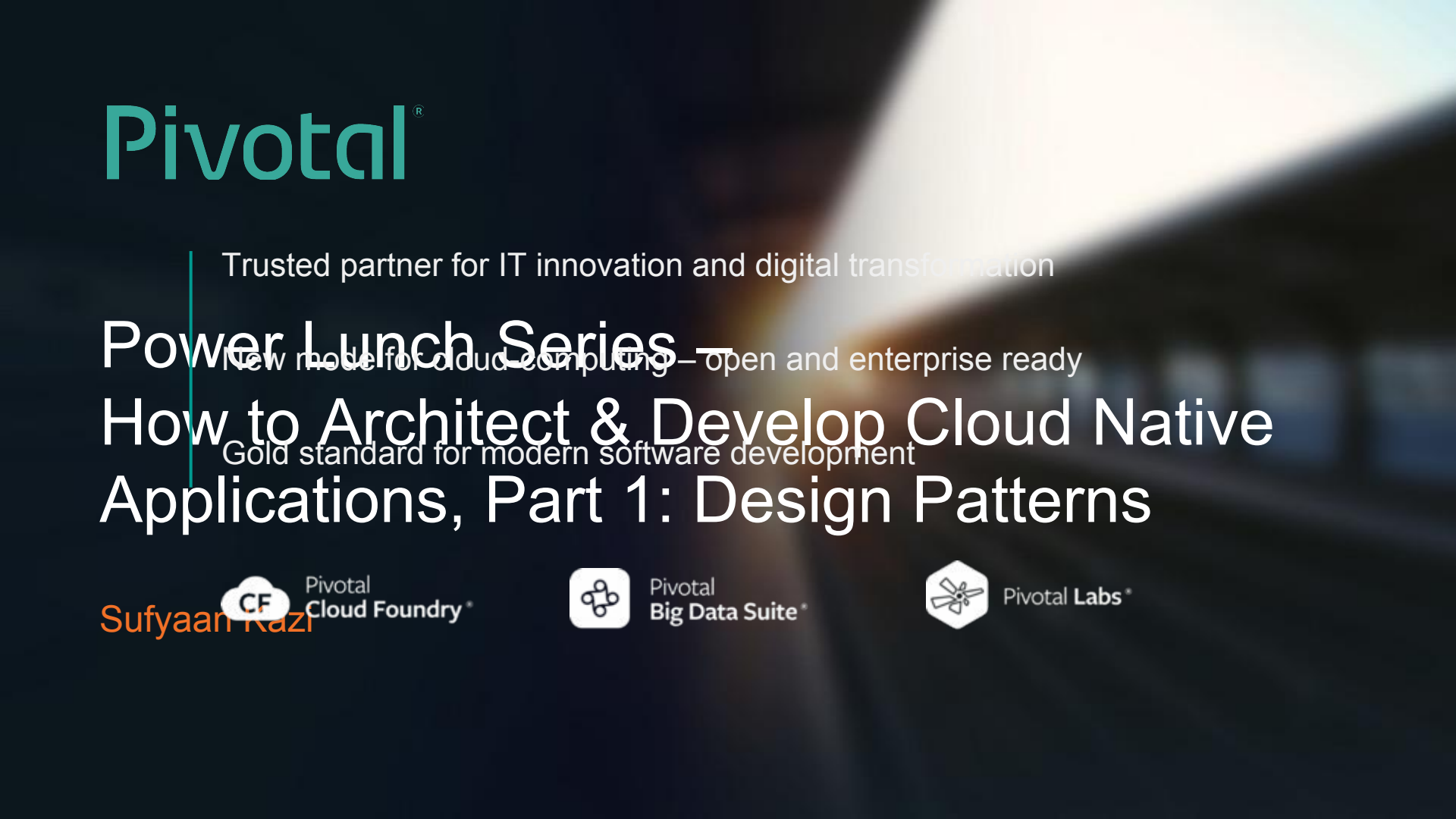
