Pivotal

Cloud Native Applications
- Developing Cloud Native Apps

Derrick Chua
Senior Platform Architect
tchua@pivotal.io
April 2019

Application Development Challenges

The need to go faster!

- Agile teams
- Pressure to implement new features quickly in order to respond to competitive pressures
- Decouple from traditional app servers

Industry Trends

- Microservices
- Cloud Native
- Containerization

spring

by Pivotal®

What is Spring Boot?

Spring Boot is a quickstart application context and container launcher for the Spring ecosystem

In essence, it gives you quick access to the Spring Framework by automatically creating an **ApplicationContext** for you

Without Spring Boot, configuring an ApplicationContext takes time and requires prior knowledge of how Spring works

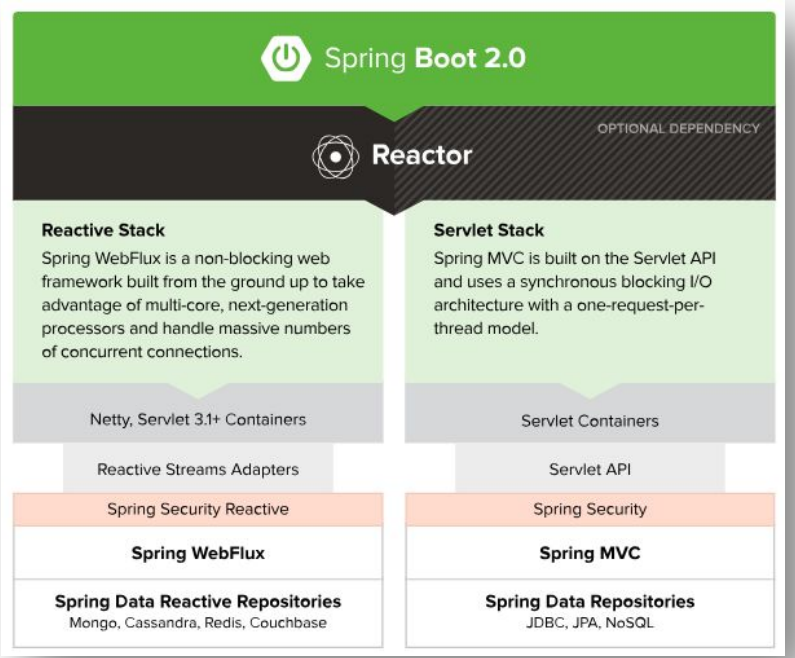
Spring Boot is “opinionated” in that it also sets up a whole bunch of defaults that the majority of developers would find helpful (like logging for example)

Spring Boot is also...

- **Fast, with a gentle learning-curve**
- **Easy to Package and Distribute**
- **Production Ready**

Pivotal

Spring Boot®



How do I get started?

The quickest way to get started with Spring Boot is to use the **Spring Initializr** at **start.spring.io**

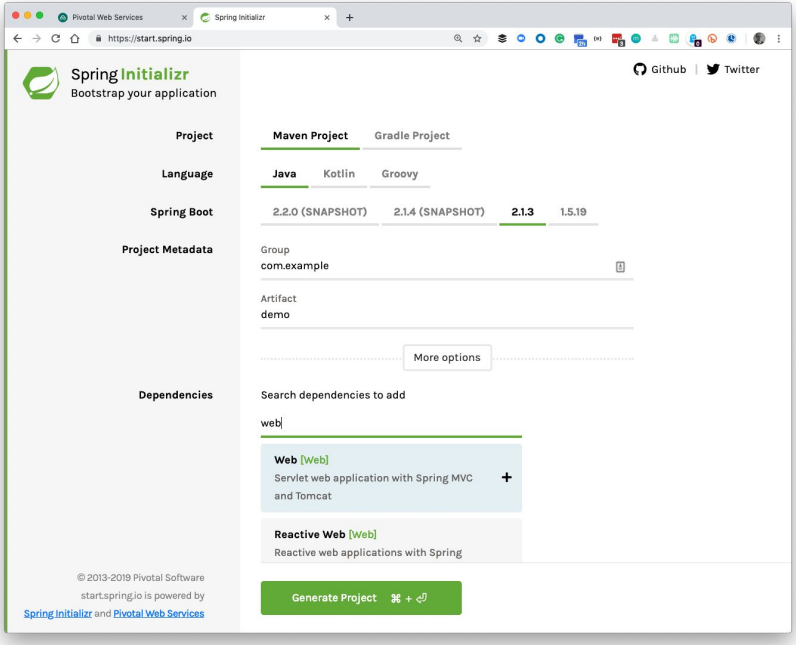
Here you can choose your preferred *build system*, your preferred *language*, a *Spring Boot version,* and choose the "**starter**" dependencies that you'd like to add to your project

