Microservices with Spring

Three Day Workshop

Building Cloud Native Applications using Spring

Version 1.0

Pivotal

© Copyright 2016 Pivotal. All rights reserved

Copyright Notice

Copyright © 2016 Pivotal Software, Inc. All rights reserved. This manual and its accompanying materials are protected by U.S. and international copyright and intellectual property laws.

Pivotal products are covered by one or more patents listed at http://www.pivotal.io/patents.

Pivotal is a registered trademark or trademark of Pivotal Software, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies. The training material is provided “as is,” and all express or implied conditions, representations, and warranties, including any implied warranty of merchantability, fitness for a particular purpose or noninfringement, are disclaimed, even if Pivotal Software, Inc., has been advised of the possibility of such claims. This training material is designed to support an instructor-led training course and is intended to be used for reference purposes in conjunction with the instructor-led training course. The training material is not a standalone training tool. Use of the training material for self-study without class attendance is not recommended.

These materials and the computer programs to which it relates are the property of, and embody trade secrets and confidential information proprietary to, Pivotal Software, Inc., and may not be reproduced, copied, disclosed, transferred, adapted or modified without the express written approval of Pivotal Software, Inc.

Microservices with Spring

A hands-on workshop using Spring and Cloud Foundry to create well-designed, testable, cloud-native, applications

Pivotal

© Copyright 2016 Pivotal. All rights reserved

Logistics

- Participants list
- Self introduction
- Course registration
- Courseware
- Internet access
- Phones on silent

Working hours
Lunch and breaks
Toilets/Restrooms
Fire alarms
Emergency exits
Other questions?

© Copyright 2016 Pivotal. All rights reserved

How You will Benefit

- Learn to use Spring Boot and Spring Cloud for cloud native and other applications
- Gain hands-on experience
  - 20/80 presentation and labs
- Access to Pivotal and Spring professionals

© Copyright 2016 Pivotal. All rights reserved

Covered in this section

- **Agenda**
- Setup
- Spring and Pivotal

© Copyright 2016 Pivotal. All rights reserved

Course Agenda: Day 1

- Cloud Native Architecture Overview
- Introduction to Spring Boot
- Deploying to a PaaS using Cloud Foundry
- Spring REST
- Spring Cloud Connectors

© Copyright 2016 Pivotal. All rights reserved

Course Agenda: Day 2

- Splitting the Monolith
- Twelve Factor Design Principles
- Microservices with Polyglot Persistence
- Cloud Native Architecture Patterns – 1
  - Shared Configuration
  - Service Discovery with Eureka

© Copyright 2016 Pivotal. All rights reserved

Course Agenda: Day 3

- Cloud Native Architecture Patterns – 2
  - Load Balancing
  - Fault-Tolerance
  - REST Clients using Feign
- Securing Cloud-Native Applications
  - Creating an OAuth Server
  - Spring Cloud Security

© Copyright 2016 Pivotal. All rights reserved

Covered in this section

- Agenda
- **Setup**
- Spring and Pivotal

© Copyright 2016 Pivotal. All rights reserved

Pre-Course Setup

The following software is required to complete this course:

- Java JDK 8
- An IDE or Editor of your choice
- If *not* using an IDE, the Maven utility: **`mvn`**
- Cloud Foundry CLI `cf`
- Curl (MS Windows users only)
- *Optional:* Browser plugin for showing*JSON Data*
- Account on PWS or in-house PCF installation

*Ideally you should have received information on how to set these up **prior** to the course*

© Copyright 2016 Pivotal. All rights reserved

Covered in this section

- Agenda
- Setup
- **Spring and Pivotal**

© Copyright 2016 Pivotal. All rights reserved

Spring and Pivotal

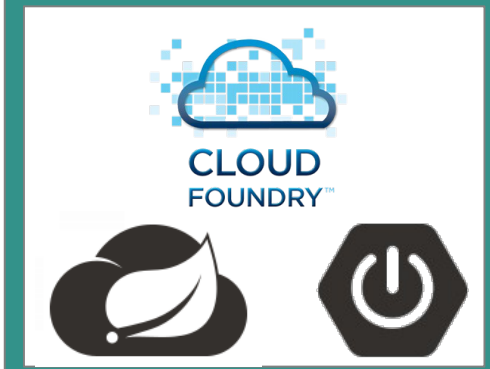
- SpringSource, the company behind Spring
  - acquired by VMware in 2009
  - transferred to Pivotal joint venture 2013
- Spring projects key to Pivotal's strategies
  - Virtualize your Java Apps
    - Save license costs
    - Deploy to private, public, hybrid clouds
    - *Cloud Native Microservice applications*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

© Copyright 2016 Pivotal. All rights reserved

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*

© Copyright 2016 Pivotal. All rights reserved

Spring Projects

© Copyright 2016 Pivotal. All rights reserved

Covered in this section

- Agenda
- Spring and Pivotal

Let's get on with the course..!

© Copyright 2016 Pivotal. All rights reserved

Spring Boot

A new way to create Spring Applications

© Copyright 2016 Pivotal. All rights reserved

Hello World example

- Only 3 files to get a running Spring application

pom.xml

Setup Spring Boot dependencies

HelloController

Basic Spring MVC controller

Application class

Application launcher

© Copyright 2016 Pivotal. All rights reserved

Hello World – Maven descriptor

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.0.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

parent

Spring MVC
Embedded Tomcat
Jackson...

pom.xml

Maven is just one option. You can also use Gradle or Ant/Ivy

© Copyright 2016 Pivotal. All rights reserved

Hello World – Spring MVC controller

- A RESTful controller to keep this example simple
  - Returns a String as the body of the HTTP Response
  - No view involved

```
@RestController
public class HelloController {
    @RequestMapping("/")
    public String hello() {
        return "Greetings from Spring Boot!";
    }
}
```

No separate View file to keep things simple

Controller.java

© Copyright 2016 Pivotal. All rights reserved

Hello World – Application Class

- *@SpringBootApplication* annotation enables Spring Boot
  - Runs Tomcat *embedded*

```
@SpringBootApplication
public class Application {

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

}
```

application.java

Main method will be used to run the packaged application from the command line – *old style!*

© Copyright 2016 Pivotal. All rights reserved

Putting it all together

```
mvn package
```

Maven command to generate an archive file

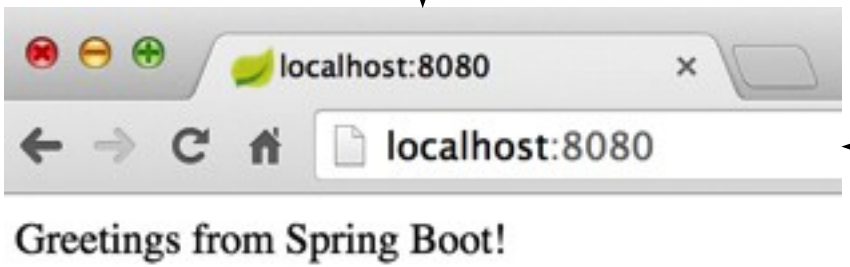
```
helloApp-0.0.1-SNAPSHOT.jar
```

Generated file

generated file

```
java -jar helloApp-0.0.1-SNAPSHOT.jar
```

App started in command line



App runs on port 8080

© Copyright 2016 Pivotal. All rights reserved

Core Spring *"Starter"* Dependencies

- Allow an easy way to bring in multiple coordinated dependencies
  - Including *"Transitive"* Dependencies

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

Resolves ~ 16 JARs!

spring-boot-.jar*
spring-core-.jar*
spring-context-.jar*
spring-aop-.jar*
spring-beans-.jar*
aopalliance-.jar*
...

Version not needed!
Defined by parent.

© Copyright 2016 Pivotal. All rights reserved

Jetty Support

- Jetty can be used instead of Tomcat

```
<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>
```

Excludes Tomcat

Adds Jetty

Jetty is automatically detected and used!

© Copyright 2016 Pivotal. All rights reserved

Spring Boot @EnableAutoConfiguration

- @EnableAutoConfiguration annotation on a Spring Java configuration class
  - Causes Spring Boot to automatically create beans it thinks you need
  - Usually based on classpath contents, can easily override

```
@Configuration
@EnableAutoConfiguration
public class MyAppConfig {
    public static void main(String[] args) {
        SpringApplication.run(MyAppConfig.class, args);
    }
}
```

Spring Boot class

© Copyright 2016 Pivotal. All rights reserved

Spring Boot as a Runtime

- Spring Boot can startup an *embedded* web server
  - You can run a web application from a JAR file!
  - Tomcat included in Web Starter

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

Simpler for testing and recommended when deploying *Cloud Native* applications

© Copyright 2016 Pivotal. All rights reserved

Controlling Logging Level

- Boot can control the logging level
- Just set the logging level in application.properties
- Works with most logging frameworks
  - Java Util Logging, Logback, Log4J, Log4J2

```
logging.level.org.springframework=DEBUG
logging.level.com.acme.your.code=INFO
```

Try to stick to SLF4J in the application.
The *advanced* section covers how to change the logging framework

© Copyright 2016 Pivotal. All rights reserved

Externalized Properties
application.properties

- Developers commonly externalize properties to files
  - Easily consumable via Spring PropertySource
- But developers name / locate their files different ways
- Spring Boot will automatically look for application.properties in the classpath root

```
database.host=localhost
database.user=admin
```

application.properties

- Starter POMs declare the properties to use
  - Check the reference documentation to know which properties can be used

© Copyright 2016 Pivotal. All rights reserved

DataSource Configuration

- Use either spring-boot-starter-jdbc or spring-boot-starter-data-jpa and include a JDBC driver on classpath
- Declare properties

application.properties

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

- That's It!
  - Spring Boot will create a DataSource with properties set
  - Will even use a connection pool if the library is found on the classpath!

© Copyright 2016 Pivotal. All rights reserved

Spring Boot Actuator

- Monitor and manage applications when pushed to production
- Actuator HTTP endpoints are only available with a Spring MVC-based application

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


© Copyright 2016 Pivotal. All rights reserved

Spring Boot Actuator: Endpoints

- Actuator endpoints allow you to monitor and interact with your application

Endpoints	**Description**
/beans	Displays a complete list of all the Spring beans in your application.
/autoconfig	Displays an auto-configuration report showing all auto-configuration candidates and the reason why they 'were' or 'were not' applied.
/env	Exposes properties from Spring's ConfigurableEnvironment.
/health	Shows application health information

© Copyright 2016 Pivotal. All rights reserved

Summary

- Spring Boot speeds up Spring application development
- You always have full control and insight
- Nothing is generated
- No special runtime requirements
- No servlet container needed (if you want)
  - E.g. ideal for microservices

© Copyright 2016 Pivotal. All rights reserved

Lab

Simplification using Spring Boot

© Copyright 2016 Pivotal. All rights reserved

Getting Started with Cloud Foundry

Deploying your First Application

Setup, Deploy and Manage

Pivotal

© Copyright 2016 Pivotal. All rights reserved

Roadmap

- **Getting Started with the Command Line Interface**
- Login
- Deploying an Application
- Using a Manifest
- The Application Manager

© Copyright 2016 Pivotal. All rights reserved

The Command Line Interface

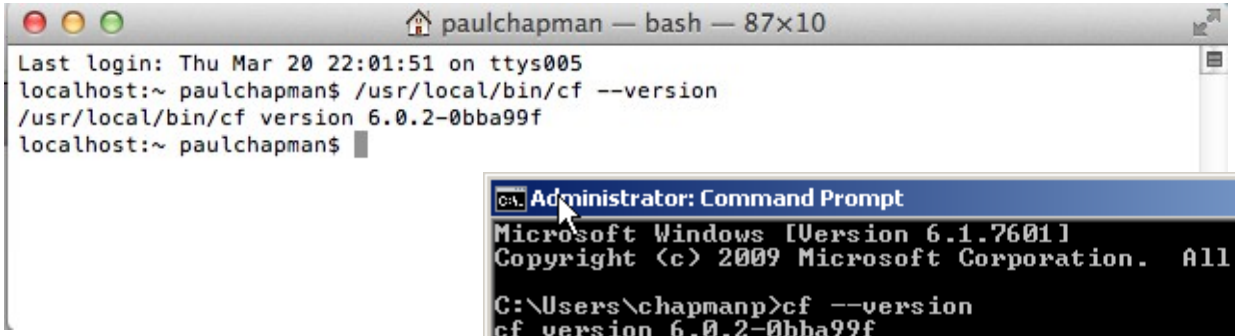
- Several interface options exist for Cloud Foundry
  - Command Line Interface (CLI)
  - Web-based *Application Manager* Console
  - Eclipse / STS plugin, IntelliJ plugin*
- Primary access is done via the CLI
  - Make sure you have it installed
    - Installation was covered in the “Setup” guide

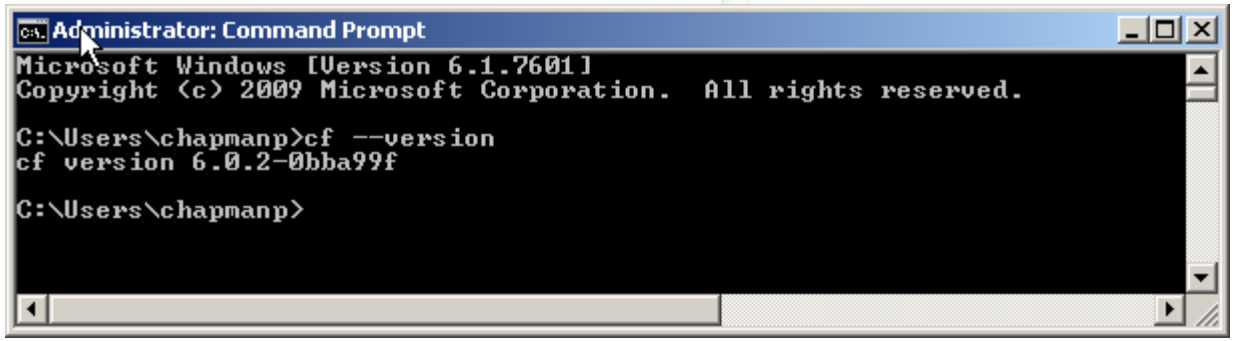
* Not discussed in this course

© Copyright 2016 Pivotal. All rights reserved

Test the CLI Utility

- It is called cf
  - Open a Command/Shell window
  - At the prompt type: `cf --version`





MS Windows: To open a Cmd window, press + R, type CMD, hit enter

© Copyright 2016 Pivotal. All rights reserved

Getting Help

- Get help at any time via **`cf help`**
- Or **`cf help <command>`**

© Copyright 2016 Pivotal. All rights reserved

DO NOW – Get Help on a Command

- Perform these steps on your computer:
  - Open a command prompt
  - Issue the **`cf help`** command
  - Get help on the login command: **`cf help login`**

- Answer these questions:
  - What option do you use to specify username?
  - Is specifying the password option encouraged?

© Copyright 2016 Pivotal. All rights reserved

Roadmap

- Getting Started with the Command Line Interface
- **Login**
- Deploying an Application
- Managing Application Instances

© Copyright 2016 Pivotal. All rights reserved

Login to Cloud Foundry

- Need to tell cf
  - What cloud foundry instance you are using
  - What your account details are
  - Use **`cf login`**

Color highlighting
MacOS, Linux only

```
> cf login -a api.run.pivotal.io -u <username>
API endpoint: api.run.pivotal.io
Authenticating...
OK

Targeted org Cloud Foundry Course

Targeted space development

API endpoint: https://api.run.pivotal.io (API version: 2.0.0)
User:         qzqz2020@yahoo.com.au
Org:          Cloud Foundry Course
Space:        development
```

Will prompt for anything you don't specify
No -p? Prompts for password

© Copyright 2016 Pivotal. All rights reserved

Cloud Foundry URLs

- To access CF you need to know 3 URLs
  - *API Endpoint*
    - Identifies your CF instance
    - Used to deploy applications, manage spaces, routes …
    - The **`cf`** utility makes *RESTful* requests to this URL
      - Actually to the Cloud Controller
  - Apps Manager
    - Application management dashboard (console)
    - *Pivotal CF only*
  - Apps Domain
    - Used to access deployed applications
    - *May* be same as System Domain

© Copyright 2016 Pivotal. All rights reserved

Cloud Foundry URLs

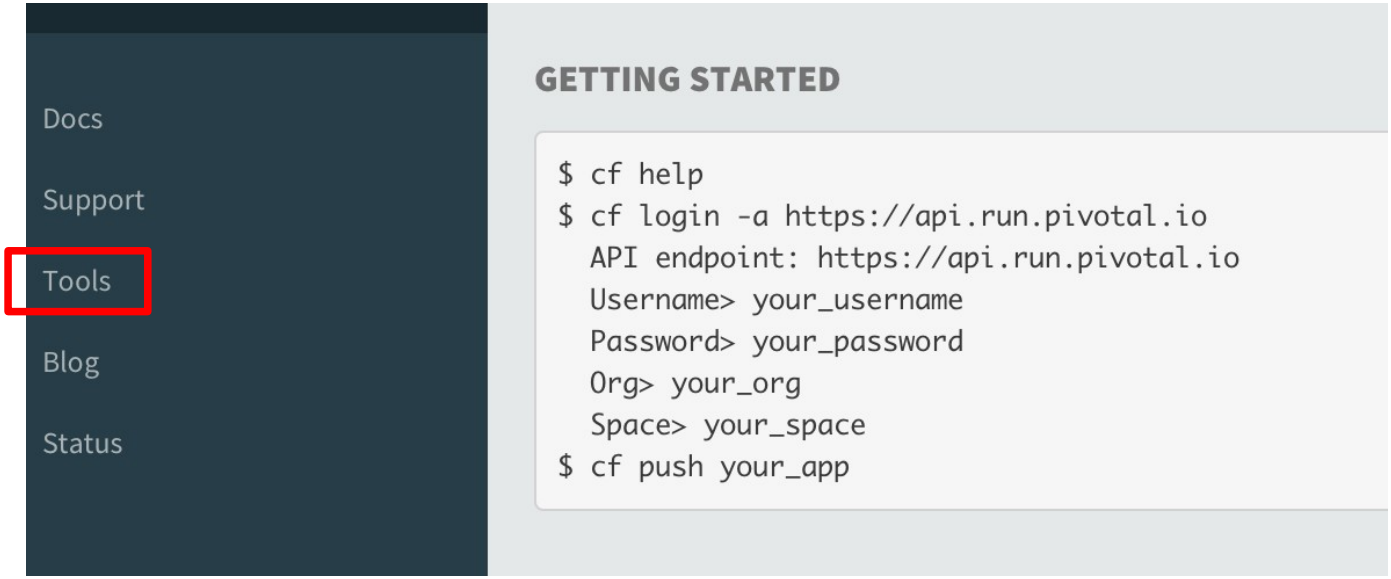
For simplicity, most examples in this section show PWS URLs

- System & App domains defined when CF was installed
- *If using PWS*
  - System domain: `run.pivotal.io`
    - API Endpoint: `api.run.pivotal.io`
    - Apps Manager: `console.run.pivotal.io`
  - Apps domain: `cfapps.io`
- *Your own CF installation*
  - System domain: `<your-cf-system-domain>`
    - API Endpoint: `api.<your-cf-system-domain>`
    - Apps Manager: `console.<your-cf-system.domain>`
  - Apps domain: `<your-cf-apps-domain>`

© Copyright 2016 Pivotal. All rights reserved

Finding the API Endpoint URL

- URL of Cloud Controller in your Cloud Foundry instance
  - On Apps Manager home-page on first login
  - Or click *Tools*
  - Shows how to run **`cf login`**, including the API Endpoint



© Copyright 2016 Pivotal. All rights reserved

DO NOW – Login

- Perform these steps on your computer:
  - Login with **`cf login`** command
    - Specify CF instance using **`-a <API-URL>`**
      - For PWS: **`-a api.run.pivotal.io`**
    - Specify email / password
    - If prompted, select an organization and space

```
$> cf login -a <API-URL> -u <your-email-or-username>
API endpoint: api.run.pivotal.io
...
```

    - Firewall issues?

      http://docs.cloudfoundry.org/devguide/installcf/http-proxy.html

© Copyright 2016 Pivotal. All rights reserved

The .cf Directory

- **`cf`** creates a ***.cf*** directory in your *home* directory
  - Stores context, logs, crash reports …
  - Remembers your CF API Endpoint
    - Don't need to specify **`-a`** option at next login

```
localhost:dev$ ls -l ~/.cf
total 48
-rw-------  1 paulchapman  staff   2491 15 Aug 14:06 config.json
-rw-r--r--  1 paulchapman  staff  11737 29 Nov  2013 crash
drwxr-xr-x  3 paulchapman  staff    102 12 Sep  2013 logs
-rw-r--r--  1 paulchapman  staff     26 12 Sep  2013 target
-rw-r--r--  1 paulchapman  staff   2084 29 Nov  2013 tokens.yml
```

You can override this location by setting CF_HOME

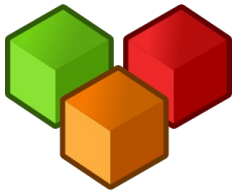
© Copyright 2016 Pivotal. All rights reserved

DO NOW – .cf folder

- Perform these steps on your computer:
  - Find the **`.cf`** folder / directory on your computer
    - You won't (yet) have all the files shown on previous slide
  - Open the **`config.json`** file, observe the contents

© Copyright 2016 Pivotal. All rights reserved

Organizations and Spaces



- Cloud Foundry is designed for collaboration
  - Developers in the same organization can work together
  - An *org* can be a department, project, company …
  - Every PWS user is put in their *own* organization
  - Org owners can invite other users to join

- Applications execute within a space
  - Spaces represent a deployment environment
  - Typically a stage in the development lifecycle
    - Development, Test, QA, Production ...

© Copyright 2016 Pivotal. All rights reserved

Current Targets

- When you first login you see output like this:
  - Notice it shows *current* organization and space
  - At any time, run cf target to get same information

```
API endpoint: https://api.run.pivotal.io (API version: 2.6.0)
User:         pchapman@pivotal.io
Org:          pivotaledu
Space:        development
```

- By default a PWS organization only has one space
  - *development*
  - **Recall:** On PWS you are setup as your *own* organization

© Copyright 2016 Pivotal. All rights reserved

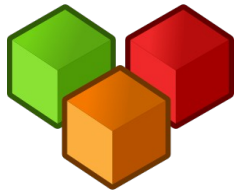
Viewing Organization

- Commands
  - **`cf orgs`** — All orgs for current user
  - **`cf org <org-name>`** — Shows specified org

```
>$ cf org pivotaledu
Getting info for org myorg as user@somedomain.com
OK

pivotaledu:
    domains:          cfapps.io
    quota:            paid (10240M memory limit, Unlimited instance
                        memory limit, 1000 routes, -1 services,
                        paid services allowed)
    spaces:           development, production, staging
    space quotas:
>$
```


© Copyright 2016 Pivotal. All rights reserved

Managing Spaces



- To see all the spaces in an organization
  - **`cf spaces`**
- Create a *new* space (in current organization by default)
  - **`cf create-space <space-name>`**
  - **`cf create-space <space-name> -o <org-name>`**
- Use *target* command to *change* space (or organization)
  - **`cf target -s <space-name>`**
  - **`cf target -o <org-name>`**

© Copyright 2016 Pivotal. All rights reserved

Roadmap

- Get Setup
- Login
- **Deploying an Application**
- Using a Manifest
- The Application Manager

© Copyright 2016 Pivotal. All rights reserved

Deploy Using the CLI

- You need a deployable application
  - For example with Java: a jar or war
    - Ant, Maven or Gradle build-tools can make it for us
    - Cloud Foundry doesn't care how you build your application
  - Other languages (Ruby, Node.js, etc.): the source will do

© Copyright 2016 Pivotal. All rights reserved

The cf push Philosophy

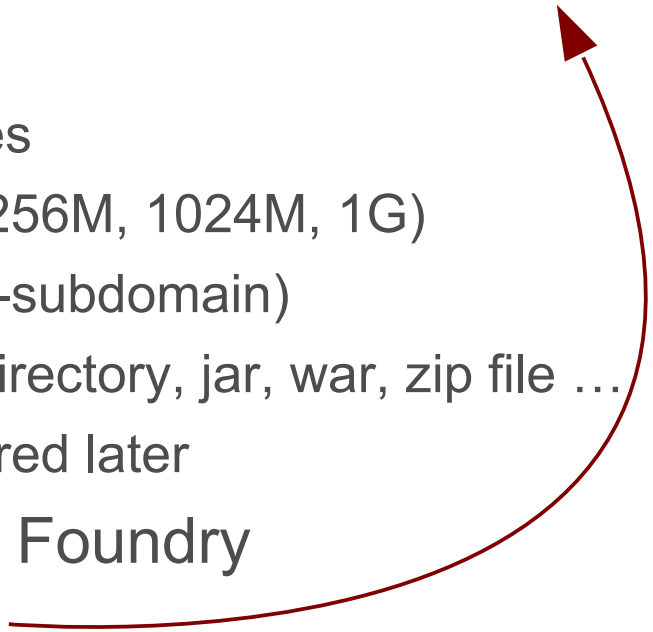
Haiku

- Onsi Fakhouri (Cloud Foundry PM)
  - *Here is my source code*
  - *Run it on the cloud for me*
  - *I do not care how*
- The architecture of CF is fascinating
  - And we *will* cover it
  - But ultimately irrelevant
- I just want to push an application
  - I no longer need to know: how that happens, how it is packaged or how it is run?

© Copyright 2016 Pivotal. All rights reserved

Deploy (*push*) to Cloud Foundry

- Deploy by running **`cf push <name-of-your-app>`**
  - Many options
    - -i Number of instances
    - -m Memory limit (e.g. 256M, 1024M, 1G)
    - -n Hostname (e.g. my-subdomain)
    - -p Local path to app directory, jar, war, zip file …
    - … Others will be covered later
- Your application appears in Cloud Foundry under the name you specify here



© Copyright 2016 Pivotal. All rights reserved

Domains and URLs

- Every CF instance is assigned a domain at installation
  - Known as the *Apps Domain*
  - For PWS this is ***cfapps.io***
- When you deploy, your application gets a unique route (URL) to access it: **hostname + app domain name**
  - By default, hostname = application name
  - Make sure hostname is **unique**
    - *cf push* returns an HTTP 400 error if not
- PWS example:
  - **`cf push spring-music ...`**
  - gets route: ***spring-music.cfapps.io***

© Copyright 2016 Pivotal. All rights reserved

Examples of Using cf push

Specify unique sub-domain by adding numbers, initials ...

- Fully specified (recommended)

```
cf push spring-music -i 1
                     -m 512M
                     -n spring-music-678
                     -p build/libs/spring-music.war
```

  - Deploys war file (specify path if needed)
  - 1 instance, 512M memory
  - Name: **`spring-music`**
    - Appears as **`spring-music`** in Cloud Foundry
  - Hostname: **`spring-music-678`**
    - Creates URL (PWS): **`spring-music-678.cfapps.io`**

© Copyright 2016 Pivotal. All rights reserved

What Happens ?

- `cf` connects to Cloud Foundry using your credentials
- It 'pushes' your application to CF and tells it to deploy it
  - The whole application is uploaded – takes a while
  - CF “stages” your application
    - Recognizes Java WAR file, prepares a “droplet” with a JRE and Tomcat server
  - “Droplet” is deployed to a container and starts running
  - All requests to the *Deployed URL* route to your application
- Whole process logged on screen
  - *See next 3 slides*

© Copyright 2016 Pivotal. All rights reserved

What Happens - 1

URL: spring-music-678.cfapps.io

```
cf push spring-music -n spring-music-678 -i 1 -m 512M
                     -p pre-built/spring-music.war
```

```
> cf push spring-music -n spring-music-678 -p build/libs/spring-music.war -i 1 -m 512M

Updating app spring-music in org your-org / space development as your-id@company.io...
OK

Using route spring-music-678.cfapps.io
Uploading spring-music...
Uploading app files from: pre-built/spring-music.war
Uploading 574.8K, 95 files
Done uploading
OK

Starting app spring-music in org your-org / space development as your-id@company.io...
...
```

Updates CF metadata
(app name, instances, memory)

Establish route

Uploads war

Next...

© Copyright 2016 Pivotal. All rights reserved

What Happens - “Staging”

CF must prepare the app before its first run

```
...
Starting app spring-music in org your-org / space development as your-id@company.io...
OK

-----> Downloaded app package (21M)
-----> Java Buildpack Version: v2.7.1 | https://github.com/cloudfoundry/java-
buildpack#fee275a
-----> Downloading Open Jdk JRE 1.8.0_40 from
https://download.run.pivotal.io/openjdk/lucid/x86_64/openjdk-1.8.0_40.tar.gz (6.1s)
       Expanding Open Jdk JRE to .java-buildpack/open_jdk_jre (1.3s)
-----> Downloading Spring Auto Reconfiguration 1.7.0_RELEASE from
https://download.run.pivotal.io/auto-reconfiguration/auto-reconfiguration-1.7.0_RELEASE.jar
(0.2s)
-----> Downloading Tomcat Instance 8.0.20 from
https://download.run.pivotal.io/tomcat/tomcat-8.0.20.tar.gz (1.1s)
       Expanding Tomcat to .java-buildpack/tomcat (0.1s)
-----> Downloading Tomcat Lifecycle Support 2.4.0_RELEASE from
https://download.run.pivotal.io/tomcat-lifecycle-support/tomcat-lifecycle-support-
2.4.0_RELEASE.jar (0.0s)
-----> Uploading droplet (73M)
...
```

“Buildpack” selected and executed

Buildpack configures Java

Reconfigure Spring for cloud environment

Buildpack obtains Tomcat

Buildpack creates “Droplet”

Next...

