Pivotal

Spring/Java Cloud-Native Bootcamp

Prasad Bopardikar
Mayuresh Krishna
Matt Gunter

Goal of this boot camp.

Introduce you to practices, platforms and tools for building modern Java applications.

Hi there.

Who are we?

9-9:15AM	Introduction	
9:15-9:30AM	Getting good at software	
9:30-10AM	All about microservices	
10-10:15AM	What's cloud-native all about?	
10:15-10:45AM	Introducing Cloud Foundry	
10:45-11AM	BREAK TIME	
11-12:30PM	Cloud-native Java technologies and patterns	
12:30-1PM	LUNCH	
1-4PM	Hands on exercises	
4-4:30PM	Wrap up	

Why do you need to be good at software?

Customers expect it.

Meet demand to operate at scale.

Gives you more business options.

Your competitors are doing it.

It makes everyone happier.

Ok, but how do I know that I'm doing well at software?

Improved speed. Faster cycle time, more frequently deployments.

Improved scale. More requests per second to apps and services.

Improved stability. Greater uptime of customer-facing service.

Improved security. Achieving 100% patch coverage.

Improved simplicity. Reduce complexity of processes and tools.

What are microservices?

It refers to an architectural style that supports constant change in your environment. This is accomplished by creating applications out of independent, loosely-coupled, domain-oriented services.

Microservices architecture	Monolithic architecture

Moving to microservices? Here's what to consider.

Do you have a **pressing** reason to do it?

Can you rearrange your teams?

Are you ready to decompose your monoliths?

How will you decompose?

Are you currently doing CI / CD?

Is your production environment automated?

How will you discover services at runtime?

What can you do to prevent cascading failures?

Are you ready to evolve your data platforms?

Do you need to modernize your messaging and event stream processing toolchain?

Are you ready for modern logging, monitoring?

What is “cloud-native” all about?

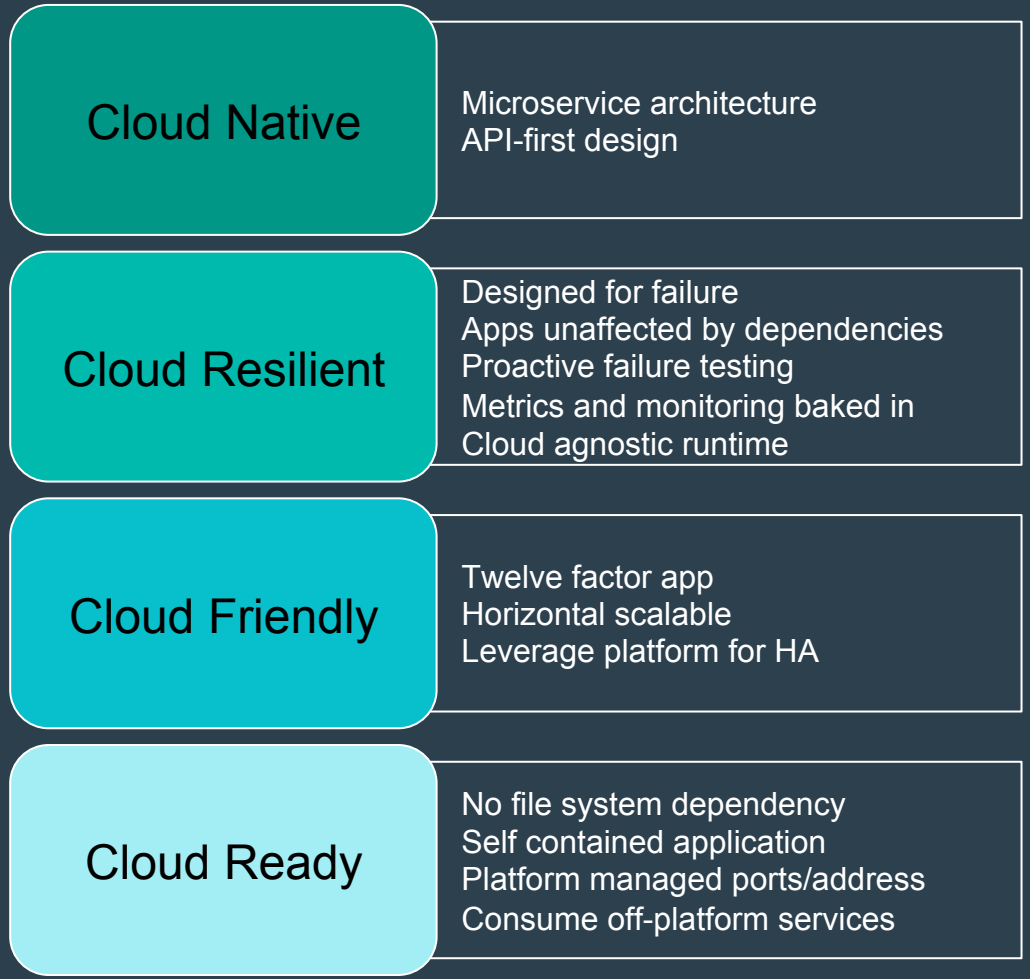
This is an approach to building and operating software that takes advantage of the cloud-computing model. Often seen as a combination of **microservices**, **continuous delivery**, **containers**, and **DevOps**.

It’s all about software that’s built for **scale**, built for **continuous change**, built to **tolerate failure**, built for **manageability**.

Most cloud-native applications comply with the 12 factor criteria.

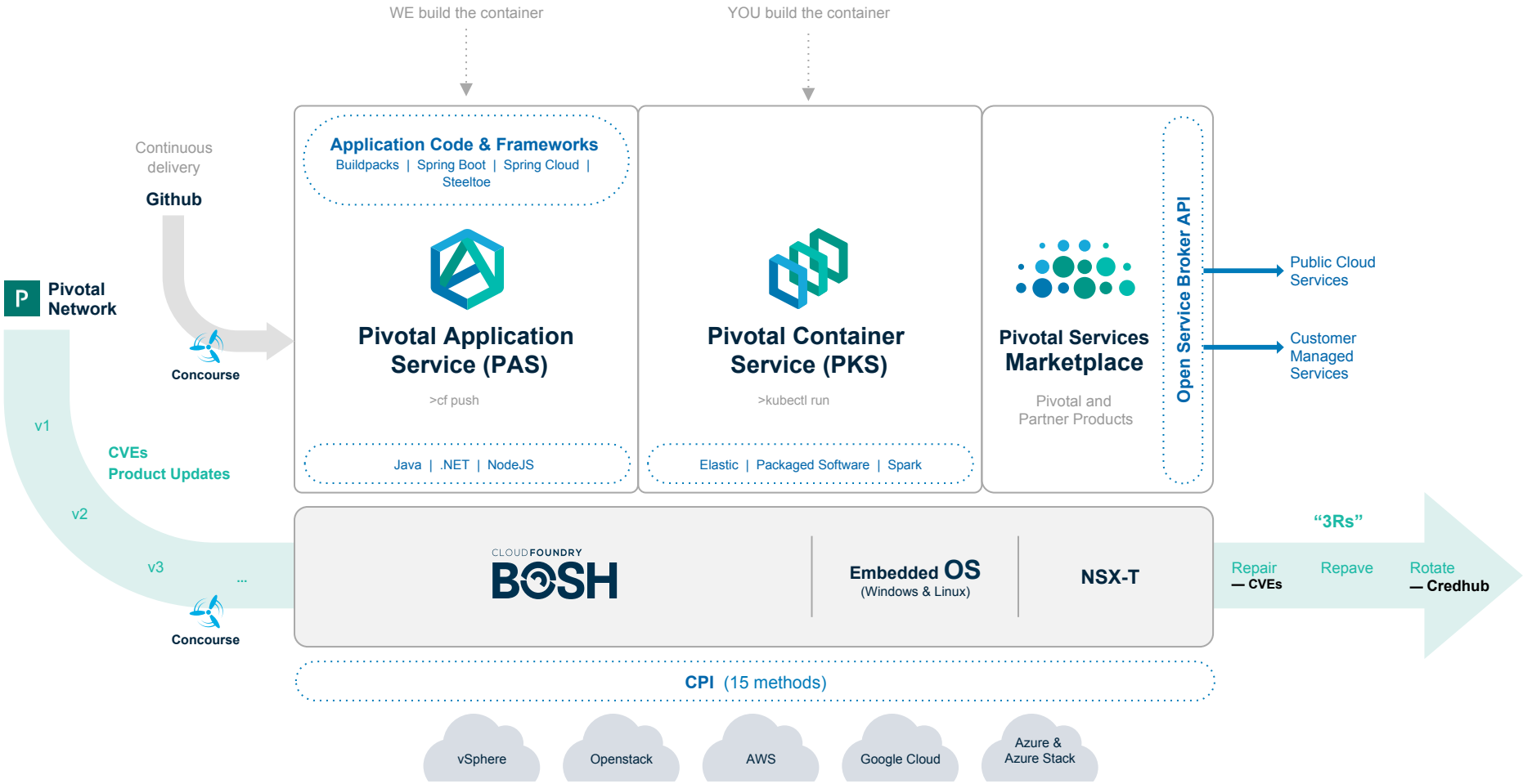
- One codebase tracked in version control
- Explicitly declared dependencies
- Configurations stored in the environment
- Backing services treated as attached resources
- Separate build, release, and run stages
- Apps executed as stateless processes
- Services exported via port binding
- Scaled out via more processes
- Fast startup and graceful shutdown
- Parity among dev, staging, and production environments
- Logs treated as event streams
- Admin tasks run as one-off processes

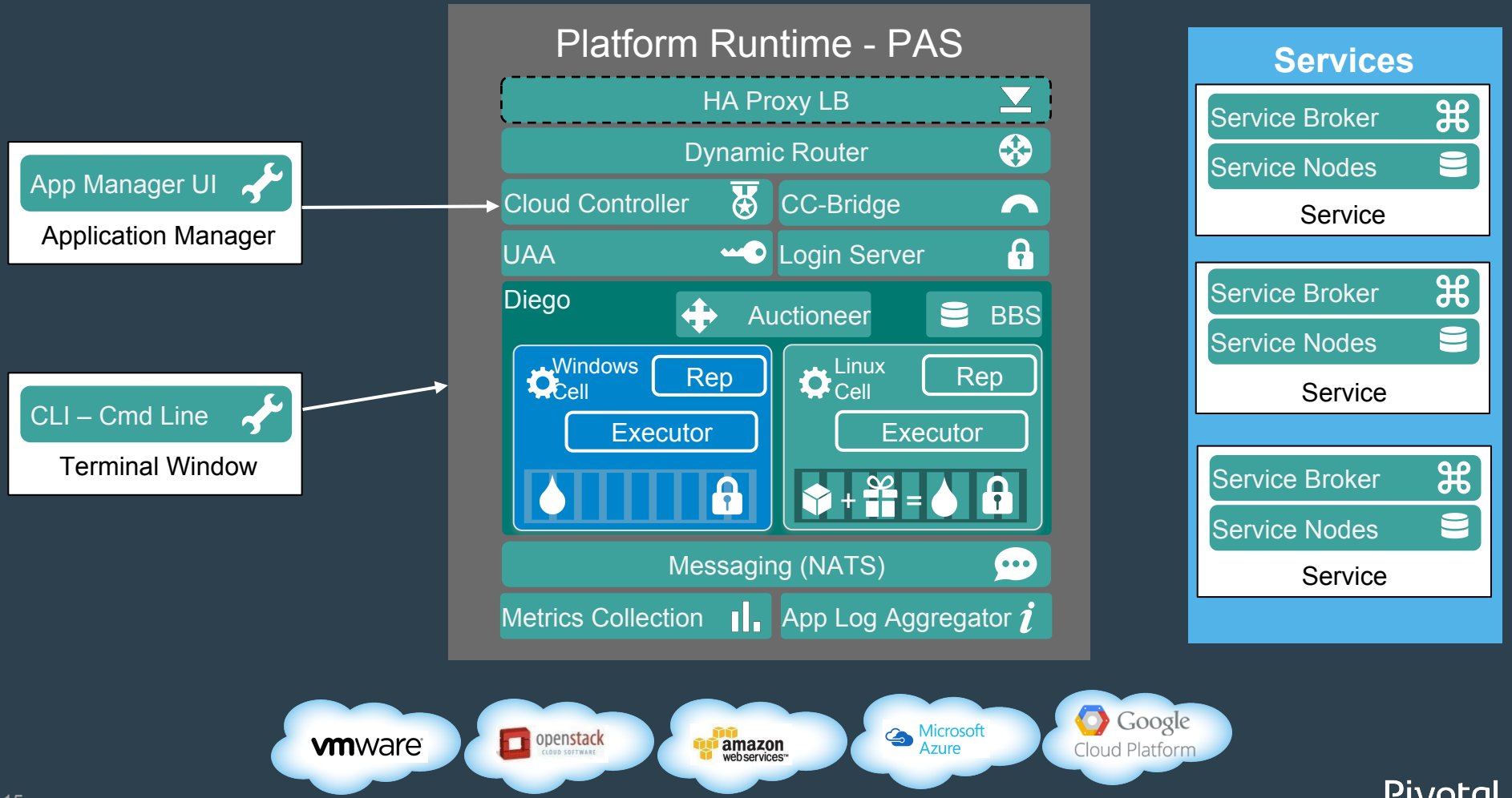
There's a maturity model on your way to cloud native.



So, what actually makes up a cloud-native platform?

Infrastructure	Operations	Deployment	Runtime & Data	Security
Container Orchestration	Service Monitoring and Dependency Management	Lifecycle Management Deploy | Patch | Ugrade | Retire	HTTP / Reverse Proxy	Control Plane Audit & Compliance
Infrastructure Orchestration	Inventory, Capacity, and Management	Release Packaging, Management & Deployment	Application Runtime	Security Event & Incident Management
Service Discovery	Event Management and Routing	CI Orchestration	In-Memory Object Cache	Secrets Management
Configuration Management	Persistent Team Chat	TDD Frameworks	Search	Certificate Management
Core IaaS	Metrics & Logging Analytics & Visualization	Artifact Repository	Messaging	Identity Management
NAT / DNS	Log Aggregation, Indexing & Search	Standard Builds & Configurations	NoSQL Docuemnt Store	Threat & Vulnerability Scanning
SDN / IPAM	Metrics Collection, Storage & Retrieval	Source Control Management	NoSQL Key/Value Store	Network Security
Firewalls / WAN & VPN				
Storage / Load Balancers				
Compute / Network				