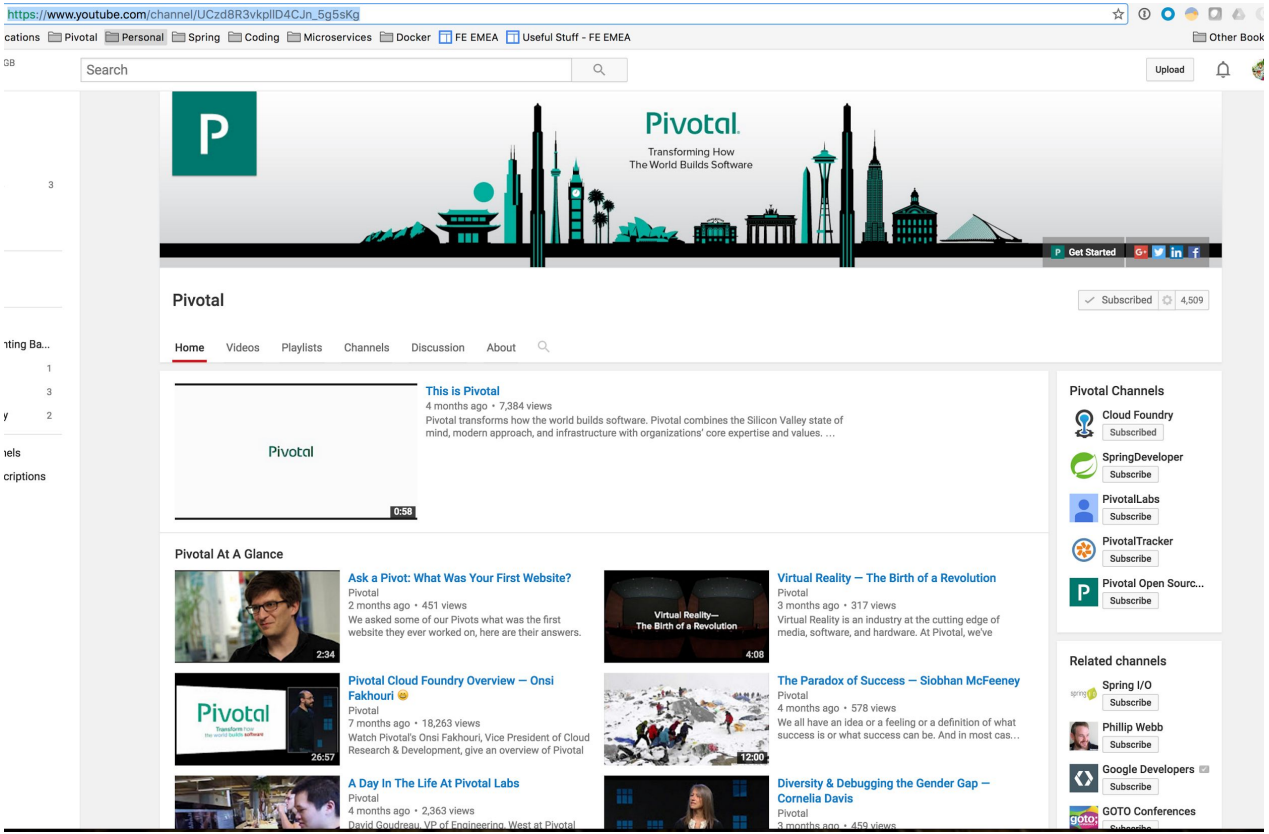