© Copyright 2016 Pivotal. All rights reserved

What Happens - “Start”

```
...
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
1 of 1 instances running

App started

OK

App spring-music was started using this command `JAVA_HOME=$PWD/.java-buildpack/open_jdk_jre JAVA_OPTS="-
Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PWD/.java-buildpack/open_jdk_jre/bin/killjava.sh
-Xmx382293K -Xms382293K -XX:MaxMetaspaceSize=64M -XX:MetaspaceSize=64M -Xss995K
-Daccess.logging.enabled=false -Dhttp.port=$PORT" $PWD/.java-buildpack/tomcat/bin/catalina.sh run`

Showing health and status for app spring-music in org your-org as your-id@company.io...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: spring-music-678.cfapps.io
last uploaded: Tue Mar 17 17:58:35 UTC 2015

     state     since                    cpu    memory           disk
#0   running   2015-03-17 01:59:35 PM   0.0%   474.4M of 512M   150.3M of 1G
```

Cloud Foundry runs the “Droplet” on a “container”

Health Check

Done! 1 application instance running on spring-music-678.cfapps.io

© Copyright 2016 Pivotal. All rights reserved

Application State and Logs

- Run **`cf apps`**

```
> cf apps
Getting apps in org pivotaledu / space development as kkrueger@pivotal.io...
OK

name            requested state   instances   memory   disk   urls
spring-music     started          1/1         512M     1G     spring-music-678.cfapps.io
```

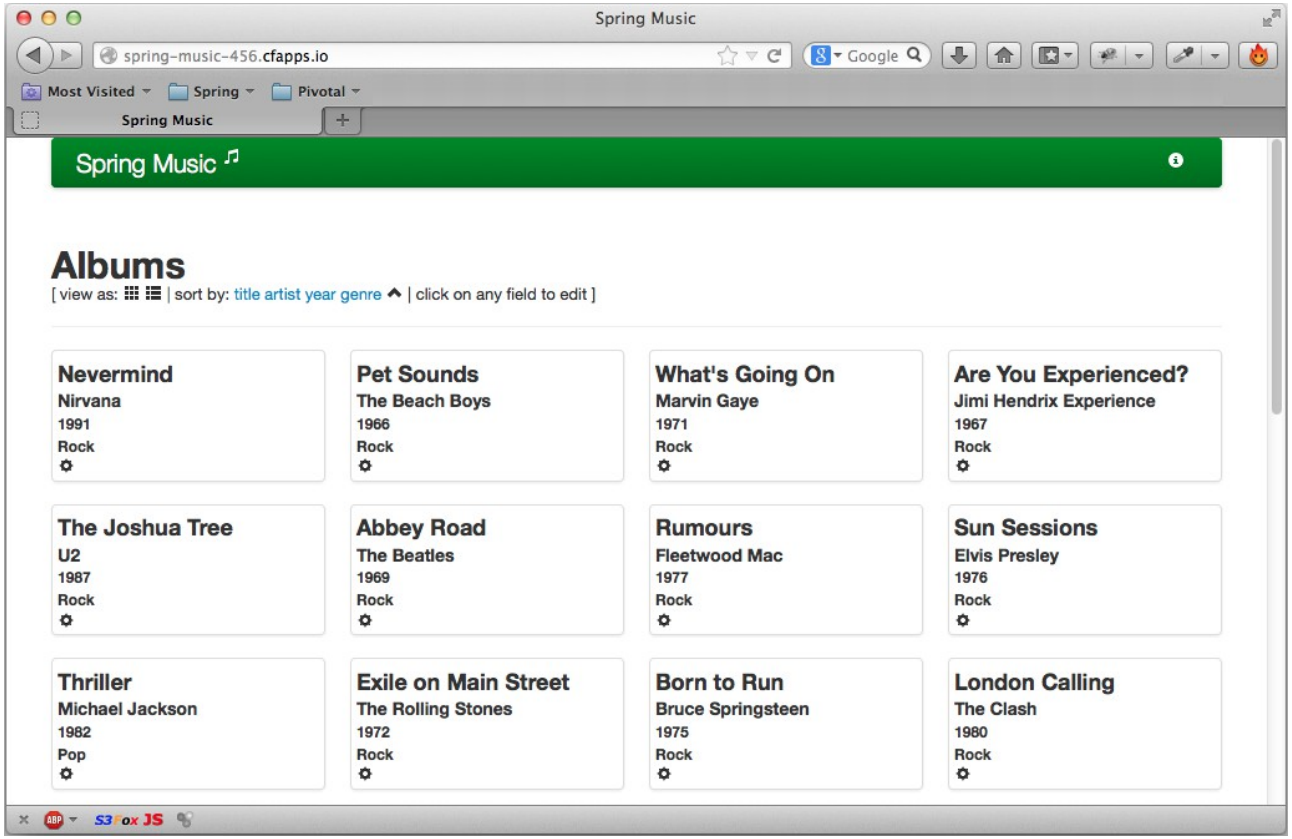
- **`cf logs spring-music`**

```
> cf logs spring-music
Connected, tailing logs for app spring-music in org pivotaledu / space development as
kkrueger@gopivotal.com...
2014-06-07T23:01:47.68-0400 [RTR]      OUT spring-music-678.cfapps.io -
[08/06/2014:03:01:47 +0000] "GET /assets/js/status.js HTTP/1.1" 200 844 "http://spring-
music-678.cfapps.io/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5)
AppleWebKit/537.73.11 (KHTML, like Gecko) Version/6.1.1 Safari/537.73.11"
10.10.66.34:64401 vcap_request_id:73037523-63ef-498f-6cd8-d3b48fe69e84
response_time:0.003693009 app_id:314f0434-d2c9-446c-ab4a-6c310878ca80
2014-06-07T23:01:48.47-0400 [RTR]      OUT spring-music-678.cfapps.io -
[08/06/2014:03:01:48 +0000] "GET /assets/templates/header.html HTTP/1.1" 200 1060
"http://spring-music-678.cfapps.io/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5)
AppleWebKit/537.73.11 (KHTML, like Gecko) Version/6.1.1 Safari/537.73.11"
10.10.66.34:64324 vcap_request_id:39fbb3f2-46fb-4bd7-78d6-8994fafade9f
response_time:0.004132254 app_id:314f0434-d2c9-446c-ab4a-6c310878ca80
```

© Copyright 2016 Pivotal. All rights reserved

See The Application Running

- Open a browser window to **`spring-music-678.cfapps.io`**



© Copyright 2016 Pivotal. All rights reserved

Configuring a Deployed Application

- Change the number of instances
  - **`cf scale <app> -i <new-value>`**
  - Two instances: **`cf scale spring-music -i 2`**
  - New instances added, *or* some existing instances stopped

- Change the memory allocation
  - **`cf scale <app> -m <new-value>`**
  - 1024M: **`cf scale spring-music -m 1024M`**
  - Requires a restart to take effect

Note: Default memory quote on PWS is only 2G for *all* applications in total

© Copyright 2016 Pivotal. All rights reserved

Stopping and Starting

`cf stop`

- Sends SIGTERM message to application
- Sends SIGKILL 10 seconds later if still running

`cf start`

- Starts existing application

`cf restart`

- **`cf stop`** followed by **`cf start`**

`cf restage`

- Repeats the staging process, and starts the app.
- Useful when environment variables / bound services change
  - (Covered later)

© Copyright 2016 Pivotal. All rights reserved

Roadmap

- Get Setup
- Login
- Deploying an Application
- **Using a Manifest**
- The Application Manager

© Copyright 2016 Pivotal. All rights reserved

Cloud Foundry Manifest file

- Describes the application deployment options
  - Automates subsequent deployments
  - Same options as `cf push` command
    - Plus options *only* available in the manifest
- Default name: `manifest.yml`
  - YAML* format
  - Human friendly data serialization standard
  - Supported by many programming languages
  - Less verbose than XML, similar to JSON
  - http://www.yaml.org

Create using your favorite text editor

** YAML Ain't Markup Language*

© Copyright 2016 Pivotal. All rights reserved

Using a Manifest with Push

- cf push automatically detects manifest
  - In current directory or parent directories
  - Expects file named manifest.yml
  - Override with -f option
    - cf push -f dev-manifest.yml
  - Or ignore with --no-manifest option

Most labs in this course use a manifest

- No manifest found?
  - cf push will default all deployment options
    - Not the best choices

© Copyright 2016 Pivotal. All rights reserved

YAML Format

- 3 dashes
  - Indicate start of section
- Indent with spaces, not tabs!
  - Determines hierarchy
  - Each indent 2 spaces
  - "-" defines "group"
- Syntax: **`property: value`** pairs
  - Space after colon *required*
- # starts a one line comment

```
---
applications:
- name: nodetestdh01
  memory: 64M
  instances: 2
  host: crn    # comment
  domain: cfapps.io
  path: .
  # comment

- name: nextapp    # group 2
  memory: 256M
  ...
```

- See http://www.yaml.org/spec/1.2/spec.html

© Copyright 2016 Pivotal. All rights reserved

manifest.yml Example

```
---
applications:
- name: cf-node-demo
  command: node app.js
  memory: 128M
  instances: 1
  host: demo-${random-word}
  domain: cfapps.io
  path: .
```

- applications: can describe one or more applications
- name: of the app – used in commands
- command: command to run – *optional* (-c)
- memory ceiling / instances to run (-m, -i)
- host: your choice, must be unique within domain (-n)
  - **Tip:** use ${random-word}
- path: to executable (-p)

© Copyright 2016 Pivotal. All rights reserved

Roadmap

- Get Setup
- Login
- Deploying an Application
- Using a Manifest
- **The Application Manager (Console)**

© Copyright 2016 Pivotal. All rights reserved

Apps Manager

- Login to Cloud Foundry using your web-browser
  - Pivotal Web Services: **`http://run.pivotal.io`**
    - Your Cloud Foundry instance URL will be different
    - **`console.<your-cf-domain>`**
  - Use the username and password you registered with
  - Our new application should show green in the Apps Manager

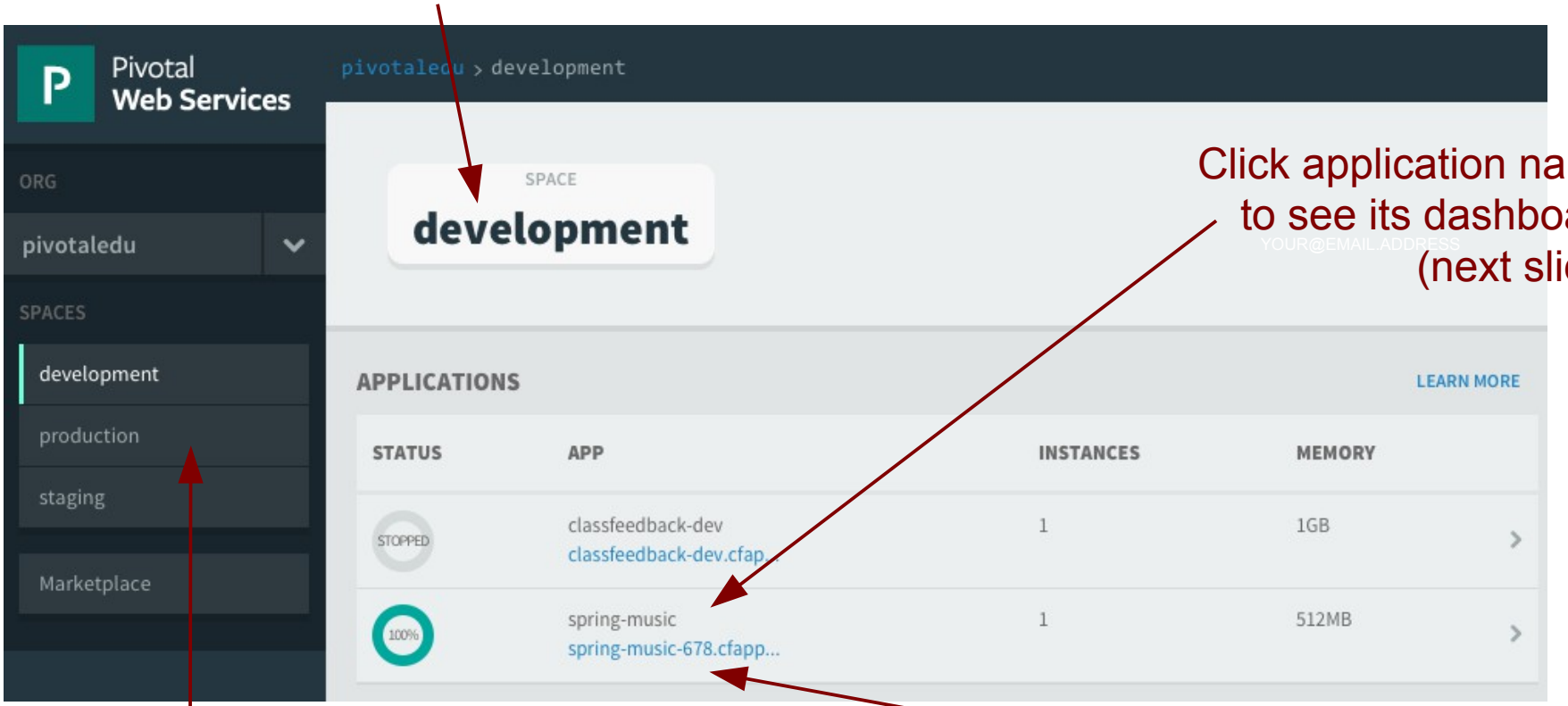
- Next slide ...

NOTE: Only Pivotal CF comes with the Apps Manager
Open Source Cloud Foundry *does not*

© Copyright 2016 Pivotal. All rights reserved

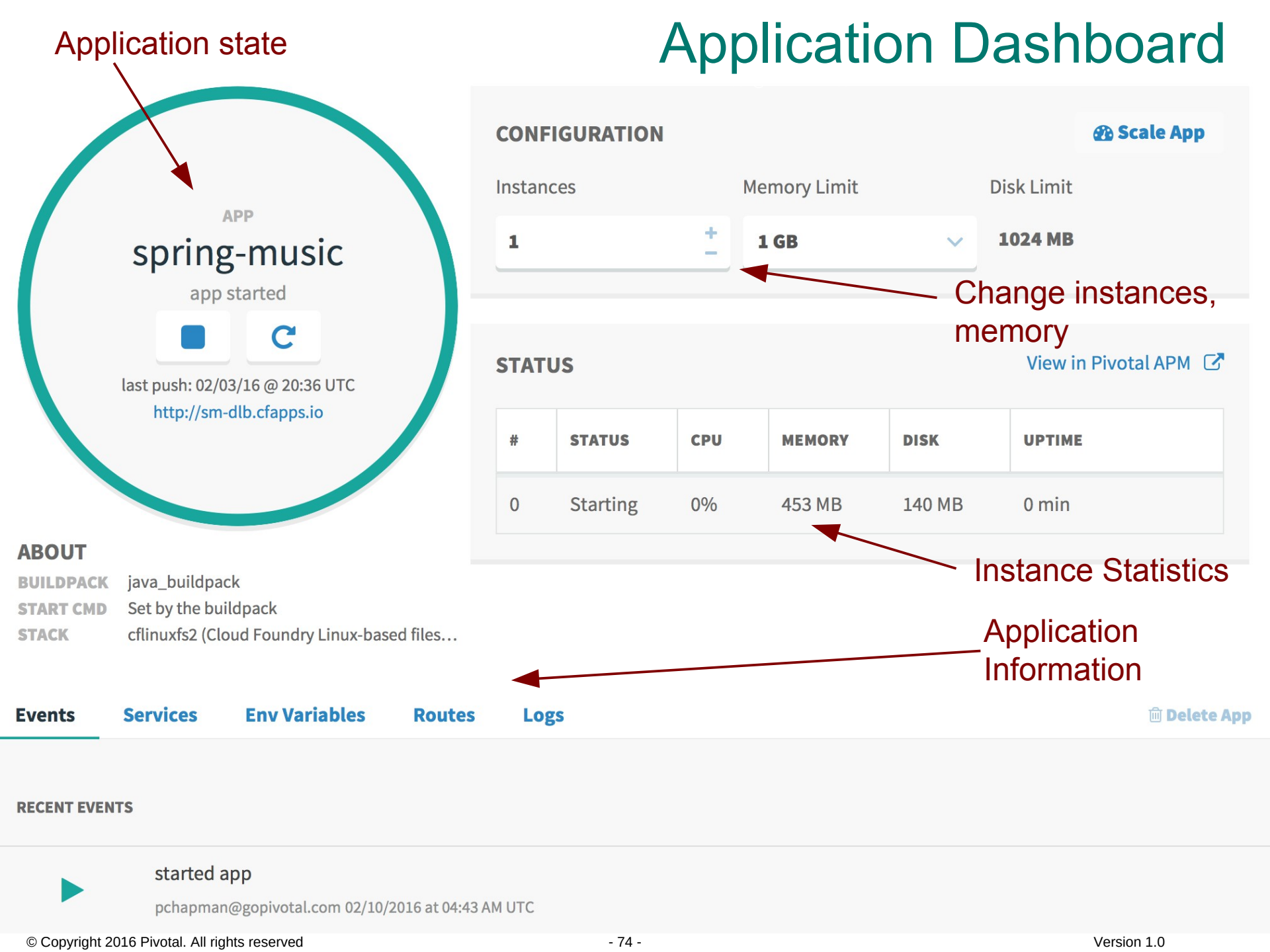
Apps Manager Home Page

- At a glance view of all your applications
  - Shows current space



© Copyright 2016 Pivotal. All rights reserved

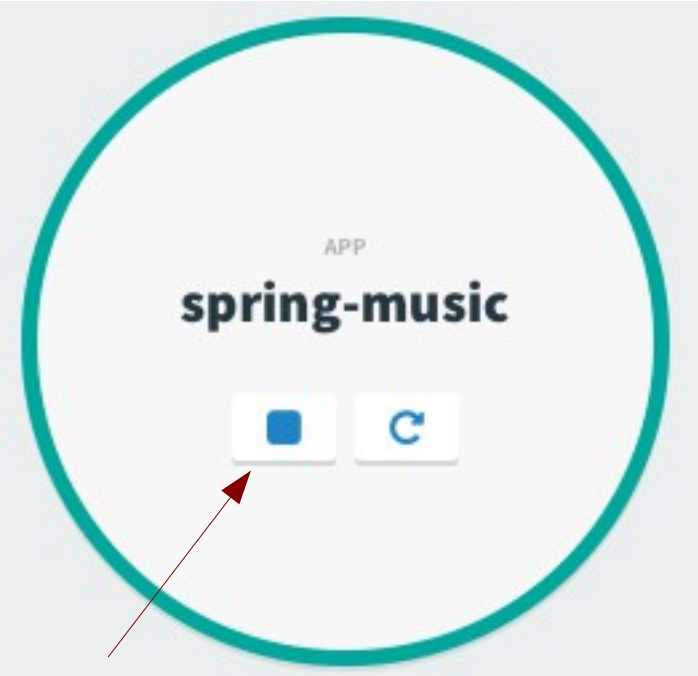
Application Dashboard



© Copyright 2016 Pivotal. All rights reserved

Stopping and Starting

- Just click the square to stop
- Click play to start



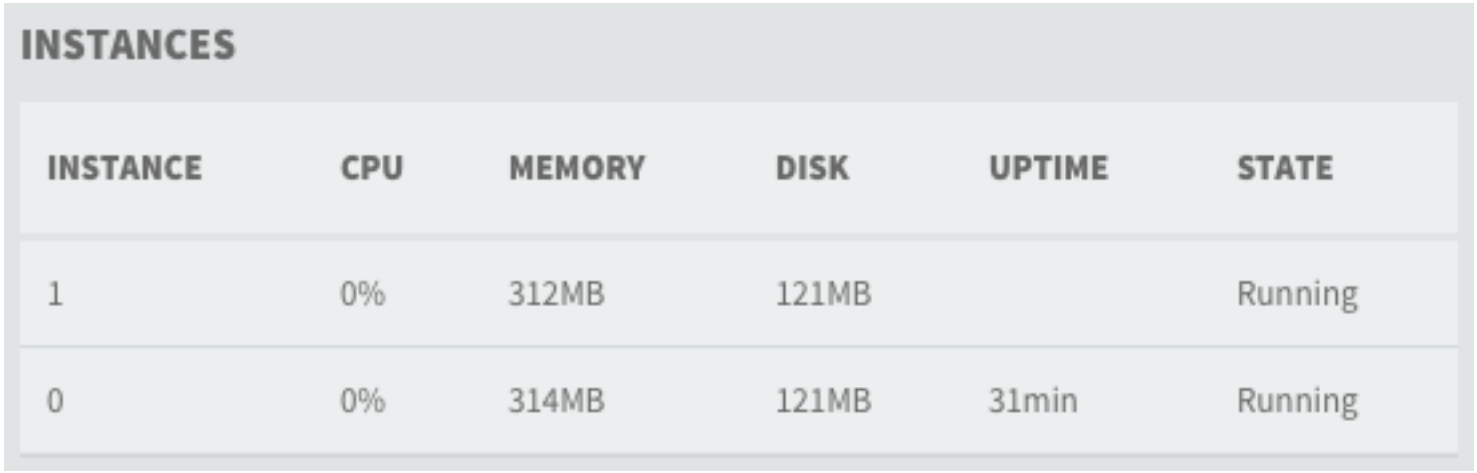
Click to Stop

Click to Start

© Copyright 2016 Pivotal. All rights reserved

Monitoring Instances

- The status panel shows all your instances
  - Provides statistics
  - Updated live (slight time-lag)



INSTANCES

INSTANCE	CPU	MEMORY	DISK	UPTIME	STATE
1	0%	312MB	121MB		Running
0	0%	314MB	121MB	31min	Running

© Copyright 2016 Pivotal. All rights reserved

Summary

- After completing this lesson, you should have learned:
  - How to Deploy an application to CloudFoundry using CLI
  - Managing application instances using Apps Manager

© Copyright 2016 Pivotal. All rights reserved

Labs

Using Cloud Foundry

- Push existing application
- Binding to services
- Scale an application
- Monitoring

© Copyright 2016 Pivotal. All rights reserved

Building Microservices with Spring Data Rest

Spring Data Rest, Spring Cloud Connector

© Copyright 2016 Pivotal. All rights reserved

Topics in this Session

- **Spring Data REST**
- Spring Cloud Connector

© Copyright 2016 Pivotal. All rights reserved

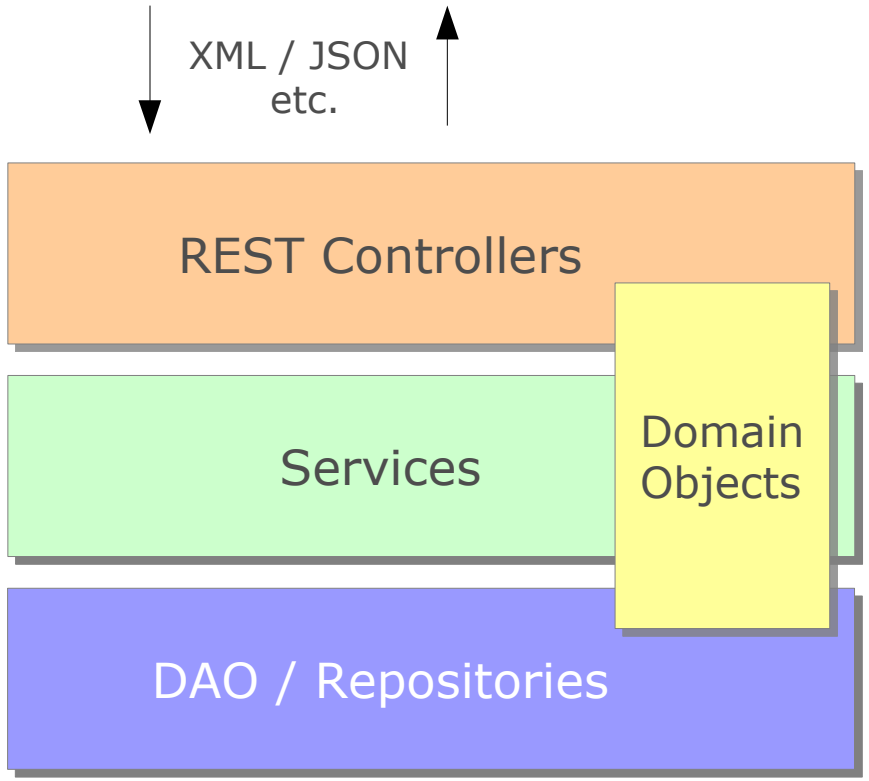
Spring Data REST

- Spring Data REST defines endpoints based on conventions
- A subproject within the *Spring Data* umbrella
  - http://projects.spring.io/spring-data-rest/

© Copyright 2016 Pivotal. All rights reserved

Typical REST Application

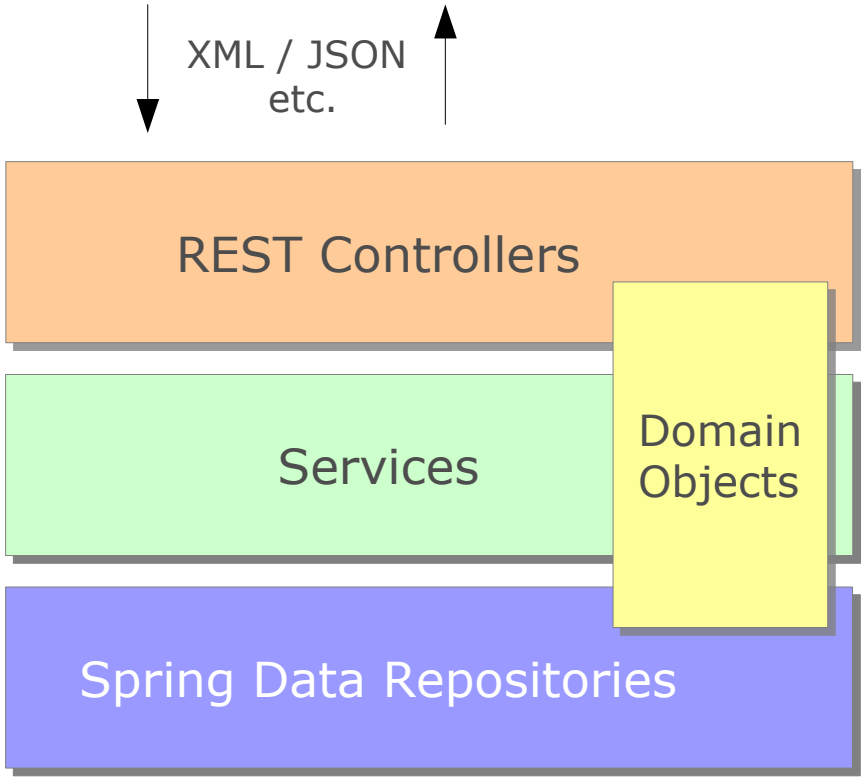
- Familiar web application architecture
- REST Controllers provide CRUD interface to clients
- DAO provides CRUD interface to DB



© Copyright 2016 Pivotal. All rights reserved

Add Spring Data

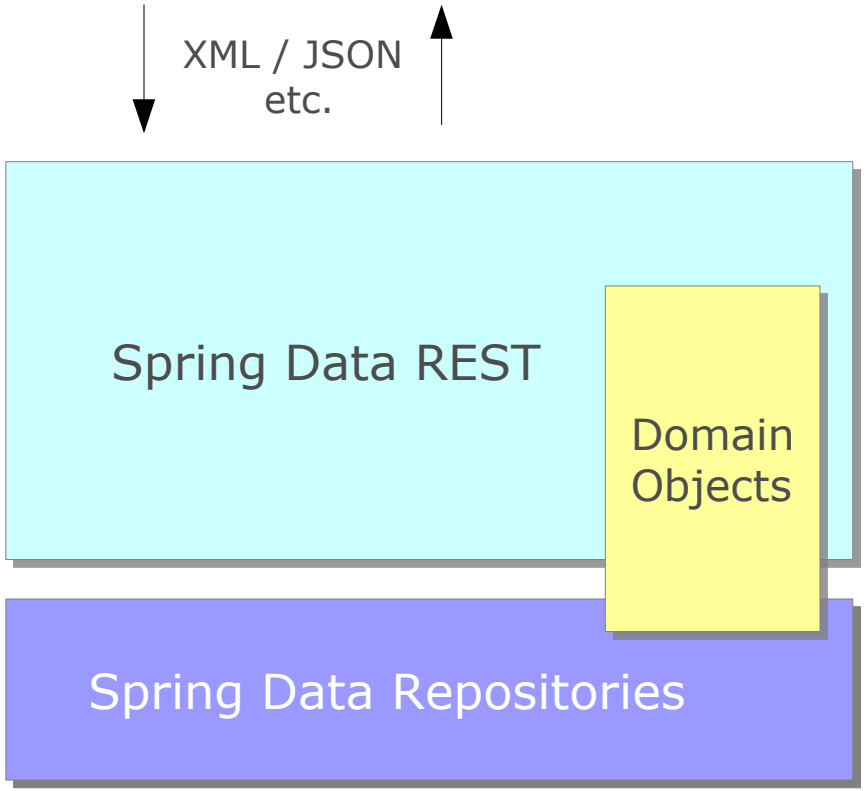
- Spring Data provides dynamic Repositories
- You provide the interface, Spring Data dynamically implements.
  - JPA, MongoDB, Neo4J, GemFire, ...
- Controllers & Service Layer have minimal logic



© Copyright 2016 Pivotal. All rights reserved

Spring Data REST

- Plugs into dynamic repositories
- Generates RESTful interface
  - *GET*, *PUT*, *POST*, *DELETE*
- Code needed only to override defaults.



