

The Pivotal logo is displayed in white text against a teal background. The background image shows a blurred office scene with people working at computers.

Pivotal®

微服务在PCF上的最佳实践

刘凡

Pivotal 云计算资深架构师

主要内容

- 微服务@Pivotal
- Spring Boot—构建微服务
- Spring Cloud—治理微服务
- Spring Data Flow—连接微服务数据
- 微服务设计原则和方法
- Q&A



Microservices @Pivotal



Pivotal **Labs**

方法论和实践指导



Pivotal
Cloud Foundry®

微服务云原生平台



spring
by Pivotal™

微服务开发框架和工具

Pivotal的Spring：云原生Java的标准



Spring Boot

构建微服务



Spring Cloud

治理微服务



Spring Cloud Data Flow

连接微服务数据

代码清晰 | 降低复杂性 | 减少技术负债 | 专注于业务逻辑 | 测试覆盖面更广 | 代码完成速度更快

云原生应用十二要素

▶ I. 基准代码

一份基准代码，多份部署

VersionControl: git+branch, 不能用多个repo。但多个微服务可以多个repo。

▶ II. 依赖

显式声明依赖关系

pom, gradle, gemfile, 通过仓库要管理二进制文件。**CI/CD**

▶ III. 配置

在环境中存储配置

SpringCloud: 环境变量的使用, 并且把环境的配置放到Version Control中 (如git, svn)

▶ IV. 后端服务

把后端服务当作附加资源

PCF: service broker, 给应用绑定后端资源和服务

▶ V. 构建, 发布, 运行

严格分离构建和运行

PCF: 通过buildpack进行构建, 通过容器运行

▶ VI. 进程

以一个或多个无状态进程运行应用

Design: 无状态设计原则

▶ VII. 端口绑定

通过端口绑定提供服务

PCF: domain+router绑定服务接口

▶ VIII. 并发

通过进程模型进行扩展

PCF+SpringCloud: 横向/纵向扩展

▶ IX. 易处理

快速启动和优雅终止可最大化健壮性

SpringBoot: 单体应用, 可快速启停, 幂等设计

▶ X. 开发环境与线上环境等价

尽可能的保持开发, 预发布, 线上环境相同

SpringCloud: 通过CI/CD持续发布版本, 不允许手动更改配置和手动修改源码的二进制文件, 源码一致, 环境配置放到Spring Config Server

▶ XI. 日志

把日志当作事件流

PCF+SpringBoot: 日志管理器, 外置日志分析工具

▶ XII. 管理进程

后台管理任务当作一次性进程运行

PCF: run tasks

12 + 3 = 十五要素

▶ API 设计先行(API First)

- 在组织内部建立统一的约定来规范和管理服务之间的调用接口
- 在开发服务之前，首先设计服务的API

SpringBoot: 通过swagger设计REST API，并通过Spring Contract test来测试契约

▶ 对系统和应用的远程监控(Telemetry)

- 应用性能监控
- 特定业务的监控
- 系统健康状况及日志收集

PCF: pcf metrics, 日志收集管理

▶ 用户校验和授权(Authentication and Authorization)

- 安全永远都应该被优先考虑
- 所有的云原生应用的endpoint都应该被RBAC的模式保护

PCF: UAA, 租户用户管理

*RBAC: Role-based Access Control

Spring Boot



Spring Boot

自动配置和自动装载机制

- 自动配置Spring
- 使用Starter注入类，简化Maven配置

快速构建可运行的应用

- 创建独立的Spring程序
- 内嵌Tomcat/Jetty/Undertow，无需部署WAR文件
- 绝对没有代码生成和对XML无配置要求

提供生产就绪功能

- 应用运行度量指标，健康检查和外置配置文件

SPRING INITIALIZR @ start.spring.io

1. bash

Last login: Sat May 12 13:43:45 on ttys005
localhost:~ fliu\$ curl start.spring.io



:: Spring Initializr :: https://start.s

This service generates quickstart project
Possible customizations include a project
build system or build structure. See bel

The services uses a HAL based hypermedia
to interact with. If you access this ro
as media type the response will contain

Rel	Description
gradle-build	Generate a Gradle bu
gradle-project	Generate a Gradle ba
maven-build	Generate a Maven pom
maven-project *	Generate a Maven bas

The URI templates take a set of paramete
to the linked resource.

Parameter	Description
applicationName	application name
artifactId	project coordinates
baseDir	base directory to cr

SPRING INITIALIZR bootstrap your application now

Generate a

Maven Project

 with

Java

 and Spring Boot

2.0.2

Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

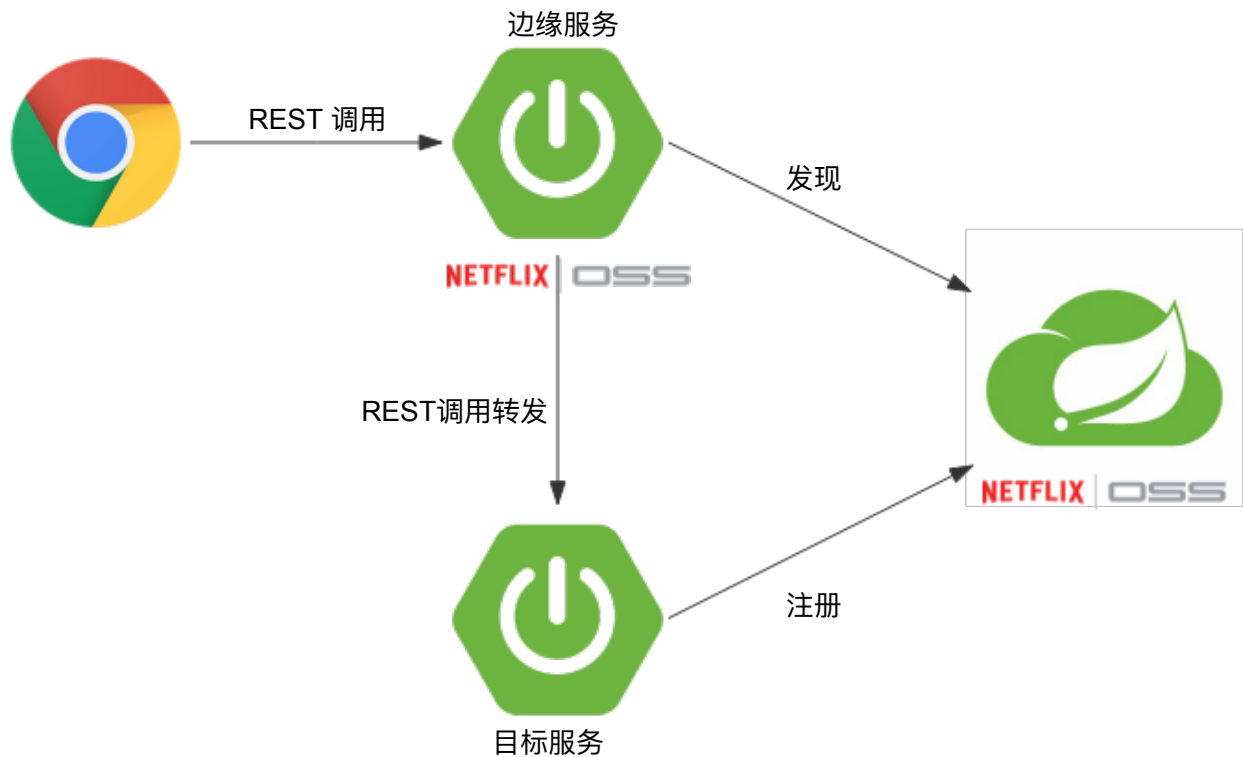
Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