Pivotal®


Pivotal
Trusted partner for IT innovation and digital transformation
New model for cloud computing – open and enterprise ready
Gold standard for modern software development
Power Lunch Series – How to Architect & Develop Cloud Native Applications, Part 1: Design Patterns
Pivotal Cloud Foundry
Pivotal Big Data Suite
Pivotal Labs
Sufyaan Kazi

Who are Pivotal?

http://bit.ly/2cmRm9m



Who are Pivotal?

http://bit.ly/2cmS86h

1	Help Developers Do What They Love — Onsi Fakhouri by Pivotal	14:21
2	Containers Will Not Fix Your Broken Culture (and Other Hard Truths) — Bridget Kromhout by Pivotal	10:23
3	A Transformation Journey — Brad Miller, Citi by Pivotal	15:31
4	How Comcast Transformed the Product Delivery Experience — Greg Otto, Comcast by Pivotal	8:21
5	A Tale of Two Ladies: On Generating Opportunity for Women in Tech — Cornelia Davis by Pivotal	14:48
6	SpringOne Platform 2016 Keynote - Spring and the Circle of Feedback by SpringDeveloper	22:36
7	SpringOne Platform 2016 Keynote - Reactive Spring by SpringDeveloper	33:12
8	SpringOne Platform 2016 Keynote - Spring Framework 5.0 by SpringDeveloper	15:05
9	Zen and the Art of Platform — Sam Ramji, Cloud Foundry by Pivotal	10:40
10	Results Should Be More Fun — Andy Zitney, McKesson by Pivotal	12:08
11	Transforming Culture at Bloomberg — Justin Erenkrantz, Bloomberg by Pivotal	8:51

Sufyaan Kazi

skazi@pivotal.io

@sufyaan_kazi

Introduction

Pivotal

Topics in the Power Lunch Series

Why Cloud Native?

18 August 2016

How to Architect & Develop Cloud Native Applications? (Part 1 & 2)

20 September & 4 October 2016

How to Modernise Legacy Applications

25 October 2016

How to enable Continuous Delivery of Software into Production

10 November

How to Operate Cloud Native Applications

6 & 20 December 2016

How quickly can you react to customer changes or the speed of your market?

Can you experiment with ideas, learn from mistakes and identify patterns that work?

THE JOURNEY TO CLOUD NATIVE

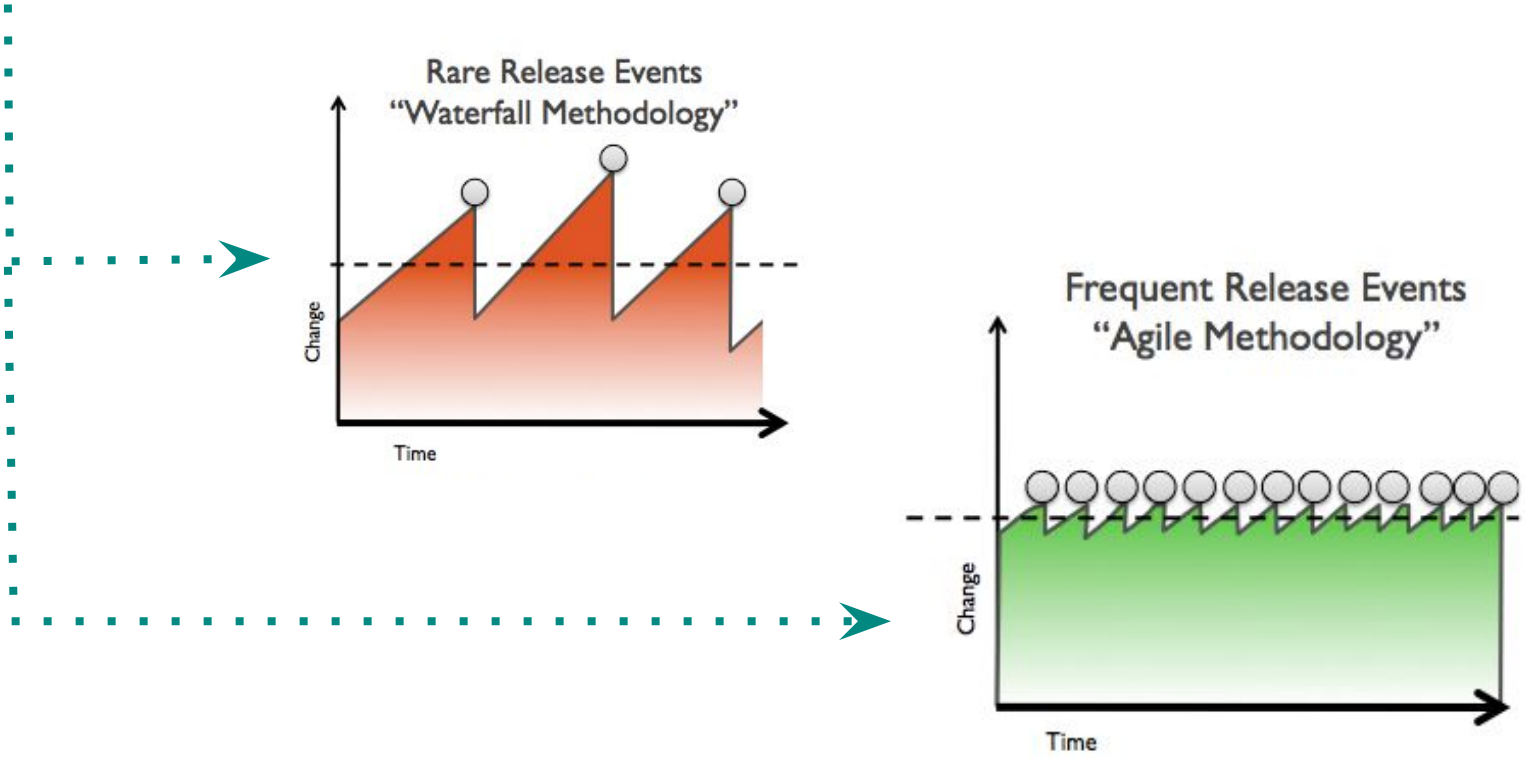
Cloud Native describes the patterns of high performing organizations delivering software faster, consistently and reliably at scale