© Copyright 2016 Pivotal. All rights reserved

Quick Start

- Annotate Domain objects
  - i.e. use JPA annotations if using JPA
- Define interface for dynamic repository
  - Implement CrudRepository at minimum
  - Add @RestResource to describe how to expose via REST

```
@RestResource(path="users", rel="users")
public interface UserRepository extends CrudRepository<User, Long> {}
```

- Configure Spring Data REST
  - `@Import(RepositoryRestMvcConfiguration.class)`
  - Or, extend this class to override defaults

© Copyright 2016 Pivotal. All rights reserved

HTTP Verb to Method Mapping

Verb	Repository Method
GET	`CrudRepository<ID,T>.findOne(ID id)`
POST	`CrudRepository<ID,T>.save(T entity)`
PUT	`CrudRepository<ID,T>.save(T entity)`
DELETE	`CrudRepository<ID,T>.delete(ID id)`

- All methods available by default
  - Disable via override / configuration

```
//  Restrict DELETE:
@RestResource(exported=false)
void delete(Long id);
```


© Copyright 2016 Pivotal. All rights reserved

Topics in this Session

- Spring Data REST
- **Spring Cloud Connector**

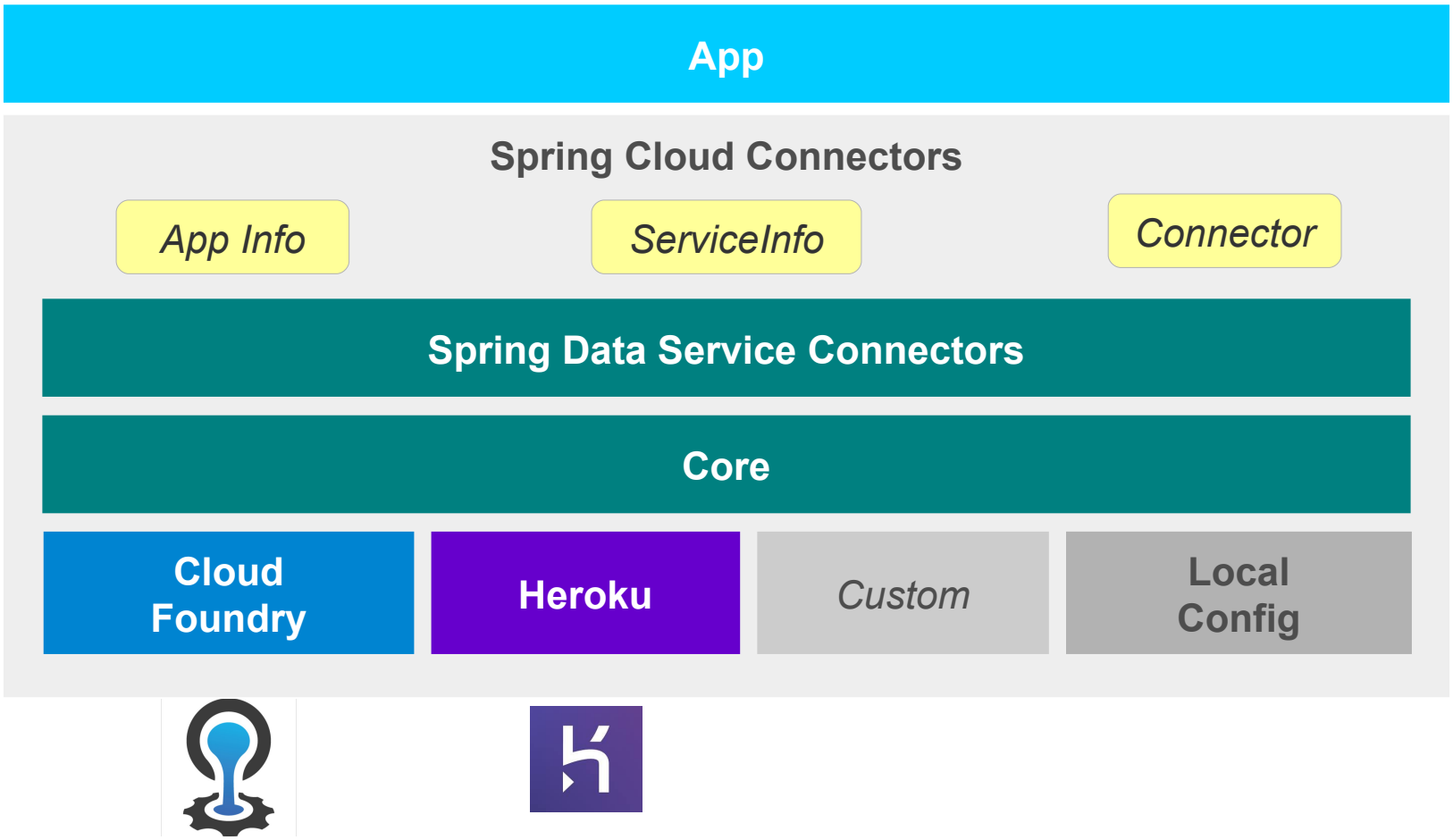
© Copyright 2016 Pivotal. All rights reserved

Spring Cloud Connector

- Allows your code to be mostly “Cloud agnostic”
- Switch easily between Cloud Providers
- Support provided for Cloud Foundry and Heroku
  - Can easily create custom extensions for other providers

© Copyright 2016 Pivotal. All rights reserved

Spring Cloud Connectors
Architecture



© Copyright 2016 Pivotal. All rights reserved

Spring Cloud Connector – Cloud Foundry

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.3.3.RELEASE</version>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-spring-service-connector</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-cloudfoundry-connector</artifactId>
  </dependency>
</dependencies>
```

parent

pom.xml

Can be replaced with spring-cloud-**heroku**-connector

© Copyright 2016 Pivotal. All rights reserved

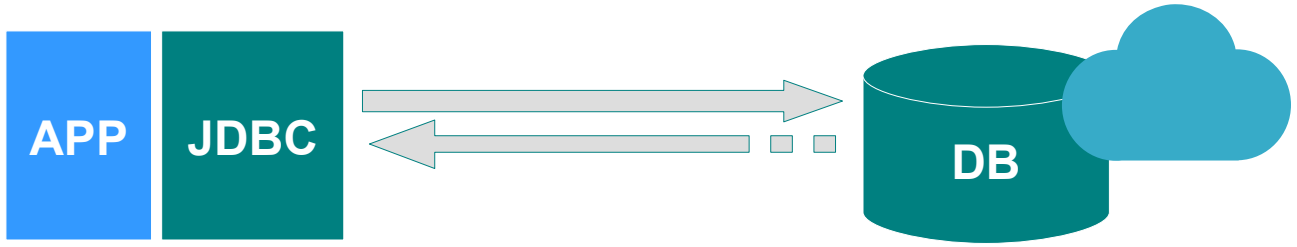
Service Bindings & Connectors – JDBC

- DataSource – Relational DB (JDBC Connector)

```
DataSource dataSource =
    cloud.getServiceConnector("mydb", DataSource.class, null);
```

- Bean & Configuration Properties

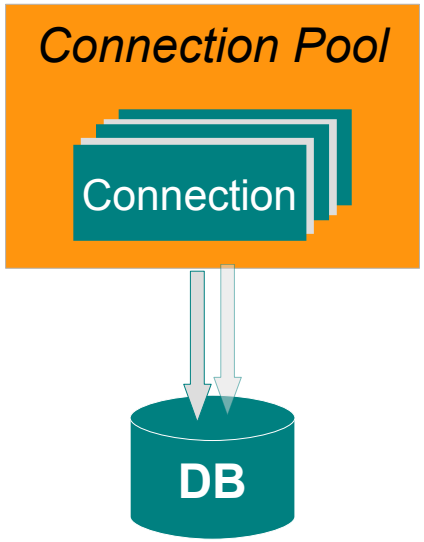
```
@Bean
@ConfigurationProperties("db.")
public DataSource dataSource() {
   return cloud.getSingletonServiceConnector(DataSource.class, null);
}
```



© Copyright 2016 Pivotal. All rights reserved

JDBC – DataSource Connection Pooling

- Connection Pools (if in classpath)
  - Default search order:
    - Apache Commons DBCP and DBCP 2
    - Tomcat DBCP
    - Tomcat JDBC Connection Pool
    - HikariCP
  - Optionally customize search order




```
List<String> cpools = Arrays.asList("TomcatJdbc", "HikariCp", "BasicDbcp");
DataSourceConfig config = new DataSourceConfig(cPools);
DataSource dataSource =
          cloud().getServiceConnector("mydb", DataSource.class, config);
```

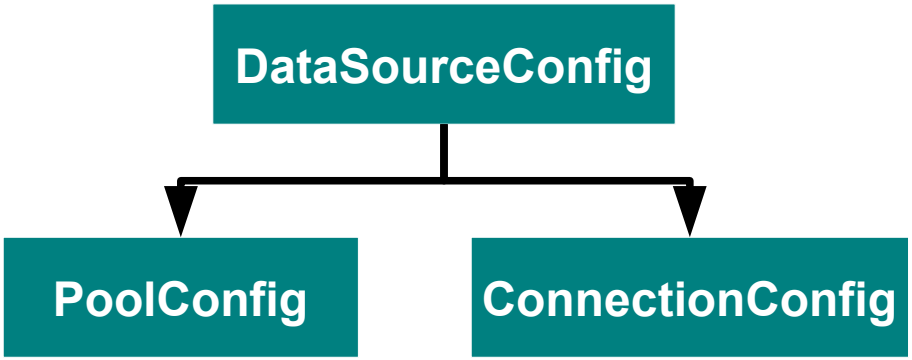

© Copyright 2016 Pivotal. All rights reserved

JDBC – DataSource Customization

- DataSourceConfig = PoolConfig + ConnectionConfig

```
import org.springframework.cloud.service.PooledServiceConnectorConfig.PoolConfig;
import org.springframework.cloud.service.relational.DataSourceConfig.ConnectionConfig;
import org.springframework.cloud.service.relational.DataSourceConfig;
```

```
int minPoolSize = 10, maxPoolSize=100, maxWaitTime=10000;
PoolConfig poolConfig = new PoolConfig(minPoolSize, maxPoolSize, maxWaitTime);
ConnectionConfig connectionConfig = new ConnectionConfig("");
DataSourceConfig config = new DataSourceConfig(poolConfig, connectionConfig);
DataSource dataSource =
                cloud().getServiceConnector("mydb", DataSource.class, config);
```



© Copyright 2016 Pivotal. All rights reserved

JDBC – Service Bindings in Cloud Foundry

- Service Binding