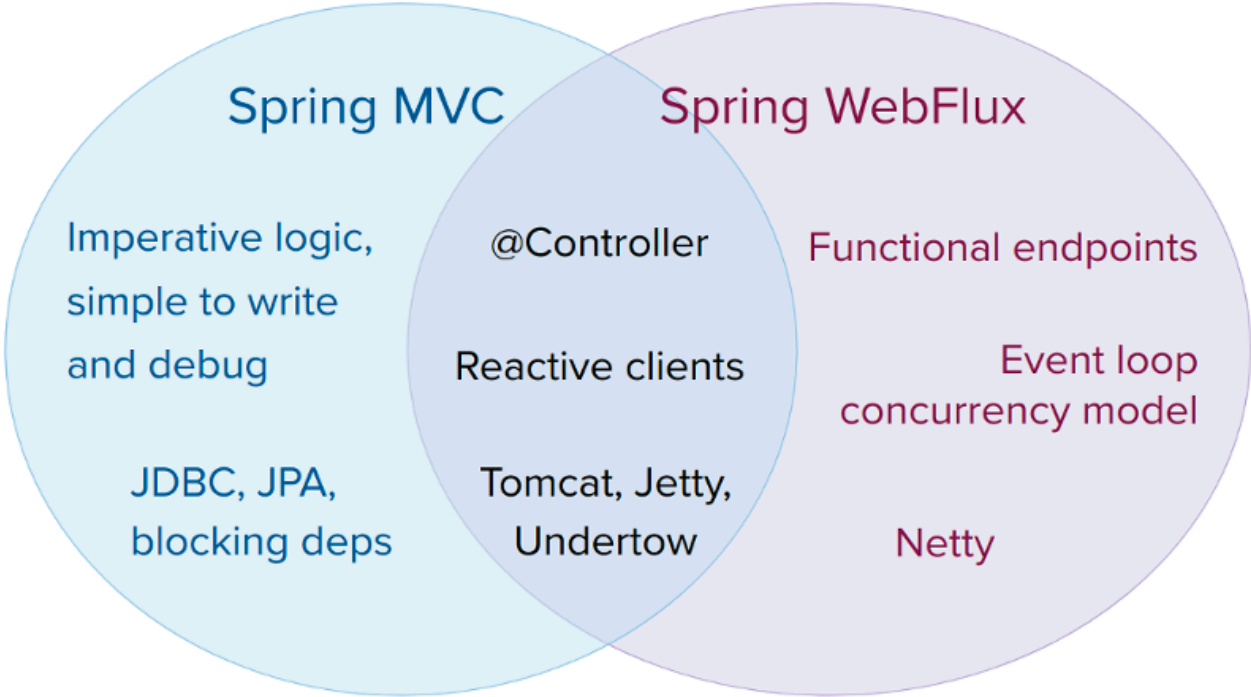
Generate Project

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

用Spring Boot构建微服务



Spring Boot 2.0



非阻塞
事件驱动
更多可扩展性而不仅是性能提升
你可以仍然使用Spring MVC

Spring WebFlux示例代码

```
@RestController
public class CitiesController {
    private CityRepository repository;

    public CitiesController(CityRepository repository) {
        this.repository = repository;
    }

    @GetMapping("/cities/{id}")
    public Mono<City> findById(@PathVariable String id) {
        return this.repository.findById(id);
    }

    @GetMapping(value = "/cities", produces = MediaType.APPLICATION_JSON_VALUE)
    public Flux<City> all() {
        return this.repository.findAll();
    }

    @PostMapping(value = "/city")
    public Mono<Void> save(@RequestBody City city) {
        return repository.save(city).then();
    }
}
```

Reactive Data

- Cassandra
- MongoDB
- Couchbase
- Redis

Spring WebFlux.fn示例代码

```
import static org.springframework.http.MediaType.APPLICATION_JSON;
import static org.springframework.web.reactive.function.server.RequestPredicates.*;
```

```
PersonRepository repository = ...
PersonHandler handler = new PersonHandler(repository);
```

```
RouterFunction<ServerResponse> route =
```

```
    route(GET("/person/{id}").and(accept(APPLICATION_JSON)), handler::getPerson)
        .andRoute(GET("/person").and(accept(APPLICATION_JSON)), handler::listPeople)
        .andRoute(POST("/person"), handler::createPerson);
```

@RequestMapping

```
public class PersonHandler {
    // ...
    public Mono<ServerResponse> listPeople(ServerRequest request) {
        // ...
    }
    public Mono<ServerResponse> createPerson(ServerRequest request) {
        // ...
    }
    public Mono<ServerResponse> getPerson(ServerRequest request) {
        // ...
    }
}
```

WebClient示例代码

```
WebClient client = WebClient.create("http://example.org");

Mono<Person> result = client.get()
    .uri("/persons/{id}", id).accept(MediaType.APPLICATION_JSON)
    .retrieve()
    .bodyToMono(Person.class);

Flux<Quote> result = client.get()
    .uri("/quotes").accept(MediaType.TEXT_EVENT_STREAM)
    .retrieve()
    .bodyToFlux(Quote.class);
```

Spring Boot + 应用运行时

实践证明，这样的组合可提高企业开发团队的速度。



Spring Boot

轻松的依赖关系管理

嵌入式应用服务器

创建可直接运行的
自成一体的应用



Pivotal Application Service

从jar生成容器

在执行cf push时立即在容器中启动应用

添加环境属性

微服务就绪 | NetflixOSS | Actuator集成 | 指标与日志记录

Spring Boot 2.0 on PCF

APP

webflux-demo



● Running

[VIEW APP](#)[Overview](#) [Services](#) [Route \(1\)](#) [Logs](#) [Tasks](#) [Settings](#)