Popular "starter" dependencies include...

- **Web (Tomcat, Spring MVC, REST)**
- **Reactive Web (Netty, Spring WebFlux)**
- **DevTools (rapid reload for iterative development)**
- **Thymeleaf (web template engine)**
- **JPA (Spring Data, ORM and Hibernate)**
- **Security (Spring Security)**
- **Actuator (Metrics, Admin, Insights)**

Whatever you choose, the resulting Zip file will contain a fully configured project that's ***ready to go***!

start.spring.io



You can use Spring Initializr without leaving your favourite IDE!
Spring Tool Suite, NetBeans, VSCode and IntelliJ IDEA Ultimate all feature Initializr wizards for starting new projects

What are Spring Boot starters?

Spring Boot "**starters**" are dependency packages that work in harmony with the automatic configuration feature in Spring Boot to add additional features to your project

When a compatible library is detected, Spring Boot will set about automatically configuring the components in that starter into the application's Spring application context

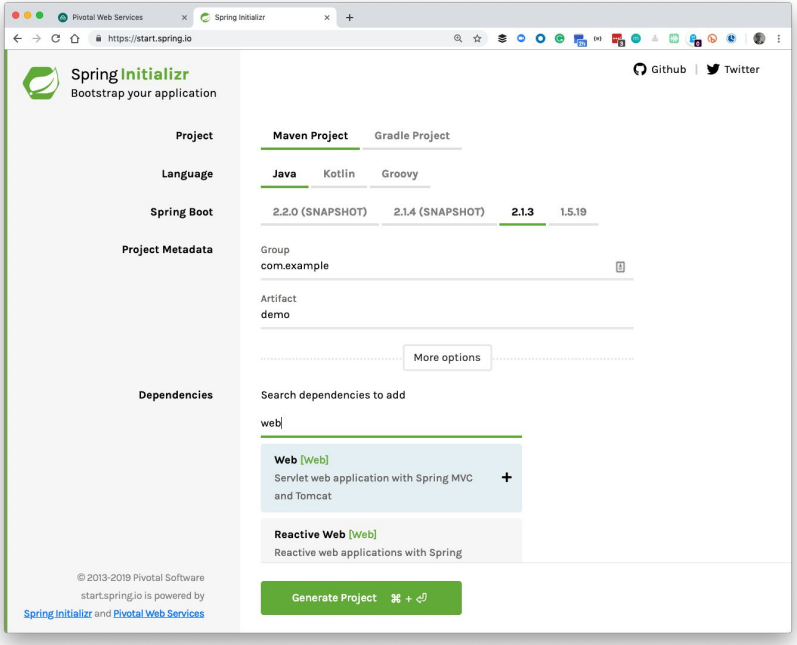
For example, when the "Actuator" starter is detected, a set of REST endpoints are added that give you new insights into your application runtime (such as `/actuator/info`)

Similarly, when JPA and a Database Driver such as H2 are detected, Spring Boot will automatically configure an in-memory H2 database, a JDBC connection pool and an Entity Manager, all using intelligent defaults

By automatically configuring itself, Spring Boot is relieving the developer from the toil of bootstrapping the application

Pivotal

start.spring.io



To find out just how easy it is to get started with Spring Boot, check out the interactive demos at the end of this deck...

How do I run Spring Boot apps?