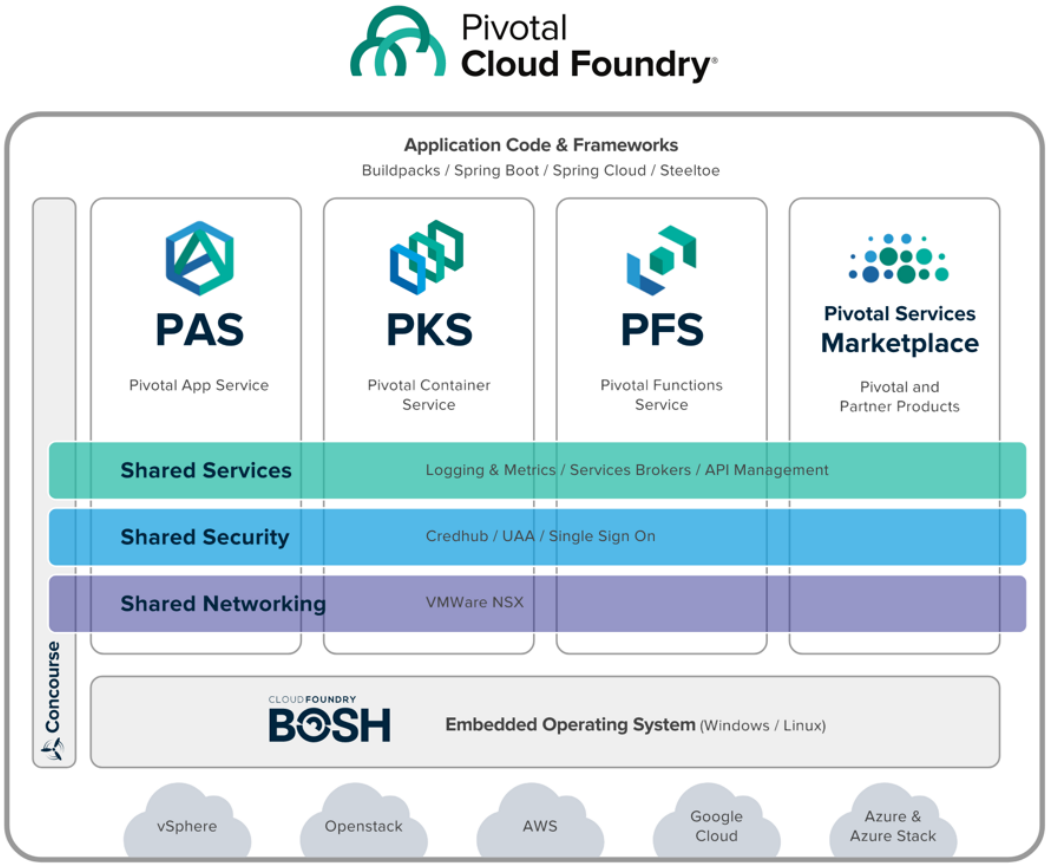

WE build the container
YOU build the container
Continuous delivery
Github
Application Code & Frameworks
Buildpacks | Spring Boot | Spring Cloud | Steeltoe
Pivotal Application Service (PAS)
>cf push
Java | .NET | NodeJS
Pivotal Container Service (PKS)
>kubectl run
Elastic | Packaged Software | Spark
Pivotal Services Marketplace
Pivotal and Partner Products
Open Service Broker API
Public Cloud Services
Customer Managed Services
Pivotal Network
Concourse
v1
CVEs
Product Updates
v2
v3
...
Concourse
CLOUDFOUNDRY BOSH
Embedded OS (Windows & Linux)
NSX-T
"3Rs"
Repair — CVEs
Repave
Rotate — Credhub
CPI (15 methods)
vSphere
Openstack
AWS
Google Cloud
Azure & Azure Stack
Pivotal


App Manager UI
Application Manager
CLI – Cmd Line
Terminal Window
Platform Runtime - PAS
HA Proxy LB
Dynamic Router
Cloud Controller
CC-Bridge
UAA
Login Server
Diego
Auctioneer
BBS
Windows Cell
Rep
Executor
Linux Cell
Rep
Executor
Messaging (NATS)
Metrics Collection
App Log Aggregator
Services
Service Broker
Service Nodes
Service
Service Broker
Service Nodes
Service
Service Broker
Service Nodes
Service
vmware
openstack
amazon web services
Microsoft Azure
Google Cloud Platform

Pivotal

PCF 2.0

Any App
Every Cloud
One Platform



Deploying .NET applications? It doesn't have to be terrible.

Traditional .NET deployment on VMs	Pivotal Cloud Foundry

~$ cf push

45 seconds later

Code Complete & Tested

Find available hosts
Install & configure runtime
Install & configure middleware
Pull application source code
Retrieve dependent libraries
Create application package
Install, configure dependent service(s)
Deploy container to host(s)
Load environment variables
Configure load balancer
Configure firewalls
Update service monitoring tools
Configure log collector
...

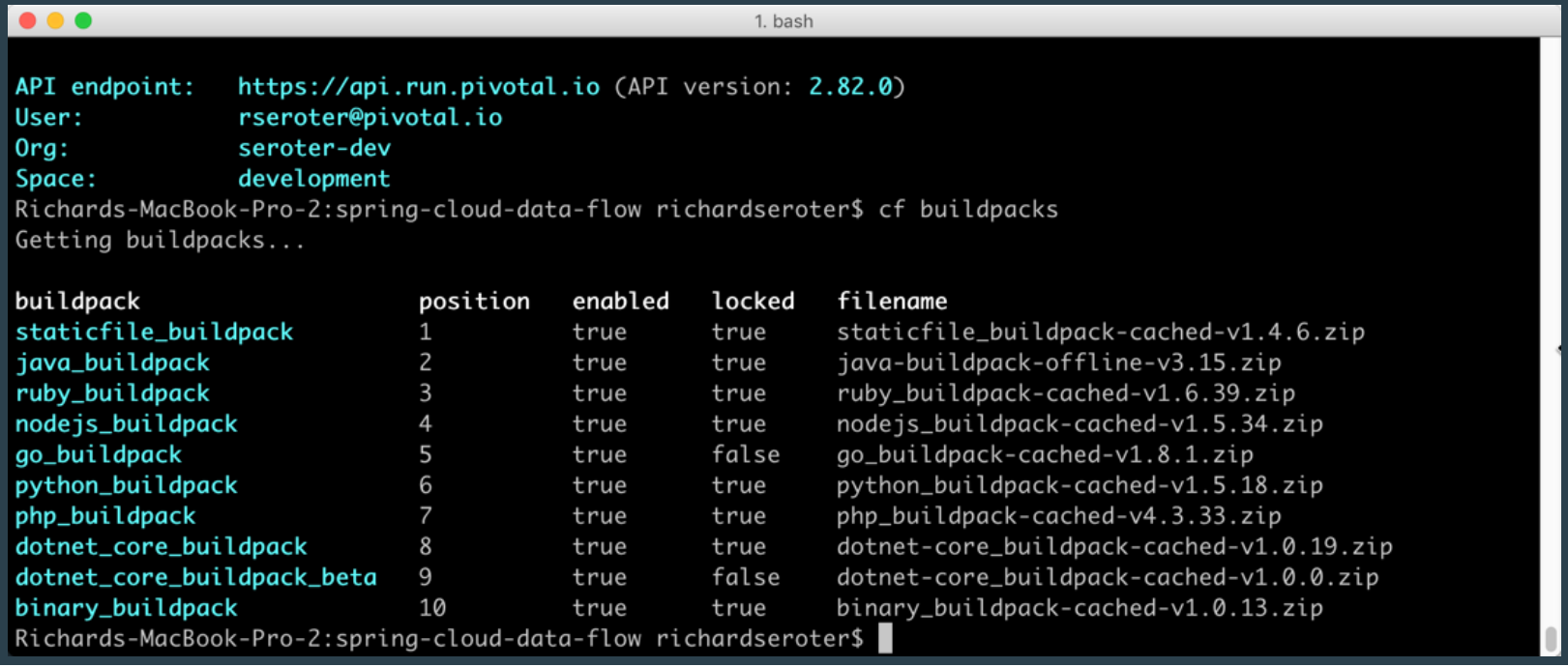
Application in Production

```
$ cf push my-app -p target/myapp.jar
```

or

```
$ cf push my-app --docker-image pivotal/my-image
```

Buildpacks are language-aware



Deploy source code that *the platform* containerizes for you

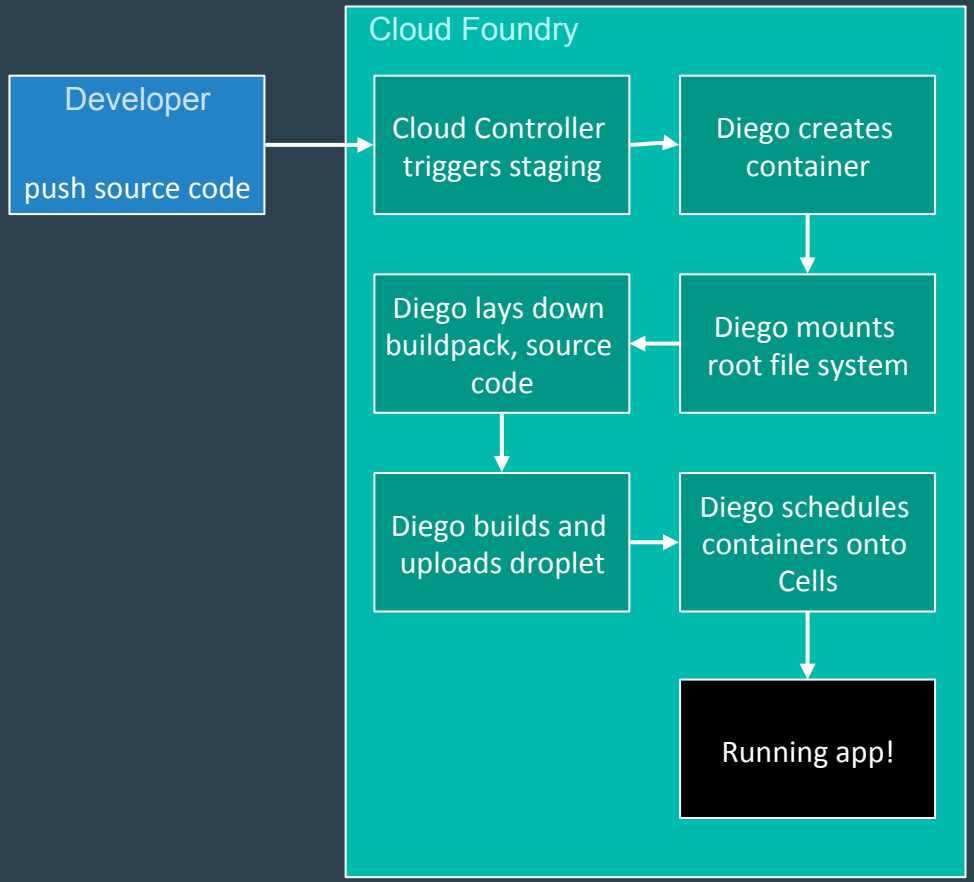
or

Deploy container images that *you* build yourself

Deploy source code that *the platform* containerizes for you

or

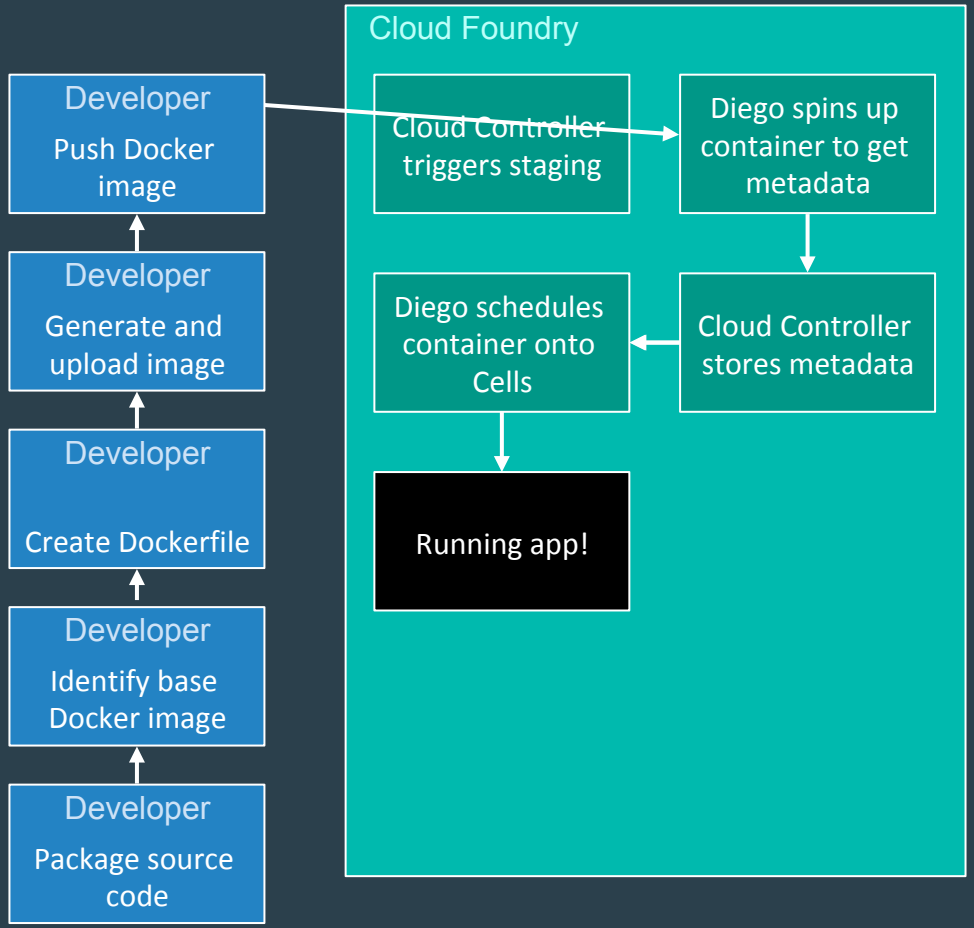
Deploy container images that *you* build yourself



Deploy source code that *the platform* containerizes for you

or

Deploy container images that *you* build yourself



Define application metadata in a manifest file.

- Application manifests tells cf push what to do with applications
- What OS stack to run on: Windows or Linux
- How many instances to create and how much memory to use.
- Helps automate deployment, specially of multiple apps at once
- Can list services to be bound to the application
- YAML format – http://yaml.org

```
1  ---
2  # all applications use these settings and services
3  domain: shared-domain.com
4  memory: 1G
5  instances: 1
6  services:
7  - clockwork-mysql
8  applications:
9  - name: springtock
10   host: tock09876
11   path: ./spring-music/build/libs/spring-music.war
12 - name: springtick
13   host: tick09875
14   path: ./spring-music/build/libs/spring-music.war
```

Things about Pivotal Cloud Foundry you should know.

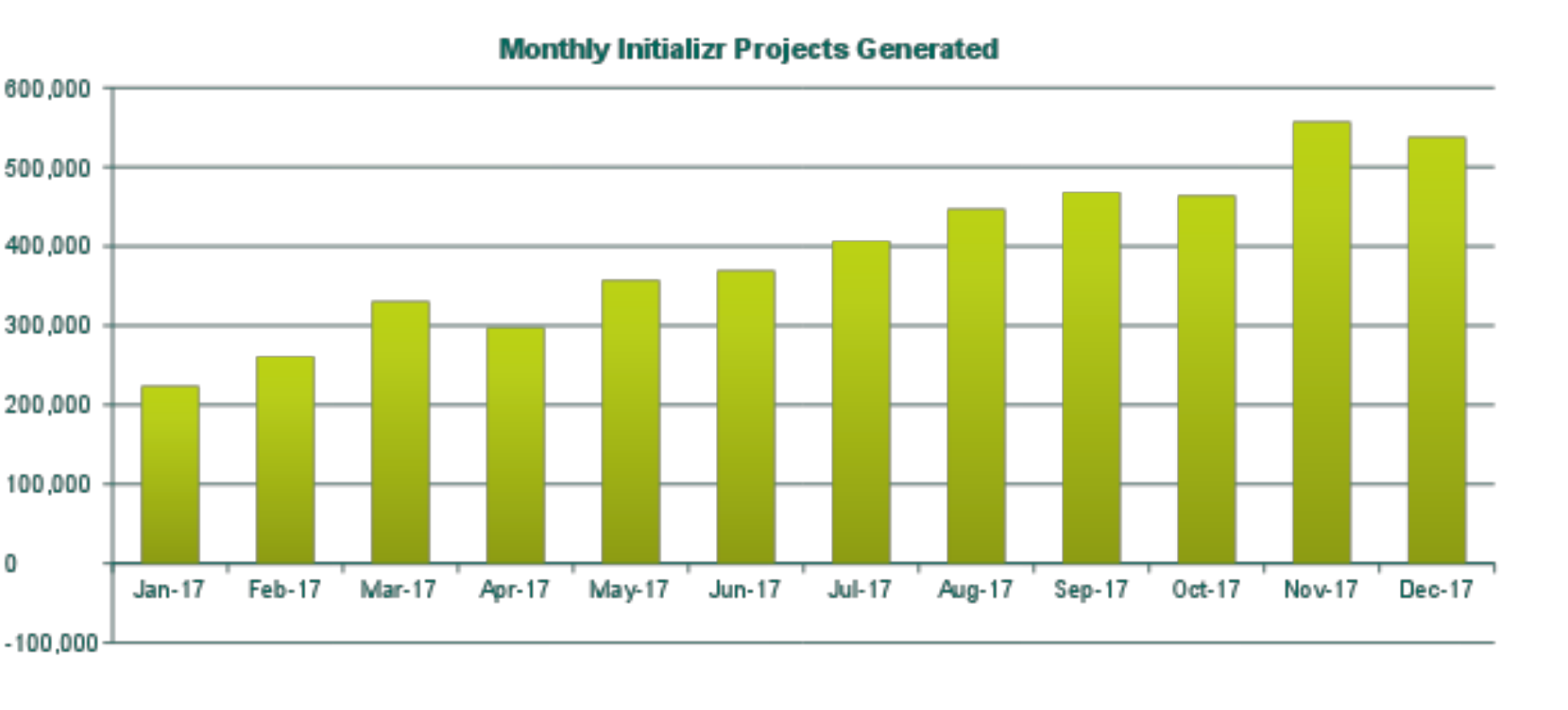
- Built for multi-tenancy
- Push code or containers
- Application manifests written in YAML
- Injects environment variables
- Aggregates app logs, correlates with metrics
- Monitors app health, reacts
- Scale manually or automatically
- Built in data services, marketplace
- Manages apps **and** underlying VMs
- Natively runs on multiple IaaS platforms
- Four layers of high availability built in
- Access via GUI, REST API, CLI
- Dev GUI = Applications Manager
- Ops GUI = Operations Manager