Buildpack: N/A

Logs



```
2018-04-24T11:38:50.482+08:00 [APP/PROC/WEB/0] [OUT] [reactor-http-nio-1] 38:50 DispatcherHandler: Processing GET request for [http://webflux-demo-smart-warthog.pcfbeta.io/cities]
2018-04-24T11:38:50.482+08:00 [APP/PROC/WEB/0] [OUT] [reactor-http-nio-1] 38:50 RequestMappingHandlerMapping: Looking up handler method for path /cities
2018-04-24T11:38:50.486+08:00 [APP/PROC/WEB/0] [OUT] [reactor-http-nio-1] 38:50 RequestMappingHandlerMapping: Returning handler method [public reactor.core.publisher.Flux<com.example.demo.City> com.example.demo.CitiesController.all()]
2018-04-24T11:38:50.758+08:00 [APP/PROC/WEB/0] [OUT] [reactor-http-nio-1] 38:50 1: onSubscribe(FluxOnErrorResume.ResumeSubscriber)
2018-04-24T11:38:50.759+08:00 [APP/PROC/WEB/0] [OUT] [reactor-http-nio-1] 38:50 1: request(unbounded)
2018-04-24T11:38:50.784+08:00 [APP/PROC/WEB/0] [OUT] [Thread-14] 38:50 EmbeddedMongo: 2018-04-24T03:38:50.783+0000 I NETWORK [initandlisten] connection accepted from 127.0.0.1:58176 #4 (4 connections now open)
2018-04-24T11:38:50.818+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 connection: Opened connection [connectionId{localValue:4, serverValue:4}] to localhost:40720
2018-04-24T11:38:50.858+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@139afb16)
2018-04-24T11:38:50.858+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@71aab343)
2018-04-24T11:38:50.858+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@2032eccf)
2018-04-24T11:38:50.858+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@107be0b7)
2018-04-24T11:38:50.859+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@547dfbe5)
2018-04-24T11:38:50.859+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@70697a82)
2018-04-24T11:38:50.859+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@478aa4d4)
2018-04-24T11:38:50.859+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@8bb8aca)
2018-04-24T11:38:50.859+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@3322faeb)
2018-04-24T11:38:50.860+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@8242b67)
2018-04-24T11:38:50.860+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onNext(com.example.demo.City@1a9a92b3)
2018-04-24T11:38:50.861+08:00 [APP/PROC/WEB/0] [OUT] [nioEventLoopGroup-2-2] 38:50 1: onComplete()
2018-04-24T11:38:50.888+08:00 [RTR/0] [OUT] webflux-demo-smart-warthog.pcfbeta.io - [2018-04-24T03:38:50.480+0000] "GET /cities HTTP/1.1" 200 0 403 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36" "192.168.4.1:47274" "192.168.16.43:61000" x_forwarded_for:"10.152.254.180, 192.168.4.1" x_forwarded_proto:"https" vcap_request_id:"db7aa57e-c183-4be9-6135-3ffb6aa82190" response_time:0.407444765 app_id:"d95da8ac-09b8-437f-9c40-4b923a263f4a" app_index:"0" x_b3_traceid:"510b92697a0aed38" x_b3_spanid:"510b92697a0aed38" x_b3_parentspanid:"-"
```

Spring Cloud Services



微服务架构的基本需求

Netflix OSS

通过实际微服务应用生产运维积累了一套开发部署运维的模式，贡献到Netflix OSS的基本实现模式

- 服务注册发现
- 断路器
- 外置化配置管理

Spring Cloud OSS

和Netflix OSS合作开发并实现了这些组件来支持Spring开发者

- Eureka
- Hystrix
- Config Server

开发者可以根据自己的需求部署SC OSS应用

NETFLIX

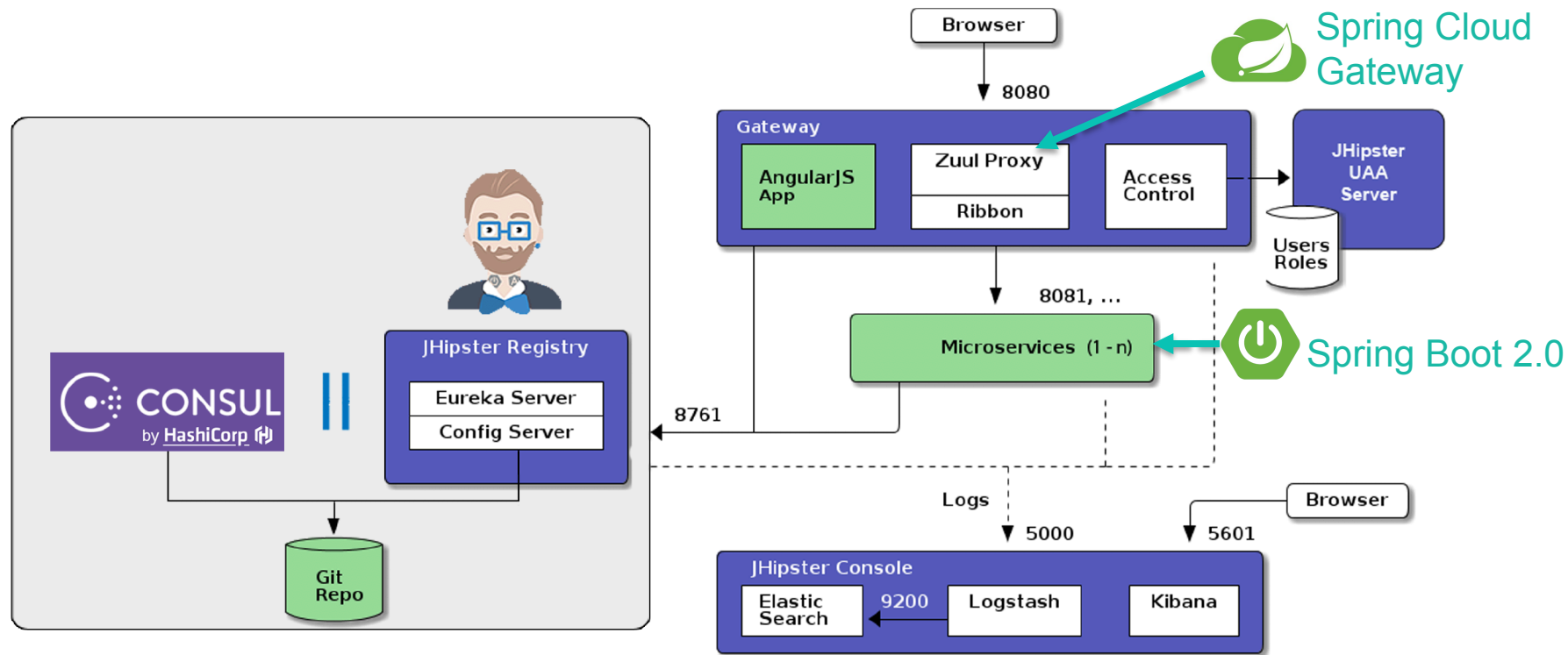
OSS



docke

Pivotal

IT大咖说



elastic



logstash



kibana

什么是Spring Cloud Services for PCF?