Testing

Pivotal

“I have an absolutely brilliant idea ……”

Now let’s build it!



Write your tests first - TDD

```
@Test
public void testNotClever() throws Exception {
   assertEquals(true, “You’re silly person” == cleverCode.giveMeACleverResponse(null));
}
```

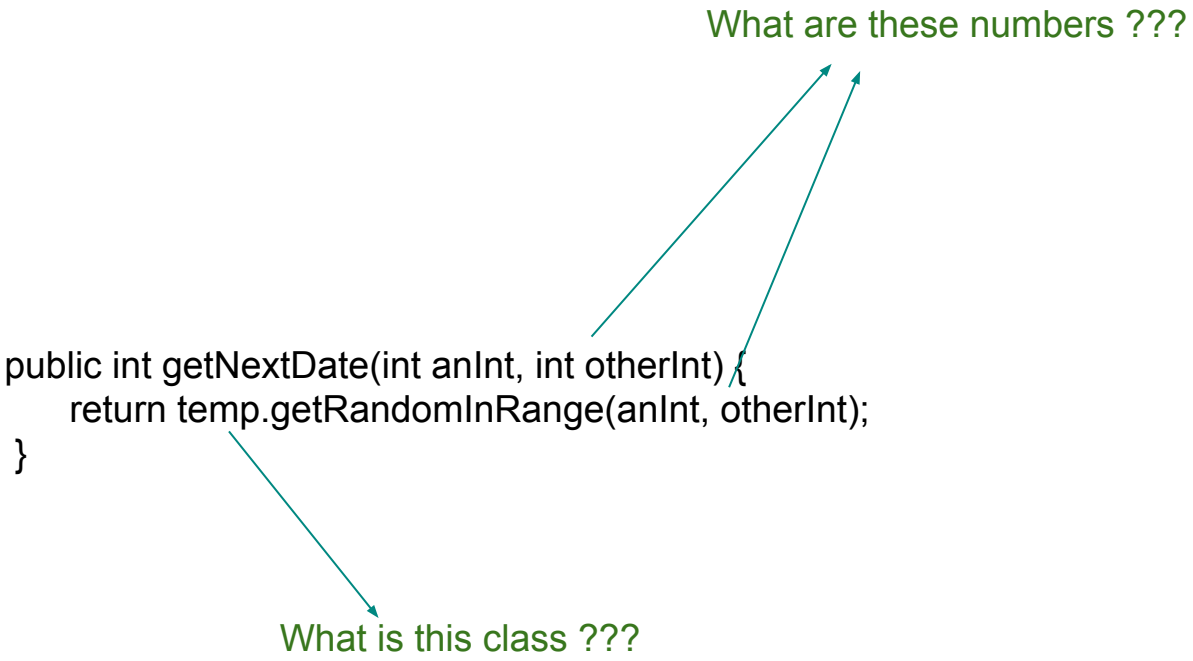
```
public boolean giveMeACleverResponse() {
   ……..
}
```