demo

break time


What’s Spring Been Up To?
Pivotal
Pivotal

Spring Boot Popularity



Pivotal

Spring Boot 101

Matt Gunter<mgunter@pivotal.io>
Senior Platform Architect | Pivotal, Tampa

Spring Packaging Is like a Cake

Pivotal

What We Will Cover

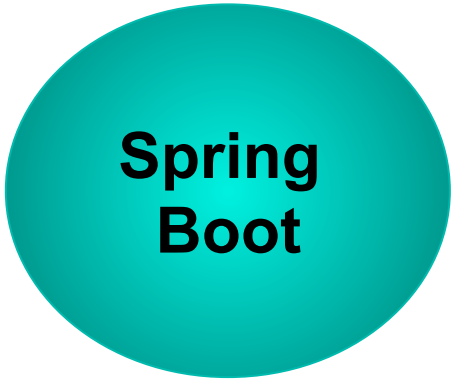
- The Case for Spring Boot
  - Problems to solve
- Spring Boot Basics
  - Where to start?
  - Dependency Management
  - Configuration
  - Metrics
  - Actuator

Challenges with Traditional Apps & Infrastructure

- Shared Dependencies (App Server)
- Tedious Configuration (“Classpath Hell”)
- Intricate Coordination between Devs & Ops
- Granularity of Scaling
- “One Size Fits All” Operations
- Large Teams, Large Releases, Long Lead times

Spring Boot Primary Goals

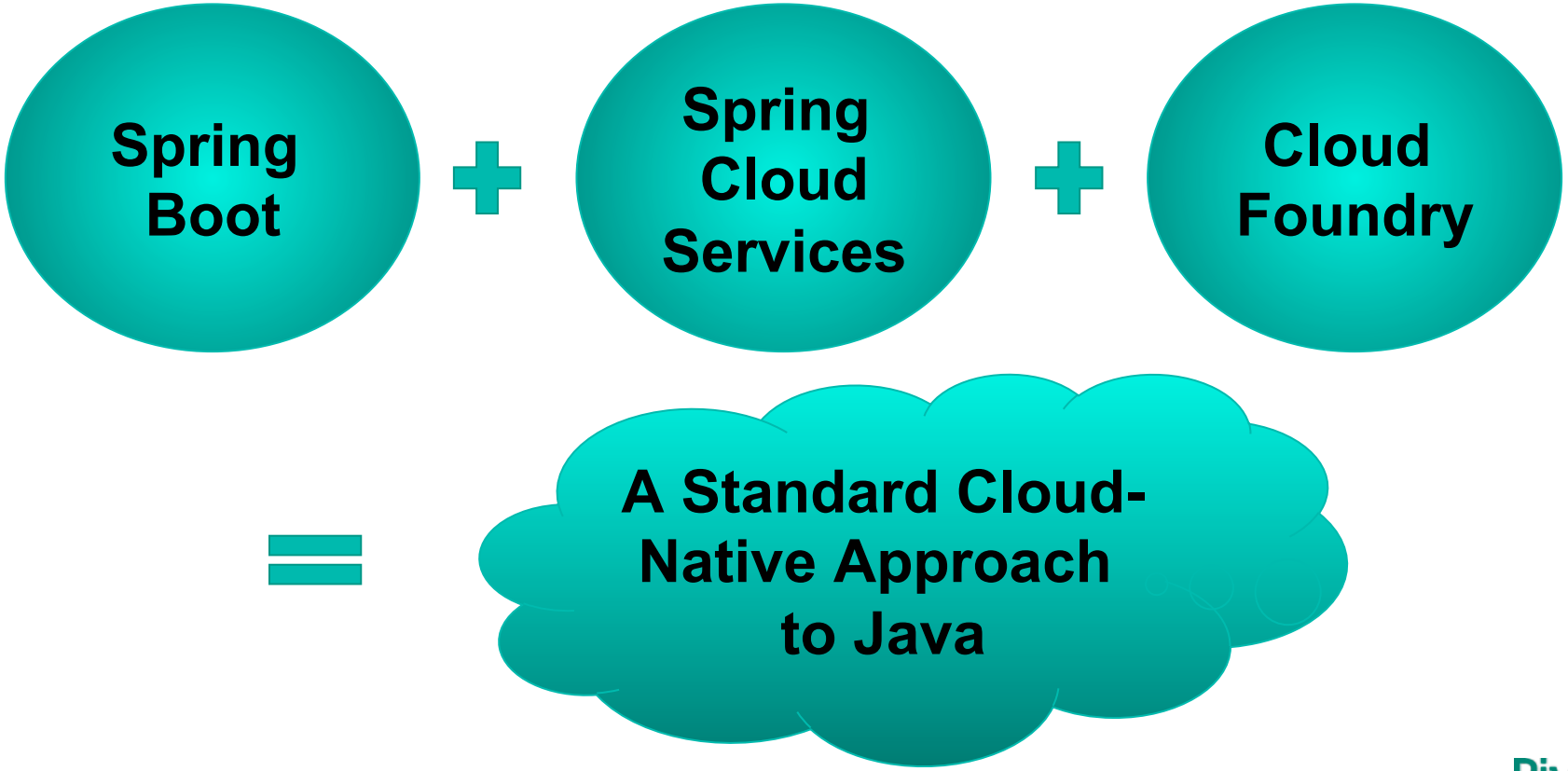
Production-grade Spring-powered apps with minimal fuss



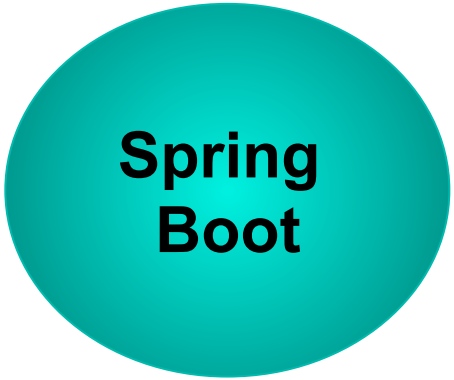
- Make it **fast & easy** to get started with Spring development
- **Be opinionated**, **but get out of the way** quickly when requirements diverge from the defaults
- **Provide** a range of common and useful **non-functional features** (embedded servers, security, metrics, health checks, externalized configuration...)
- **Clean configuration** - no code generation and no requirement for XML configuration

Reference: https://github.com/spring-projects/spring-boot

Cloud-Native Building Blocks



Pivotal



Self-Contained Runnable Unit of Work

- Decoupled from environment
- Executable JAR
- Microservice and devops friendly
- “Just enough” embedded app server
- Autonomous dependency management
- Automated build (maven/gradle)
- Easy to scale horizontally
- Health and telemetry features for operationalization

Environment-Dependent Collection of Logic

- Requires env setup, config, and care
- Non-executable WAR or EAR
- Majority monolithic deployment
- Large external app server
- Classpath dependency, shared libraries
- Build not always automated
- Limited/complex scaling
- Relies on external solutions for health/ telemetry operationalization

> ***Spring Boot lets you pair-program with the Spring team.***

Josh Long, @starbuxman

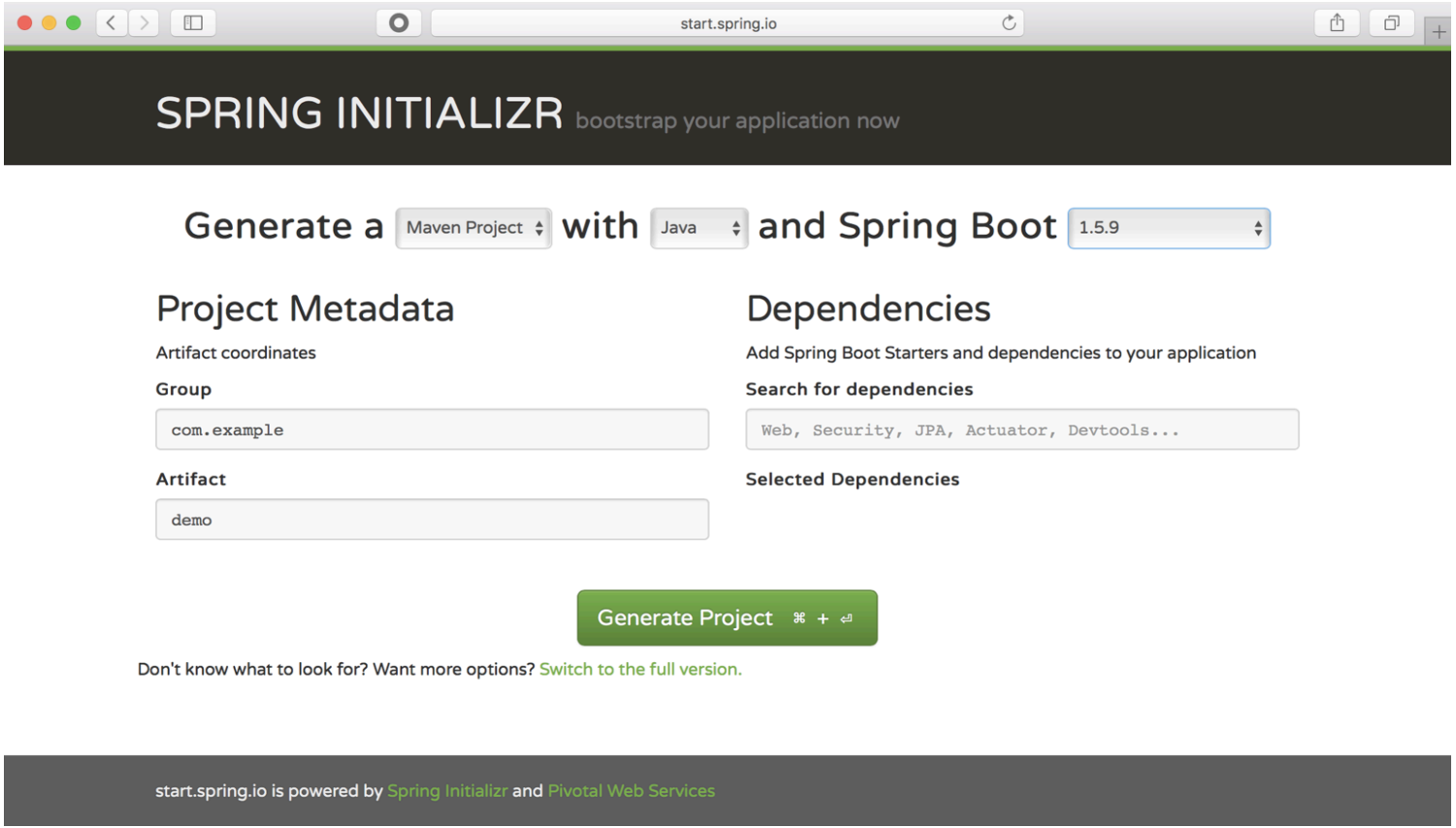
Pivotal

Key Advantages offered by SpringBoot

- Spring Boot provides an **executable scaffolding** for cloud-native applications
- Spring Boot has **built-in opinions** that offer the most common configurations and best practices, but allows you to **easily diverge** when necessary
- Strives to **maximize productivity** for bootstrapping a new application, and for keeping individual components of a broader system independent and isolated from one another
- Spring Boot includes **non-functional features** for production-grade code on Day 1

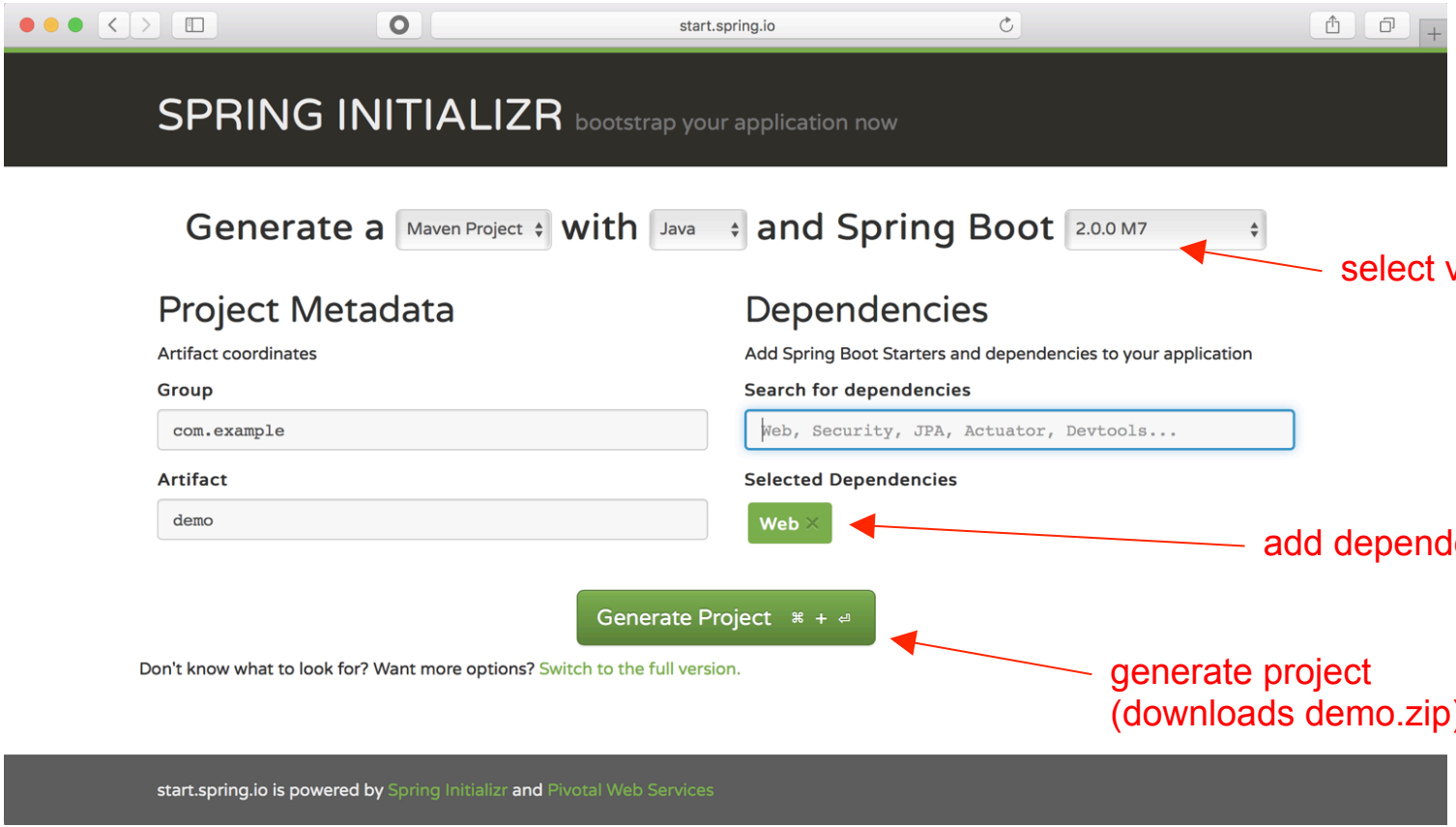
The Basics of Spring Boot

Getting Started: How Easy Is It?



