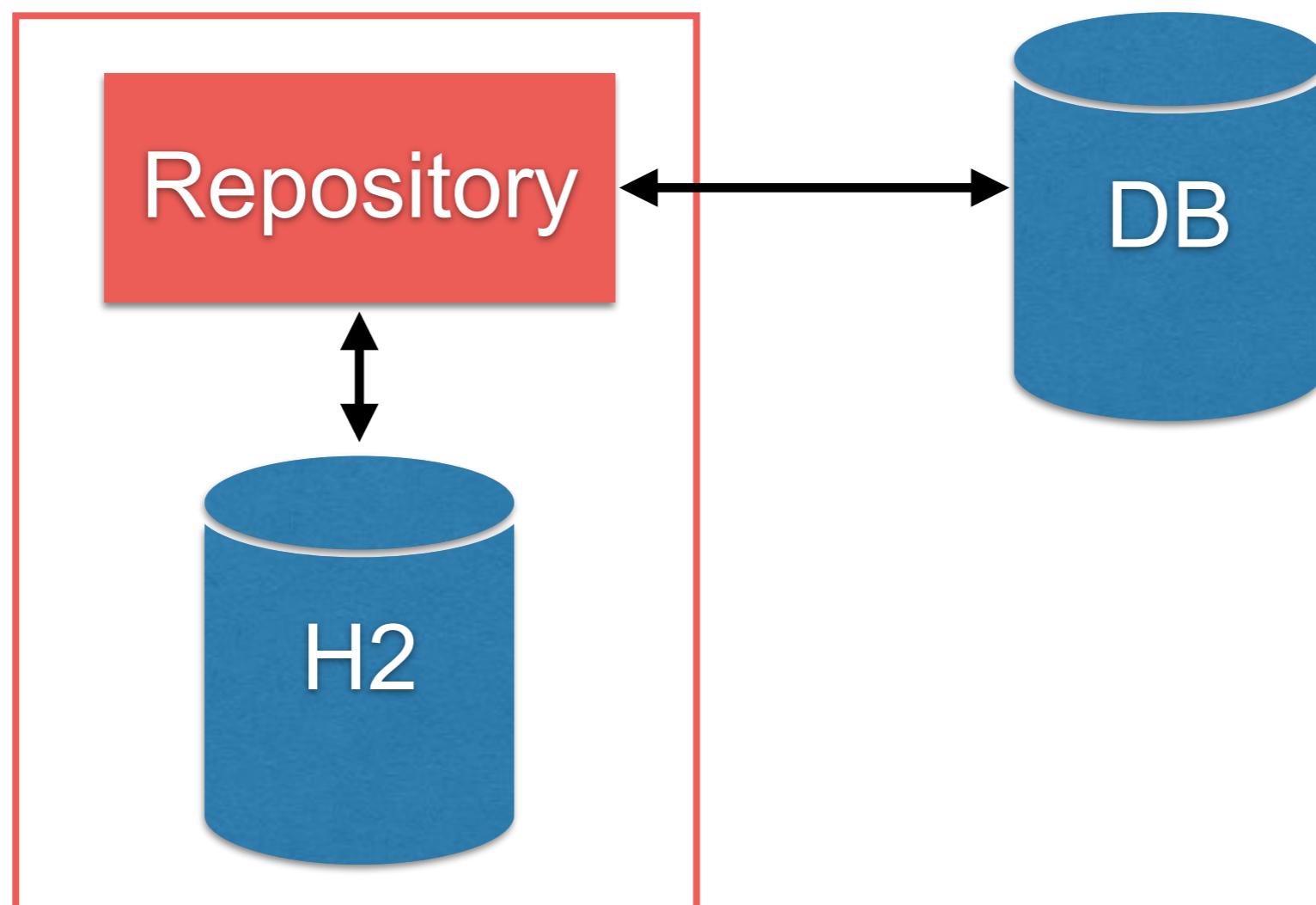


How to testing repository ?



Repository Testing

Using `@DataJpaTest` (slice testing)



Working with In-memory database



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
```

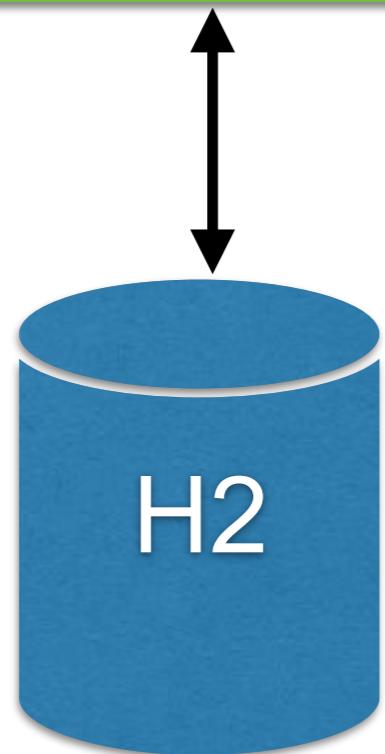


Working with Real Database

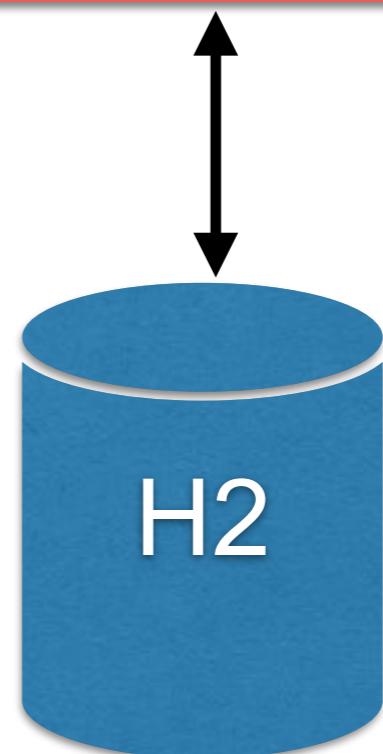


Working with Database

Production



Testing



Initial database #1

Using @PostConstruct

```
@PostConstruct  
public void initData() {  
    Account account1 = new Account();  
    account1.setAccountId("01");  
    accountRepository.save(account1);  
    Account account2 = new Account();  
    account2.setAccountId("02");  
    accountRepository.save(account2);  
}
```



Initial database #2

Schema (resources/schema.sql)

Data (resources/data.sql)

Schema.sql

```
CREATE TABLE account(
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    account_Id VARCHAR(16) NOT NULL UNIQUE,
    mobile_No VARCHAR(10),
    name VARCHAR(50),
    account_Type CHAR(2)
);
```

Data.sql

```
INSERT INTO account (account_Id) VALUES ('01');
INSERT INTO account (account_Id) VALUES ('02');
```



Initial database 2

Disable auto generate DDL from JPA in file application.yml

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none
```



Initial database 2

Problem with naming strategy !!

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none  
      naming:  
        physical-strategy:  
          org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy  
        implicit-strategy:  
          org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
```

<https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/orm/jpa/hibernate>



Run and see from logging

Execute file schema.sql and data.sql

```
.datasource.init.ScriptUtils      : Executing SQL script from URL [file:/Users/somki...  
.datasource.init.ScriptUtils      : Executed SQL script from URL [file:/Users/somki...  
.datasource.init.ScriptUtils      : Executing SQL script from URL [file:/Users/somki...  
.datasource.init.ScriptUtils      : Executed SQL script from URL [file:/Users/somki...
```



Run service

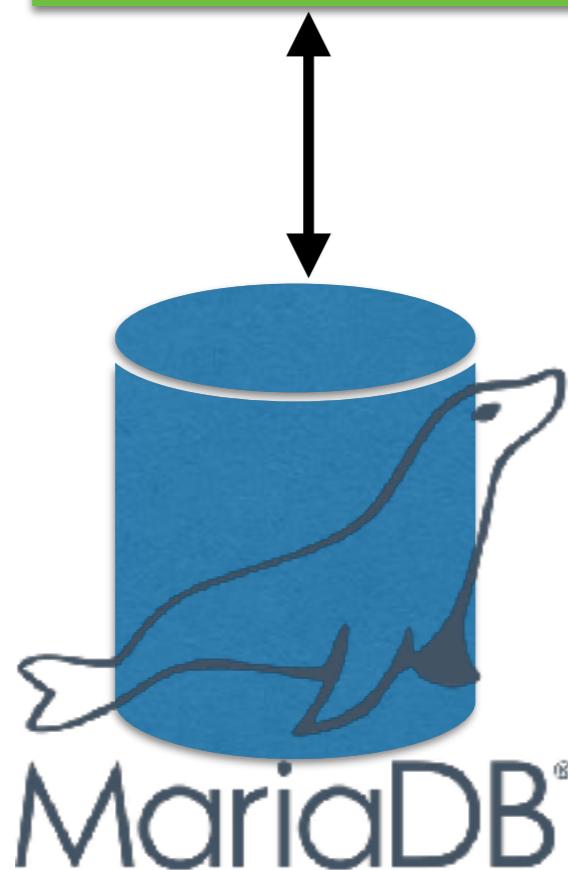
http://localhost:8080/account/0868696209

```
← → ⌂ ⓘ localhost:8080/account/0868696209
[
  - {
    accountNo: "01",
    mobileNo: null,
    name: "",
    accountType: ""
  },
  - {
    accountNo: "02",
    mobileNo: null,
    name: "",
    accountType: ""
  }
]
```

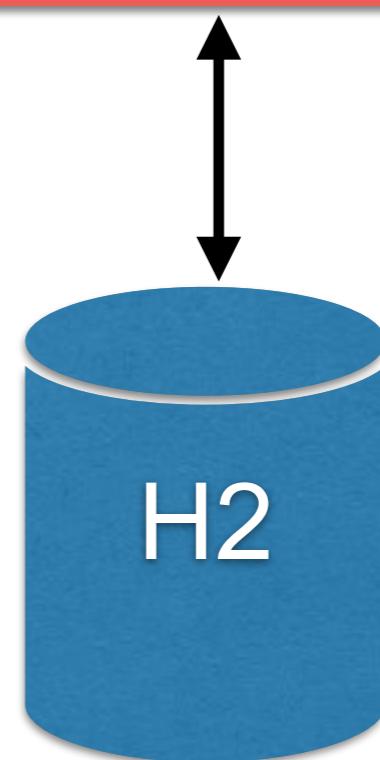


Working with Database

Production



Testing



Add database dependency

In file pom.xml

```
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.2.5</version>
</dependency>
```

<https://mariadb.com/kb/en/library/about-mariadb-connector-j/>



Config database in application

In file resources/application.yml

```
spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://35.198.254.195:3306/account
    username: user01
    password: password

    initialization-mode: always
    testWhileIdle: true
    validationQuery: SELECT 1

  jpa:
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```



Config database in application

Required config for database

```
spring:
```

```
  datasource:  
    driver-class-name: org.mariadb.jdbc.Driver  
    url: jdbc:mariadb://35.198.254.195:3306/account  
    username: user01  
    password: password
```

```
    initialization-mode: always
```

```
    testWhileIdle: true
```

```
    validationQuery: SELECT 1
```

```
jpa:
```

```
  show-sql: true
```

```
  properties:
```

```
    hibernate:
```

```
      dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```



Config database in application

Required config for database

```
spring:  
  datasource:  
    driver-class-name: org.mariadb.jdbc.Driver  
    url: jdbc:mariadb://35.198.254.195:3306/account  
    username: user01  
    password: password
```

```
  initialization-mode: always
```

```
  testWhileIdle: true  
  validationQuery: SELECT 1  
  
jpa:  
  show-sql: true  
  properties:  
    hibernate:  
      dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```

By default not run



Run service

http://localhost:8080/account/0868696209

```
← → ⌂ ⓘ localhost:8080/account/0868696209
[
  - {
    accountNo: "01",
    mobileNo: null,
    name: "",
    accountType: ""
  },
  - {
    accountNo: "02",
    mobileNo: null,
    name: "",
    accountType: ""
  }
]
```



Check your test ?

\$mvnw clean test



Step to test

Stop the real database before run test

See result



Error ?

Spring boot try to connect to the real database ?