Style

Pivotal

Code Hospitality

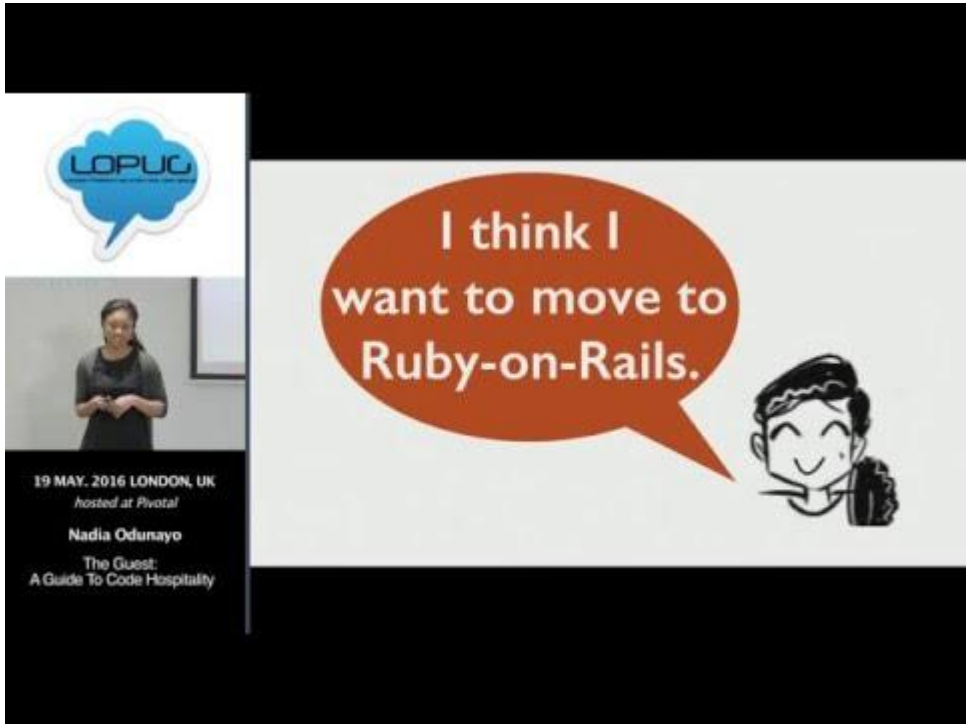
Code Hospitality



Code Hospitality

```
public int getNextBillingDate(int anInt, int OtherInt) {
      return temp.getRandomInRange(anInt, OtherInt);
 }
```

Code Hospitality

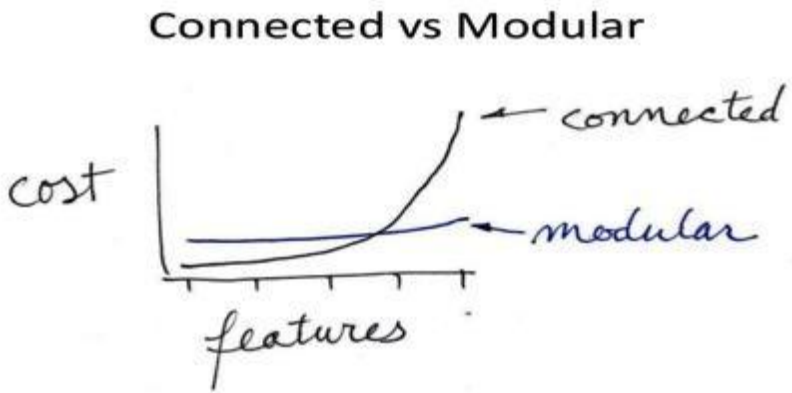


https://www.youtube.com/watch?v=U3XcUrupcYg

Architecture

Pivotal

What happens to software over time?