Spring Cloud Services(SCS) 把 Spring Cloud OSS中服务器端的组件打包成为服务并在PCF marketplace 上提供给应用去访问。它大大简化了自行实现和维护这些SC服务的工作, SCS专注于:

- 易用性
- 安全性
- 可靠性



Config Server



Circuit Breaker
Dashboard



Service Registry

易用性（服务器端）：创建Spring Cloud Server

使用Spring Cloud OSS，需要自己实现

- 创建Spring Boot App
- 添加Spring Cloud starters
- 添加@EnableXXXServer
- 添加configuration
- 添加security
- 部署...
- 更新

使用Spring Cloud Services

- `cf create-service`

易用性（客户端）：创建Spring Cloud客户端应用

使用Spring Cloud OSS，需要自己实现

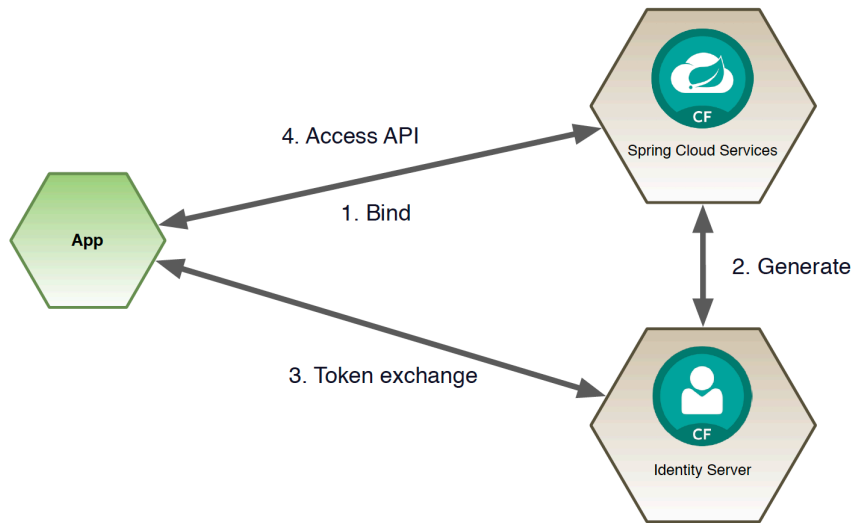
- 创建Spring Boot应用
- 添加Spring Cloud starters
- 添加configuration
 - Eureka-first 还是config server-first Bootstrap的配置
 - 环境的profiles
- 添加security

使用Spring Cloud Services

- 创建Spring Boot应用
- 添加Spring Cloud Services starters
- cf bind-service ...

安全性 – OAuth2 服务访问

- Spring Cloud Services (SCS) 服务实例是通过OAuth2来保证安全访问的, 它是通过在PCF UAA服务上的一个专门的identity zone来实现的
- 当绑定一个SCS服务实例时, 将为客户端应用生成的client ID和secret
- 当访问SCS服务时, 客户端应用提供client ID和secret给OAuth2, 来获取authorization token, 从而保证对SCS服务实例API访问的安全性



可靠性 – 通过PCF来保证

- 高可用和应用健康管理通过PCF平台来保证



SCS基于Spring OSS项目构建

Spring Boot

Spring Cloud Netflix

Spring Cloud Server

Spring Cloud Connectors

Spring Security

Spring Security OAuth2

Spring Data JPA

Spring AMQP

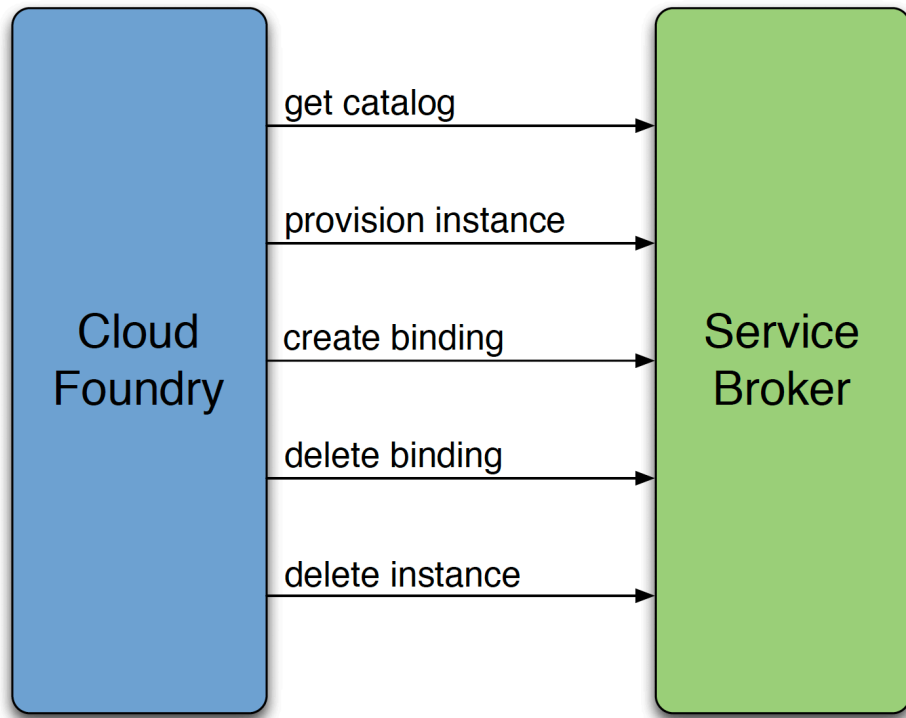
Spring Boot CF Service Broker

Cloud Foundry Java Client



Cloud Foundry Service Brokers

- CF提供的Service Broker可以为云原生应用提供资源 (resources)
- <https://docs.cloudfoundry.org/services/>
- Spring Cloud Services 提供了一个专门给应用提供spring cloud服务实例的 service broker



Spring Cloud Services Service Broker



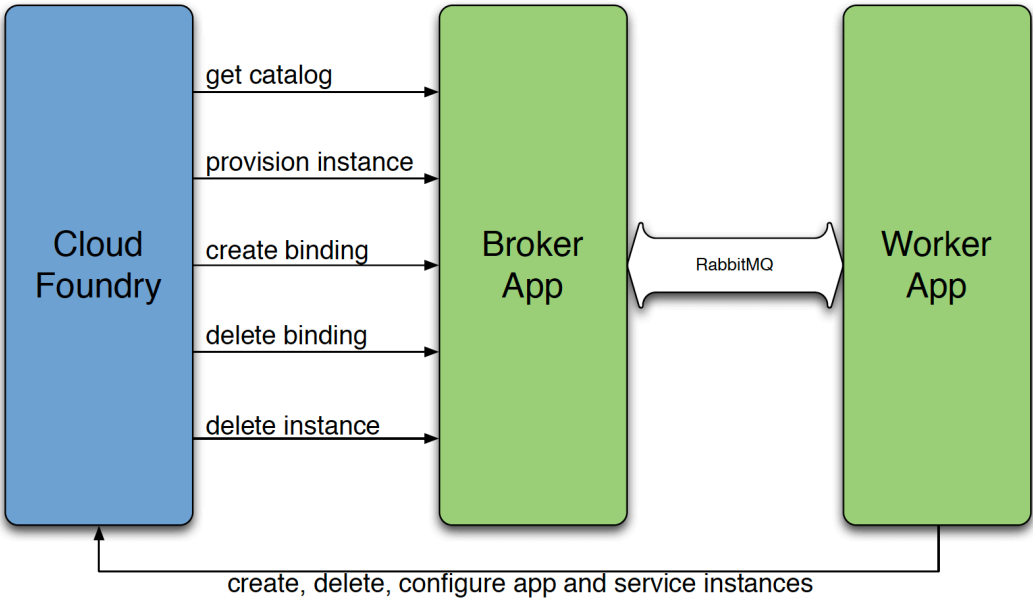
Circuit Breaker
Circuit Breaker Dashboard for Spring Cloud Applications



