Spring Boot Developer course

2 Day Instructor-led Training

Version 1.5.b

Copyright Notice

Course Introduction

Spring Boot Developer

Pivotal

Logistics

- Student introductions
- Self introduction
- Course registration (if needed)
- Courseware
- Internet access
- Phones on silent

Working hours

Lunch and breaks

Toilets/Restrooms

Fire alarms

Emergency exits

Any other questions?

Course Objectives

- Learn to use Spring Boot for web and other applications
- Gain hands-on experience
- Generous mixture of presentation and labs

Covered in this section

- Agenda
- Spring and Pivotal

Agenda: Day 1

- Spring Framework
- Spring Boot Overview
- Spring Boot Internals
- Spring Boot Features
- Web Development with Spring Boot
- Data Access with Spring Boot

Pivotal

Agenda: Day 2

- Spring Boot Testing
- Spring Boot Actuator
- Spring Boot Security
- Spring Boot Messaging
- Spring Boot MicroServices

Pivotal

Covered in this section

- Agenda
- Spring and Pivotal

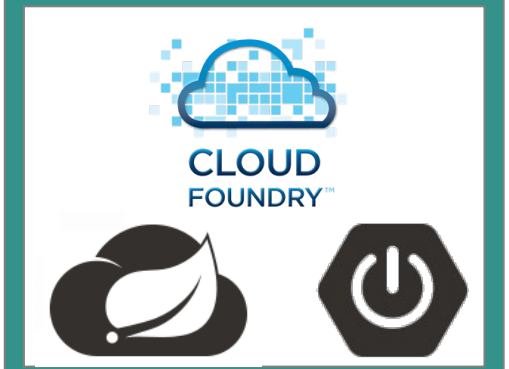
Spring and Pivotal

- SpringSource, the company behind Spring
  - acquired by VMware in 2009
  - transferred to Pivotal joint venture 2013
- Spring projects key to Pivotal's big-data and cloud strategies
  - Virtualize your Java Apps
    - Save license cost
    - Deploy to private, public, hybrid clouds
  - Real-time analytics
    - Spot trends as they happen
    - Spring Data, Spring Hadoop, Spring XD & Pivotal HD

The Pivotal World

Cloud Foundry

Cloud Independence
Microservices
Continuous Delivery
Dev Ops



Development

Frameworks
Services
Analytics

Big Data Suite

High Capacity
Real-time Ingest
SQL Query
Scale-out Storage

Pivotal **Labs** *Working with clients to build better apps more quickly*

Pivotal™

Spring Projects

Spring Security

Spring Data

Spring Batch

Spring Integration

Spring (SOAP) Web Services

Spring Social

Spring Cloud

Spring Framework

Spring AMQP

Spring Session

Spring Hateoas

Spring Android

Spring Web Flow

Spring Reactor

Spring Cloud Data Flow

Spring Boot

Spring Mobile

Labs

- repetition is the key of mastering
- better to start fresh
- some times copy is good

A NEW PLATFORM FOR A NEW ERA

Spring Framework

Spring Boot Developer

A quick introduction

Pivotal

Agenda

- Spring Framework
- Spring Application Development

Spring Framework

- open source
- lightweight
- container
- framework

Spring Framework

Open Source

- binary and source freely available
- apache 2 license
- maven central
- well documented

Spring Framework

Lightweight

- a J2EE Server is not required
- is not invasive
- low overhead

Spring Framework

Container

- spring serves as a container for your application objects
- uses dependency injection to instantiate your objects

Spring Framework

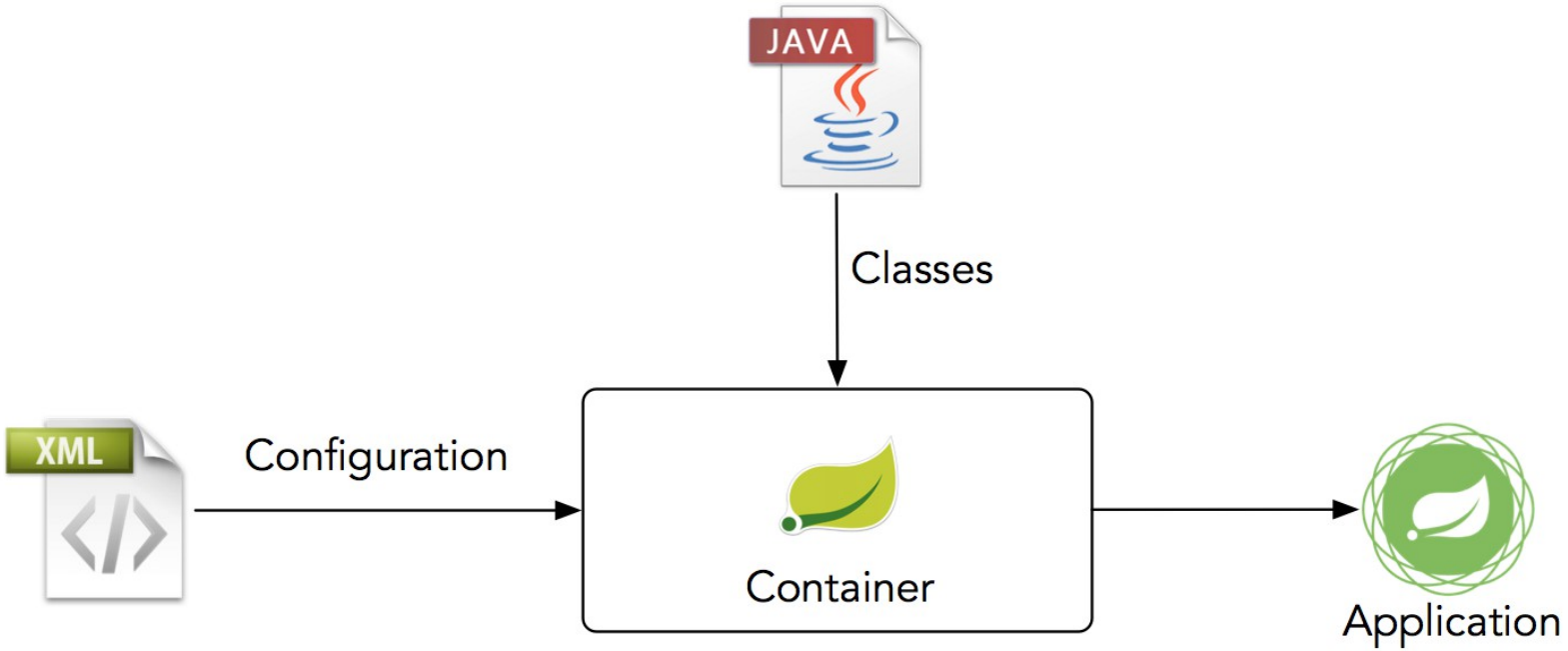
Framework

- provides framework classes to simplify working with lower-level technologies

Agenda

- Spring Framework
- Spring Application Development

Spring Application Development



Spring Application Development

Configuration

- XML
- Java Config
- Annotations

Classes

- POJOs

Demo

REST API with Spring

Summary

- Spring Framework
  - open source, lightweight, container, framework
- Spring Application Development
  - configuration + classes >> container = application
  - web Application
    - DispatcherServlet, JPA and XML Config, deployment

A NEW PLATFORM FOR A NEW ERA

Spring Boot

Spring Boot Developer

An overview of Spring Boot

Pivotal

Spring Framework

remember?

- spring web development requirements:
- web:
  - DispatcherServlet, XML Context
- data:
  - DataSource, TransactionManager, EntityManagerFactory
- dependency:
  - maven, gradle, ant, ivy
- logging, property files, monitoring, metrics, security?

Agenda

- spring boot
- spring boot application development

Spring Boot

what is spring boot?

- OPINIONATED runtime for Spring projects
- next generation of Spring applications
- rapid application development
- easy to use features

Spring Boot

provides

- sensible defaults
- auto-configuration
- ability to create stand-alone (server-less: runnable) and deployable applications
- full control over any configuration:
  - xml, java config, annotations, application.properties/yml

Spring Boot

supports different project types

- web, batch, jdbc, integration, messaging, cloud, and more...

Spring Boot

It is not

- IDE plugin
- code generator
- scaffolding

Agenda

- spring boot
- spring boot application development

Spring Boot Application Development

spring boot components

- dependency management:
  - maven, gradle, ant, ivy
  - spring-boot-starter technology
- main application
  - @SpringBootApplication
  - SpringApplication.run

Spring Boot Application Development

ways to create a spring boot application

- spring boot initializr
- IDE (spring tool suite / intellij / Netbeans)
- spring boot CLI

Demo

Simple Spring Boot app

Lab

Create a REST Spring Boot Web App

Summary

- Spring Boot
  - opinionated runtime for spring projects
  - provides sensible defaults (best practices)
  - components:
    - dependency, starter, @SpringBootApplication, SpringApplication.run
  - spring initializr, IDE, spring boot cli

A NEW PLATFORM FOR A NEW ERA

Spring Boot Internals

Spring Boot Developer

An overview of auto-configuration

Pivotal

Spring Boot Internals

Remember?