```
cf bind-service myapp mydb
cf services
```

- Marketplace Service

```
cf create-service cleardb spark mydb
```

- User Defined Service

```
cf create-user-defined-service mydb -p "db_url,db_username,db_password"
```


© Copyright 2016 Pivotal. All rights reserved

Summary

- Spring Data REST provides automatic REST implementations for CRUD repositories
- Spring cloud connector makes your application portable

© Copyright 2016 Pivotal. All rights reserved

Splitting the Monolith

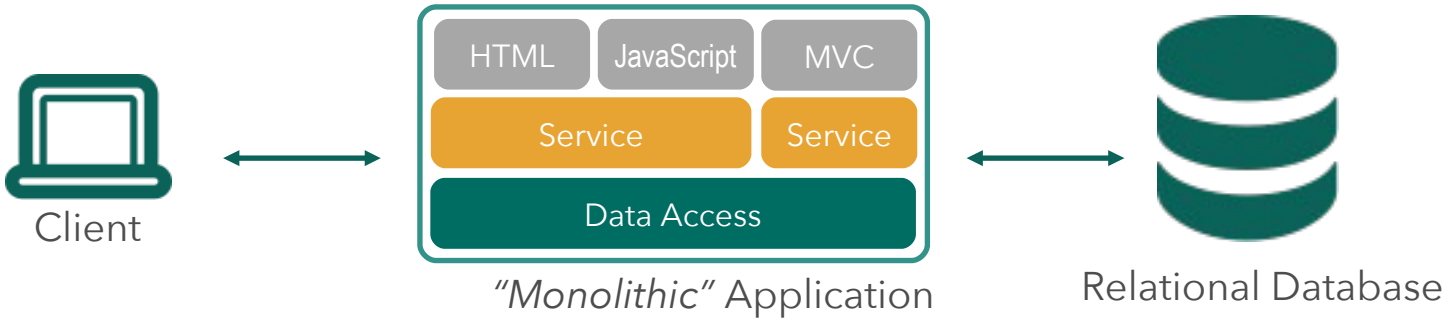
Building Cloud Native Applications

© Copyright 2016 Pivotal. All rights reserved

Classic vs Microservice Architectures

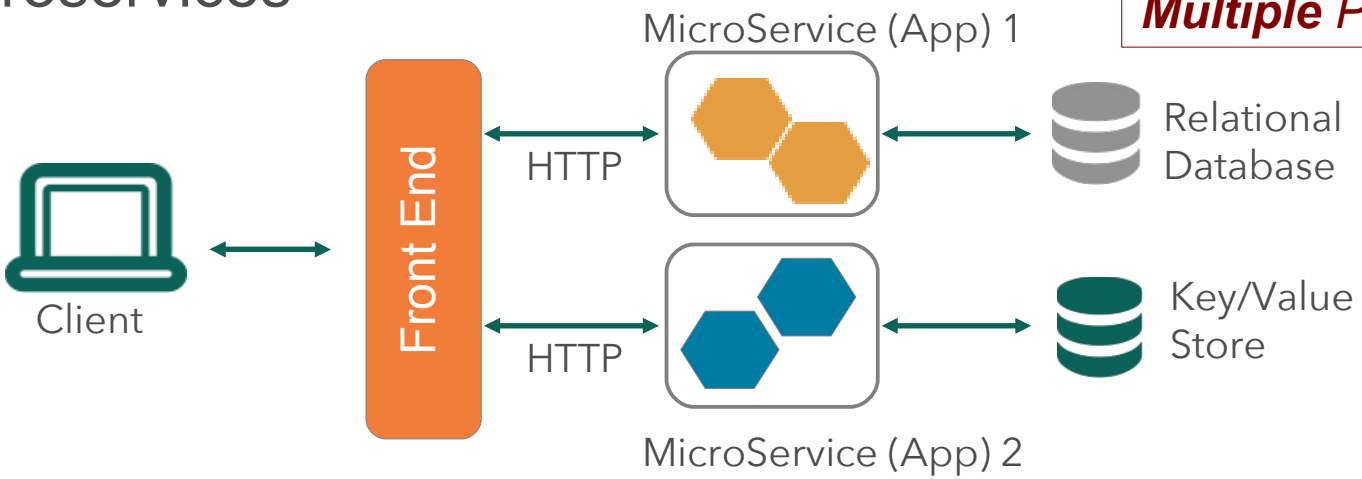
*Monolithic = **Single** Process*

- Classic *three-tier* application



- Microservices

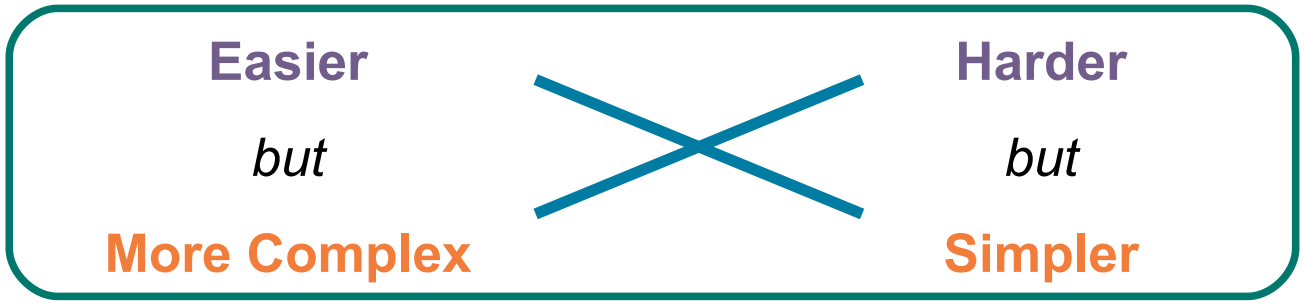
***Multiple** Processes*



© Copyright 2016 Pivotal. All rights reserved

Trade-Offs

- **Single App**
  - Easier to build
  - Large process to deploy
  - *Ultimately* more complex to enhance and maintain
  - Scaling Up (bigger processors) limited
- **Microservices**
  - Harder to build
  - Network overheads
  - *Ultimately* simpler to extend and maintain
  - Scaling Out (more processes) easier



© Copyright 2016 Pivotal. All rights reserved

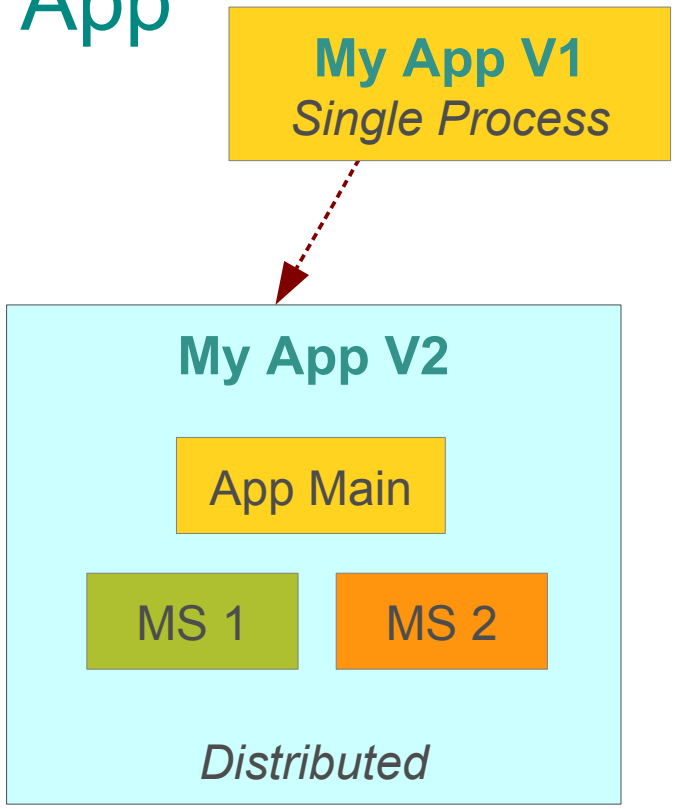
Microservice Prerequisites

- Microservices are not for everyone
  - It's as much *how* you develop as *what* you develop

- *"You must be this tall" to "ride" Microservices*
  - *Do you develop using*
    - Rapid provisioning?
    - Basic monitoring?
    - Continuous Integration / Continuous Deployment?
    - A DevOps culture?

© Copyright 2016 Pivotal. All rights reserved

Route to Microservices: New App



- Start with a “Monolith”
  - Keep it simple, at first
  - Single process application
  - Apply 12-factor patterns
    - http://12factor.net
  - *Cloud-ready even at this stage*
- As it grows
  - Decompose into micro-service(s)
  - Enables separately manageable and deployable units
  - Each can use own storage solution (*polyglot persistence*)

© Copyright 2016 Pivotal. All rights reserved

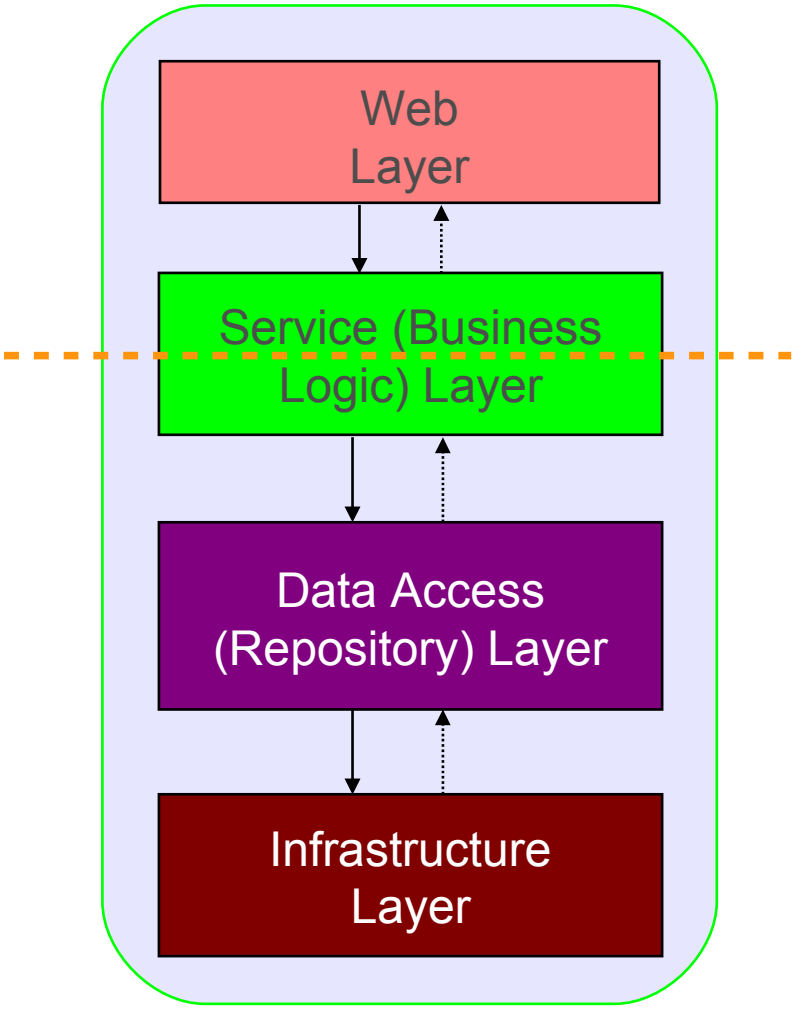
Route To Microservices: Existing App

- Develop *new* functionality as microservice(s) *around* existing single-process application
  - Use Facades/Adapters/Translators to integrate them
- *"Strangle the Monolith"*
  - Refactor *existing* monolith functionality into new microservice(s)
  - Long-term evolution:
    - Monolith withers to nothing
    - Or is reduced to a solid, *reliable* core that is not worth refactoring (because we *know* it works)

© Copyright 2016 Pivotal. All rights reserved

Decomposing the “Monolith”

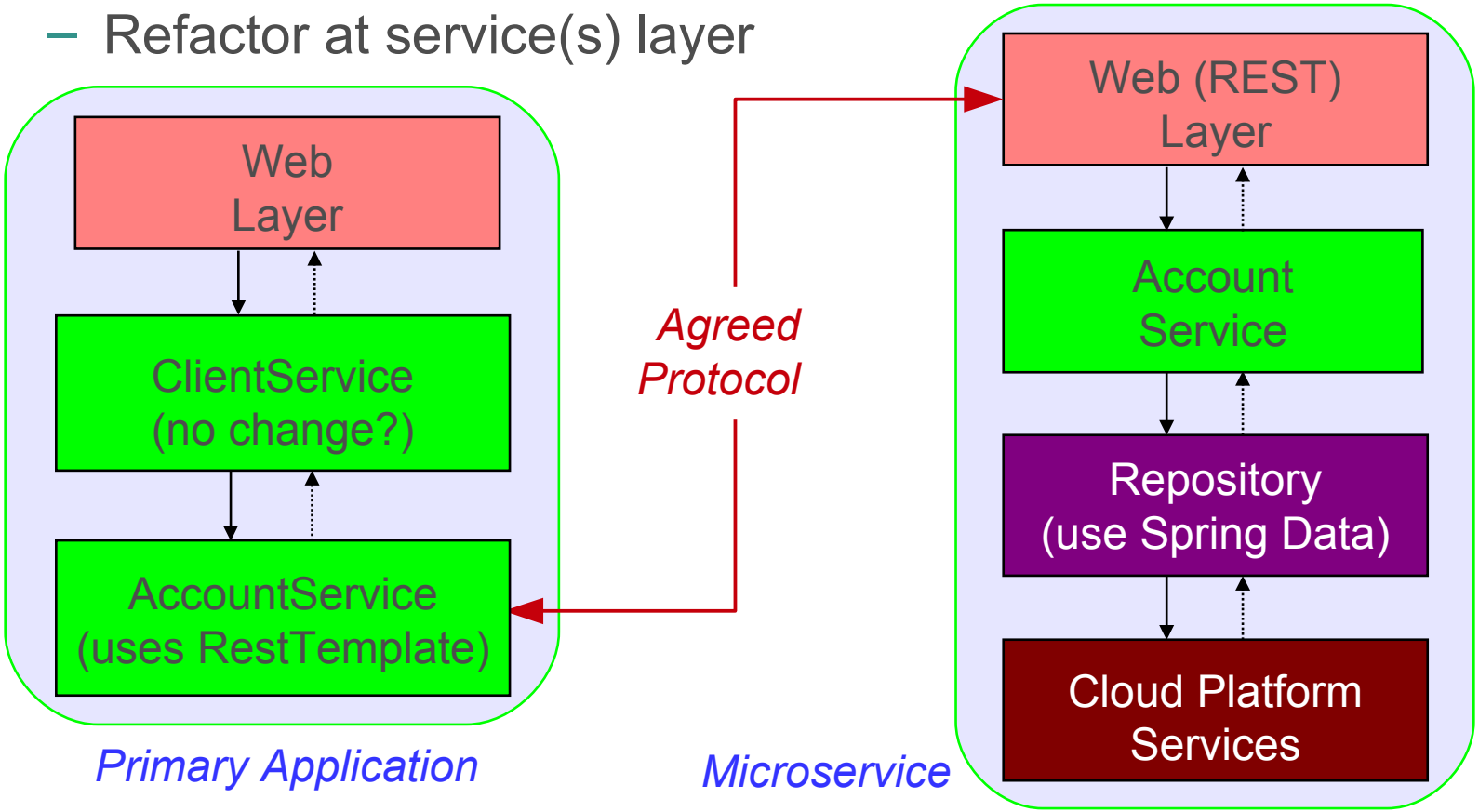
- Many Java applications use a classic three-layer internal architecture
  - Services (business logic)
  - Repositories (data-access)
  - Infrastructure (interface to external resources)
  - Web-layer (optional), other interfaces possible
- Refactor as two processes
  - See next slide



© Copyright 2016 Pivotal. All rights reserved

Refactoring to Microservices Architecture

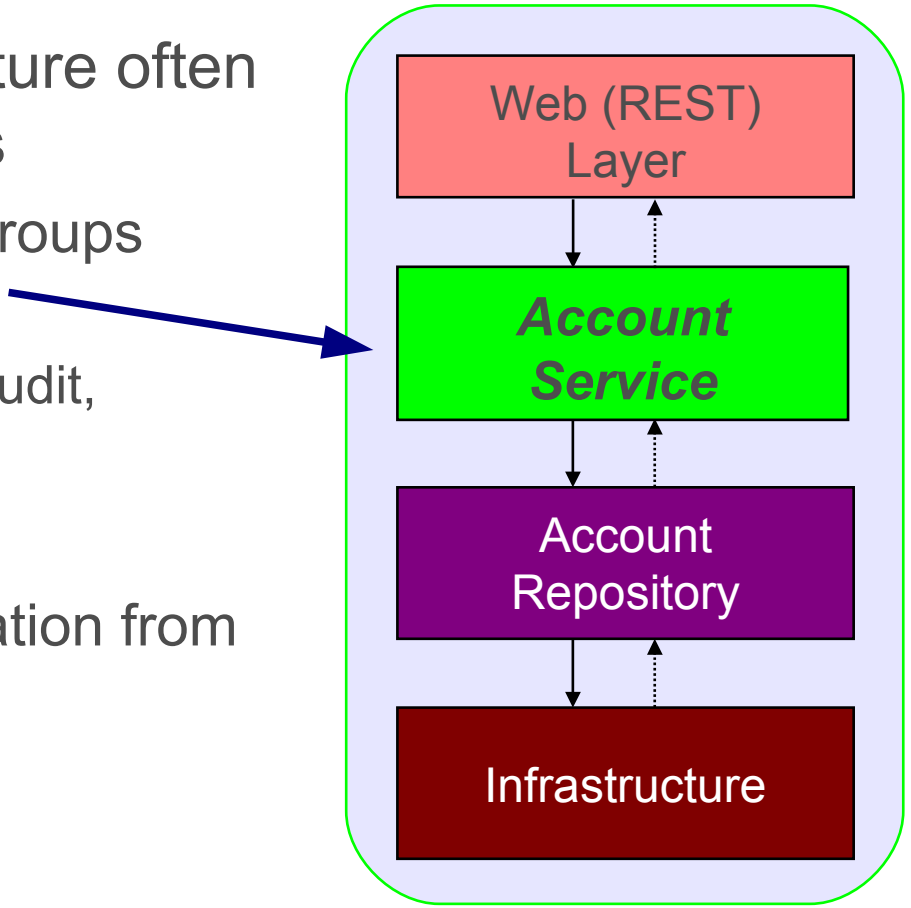
- The Microservice *is* a service (or set of services)
  - Refactor at service(s) layer



© Copyright 2016 Pivotal. All rights reserved

Clarification: Services vs Microservices – I

- In the multi-layer architecture often used by Java applications
  - A service-layer *service* groups related business logic
    - Accounts, customers, audit, invoices, users …
- Separation of Concerns
  - Separates data manipulation from access/storage



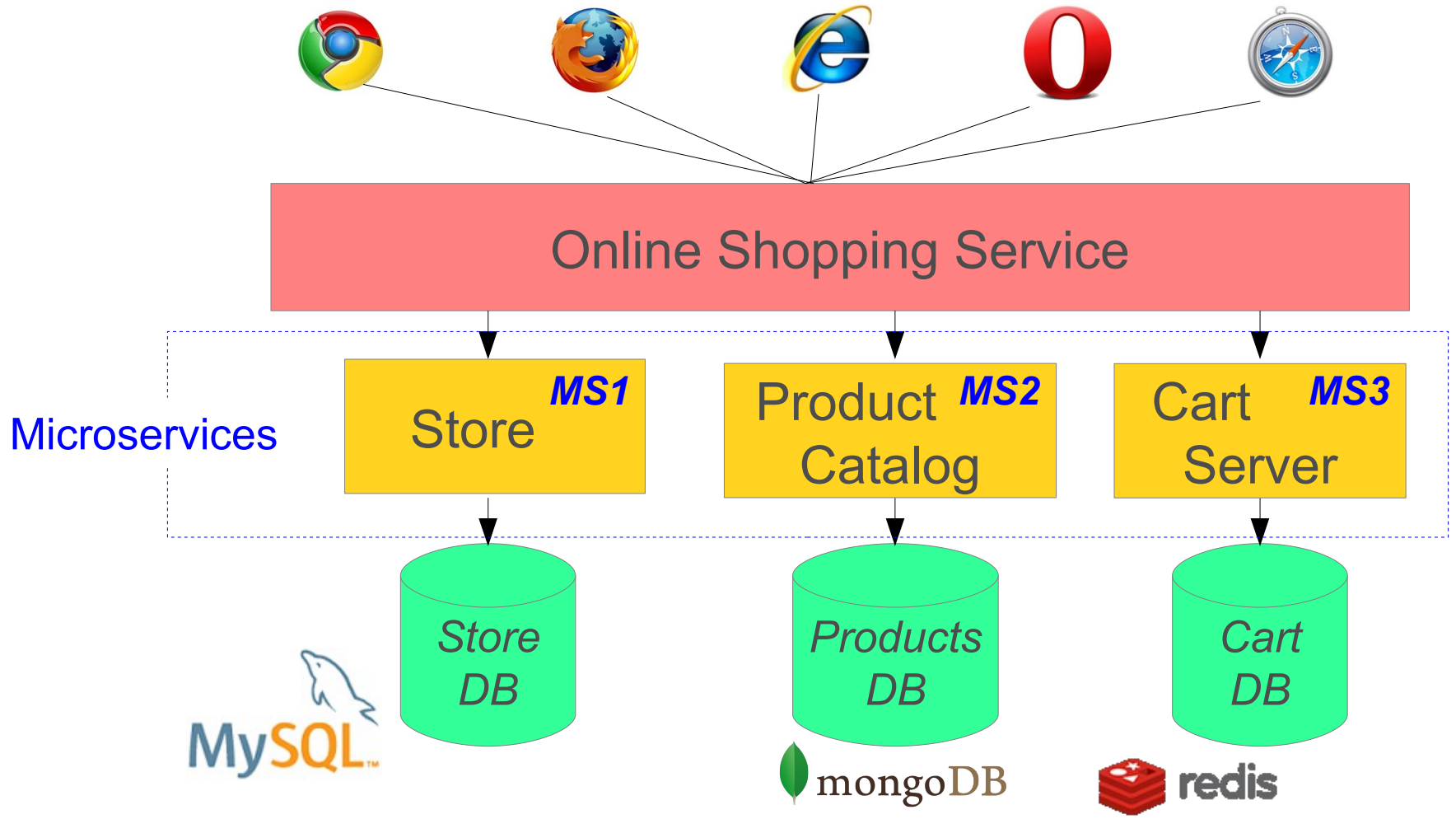
© Copyright 2016 Pivotal. All rights reserved

Clarification: Services vs Microservices – II

- A "*microservice*" is a *process* that wraps up *multiple* service-layer services
  - Typically all the data that belongs in the same data store
- Example: *Online-Shop (next slide)*
  - **Store**
    - Accounts, customers, inventory, invoices ...
    - One microservice, typically using a transactional RDB
  - **Product Catalog**
    - Separate service using a document store
  - **Shopping Carts:**
    - Separate service using key-value store

© Copyright 2016 Pivotal. All rights reserved

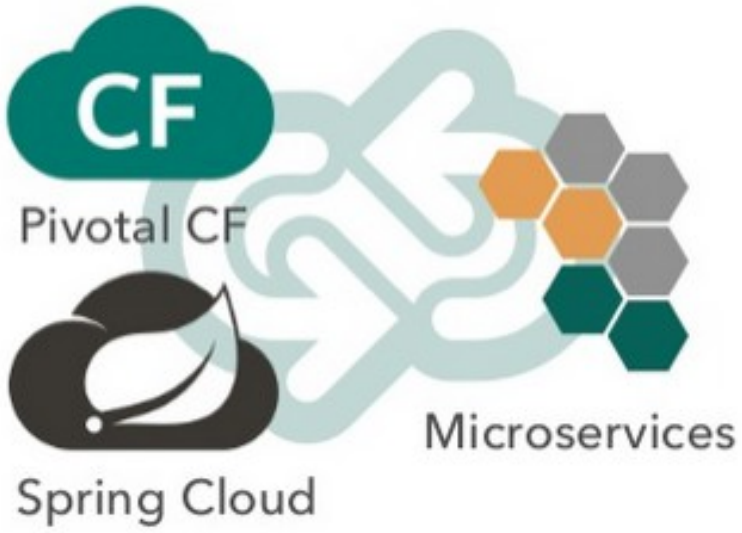
Microservices Example



© Copyright 2016 Pivotal. All rights reserved

Summary

- After completing this lesson, you should have learned:
  - What is a Microservice Architecture?
  - Advantages and Challenges of Microservices
  - Implementation using Spring Cloud projects



© Copyright 2016 Pivotal. All rights reserved

Lab

Identifying bounded contexts of a monolith App

© Copyright 2016 Pivotal. All rights reserved

Designing Applications for Cloud Foundry

Writing Applications that Scale

Getting it right *By Design*

Pivotal

© Copyright 2016 Pivotal. All rights reserved

Overview

- After completing this lesson, you should be able to:
  - Consider design considerations for Cloud Foundry

© Copyright 2016 Pivotal. All rights reserved

Roadmap

- 12-Factor Applications

© Copyright 2016 Pivotal. All rights reserved

Developer Topics / Application Architecture

- Applications may require adjustments to run successfully in Cloud environment
  - See Developer Topics:
  - http://docs.cloudfoundry.org/devguide/deploy-apps/prepare-to-deploy.html

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

- http://12factor.net

- Outlines architectural principles for modern apps
  - Focus on scaling, continuous delivery, portable, and cloud ready

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

I. Codebase

One codebase tracked in SCM, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Configuration

Store config in the environment

IV. Backing Services

Treat backing services as attached resources

V. Build, Release, Run

Strictly separate build and run stages

VI. Processes

Execute app as stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep dev, staging, prod as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin / mgmt tasks as one-off processes

Pivotal

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

I. Codebase

One codebase tracked in SCM, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Configuration

Store config in the environment

- Codebase
  - An application has a single codebase
    - Multiple codebase = distributed system (not an app)
      - Each component in a codebase can (should) be an app
  - Tracked in version control
    - Git, Subversion, Mercurial, etc.
  - Multiple Deployments
    - i.e. development, testing, staging, production, etc.

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

I. Codebase

One codebase tracked in SCM, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Configuration

Store config in the environment

- Dependencies
  - Packaged as jars (Java), RubyGems, CPAN (Perl)
  - Declared in a Manifest
    - Maven POM, Gemfile / bundle exec, etc.
  - No reliance on specific system tools
    - i.e. Linux tool not available on Windows

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

I. Codebase

One codebase tracked in SCM, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Configuration

Store config in the environment

- Configuration
  - Separate from the code
  - Also separate from the application
    - i.e. DB credentials, hostnames, passwords
    - Acid Test – could the codebase be made open source?
    - Internal wiring (i.e. Spring configuration) considered part of codebase.
  - Environment Variables recommended.

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

IV. Backing Services
Treat backing services as attached resources

V. Build, Release, Run
Strictly separate build and run stages

VI. Processes
Execute app as stateless processes

- Backing Services
  - Service consumed by app as part of normal operations
    - DB, Message Queues, SMTP servers
    - May be locally managed or third-party managed
  - Services should be treated as resources
    - Connected to via URL / configuration
    - Swappable (change in-memory DB for MySQL)

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

IV. Backing Services	V. Build, Release, Run	VI. Processes
Treat backing services as attached resources	Strictly separate build and run stages	Execute app as stateless processes

- Build, Release, Run
  - Build stage – converts codebase into build (version)
    - Including managed dependencies
  - Release stage – build + config = release
    - Ready to run
  - Run – Runs app in execution environment

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

IV. Backing Services	V. Build, Release, Run	VI. Processes
Treat backing services as attached resources	Strictly separate build and run stages	Execute app as stateless processes

- Processes
  - One or more discrete running processes
  - Stateless
    - Processes should not store internal state (HTTP Sessions)
  - Shared Nothing
    - Data needing to be shared should be persisted
  - Memory / local tmp storage considered volatile
  - Processes may intercommunicate via messaging / persistent storage

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

- Port Binding
  - App should not need a “container”
    - Java App Server, Apache HTTPD for PHP ...
    - PaaS now takes that role
  - Apps should export HTTP as a service
    - Define as a dependency (#2)
      - Tornado (Python), Thin (Ruby), embedded Jetty/Tomcat (Java)
    - Execute at runtime
  - One App can become another App's service (#4, #6)

http://www.grahambrooks.com/2014/04/07/container-less-web-applications.html

Pivotal

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

VII. Port binding	VIII. Concurrency	IX. Disposability
Export services via port binding	Scale out via the process model	Maximize robustness with fast startup and graceful shutdown

- Concurrency
  - Processes are first class citizens
    - Like Unix service daemons
    - Unlike Java threads
  - Individual processes are free to multithread
    - BUT a VM can only get so large (vertical scaling).
    - Must be able to span multiple machines (horizontal scaling)

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

- Disposability
  - Processes should be disposable
    - Remember, they're stateless!
  - Should be quick to start and stop
    - Should exit gracefully / finish current requests.
    - Or should be idempotent / reentrant
  - Enhances scalability and fault tolerance
  - Design *crash-only* software

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

X. Dev/prod parity

Keep dev, staging, prod as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin / mgmt tasks as one-off processes

- Development, Staging, Production should be similar
  - Dev / Prod environments often different
    - Tool gap – devs use SQLLite/Nginx, prod uses Apache/Oracle
    - Personnel gap – developers develop, admins deploy
    - Time gap - (development over weeks / months)
  - Keep differences minor
    - Reduce tool gap – use same software
    - Reduce time gap - small changes & continuous deployment
    - Reduce personnel gap - involve developers in deployment and monitoring

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

X. Dev/prod parity

Keep dev, staging, prod as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin / mgmt tasks as one-off processes

- Logs are streams of aggregated, time-ordered events
  - Apps are not concerned with log management
    - Just write to sysout.
  - Separate log managers handle management
    - Logging as a service
- Can be managed via tools like Papertrail, Splunk ...
  - Log indexing and analysis

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

X. Dev/prod parity

Keep dev, staging, prod as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin / mgmt tasks as one-off processes

- Admin Processes / Management Tasks Run as One-Off Processes.
  - DB Migrations, one time scripts, etc.
  - Use same environment, tools, language as application processes
    - REPL

Read–Eval–Print Loop = command-shell for running non-interactive shell scripts

© Copyright 2016 Pivotal. All rights reserved

12-Factor Application

I. Codebase

One codebase tracked in SCM, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Configuration

Store config in the environment

IV. Backing Services

Treat backing services as attached resources

V. Build, Release, Run

Strictly separate build and run stages

VI. Processes

Execute app as stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep dev, staging, prod as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin / mgmt tasks as one-off processes

Pivotal

© Copyright 2016 Pivotal. All rights reserved

Summary

- After completing this lesson, you should have learned:
  - Architectural design factors for building scalable applications in Cloud Foundry

© Copyright 2016 Pivotal. All rights reserved

Polyglot Persistence

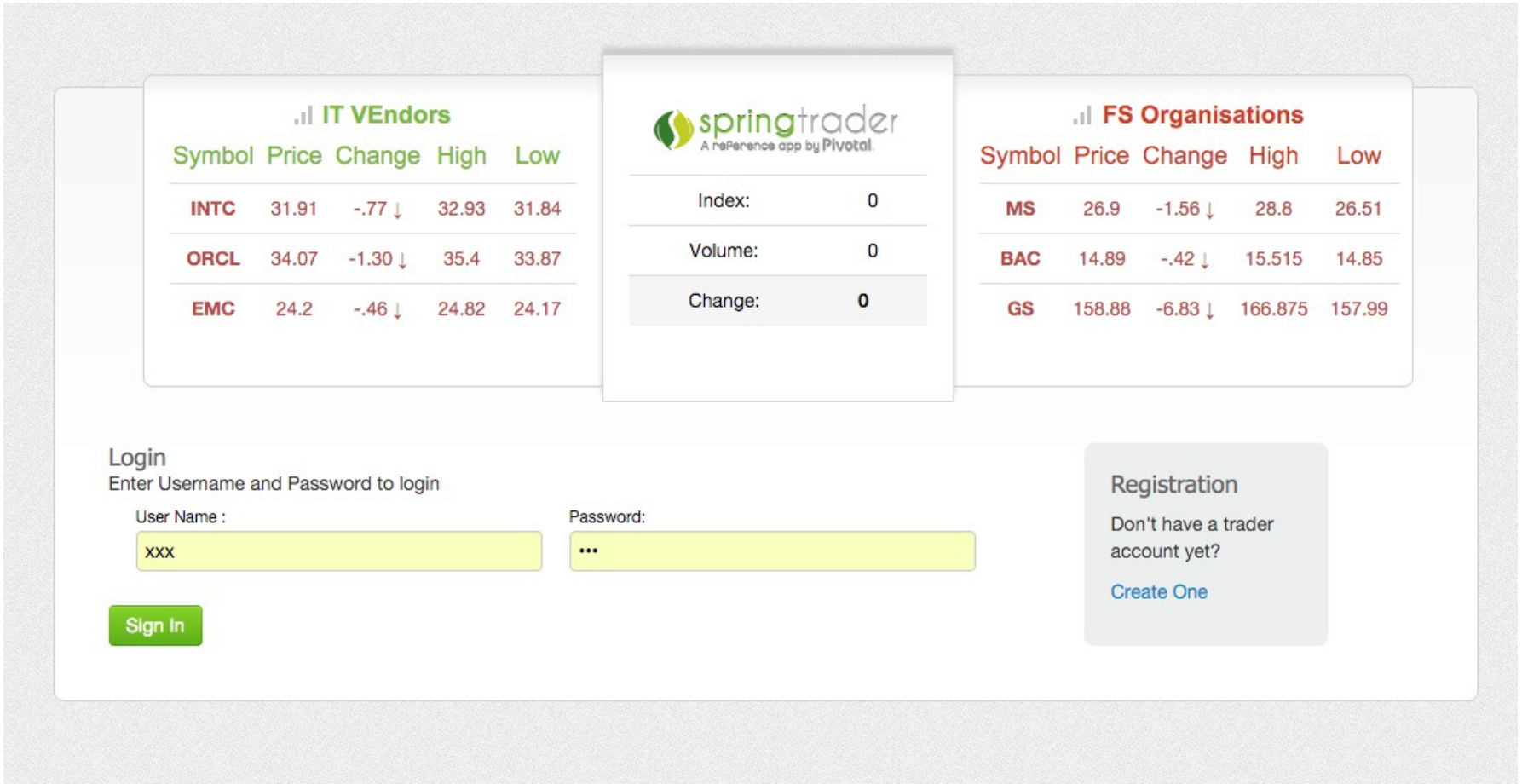
Different Databases for Different Services

Use the right tool for the job at hand

Pivotal

© Copyright 2016 Pivotal. All rights reserved

SpringTrader



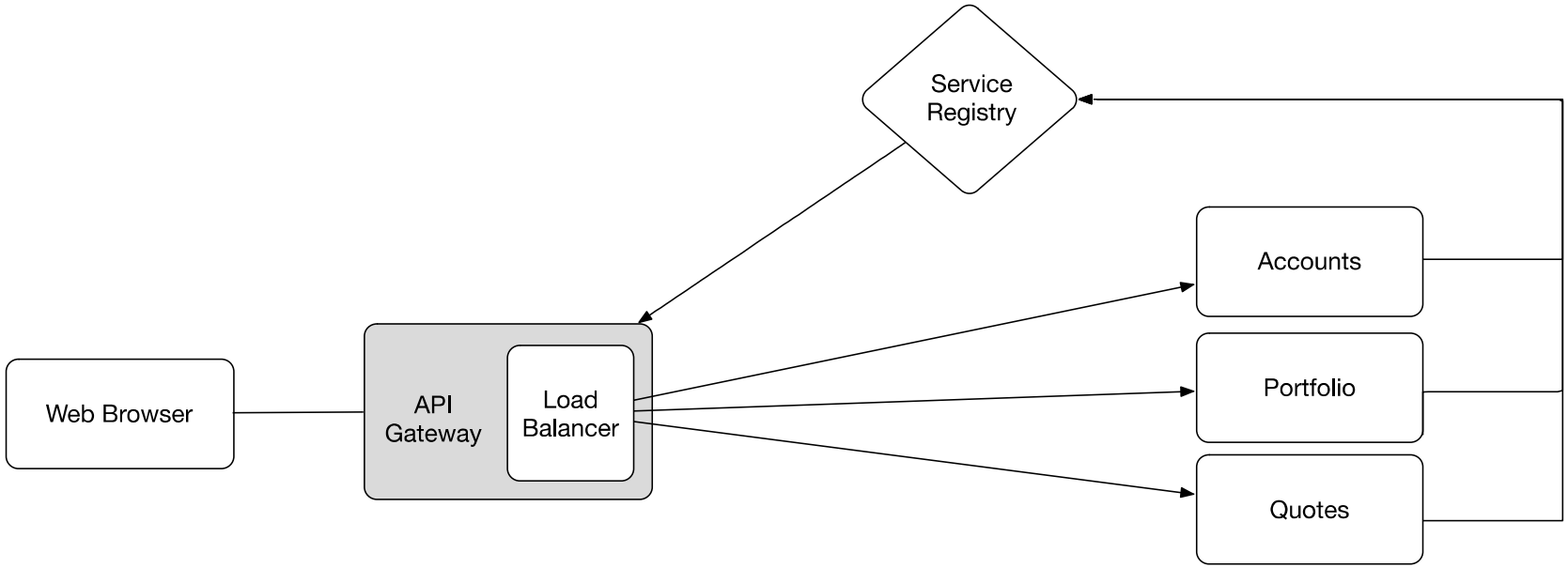
© Copyright 2016 Pivotal. All rights reserved

Imagine 4 Teams

- Quotes
- Accounts
- Portfolio
- Web UI

© Copyright 2016 Pivotal. All rights reserved

SpringTrader Cloud
High-Level Architecture



© Copyright 2016 Pivotal. All rights reserved

In this session

- Quotes (MongoDB)
- Accounts (MySQL/Postgres)
- Portfolio (MySQL/Postgres)
- All via Spring Data

© Copyright 2016 Pivotal. All rights reserved

Lab

Picking the best persistence

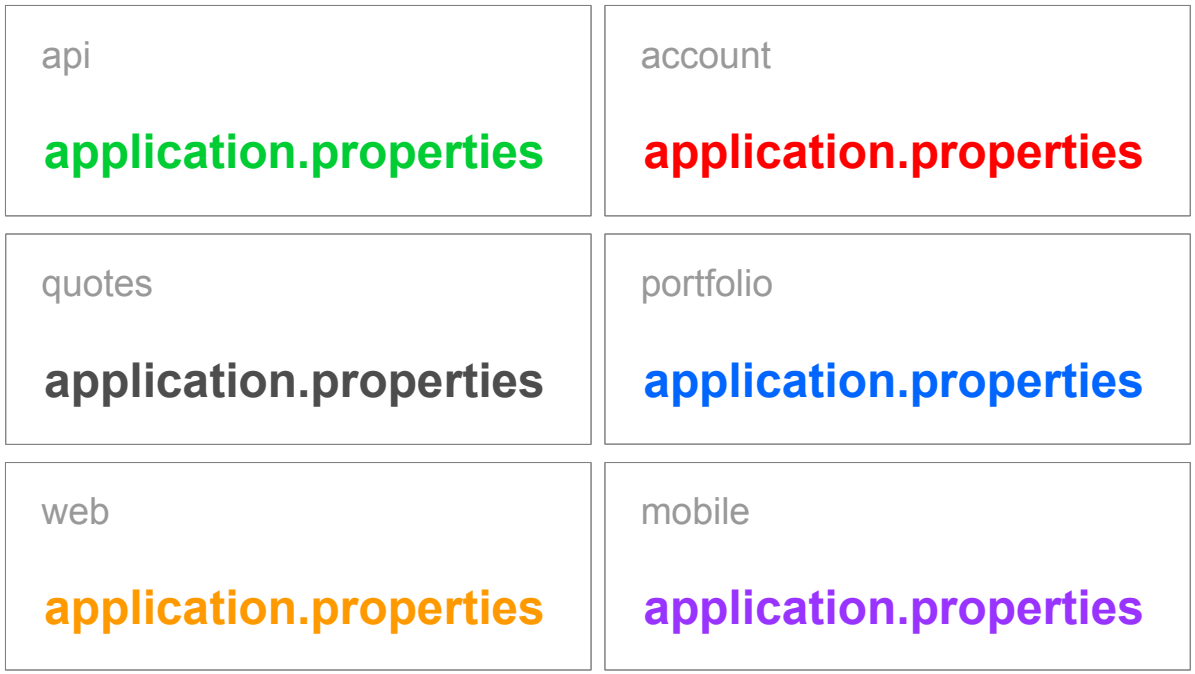
© Copyright 2016 Pivotal. All rights reserved

Cloud Native Architecture

Spring Cloud Config

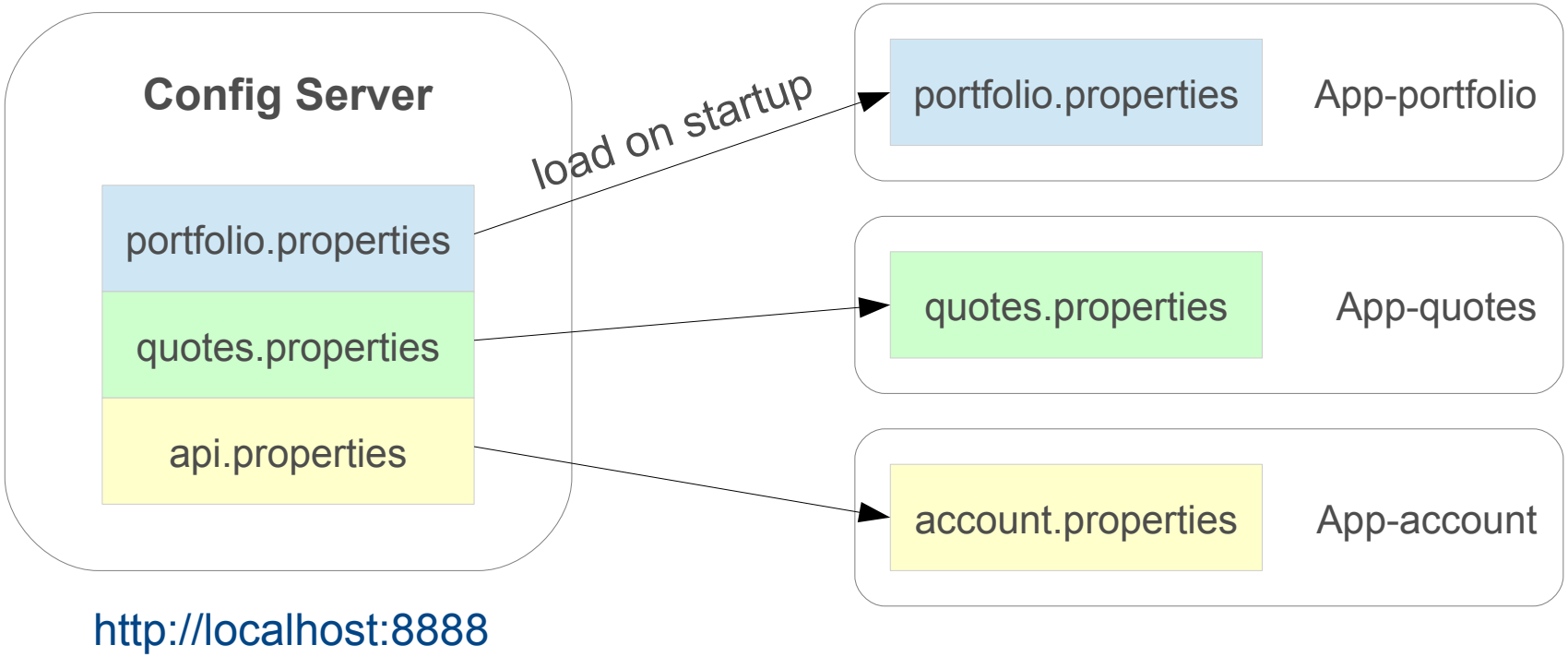
© Copyright 2016 Pivotal. All rights reserved

Microservices Configuration Forest



© Copyright 2016 Pivotal. All rights reserved

Centralize Config Management



© Copyright 2016 Pivotal. All rights reserved

Where to Store Config Files

- File System
  - Easy to startup, but no version control
- Git
  - Local repo, Remote repo, Github
- Svn

© Copyright 2016 Pivotal. All rights reserved

Setup Config Server

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

pom.xml

```
@SpringBootApplication
@EnableConfigServer
public class Application { ... }
```

application.java

```
server.port=8888
spring.cloud.config.server.native.searchLocations=file:../config/
spring.profiles.active=native
```

application.properties

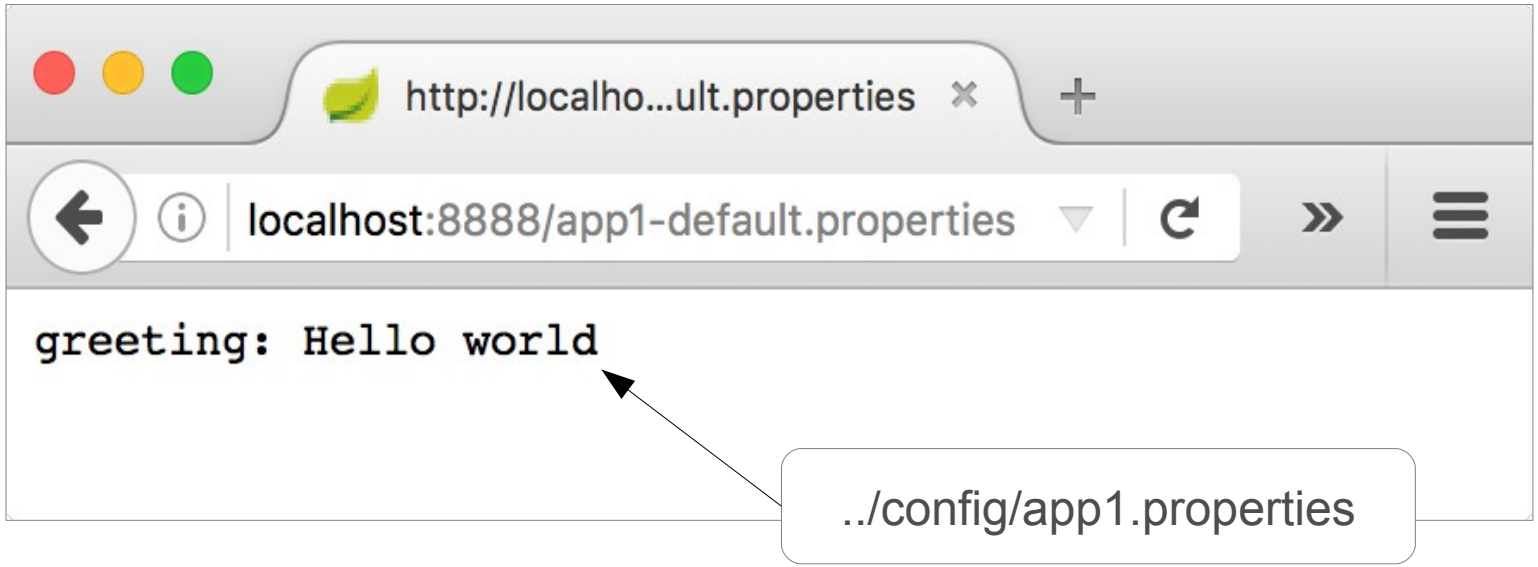
```
greeting=Hello world
```

*../config/**app1**.properties*

© Copyright 2016 Pivotal. All rights reserved

Testing Config Server

`http://localhost:8888/{app}-default.properties`



"default" is the profile name, profiles will be discussed later

© Copyright 2016 Pivotal. All rights reserved

Setup the Client App

```
<dependencies>
  <dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-config</artifactId>
  </dependency>
</dependencies>
```

pom.xml