Config Server
Config Server for Spring Cloud Applications



Service Registry
Service Registry for Spring Cloud Applications



SCS Config Server Overview

服务

- 使客户端应用可以访问中心化配置
- 为每个服务实例部署一个Spring Cloud Config Server
- 支持Git Basic认证和SSH 私钥认证
- 允许多个repo的组合

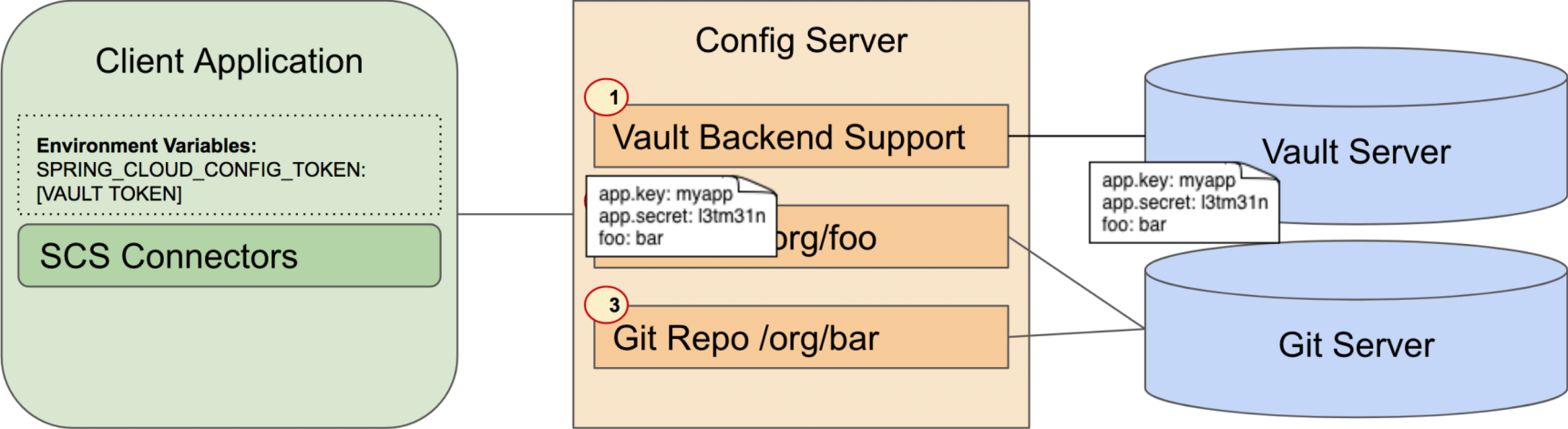
安全

- 只有绑定的应用才能访问到配置

客户端应用

- SCS starter自动配置了property source到指定的Config Server

SCS Config Server Flow



SCS Service Registry Overview

服务

- 使客户端应用可以动态发现服务实例
- 为每个服务实例部署一个Spring Cloud Netflix Eureka Server

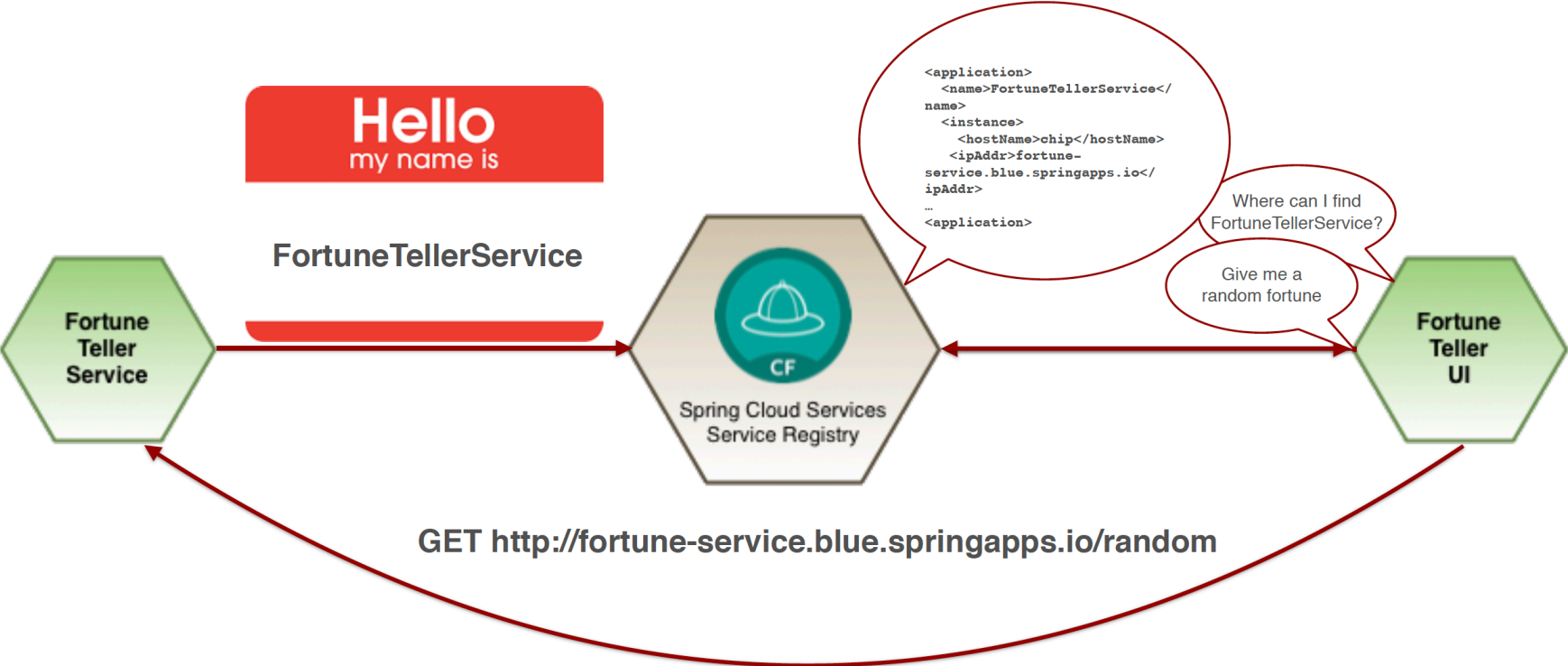
安全

- 应用只有初始创建了注册，才允许更新或删除
- 避免人为干预注册

客户端应用

- SCS starter自动配置了DiscoveryClient

SCS Service Registry Flow



SCS Circuit Breaker Dashboard Overview

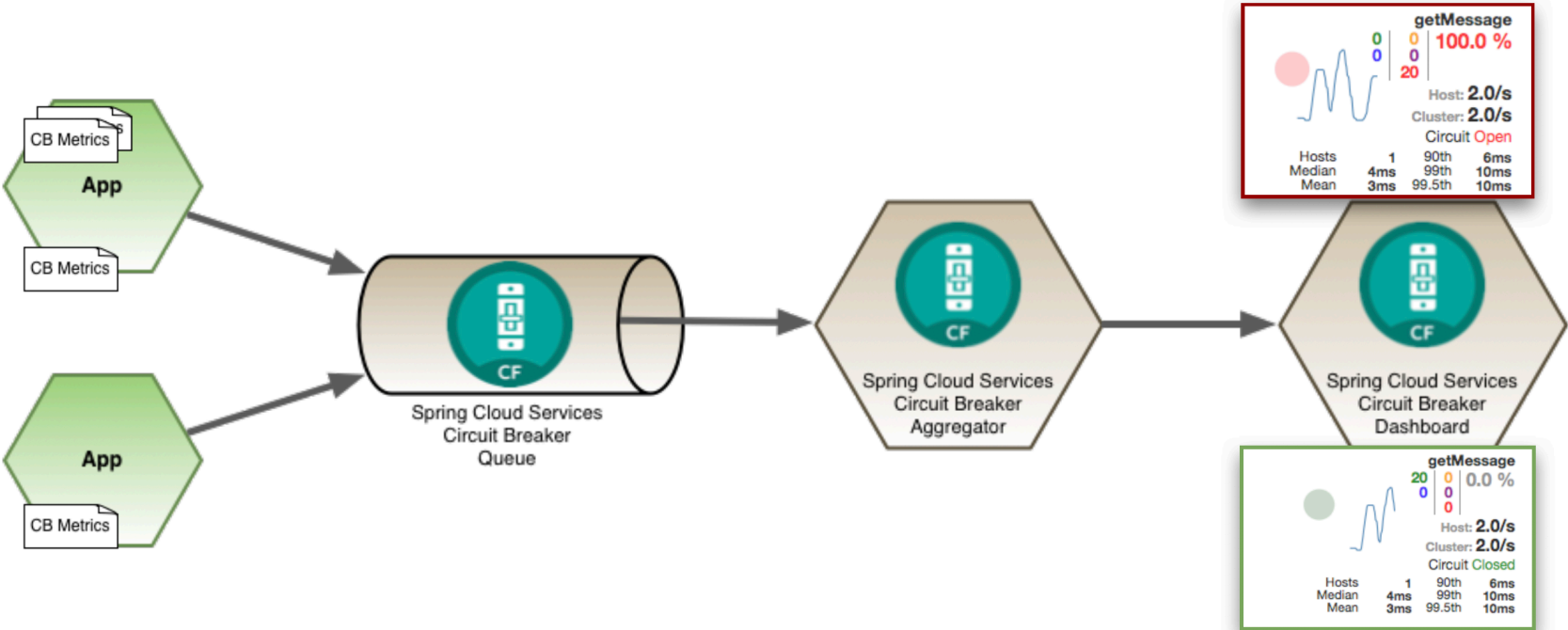
服务

- 可视化应用的断路状态
- 为每个服务实例部署
 - Spring Cloud Netflix Hystrix dashboard
 - Spring Cloud Netflix Turbine server
 - Pivotal RabbitMq for PCF service instance

客户端应用

- SCS starter自动配置了Turbine AMQP client

SCS Circuit Breaker Dashboard Flow



Spring Cloud Services Client Connectors

- 服务发现
 - 发现应用绑定的服务
 - VCAP_SERVICES
- 获取Credentials
- 通过Config Server
 - 获取应用的配置信息

```
{
  "VCAP_SERVICES": {
    "p-config-server": [
      {
        "credentials": {
          "access_token_uri": "https://p-spring-cloud-services.uaa.cf.wise.com/oauth/token",