Spring Initializr will add a “Main” class to the project that contains the **`@SpringBootApplication`** annotation and some simple boilerplate code, as shown opposite

The Initializr will also add a **`parent`** project and a spring-boot **`plugin`** dependency to the project’s build file. This allows the build system to *build*, *package* and *run* the application for you with just one simple command…

```
$> ./mvnw spring-boot:run
```

In the case of applications using the “Web” starter dependency, Spring Boot will run the application within an embedded Tomcat server (packaged in a ***Jar*** file also known as *Fat Jar*)

Bootstrapping a Java project and configuring all these dependencies used to take *hours*. Now, with Spring Boot, it’s done in minutes, helping developers everywhere to remain productive

MyApplication.java

```java
@SpringBootApplication
public class MyApplication {

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

pom.xml

```xml
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.2.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
…
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

The illustration here is for a Java based Spring Boot application using the Maven build system, however, Kotlin and Gradle are also supported

How does Spring Boot work?

The **`@SpringBootApplication`** annotation you saw is in part a collection of other annotations, one of which is **`@EnableAutoConfiguration`** which looks in the classpath for a list of auto-configurations to apply and runs them

The classes found are applied based on the rules set within them

In the **`H2ConsoleAutoConfiguration`** example opposite, these conditions include "*am I a web application?*", and "*has my property spring.h2.console been enabled*"?

This automatic configuration behaviour can easily be "deactivated" if required using a number of techniques, for example...

In the **`@SpringBootApplication`** annotation...

```
@SpringBootApplication(exclude={H2ConsoleAutoConfiguration.class})
```

In a properties file (such as **`application.properties`**)

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.h2.H2ConsoleAutoConfiguration
```

H2ConsoleAutoConfiguration.java

```
@Configuration
@ConditionalOnWebApplication(type = Type.SERVLET)
@ConditionalOnClass(WebServlet.class)
@ConditionalOnProperty(prefix = "spring.h2.console", name = "enabled",
havingValue = "true", matchIfMissing = false)
@EnableConfigurationProperties(H2ConsoleProperties.class)
public class H2ConsoleAutoConfiguration {

        private final H2ConsoleProperties properties;

        public H2ConsoleAutoConfiguration(H2ConsoleProperties
        properties) {this.properties = properties;}

        @Bean
        public ServletRegistrationBean<WebServlet> h2Console() {

         String path = this.properties.getPath();
         String urlMapping = path + (path.endsWith("/") ? "*" : "/*");

        ServletRegistrationBean<WebServlet> registration = new
        ServletRegistrationBean<>(new WebServlet(), urlMapping);

        H2ConsoleProperties.Settings settings =
        this.properties.getSettings();