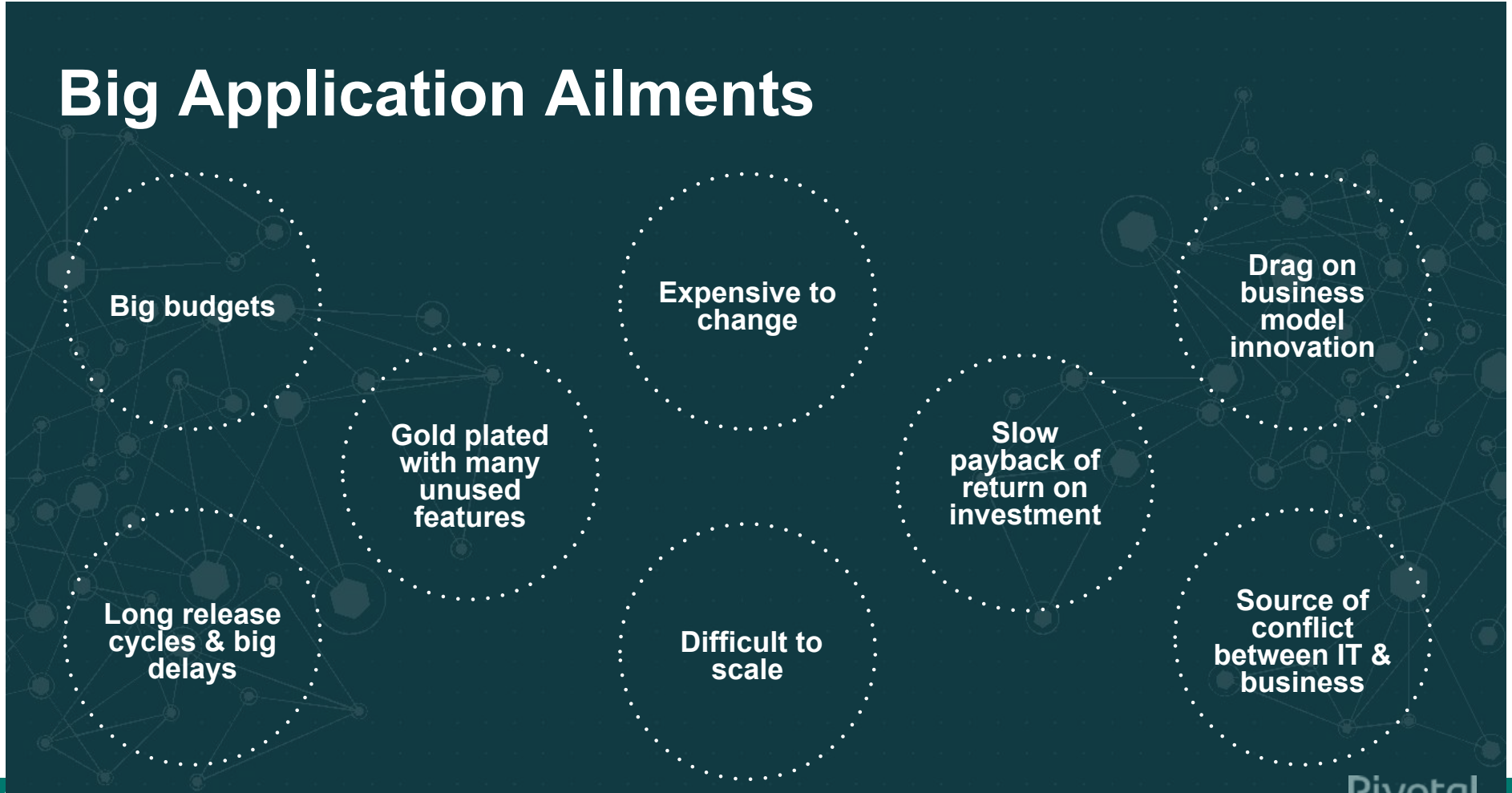
“In a connected system, elements are highly available to each other (via global state, for example). A modular design has connections deliberately kept to a minimum.

….

During the takeoff phase, the team is constantly trying to add value by increasing the chance of survival. During the cruise phase, reducing costs adds the most value. A different mix of activities goes into achieving these different goals.”