          "client_id": "p-config-server-c4a56a3d-9507-4c2f-9cd1-f858dbf9e11c",
          "client_secret": "9aGx9K5Vx0cM",
          "uri": "https://config-51711835-4626-4823-b5a1-e5d91012f3f2.wise.com"
        },
        "label": "p-config-server",
        "name": "config-server",
        "plan": "standard",
        "tags": [
          "configuration",
          "spring-cloud"
        ]
      }
    ]
  }
}
```


Spring Cloud Services Client Dependencies

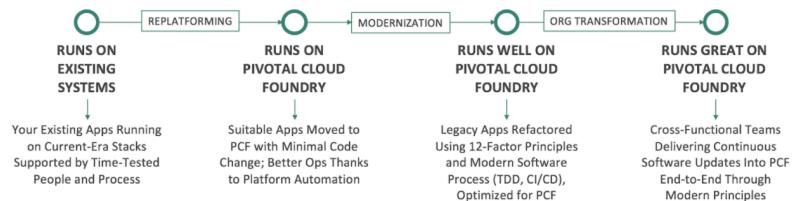
```
<dependencies>
  <dependency>
    <groupId>io.pivotal.spring.cloud</groupId>
    <artifactId>spring-cloud-services-starter-config-client</artifactId>
  </dependency>
  <dependency>
    <groupId>io.pivotal.spring.cloud</groupId>
    <artifactId>spring-cloud-services-starter-service-registry</artifactId>
  </dependency>
  <dependency>
    <groupId>io.pivotal.spring.cloud</groupId>
    <artifactId>spring-cloud-services-starter-circuit-breaker</artifactId>
  </dependency>
</dependencies>
```

微服务设计原则和方法

Pivotal推行的转型微服务的原则和步骤

基本原则

- 从小应用开始
- 从实践中学习
- 大事化小
- 自动化所有的事

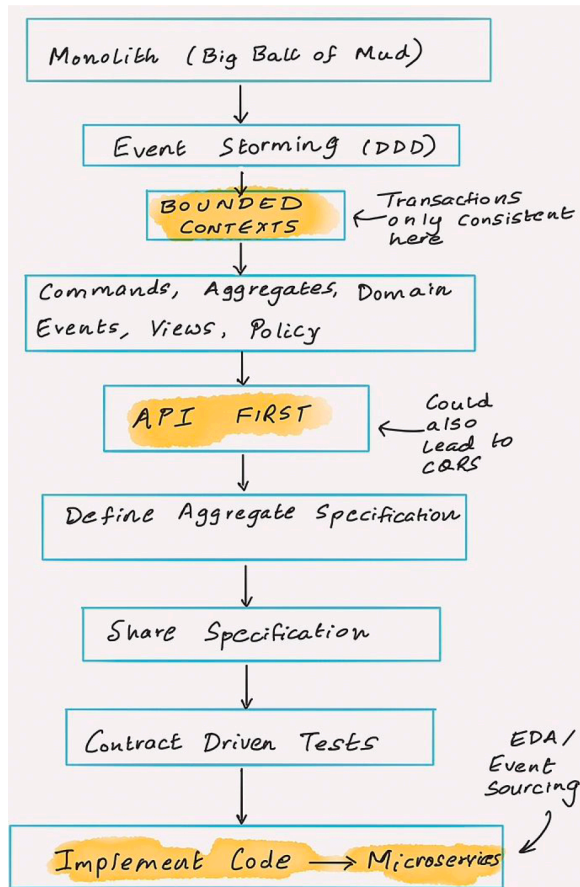


基本步骤

- 选择合适的应用
 - TIME分析
 - 近期开发的，且有技术支持
- 云化改造
 - 做好平台化改造跟踪记录
 - 自动化测试
 - 持续集成
- 创建云原生微服务应用
 - 拆分巨石应用
 - 使用DDD设计微服务