```
spring.application.name=app1
spring.cloud.config.uri=http://localhost:8888
spring.cloud.config.failFast=true
```

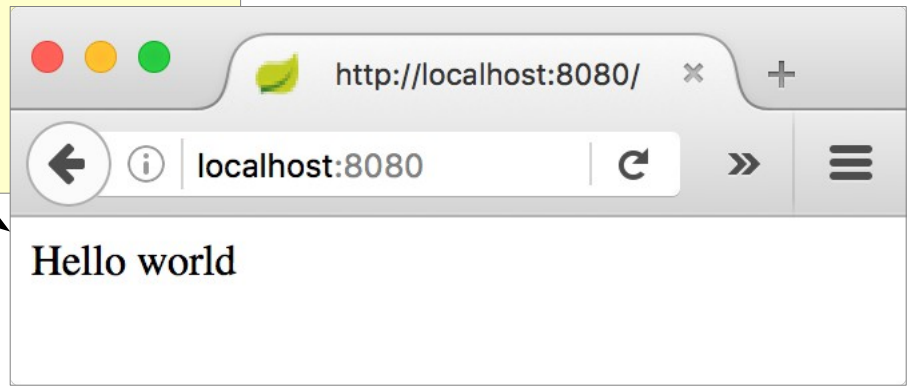
bootstrap.properties

© Copyright 2016 Pivotal. All rights reserved

Setup the Client App

```
@RestController
public class GreetingController {

    @Value("${greeting}")
    String greeting;

    @RequestMapping("/")
    public String hi() {
        return greeting;
    }
}
```

loaded from config server



© Copyright 2016 Pivotal. All rights reserved

Project Structure

```
client
      pom.xml
      bootstrap.properties
      src/main/java/client
          GreetingController.java
          ClientApplication.java
```

```
config
      app.properties
```

```
config-server
      pom.xml
      application.properties
      src/main/java/configserver
          ConfigServerApplication.java
```


© Copyright 2016 Pivotal. All rights reserved

Quiz

- What happens when the client app is started but config server is not running?
- Does config server always serve the latest config values?
- Does the client app reload the config at runtime?

© Copyright 2016 Pivotal. All rights reserved

Profiles

```
spring.cloud.config.uri=http://localhost:8888
spring.application.name=app1
spring.profiles.active=dev,jdbc
```

bootstrap.properties

- Config Files Loaded From Config Server
  - **app1**.properties
  - **app1-dev**.properties
  - **app1-jdbc**.properties

© Copyright 2016 Pivotal. All rights reserved

Merge Properties

Server app.properties

```
prop1=val-server
prop2=val-2-server
```

Local application.properties

```
prop1=val-local
prop3=val-3-local
```

Runtime view from App

```
prop1=val-server
prop2=val-2-server
prop3=val-3-local
```


© Copyright 2016 Pivotal. All rights reserved

Use Git Backend – Config Server

- Use Local Git Repo

```
spring.cloud.config.server.git.uri=file://var/my-config-repo
```

application.properties

- Use Github

```
spring.cloud.config.server.git.uri
   =git@github.com:pivotal-education/springtrader-config-repo.git
```

application.properties

© Copyright 2016 Pivotal. All rights reserved

Securing Your Config Server

- Config Server is not secured by default
- May be ok when working in a private network
  - Example: VPC
- Otherwise you need to secure it yourself
  - As you would do for any Spring application
  - Use Spring-Security

© Copyright 2016 Pivotal. All rights reserved

Summary

- Use cloud config to centralize config management
- Run config server with **`@EnableConfigServer`**
- Use **`bootstrap.properties`** to config client app
- Config Management
  - Split configurations using profiles
  - Resolve ambiguity for duplicated keys
- Secure your config server

© Copyright 2016 Pivotal. All rights reserved

Lab

Setup and use the config-server

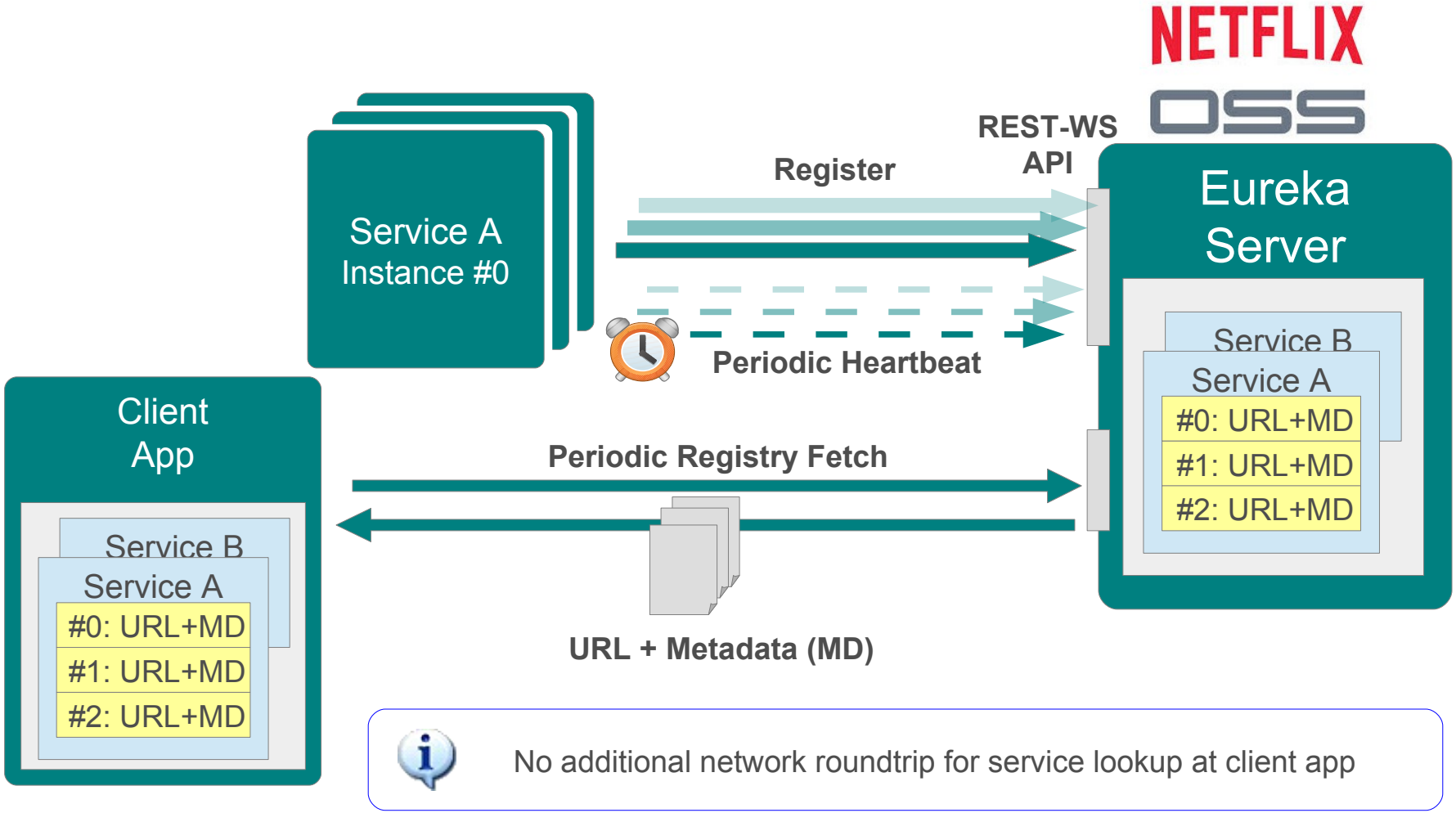
© Copyright 2016 Pivotal. All rights reserved

Service Registration and Discovery

Spring Cloud Eureka

© Copyright 2016 Pivotal. All rights reserved

Eureka Server
Distributed Architecture



© Copyright 2016 Pivotal. All rights reserved

Eureka Server - Dependency

- Maven example

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-eureka-server</artifactId>
    </dependency>
</dependencies>
```

pom.xml

© Copyright 2016 Pivotal. All rights reserved

Eureka Server – Standalone

- Start Eureka Standalone Server – w/o Peers

```
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
```

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApp {

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

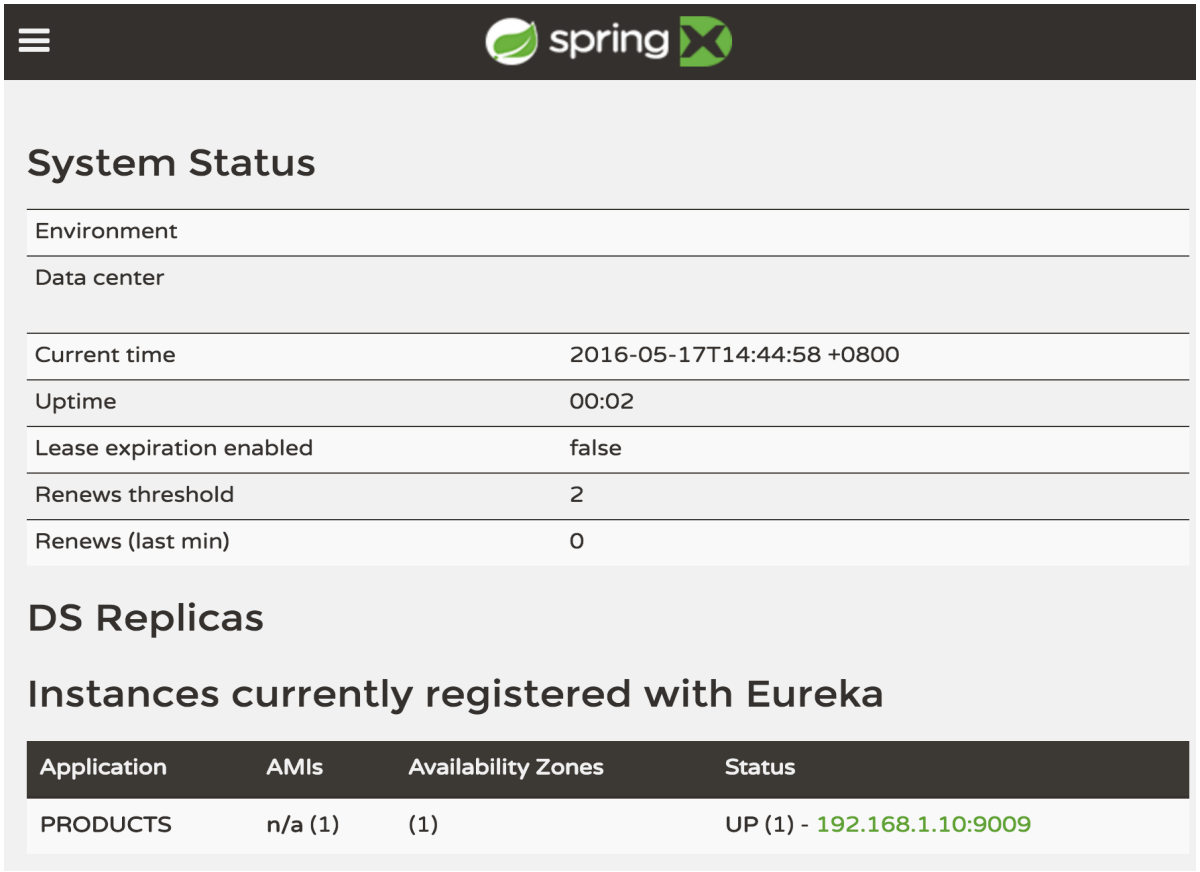
application.yml

```
eureka:
  client:
    registerWithEureka: false
    fetchRegistry: false
```


© Copyright 2016 Pivotal. All rights reserved

Eureka Server – Dashboard

`http://localhost:8761/`



© Copyright 2016 Pivotal. All rights reserved

Eureka – Client Responsibility

- Service Registration with Eureka Server
- Service Discovery
  - replicating the service registry into the client

© Copyright 2016 Pivotal. All rights reserved

Eureka – Client Service Registration

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>
</dependencies>
```

pom.xml

```
@SpringBootApplication
@EnableEurekaClient
public class MyEurekaClientApp {
    public static void main(String[] args) {
        SpringApplication.run(MyEurekaClientApp.class);
    }
}
```

MyEurekaClientApp.java

© Copyright 2016 Pivotal. All rights reserved

Eureka – Service Registration
Eureka Client Configuration

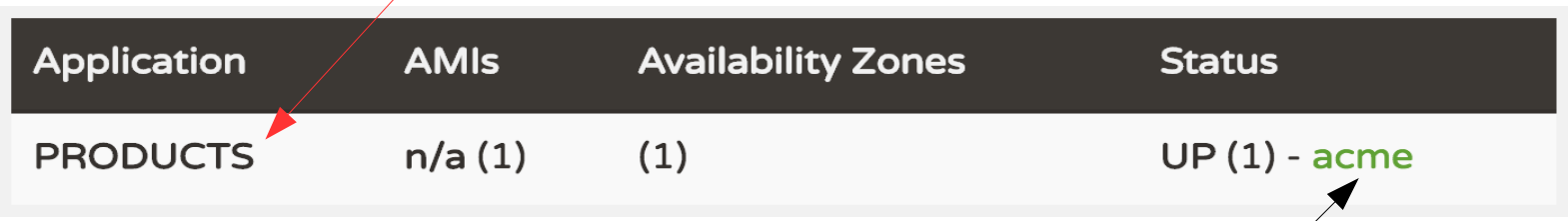
- Eureka Single Server Coordinates Configuration

```
eureka:                                                    application.yml
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

© Copyright 2016 Pivotal. All rights reserved

Eureka – App Name & Instance Id

Uses `${spring.application.name}` by default
Can be overridden by `${eureka.instance.appname}`



Application	AMIs	Availability Zones	Status
PRODUCTS	n/a (1)	(1)	UP (1) - acme

- Default server's hostname
- can be overridden by `${eureka.instance.preferIpAddress}`

© Copyright 2016 Pivotal. All rights reserved

Eureka – Service Lookup
Resolving Service Names – Spring Cloud API

- Service Instance URI Lookup – Generic API

```
import org.springframework.cloud.netflix.eureka.client.EnableDiscoveryClient;
import org.springframework.cloud.client.discovery.DiscoveryClient;
```

```
@SpringBootApplication
@EnableDiscoveryClient
public class MyClient {

    @Autowired    // Spring creates this automatically for you
    private DiscoveryClient discoveryClient;

    public URI getServiceUrl() {
        List<ServiceInstance> list =
            discoveryClient.getInstances("products");
        return list.get(0).getUri();
    }
}
```


© Copyright 2016 Pivotal. All rights reserved

Lab

Using a Service Registry

© Copyright 2016 Pivotal. All rights reserved

Cloud Native Architecture - I I

Spring Cloud – Ribbon, Hystrix, Feign

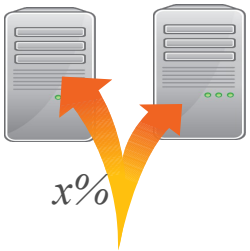
© Copyright 2016 Pivotal. All rights reserved

Roadmap

- **Loadbalancing: Ribbon**
- Circuit Breaker: Hystrix
- REST Client: Feign

© Copyright 2016 Pivotal. All rights reserved

Load-Balancing & Routing
Challenges

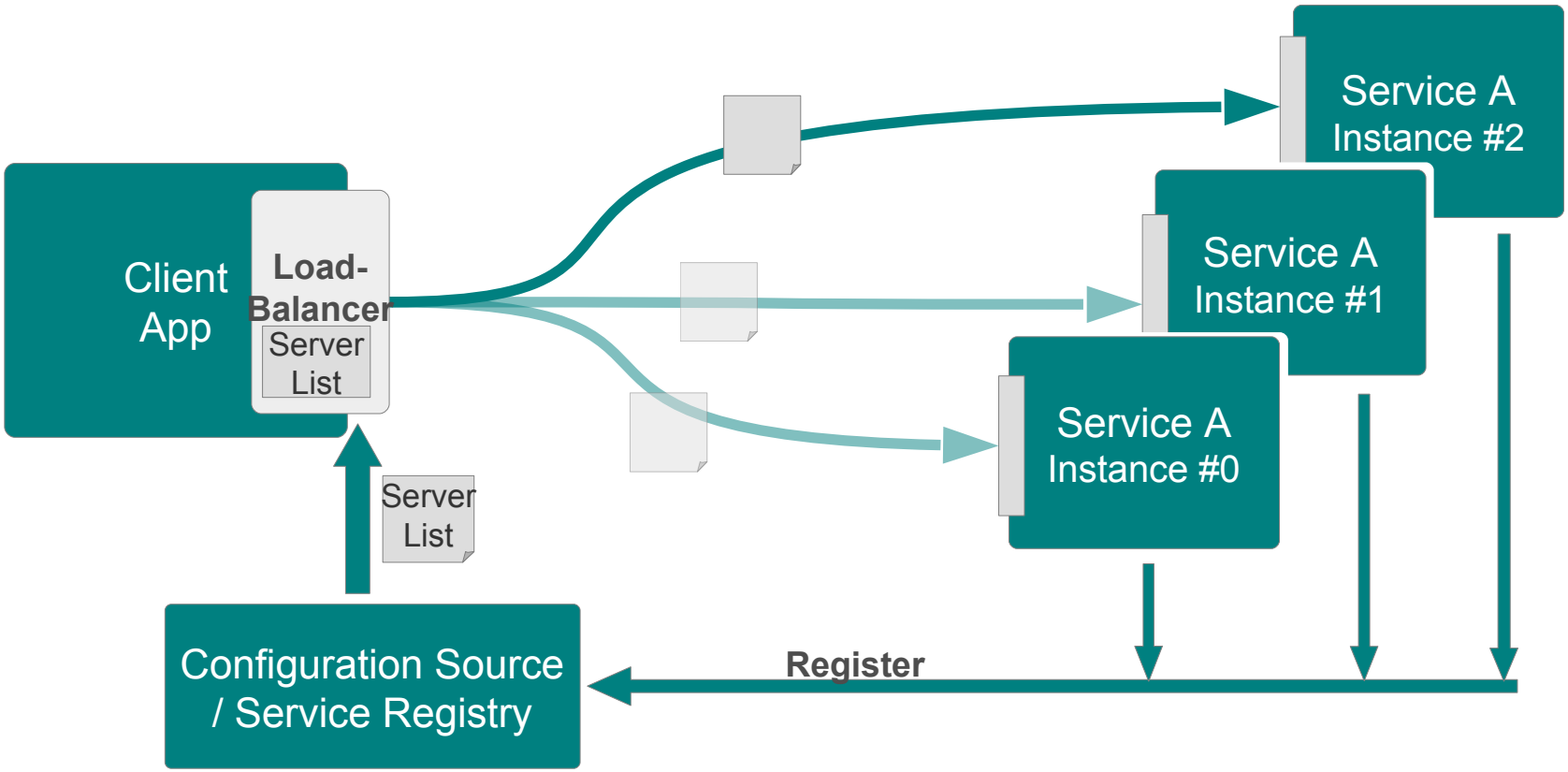


- **Challenge I** – Load-Balancing:
  - How to distribute load across multiple (functional equivalent) *service-instances* ?
- **Challenge II** – Routing:
  - What simple configuration & programming model to use to distribute work across multiple (functional dissimilar) *services* ?
- **Challenge III** – Load-Balancing & Routing:
  - How to seamlessly integrate Load-Balancing & Routing in a coherent architecture ?

© Copyright 2016 Pivotal. All rights reserved

Client-Side Load-Balancing

Generic Distributed Architecture



© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Enabling & Service Instance Resolution

- Access Load-Balanced Service

```
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
```

```
@Service
public class RemoteProductsService implements ProductsService {
    @Autowired
    private LoadBalancerClient loadBalancer;

    public static List<Map<?,?>> getProducts() {
        ServiceInstance instance = loadBalancer.choose("products");
        String uri = String.format("http://%s:%s/product",
            instance.getHost(), instance.getPort());
        return new RestTemplate().getForObject(uri);
    }
}
```

```
@RibbonClient
public class AppConfig {
}
```


© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Dependencies

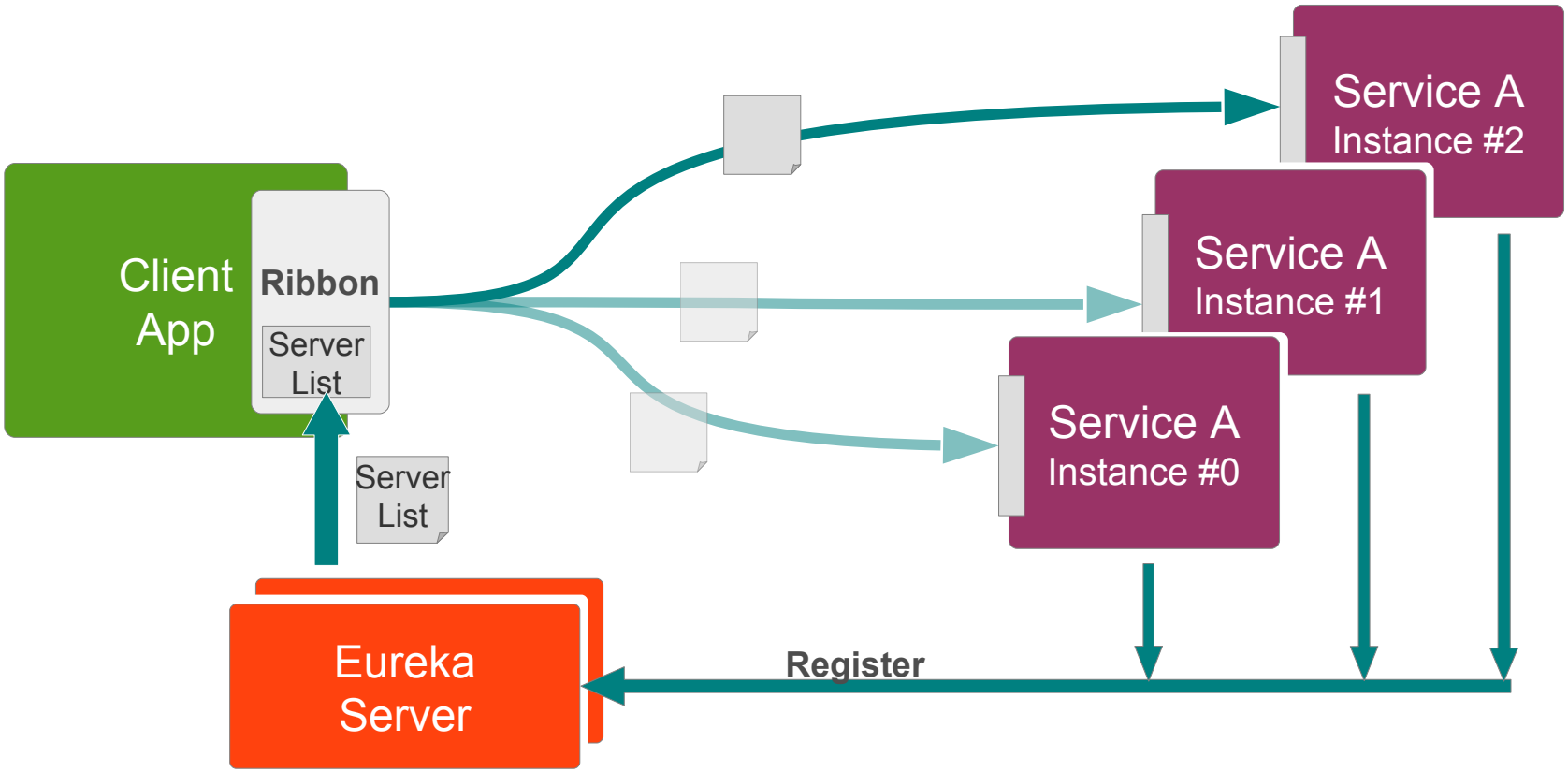
- Dependencies

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-ribbon</artifactId>
    </dependency>
</dependencies>
```


© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon

Eureka Provided Configuration – Architecture



© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Configuration w/ Eureka Integration

- Load-Balancing Eureka Registered Service Instances

application.yml

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

- Maven Dependencies

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```


© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Ribbon Global Client Configuration

- Global Ribbon Configuration – Failure-Detection Rule

```
@Configuration
public class ClientConfig {
    @Bean
    public IPing ribbonPing(IClientConfig config) {
        return new PingUrl();
    }
}
```

- Client with Global Configuration

```
@SpringBootApplication
@RibbonClient(name="products")
public class MyClientApp {
    …
}
```


© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Ribbon Client Specific Configuration

- Client Specific Configuration – Failure-Detection Rule

```
@SpringBootApplication
@RibbonClient(name="products", configuration=ProductsClientConfig.class)
public class MyClientApp { … }
```

```
public class ProductsClientConfig {
    public IPing ribbonPing(IClientConfig config) {
        return new PingUrl();
    }
}
```

- Multi-Client Combined Configuration

```
@SpringBootApplication
@RibbonClients({
   @RibbonClient(name="products", configuration=ProductsClientConfig.class),
   @RibbonClient(name="orders")
})
public class MyClientApp { … }
```


© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Ribbon Client Configuration Strategies - 1

- Strategy Beans

Bean Type	Default Implementation	Description
IClientConfig	DefaultClientConfigImpl	General configuration
IPing	NoOpPing	Failure Detection
ServerList<Server>	ConfigurationBasedServerList	Service Instance List
ServerListFilter<Server>	ZonePreferenceServerListFilter	Filtered Server List
IRule	ZoneAvoidanceRule	
ILoadBalancer	ZoneAwareLoadBalancer	Service instance selection

© Copyright 2016 Pivotal. All rights reserved

Load-Balancing w/ Ribbon
Ribbon Client Configuration Strategies - 2

- Eureka Integration Specific Configuration

Bean Type	Eureka Implementation	Description
`IClientConfig`	`DefaultClientConfigImpl`	General configuration
`IPing`	`NIWSDiscoveryPing`	Check Status in Eureka Cache
`ServerList<Server>`	`DiscoveryEnabledNIWSServerList`	Lookup in Eureka
`ServerListFilter <Server>`	`ZonePreferenceServerListFilter`	Prefer Server in Zone (or Domain)
`IRule`	`ZoneAvoidanceRule`	
`ILoadBalancer`	`ZoneAwareLoadBalancer`	

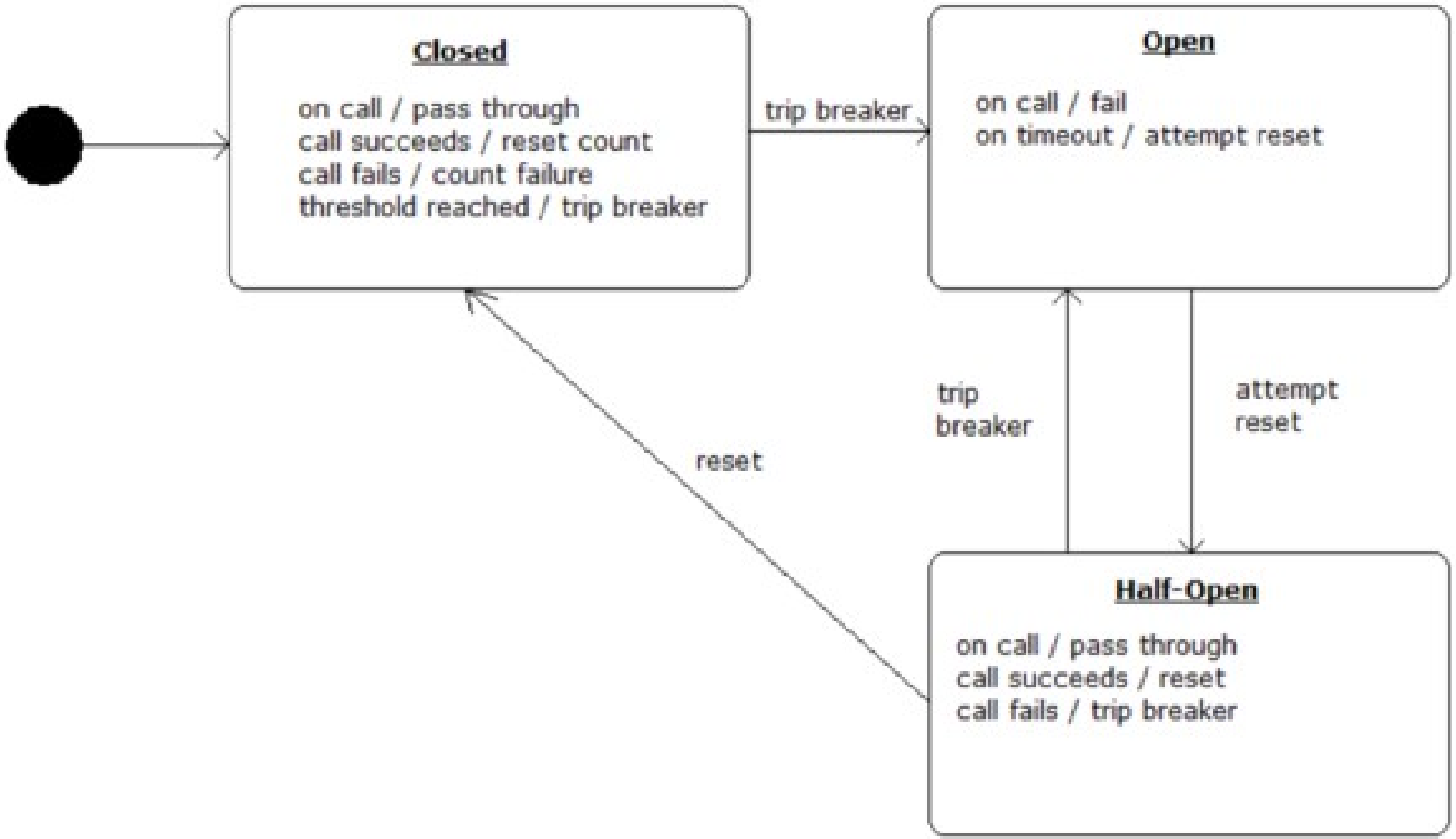
© Copyright 2016 Pivotal. All rights reserved

Roadmap

- Loadbalancing: Ribbon
- **Circuit Breaker: Hystrix**
- REST Client: Feign

© Copyright 2016 Pivotal. All rights reserved

Circuit Breaker Pattern



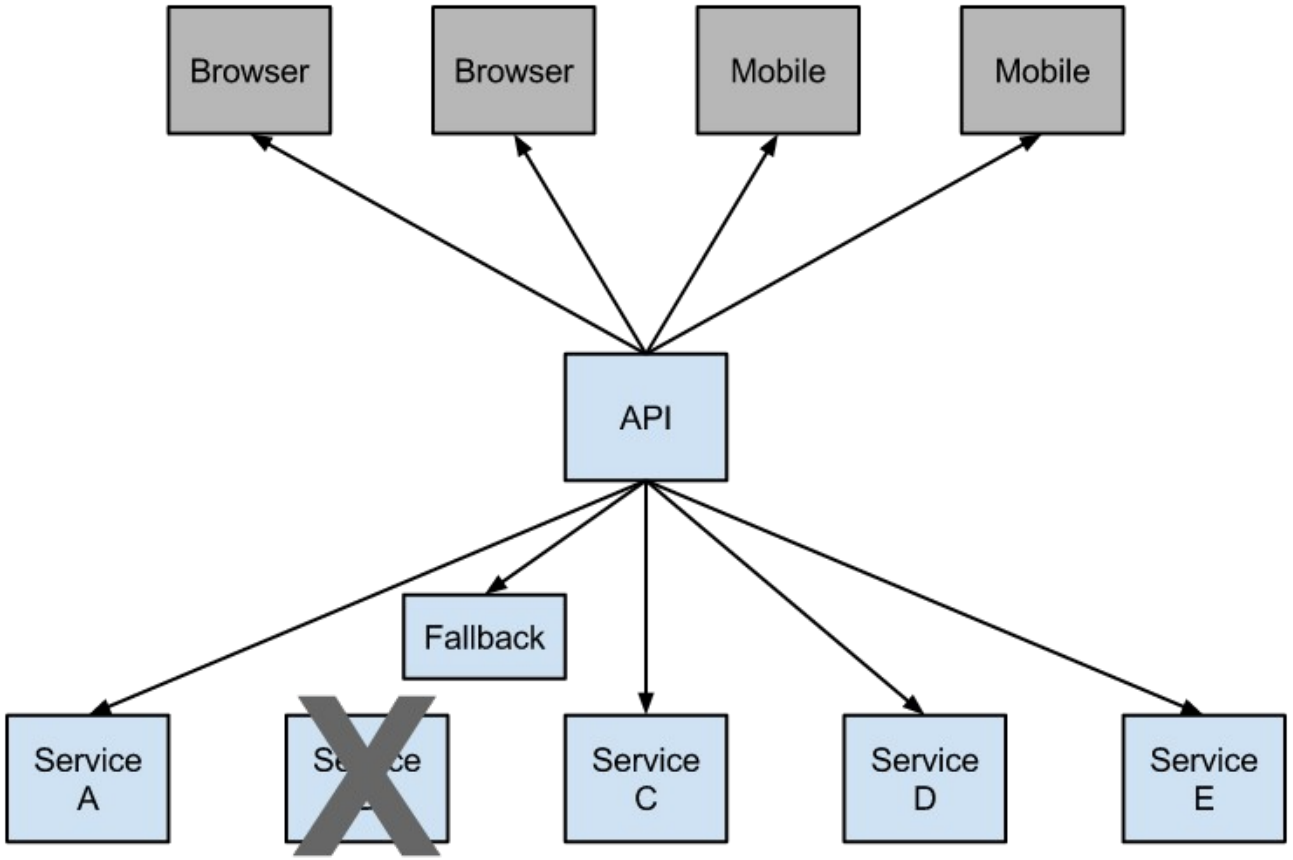
© Copyright 2016 Pivotal. All rights reserved

Circuit Breaker: Hystrix

- Latency and fault tolerance
- Isolates access to other services
- Stops cascading failures
- Enables resilience
- Circuit breaker pattern
- Dashboard

© Copyright 2016 Pivotal. All rights reserved

Hystrix Fallback



© Copyright 2016 Pivotal. All rights reserved

Declarative Hystrix

- Programmatic access is cumbersome
- `@HystrixCommand` to the rescue
- `@EnableHystrix` via starter pom
- Wires up spring aop aspect

© Copyright 2016 Pivotal. All rights reserved

Fallback: Synchronous

```
...