Fix Error

Create file /resources/application.yml in test folder

spring: **config of H2 database**

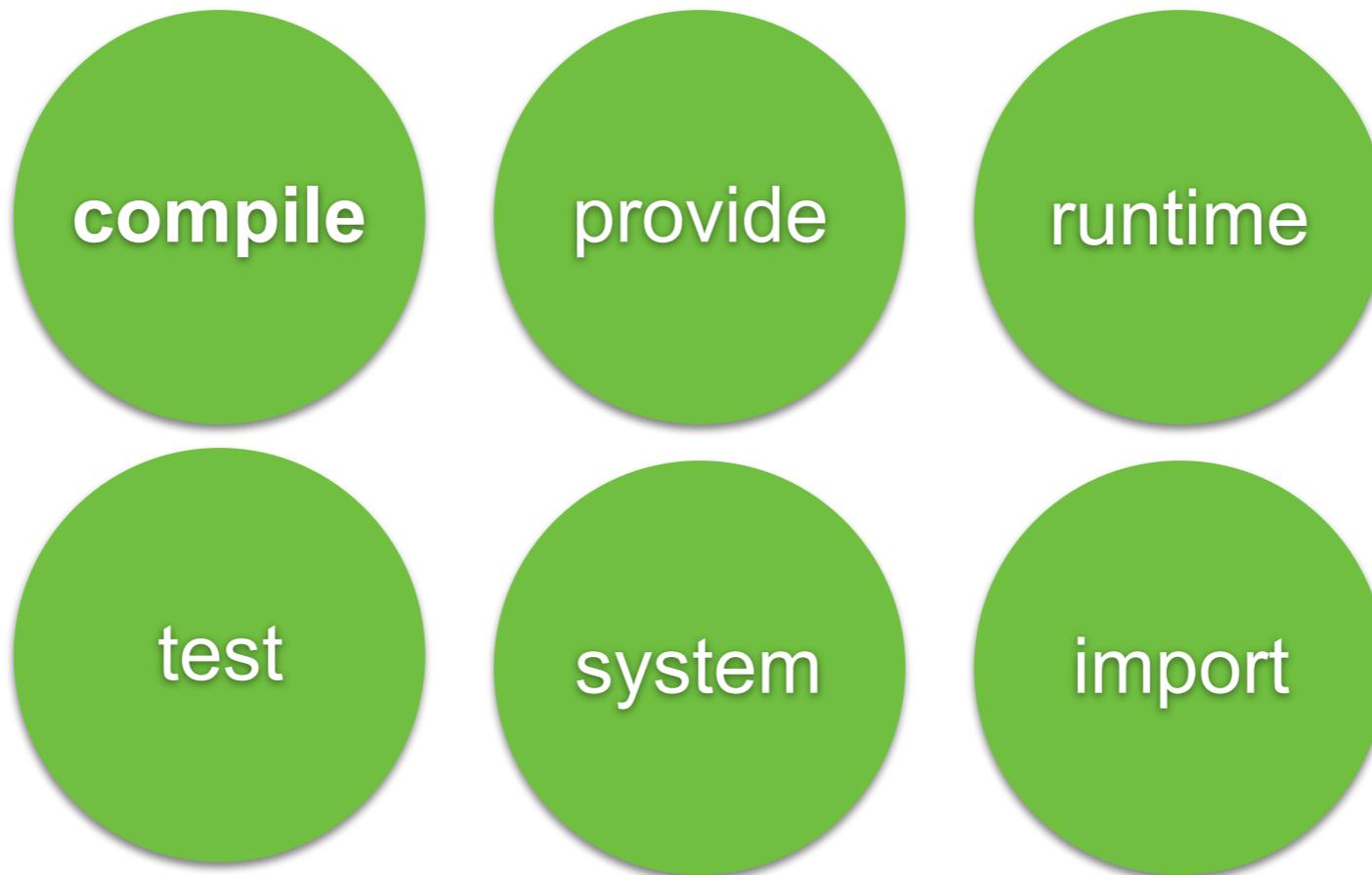
```
datasource:  
  driver-class-name: org.h2.Driver  
  url: jdbc:h2:mem:db;DB_CLOSE_DELAY=-1  
  username: sa  
  password: sa
```

```
jpa:  
  show-sql: true  
  properties:  
    hibernate:  
      dialect: org.hibernate.dialect.H2Dialect
```



Question ?

Dependency Scopes in file pom.xml
runtime ?



https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency_Scope



Question ?

show_sql=true not working ?

```
jpa:  
  show_sql: true  
  hibernate:  
    ddl-auto: none  
properties:  
  hibernate:  
    format_sql: true  
  dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```

```
logging:  
  level:  
    org.hibernate.SQL: DEBUG
```

Beautiful sql format

Default log level = INFO

<http://www.baeldung.com/sql-logging-spring-boot>



Result of logging message