传统巨石应用重构拆分成微服务的基本方法

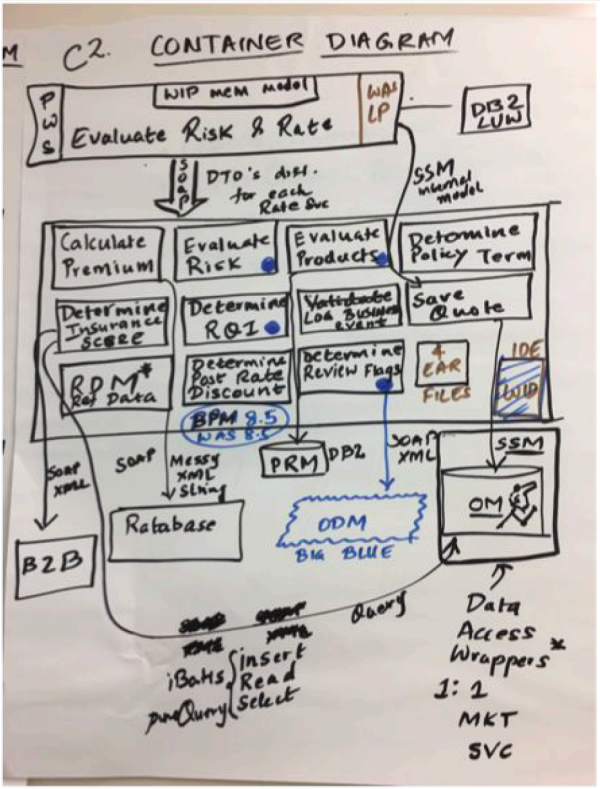
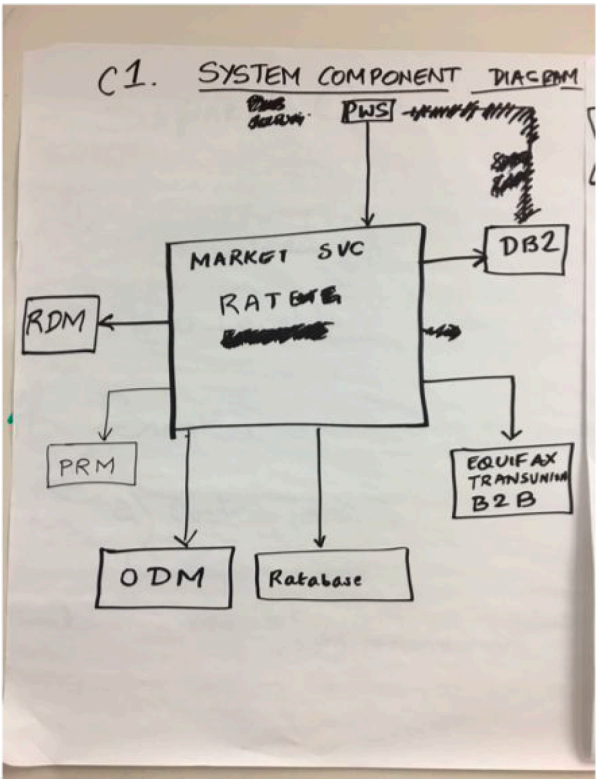
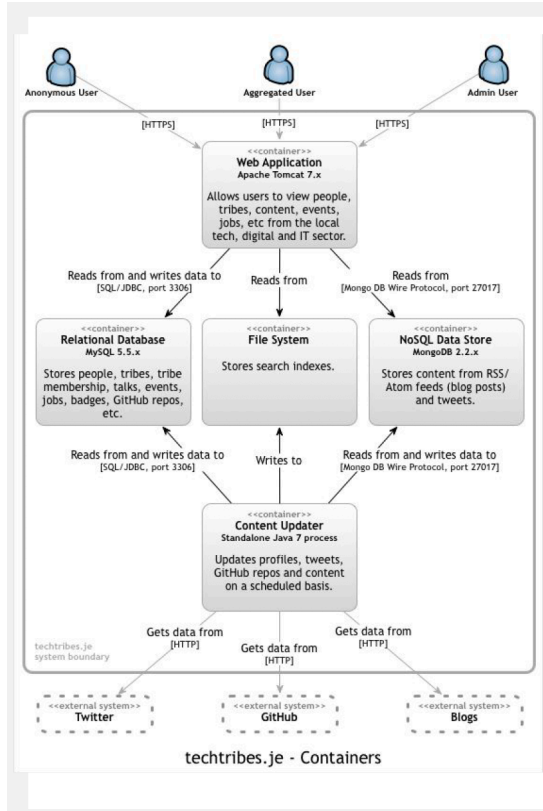
1. 定义目标和主要结果(OKRs)
2. 进行事件风暴来识别出核心Domain(DDD)和关键约束
3. 挑选核心Domain中的事件流
4. 用C4图表分析识别出产生事件流的应用组件的分层架构
5. 确定目标应用的分层架构
6. 运用SNAP分析进行评分, 估算出应用改造成符合15要素所需的工作量
7. 为User Story创建Backlog, 并指定优先级, 需符合先前定义的OKRs
8. 映射user story到MVPs



选择核心Domain



C4图表分析



SNAP分析

DATA ARCHITECTURE

ORM / Data Libraries

Integrations

ETL Jobs

Transactions

State Management

Batch Processing

File System Access

APP ARCHITECTURE

Languages / Versions

Spring Components

JEE APIs

3rd Party Libraries

Front End Techniques

Testing / Test Coverage

Security / Identity Mgmt

APP USAGE & TEAM

Number of Users

Team Location(s)