- Spring Boot is an ***OPINIONATED*** runtime for Spring projects

Agenda

- Spring Boot auto-configuration

Spring Boot auto-configuration

- Spring Boot uses sensible defaults based on what dependencies are on the classpath

Spring Boot auto-configuration

- auto-configuration is enabled by using the *@EnableAutoConfiguration* annotation

- where or how to use this annotation?

Spring Boot auto-configuration

- *@SpringBootApplication* is a composite annotation.

```
1
2  //...
3  @Inherited
4  @SpringBootConfiguration
5  @EnableAutoConfiguration
6  @ComponentScan
7  public @interface SpringBootApplication {
8
9      //...
10
11 }
```

Spring Boot auto-configuration

- *@EnableAutoConfiguration* reads the spring-boot-autoconfigure/META-INF/spring.factories

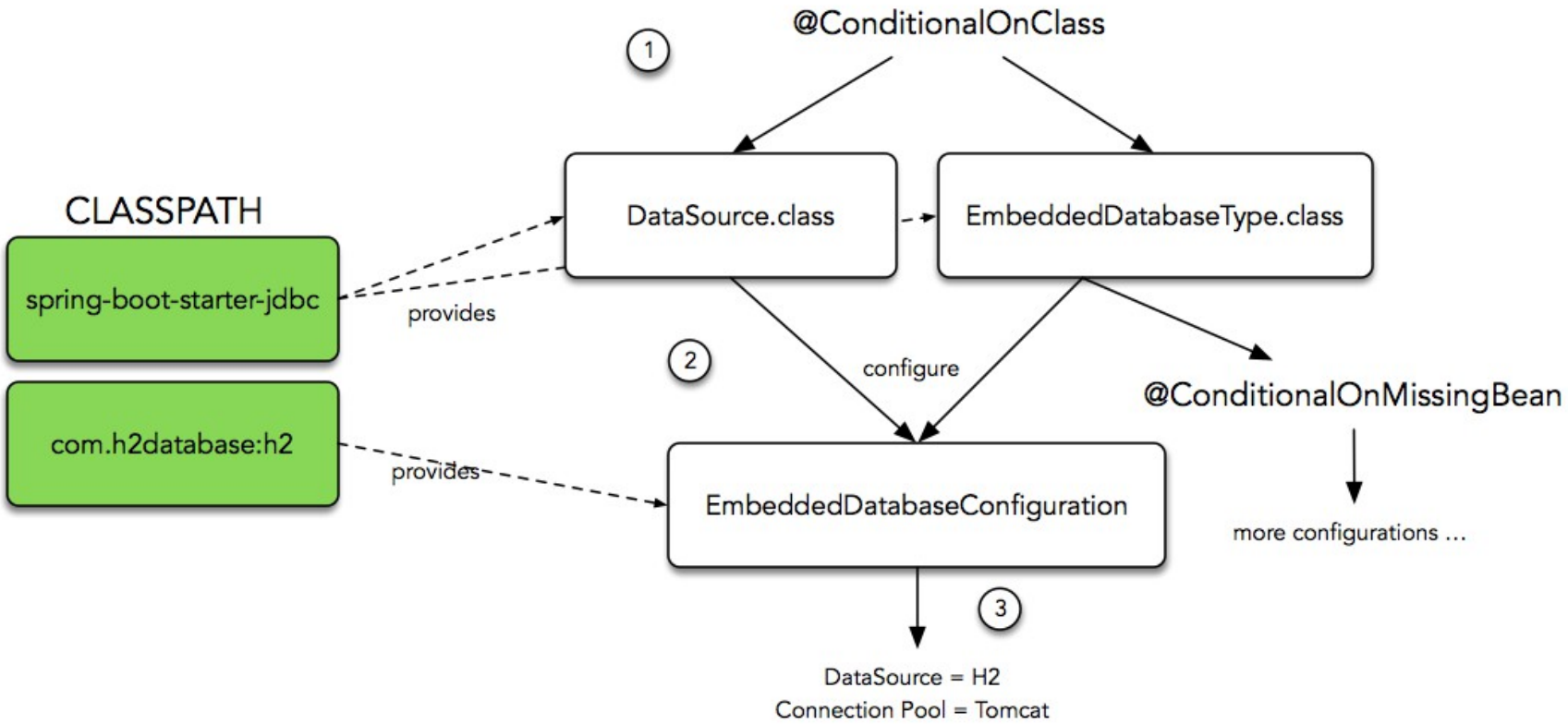
- the spring.factories contains all the classes that will match on what there is in the classpath.

Spring Boot auto-configuration

auto-configuration will use:

- *@ConditionalOnClass*
- *@ConditionalOnBean*
- and more ... to set the defaults for the Spring application.

Spring Boot auto-configuration



1. Is there a DataSource and EmbeddedDatabaseType classes in the classpath? Is there any DataSource Bean defined?

Demo

DataSourceAutoConfiguration review

Lab

Using @Conditional annotations...

Summary

- Spring Boot Internals
  - opinionated runtime for spring projects
  - provides sensible defaults (best practices)
- auto-configuration
  - based on annotations: *@Conditional*

A NEW PLATFORM FOR A NEW ERA

Spring Boot Features

Spring Boot Developer

Discovering Spring Boot features

Pivotal

Agenda

- Packaging
- Spring Application
- External Configuration
- Profiles
- Logging

Spring Boot Features: packaging

spring boot can create executable applications

Maven:

./mvnw package

Gradle:

./gradlew build

Run:

java -jar myapp.jar

Spring Boot Features: packaging

- a spring boot web application will have an embedded servlet container
- spring boot supports: tomcat, undertow and jetty
- tomcat is the default

Spring Boot Features: packaging

a executable / deployable WAR must have:

maven

```
<packaging>war</packaging>

<!-- ... -->

<dependencies>

    <!-- ... -->

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>

</dependencies>
```

gradle

```
//...
apply plugin: 'war'


dependencies {
    //...
    providedRuntime('org.springframework.boot:spring-boot-starter-tomcat')
    //...
}
```

```
public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(DemoApplication.class);
    }

}
```

Spring Boot Features: packaging

spring boot allows to override the defaults:

maven

```
<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

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

</dependencies>
```

gradle

```
configurations {
    compile.exclude module: "spring-boot-starter-tomcat"
}


dependencies {
    compile('org.springframework.boot:spring-boot-starter-web')
    compile('org.springframework.boot:spring-boot-starter-jetty')
    // ...
}
```

Demo

packaging

Agenda

- Packaging
- Spring Application
- External Configuration
- Profiles
- Logging

Spring Boot Features: SpringApplication

SpringBootApplication bootstraps a Spring application, it provides:

- a way to customize the banner
- customize the application through application.properties
- a fluent API builder: SpringApplicationBuilder
- events and listeners
- web environment: setWebEnvironment(false)
- access to application arguments
- run specific code once the SpringApplication has started
- admin features: MBeanServer

Demo

SpringApplication

Agenda

- Packaging
- Spring Application
- External Configuration
- Profiles
- Logging

Spring Boot Features: External Configuration

spring boot allows externalize configuration to use the same code in different environments through:

- application.properties / application.yml
- environment variables
- command line arguments

Spring Boot Features: External Configuration

- property values can be injected using *@Value* annotation or can be bound to structured objects via *@ConfigurationProperties*
- spring boot uses a *PropertySource* order to allow value overriding
- spring boot uses relaxed binding rules for binding

Spring Boot Features: External Configuration

spring boot uses a very particular *PropertySource* order that is designed to allow sensible overriding of values; to name a few:

...

Command line arguments.

Properties from SPRING_APPLICATION_JSON

...

Java System properties (System.getProperties()).

OS environment variables.

...

Profile-specific application properties outside of your packaged jar

Profile-specific application properties packaged inside your jar

Application properties outside of your packaged jar

Application properties packaged inside your jar

...