```
13:18:36.336 INFO 79616 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : initialization completed in 37 ms
13:18:36.864 INFO 79616 --- [nio-8080-exec-2] o.h.h.i.QueryTranslatorFactory      : HHH000397: Using ASTQueryTranslatorFactory
13:18:37.138 DEBUG 79616 --- [nio-8080-exec-2] org.hibernate.SQL
select account0_.id as id1_0_, account0_.account_id as account_2_0_, ac
 as account_3_0_, account0_.mobile_no as mobile_n4_0_, account0_.name a
unt account0_
```



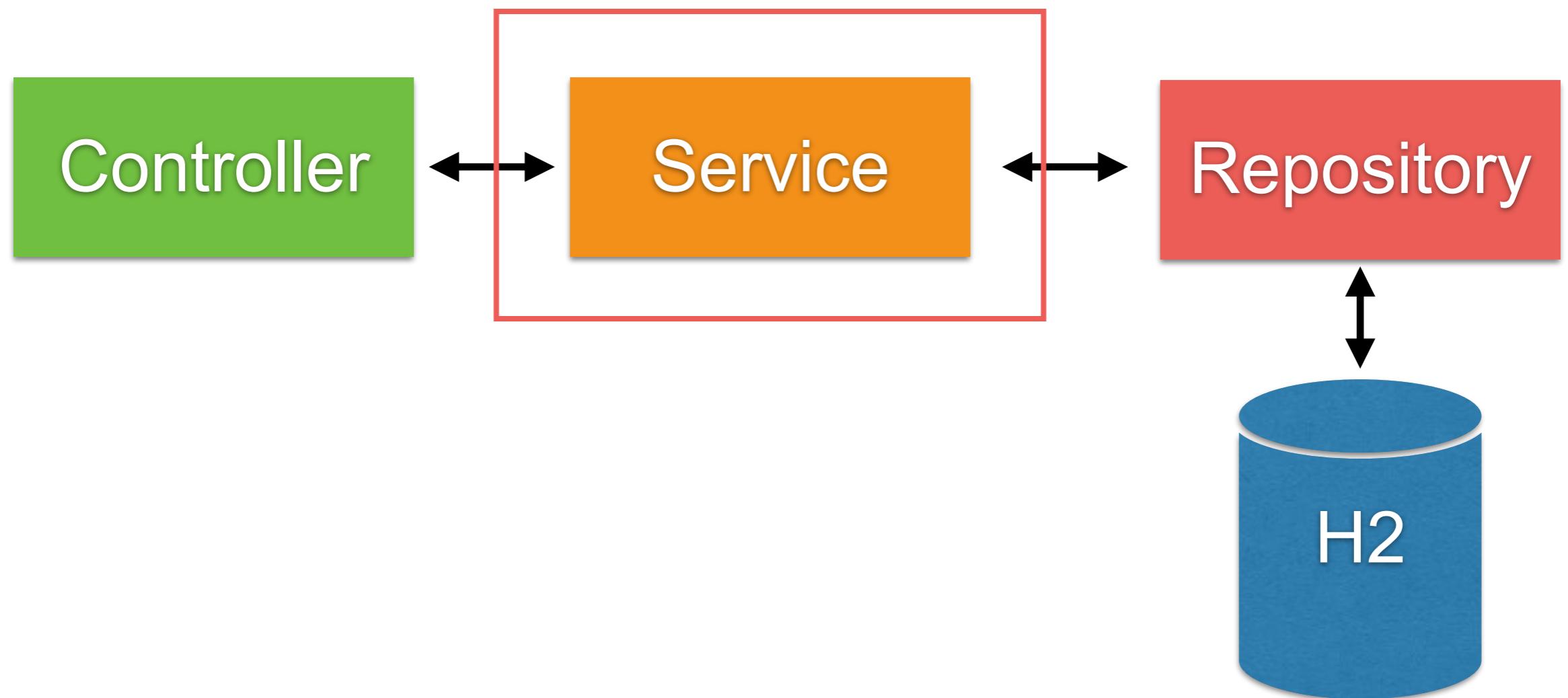
Log Levels

Level	Color
FATAL	Red
ERROR	Red
WARN	Yellow
INFO (default)	Green
DEBUG	Green
TRACE	Green

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-logging.html>



Controller Testing with Unit test



Working with transaction !!

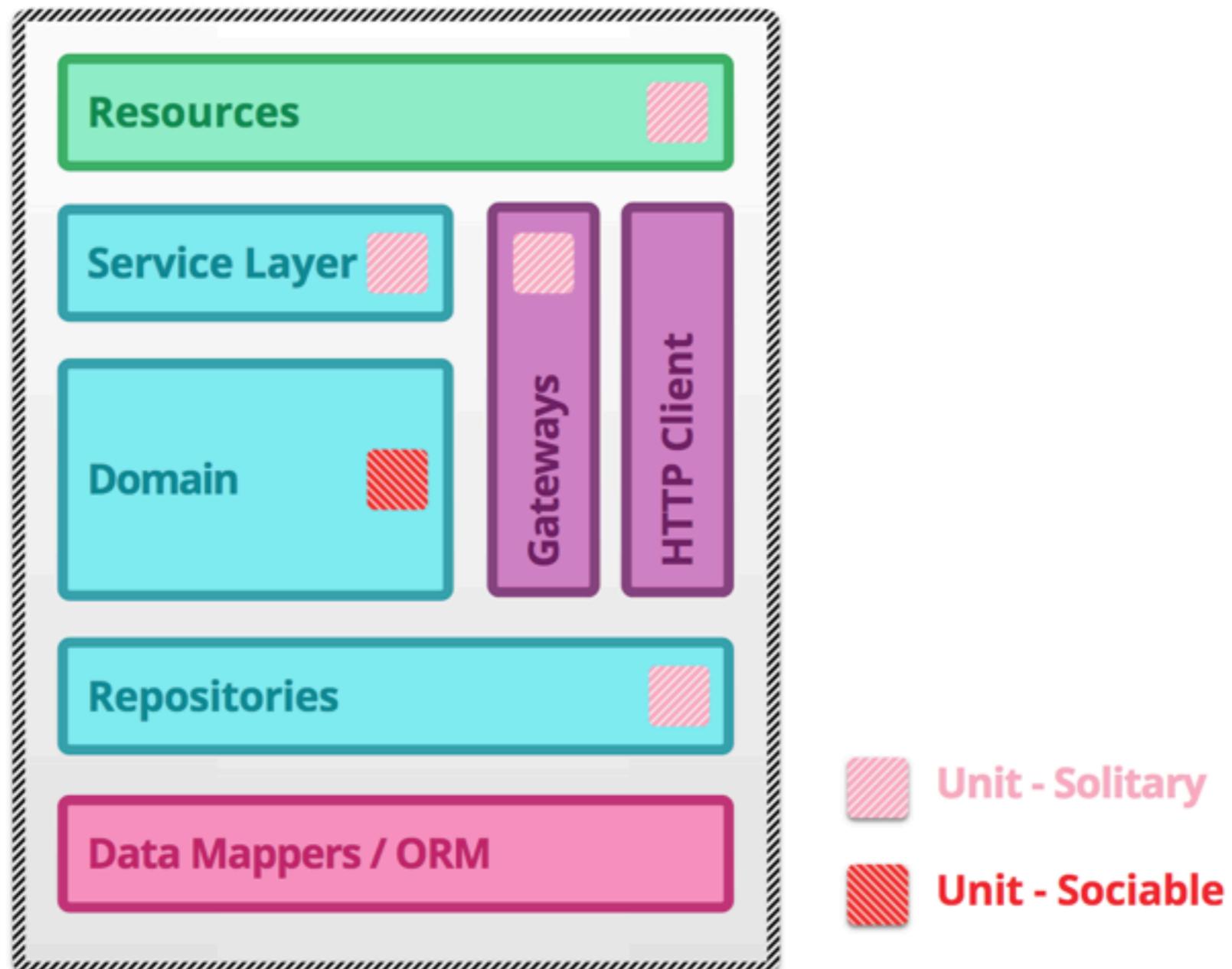


Testing Strategies

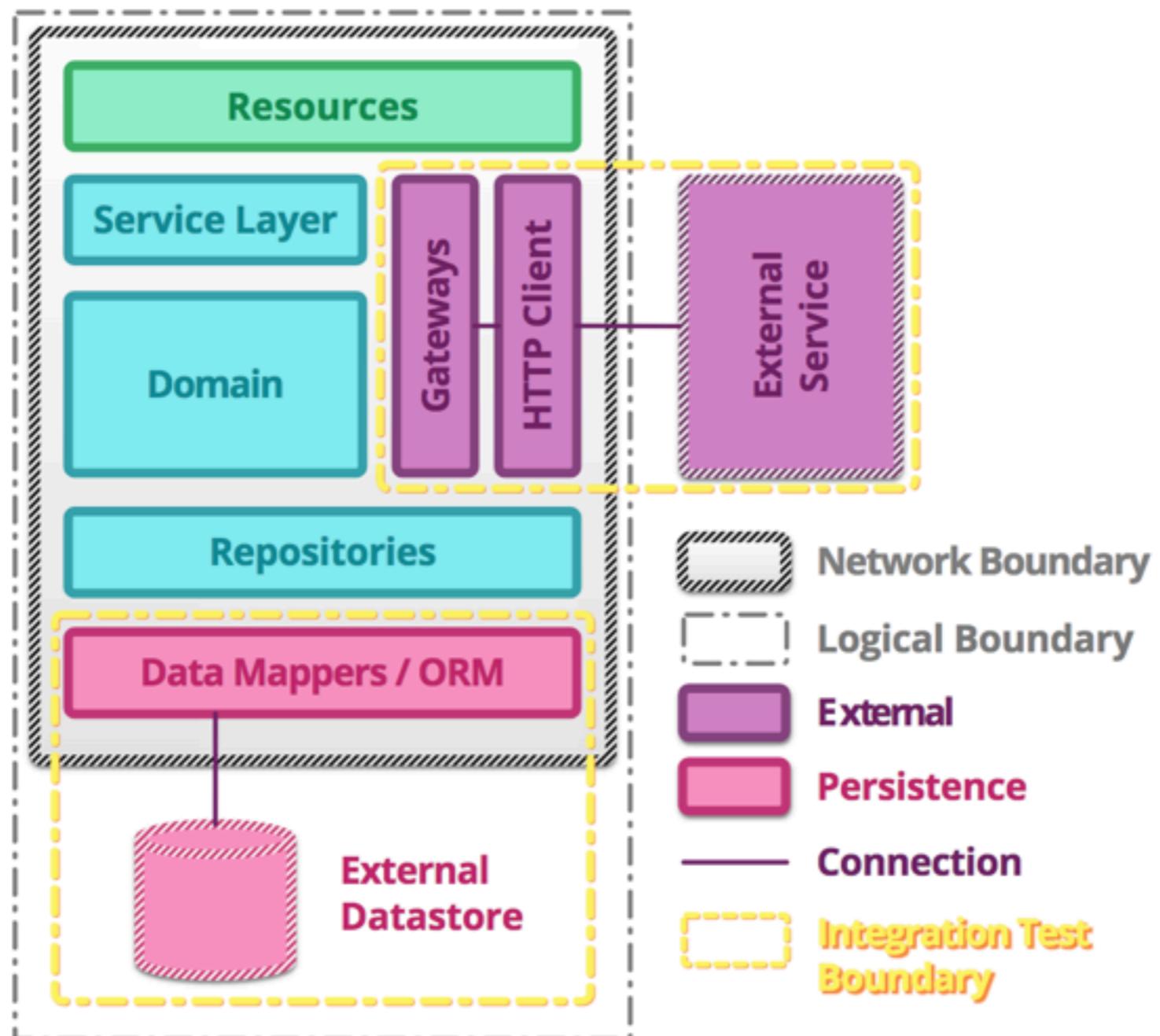




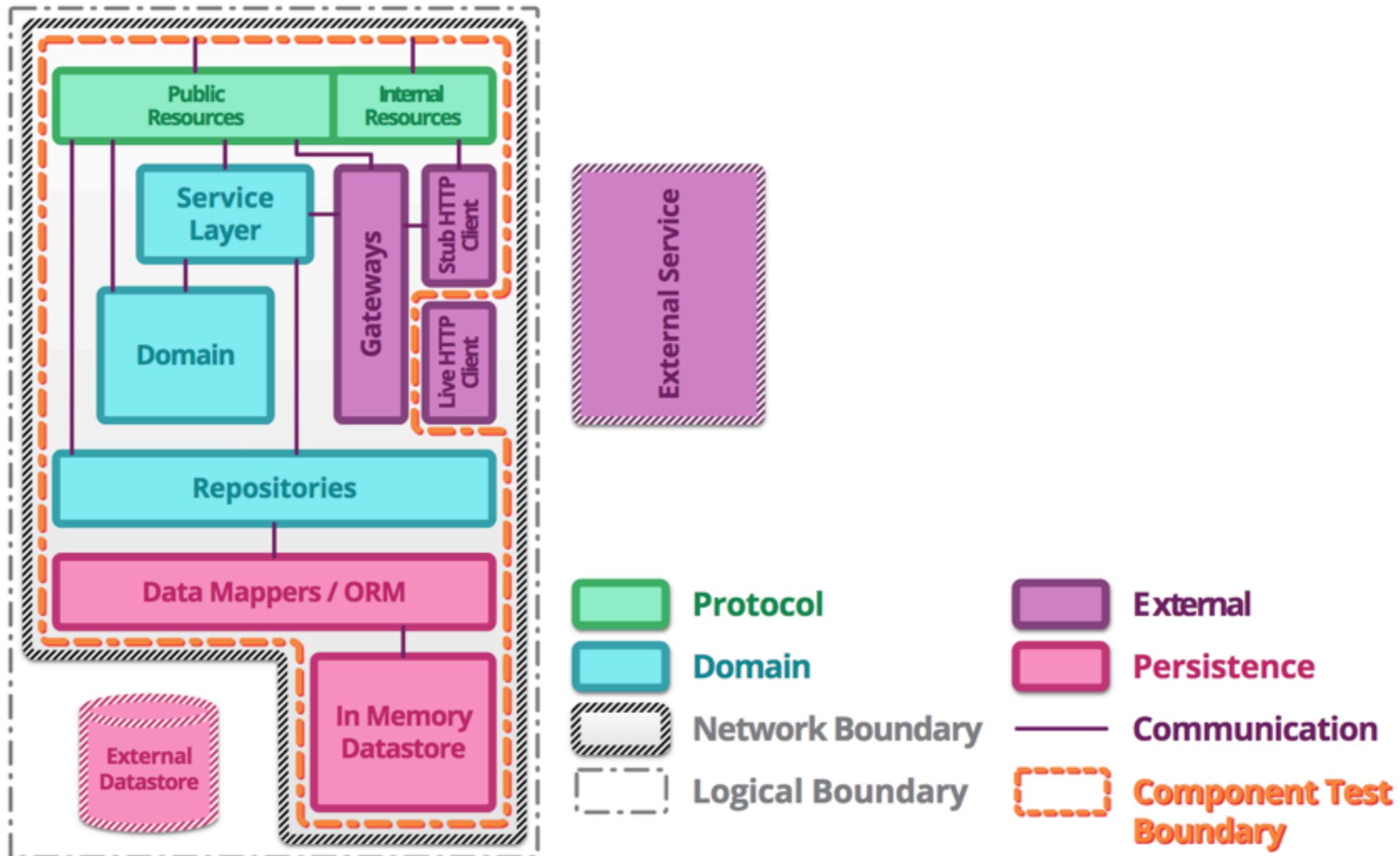
Unit testing



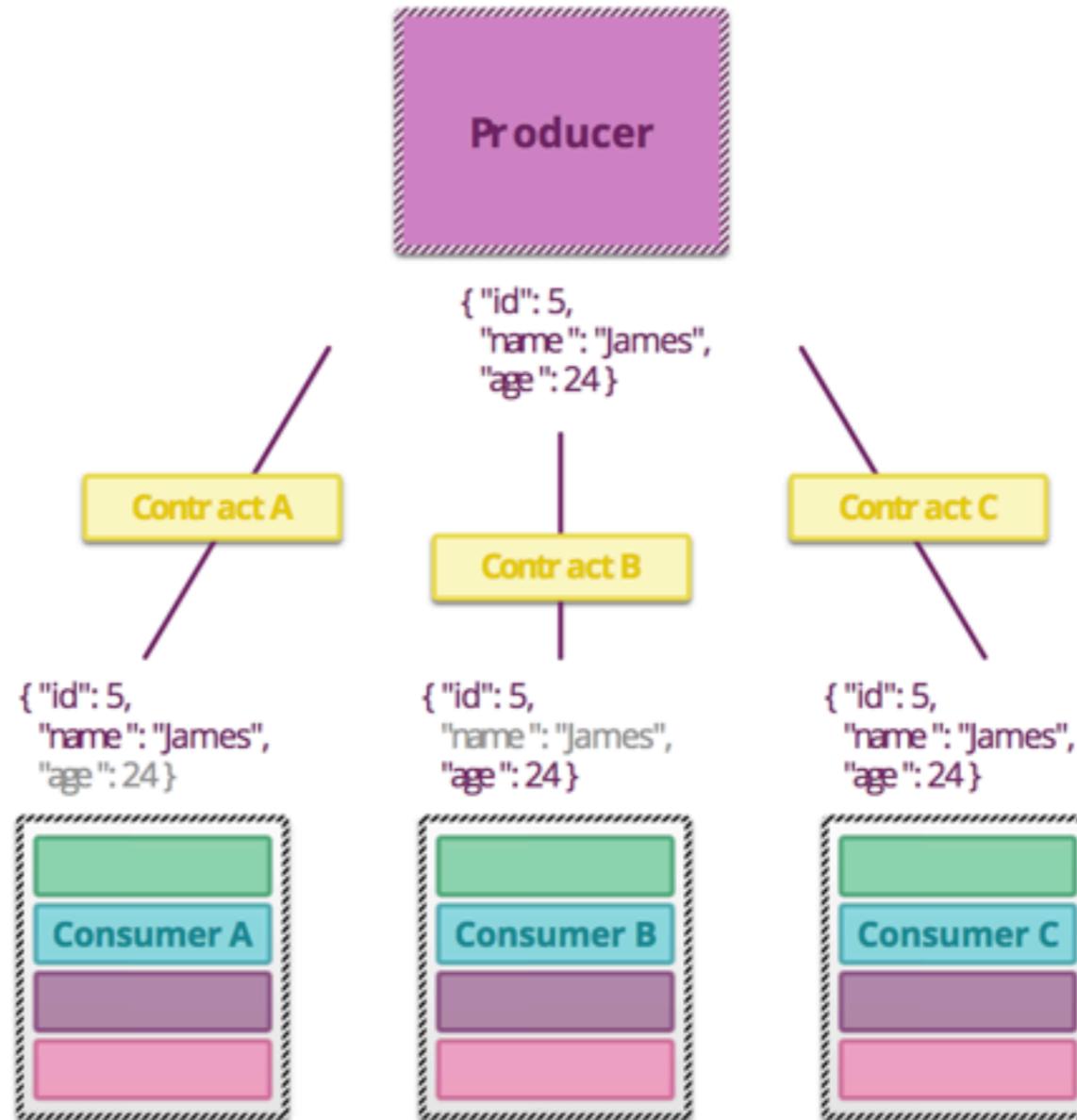
Integration testing



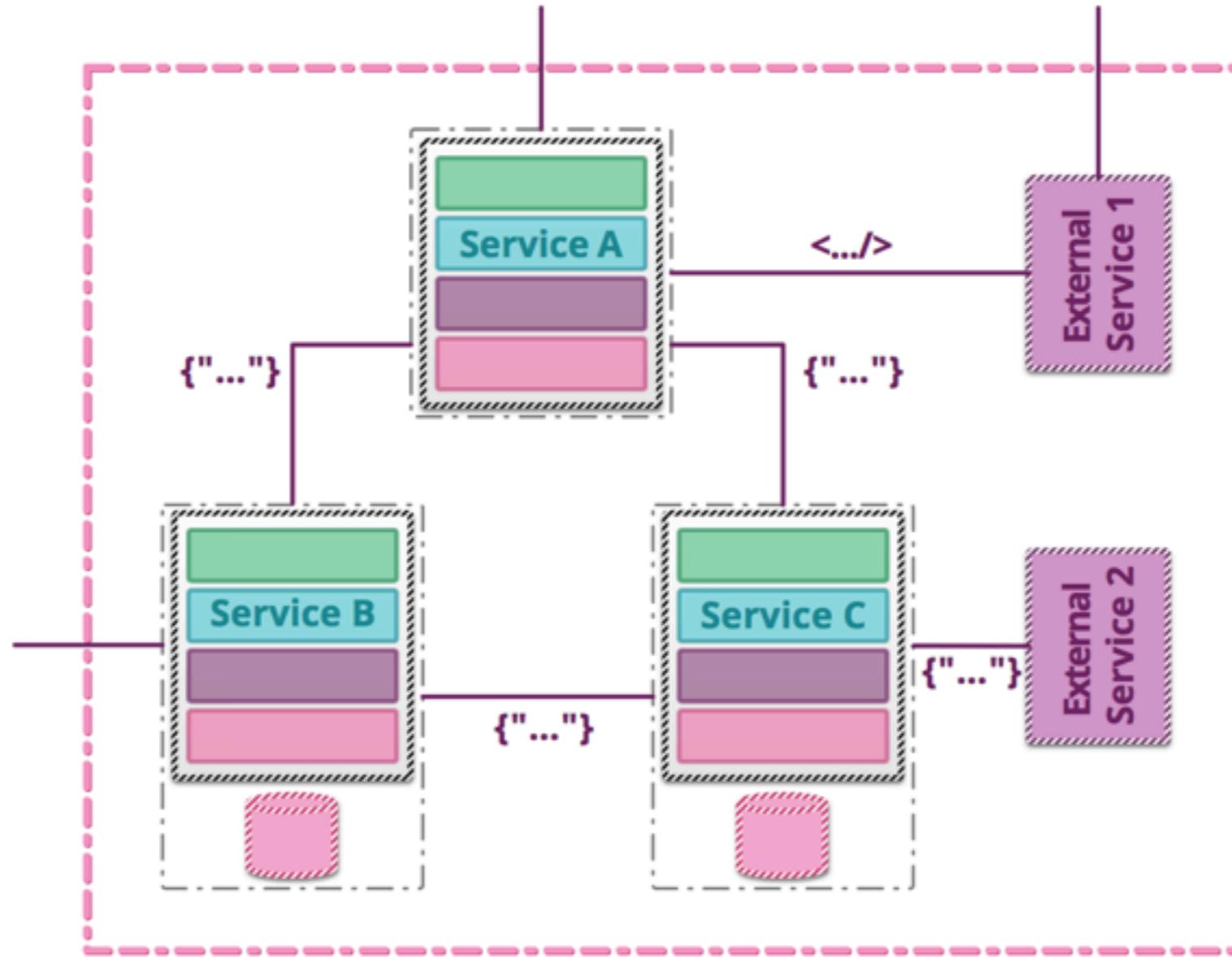
Component testing



Contract testing



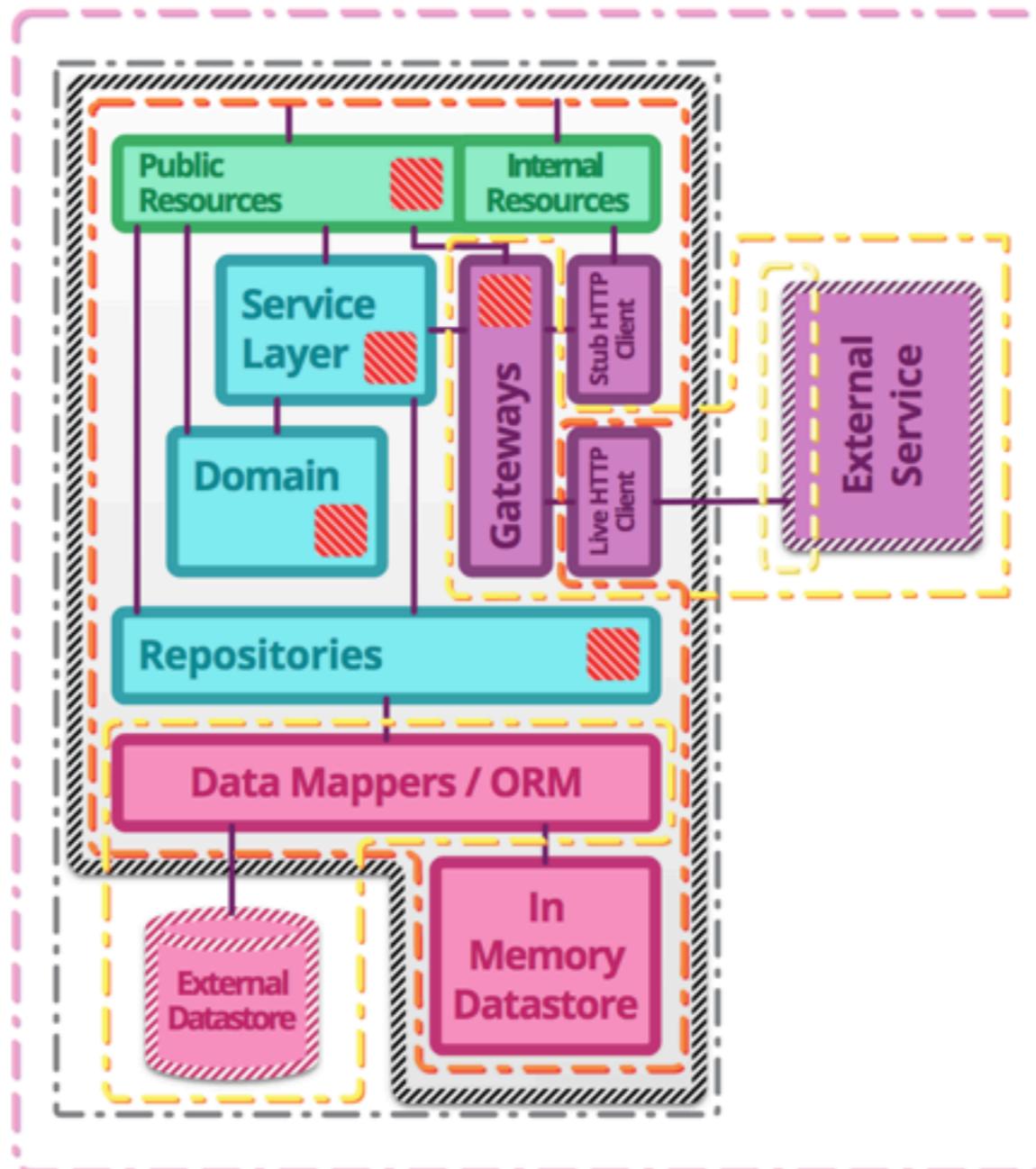
End-to-End testing



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.

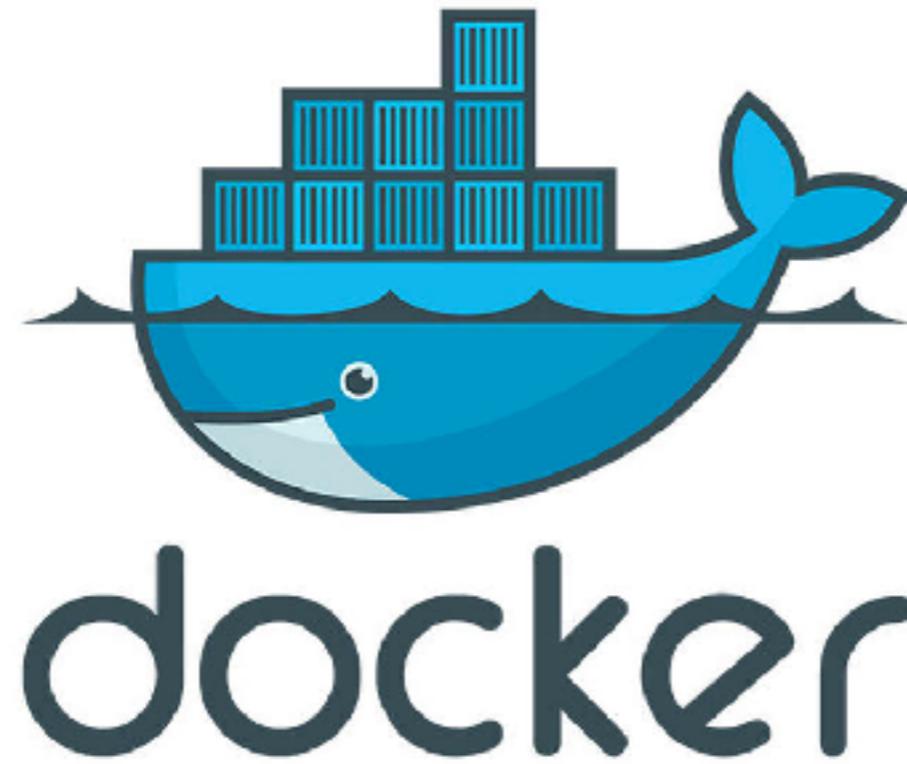


 **Component tests** : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

 **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.



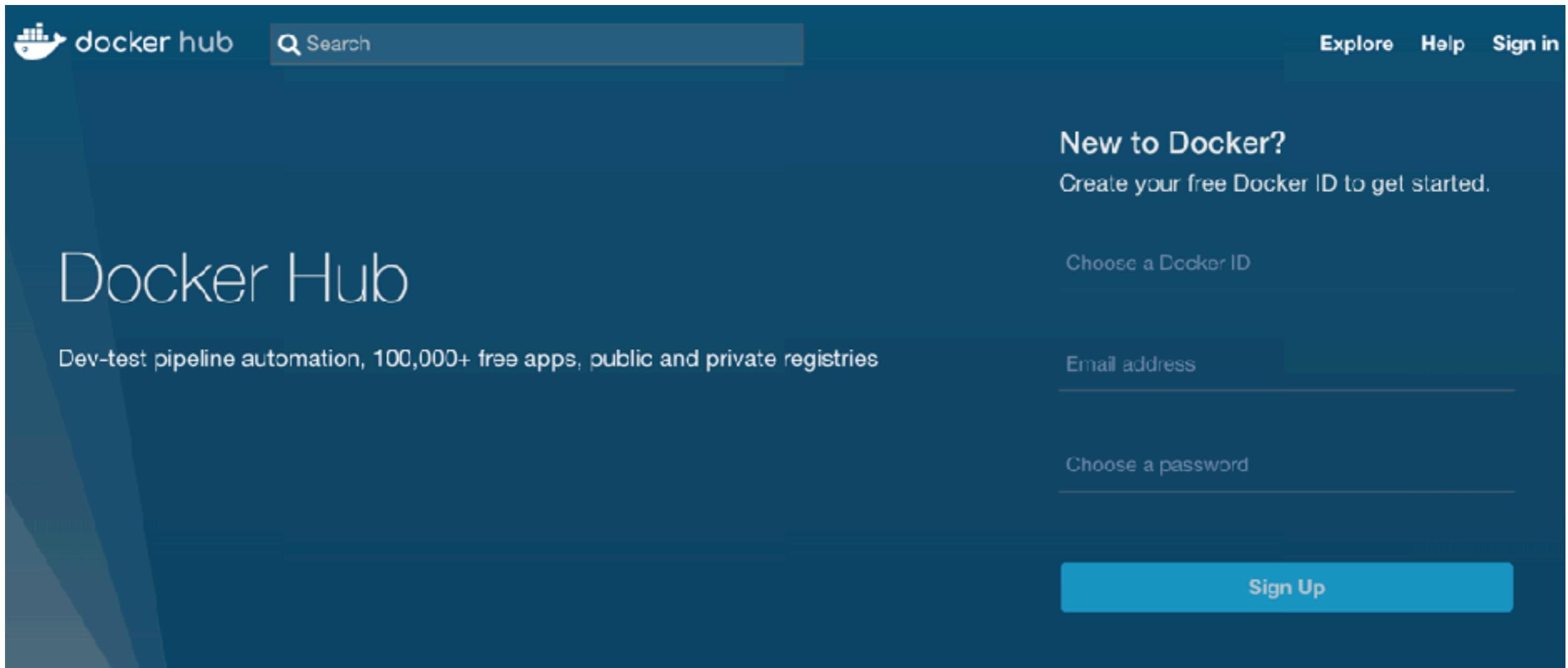


Working with container



Choose your image

from hub.docker.com