        //Omitted code

         return registration;
        }
}
```

Why is Spring Boot so popular?

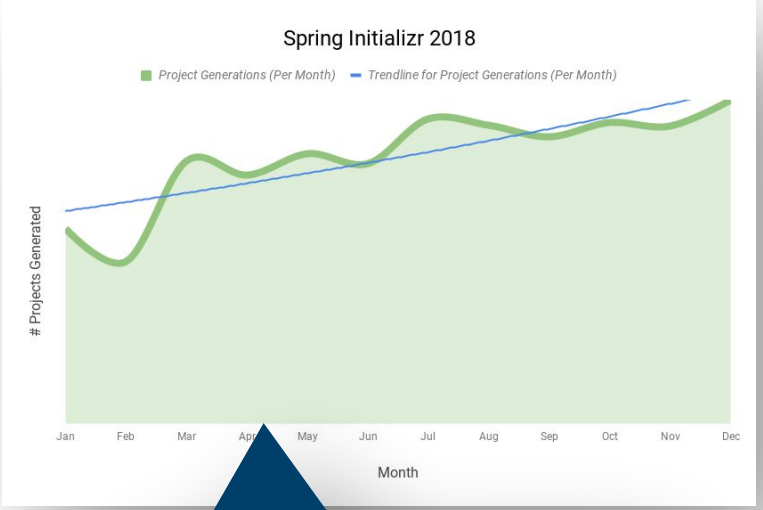
Many surveys indicate that **Spring Boot** is used by approximately 4 in 10 developers; no other platform comes close

Spring Boot is great for writing 12-factor applications, and works really well in the cloud, especially when used in conjunction with Spring Cloud's starters

Developers 💚Spring Boot because it's **easy to start** and **easy to use**, but gives them full control when they want it and flexibility when they need it

Spring Boot "just works", with **hundreds** of curated and tested dependencies right out of the box - greatly reducing the guesswork, wiring and troubleshooting required

Executable **JAR packaging** means that applications can truly "run-anywhere" there's a compatible JVM



In 2018 Spring Initializr generated over 10 million Spring projects!

For more on the popularity and adoption of Spring, including endorsements, see Spring Boot Adoption

Pivotal

What about the Cloud?

Spring Cloud extends Spring Boot's "convention over configuration" approach into the **cloud**

Spring Cloud provides a set of tools or "**common patterns**" that reduce complexity and allow developers to quickly assemble distributed systems

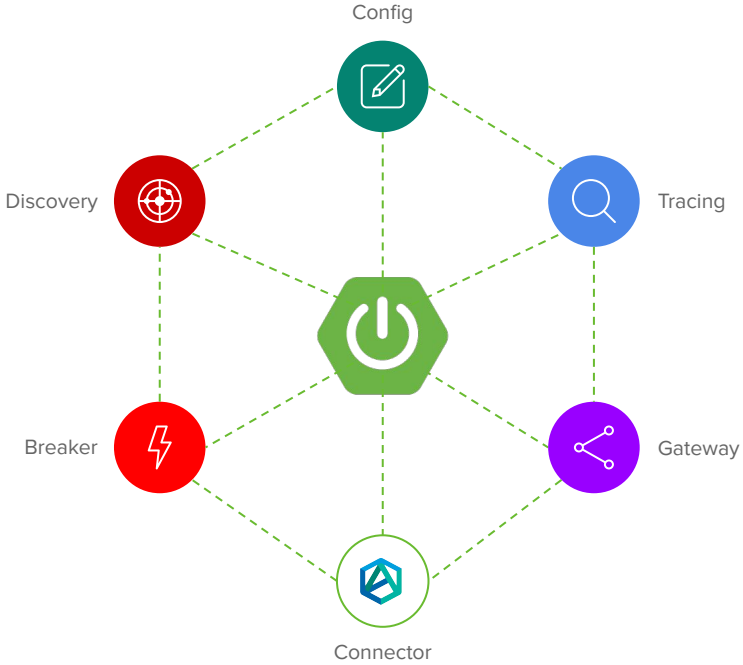
These common patterns will be familiar to many and include...

- ★ Service Discovery