Spring Boot Features: External Configuration

SpringApplication will load properties from *application.properties* files in the following locations and add them to the spring Environment:

- */config* subdirectory of the current directory.
- current directory
- classpath */config* package
- classpath root

Demo

External Configuration

Agenda

- Packaging
- Spring Application
- External Configuration
- Profiles
- Logging

Spring Boot Features: Profiles

spring boot allows to use profile-specific properties:

- *application-{profile}.properties*
- a single *application.yml* that contains profiles blocks:

```
spring:
  application:
    name: directory-service

---
spring:
  profiles: qa

directory:
  host: 192.168.3.12
  user: qauser
  pass: qapwd

---
spring:
  profiles: production

directory:
  host: directory-service.cfapps.io
  user: dsuser
  pass: {cypher}{682bc583f4641835fa2db009355293665d2647dade3375c0ee201de2a49f7bda}
```

Spring Boot Features: Profiles

maven:

./mvnw spring-boot:run -Dspring.profiles.active=dev

gradle:

```
//build.gradle

bootRun {
    systemProperty "spring.profiles.active", System.getProperty("spring.profiles.active")
}
```

./gradlew bootRun -Dspring.profiles.active=dev

JAR:

SPRING_PROFILES_ACTIVE=production java -jar myapp.jar

or

java –Dspring.profiles.active=qa -jar myapp.jar

Agenda

- Packaging
- Spring Application
- External Configuration
- Profiles
- Logging

Spring Boot Features: Logging

- spring boot uses Commons Logging for all internal logging, Logback will be used by default.

- spring boot support logger levels configuration: *TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF* through the *logging.level.** properties in the *application.properties/yml* file

```
logging.level.root=WARN
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR
logging.level.io.pivotal.workshop=DEBUG
```

Lab

Spring Boot Features

Summary

- packaging: JAR and WAR
  - WAR: executable and deployable
- SpringApplication
  - customizable: banner, fluent API builder, etc
- external configuration
- profiles
- logging

Pivotal

A NEW PLATFORM FOR A NEW ERA

Web Development with Spring Boot

Spring Boot Developer

Spring MVC

Pivotal

Spring Web Development

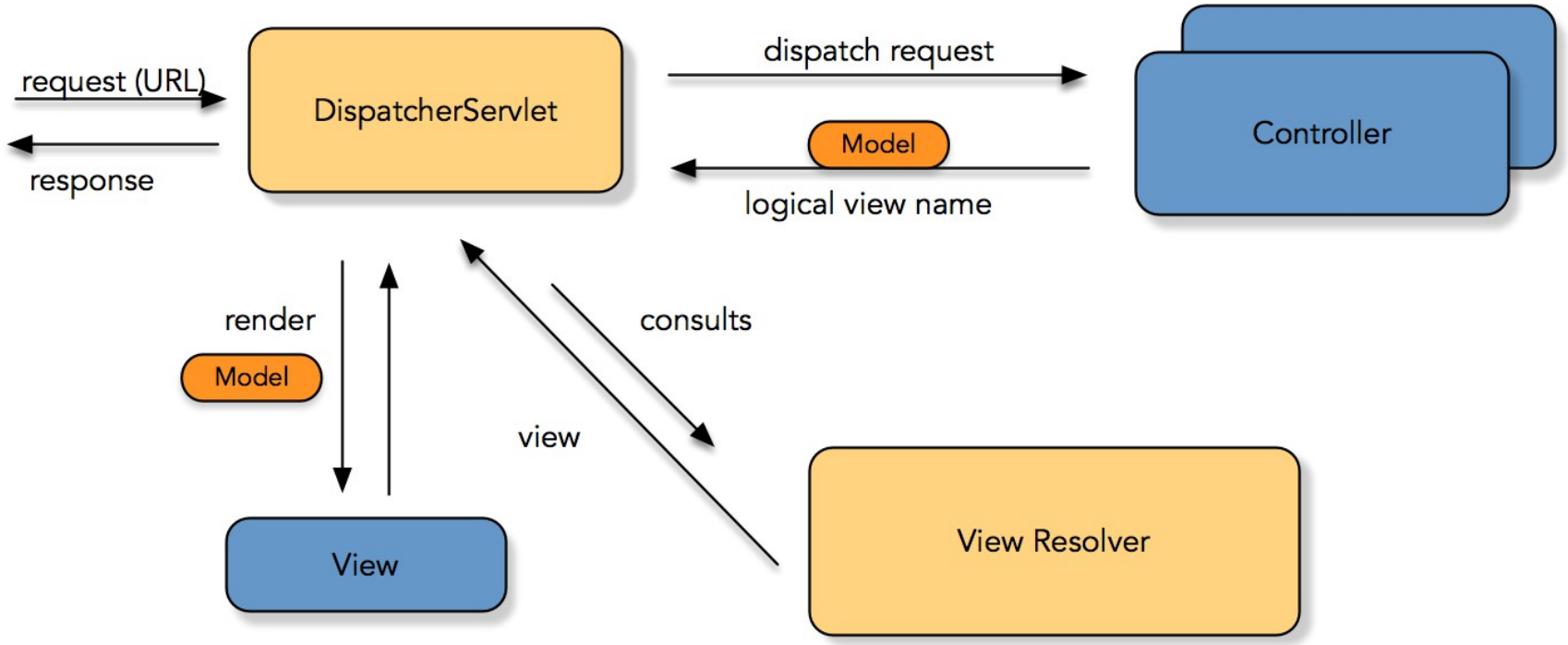
Remember?

- web.xml
- DispatcherServlet
- component-scan
- view resolvers

Agenda

- Spring Web MVC
- Spring Boot web development

Spring Web MVC



Agenda

- Spring Web MVC
- Spring Boot web development

Spring Boot Web Development

- spring boot uses the power of Spring Web MVC to create powerful web application with ease

- spring boot web applications can be created by adding the spring-boot-starter-web dependency

Spring Boot Web Development

- the spring-boot-starter-web dependency brings the spring-web, spring-webmvc jars and additional libraries that make the web application server-less:
  - tomcat, jackson, etc.

Spring Boot Web Development

- spring boot will auto-configure the DispatcherServlet, content and view resolvers
- you can use all the spring-mvc annotations:
  - @Controller / @RestController
  - @RequestMapping
  - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping
  - @PathVariable, @RequestParam, @RequestHeader, @RequestBody, @RequestAttribute, @ModelViewAttribute
  - @SessionAttributes, @ModelAttribute, @CookieValue
  - @ControllerAdvice, @RestControllerAdvice
  - @RequestPart, @ExceptionHandler,
  - ServletRequest, HttpServletRequest, Principal, and more...

Spring Boot Web Development

supports for serving static resources:

- */static*, */public*, */META-INF/resources*
- */webjars/***
- *index.html* and custom *favicon*

Spring Boot Web Development

- support for json and xml serialization
- multiple template engine support:
  - groovy server pages
  - freemaker
  - velocity
  - mustache
  - thymeleaf

Spring Boot Web Development

support embedded servlet containers:

- servlet 3.x engines
- access and compatibility with J2EE annotations:
  - @WebServlet
  - @WebFilter
  - @WebListener

Spring Boot Web Development

customization through *application.properties/yml*

custom network configuration:

- server.port
- server.address

custom embedded servlet container configuration:

- server.session.*
- server.compression.*

programmatic customization by implementing the *EmbeddedServletContainerCustomizer* interface

Spring Boot Web Development

custom error pages by providing:

- *resources/public/error/<status-code>.html*

Demo

Web App with Spring Boot

Lab

Web Directory application

Summary

- Spring Boot Web Development
  - opinionated runtime for web projects
  - uses the power of spring web mvc
  - highly customizable

Pivotal

A NEW PLATFORM FOR A NEW ERA

Data Access with Spring Boot

Spring Boot Developer

Data access with JDBC, JPA and REST

Pivotal

Spring Boot Data Access

Remember?

- persistence.xml
- DataSource
- TransactionManager
- EntityFactoryManager

Agenda

- jdbc
- data-jpa
- data-rest
- NoSQL
- additional features

Spring Boot Data Access: jdbc

- spring boot uses the extensive support from the Spring Framework for working with SQL databases
- spring boot uses direct access from JdbcTemplate to complete ORM technologies like Hibernate
- spring boot jdbc applications can be created by adding the *spring-boot-starter-jdbc* and the SQL driver dependencies

Spring Boot Data Access: jdbc

- spring boot will auto-configure the DataSource based on default properties or any existing configuration

- DataSource properties can be overridden in the *application.properties*/*yml* file

```
spring.datasource.url=jdbc:mysql://localhost/testdb
spring.datasource.username=mysqluser
spring.datasource.password=mysqlpasswd
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

- Multiple DataSource bean definitions can exist

Spring Boot Data Access: jdbc

- spring boot uses Tomcat JDBC as the default connection pool
- spring boot support embedded databases: H2, HSQL and Derby
- spring boot uses the Spring JDBC initializer feature, it loads SQL from *schema.sql* and *data.sql*
- spring boot also supports the *schema-${platform}.sql* and *data-${platform}.sql*

Spring Boot Data Access: jdbc

- spring boot auto-configures the *JdbcTemplate* so it's easy to use in any spring bean

```
@Service
public class DirectoryService {

    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public DirectoryService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    // ...