Pivotal

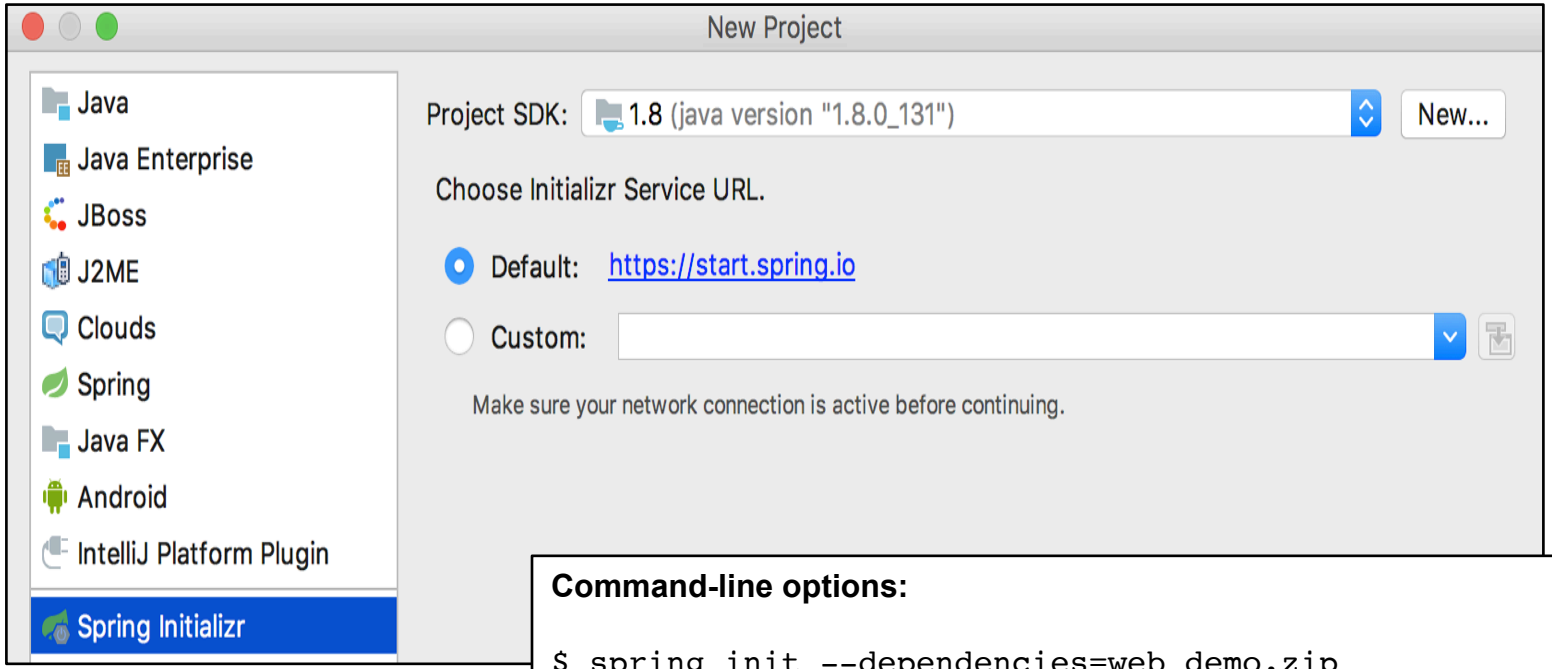
Getting Started: How Easy Is It?



Pivotal

Spring Initializr Alternative Clients

IDEs (STS/IntelliJ/NetBeans) and Command Line (Spring Boot CLI/curl/HTTPie)

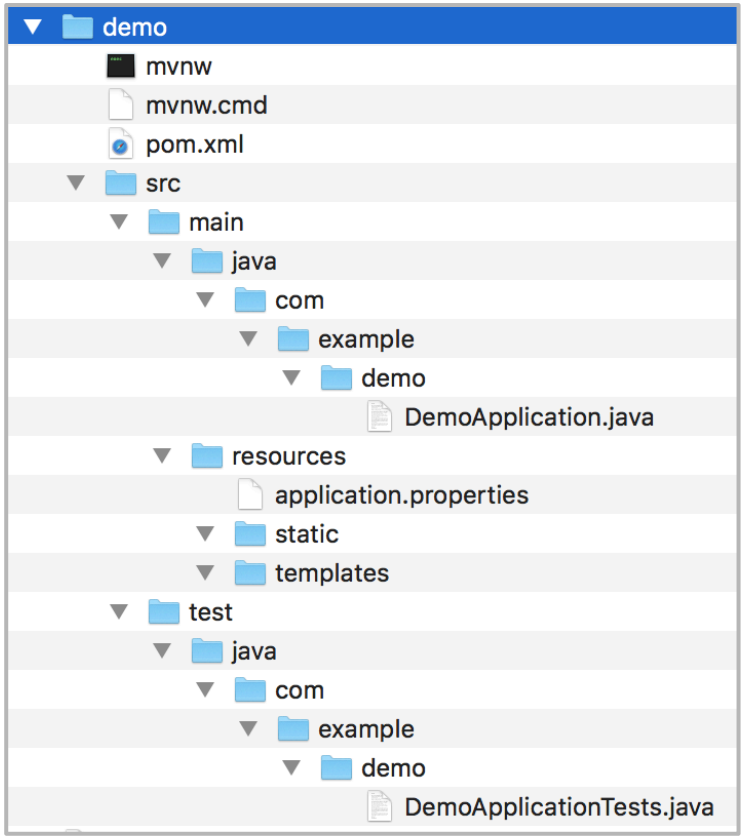


Command-line options:

```
$ spring init --dependencies=web demo.zip

$ curl start.spring.io/starter.zip -d dependencies=web -o demo.zip
```

Pivotal

Skeleton Application



- Project Structure
- Pom File
- Maven Wrapper
- Main & Test Java Classes
- Application Properties File (initially empty)
- Ready to run!

Pivotal

Dependency Management


```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.M7</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Spring Boot Starter Parent defines default properties (e.g. Java version), plugin config, and dependency lists & versions

Pivotal

Dependency Management

- Spring Boot Dependencies contains the versions of included dependencies
- This information is published to Maven repo

```
<name>Spring Boot Dependencies</name>
<description>Spring Boot Dependencies</description>
<url>http://projects.spring.io/spring-boot/</url>
<licenses>
    <license>
        <name>Apache License, Version 2.0</name>
        <url>http://www.apache.org/licenses/LICENSE-2.0</url>
    </license>
</licenses>
<scm>
    <url>https://github.com/spring-projects/spring-boot</url>
</scm>
<properties>
    <!-- Dependency versions -->
    <activemq.version>5.15.2</activemq.version>
    <antlr2.version>2.7.7</antlr2.version>
    <appengine-sdk.version>1.9.60</appengine-sdk.version>
    <artemis.version>2.4.0</artemis.version>
    <aspectj.version>1.8.13</aspectj.version>
    <assertj.version>3.9.0</assertj.version>
    <atomikos.version>4.0.5</atomikos.version>
    <bitronix.version>2.1.4</bitronix.version>
    <byte-buddy.version>1.7.9</byte-buddy.version>
    <caffeine.version>2.6.1</caffeine.version>
```

Feature-Specific Starters

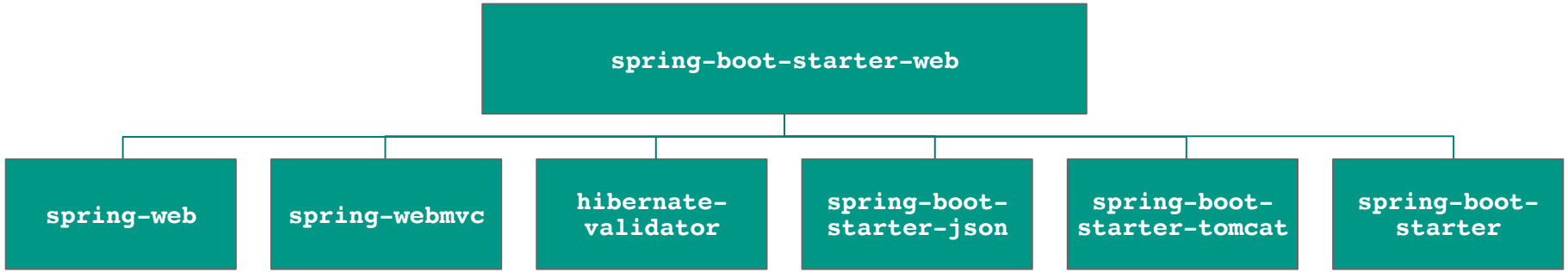
Less dependencies - and
no versions - to manage!

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

Feature-Specific Starters

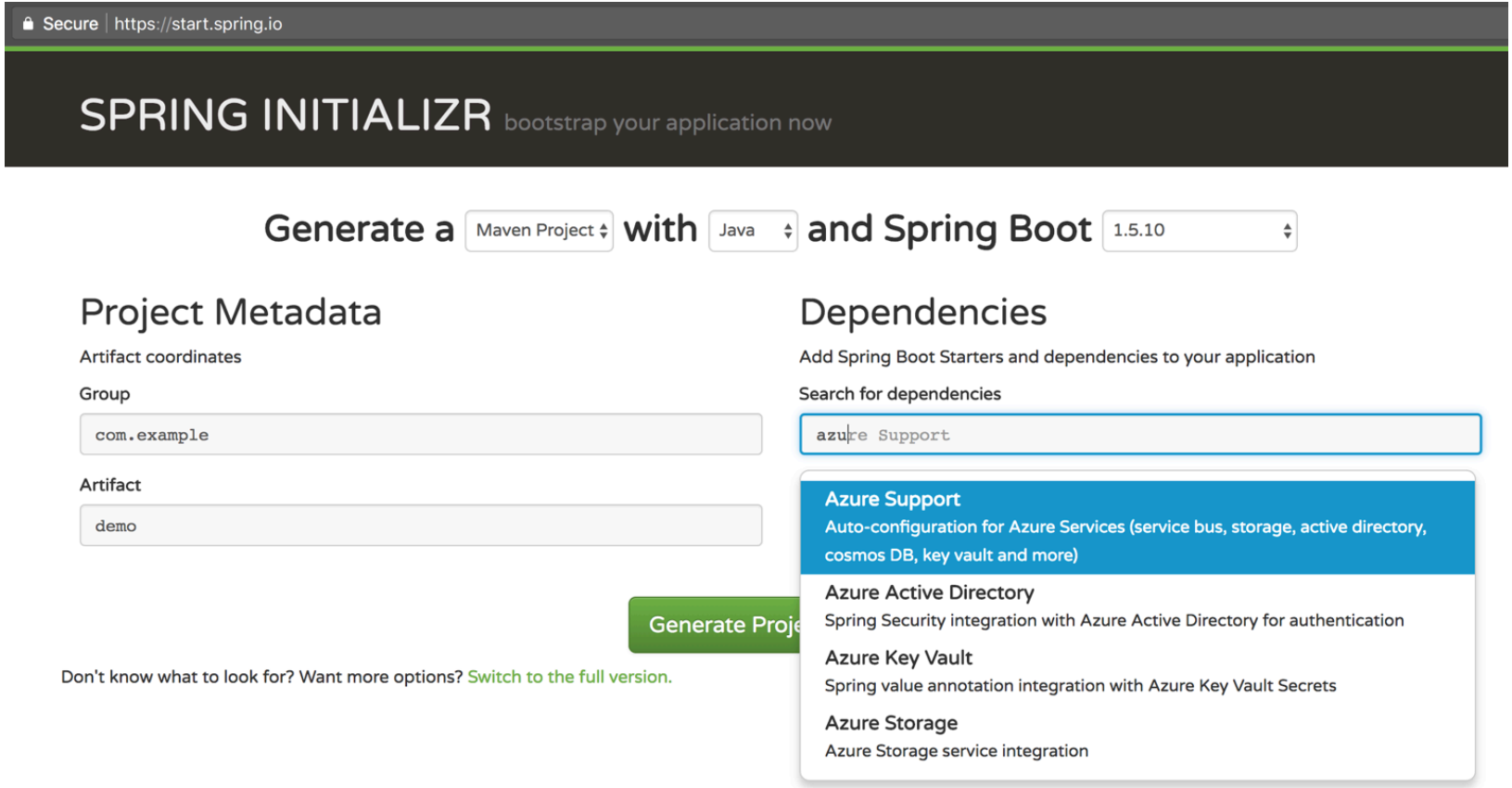
Less dependencies - and
no versions - to manage!

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



Pivotal

Spring Boot Starters for Azure



Pivotal

Skeleton Code - Ready to Run


```
package com.example.demo;

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

@SpringBootApplication
public class DemoApplication {

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

Skeleton Code - Ready to Test


```java
package com.example.demo;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class DemoApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```

Run it!

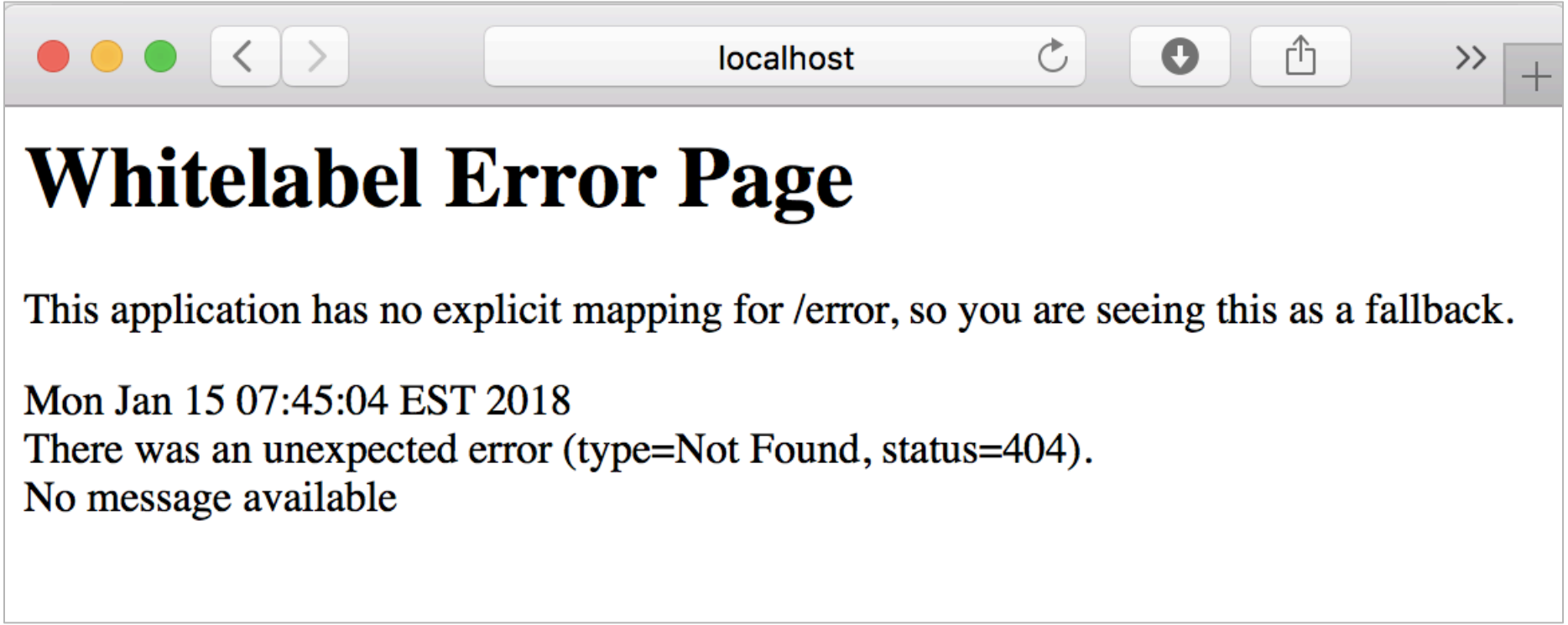
We asked for “Starter Web”
Spring Boot recommends MVC
and embedded Tomcat.

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.0.0.M7)

2018-01-15 07:46:51.550  INFO 50331 --- [           main] com.example.demo.DemoApplication         : Starting DemoApplication on Coras-MacBook-Pro.local with PID 50331
2018-01-15 07:46:51.553  INFO 50331 --- [           main] com.example.demo.DemoApplication         : No active profile set, falling back to default profiles: default
2018-01-15 07:46:51.590  INFO 50331 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationC
2018-01-15 07:46:52.351  INFO 50331 --- [           main] o.h.v.i.engine.ValidatorFactoryImpl      : HV000238: Temporal validation tolerance set to 0.
2018-01-15 07:46:52.542  INFO 50331 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2018-01-15 07:46:52.550  INFO 50331 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-01-15 07:46:52.551  INFO 50331 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.23
2018-01-15 07:46:52.559  INFO 50331 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal per
2018-01-15 07:46:52.620  INFO 50331 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-01-15 07:46:52.621  INFO 50331 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1034 ms
2018-01-15 07:46:52.714  INFO 50331 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Mapping servlet: 'dispatcherServlet' to [/]
2018-01-15 07:46:52.717  INFO 50331 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-01-15 07:46:52.718  INFO 50331 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-01-15 07:46:52.718  INFO 50331 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-01-15 07:46:52.718  INFO 50331 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-01-15 07:46:52.848  INFO 50331 --- [           main] o.h.v.i.engine.ValidatorFactoryImpl      : HV000238: Temporal validation tolerance set to 0.
2018-01-15 07:46:52.939  INFO 50331 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet
2018-01-15 07:46:52.978  INFO 50331 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEn
2018-01-15 07:46:52.979  INFO 50331 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springfram
2018-01-15 07:46:52.997  INFO 50331 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.sprin
2018-01-15 07:46:52.998  INFO 50331 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframewo
2018-01-15 07:46:53.020  INFO 50331 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.s
2018-01-15 07:46:53.111  INFO 50331 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2018-01-15 07:46:53.155  INFO 50331 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2018-01-15 07:46:53.158  INFO 50331 --- [           main] com.example.demo.DemoApplication         : Started DemoApplication in 1.915 seconds (JVM running for 2.471)
```