@HystrixCommand(fallbackMethod = "getProducerFallback")
public ProducerResponse getValue() {
   return restTemplate.getForObject("http://producer", ProducerResponse.class);
}

private ProducerResponse getProducerFallback() {
   return new ProducerResponse(42);
}

...
```

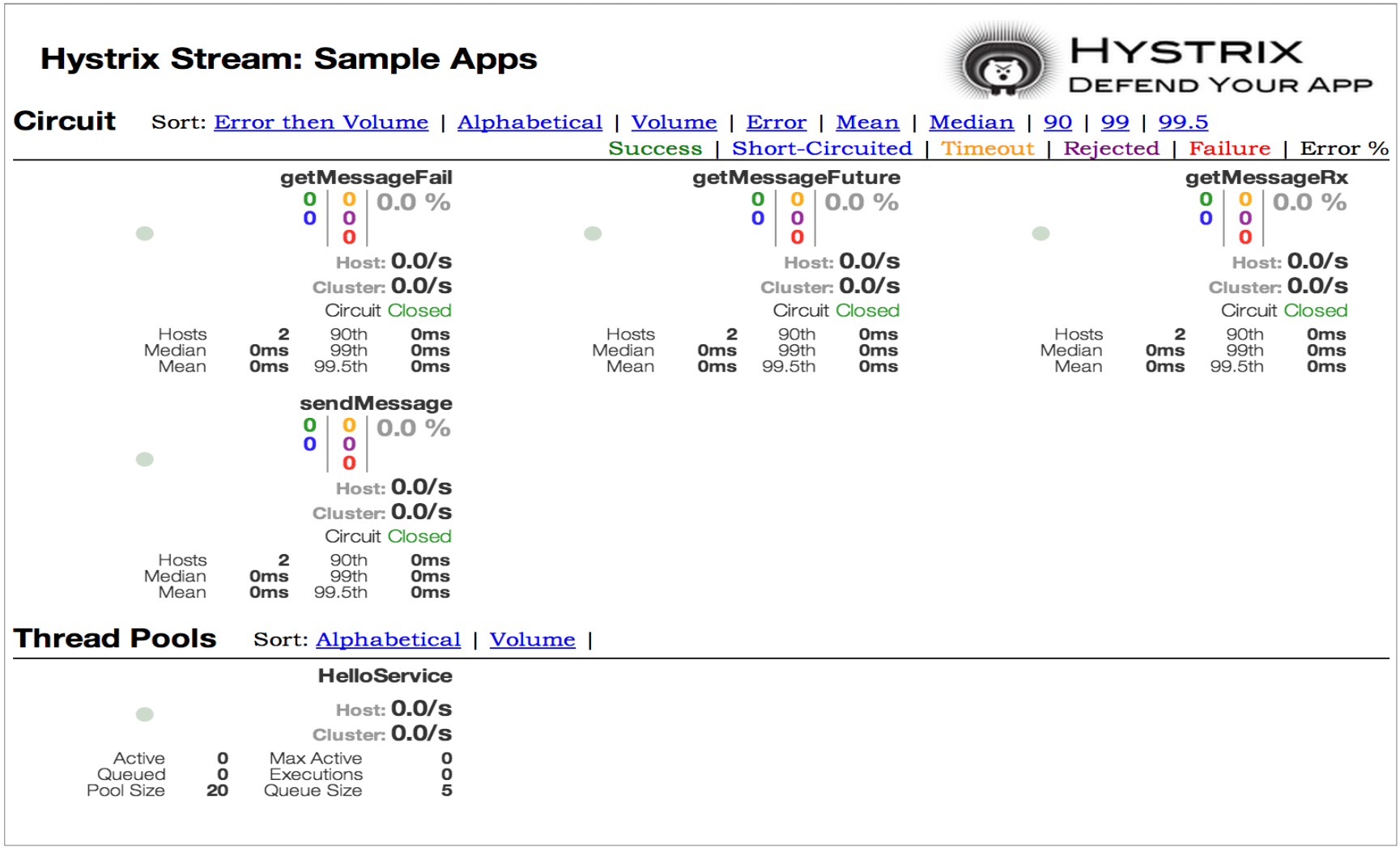

© Copyright 2016 Pivotal. All rights reserved

Ribbon vs Hystrix

- Ribbon removes the **dead** node (with Eureka's help)
- Hystrix deals with REST call errors when node is still UP
  - Service database constrain violation
- Hystrix error recovery is more general (doesn't need to be REST call)
  - Local database connection is down
  - Message queue is down

© Copyright 2016 Pivotal. All rights reserved

Hystrix Dashboard



© Copyright 2016 Pivotal. All rights reserved

Roadmap

- Loadbalancing: Ribbon
- Circuit Breaker: Hystrix
- **REST Client: Feign**

© Copyright 2016 Pivotal. All rights reserved

Declarative REST-WS with Feign

REST-WS Defined as Java Interface

Ribbon Client Binding & Spring MVC Annotated Interface

```
@FeignClient("products")
public interface ProductService {

    @RequestMapping(value="/product", method=RequestMethod.GET)
    List<Map> getProducts();

    @RequestMapping(value="/product", method=RequestMethod.POST)
    void newProduct(Map<String,Object> product);

}
```

Hard-Coded Static Server URL

```
@FeignClient(url="http://products.myorg.org")
public interface MyService {
    …
}
```


© Copyright 2016 Pivotal. All rights reserved

Declarative REST-WS with Feign

REST-WS Defined as Java Interface

Enable Feign Proxy Creation

```
@SpringBootApplication
@EnableFeignClients
public class MyClientApp {
    …
}
```

Proxy Injection & Service Invocation

```
@Controller
public class ProductController {
    @Autowired
    private ProductService service;

    @RequestMapping(value="/catalog")
    public static void List<Map> listProducts() {
        return service.getProducts();
    }
}
```

PROXY

© Copyright 2016 Pivotal. All rights reserved

Labs!

© Copyright 2016 Pivotal. All rights reserved

Microservice Security

OAuth2 & Authorization Server

© Copyright 2016 Pivotal. All rights reserved

OAuth2 - Terminologies

- Resource Owner
  - typically a person
- Client
  - apps operating on behave of the resource owner (person)
  - apps running inside browser, server, desktop, mobile
- Authorization Server
  - authenticate resource owner (typically by username password)
  - issuing **access token**
- Resource Server
  - guarding resource owner's info based on **access token**

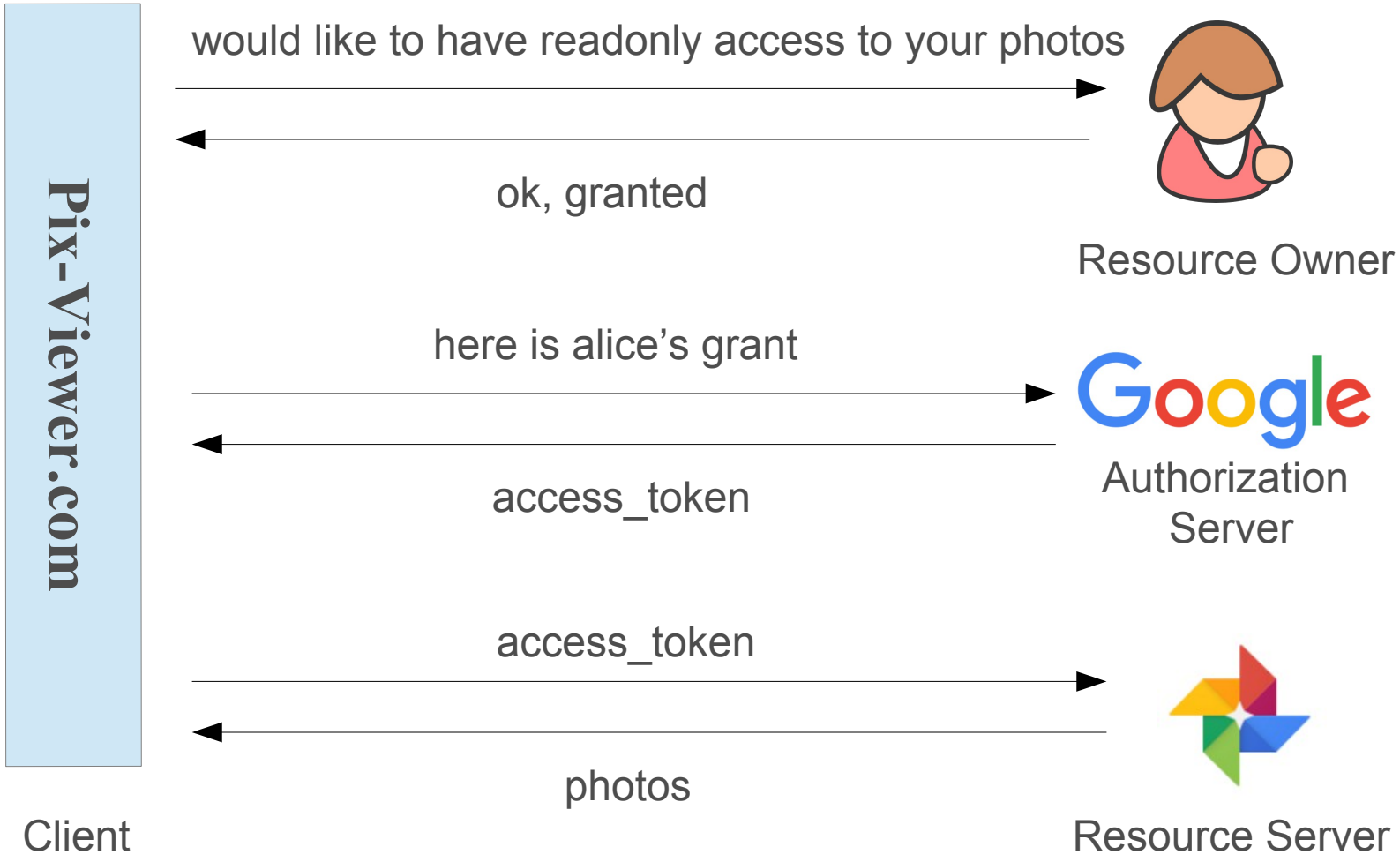
© Copyright 2016 Pivotal. All rights reserved

OAuth2 - Why

- Alice has photos in her Google Photos
- A new hipster app www.pix-viewer.com
- Alice wants to use the site to see her photos
- But
  - She doesn’t want to share her Google login credential
  - She is only willing to grant readonly access

© Copyright 2016 Pivotal. All rights reserved

OAuth2 – Protocol



© Copyright 2016 Pivotal. All rights reserved

Spring Cloud Security

- All three components are supported
  - Authorization Server
  - Resource Server
  - Client

© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
</dependencies>
```

pom.xml

© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
@SpringBootApplication
@EnableAuthorizationServer
public class AuthServerApp {

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

AuthServerApp.java

Several Controllers are added automatically

- **`/oauth/token`** for issuing access token
- **`/oauth/check_token`** for checking token

© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
@Configuration
public class AuthenticationManagerConfiguration
        extends GlobalAuthenticationConfigurerAdapter {

    @Override
    public void init(AuthenticationManagerBuilder auth)
        throws Exception {
        auth.inMemoryAuthentication()...
    }

}
```

This tells Authorization Server where to find Resource Owner's login credentials

© Copyright 2016 Pivotal. All rights reserved

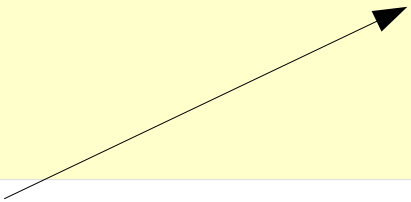
Setup Authorization Server

```
@Configuration
class OAuth2Config extends AuthorizationServerConfigurerAdapter {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
            throws Exception {
        endpoints.authenticationManager(authenticationManager);
    }

    @Override
    public void configure(AuthorizationServerSecurityConfigurer server)
            throws Exception {
        oauthServer.checkTokenAccess("permitAll()");
    }
}
```


© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
@Configuration
class OAuth2Config extends AuthorizationServerConfigurerAdapter {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
            throws Exception {
        endpoints.authenticationManager(authenticationManager);
    }
    // ...
}
```



AuthenticationManager used by password grant

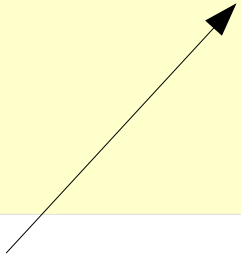
There are more grant types, refer to OAuth2 RFC

https://tools.ietf.org/html/rfc6749

© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
@Configuration
class OAuth2Config extends AuthorizationServerConfigurerAdapter {

    // ...

    @Override
    public void configure(ClientDetailsServiceConfigurer clients)
            throws Exception {
        clients.inMemory()
                .withClient("acme")
                .secret("acmesecret")
                .authorizedGrantTypes("authorization_code", "refresh_token",
                        "password").scopes("openid");
    }
}
```

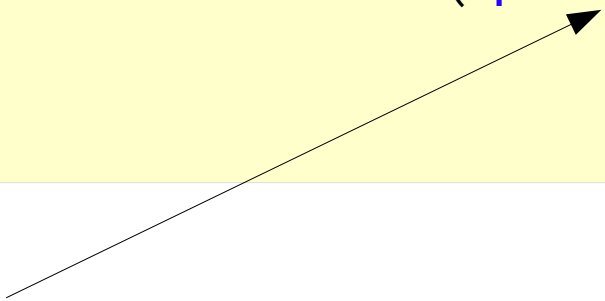


A single client is configured here

© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
@Configuration
class OAuth2Config extends AuthorizationServerConfigurerAdapter {

    // ...

    @Override
    public void configure(AuthorizationServerSecurityConfigurer server)
            throws Exception {
        oauthServer.checkTokenAccess("permitAll()");
    }
}
```



/oauth/check_token endpoint is configured to rejectAll by default

© Copyright 2016 Pivotal. All rights reserved

Setup Authorization Server

```
server.context-path=/auth-server
```

application.properties

This step is necessary if we are going to run the authorization server and Client on the same host, otherwise browsers may be confused about the JSESSIONID cookie

© Copyright 2016 Pivotal. All rights reserved

Test Authorization Server

```
curl http://localhost:8000/auth-server/oauth/token \
    -u acme:acmesecret \
    -d grant_type=password \
    -d username=mstine \
    -d password=secret
```

```
{
  "access_token":"661aac97-55ca-49a0-b8b6-a4a1d8cb63de",
  "token_type":"bearer",
  "refresh_token":"9a605803-4013-4818-ae24-22de7b399018",
  "expires_in":43199,
  "scope":"openid"
}
```


© Copyright 2016 Pivotal. All rights reserved

Lab

© Copyright 2016 Pivotal. All rights reserved

Microservice Security

OAuth2 Resource Server

© Copyright 2016 Pivotal. All rights reserved

Maven Dependency

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
</dependencies>
```

pom.xml

© Copyright 2016 Pivotal. All rights reserved

Setup Resource Server

```
@SpringBootApplication
@EnableResourceServer
public class ResourceServerApp {

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

ResourceServerApp.java

© Copyright 2016 Pivotal. All rights reserved

Create a Simple Rest Controller

```
@RestController
public class GreetingController  {

    @RequestMapping("/greeting")
    public Object greeting() {
        return Collections.singletonMap("content", "Hello user");
    }

}
```

GreetingController.java

© Copyright 2016 Pivotal. All rights reserved

Config Token Validation

```
security:
  oauth2:
    resource:
      token-info-uri: http://localhost:8000/auth-server/oauth/check_token
    client:
      client-id: acme
      client-secret: acmesecret
```

application.yml

© Copyright 2016 Pivotal. All rights reserved

Test

- Obtain an access token (refer to prev session)
- Send the access token via http Authorization header

```
curl -H "Authorization: bearer 5d04666c-11c2-4a4c-9dae-f0def0acc9c1" \
        http://localhost:8001/greeting
```


© Copyright 2016 Pivotal. All rights reserved

Lab

© Copyright 2016 Pivotal. All rights reserved

Microservice Security

OAuth2 Client

© Copyright 2016 Pivotal. All rights reserved

Maven Dependency

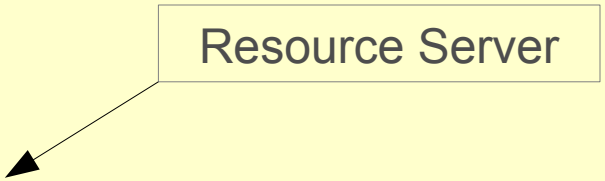
```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
</dependencies>
```

pom.xml

© Copyright 2016 Pivotal. All rights reserved

Setup Client Application

```
@SpringBootApplication
@EnableOAuth2Client
@RestController
public class ClientApplication {
    @Autowired
    private OAuth2RestOperations restTemplate;

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

    @RequestMapping("/")
    public String home() {
        return restTemplate
            .getForObject("http://localhost:8001/greeting", String.class);
    }
}
```



ResourceServerApp.java

© Copyright 2016 Pivotal. All rights reserved

Config Client

```
security:
  oauth2:
    client:
      accessTokenUri: http://localhost:8000/auth-server/oauth/token
      userAuthorizationUri: http://localhost:8000/auth-server/oauth/authorize
      clientId: acme
      clientSecret: acmesecret
      scope: openid
      clientAuthenticationScheme: header
    sso:
      login-path: /login
  basic:
    enabled: false
```

Authorization Server

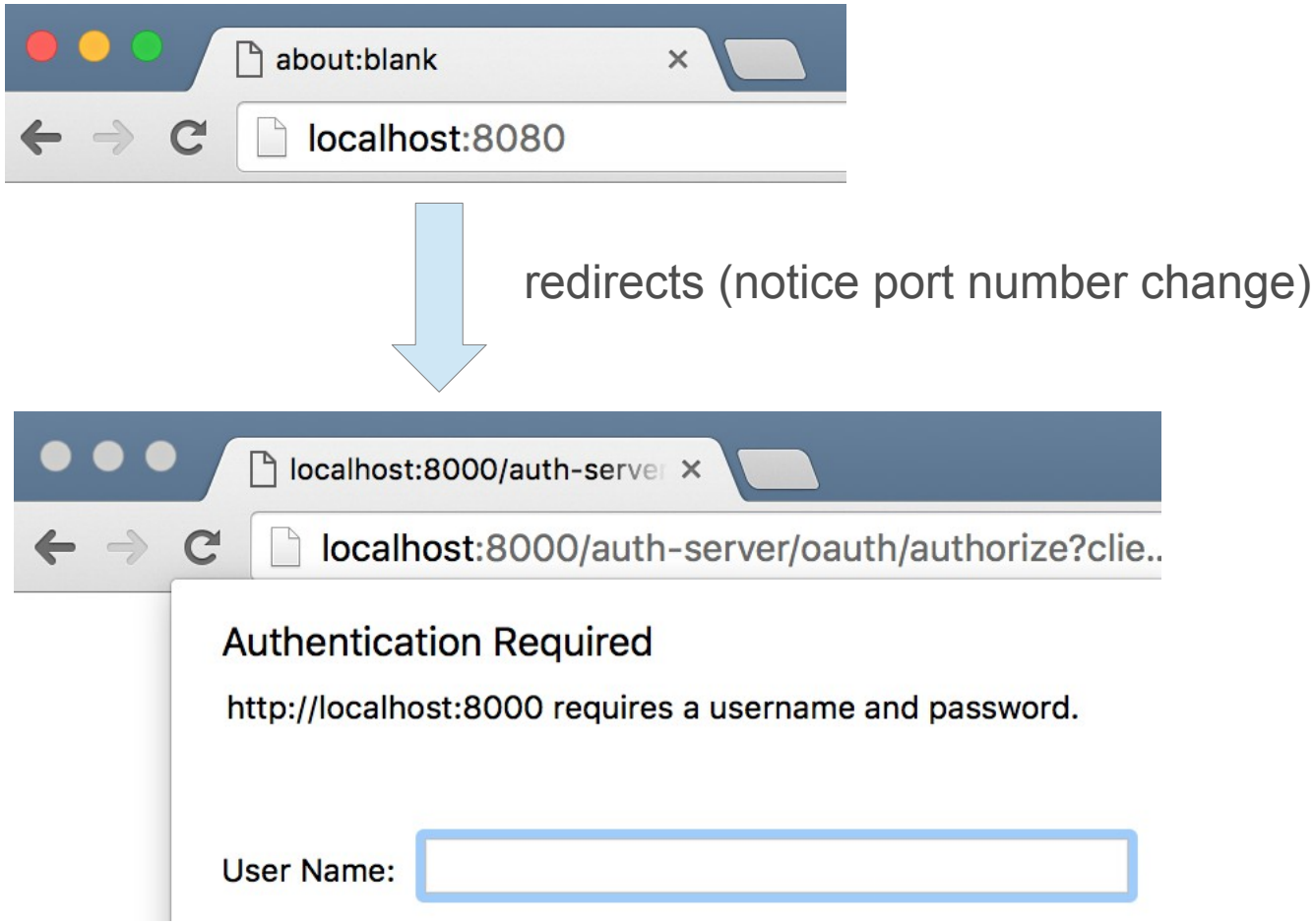
Client Info

No need to do local authentication

application.yml

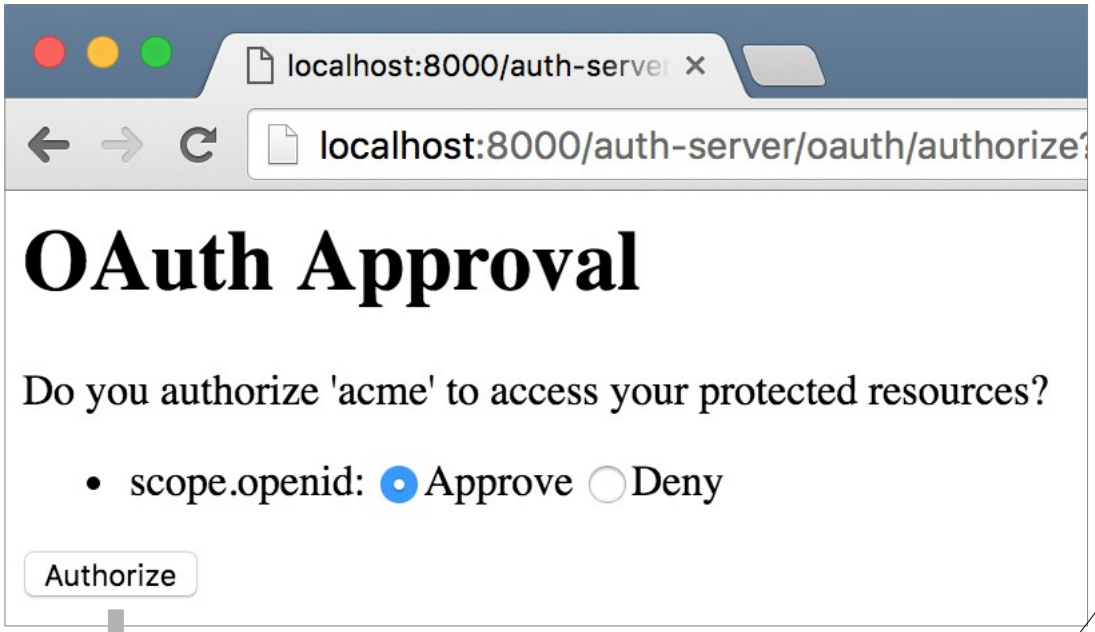
© Copyright 2016 Pivotal. All rights reserved

Test



© Copyright 2016 Pivotal. All rights reserved

Test



redirects back to client app

Content here is read from the resource server

© Copyright 2016 Pivotal. All rights reserved

Lab

© Copyright 2016 Pivotal. All rights reserved

Finishing Up

Course Completed

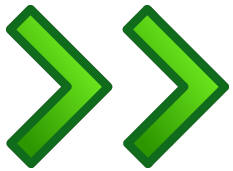
What's Next?

© Copyright 2016 Pivotal. All rights reserved

What's Next

- Congratulations, you've finished this course

- What do do next?
  - Other courses
  - Resources
  - Evaluation



© Copyright 2016 Pivotal. All rights reserved

Other courses

- Many courses available
  - PCF Immersion
  - Developing Applications with Cloud Foundry
  - Core Spring
  - Spring Web
  - Spring Enterprise
- More details here:
  - http://www.pivotal.io/training

© Copyright 2016 Pivotal. All rights reserved

Core Spring

Pivotal CERTIFIED Spring Professional

- Four-day workshop
- Complete Spring Overview
  - Spring Configuration and Spring Boot
  - AOP and Testing
  - Data Management, Transactions, JDBC, JPA
  - Web Applications using Spring MVC (including REST)
  - Security of Web applications
  - Implementing Microservices
- Spring Professional certification

© Copyright 2016 Pivotal. All rights reserved

Spring Web

Pivotal CERTIFIED Spring Web Application Developer

- Four-day workshop
- Making the most of Spring in the web layer
  - Spring MVC
  - REST using MVC and AJAX
  - Security of Web applications
  - Mock MVC testing framework
  - Spring Boot and Web Sockets
- Spring Web Application Developer certification

© Copyright 2016 Pivotal. All rights reserved

Spring Enterprise

Pivotal CERTIFIED Spring Enterprise Integration Specialist

- Building loosely coupled,event-driven architectures
  - Separate processing, communications & integration
  - Formerly Enterprise Integration & Web Services
- Four day course covering
  - Concurrency
  - Advanced transaction management
  - Messaging with JMS
  - REST Web Services with Spring MVC
  - Spring Batch
  - Spring Integration
  - Spring XD

© Copyright 2016 Pivotal. All rights reserved

PCF Immersion

- Three day course covering
  - Cloud Foundry Basics and Concepts
  - Installation using Ops Manager
  - BOSH Deployment Tool
  - Administration, Troubleshooting, Cloud Controller API
  - High Availability
  - Installing and using Services and Service Brokers
  - Logging, autoscaling, SSO
  - Buildpacks, Continuous delivery
  - 12 factor apps, Microservices

© Copyright 2016 Pivotal. All rights reserved

Developing Applications with Cloud Foundry

- Cloud Foundry for Developers
  - Application deployment to Cloud Foundry
  - Cloud Foundry Concepts
  - Deployment using cf tool or an IDE
  - Accessing and defining Services
  - Using and customizing Buildpacks
  - Design considerations: “12 Factor”
  - JVM Application specifics
  - Using Spring Cloud

Pivotal

© Copyright 2016 Pivotal. All rights reserved

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

© Copyright 2016 Pivotal. All rights reserved

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

© Copyright 2016 Pivotal. All rights reserved

Thank You!