}
```

Demo

JDBC Demo with Spring Boot

Lab

JDBC

Agenda

- jdbc
- data-jpa
- data-rest
- NoSQL
- additional features

Spring Boot Data Access: data-jpa

- spring boot uses the power of the spring data project to create data applications with ease

- spring boot jpa applications can be created by adding the *spring-boot-starter-data-jpa* and the SQL driver dependencies.

Spring Boot Data Access: data-jpa

spring-data provides:

- relies on the Java Persistence API
- repository generation base on interfaces: *Repository*, *CrudRepository*, *JpaRepository*
- custom object mapping
- dynamic query derivation from repository method names
- schema generation through *spring.jpa.** properties
- initialization through an import.sql file

Lab

JPA

Agenda

- jdbc
- data-jpa
- data-rest
- NoSQL
- additional features

Spring Boot Data Access: data-rest

- spring boot will use spring data rest project to create hypermedia-driven REST web services on top of repositories

- spring boot data-rest applications can be created by adding the *spring-boot-starter-data-jpa*, *spring-boot-starter-data-rest* and the SQL driver dependencies

Spring Boot Data Access: data-rest

data-rest features:

- exposes a discoverable REST API for your domain model using HAL as media type
- exposes collection, item and association resources representing your model
- supports pagination via navigational links
- allows to dynamically filter collection resources
- ships a customized variant of the HAL Browser
- currently supports JPA, MongoDB, Neo4j, Solr, Cassandra, Gemfire
- allows advanced customizations of the default resources exposed
- and more ...

Pivotal

Lab

Data Rest

Agenda

- jdbc
- data-jpa
- data-rest
- NoSQL
- additional features

Spring Boot Data Access: NoSQL

- spring boot will use spring data project to create NoSQL data applications with ease

- spring boot will provide auto-configuration for: Redis, MongoDB, Neo4j, Elasticsearch, Solr, Cassandra, Couchbase and LDAP, so is easy to use its respective *<data-technology>Template* class.

- spring boot NoSQL applications can be created by adding the necessary data starter technology dependency

Agenda

- jdbc
- data-jpa
- data-rest
- NoSQL
- additional features

Spring Boot Data Access: additional features

spring boot provides additional tools and functionality:

- higher-level database migration tool: *flyway* and *liquibase*
- h2's web console through: /h2-console endpoint
  - *spring.h2.console.enable = true* to enable it
  - it can be secured
- support for *jOOQ* (Java Object Oriented Querying)
  - code generation through jooq-codegen-maven plugin
  - auto-configuration of the DSLContext interface
  - customization of *jOOQ* by setting spring.jooq.sql-dialect property

Summary

- Spring Boot Data Access
- jdbc: auto-configuration of the DataSource / JdbcTemplate
- jpa: based on spring data > repositories, mapping, query methods
- rest: based on spring data rest > restful implementation of the domain object through HATEOAS
- NoSQL: based on spring data
- additional features: h2-console, flyway and liquibase, jooq

A NEW PLATFORM FOR A NEW ERA

Testing with Spring Boot

Spring Boot Developer

TDD with Spring Boot

Pivotal

Agenda

- Testing
- Spring Boot Testing

Testing

- structure your code with clean separation of concerns so that individual parts can be unit tested.
- TDD is a good way to achieve this.
- use constructor injection to ensure that objects can be instantiated directly. Don’t use field injection as it just makes your tests harder to write.

Agenda

- Testing
- Spring Boot Testing

Spring Boot Testing

- spring boot uses the spring test project to provide an easy way to execute unit and integration tests, facilitating a TDD approach

- spring boot tests can be created by adding the *spring-boot-starter-test* dependency

Spring Boot Testing

- the spring-boot-starter-test dependency provides:
- junit
- spring test & spring boot test
- assertj
- hamcrest
- mockito
- jsonassert
- jsonpath

Old Spring Boot Testing

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=MyApp.class, loader=SpringApplicationContextLoa
der.class)
public class MyTest {

    // ...

}
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
public class MyTest {

    // ...

}
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
@IntegrationTest
public class MyTest {

    // ...

}
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
@WebIntegrationTest
public class MyTest {

    // ...

}
```

Spring Boot Testing

a new spring boot integration test will look like this:

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

    // ...

}
```

Spring Boot Testing

A more concrete example that actually hits a real REST endpoint:

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

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void test() {
        this.restTemplate.getForEntity(
            "/{username}/vehicle", String.class, "Phil");
    }

}
```

Spring Boot Testing

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

    @Autowired
    private TestRestTemplate restTemplate;

    @MockBean
    private VehicleDetailsService vehicleDetailsService;

    @Before
    public void setup() {
        given(this.vehicleDetailsService.
            getVehicleDetails("123")
        ).willReturn(
            new VehicleDetails("Honda", "Civic"));
    }

    @Test
    public void test() {
        this.restTemplate.getForEntity("/{username}/vehicle",
            String.class, "sframework");
    }

}
```

Mocking and Spying

Spring Boot Testing

```
public class VehicleDetailsJsonTests {

    private JacksonTester<VehicleDetails> json;

    @Before
    public void setup() {
        ObjectMapper objectMapper = new ObjectMapper();
        // Possibly configure the mapper
        JacksonTester.initFields(this, objectMapper);
    }

    @Test
    public void serializeJson() {
        VehicleDetails details =
            new VehicleDetails("Honda", "Civic");

        assertThat(this.json.write(details))
            .isEqualToJson("vehicledetails.json");

        assertThat(this.json.write(details))
            .hasJsonPathStringValue("@.make");

        assertThat(this.json.write(details))
            .extractingJsonPathStringValue("@.make")
            .isEqualTo("Honda");
    }

    @Test
    public void deserializeJson() {
        String content = "{\"make\":\"Ford\",\"model\":\"Focus\"}";

        assertThat(this.json.parse(content))
            .isEqualTo(new VehicleDetails("Ford", "Focus"));

        assertThat(this.json.parseObject(content).getMake())
            .isEqualTo("Ford");
    }

}
```

JSON assertions

Spring Boot Testing

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

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private UserRepository repository;

    @Test
    public void findByUsernameShouldReturnUser() {
        this.entityManager.persist(new User("sboot", "123"));
        User user = this.repository.findByUsername("sboot");

        assertThat(user.getUsername()).isEqualTo("sboot");
        assertThat(user.getVin()).isEqualTo("123");
    }

}
```

JPA slice

Spring Boot Testing

```
@RunWith(SpringRunner.class)
@WebMvcTest(UserVehicleController.class)
public class UserVehicleControllerTests {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private UserVehicleService userVehicleService;

    @Test
    public void getVehicleShouldReturnMakeAndModel() {
        given(this.userVehicleService.getVehicleDetails("sboot"))
            .willReturn(new VehicleDetails("Honda", "Civic"));

        this.mvc.perform(get("/sboot/vehicle")
            .accept(MediaType.TEXT_PLAIN))
            .andExpect(status().isOk())
            .andExpect(content().string("Honda Civic"));
    }

}
```

MVC slice

Spring Boot Testing

JSON slice

```
@RunWith(SpringRunner.class)
@JsonTest
public class VehicleDetailsJsonTests {

    private JacksonTester<VehicleDetails> json;

    @Test
    public void serializeJson() {
        VehicleDetails details = new VehicleDetails(
            "Honda", "Civic");

        assertThat(this.json.write(details))
            .extractingJsonPathStringValue("@.make")
            .isEqualTo("Honda");
    }

}
```

Lab

Testing

Summary

- Spring Boot Testing
- provides different libraries: mockito, jsonassert, etc
- provides: @RunWith and @SpringBootTest annotations
- slices: jpa, mvc, json

A NEW PLATFORM FOR A NEW ERA

Spring Boot Actuator

Spring Boot Developer

Out-of-the-box production-ready features

Pivotal

Non-Functional Requirements

- every application nowadays required non-functional requirements, like monitoring, health checks and management

Agenda

- Spring Boot Actuator
- Metrics
- Health Indicators

Spring Boot Actuator

- spring boot includes a number of additional production-ready features to help you monitor and manage your application when it's pushed to production

- adding these production-ready features to a spring boot application is as easy as including the *spring-boot-starter-actuator*

Spring Boot Actuator

- spring boot actuator provides HTTP endpoints through a spring mvc based application
- actuator endpoints allow you to monitor and interact with your application
- actuator endpoints can be exposed also through JMX using jolokia

Spring Boot Actuator

spring boot includes a number of built-in endpoints:

- /autoconfig displays an auto-configuration report
- /beans displays a complete list of all the spring beans
- /dump performs a thread dump
- /env exposes Spring's ConfigurableEnvironment
- /health shows application health information
- /info displays arbitrary application info
- /metrics shows metrics information for the current application
- /mappings displays a collated list of all @RequestMapping paths
- /shutdown allows the application to be gracefully shutdown
- /trace displays trace information

Spring Boot Actuator

- endpoints can be customized using the *application.properties*/*yml*, you can change if an endpoint is enabled, if it's sensitive and its id

syntax:

- *endpoints.[endpoint-name].id*
- *endpoints.[endpoint-name].sensitive*
- *endpoints.[endpoint-name].enabled*

```
endpoints.health.id=status
endpoints.health.sensitive=true

endpoints.beans.id=springbeans
endpoints.beans.sensitive=false

endpoints.shutdown.enabled=true
```

- by default the *spring-boot-actuator* uses the role ACTUATOR to get access to the endpoints if secured.