The screenshot shows the Docker Hub sign-up page. At the top left is the Docker Hub logo and a search bar with the placeholder "Search". To the right are links for "Explore", "Help", and "Sign in". The main heading "Docker Hub" is on the left, followed by a subtext about pipeline automation and registries. On the right, there's a "New to Docker?" section with a "Create your free Docker ID to get started." link, a "Choose a Docker ID" input field, an "Email address" input field, a "Choose a password" input field, and a prominent blue "Sign Up" button.



Images in workshop

Java => OpenJDK

Apache Maven => maven

PostgreSQL => progrès



Java => OpenJDK

OFFICIAL REPOSITORY

openjdk 

Last pushed: 5 days ago

[Repo Info](#) [Tags](#)

Short Description

OpenJDK is an open-source implementation of the Java Platform, Standard Edition

Docker Pull Command

`docker pull openjdk`

Full Description

Supported tags and respective [Dockerfile](#) [links](#)

Simple Tags

- `10-ea-32-jre-experimental`, `10-ea-jre-experimental`, `10-jre-`

https://hub.docker.com/_/openjdk/



Apache Maven => maven

OFFICIAL REPOSITORY

maven 

Last pushed: 5 days ago

Repo Info Tags

Short Description

Apache Maven is a software project management and comprehension tool.

Docker Pull Command

`docker pull maven`

Full Description

Supported tags and respective [Dockerfile](#) links

- `3.5.2-jdk-7-alpine` ([jdk-7-alpine/Dockerfile](#))
- `3.5.2-jdk-7-slim` ([jdk-7-slim/Dockerfile](#))
- `3.5.2-slim` ([slim/Dockerfile](#))

https://hub.docker.com/_/maven/



PostgreSQL => progrès

OFFICIAL REPOSITORY

postgres 

Last pushed: 3 days ago

Repo Info [Tags](#)

Short Description

The PostgreSQL object-relational database system provides reliability and data integrity.

Docker Pull Command