- ★ Externalised Configuration
- ★ Circuit Breaker
- ★ API Gateway
- ★ Client Side Routing & Load Balancing
- ★ Distributed Tracing
- ★ Admin Tasks, API Contracts, Streams, Connectors
- ★ ....and much more

Spring Cloud is the logical next step for anyone scaling out microservices or distributed systems built with Spring Boot

Pivotal

Spring Cloud®



What about Production?

Pivotal Application Service (PAS) is the best place to run Spring Boot applications; just look at all the built-in features...

- ★ **Java Buildpack** with autoreconfirguration, a choice of JVM variants, a hardened Ubuntu OS and OCI containerisation
- ★ **PCF Container Management** scales up and down with ease
- ★ **PCF Apps Manager** is a graphical UI dashboard which shows your app's Actuators, ENV vars, Traces and more
- ★ **PCF Marketplace** allows on-demand provisioning of highly-available backing services such as Databases, Messaging etc.
- ★ **Spring Cloud Services** brings popular Spring Cloud features to the marketplace including registry and config
- ★ **PCF UAA** allows single-sign-on and consolidation of LDAP, SAML and OAuth for comprehensive application security
- ★ **PCF Routing** gives your customers instant and secure access to your application with no firewalls, dns or tickets
- ★ **PCF Networking & Isolation Segments** allows you to easily configure application networking permissions

Pivotal

Pivotal Application Service