Key User Workflows

Transactions Per Second

CONFIGURATION

Deployment Type

Continuous Integration

Config / Properties Files

Build System

Infrastructure - Platform & Middleware

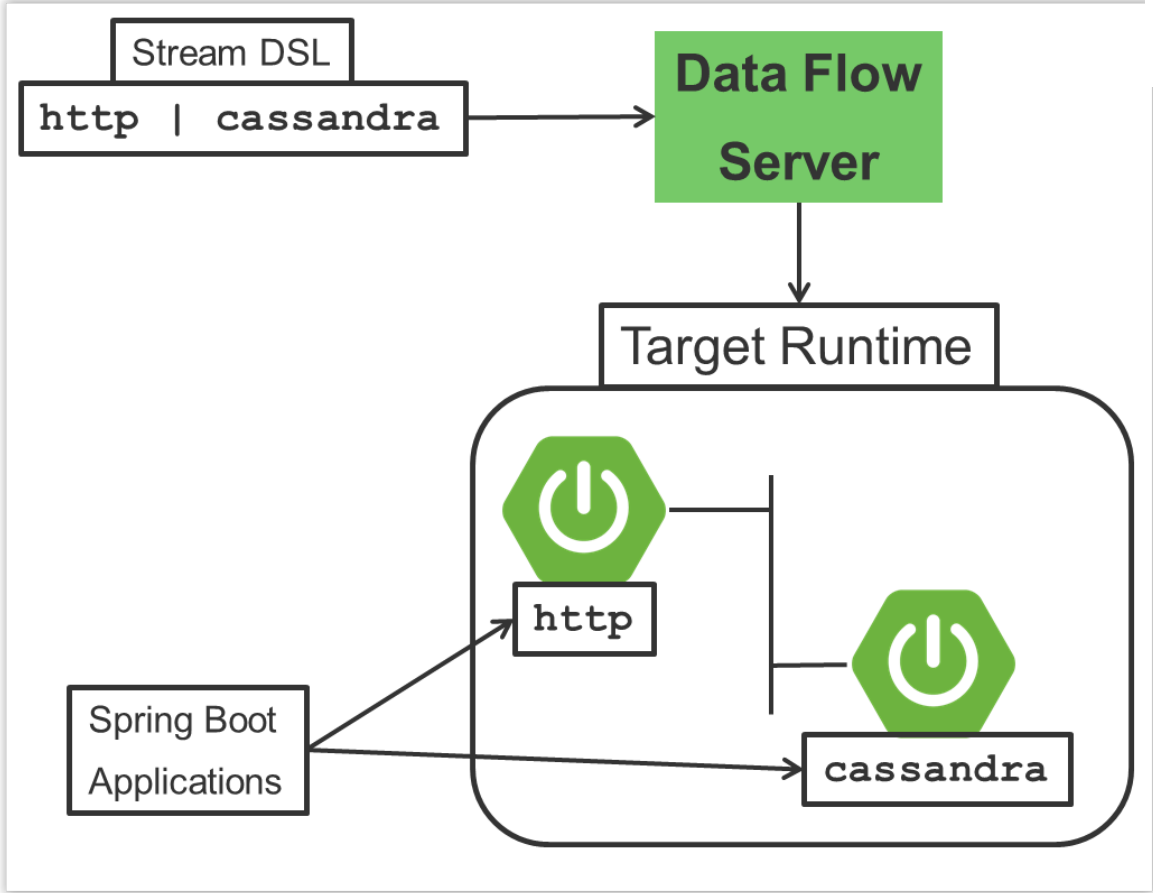
Spring Data Flow



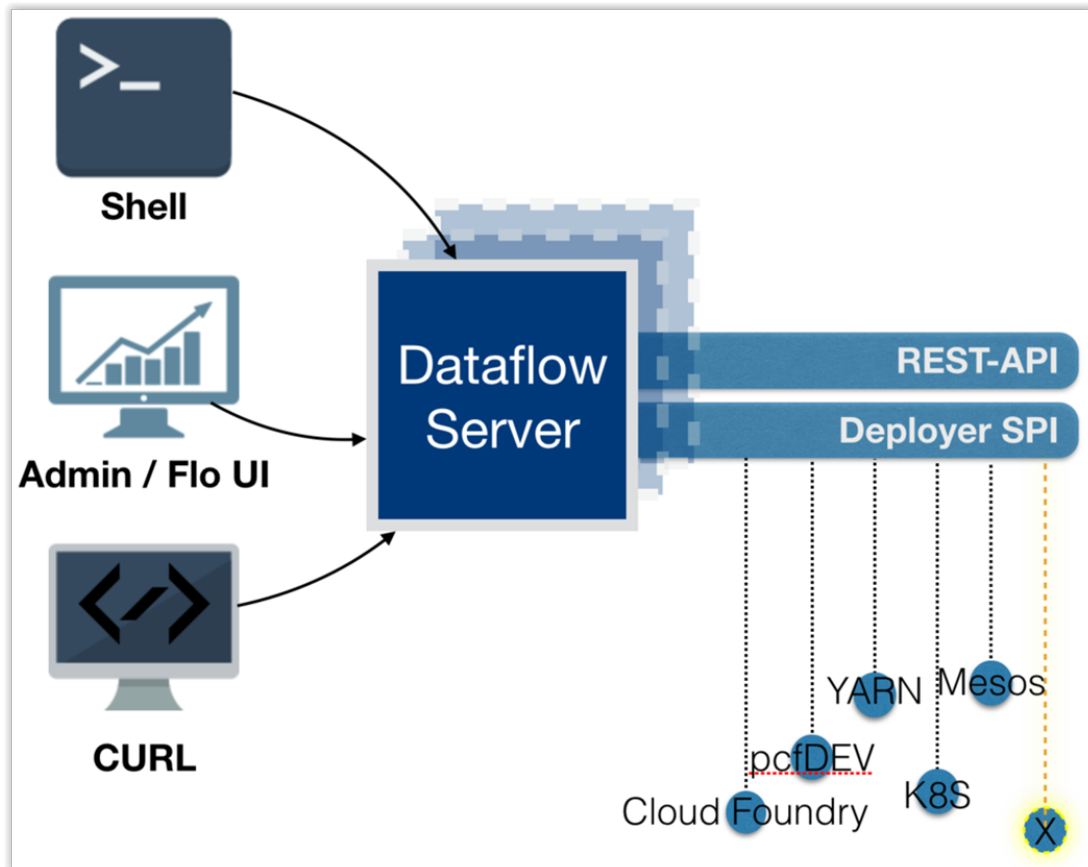
什么是Spring Data Flow

- Spring Data Flow使你能够轻松为持续的流式事件或有限的批处理任务创建数据集成处理的pipelines。
- Pipelines由Spring Boot应用组成，针对事件使用Spring Cloud Stream，针对批处理进程使用Spring Cloud Task来实现。
- Data Flow Server提供多种接口来组建和部署pipelines在不同的云平台上使用，
 - DSLs
 - UI designer
 - Stream and Task administration UI

Spring Data Flow Architecture

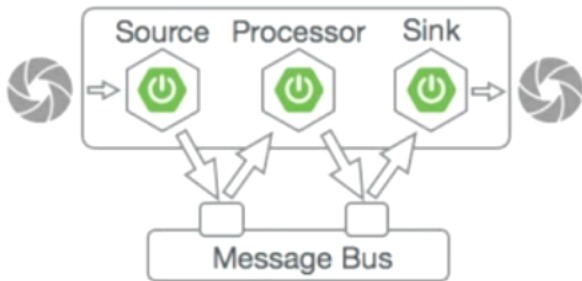


Spring Data Flow Server



微服务不仅仅是Web应用

- Spring Cloud Stream
 - 基于微服务的Event Driven
 - 通过pub/sub消息机制来解耦
 - 适用于系统集成和流处理的use case



Spring Data Flow示例

spring

Apps

Runtime

Streams

Tasks

Jobs

Analytics

About

Streams

Create a stream using text based input or the visual editor.

Definitions

Create Stream

CREATE STREAM

CLEAR

LAYOUT

AUTO LINK

GRID

```
1 tweets=twitterstream --consumer-key=aaa --consumer-secret=aaa --access-token=aaa --access-token-secret=aaa |
  log
2 :tweets.twitterstream >| field-value-counter --field-name=lang --name=language
3 :tweets.twitterstream > field-value-counter --field-name=entities.hashtags.text --name=hashtags
```

source

file

ftp

gemfire

gemfire-cq

http

jdbc

jms

tweets

twitterstream

log

field-value-c...

field-value-c...

Pivotal®

Transforming How The World Builds Software