```
docker pull postgres
```

Full Description

Supported tags and respective [Dockerfile](#) links

- [10.3](#), [10](#), [latest](#) ([10/Dockerfile](#))
[10.3](#) → [10.3/Dockerfile](#) → [10/Dockerfile](#) → [latest/Dockerfile](#)

https://hub.docker.com/_/postgres/



Pull images from Docker Hub

\$docker image pull openjdk:<tag>

\$docker image pull maven:<tag>

\$docker image pull postgres:<tag>



Create container to run Spring Boot



Create container to run Spring Boot

```
$docker container run -d  
-v $(pwd)/target/toystore.jar:/xxx/toystore.jar  
-p 8080:8080  
--name web  
openjdk:8-jre java -jar /xxx/toystore.jar
```



Create new Docker image (1)

from Dockerfile

```
FROM openjdk:8-jre  
COPY ./target/toystore.jar /xxx/toystore.jar  
CMD java -jar /xxx/toystore.jar
```



Create new Docker image (2)

Build image from Dockerfile

```
$ docker image build -t toystore:0.1 .
```

```
Sending build context to Docker daemon 33.47MB
```

```
Step 1/3 : FROM openjdk:8-jre
```

```
---> e956268fd4ed
```

```
Step 2/3 : COPY ./target/toystore.jar /xxx/toystore.jar
```

```
---> 3dd837b158eb
```

```
Step 3/3 : CMD java -jar /xxx/toystore.jar
```

```
---> Running in 06994b290e74
```

```
Removing intermediate container 06994b290e74
```

```
---> 0bc2054f4ba8
```

```
Successfully built 0bc2054f4ba8
```

```
Successfully tagged toystore:0.1
```



Create container to run Spring Boot

```
$ docker container run -d -p 8080:8080 toystore:0.1
```

```
$ docker container run -d -p 8081:8080 toystore:0.1
```

```
$ docker container run -d -p 8082:8080 toystore:0.1
```



Create container to run Build Maven Project



Create container to build

```
$docker container run --rm  
-v $(pwd):/xxx  
-w /xxx  
maven:3.5.2-alpine mvn clean package
```



Run your application

\$java -jar target/hello.jar

```
org.postgresql.util.PSQLException: Connection to 127.0.0.1:15432 refused. Check that the postmaster is accepting TCP/IP connections.
    at org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.java:421)
    at org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java:84)
    at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:194) ~[postgresql-42.1.1.jar!/:42.1.1]
    at org.postgresql.Driver.makeConnection(Driver.java:450) ~[postgresql-42.1.1.jar!/:42.1.1]
    at org.postgresql.Driver.connect(Driver.java:252) ~[postgresql-42.1.1.jar!/:42.1.1]
    at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:104)
    at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:92)
    at com.zaxxer.hikari.pool.PoolBase.newConnection(PoolBase.java:365) [HikariCP-2.7.0.jar!/:2.7.0]
    at com.zaxxer.hikari.pool.PoolBase.newPoolEntry(PoolBase.java:194) [HikariCP-2.7.0.jar!/:2.7.0]
    at com.zaxxer.hikari.pool.HikariPool.createPoolEntry(HikariPool.java:460) [HikariCP-2.7.0.jar!/:2.7.0]
    at com.zaxxer.hikari.pool.HikariPool.checkFailFast(HikariPool.java:534) [HikariCP-2.7.0.jar!/:2.7.0]
    at com.zaxxer.hikari.pool.HikariPool.<init>(HikariPool.java:115) [HikariCP-2.7.0.jar!/:2.7.0]
    at com.zaxxer.hikari.HikariDataSource.getConnection(HikariDataSource.java:112)
    at sun.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.springframework.jdbc.datasource.DataSourceUtils.fetchConnection(DataSourceUtils.java:119)
    at org.springframework.jdbc.datasource.DataSourceUtils.doGetConnection(DataSourceUtils.java:80)
    at org.springframework.jdbc.datasource.DataSourceUtils.getConnection(DataSourceUtils.java:63)
    at org.springframework.jdbc.core.JdbcTemplate.init(JdbcTemplate.java:334)
    at org.springframework.jdbc.core.JdbcTemplate.<init>(JdbcTemplate.java:150)
    at org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate.<init>(NamedParameterJdbcTemplate.java:77)
    at org.springframework.boot.autoconfigure.jdbc.DataSourceProperties.createNamedParameterJdbcTemplate(DataSourceProperties.java:160)
    at org.springframework.boot.autoconfigure.jdbc.DataSourceProperties.createDataSource(DataSourceProperties.java:134)
    at org.springframework.boot.autoconfigure.jdbc.DataSourceProperties.createDataSource(DataSourceProperties.java:116)
    at org.springframework.boot.autoconfigure.jdbc.DataSourceConfiguration.createDataSource(DataSourceConfiguration.java:44)
    at org.springframework.boot.autoconfigure.jdbc.DataSourceConfiguration.createDataSource(DataSourceConfiguration.java:37)
    at org.springframework.context.annotation.ConfigurationClassEnhancer$BeanMethodInterceptor.intercept(ConfigurationClassEnhancer.java:361)
    at org.springframework.boot.autoconfigure.jdbc.DataSourceConfiguration$$EnhancerBySpringCGLIB$$d9a2a2f1.createDataSource(+ 13 unexpended stack frames from org.springframework.boot.autoconfigure.jdbc.DataSourceConfiguration$$EnhancerBySpringCGLIB$$d9a2a2f1)
```



Create container of PostgreSQL



Create container of PostgreSQL

```
$ docker container run  
-p 15432:5432  
-e POSTGRES_USER=user  
-e POSTGRES_PASSWORD=password  
postgres:10.3-alpine
```



Run your application again !!

\$java -jar target/hello.jar

```
Hibernate: create sequence hibernate_sequence start 1 increment 1
Hibernate: create table person (id int4 not null, first_name varchar(255), last_name varchar(255), primary key (id))
2018-03-06 19:45:41.115  INFO 56495 --- [           main] o.h.t.schema.internal.SchemaCreatorImpl : HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@7e998ed7'
2018-03-06 19:45:41.121  INFO 56495 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2018-03-06 19:45:42.592  INFO 56495 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@7cef4e59: startup date [Tue Mar 06 19:45:34 ICT 2018]; root of context hierarchy
2018-03-06 19:45:42.716  WARN 56495 --- [           main] aWebConfigurationJpaWebMvcConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2018-03-06 19:45:42.857  INFO 56495 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/hello/{name}],methods=[GET]}" onto public toystore.domain.Hello toystore.controller.HelloController.sayHi(java.lang.String)
2018-03-06 19:45:42.881  INFO 56495 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/hello/data/{name}],methods=[GET]}" onto public toystore.domain.Hello toystore.controller.HelloWithRepositoryController.sayHi(java.lang.String)
2018-03-06 19:45:42.889  INFO 56495 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
2018-03-06 19:45:42.893  INFO 56495 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2018-03-06 19:45:43.046  INFO 56495 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-06 19:45:43.046  INFO 56495 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-06 19:45:43.168  INFO 56495 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-06 19:45:43.783  INFO 56495 --- [           main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
2018-03-06 19:45:43.785  INFO 56495 --- [           main] o.s.j.e.a.AnnotationMBeanExporter : Bean with name 'dataSource' has been autodetected for JMX exposure
2018-03-06 19:45:43.793  INFO 56495 --- [           main] o.s.j.e.a.AnnotationMBeanExporter : Located MBean 'dataSource': registering with JMX server as MBean [com.zaxxer.hikari:name=dataSource,type=HikariDataSource]
2018-03-06 19:45:43.873  INFO 56495 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2018-03-06 19:45:43.877  INFO 56495 --- [           main] toystore.ToyStoreApplication : Started ToyStoreApplication in 10.18 seconds (JVM running for 10.912)
2018-03-06 19:45:47.875  INFO 56495 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring FrameworkServlet 'dispatcherServlet'
2018-03-06 19:45:47.876  INFO 56495 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization started
2018-03-06 19:45:47.917  INFO 56495 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization completed in 41 ms
```



Monitoring and Metric



Metric in Spring Boot

Spring Boot Actuator for Spring Boot 1.x
MicroMeter for Spring Boot 2.0



Spring Boot Actuator (1)

Add library to pom.xml

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



Spring Boot Actuator (2)

Enabled endpoint in application.properties

```
info.app.name=Toy Store
info.app.description=This is my first spring boot application
info.app.version=1.0.0

management.endpoints.web.exposure.include=health,info,metrics,httptrace
```



Spring Boot Actuator (3)

List of endpoints = /actuator/

```
← → ⌂ ⓘ localhost:8080/actuator/  
  
{  
  - _links: {  
    - self: {  
      href: "http://localhost:8080/actuator",  
      templated: false  
    },  
    - health: {  
      href: "http://localhost:8080/actuator/health",  
      templated: false  
    },  
    - info: {  
      href: "http://localhost:8080/actuator/info",  
      templated: false  
    },  
    - metrics-requiredMetricName: {  
      href: "http://localhost:8080/actuator/metrics/{requiredMetricName}",  
      templated: true  
    },  
    - metrics: {  
      href: "http://localhost:8080/actuator/metrics",  
      templated: false  
    },  
    - httptrace: {  
      href: "http://localhost:8080/actuator/httptrace",  
      templated: false  
    }  
  }  
}
```



Spring Boot Actuator (4)

Info endpoint = /actuator/info

```
← → ⌂ ⓘ localhost:8080/actuator/info

{
  - app: {
      name: "Toy Store",
      description: "This is my first spring boot application",
      version: "1.0.0"
    }
}
```



Spring Boot Actuator (5)

Info endpoint = /actuator/info