Spring Boot Actuator

- actuator has *CORS* support, endpoints can be configure what kind of cross domain request are authorized

```
endpoints.cors.allowed-origins=http://monitor-spluk.mydomain.com
endpoints.cors.allowed-methods=GET,POST
```

- you can create a custom endpoint by implementing the *MvcEndpoint* interface

Agenda

- Spring Boot Actuator
- Metrics
- Health Indicators

Spring Boot Actuator: metrics

actuator includes a */metrics* endpoint that exposes system metrics:

- mem — the total system memory
- mem.free — the amount of free memory
- processors — the number of processors
- uptime — the system uptime in ms
- instance.uptime — application context uptime in ms
- systemload.average — the average system load
- heap.* — heap information in KB
- thread.* — thread information
- classes.* — class load information
- gc.* — garbage collection information


```
{
    "counter.status.200.root": 20,
    "counter.status.200.metrics": 3,
    "counter.status.200.star-star": 5,
    "counter.status.401.root": 4,
    "gauge.response.star-star": 6,
    "gauge.response.root": 2,
    "gauge.response.metrics": 3,
    "classes": 5808,
    "classes.loaded": 5808,
    "classes.unloaded": 0,
    "heap": 3728384,
    "heap.committed": 986624,
    "heap.init": 262144,
    "heap.used": 52765,
    "nonheap": 0,
    "nonheap.committed": 77568,
    "nonheap.init": 2496,
    "nonheap.used": 75826,
    "mem": 986624,
    "mem.free": 933858,
    "processors": 8,
    "threads": 15,
    "threads.daemon": 11,
    "threads.peak": 15,
    "threads.totalStarted": 42,
    "uptime": 494836,
    "instance.uptime": 489782,
    "datasource.primary.active": 5,
    "datasource.primary.usage": 0.25
}
```


Pivotal

Spring Boot Actuator: metrics

- actuator supports recording of your own metrics by exposing a *CounterService* and *GaugeService* interfaces.
  - the *CounterService* exposes increment, decrement and reset methods
  - the *GaugeService* provides a submit method

```
@Service
public class TwitterService {

    private final CounterService counterService;

    @Autowired
    public TwitterService(CounterService counterService) {
        this.counterService = counterService;
    }

    public void search(Search query) {
        this.counterService.increment("services.twitter.search.invoked");

        // ...
    }

}
```

Spring Boot Actuator: metrics

- actuator supports custom metrics by registering additional *PublicMetrics* implementation bean(s)

- actuator also supports aggregation by using the *AggregateMetricReader* that can consolidate metrics from different physical sources

Agenda

- Spring Boot Actuator
- Metrics
- Health Indicators

Spring Boot Actuator: health indicators

- health information can be used to check the status of your running application
- spring boot actuator provides the */health* endpoint that shows the current health of your application
- spring boot actuator include a number of auto-configured health indicators and provides you an easy way to create a custom one

```
{
    "status" : "UP"
}
```

Spring Boot Actuator: health indicators

out-of-the-box health indicators:

- *CassandraHealthIndicator*
- *DiskSpaceHealthIndicator*
- *DataSourceHealthIndicator*
- *ElasticsearchHealthIndicator*
- *JmsHealthIndicator*
- *MailHealthIndicator*
- *MongoHealthIndicator*
- *RabbitHealthIndicator*
- *RedisHealthIndicator*
- *SolrHealthIndicator*

Spring Boot Actuator: health indicators

- actuator provides the *HealthIndicator* interface and the *AbstractHealthIndicator* class to create a custom health indicator

```
@Component
public class TwitterServiceHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        int errorCode = check(); // perform some specific health check
        if (errorCode != 0) {
            return Health.down().withDetail("Error Code", errorCode).build();
        }
        return Health.up().build();
    }

}
```

Demo

Actuator

Lab

Custom Metrics and HealthIndicator

Summary

- Spring Boot Actuator
- actuator exposes built-in endpoints
- allows recording of metrics: CounterService, GaugeService
- allows creation of custom metrics
- built-in health indicators
- allows creation of custom health indicators

A NEW PLATFORM FOR A NEW ERA

Security with Spring Boot

Spring Boot Developer

Securing Web Applications

Pivotal

Agenda

- Security with Spring Boot
- OAuth2 with Spring Boot

Security with Spring Boot

- spring boot uses the *spring-security* project to simplify the protection of applications
- to create secured spring boot applications it is necessary to add the *spring-boot-starter-security* dependency
- spring boot will auto-configure basic security by default

Security with Spring Boot

- spring boot will auto-configure a basic security by default and print out a *default security password* on application startup

- basic security is easy to override by using the *security.** properties

```
security.user.name = springone
security.user.password = workshop
security.user.role = USER
```

Security with Spring Boot

spring boot allows you to configure security programmatically by extending the *WebSecurityConfigurerAdapter* in-memory:

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
            .withUser("springone").password("workshop").roles("USER")
            .and()
            .withUser("admin").password("password").roles("ADMIN");

    }

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

Security with Spring Boot

jdbc:

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private DataSource dataSource;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .jdbcAuthentication().dataSource(this.dataSource);
    }

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

Security with Spring Boot

- disable security by setting the *security.basic.enabled=false* property
- SSL can be configured with the *server.ssl.** properties
- you can connect a persistence mechanism to hold user information for authentication and authorization by implementing *UserDetailsService*

Demo

Security a Web App

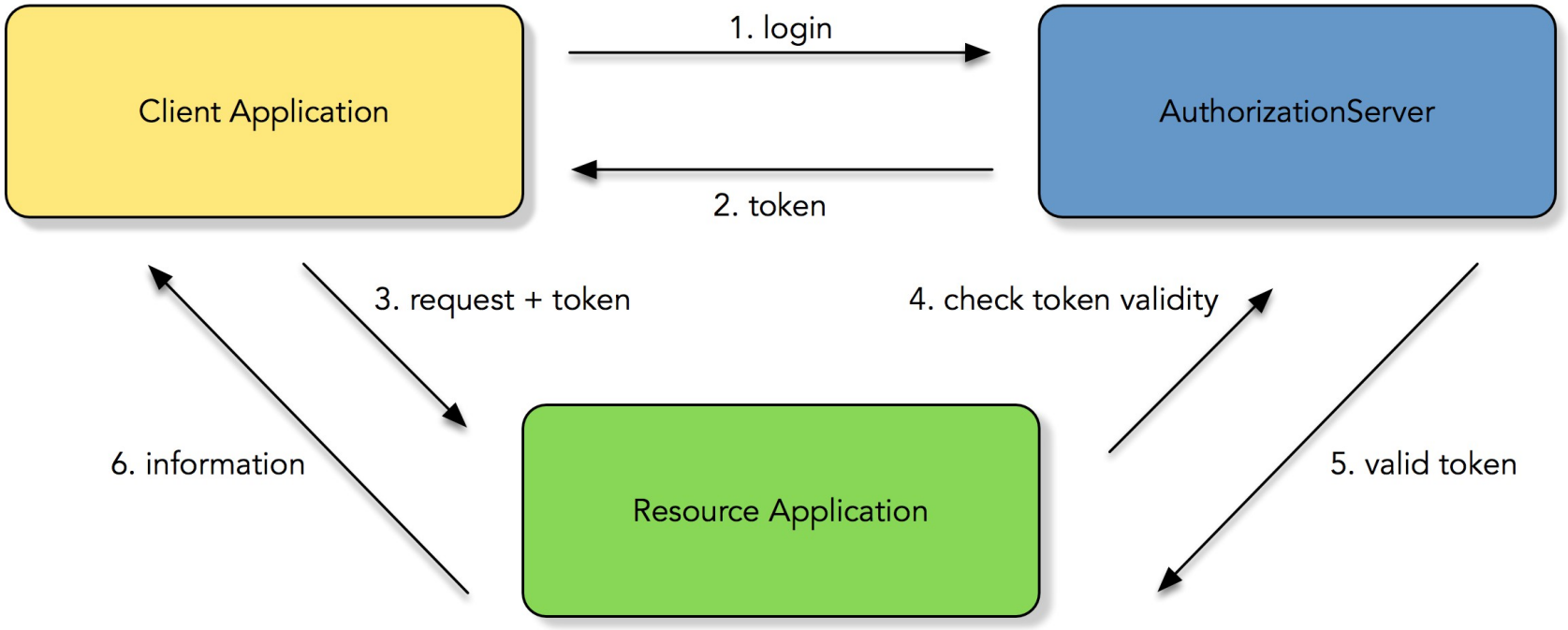
Lab

Jdbc Security

Agenda

- Security with Spring Boot
- OAuth2 with Spring Boot

OAuth2 with Spring Boot



OAuth2 with Spring Boot

- spring boot uses the spring security oauth to support OAuth(1a) and OAuth2
- it brings support for OAuth providers and consumers
- spring boot applications with OAuth2 can be created by adding the *org.springframework.security.oauth:spring-security-oauth2* dependency