Tomcat started
on port 8080

Pivotal

Default Error Handling



Strong Opinions Weakly Held

Override defaults by specifying your desired configuration

```
<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>
```

application.properties

```
1  server.port: 9090
2
```

Pivotal

Run it!

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.0.0.M7)

2018-01-17 07:02:37.185  INFO 16684 --- [           main] com.example.demo.DemoApplication         : Starting DemoApplication on Coras-MacBook-Pro.local with PID 16684 (/Us
2018-01-17 07:02:37.189  INFO 16684 --- [           main] com.example.demo.DemoApplication         : No active profile set, falling back to default profiles: default
2018-01-17 07:02:37.324  INFO 16684 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfi
2018-01-17 07:02:38.132  INFO 16684 --- [           main] o.h.v.i.engine.ValidatorFactoryImpl      : HV000238: Temporal validation tolerance set to 0.
2018-01-17 07:02:38.199  INFO 16684 --- [           main] org.eclipse.jetty.util.log               : Logging initialized @1978ms to org.eclipse.jetty.util.log.Slf4jLog
2018-01-17 07:02:38.249  INFO 16684 --- [           main] o.s.b.w.e.j.JettyServletWebServerFactory : Server initialized with port: 9090
2018-01-17 07:02:38.251  INFO 16684 --- [           main] org.eclipse.jetty.server.Server          : jetty-9.4.7.v20170914
2018-01-17 07:02:38.329  INFO 16684 --- [           main] org.eclipse.jetty.server.session         : DefaultSessionIdManager workerName=node0
2018-01-17 07:02:38.329  INFO 16684 --- [           main] org.eclipse.jetty.server.session         : No SessionScavenger set, using defaults
2018-01-17 07:02:38.330  INFO 16684 --- [           main] org.eclipse.jetty.server.session         : Scavenging every 660000ms
2018-01-17 07:02:38.334  INFO 16684 --- [           main] o.e.j.s.h.ContextHandler.application     : Initializing Spring embedded WebApplicationContext
2018-01-17 07:02:38.334  INFO 16684 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1016 ms
2018-01-17 07:02:38.422  INFO 16684 --- [           main] o.s.b.w.servlet.ServletRegistrationBean  : Mapping servlet: 'dispatcherServlet' to [/]
2018-01-17 07:02:38.424  INFO 16684 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-01-17 07:02:38.424  INFO 16684 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-01-17 07:02:38.424  INFO 16684 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-01-17 07:02:38.424  INFO 16684 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-01-17 07:02:38.427  INFO 16684 --- [           main] o.e.jetty.server.handler.ContextHandler  : Started o.s.b.w.e.j.JettyEmbeddedWebAppContext@325f7fa9{/,[file:///priva
2018-01-17 07:02:38.428  INFO 16684 --- [           main] org.eclipse.jetty.server.Server          : Started @2207ms
2018-01-17 07:02:38.531  INFO 16684 --- [           main] o.h.v.i.engine.ValidatorFactoryImpl      : HV000238: Temporal validation tolerance set to 0.
2018-01-17 07:02:38.668  INFO 16684 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.con
2018-01-17 07:02:38.705  INFO 16684 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/]}" onto java.lang.String com.example.demo.DemoApplication.h
2018-01-17 07:02:38.710  INFO 16684 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity
2018-01-17 07:02:38.710  INFO 16684 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframewor
2018-01-17 07:02:38.732  INFO 16684 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springfra
2018-01-17 07:02:38.732  INFO 16684 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.w
2018-01-17 07:02:38.765  INFO 16684 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.spring
2018-01-17 07:02:38.864  INFO 16684 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2018-01-17 07:02:38.877  INFO 16684 --- [           main] o.e.j.s.h.ContextHandler.application     : Initializing Spring FrameworkServlet 'dispatcherServlet'
2018-01-17 07:02:38.877  INFO 16684 --- [           main] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2018-01-17 07:02:38.886  INFO 16684 --- [           main] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 9 ms
2018-01-17 07:02:38.918  INFO 16684 --- [           main] o.e.jetty.server.AbstractConnector       : Started ServerConnector@151335cb{HTTP/1.1,[http/1.1]}{0.0.0.0:9090}
2018-01-17 07:02:38.922  INFO 16684 --- [           main] o.s.b.web.embedded.jetty.JettyWebServer  : Jetty started on port(s) 9090 (http/1.1) with context path '/'
2018-01-17 07:02:38.925  INFO 16684 --- [           main] com.example.demo.DemoApplication         : Started DemoApplication in 2.157 seconds (JVM running for 2.704)
```

Jetty started
on port 9090

Pivotal

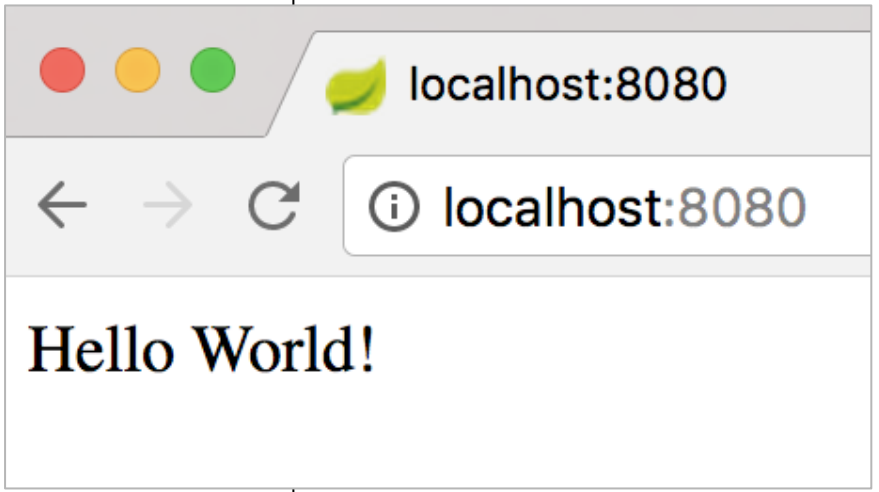
Spring Boot Hello World Web App

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@SpringBootApplication
public class DemoApplication {

  @RequestMapping("/")
  String home() {
    return "Hello World!";
  }

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

Pivotal

Spring Boot Hello World Web App

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@SpringBootApplication
public class DemoApplication {

  @RequestMapping("/")
  String home() {
    return "Hello World!";
  }

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



Pivotal

What Happened?

Demystifying @SpringBootApplication

Convenience annotation equivalent to:

@Configuration

@ComponentScan

@EnableAutoConfiguration

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {

    /**
     * Exclude specific auto-configuration classes such that they will never be applied.
     * @return the classes to exclude
     */
    @AliasFor(annotation = EnableAutoConfiguration.class, attribute = "exclude")
    Class<?>[] exclude() default {};

    /**
     * Exclude specific auto-configuration class names such that they will never be
     * applied.
     * @return the class names to exclude
     * @since 1.3.0
     */
    @AliasFor(annotation = EnableAutoConfiguration.class, attribute = "excludeName")
    String[] excludeName() default {};

    /**
     * Base packages to scan for annotated components. Use {@link #scanBasePackageClasses}
     * for a type-safe alternative to String-based package names.
```

Pivotal

Remember, Spring Boot Provides…

- Easy to get started
- **Starters** dramatically simplify dependency management
- SpringBoot packaging as self-contained, **executable (“fat”) jar**
- **Opinionated** - like pairing with the Spring team!
- **Configurable** - easy to diverge from opinionated defaults
- IOC “magic” via **auto-configuration**

Actuator

Metrics

- Record
- Gauges
- Counters
- Compose

Health

- Disk Space
- Data Source Connection
- JMS
- Rabbit
- Redis
- ...

Data

- Liquibase
- Flyway
- Configuration

Info

- Build Info
- Git Version
- Build Date

Pivotal

Actuator Information- Types

Configuration
- /beans
- /autoconfig
- /env
- /configprops
- /controller

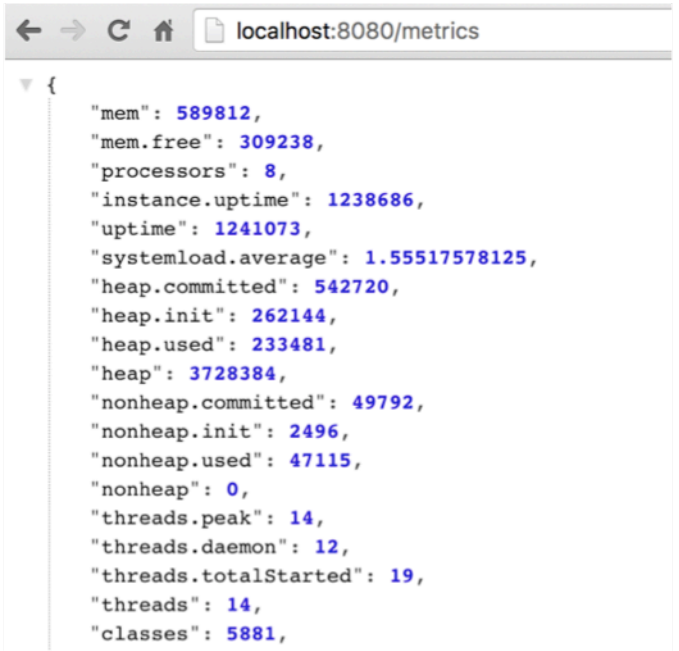
Metrics
- /metrics
- /trace
- /dump

Miscellaneous
- /shutdown
- /info

/metrics

Report application metrics

Memory usage
HTTP Request Counters
And more...



Pivotal

Auto-Configured Health Indicators

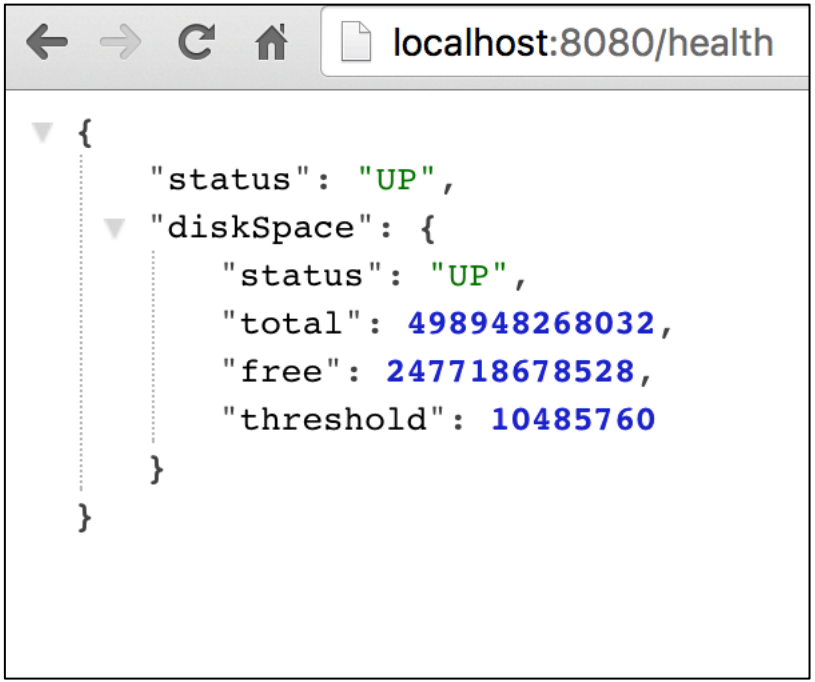
`DiskSpaceHealthIndicator`	Checks for low disk space
`DataSourceHealthIndicator`	Validates DataSource connection
`ElasticsearchHealthIndicator`	Checks if Elasticsearch is up
`JmsHealthIndicator`	Checks if a JMS broker is up
`MailHealthIndicator`	Checks if a mail server is up
`MongoHealthIndicator`	Checks if MongoDB is up
`RabbitHealthIndicator`	Checks if RabbitMQ server is up
`RedisHealthIndicator`	Checks if Redis server is up
`SolrHealthIndicator`	Checks if Solr server is up
`CassandraHealthIndicator`	Checks if Cassandra DB is up

Pivotal

/health

Reports health metrics for the App

- Each check returns a Health object
  - status
    - UP
    - DOWN
    - UNKNOWN
    - OUT_OF_SERVICE
- Custom….



Pivotal

Custom Health Indicator

```
import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class MyHealthIndicator implements HealthIndicator {

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

}
```


Pivotal

Spring Profiles

Why?

- Flexibility
- Runtime
- Feature Control

```
server:
    port: 9000
---

spring:
    profiles: development
server:
    port: 9001

---

spring:
    profiles: production
server:
    port: 0
```


Pivotal

SESSION FOUR

Questions?