```
← → ⌂ ⓘ localhost:8080/actuator/info

{
  - app: {
      name: "Toy Store",
      description: "This is my first spring boot application",
      version: "1.0.0"
    }
}
```



Spring Boot Actuator (6)

Info endpoint = /actuator/httptrace

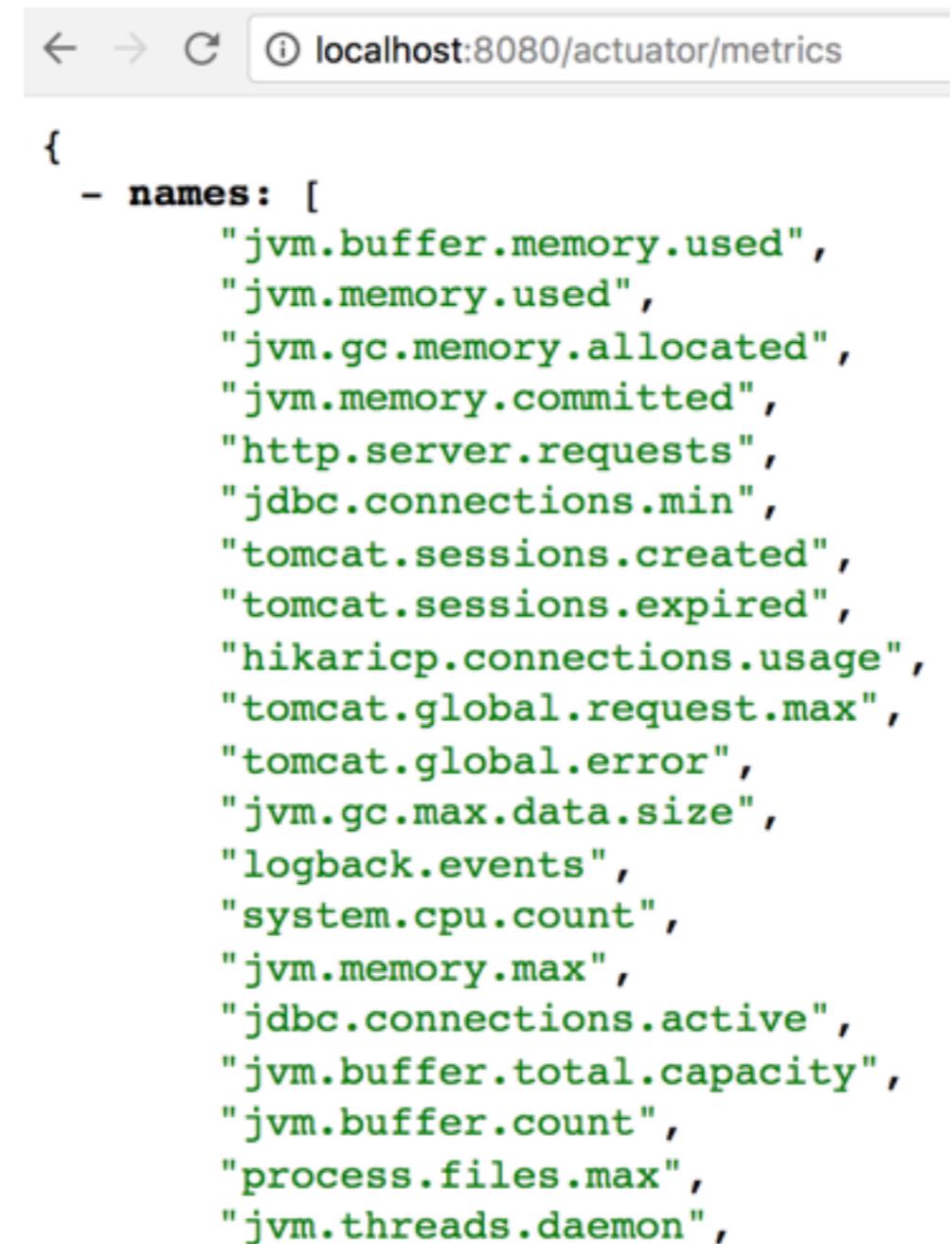
```
← → ⌂ ⓘ localhost:8080/actuator/httptrace

{
  - traces: [
    - {
      timestamp: "2018-03-06T13:33:02.800Z",
      principal: null,
      session: null,
      - request: {
        method: "GET",
        uri: "http://localhost:8080/prometheus",
        - headers: {
          - host: [
            "localhost:8080"
          ],
          - user-agent: [
            "Prometheus/2.0.0"
          ],
          - accept: [
            "text/plain;version=0.0.4;q=1,*/*;q=0.1"
          ],
          - accept-encoding: [
            "gzip"
          ],
          - x-prometheus-scrape-timeout-seconds: [
            "5.000000"
          ]
        },
        remoteAddress: null
      },
    }
  ]
}
```



Spring Boot Actuator (7)

List of metrics endpoint = /actuator/metrics



The screenshot shows a browser window with the URL `localhost:8080/actuator/metrics` in the address bar. The page displays a JSON object representing a list of metric names:

```
{  
  - names: [  
    "jvm.buffer.memory.used",  
    "jvm.memory.used",  
    "jvm.gc.memory.allocated",  
    "jvm.memory.committed",  
    "http.server.requests",  
    "jdbc.connections.min",  
    "tomcat.sessions.created",  
    "tomcat.sessions.expired",  
    "hikaricp.connections.usage",  
    "tomcat.global.request.max",  
    "tomcat.global.error",  
    "jvm.gc.max.data.size",  
    "logback.events",  
    "system.cpu.count",  
    "jvm.memory.max",  
    "jdbc.connections.active",  
    "jvm.buffer.total.capacity",  
    "jvm.buffer.count",  
    "process.files.max",  
    "jvm.threads.daemon",  
  ]  
}
```



Spring Boot Actuator (8)

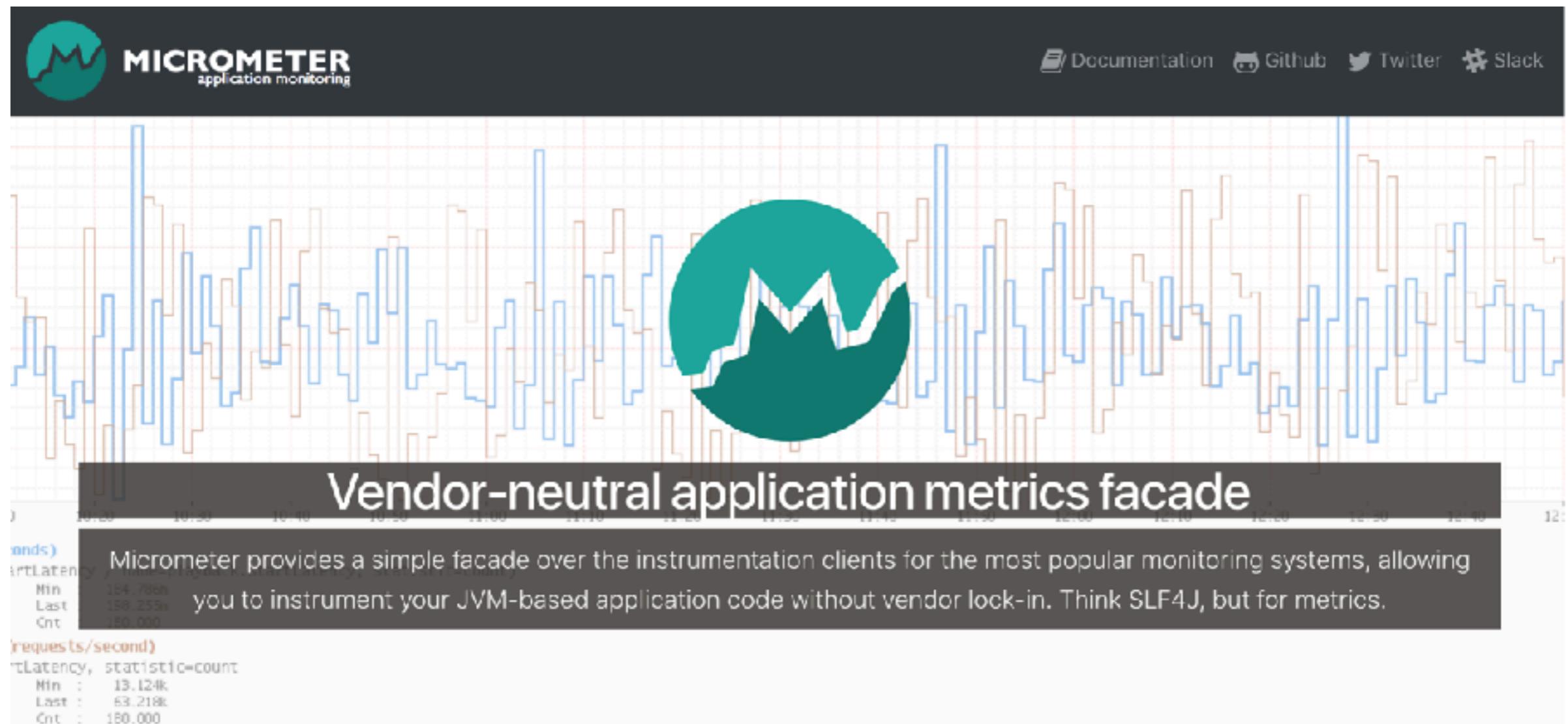
/actuator/metrics/http.server.requests

```
← → ⌂ localhost:8080/actuator/metrics/http.server.requests

{
  name: "http.server.requests",
  - measurements: [
    - {
      statistic: "COUNT",
      value: 269
    },
    - {
      statistic: "TOTAL_TIME",
      value: 1.1072010200000002
    },
    - {
      statistic: "MAX",
      value: 0.04373569
    }
  ],
  - availableTags: [
    - {
      tag: "exception",
      - values: [
        "None"
      ]
    },
    - {
      tag: "method",
      - values: [
        "GET"
      ]
    }
  ],
}
```



Spring Boot 2.0 with MicroMeter



<https://micrometer.io/>



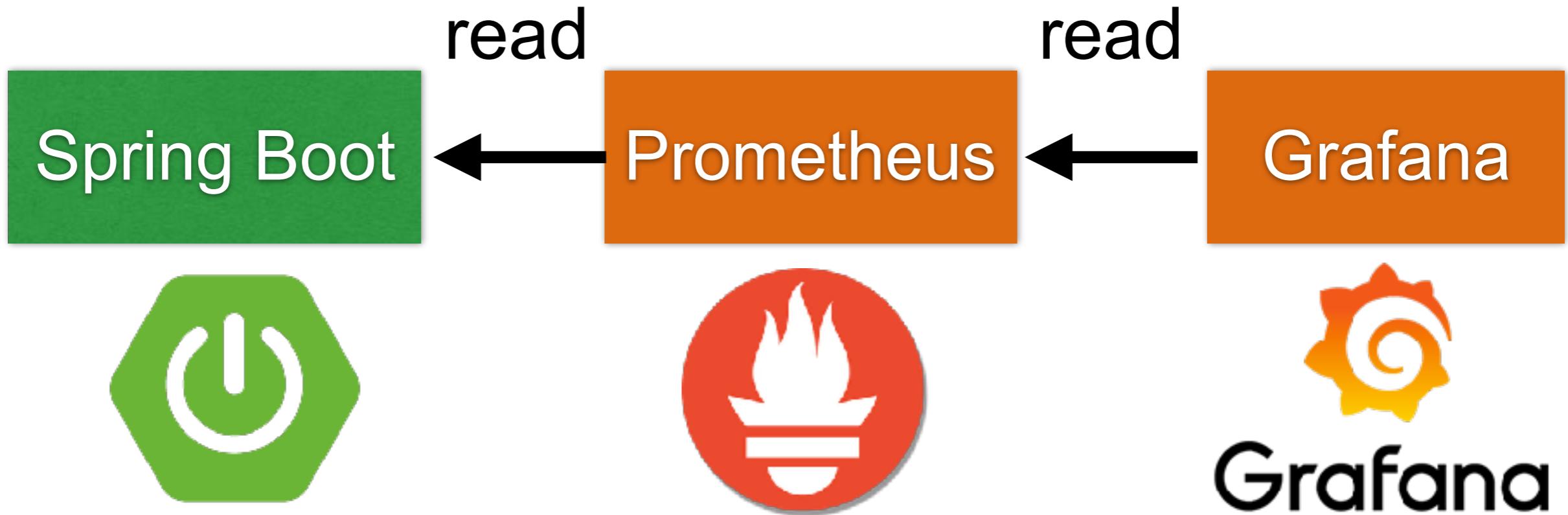
Keep and Visualize Metric of Spring Boot Services



Sample Architecture



Sample Architecture



Service metric for Prometheus



Enable Prometheus (1)

Add library to pom.xml

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
    <version>1.0.1</version>
</dependency>
```



Enable Prometheus (2)

Enabled endpoint in application.properties

```
management.endpoints.web.exposure.include  
=....,prometheus
```



Enable Prometheus (3)

New endpoint = actuator/prometheus

```
← → ⌂ ⓘ localhost:8080/actuator/prometheus

# HELP jvm_memory_used_bytes The amount of used memory
# TYPE jvm_memory_used_bytes gauge
jvm_memory_used_bytes{area="nonheap",id="Code Cache",} 1.49056E7
jvm_memory_used_bytes{area="nonheap",id="Metaspace",} 5.6766712E7
jvm_memory_used_bytes{area="nonheap",id="Compressed Class Space",} 7617096.0
jvm_memory_used_bytes{area="heap",id="PS Eden Space",} 1.7135864E7
jvm_memory_used_bytes{area="heap",id="PS Survivor Space",} 1.6235192E7
jvm_memory_used_bytes{area="heap",id="PS Old Gen",} 2.1936456E7
# HELP hikaricp_connections_idle Idle connections
# TYPE hikaricp_connections_idle gauge
hikaricp_connections_idle{pool="HikariPool-1",} NaN
# HELP tomcat_threads_config_max
# TYPE tomcat_threads_config_max gauge
tomcat_threads_config_max{name="http-nio-8080",} 200.0
# HELP tomcat_servlet_error_total
# TYPE tomcat_servlet_error_total counter
tomcat_servlet_error_total{name="default",} 0.0
# HELP jvm_threads_peak The peak live thread count since the Java virtual machine start
# TYPE jvm_threads_peak gauge
jvm_threads_peak 28.0
# HELP hikaricp_connections_pending Pending threads
# TYPE hikaricp_connections_pending gauge
hikaricp_connections_pending{pool="HikariPool-1",} NaN
# HELP system_cpu_count The number of processors available to the Java virtual machine
```



Keep data in Prometheus

<https://prometheus.io/>



Prometheus



Prometheus

DOCS

DOWNLOAD

COMMUNITY

BLOG



From metrics to insight

Power your metrics and alerting with a leading
open-source monitoring solution.