OAuth2 with Spring Boot

to create an authorization server and grant access tokens, you can use the *@EnableAuthorizationServer*:

- */oauth/authorize* endpoint to service requests for authorization
- */oauth/token* endpoint to service request for access tokens

OAuth2 with Spring Boot

to serve resources that are protected by the OAuth2 token, you can use the *@EnableResourceServer* annotation

- this annotation adds a filter automatically to the spring security filter chain.
- you can configure: *tokenServices*, *resourceId*, request matchers, access rules, and fully customizable by the *HttpSecurity* configurer

OAuth2 with Spring Boot

- to access secured resources by OAuth2, you can use the *@EnableAOuth2Client* annotation

- you can use the *OAuth2RestTemplate* and the *OAuth2ClientContext* to manage all the access to the secured resources

Lab

Security with OAuth2

Summary

- Security with Spring Boot
- spring boot uses spring-security project for securing applications
- include spring-boot-starter-security for basic security
- highly customizable: in-memory, jdbc, ldap
- spring-security-oauth2 project provides @EnableAuthorizationServer, @EnableResourceServer, @EnableOAuth2Client

A NEW PLATFORM FOR A NEW ERA

Messaging with Spring Boot

Spring Boot Developer

RabbitMQ

Pivotal

Agenda

- Spring Messaging
- Spring Boot Messaging with RabbitMQ
  - quick overview
  - exchanges, bindings and queues
  - sending messages
  - consuming messages

Spring Messaging

- the spring framework provides extensive support for integrating with messaging systems: from simplified use of the JMS API using *JmsTemplate* to a complete infrastructure to receive messages asynchronously

- the *spring-amqp* project provides a similar feature set for the '*advanced message queuing protocol*' providing a *RabbitTemplate* class for sending and receiving message plus some useful annotations

- there is also support for stomp messaging natively in spring, websockets and kafka

Agenda

- Spring Messaging
- Spring Boot Messaging with RabbitMQ
  - quick overview
  - exchanges, bindings and queues
  - sending messages
  - consuming messages

Spring Boot Messaging with RabbitMQ

- *spring-amqp* provides the *@EnableRabbit* that scans for annotations like *@RabbitListener* and *@SendTo*, for listening and reply

- spring boot uses the power of spring messaging by adding several auto-configuration options for *RabbitTemplate* and defaults for *ConnectionFactory* classes

- spring boot defaults can be controlled by external configuration properties in *spring.rabbitmq.**

Spring Boot Messaging with RabbitMQ

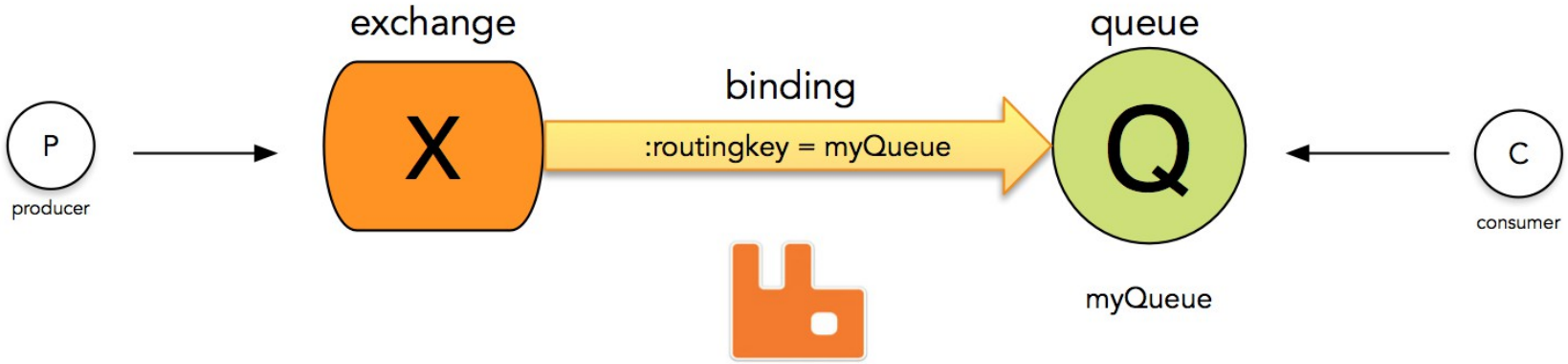
- spring boot messaging applications with RabbitMQ can be created by adding the *spring-boot-starter-amqp* dependency

RabbitMQ: overview

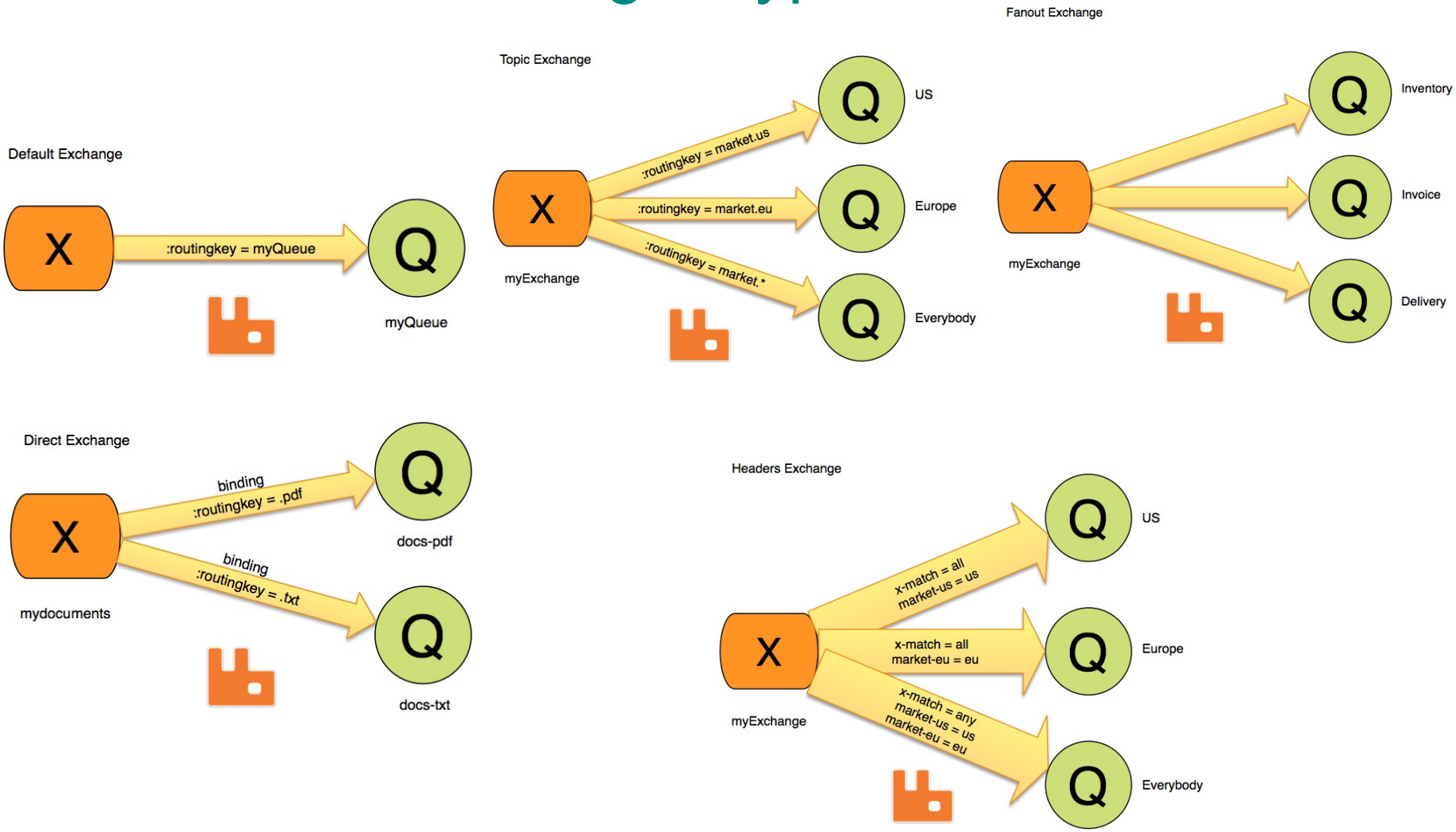
- rabbitmq is an amqp message broker
- platform agnostic, broadly applicable for enterprise, totally open source
- implemented with *erlang*
- distributed: cluster ready, reliability/scalability out of the box
- high availability: mirror queues, data/state replication with full ACID, routing capabilities
- multiple protocol support: *amqp*, *mqtt*, *stomp*, *smtp*, *xmpp*
- security: ssl, ldap
- plugin based: federation, shovel, consistent hash, sharding, ...
- multiple client libraries: java, .net, ruby, erlang, python, php, ...