- We hope you enjoyed the course

- Please fill out the evaluation form
- Asia-Pac: http://tinyurl.com/pivotalAPACeval
- MyLearn: http://tinyurl.com/mylearneval

Pivotal

© Copyright 2016 Pivotal. All rights reserved

A NEW PLATFORM FOR A NEW ERA

© Copyright 2016 Pivotal. All rights reserved

Spring Introduction

Spring Background for the Course

Spring Configuration, Dependency Injection,
Bean Creation, Proxies, Controllers

© Copyright 2016 Pivotal. All rights reserved

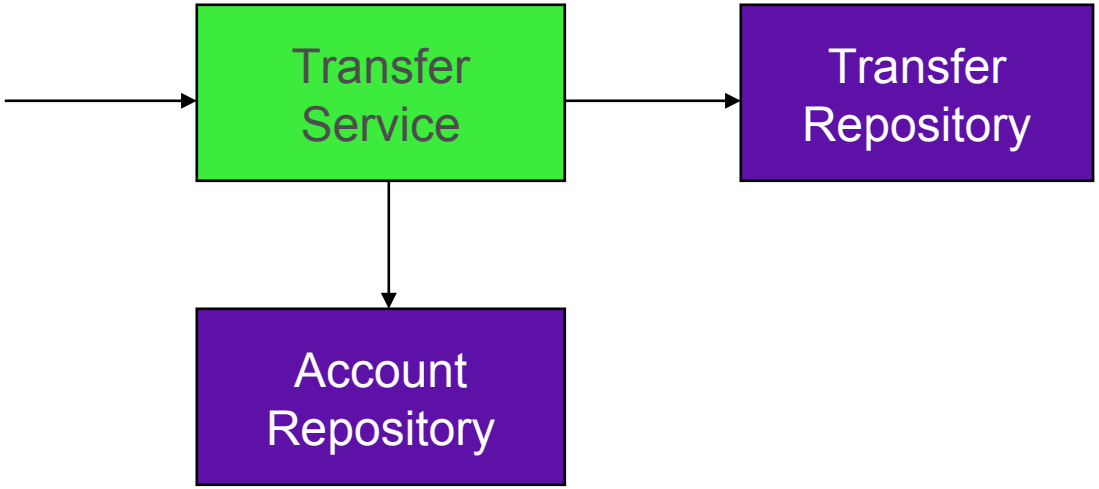
Topics in this session

- **Why Spring?**
- Configuration using Spring
- Aspects and Proxies
- Bean Creation
- REST Controllers

© Copyright 2016 Pivotal. All rights reserved

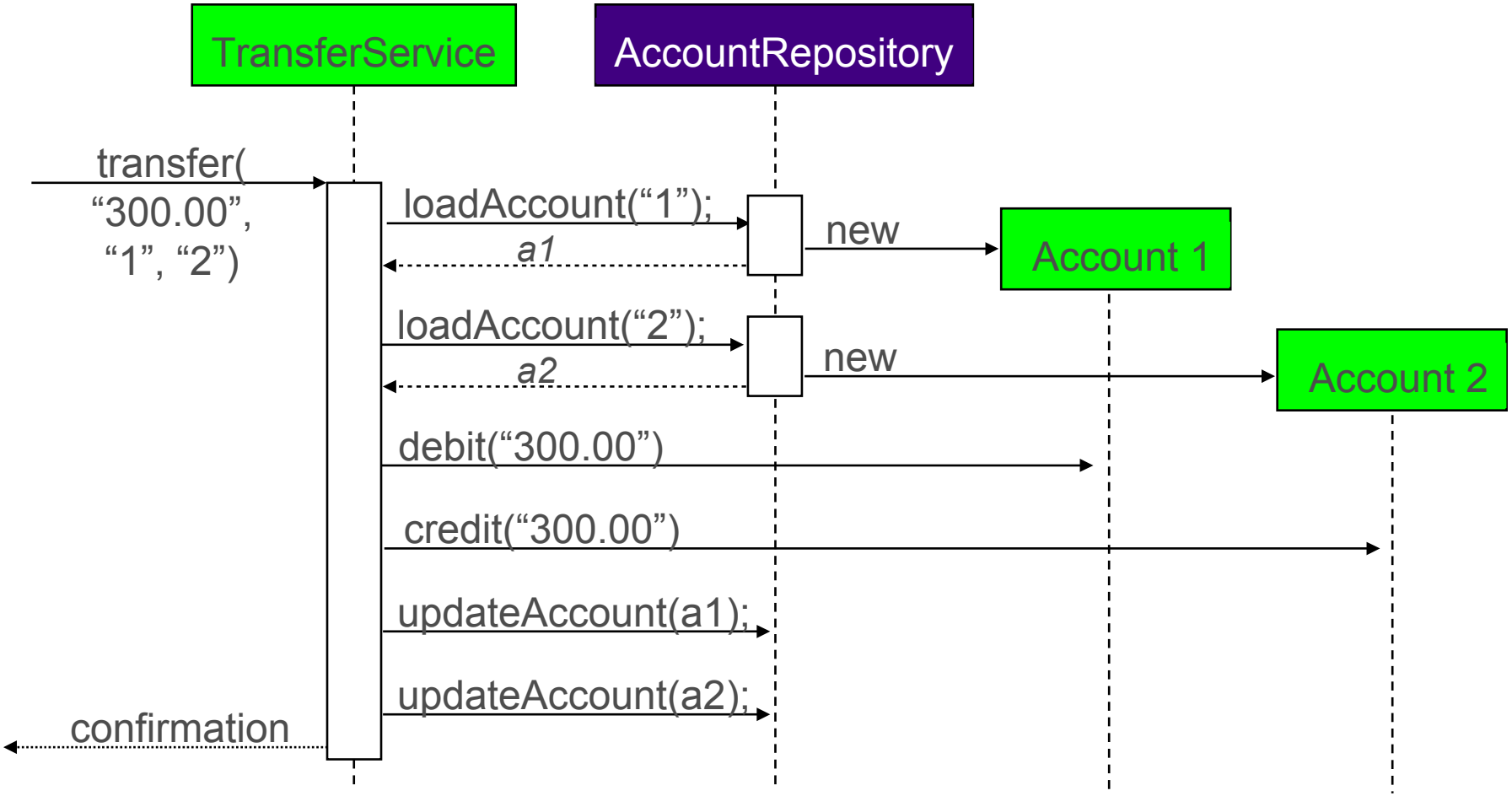
Application Configuration

- A typical application system consists of several parts working together to carry out a use case



© Copyright 2016 Pivotal. All rights reserved

Example: Money Transfer System



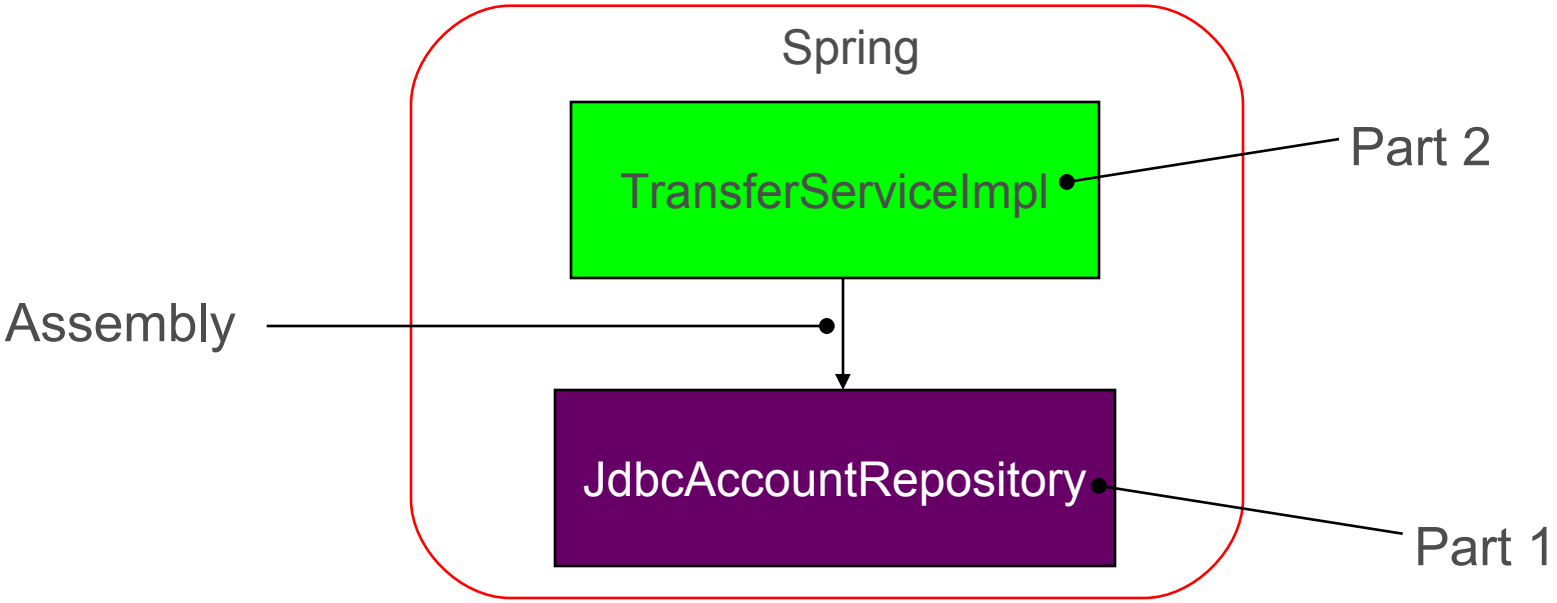
© Copyright 2016 Pivotal. All rights reserved

Spring's Configuration Support

- Spring provides support for assembling such an application system from its parts
  - Parts do not worry about finding each other
  - Any part can easily be swapped out

© Copyright 2016 Pivotal. All rights reserved

Money Transfer System Assembly




```
(1) repository = new JdbcAccountRepository(…);
(2) service = new TransferServiceImpl();
(3) service.setAccountRepository(repository);
```


© Copyright 2016 Pivotal. All rights reserved

Parts are Just Plain Old Java Objects

```
public class JdbcAccountRepository implements
    AccountRepository {
    …
}
```

Implements a service/business interface

Part 1

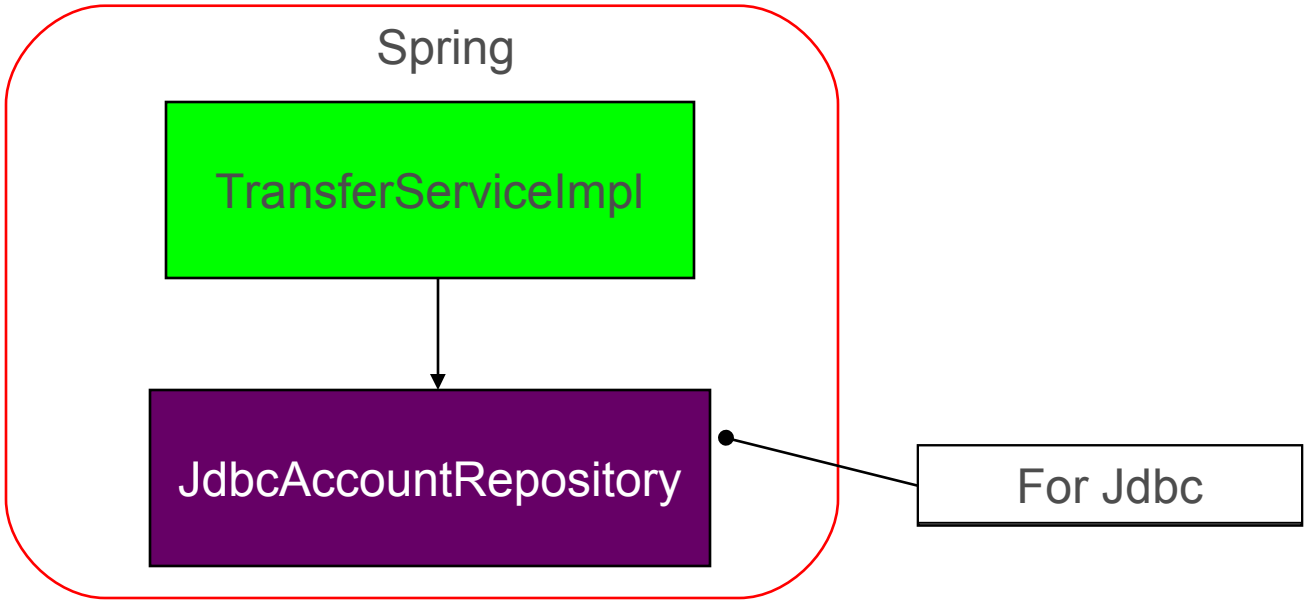
```
public class TransferServiceImpl implements TransferService {
    private AccountRepository accountRepository;

    public void setAccountRepository(AccountRepository ar) {
        accountRepository = ar;
    }
    …
}
```

Depends on *interface*;
conceals complexity of implementation;
allows for swapping out implementation

Part 2

© Copyright 2016 Pivotal. All rights reserved

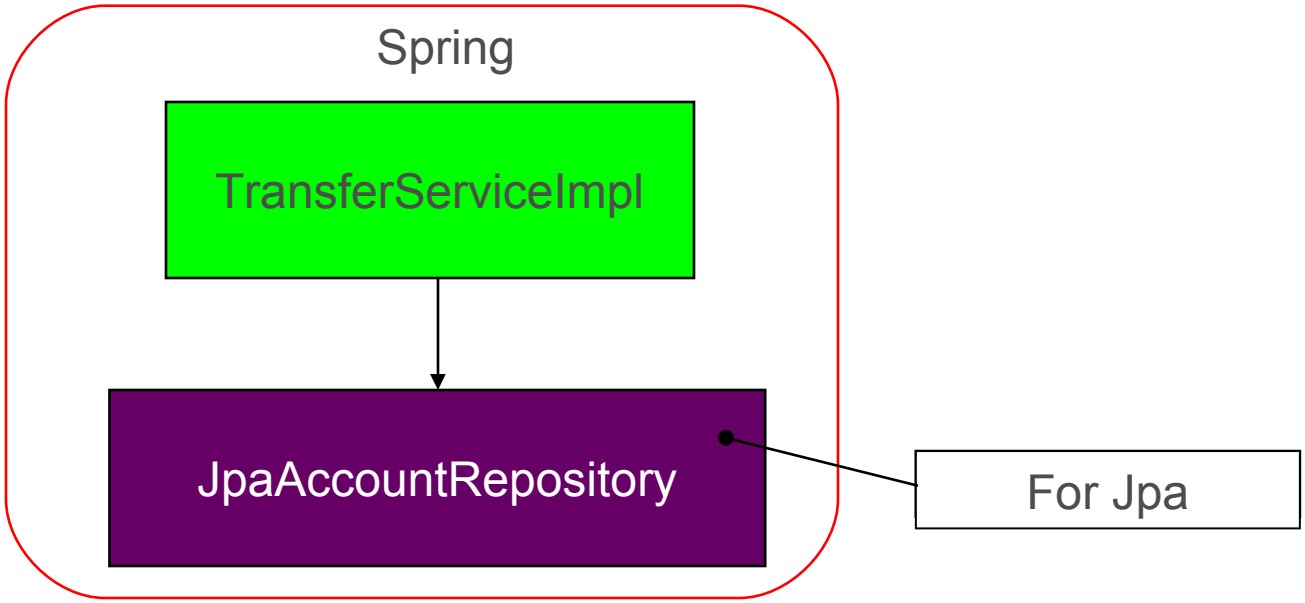
Swapping Out Part Implementations




```
(1) new JdbcAccountRepository(…);
(2) new TransferServiceImpl();
(3) service.setAccountRepository(repository);
```


© Copyright 2016 Pivotal. All rights reserved

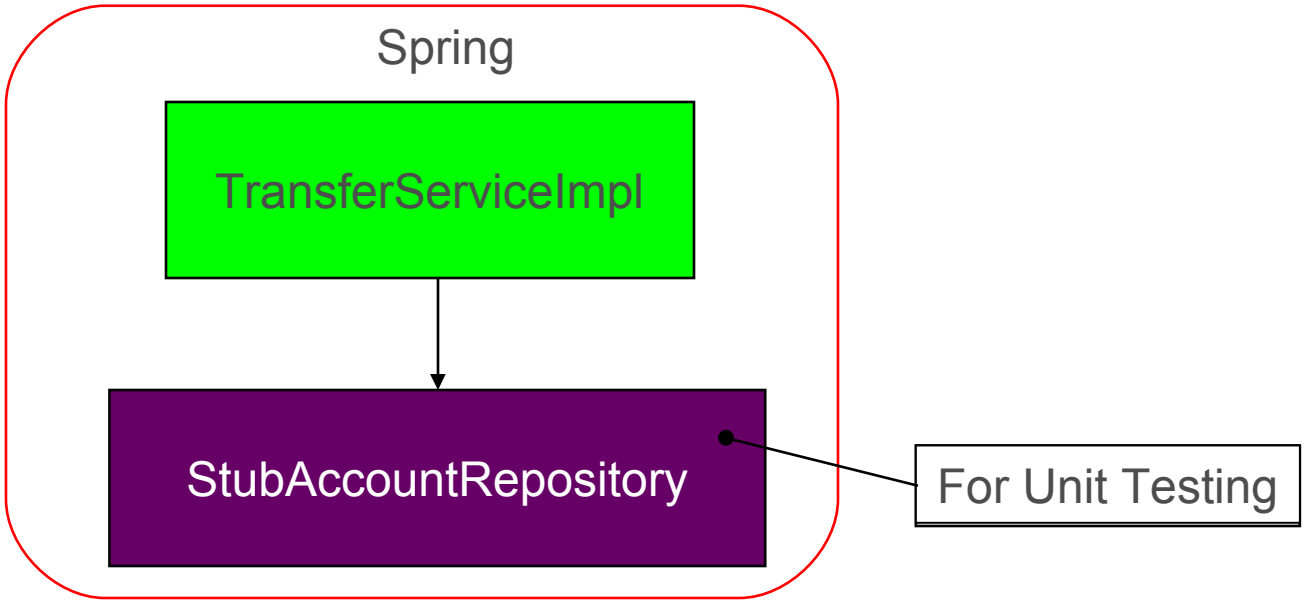
Swapping Out Part Implementations




```
(1) new JpaAccountRepository(…);
(2) new TransferServiceImpl();
(3) service.setAccountRepository(repository);
```


© Copyright 2016 Pivotal. All rights reserved

Swapping Out Part Implementations




```
(1) new StubAccountRepository();
(2) new TransferServiceImpl();
(3) service.setAccountRepository(repository);
```

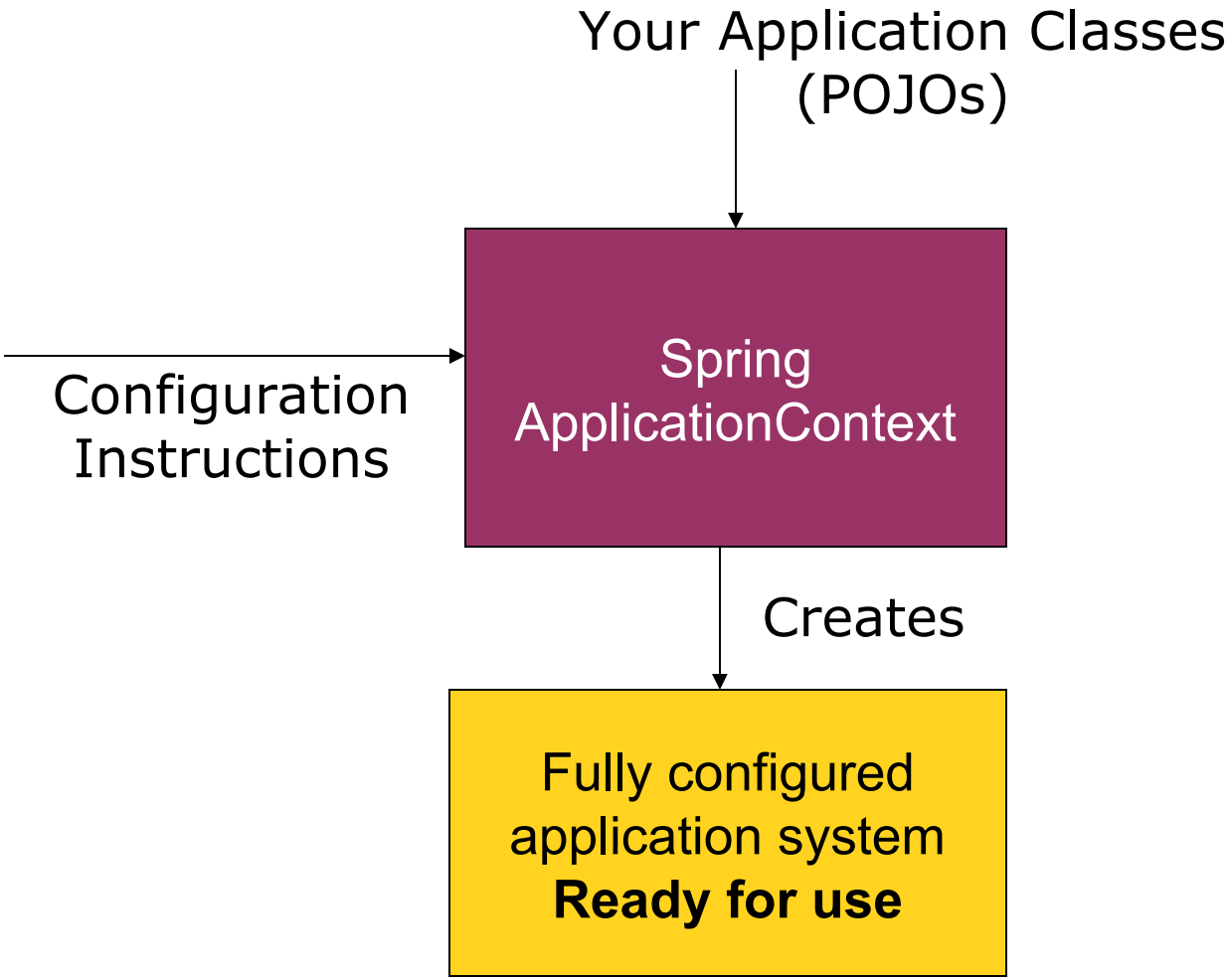

© Copyright 2016 Pivotal. All rights reserved

Topics in this session

- Why Spring?
- **Configuration using Spring**
- Aspects and Proxies
- Bean Creation
- REST Controllers

© Copyright 2016 Pivotal. All rights reserved

How Spring Works



© Copyright 2016 Pivotal. All rights reserved

Your Application Classes

```
public class TransferServiceImpl implements TransferService {
    public TransferServiceImpl(AccountRepository ar) {
        this.accountRepository = ar;
    }
    …
}
```

Needed to perform money transfers between accounts

```
public class JdbcAccountRepository implements AccountRepository {
    public JdbcAccountRepository(DataSource ds) {
        this.dataSource = ds;
    }
    …
}
```

Needed to load accounts from the database

© Copyright 2016 Pivotal. All rights reserved

Configuration Instructions – Java

```
@Configuration
public class ApplicationConfig {
  @Bean public TransferService transferService() {
    return new TransferServiceImpl( accountRepository() );
  }
  @Bean public AccountRepository accountRepository()  {
    return new JdbcAccountRepository( dataSource() );
  }
  @Bean public DataSource dataSource() {
    DataSource dataSource = new BasicDataSource();
    dataSource.setDriverClassName("org.postgresql.Driver");
    dataSource.setUrl("jdbc:postgresql://localhost/transfer" );
    dataSource.setUser("transfer-app");
    dataSource.setPassword("secret45" );
    return dataSource;
  }
}
```

Dependency injection

Dependency injection

Bean ID defaults to method name or use @Bean(name=...)

© Copyright 2016 Pivotal. All rights reserved

Configuration Instructions – XML

```
<beans>

   <bean id="transferService" class="com.acme.TransferServiceImpl">
      <constructor-arg ref="accountRepository" />
   </bean>

   <bean id="accountRepository" class="com.acme.JdbcAccountRepository">
      <constructor-arg ref="dataSource" />
   </bean>

   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
     <property name="driverClassName" value="org.postgresql.Driver" />
     <property name="url" value="jdbc:postgresql://localhost/transfer" />
     <property name="user" value="transfer-app" />
     <property name="password" value="secret45" />
   </bean>

</beans>
```

Dependency injection

Dependency injection

Bean ID specified explicitly via id attribute

© Copyright 2016 Pivotal. All rights reserved

Implicit Configuration using Annotations

- Annotation-based configuration w*ithin* bean-class

```
@Component ( value="transferService" )
public class TransferServiceImpl implements TransferService {
        @Autowire
        public TransferServiceImpl(AccountRepository repo) {
            this.accountRepository = repo;
        }
}
```

Bean ID

Dependency injection

Annotations embedded *within* POJOs

```
@Configuration
@ComponentScan ( "com.bank" )
public class AnnotationConfig {
    // No bean definition needed any more
}
```

Find @Component classes within designated (sub)packages

```
<context:component-scan base-packages="com.bank">
```


© Copyright 2016 Pivotal. All rights reserved

Creating and Using the Application

```
// Create the application from the configuration
ApplicationContext context =
    SpringApplication.run( ApplicationConfig.class );

// Look up the application service interface
TransferService service =
    context.getBean("transferService", TransferService.class);

// Use the application
service.transfer(new MonetaryAmount("300.00"), "1", "2");
```

© Copyright 2016 Pivotal. All rights reserved

Topics in this session

- Why Spring?
- Configuration using Spring
- **Aspects and Proxies**
- Bean Creation
- REST Controllers

© Copyright 2016 Pivotal. All rights reserved

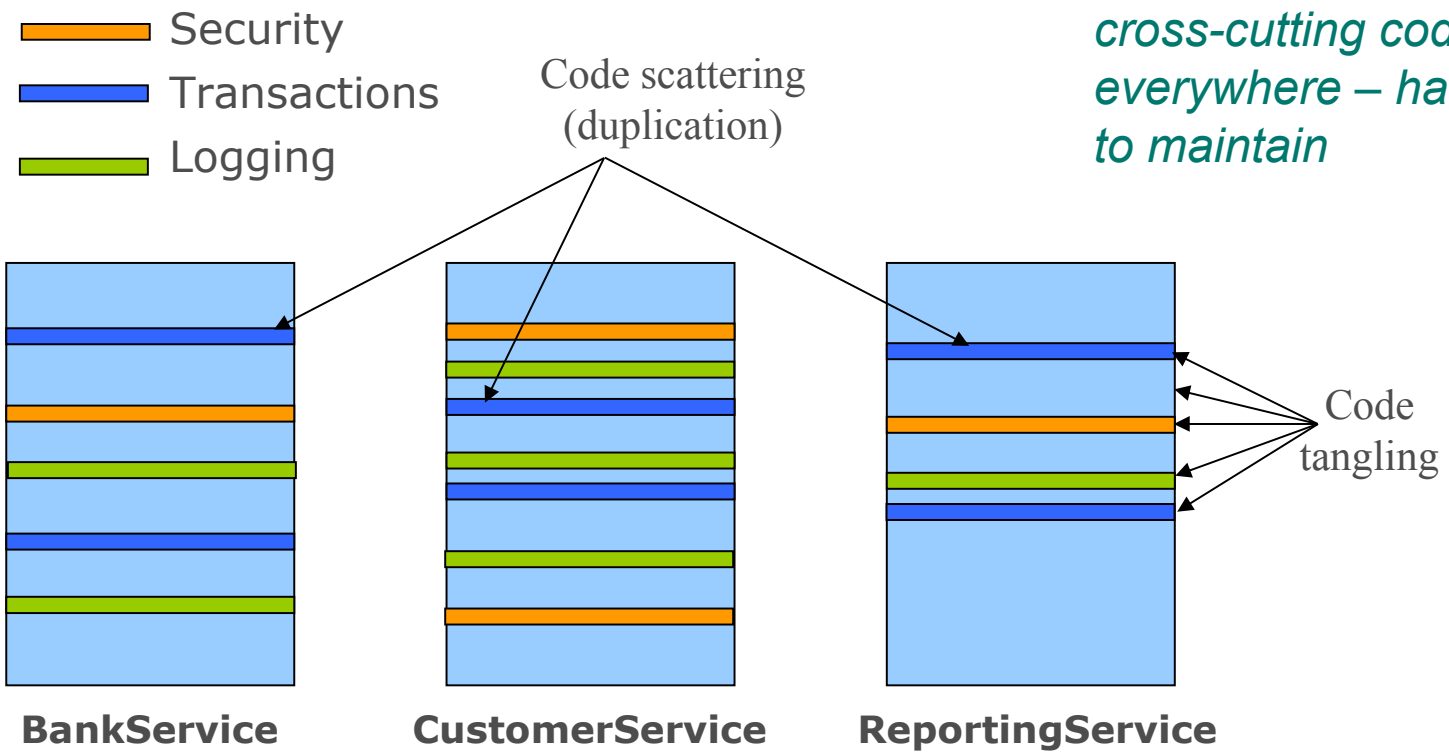
What Problem Does AOP Solve?