GET STARTED

DOWNLOAD

Prometheus v2.0 is available now — [Read the announcement blog post!](#)

<https://prometheus.io/>



Prometheus

PUBLIC | AUTOMATED BUILD

[prom/prometheus](#) 

Last pushed: 17 hours ago

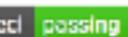
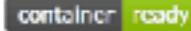
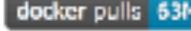
[Repo Info](#) [Tags](#) [Dockerfile](#) [Build Details](#)

Short Description

Short description is empty for this repo.

Full Description

Prometheus 

 

 53M

Visit prometheus.io for the full documentation, examples and guides.

Prometheus is a systems and service monitoring system. It collects metrics

Docker Pull Command

`docker pull prom/prometheus`

Owner



prom

Source Repository

 [prometheus/prometheus](#)

<https://hub.docker.com/r/prom/prometheus/>



Create container of Prometheus

```
$ docker container run --rm  
  -p 9090:9090  
  -v $(pwd)/prometheus.yml:/etc/prometheus/  
prometheus.yml  
  --name monitor prom/prometheus
```



Check Data in Prometheus

http://localhost:9090/

The screenshot shows the Prometheus web interface at the URL `http://localhost:9090/graph`. The top navigation bar includes links for Prometheus, Alerts, Graph, Status, and Help. A checkbox labeled "Enable query history" is checked. Below it is a text input field for the expression query, containing the placeholder text "- insert metric at cursor -". There are two tabs: "Graph" (which is selected) and "Console". A table below shows one row with the element "no data" and the value "Value". At the bottom left is a blue button labeled "Add Graph".



Check Target in Prometheus

Status -> Targets

The screenshot shows the Prometheus web interface at the URL `localhost:9090/targets`. The top navigation bar includes links for Prometheus, Alerts, Graph, Status, and Help. The main title is "Targets". A checkbox labeled "Only unhealthy jobs" is unchecked. Below this, a section titled "spring-boot (1/1 up)" contains a "show less" button. A table lists one target endpoint:

Endpoint	State	Labels	Last Scrape	Error
http://10.10.99.59:8080/actuator/prometheus	UP	instance="10.10.99.59:8080"	2.355s ago	



Show data in Grafana

<https://grafana.com/>



Grafana

The open platform for beautiful analytics and monitoring

The leading open source software for time series analytics

Get Grafana

APP Grafana TestData By Grafana Project

APP kubernetes By Raintank Inc.

APP Kentik Connect Pro

APP NS1 for Grafana By NS1.

Grafana

<https://grafana.com/>



Grafana

PUBLIC REPOSITORY

[grafana/grafana](#) 

Last pushed: 25 minutes ago

[Repo Info](#)

[Tags](#)

Short Description

The official Grafana docker container

Full Description

Grafana Docker image

This project builds a Docker image with the latest master build of Grafana.

Running your Grafana container

Start your container binding the external port 3000 .

```
docker run -d --name=grafana -p 3000:3000 grafana/grafana
```

Docker Pull Command

```
docker pull grafana/grafana
```

Owner



grafana

<https://hub.docker.com/r/grafana/grafana/>



Create container of Grafana

```
$docker container run  
--name=grafana  
-p 3000:3000 grafana/grafana
```



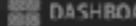
Grafana Dashboard

<https://grafana.com/dashboards/4701>

All dashboards » **JVM (Micrometer)**

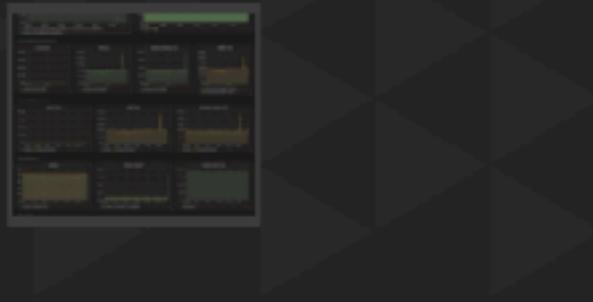


JVM (Micrometer) by mweirauch

 DASHBOARD

Dashboard for Micrometer instrumented applications (Java, Spring Boot)
Last updated: 21 days ago

[Overview](#) [Revisions](#)



Get this dashboard:

[4701](#) [Copy ID to Clipboard](#)

A dashboard for **Micrometer** instrumented applications (Java, Spring Boot).

Features

- JVM memory
- Process memory (provided by `micrometer-jvm-extras`)
- CPU-Usage, Load, Threads, File Descriptors, Log Events
- JVM Memory Pools (Heap, Non-Heap)
- Garbage Collection

Dependencies:

 GRAFANA 4.6.3

 GRAPH

[Download JSON](#) [How do I import this dashboard?](#)