RabbitMQ: overview

RabbitMQ: exchanges, bindings, queues



RabbitMQ: exchanges types



RabbitMQ: sending messages

```
@EnableScheduling
@SpringBootApplication
public class ProducerApplication {

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

    @Autowired
    private RabbitTemplate template;

    @Scheduled(fixedDelay = 1000)
    public void sender() {
        this.template.convertAndSend("spring-boot","Hello World at " + (new Date()));
    }
}
```

RabbitMQ: consuming messages

```
@SpringBootApplication
public class ConsumerApplication {

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

    @RabbitListener(queues = "spring-boot")
    public void receiveMessage(String message) {
        System.out.println("Received: " + message);
    }
}
```

Demo

RabbitMQ

Lab

Messaging using RabbitMQ

Summary

- spring boot simplifies messaging by providing multiple auto-configuration options for jms, amqp, websockets (stomp) and kafka

A NEW PLATFORM FOR A NEW ERA

Microservices with Spring Boot

Spring Boot Developer

Deploying Microservices to Pivotal Cloud Foundry

Pivotal

Agenda

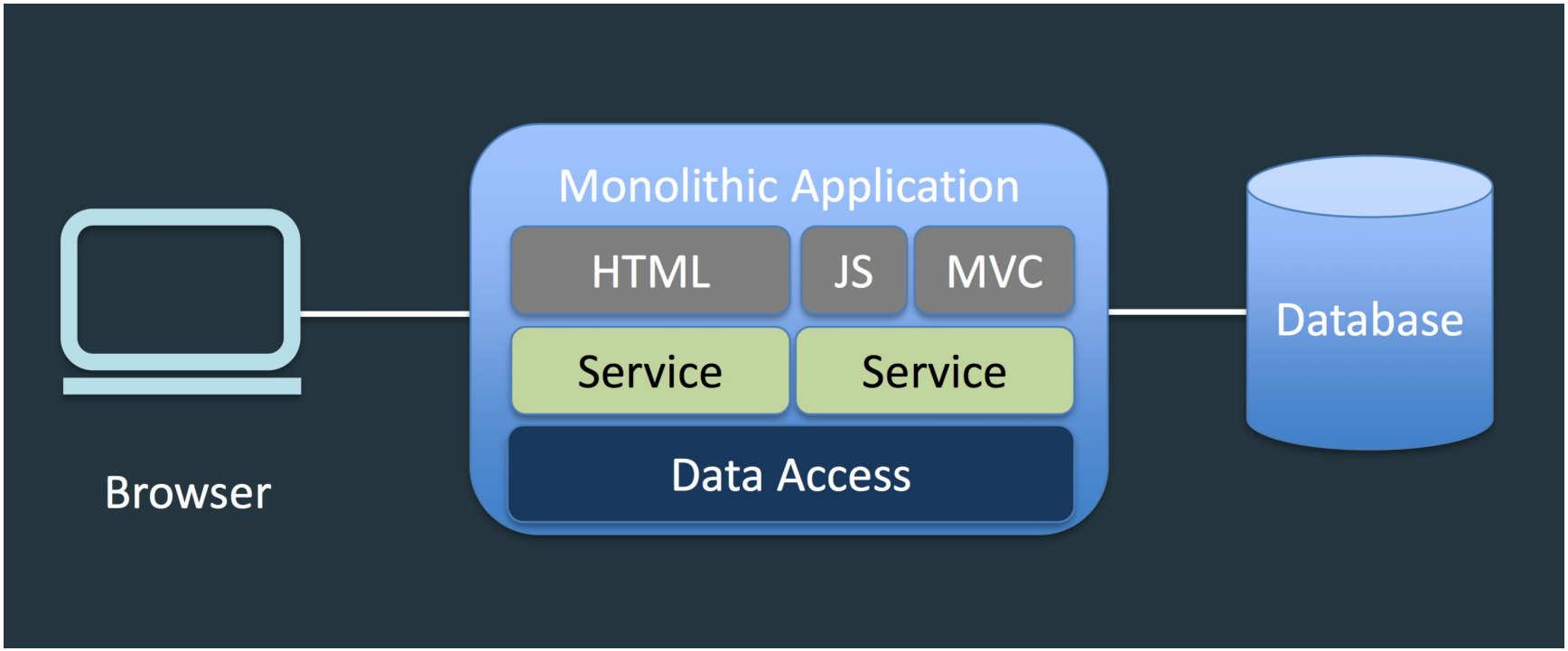
- Microservices
- Cloud Foundry
- Spring Boot in the Cloud

Microservices

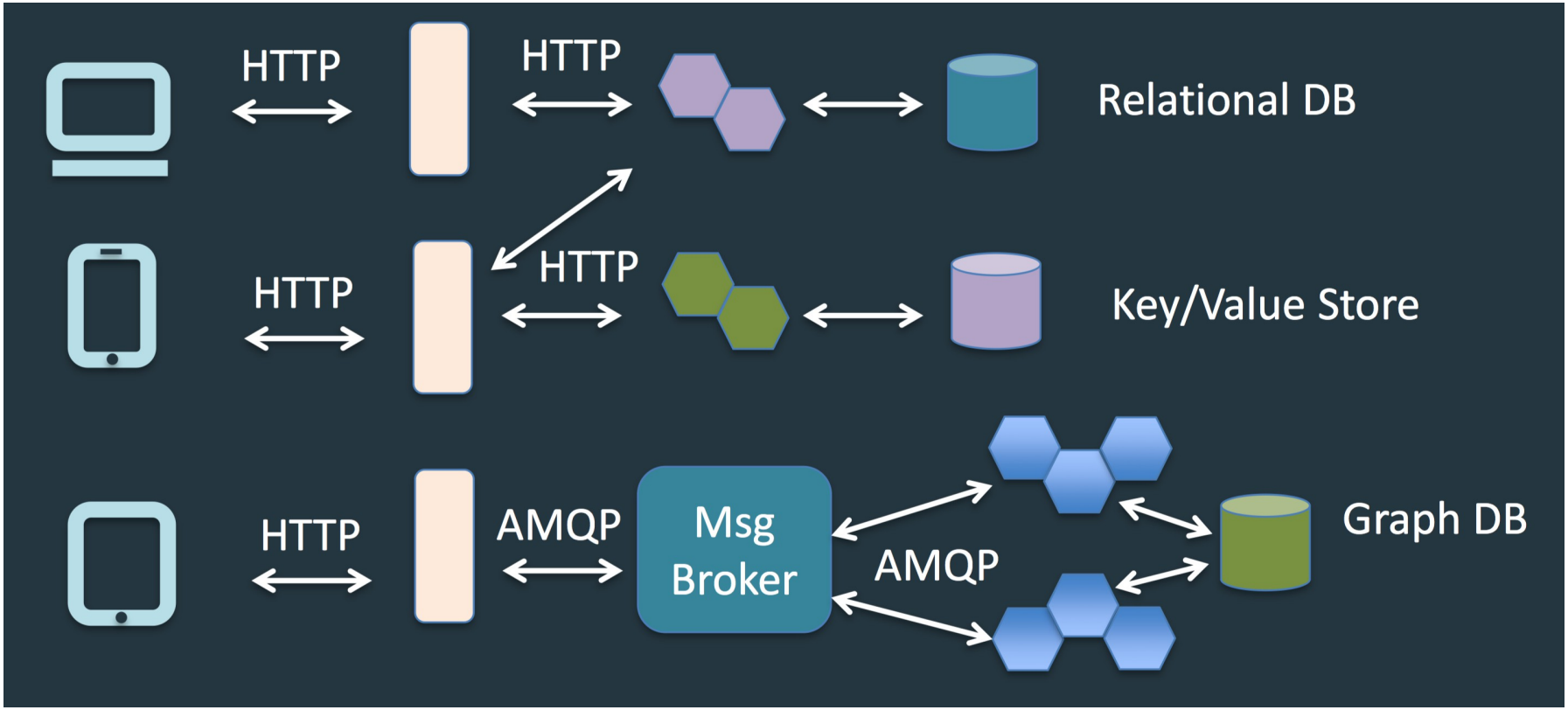
- microservices is not a new word
  - term coined in 2005 by Dr. Peter Rodgers
  - back then called: micro web services, based on SOAP

microservices are loosely-couple services with bounded-contexts that perform a single well-defined function