Pivotal

SESSION FIVE

Spring Cloud Services

Pivotal

Introducing Spring Cloud Services

- Eureka Service **Registry for Service Discovery**
- Hystrix **Circuit Breaker** Dashboard (with Turbine Metrics Aggregation)
- Spring Cloud **Config** Server

Spring Cloud ….how we got here

NETFLIX | OSS

Client

- Ribbon
- Archaius
- **Hystrix**
- Feign
- Zuul

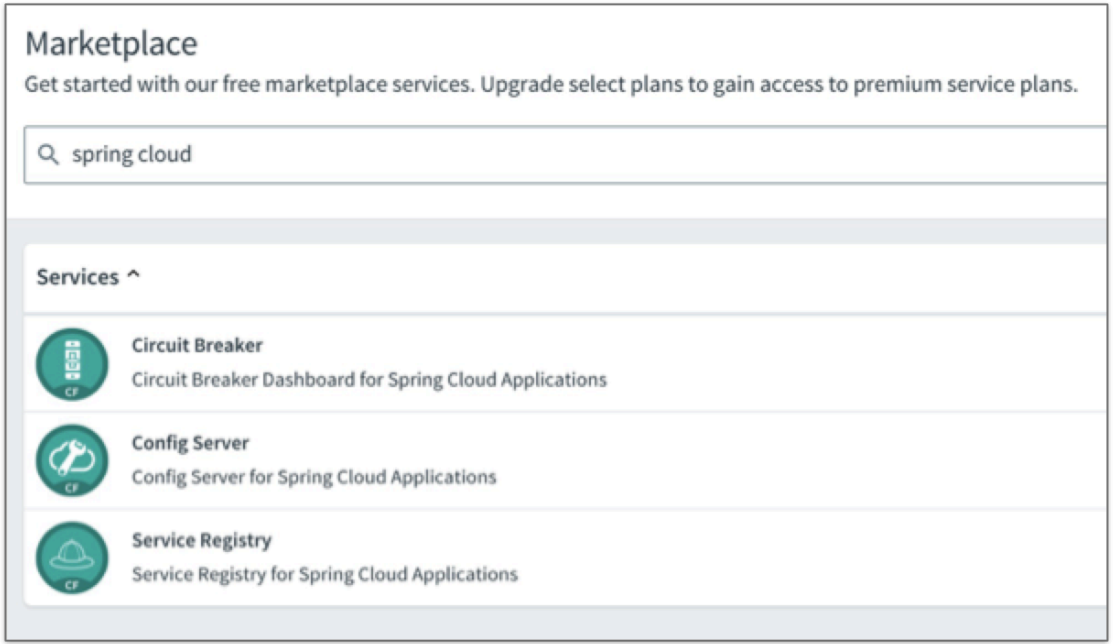
Server

- **Eureka**
- **Turbine**

Pivotal

Cloud Foundry Marketplace

Filter for Spring Cloud...



Pivotal

Why Microservices?

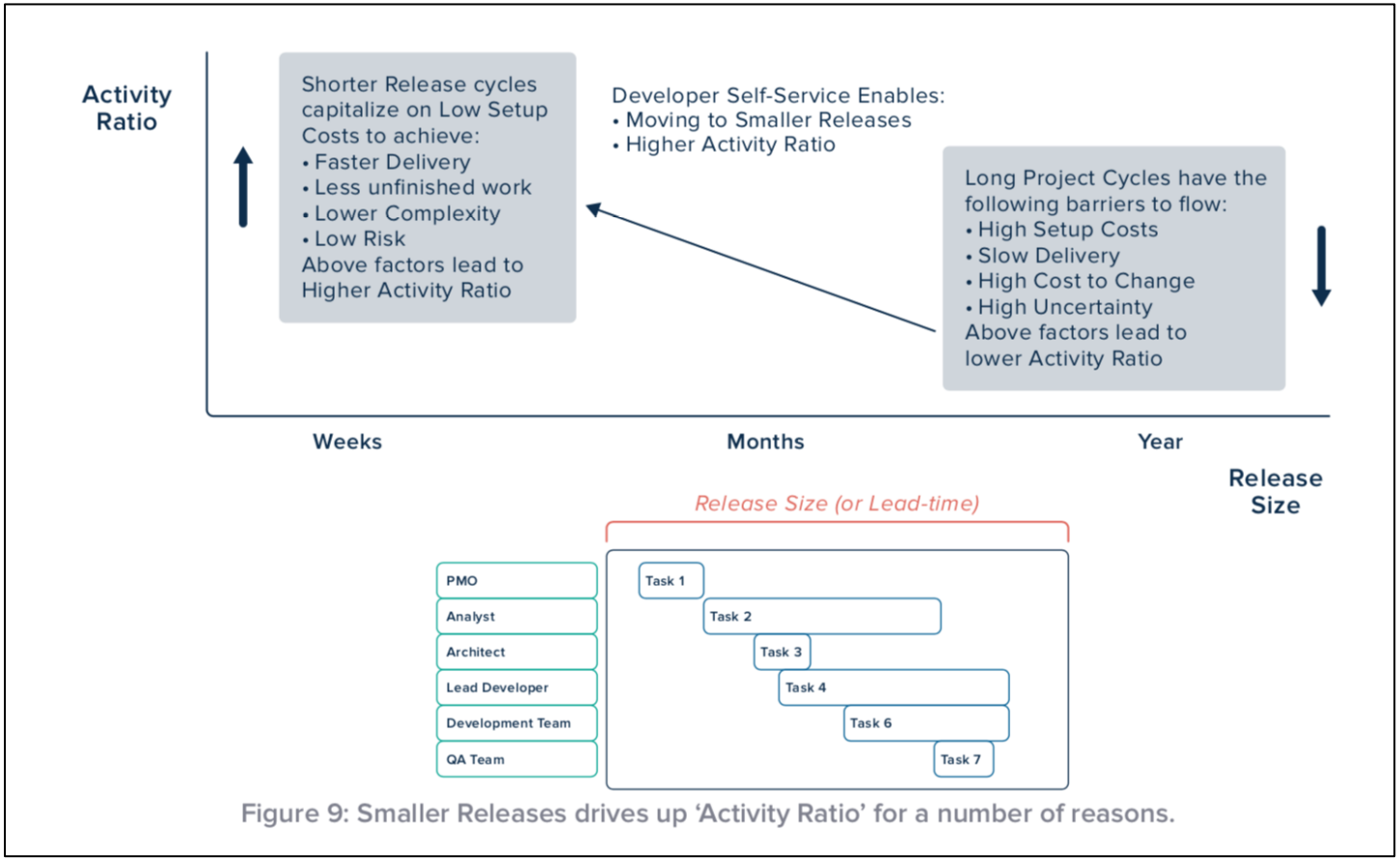


Figure 9: Smaller Releases drives up 'Activity Ratio' for a number of reasons.

Pivotal

What's Harder with Microservices?

- Must maintain and deploy more codebases, more applications. Without automation, this can become a real problem
- Service calls are no longer a method call away
- Application becomes a distributed system: distributed computing is hard

What's Harder with Microservices?

- Must maintain and deploy more codebases, more applications. Without automation, this can become a real problem
- Service calls are no longer a method call away
- **Application becomes a distributed system: distributed computing is hard**

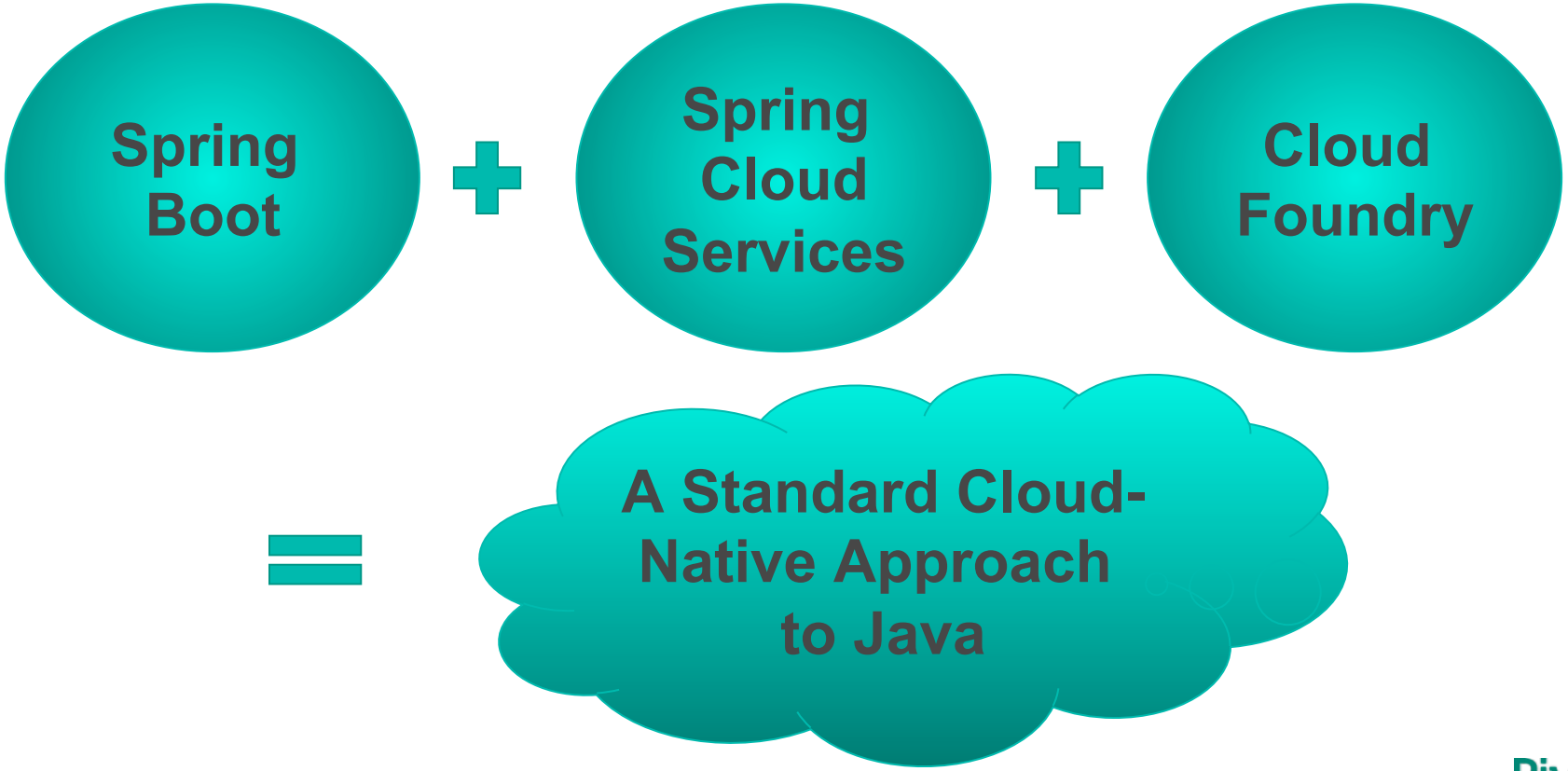
This will be our focus for today

Challenges of Distributed Systems

- **Configuration management**
- **Registration and discovery**
- Routing and load balancing
- **Fault tolerance and isolation**
- Aggregation and transformation
- Monitoring and distributed tracing
- Process management

These will be our focus for today

Cloud-Native Building Blocks



Pivotal

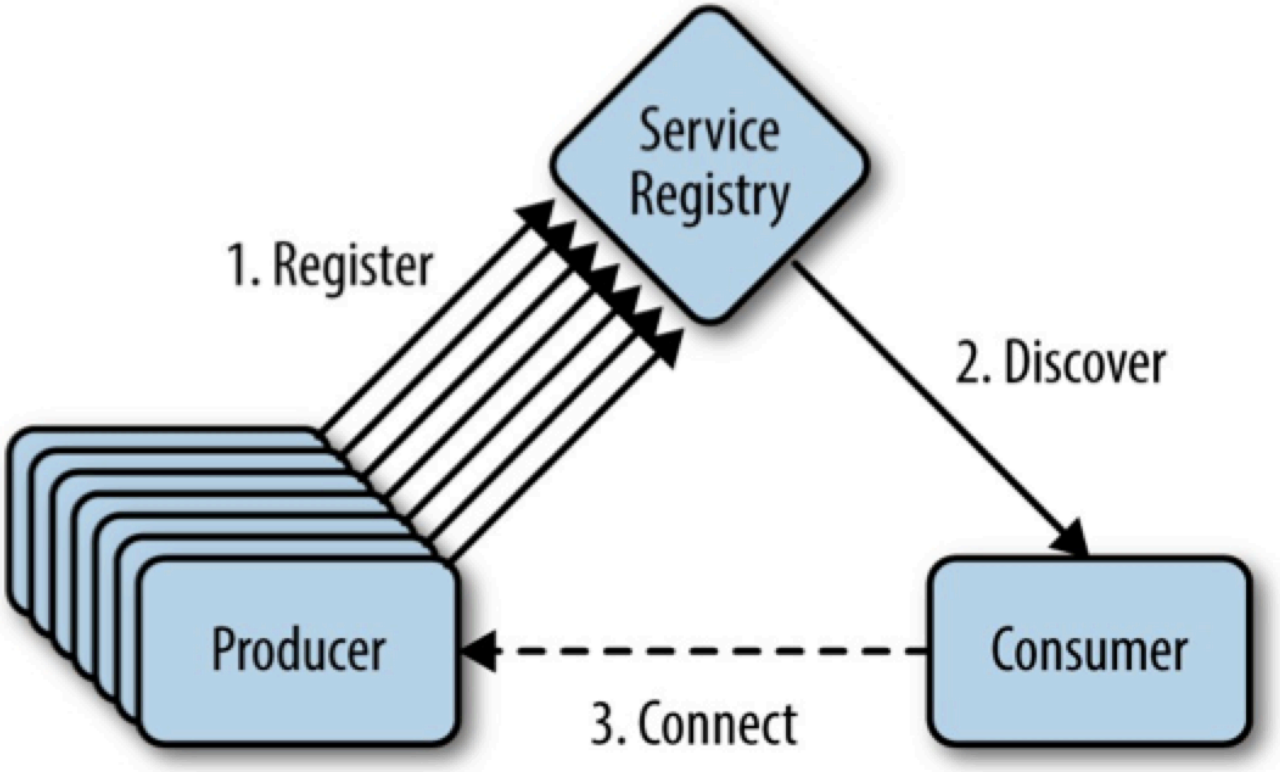
Spring Cloud Services

Production-grade Spring Boot-powered app **services** with minimal fuss

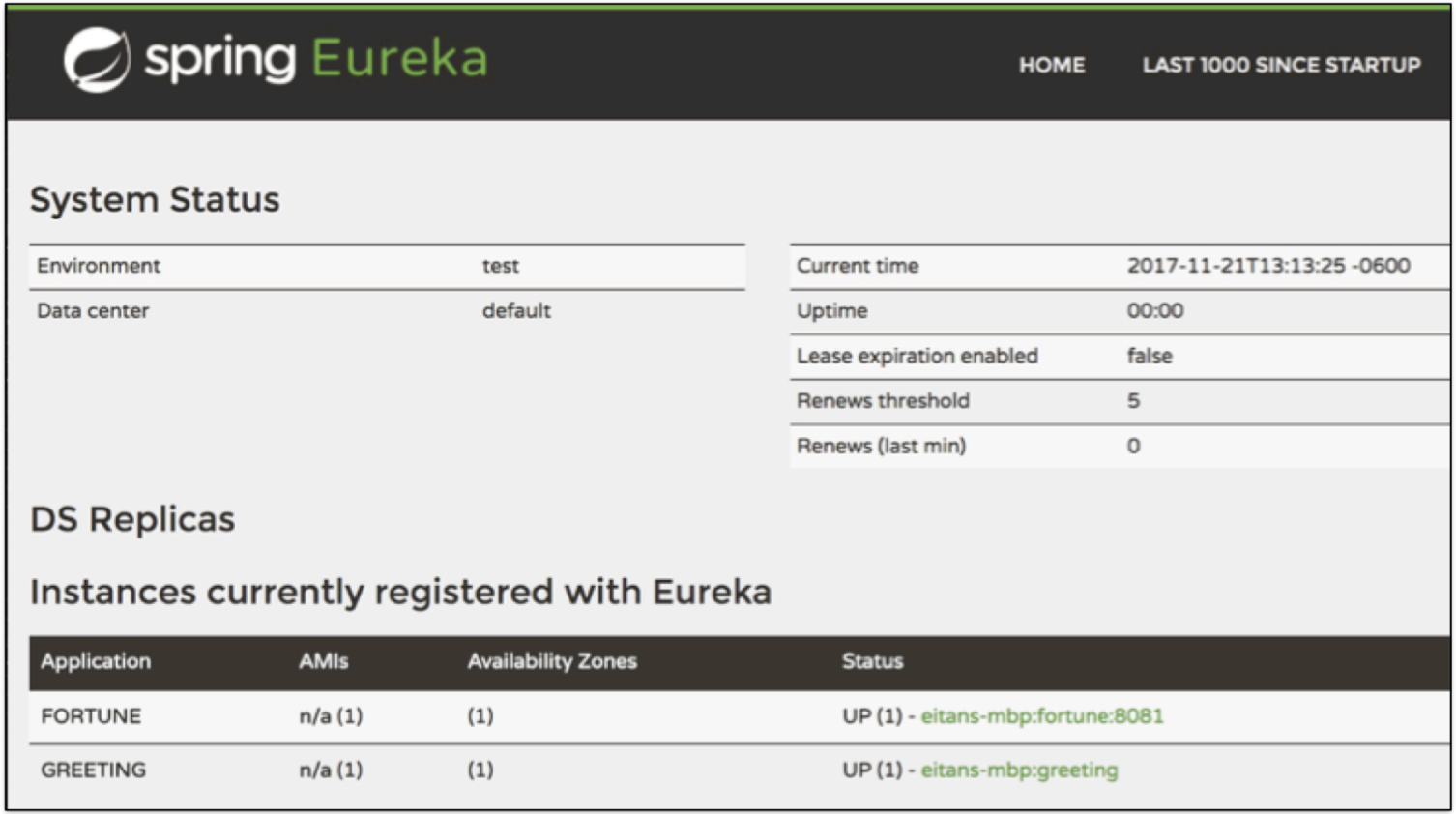
- Spring Cloud + Netflix OSS + PCF Marketplace
  - Select group of Spring Cloud projects provided as managed services in PCF
- Spring Cloud Config for externalizing **configuration**
- Spring Cloud Netflix and Eureka for **locating** services
- Spring Cloud Netflix, Hystrix, and Turbine for **protecting** your services against cascading effects of latency and failure