Take to your home

Always improve, always practice



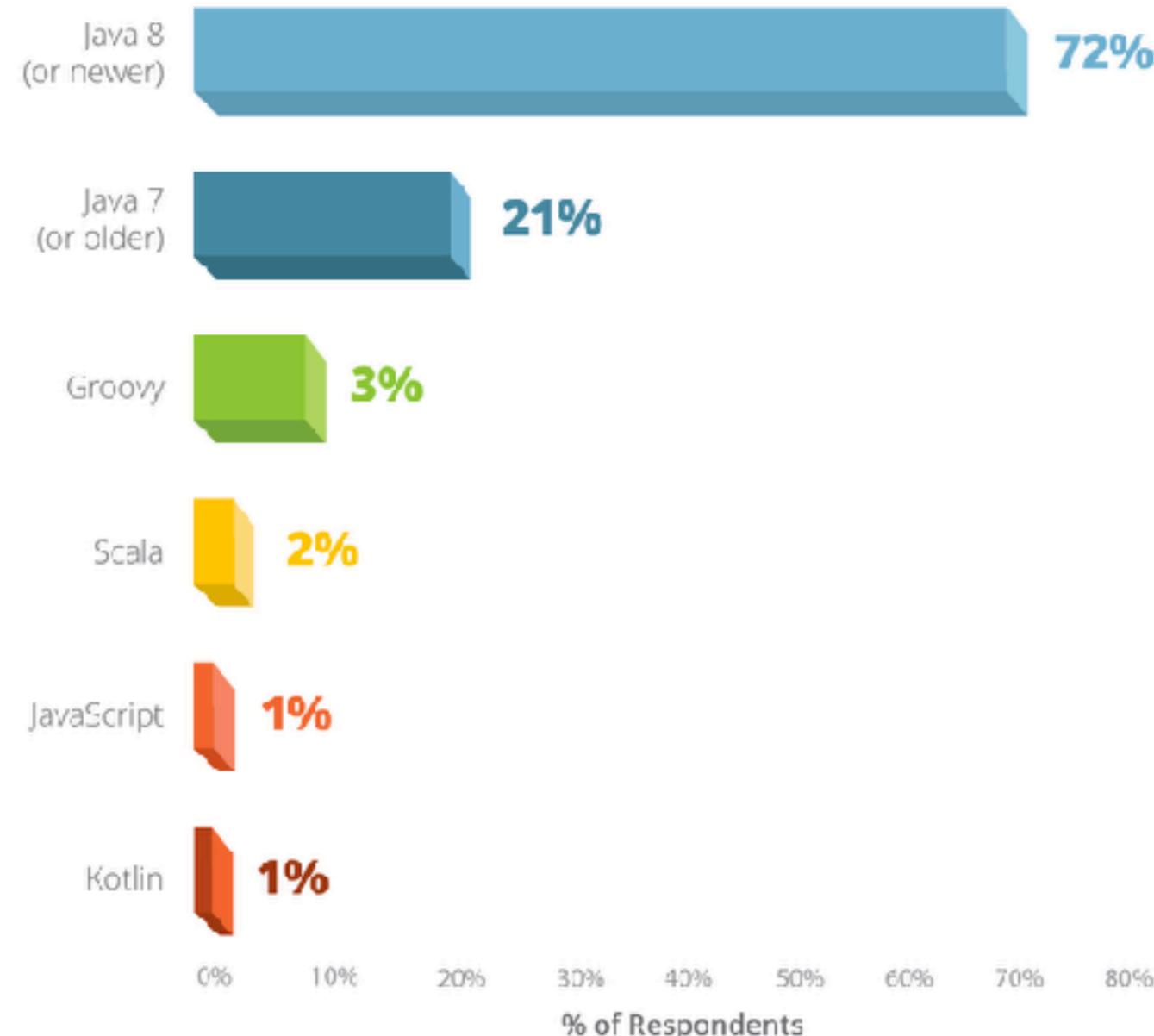
Developer productivity report

<https://zeroturnaround.com/>



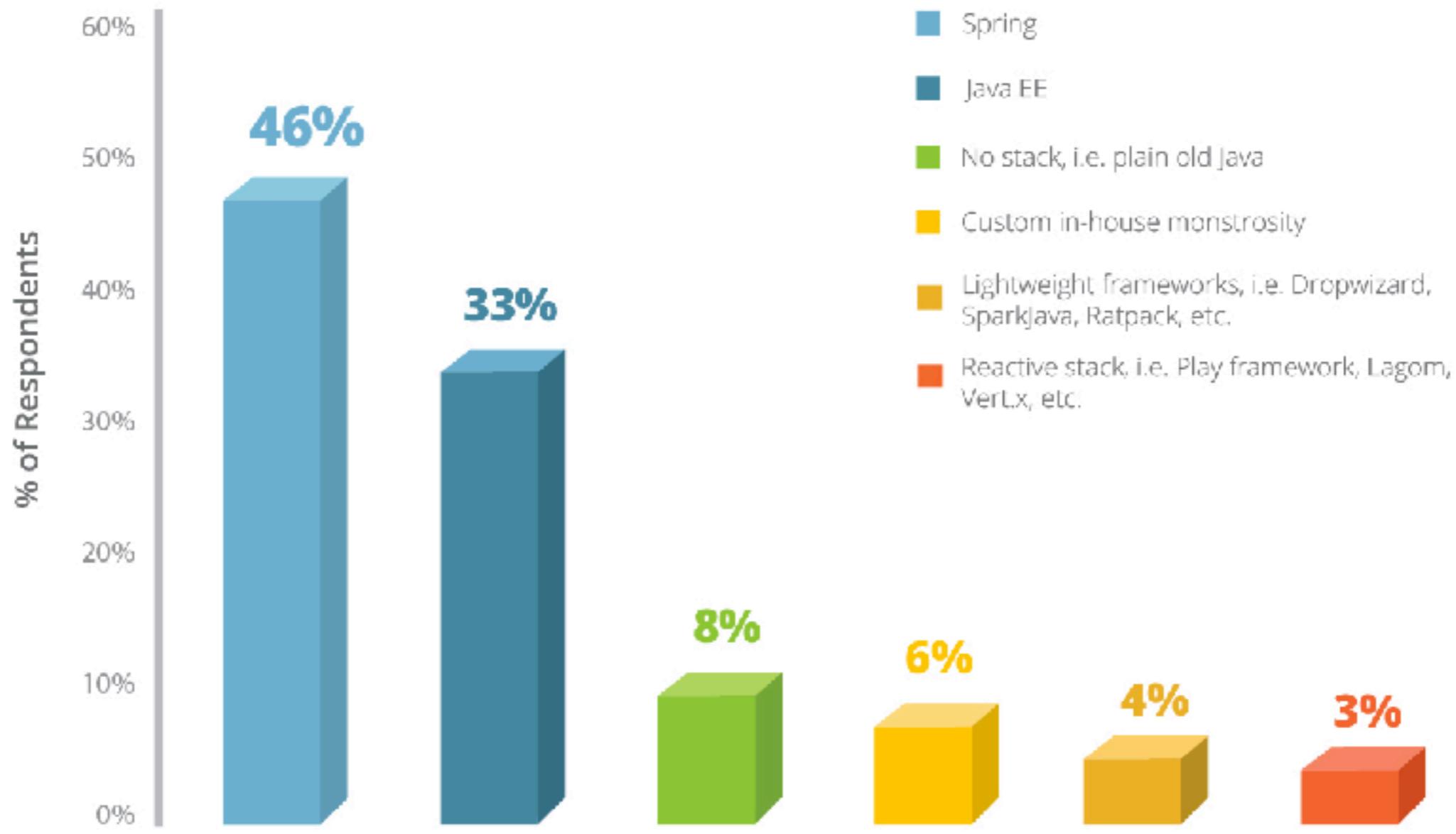
JVM languages

Plain old Java **dominates the language preference**



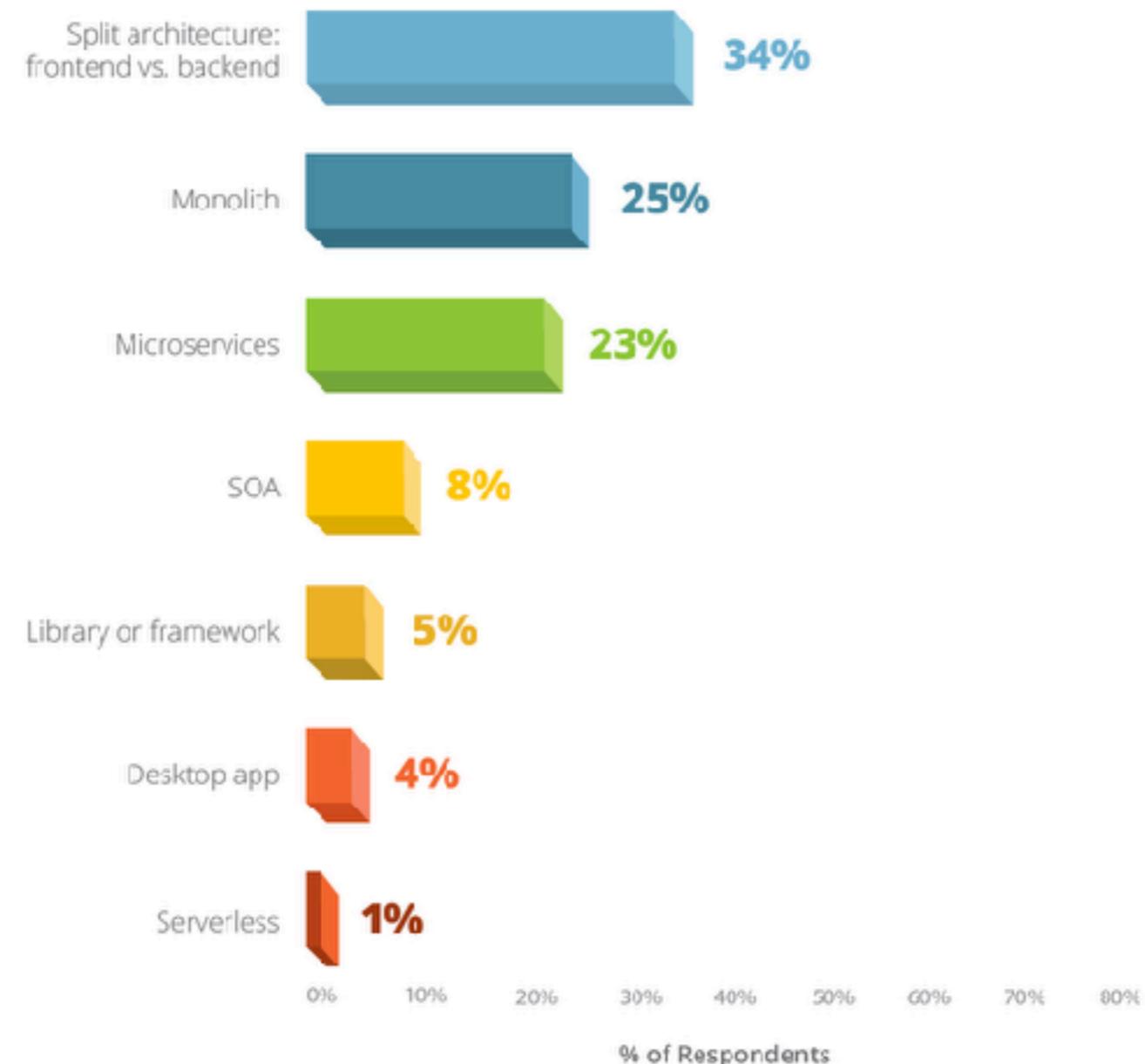
Java frameworks

The **Spring vs. Java EE debate** is going nowhere

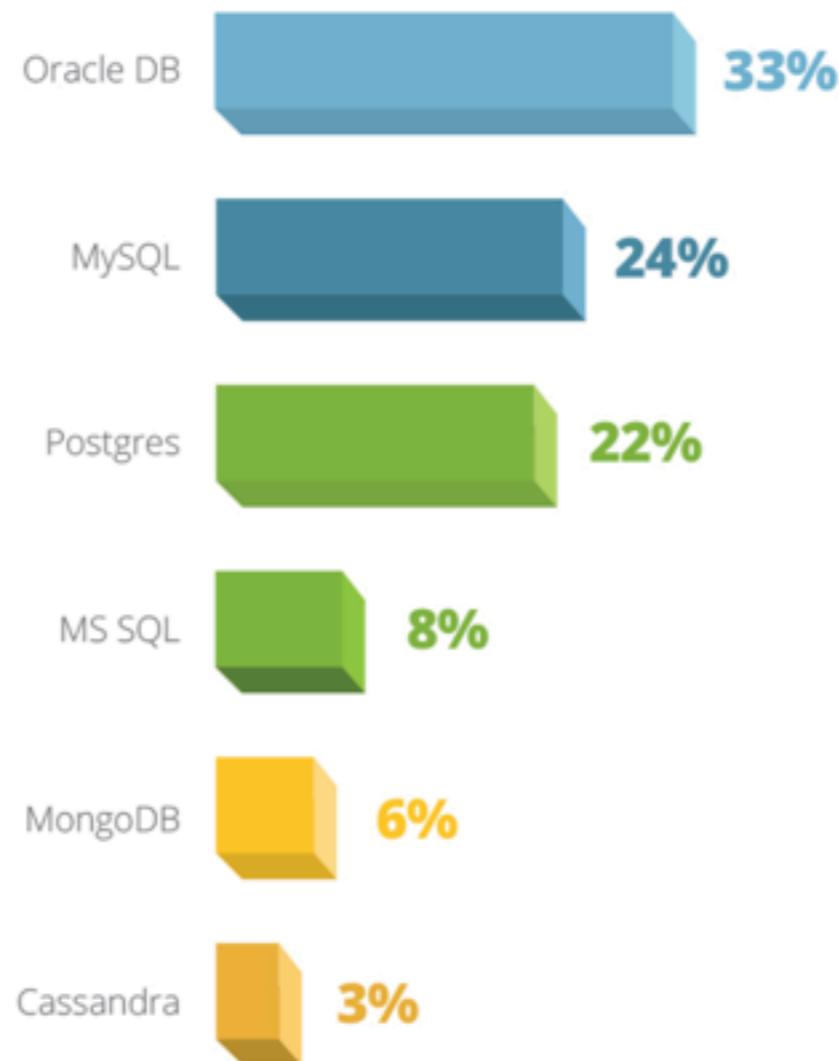


Architecture ?

Architecture choice is largely split, except for split architecture



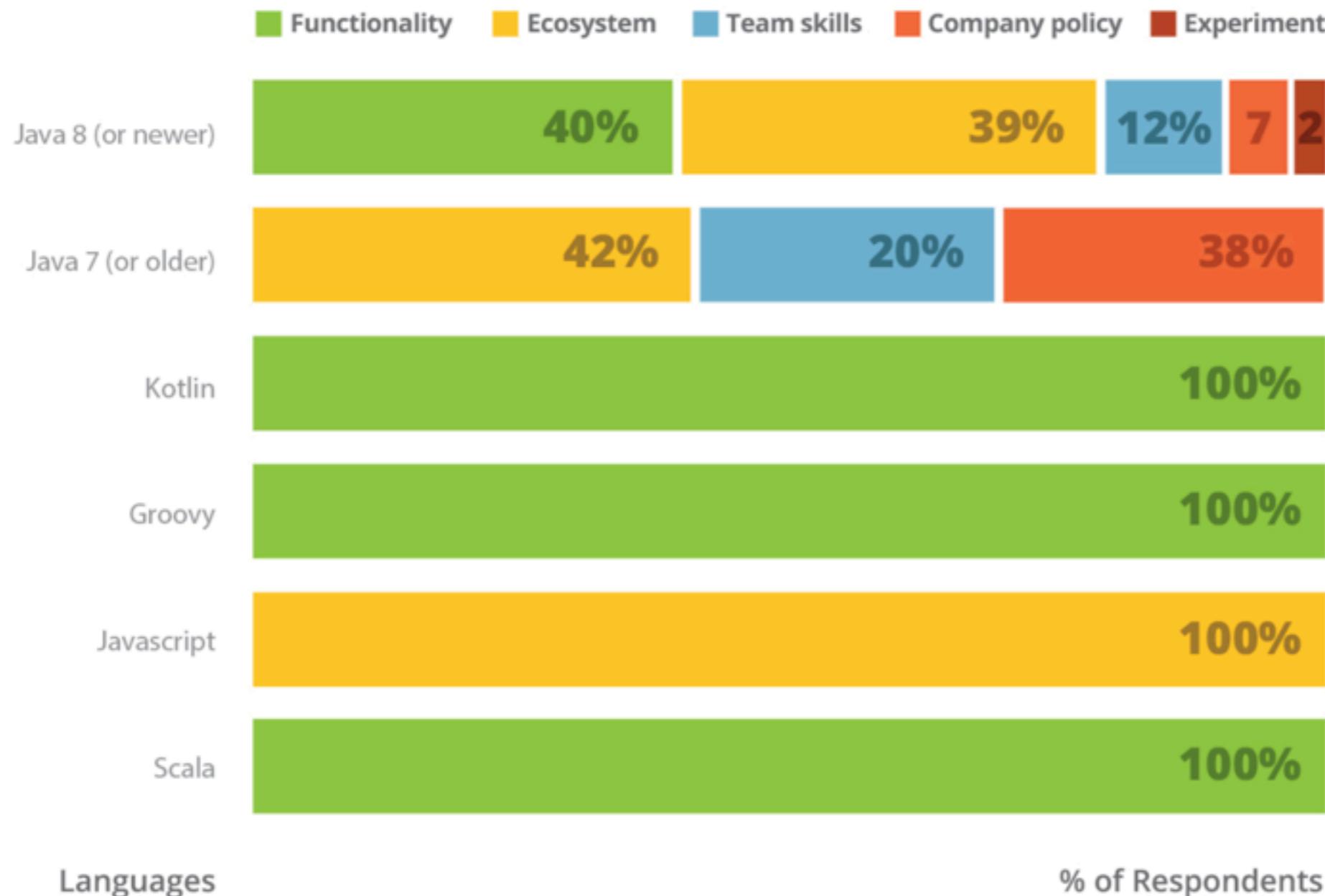
Database



Other languages



Reasons to use



Are you too busy to improve?



Thank you
Q/A