Microservices: Monolith



Microservices



Microservices

benefits of microservices:

- smaller code base, easy to maintain
- easy to scale, independent deployment
- technology diversity
- fault isolation
  - component failure unlikely to bring down the whole system
- better support for parallel teams

Microservices

microservices features:

- API (contracts) interaction only
  - loosely coupled
  - RESTful APIs
- bounded-context / domain-driven-design
  - single view of data
- polyglot persistence and development
- easy to scale, independent deployment

Microservices

tradeoffs

- monolith:
  - easier to build at first
  - more complex to enhance and maintain

- microservices:
  - harder to build at first
  - simpler to extend, enhance and maintain
  - scaling out (more processes) easier
  - many more moving parts to manage

Agenda

- Microservices
- Cloud Foundry
- Spring Boot in the Cloud

Cloud Foundry

why a platform?

- deploying distributed systems is complicated
  - security, resilience, redundancy, load-balancing

- a platform provides the necessary tools:
  - natural fit for deploying a microservices-based system
  - applications instances are the unit of deployment
  - can be started, stopped and restarted independently on-demand
  - provide dynamic load-balancing, scaling and routing

Agenda

- Microservices
- Cloud Foundry
- Spring Boot in the Cloud

Spring Boot in the Cloud

spring boot can easily be deploy to cloud foundry, just by executing:

cf push my-spring-boot-app.jar

once deployed multiple microservices issues that now arise:

- how do they find each other?
- how do we decide with instance to use?
- what happens if a microservice is not responding?
- how do we control access?
- how do they communicate?

Lab

Deploying Microservices to Pivotal Cloud Foundry

Summary

- Microservices / Cloud Foundry
- monolith vs. microservices
- tradeoffs
- cloud foundry
- spring boot in the cloud

Pivotal

A NEW PLATFORM FOR A NEW ERA

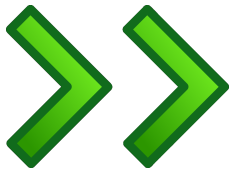
Finishing Up

Course Completed

What's next?

What's Next

- Congratulations, we've finished the course
- What to do next?
  - Certification
  - Other courses
  - Resources
  - Evaluation
- Check-out optional sections on ...



Certification

- Computer-based exam
  - 50 multiple-choice questions
  - 90 minutes
  - Passing score: 76% (38 questions answered successfully)

- Preparation
  - Review all the slides
  - Redo the labs

Certification: Questions

Pivotal CERTIFIED Spring Professional

Typical question

- Statements
  - a. An application context holds Spring beans
  - b. An application context manages bean scope
  - c. Spring provides many types of application context

- Pick the correct response:
  1. Only a. is correct
  2. Both a. and c. are correct
  3. All are correct
  4. None are correct

Certification: Logistics

Voucher is valid for 3 months – *do it soon!*

- Where?
  - Online at PSI (Innovative Exams)
    - https://www.examslocal.com
- How?
  - You should receive a certification voucher by email
  - Register/sign-in and book an exam using the voucher
    - http://it.psionline.com/exam-faqs/pivotal-faq
  - Take the test from *any* location
- For more information, email
  - education@pivotal.io

Other courses

- Many courses available
  - Core Spring
  - Web Applications with Spring
  - Enterprise Spring
  - Spring Boot
  - Spring Cloud Services
  - Pivotal Cloud Foundry
  - Gemfire, Rabbit MQ …
- More details here:
  - http://www.pivotal.io/training

Core Spring

Pivotal CERTIFIED Spring Professional

- Four day course covering
  - Application configuration using
    Java Configuration, XML and/or Annotations
  - How Spring works internally and makes use of Aspect Oriented Programming
  - Data persistence using JDBC and JPA
  - Declarative Transaction Management
  - Introduction to web-applications and Spring MVC
  - Building RESTful Servers
  - Spring Boot, Spring Cloud and Microservices

Spring Web

Pivotal CERTIFIED Spring Web Application Developer

- 4-day workshop
- Making the most of Spring in the web layer
  - Spring MVC
  - Spring Web Flow
  - REST using MVC and AJAX
  - Security of Web applications
  - Performance testing
- Spring Web Application Developer certification

Enterprise Spring

Pivotal CERTIFIED Spring Enterprise Integration Specialist

- Building loosely coupled event-driven architectures
  - Separate processing, communications & integration
- 4 day course covering
  - Tasks, Scheduling and Concurrency
  - Advanced transaction management
  - REST Web Services with Spring MVC
  - Spring Batch
  - Spring Integration
  - Data Ingestion, Transformation and Extractions

Spring Boot Developer

- 2 day workshop
  - Introduction to Spring Boot
  - Building Web and REST Applications
  - Integrating Data Management
  - Using Actuators, Health Monitoring
  - Security and OAuth2
  - Messaging using RabbitMQ
  - Deployment

Spring Cloud Services
Microservices With Spring

- 2 day course
  - Introduction to Spring Boot
    - Underpins all Spring Cloud projects
  - Pushing Applications to a PaaS
    - Using Pivotal Cloud Foundry
  - What are Microservices?
    - Architecting a microservices solution
  - Cloud infrastructure services and Netflix OSS
    - Service Configuration
    - Service Registration
    - Load-balancing and fault tollerence

Pivotal

Cloud Foundry Developer

CLOUDFOUNDRY

- 3 day course covering
  - Application deployment to Cloud Foundry
    - Typically these are Web and/or REST applications
    - Deployment using cf tool or an IDE
  - Cloud Foundry Concepts
    - Logging, Continuous Integration, Monitoring
    - Accessing and defining Services
    - Using and customizing Buildpacks
  - Design considerations: “12 Factor”
    - JVM application specifics, using Spring Cloud

Formerly: Developing Applications with Cloud Foundry

Cloud Foundry Administrator

CLOUDFOUNDRY

- 3 day course covering
  - Administration
    - Deploying Cloud Foundry to vSphere or AWS
    - Configuring and Managing Cloud Foundry
    - Working with BOSH
  - Application deployment to Cloud Foundry
    - Includes the basic topics from the Developer course
    - Logging, Continuous Integration, Monitoring
    - Design considerations: “12 Factor” Applications

Broader course than Developer with administrator emphasis

Pivotal

Pivotal Support Offerings

Pivotal Support

- Global organization provides 24x7 support
  - How to Register: http://tinyurl.com/piv-support

- Premium and Developer support offerings:
  - http://www.pivotal.io/support/offerings
  - http://www.pivotal.io/support/oss
  - Both Pivotal App Suite *and* Open Source products

- Support Portal: https://support.pivotal.io
  - Community forums, Knowledge Base, Product documents

Pivotal Consulting

- Custom consulting engagement?
  - Contact us to arrange it
    - http://www.pivotal.io/contact/spring-support
    - Even if you don't have a support contract!

- Pivotal Labs
  - Agile development experts
  - Assist with design, development and product management
    - http://www.pivotal.io/agile
    - http://pivotallabs.com

Resources

- The Spring reference documentation
  - http://spring.io/docs
  - Already 800+ pages!
- The official technical blog
  - http://spring.io/blog
- Stack Overflow – Active Spring Forums
  - http://stackoverflow.com

Resources (2)

- You can register issues on our Jira repository
  - https://jira.spring.io
- The source code is available here
  - https://github.com/spring-projects/spring-framework
- Follow Spring development
  - https://fisheye.springsource.org/browse/

Thank You!

- We hope you enjoyed the course

- Please fill out the evaluation form
  - Americas: http://tinyurl.com/usa-eval
  - EMEA: http://tinyurl.com/emea-eval
  - Asia-Pac: http://tinyurl.com/apj-eval
- Once you've done, login to *Pivotal Academy*
  - You can download your Attendance Certificate

If your course is registered at Pivotal Academy

Don't forget the optional sections