Service Discovery

General Concept



The Eureka Dashboard



Pivotal

Eureka Lookup Example

```
29      String getFortune() {
30        String fortuneUrl = lookupUrlFor( appName: "FORTUNE");
31        Map map = restTemplate.getForObject(fortuneUrl, Map.class);
32        return (String) map.get("fortune");
33      }
34
35      private String lookupUrlFor(String appName) {
36        InstanceInfo instance = eurekaClient.getNextServerFromEureka(appName,  secure: false);
37        return instance.getHomePageUrl();
38      }
```

Eureka Lookup Example… Using Feign

```
@SpringBootApplication
@EnableFeignClients
@EnableDiscoveryClient
public class CloudNativeSpringUiApplication {

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

    @FeignClient("cloud-native-spring")
    public interface CityClient {

        @GetMapping(value = "/cities", consumes = "application/hal+json")
        Resources<City> getCities();
    }
}
```

Circuit Breakers

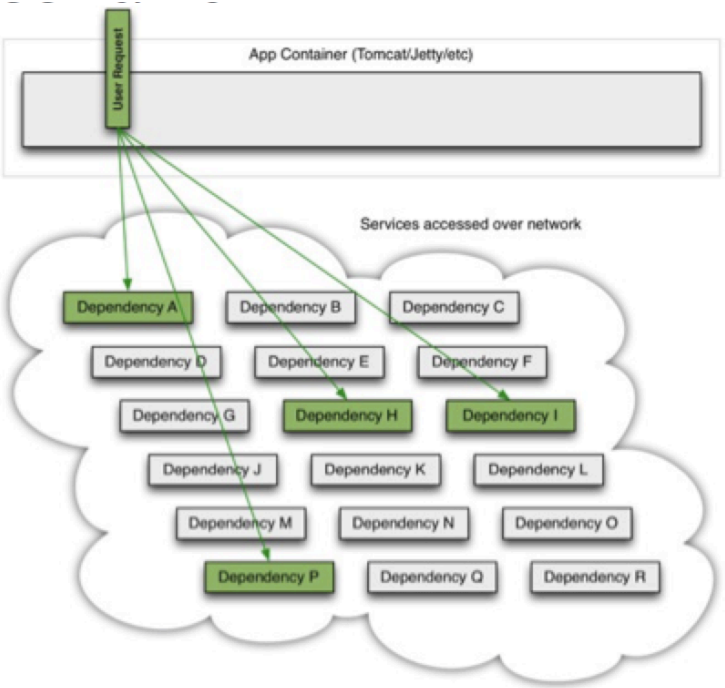
Hystrix Circuit Breaker Framework

“Hystrix adds **latency tolerance and fault tolerance** logic Transparently to your app.

Like the ***electrical circuit breakers*** in your house, Hystrix works by isolating the interactions between the services, stopping any cascading failures across them, and providing fallback options, all of which improve your system’s overall resiliency.”

Service Dependency Scenario

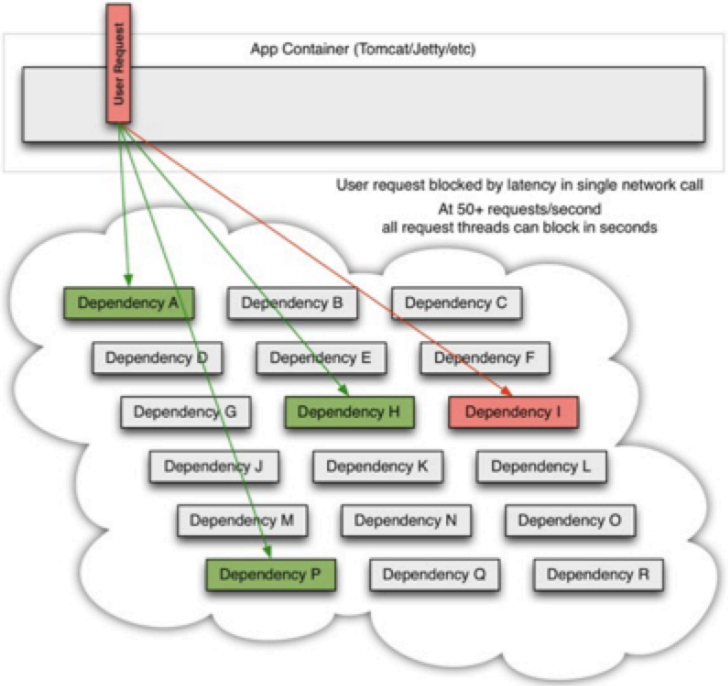
- A typical application depending on a number of backing services
- All services are up and behaving normally
- Circuit is *Closed*



Pivotal

Failing Dependency

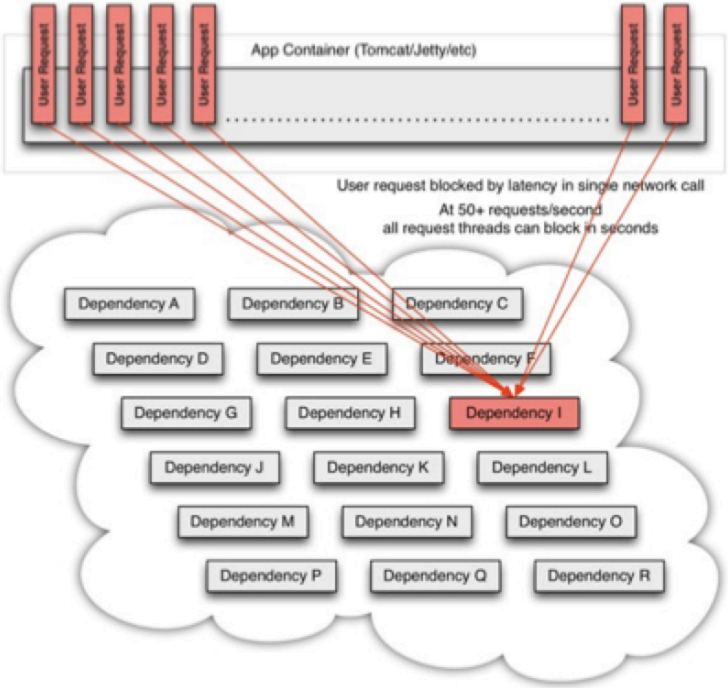
- A dependency begins misbehaving
- Response latency increases, tying up thread in calling application



Pivotal

Failure Cascades to Caller

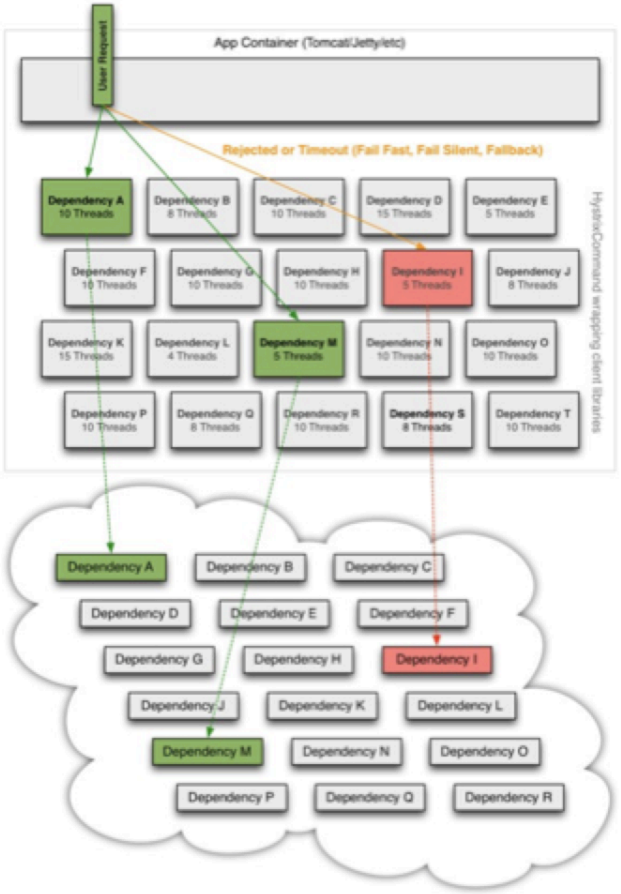
- Calling application's thread pool is exhausted waiting on misbehaving dependency
- Failure cascades to caller



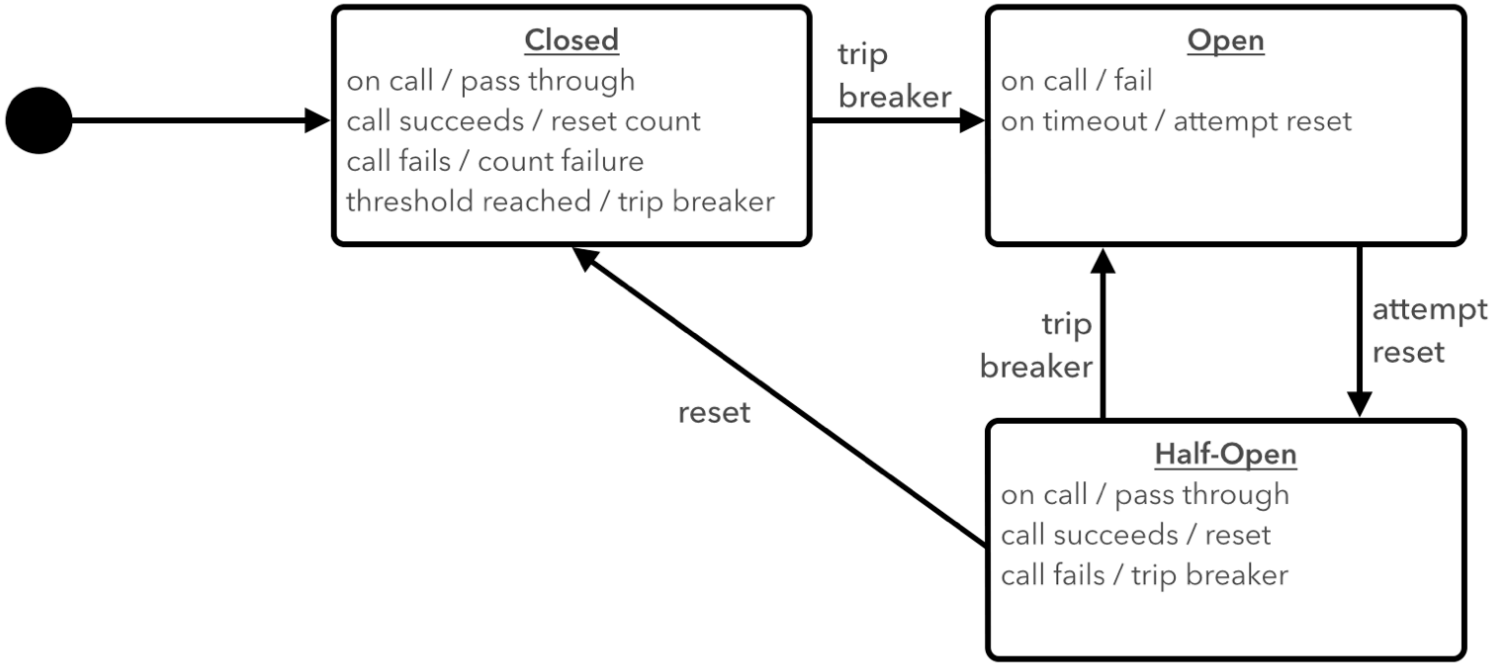
Pivotal

Application Protected with Hystrix

- Application is isolated from a misbehaving backing service
- When backing service health is restored, calling application will automatically reconfigure itself to call it once more



Pivotal

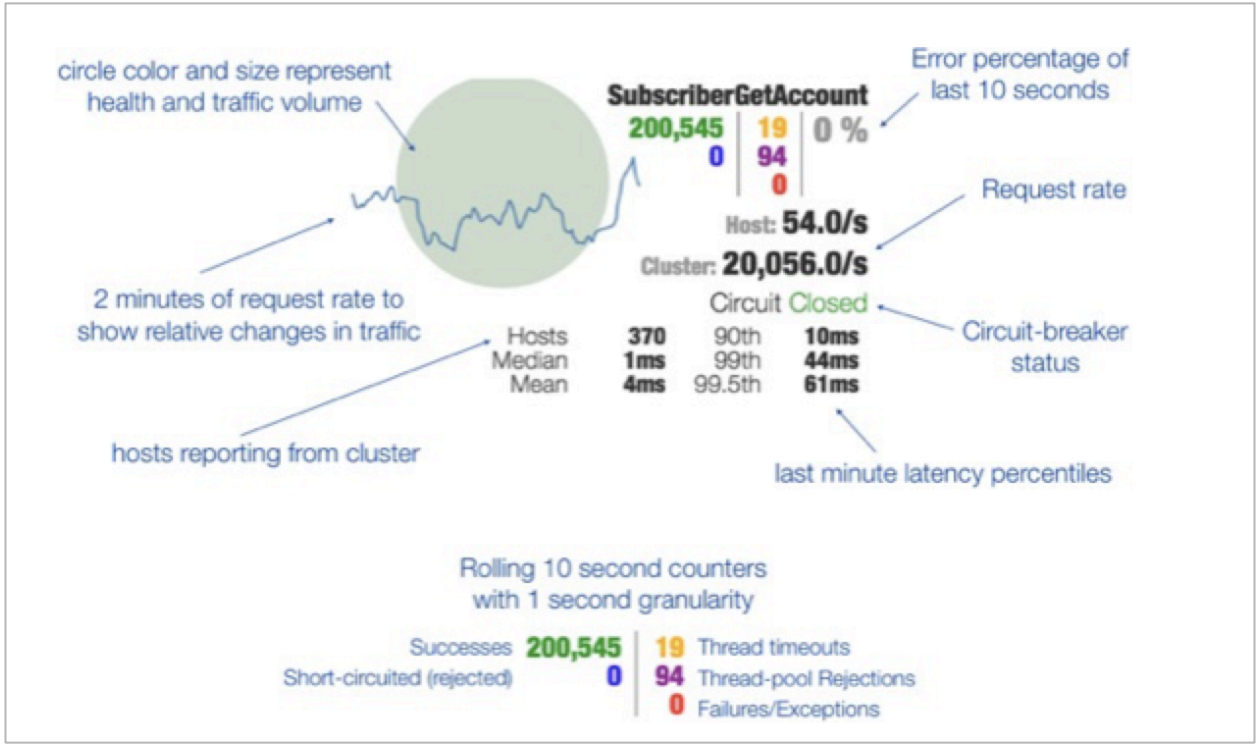

Closed
on call / pass through
call succeeds / reset count
call fails / count failure
threshold reached / trip breaker
trip
breaker
Open
on call / fail
on timeout / attempt reset
attempt
reset
trip
breaker
reset
Half-Open
on call / pass through
call succeeds / reset
call fails / trip breaker