- Aspect-Oriented Programming (AOP) enables modularization of cross-cutting concerns
  - Generic functionality needed in many places in your application
- Examples
  - Logging and Tracing
  - Transaction Management
  - Security, Caching, Error Handling
  - Performance Monitoring
  - Custom Business Rules

© Copyright 2016 Pivotal. All rights reserved

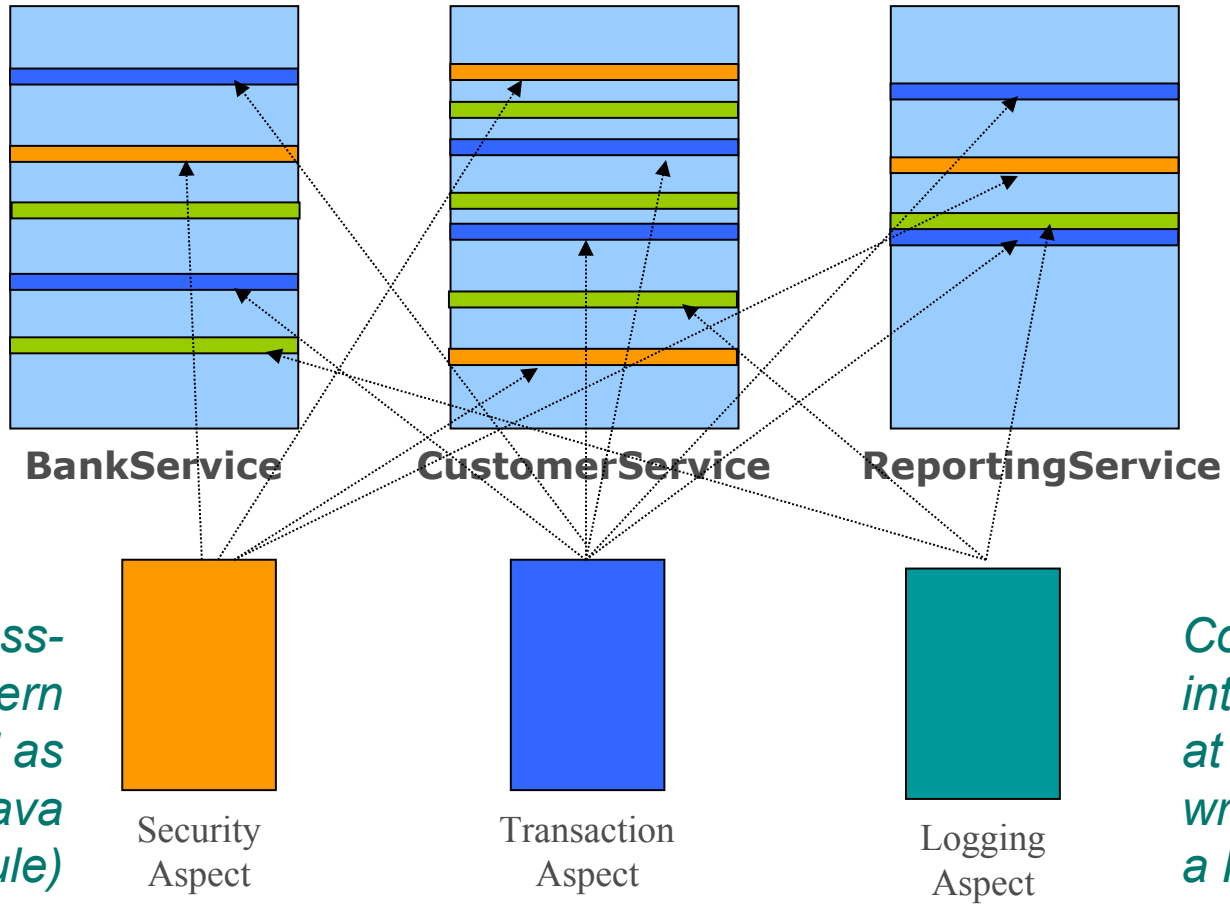
System Evolution Without Modularization

You cut and paste (duplicate) the cross-cutting code everywhere – hard to maintain



© Copyright 2016 Pivotal. All rights reserved

System Evolution: AOP based



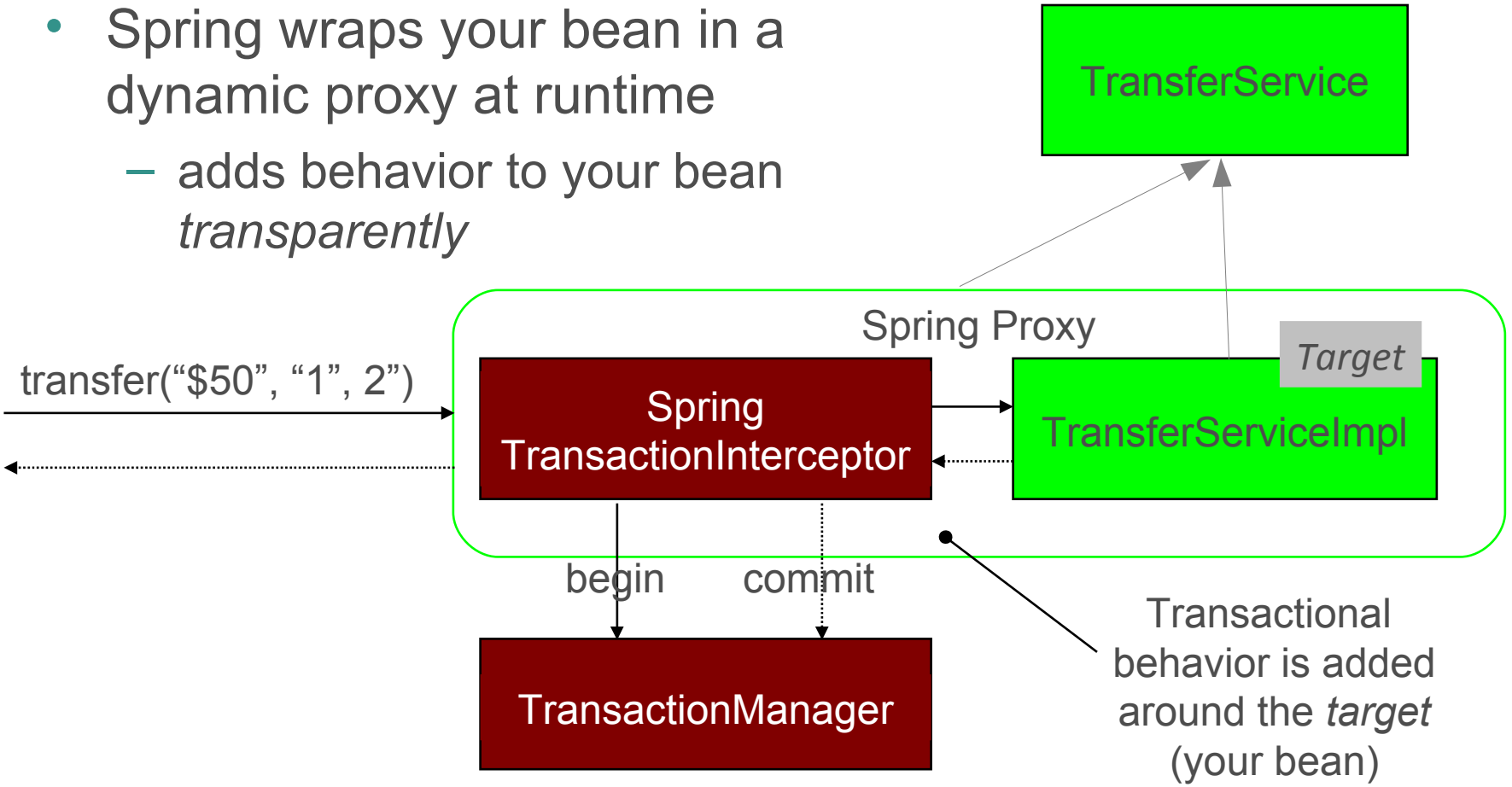
*Each cross-cutting concern is defined as a **single** Java class (module)*

*Code is **woven** into your objects at runtime – by wrapping them in a Proxy*

© Copyright 2016 Pivotal. All rights reserved

Aspects Implemented Using a Proxy
Transactional Example

- Spring wraps your bean in a dynamic proxy at runtime
  - adds behavior to your bean *transparently*



© Copyright 2016 Pivotal. All rights reserved

Kinds of Proxies

- Spring will create either JDK or CGLib proxies

JDK Proxy

- Also called dynamic proxies
- API is built into the JDK
- Requirements: Java interface(s)

CGLib Proxy

- NOT built into JDK
- Included in Spring jars
- Used when interface not available
- Cannot be applied to final classes or methods

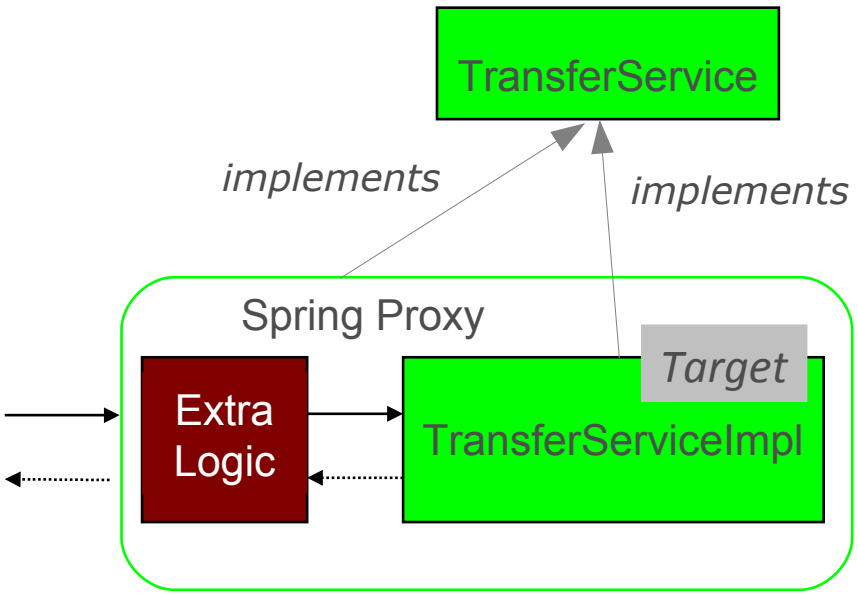
Recommendation: Code to interfaces / Use JDK proxies (default)

See Spring Reference - 10.5.3 JDK- and CGLIB-based proxies

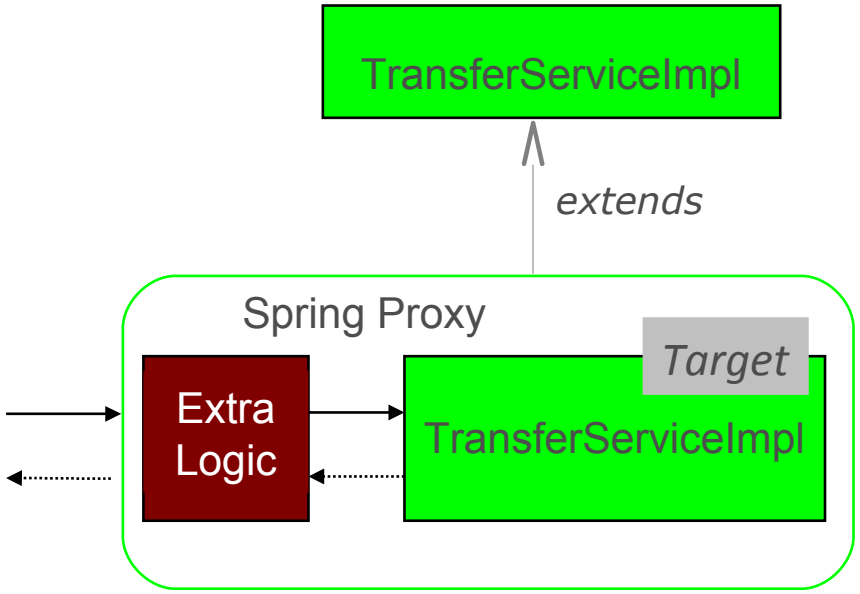
© Copyright 2016 Pivotal. All rights reserved

JDK vs CGLib Proxies

- JDK Proxy
  - Interface based



- CGLib Proxy
  - subclass based



© Copyright 2016 Pivotal. All rights reserved

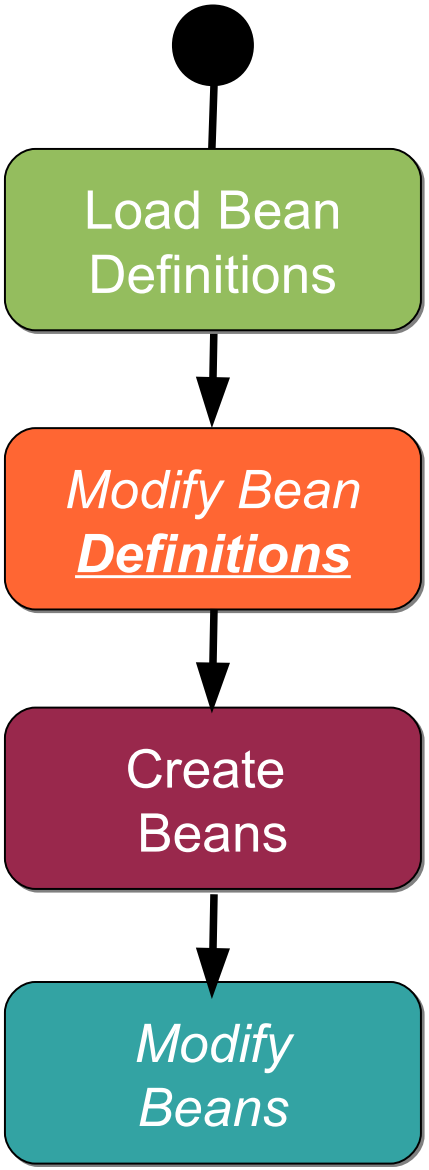
Topics in this session

- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- **Bean Creation**
  - **Using Properties**
  - Manipulating Beans
- REST Controllers

© Copyright 2016 Pivotal. All rights reserved

Bean Creation

- Spring performs a number of steps:
  - Load bean definitions
  - *Optionally* modify definitions
  - Create each bean and inject dependencies
  - *Optionally* post-process the beans



© Copyright 2016 Pivotal. All rights reserved

Why Modify Bean Definitions? *Properties*

```
@Configuration
@PropertySource ( “classpath:db-config.properties” )
public class ApplicationConfig {

  @Bean
  public DataSource dataSource(
        @Value("${db.driver}") String dbDriver,
        @Value("${db.url}") String dbUrl,
        @Value("${db.user}") String dbUser,
        @Value("${db.password}") String dbPassword) {
    DataSource ds = new BasicDataSource();
    ds.setDriverClassName(  dbDriver);
    ds.setUrl( dbUrl);
    ds.setUser( dbUser);
    ds.setPassword( dbPassword ));
    return ds;
  }
  …
```

Load properties from specified file(s)

Replace $ variables with properties

```
db.url=jdbc:oracle:...
db.user=moneytransfer-app
...
```


© Copyright 2016 Pivotal. All rights reserved

XML Equivalent

Expands $ variables from properties file(s)

```
<beans  ...>
   <context:property-placeholder location="db-config.properties" />

   <bean id="dataSource" class="com.oracle.jdbc.pool.OracleDataSource">
      <property name="URL" value="${db.url}" />
      <property name="user" value="${db.user}" />
   </bean>
</beans>
```

```
dbUrl=jdbc:oracle:...
dbUserName=moneytransfer-app
```

```
   <bean id="dataSource"
         class="com.oracle.jdbc.pool.OracleDataSource">
      <property name="URL" value="jdbc:oracle:..." />
      <property name="user" value="moneytransfer-app" />
   </bean>
```


© Copyright 2016 Pivotal. All rights reserved

Loading Property Sources

Spring Boot does this automatically

- Property sources are loaded by a dedicated Spring bean
  - The **`PropertySourcesPlaceholderConfigurer`**
  - **`@PropertySource`** ignored if this is not defined

Explicit: Java Configuration

```
@Bean
public static PropertySourcesPlaceholderConfigurer
                        propertySourcesPlaceholderConfigurer() {
    return new PropertySourcesPlaceholderConfigurer();
}
```

Implicit: XML

```
<beans  ...>
   <context:property-placeholder location="db-config.properties" />
   ...
</beans>
```


© Copyright 2016 Pivotal. All rights reserved

Spring Expression Language
Annotation or Java Config Examples

```
@Repository
public class RewardsDatabase {

    @Value("#{systemProperties.databaseName}")
    public void setDatabaseName(String dbName) { ... }

    @Value("#{strategyBean.databaseKeyGenerator}")
    public void setKeyGenerator(KeyGenerator kg) { ... }
}
```

Equivalent to System.getProperty(...)

Can refer a nested property

```
@Configuration
class RewardConfig
{
    @Bean public RewardsDatabase rewardsDatabase
            (@Value("#{systemProperties.databaseName}") String databaseName, …)  {
        ...
    }
}
```


© Copyright 2016 Pivotal. All rights reserved

Spring Expression Language
XML Examples

```
<bean id="rewardsDb" class="com.acme.RewardsDatabase">
      <property name="keyGenerator"
                value="#{strategyBean.databaseKeyGenerator}" />
</bean>

<bean id="strategyBean" class="com.acme.DefaultStrategies">
   <property name="databaseKeyGenerator" ref="myKeyGenerator"/>
</bean>

<bean id="taxCalculator" class="com.acme.TaxCalculator">
  <property name="defaultLocale" value="#{ systemProperties['user.region'] }"/>
</bean>
```

Can refer a nested property

Equivalent to System.getProperty(...)

© Copyright 2016 Pivotal. All rights reserved

External Properties with Spring Boot

- Spring Boot automatically loads properties from
  - The environment – `System.getenv()`
  - System properties – `System.getProperty()`
  - Java properties files
  - YML configuration files
  - JNDI
  - Servlet initialization parameters
- *"Relaxed"* binding
  - Can use `MY_VALUE` from environment or `my-value` from a properties/YML file to set `@Value("${myValue}")`

© Copyright 2016 Pivotal. All rights reserved

Spring Boot Application Properties

- Spring Boot automatically looks for application.properties in the classpath root
  - Or use application.yml if you prefer YAML
- Many predefined properties for controling Spring Boot
  - See *Appendix A* of online docs for list

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

application.properties

```
spring:
  datasource:
    url: jdbc:mysql://localhost/test
    username: dbuser
    password: dbpass
    driver-class-name: com.mysql.jdbc.Driver
```

No tabs!

application.yml

© Copyright 2016 Pivotal. All rights reserved

Java Buildpack Auto-Configuration

- During staging of a Java Application
  - Buildpack scans Spring configuration
  - Looks for beans that define known services
    - Such as an RDBMS data-source
    - Or a Rabbit/MQ connection-factory
  - Swaps them for equivalent beans using services from CF
    - Uses connection info from **`VCAP_SERVICES`**

- How?
  - Using a *custom* bean-definition modifier

© Copyright 2016 Pivotal. All rights reserved

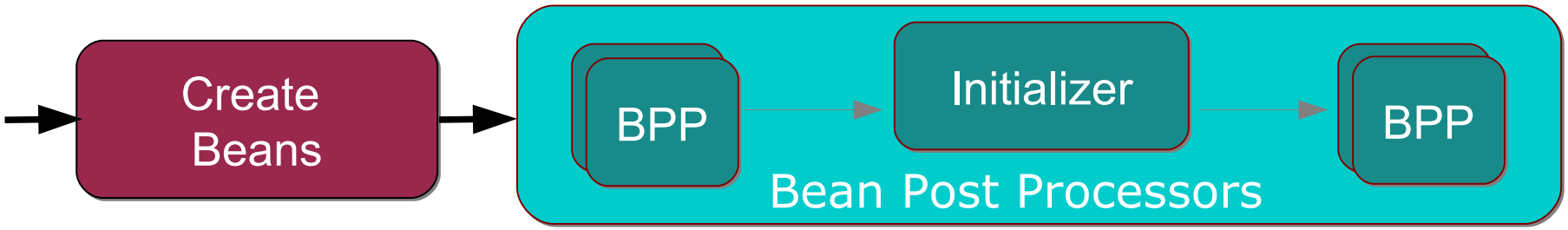
Topics in this session

- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- **Bean Creation**
  - Using Properties
  - **Manipulating Beans**
- REST Controllers

© Copyright 2016 Pivotal. All rights reserved

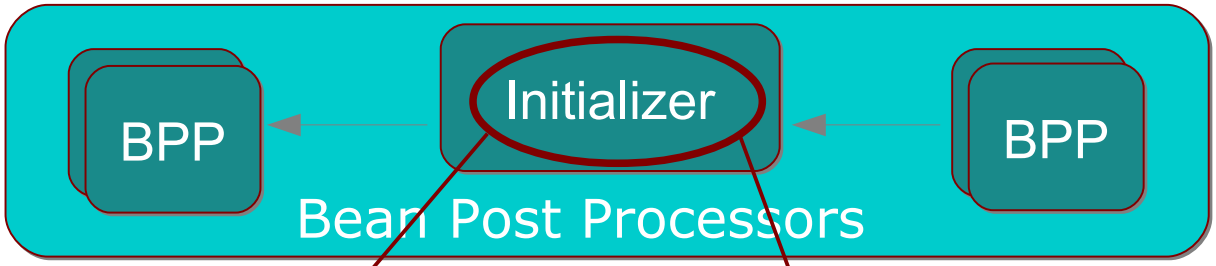
Manipulating Bean Instances

- After dependency injection each bean goes through a **post-processing** phase
  - Further configuration and initialization may occur
    - Such as Initialization or Proxying
    - Performed by dedicated Spring components called *Bean Post Processors* (BPP)
  - **Powerful enabling feature: "Spring Magic"**
    - Spring provides many BPPs out-of-the-box



© Copyright 2016 Pivotal. All rights reserved

Initializer Bean Post-Processors




```
public class JdbcAccountRepo {
    @PostConstruct
    public void populateCache() {
        //...
    }
    ...
```

By Annotation

```
@ComponentScan
```

```
<bean id="accountRepository"
      class="com.acme.JdbcAccountRepo"
      init-method="populateCache">
    ...
</bean>
```

Using XML

```
<context:component-scan ... />
```

Usually enabled as a side-effect of using component-scanning.

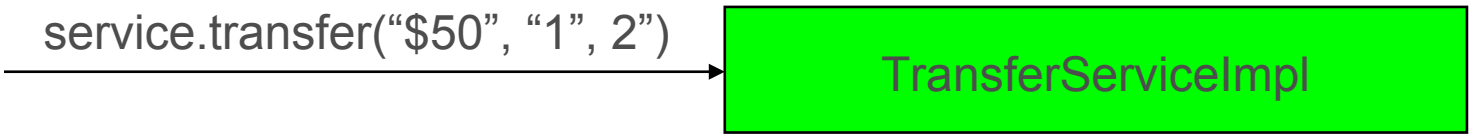
© Copyright 2016 Pivotal. All rights reserved

BPPs Enable Proxies

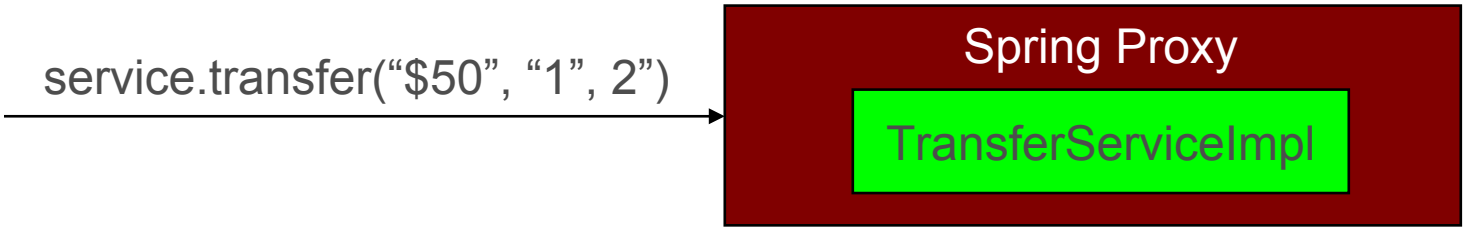
- Spring *doesn't always* give you the bean you asked for

```
service = context.getBean("transferService", TransferService.class);
```

- Sometimes the bean is just your raw object



- Or your bean has been wrapped in a *proxy* by *a BPP*
  - To implement extra behavior (*aspects*)



© Copyright 2016 Pivotal. All rights reserved

Topics in this session

- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- Bean Creation
- **REST Controllers**

© Copyright 2016 Pivotal. All rights reserved

Example REST Controller

- Annotate controllers with @RestController
- @RequestMapping tells Spring what method to execute when processing a particular request

```
@RestController
public class AccountController {
    @Autowired
    public AccountService accountService;

    @RequestMapping("/listAccounts")
    public List<Account> list(Principal owner) {...}
}
```

Example of calling URL: http://localhost:8080 / listAccounts

application url request mapping

© Copyright 2016 Pivotal. All rights reserved

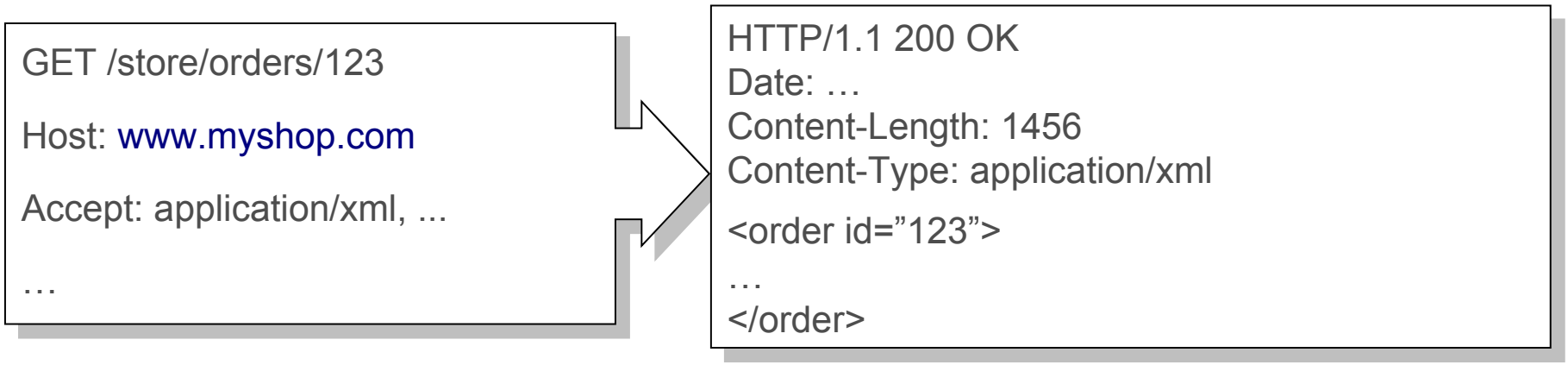
HttpMessageConverter

- Converts between HTTP request/response and object
  - Controller method deals in objects
  - Conversion hidden and automatic
- Various implementations registered by default by Spring Boot
  - XML (JAXB2* or Jackson*)
  - Feed data*, i.e. Atom/RSS
  - Protocol Buffers*
  - JSON* and GSON*
  - Byte[], String, BufferedImage, Form-based data
- Can customize to register other HttpMessageConverters

* Requires 3rd party libraries on classpath

© Copyright 2016 Pivotal. All rights reserved

Retrieving a Representation: GET




```
@RequestMapping(value="/orders/{id}", method=RequestMethod.GET)
@ResponseStatus(HttpStatus.OK)  // 200: this is the default
public @ResponseBody Order getOrder(@PathVariable("id") long id) {
   return orderService.findOrderById(id);
}
```

Outgoing XML generated from the Order object returned from controller method

@ResponseBody not strictly necessary when using @RestController (assumed)

© Copyright 2016 Pivotal. All rights reserved

Updating existing Resource: PUT

```
PUT /store/orders/123/items/abc
Host: www.myshop.com
Content-Type: application/xml
<item>
…
</item>
```

Incoming XML converted to the Order passed into controller method

```
HTTP/1.1 204 No Content
Date: …
Content-Length: 0
```

```
@RequestMapping(value="/orders/{orderId}/items/{itemId}",
                method=RequestMethod.PUT)
@ResponseStatus(HttpStatus.NO_CONTENT)  // 204
public void updateItem(
        @PathVariable("orderId") long orderId,
        @PathVariable("itemId") String itemId
        @RequestBody Item item) {
  orderService.findOrderById(orderId).updateItem(itemId, item);
}
```


© Copyright 2016 Pivotal. All rights reserved

Topics Covered in This Session

- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- Bean Creation
- REST Controllers

© Copyright 2016 Pivotal. All rights reserved