DEVELOPERS

PAS is Built for High-Velocity Dev Teams

Focus on your code and let PAS do the rest. When you deploy new code with a simple cf push, PAS gets it into production. Dependency management, load balancing, container orchestration, logging, and auditing are done for you!

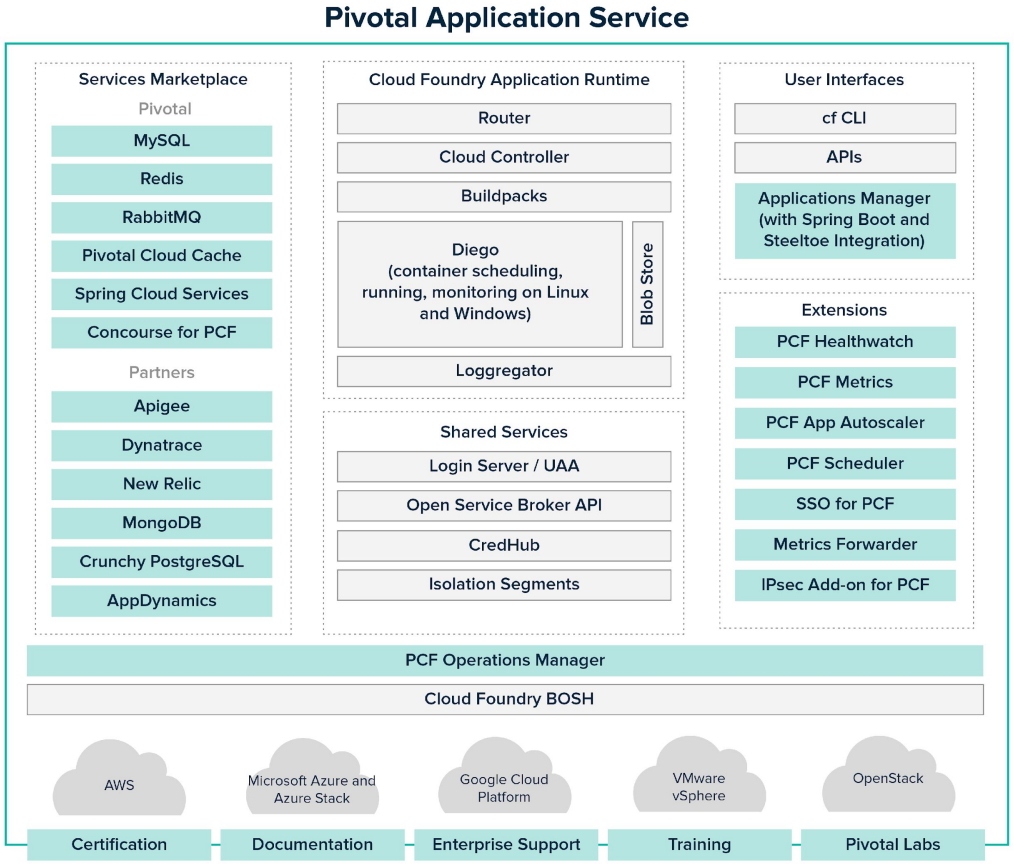
ENTERPRISE

PAS Helps You Defend Against Modern Security Threats

Reduce risk. PAS includes cloud-native security capabilities to mitigate risks posed by malware, advanced persistent threats, and leaked credentials.

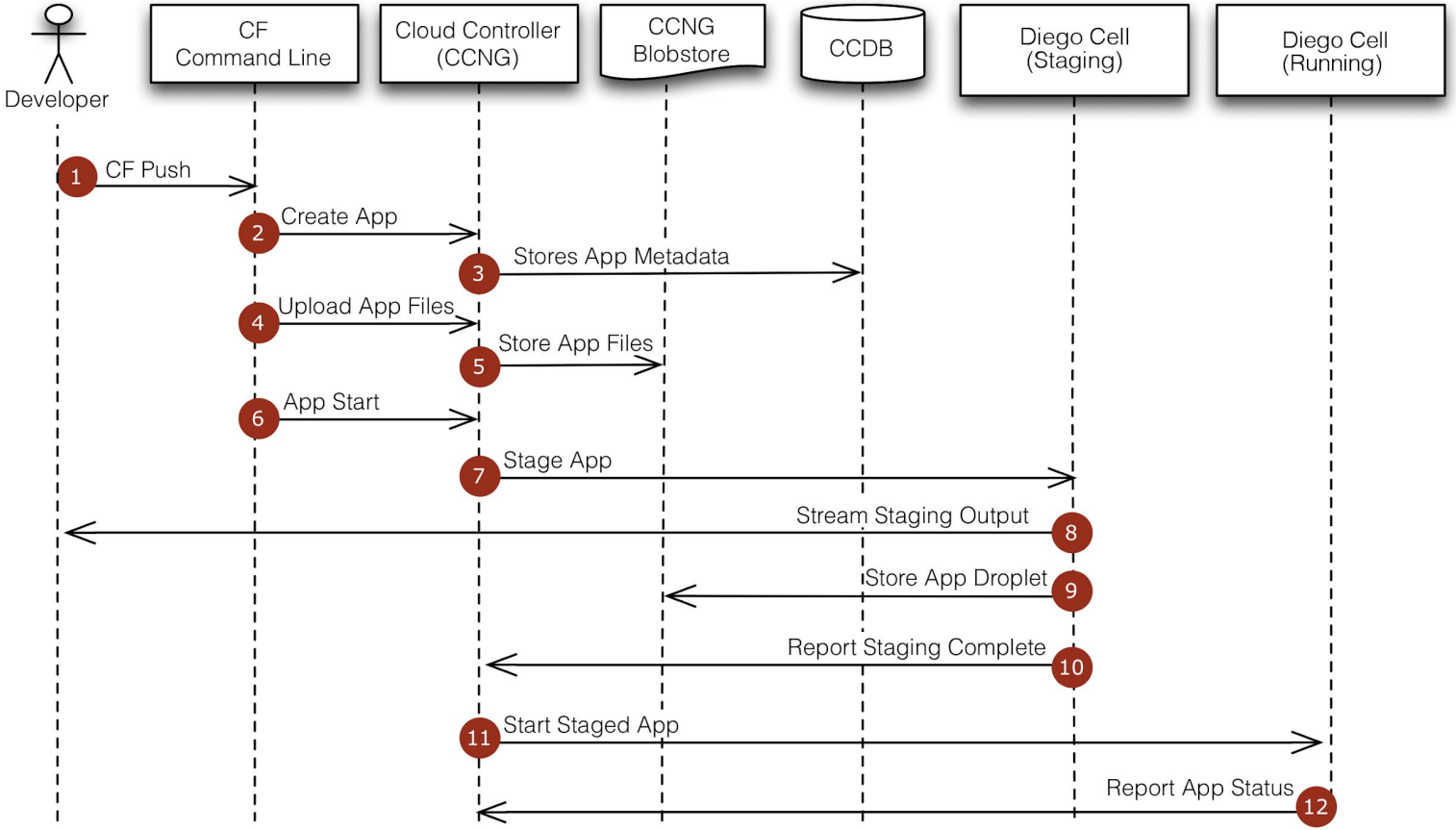
Pivotal

PAS overview



Pivotal

cf push


Developer
CF Command Line
Cloud Controller (CCNG)
CCNG Blobstore
CCDB
Diego Cell (Staging)
Diego Cell (Running)
1 CF Push
2 Create App
3 Stores App Metadata
4 Upload App Files
5 Store App Files
6 App Start
7 Stage App
8 Stream Staging Output
9 Store App Droplet
10 Report Staging Complete
11 Start Staged App
12 Report App Status

Buildpacks

- Provide framework and runtime support for apps.
- Typically examine your apps to determine what dependencies to download and how to configure the apps to communicate with bound services.
- When you push an app, Cloud Foundry automatically detects an appropriate buildpack for it. This buildpack is used to compile or prepare your app for launch.
- E.g. - https://github.com/cloudfoundry/java-buildpack

Transforming How The World Builds Software