Annotating a Service Call

1. Add @EnableCircuitBreaker annotation to main application class
2. Annotate methods to add individual circuit breakers and fallbacks

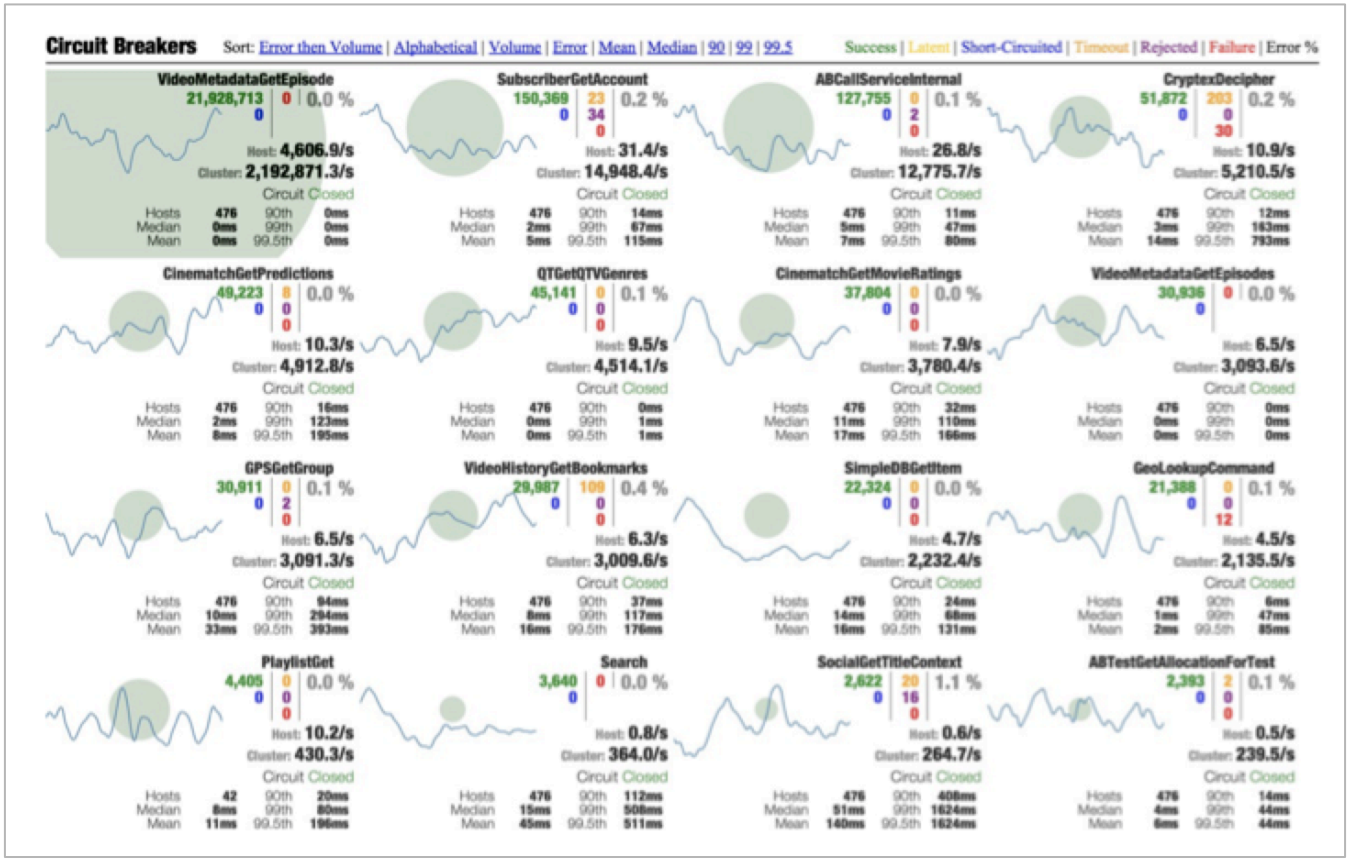
```
25    @HystrixCommand(fallbackMethod = "defaultFortune")
26    String getFortune() {
27      Map map = restTemplate.getForObject(fortuneUrl, Map.class);
28      return (String) map.get("fortune");
29    }
30
31    String defaultFortune() {
32      log.info("Default fortune used");
33      return "Your future is uncertain";
34    }
```

Circuit Breaker Monitoring



Pivotal

Example Dashboard

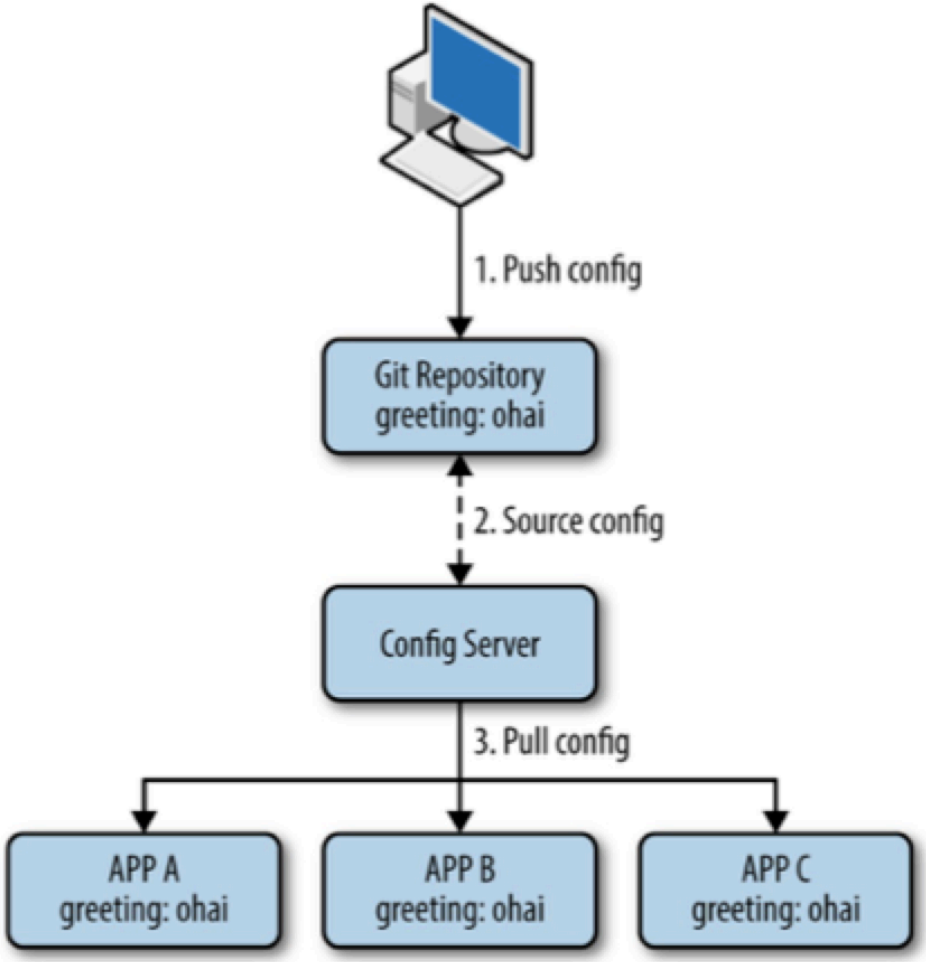


Pivotal

Spring Cloud Config

Config Server - Main Concepts

- **Externalization** of configuration (outside the application)
- **Centralization** of configuration information for multiple services and environments
- Configuration *as a service*
  - i.e. **HTTP/REST endpoints** for reading application configuration


1. Push config
Git Repository
greeting: ohai
2. Source config
Config Server
3. Pull config
APP A
greeting: ohai
APP B
greeting: ohai
APP C
greeting: ohai

Design

- Notion of a "**backend**" where configuration files are stored
- Config server reads configurations from the back-end and exposes HTTP **REST endpoints** for applications to consume
- **Conventions**: how files and REST endpoints are named make it easy to expose configuration for different applications in different environments

Backends

- Supports multiple types of backends
  - Git,
  - Subversion,
  - HashiCorp Vault,
  - File System,
  - JDBC
- Also supports a composite of backends

Backend File Naming

- Default Pattern:

{application}-{profile}.[properties|yml]

- Example:

spring.application.name: **greeting** spring.profiles.active: **qa**

Configuration stored in a file: **greeting-qa.yml** (or greeting-qa.properties*)*

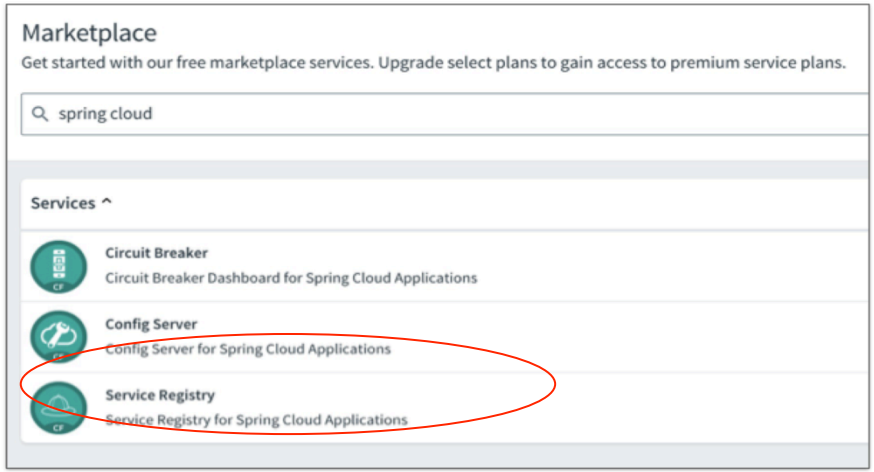
NOTE: The pattern can be customized via a configuration property named searchPaths

Pivotal

Server-Side Configuration

1. Create an instance of Config Server in your space
2. Configuration git backend to a public repository

```
{"git": { "uri": "http://example.com/config" } }
```



Pivotal

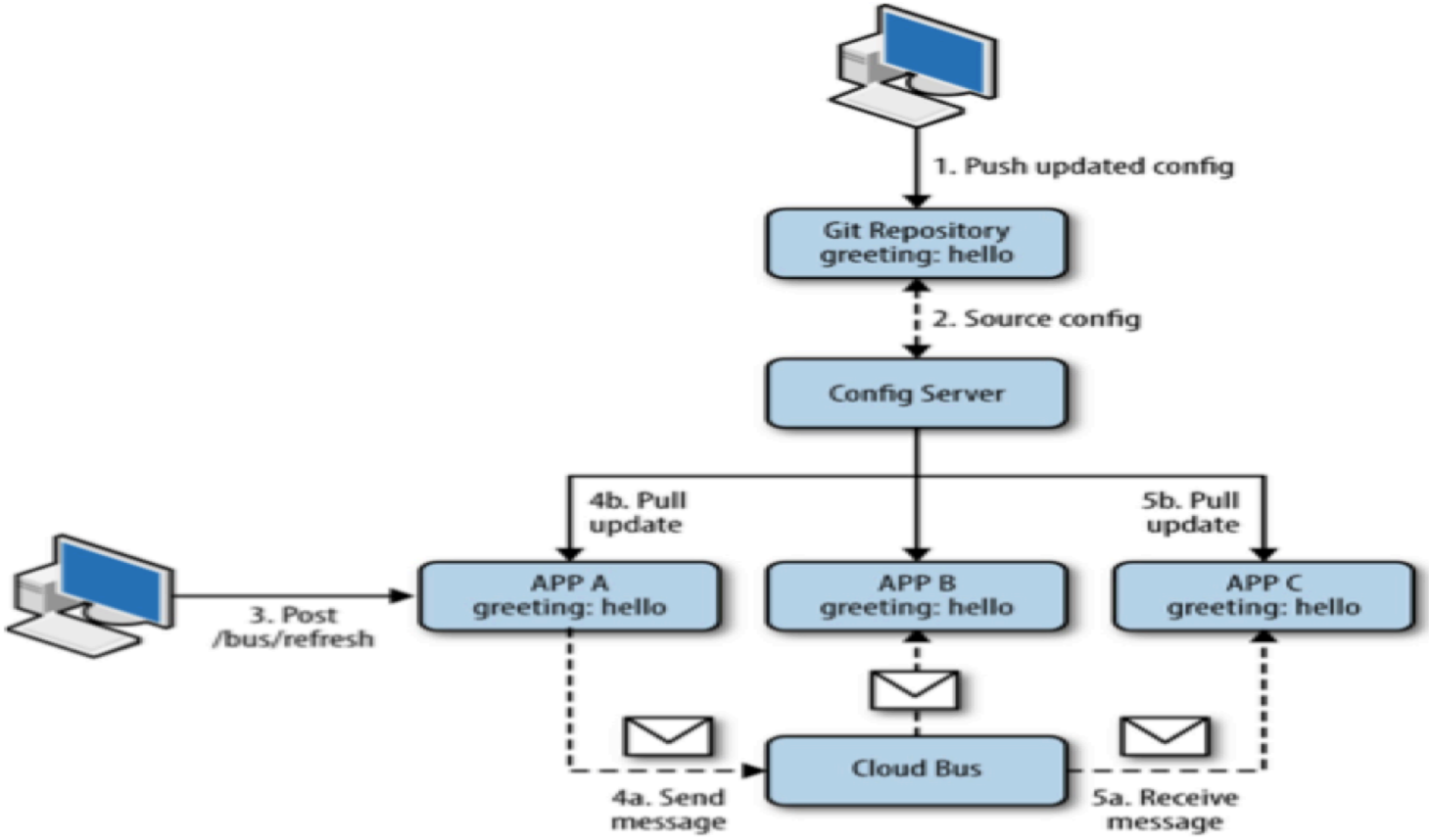
Example

```
@SpringBootApplication
@RestController
@RefreshScope // <-- Add RefreshScope annotation
public class ClientApplication {

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

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

    @RequestMapping("/greeting")
    String greeting() {
        return String.format("%s World", greeting);
    }

}
```

Spring Cloud Bus

- A solution to the problem of having to refresh multiple instances of multiple applications, without having to send each application instance a /refresh
- Leverages a message bus, supports AMQP protocol (e.g. RabbitMQ), and Kafka
- Exposes its own /bus/refresh endpoint, where receiving application instance broadcasts refresh message to all other instances

Config Server and Spring Cloud Bus



Benefits of Config Server

- **One place** to review and modify configuration)
- Separation of application development lifecycle from **configuration lifecycle**
- Ability to re-configure aspects of a running application **without downtime** (e.g. log level, feature toggles)
- Supports **encryption** of sensitive configuration properties using a number of mechanisms (symmetric encryption, asymmetric key pair)
- Choice of **git backend** provides complete configuration history, auditability (who made what configuration change, and when)

Recap

- Service Discovery with Netflix Eureka
  - Enable your apps to dynamically locate each other
- Circuit Breakers with Netflix Hystrix
  - Protect your apps against cascading effects of failure and latency
- Spring Cloud Config
  - Externalize and centralize configuration, providing it to your apps as a service

Spring Cloud Services, Connectors, Integrations


Circuit Breaker


Config Server


API Gateway


RabbitMQ


Service Registry


MySQL

Pivotal Cloud Cache


Pivotal GemFire

Pivotal Push Notifications


Redis


Riak


Session State Caching

Pivotal

SESSION SEVEN

Questions?

Pivotal

Learn more!

http://pivotal.io/microservices

https://www.microsoft.com/net/learn/architecture

http://spring.io/

http://start.spring.io/

http://steeltoe.io

https://github.com/steeltoeoss