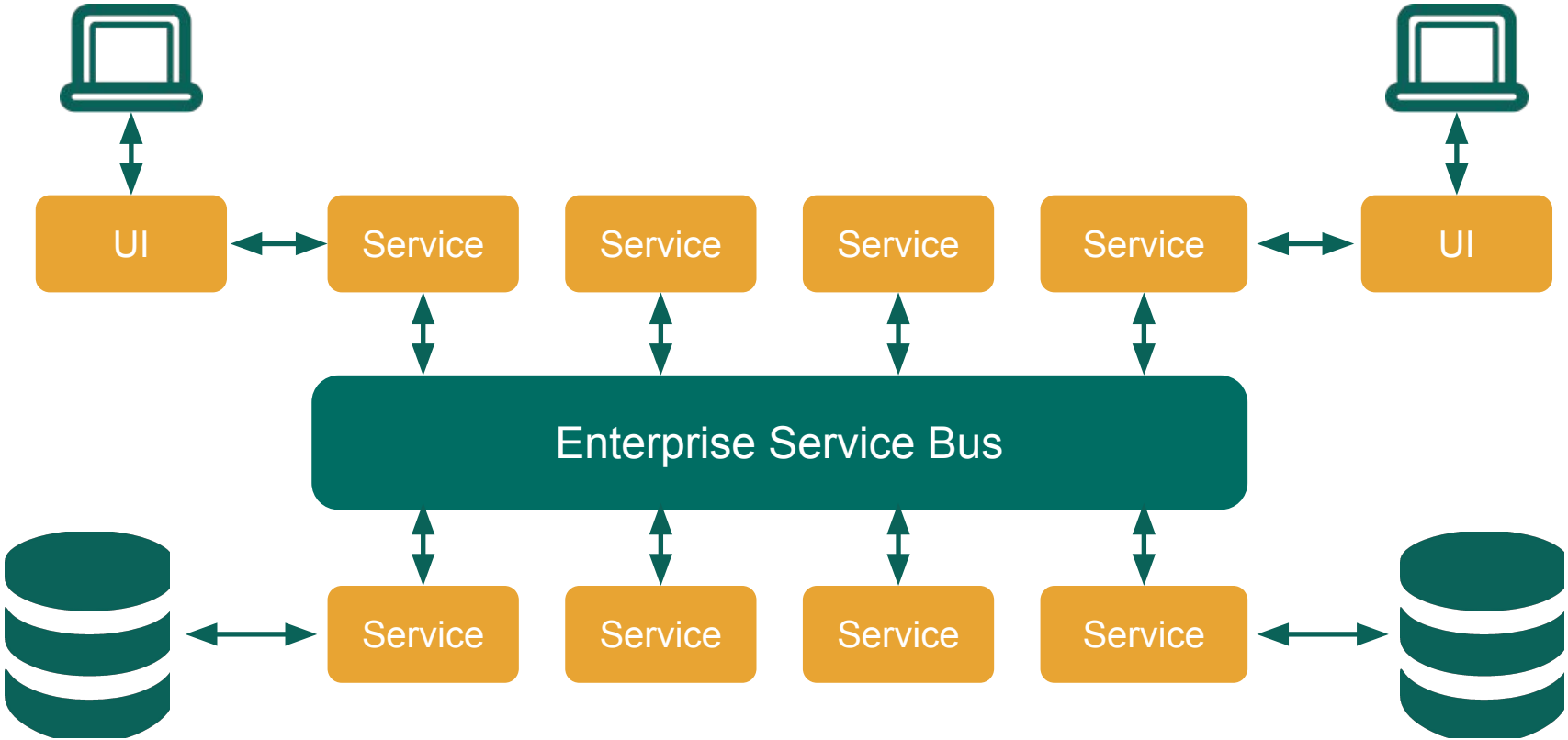
Kent Beck August 12th, 2009 in Responsible Development, Startups

Big Application Ailments



Pivotal

Not Traditional (ESB-centric) SOA



Pivotal

Why Microservices?

- A Distributed System creates reusable services
  - So, it's just like SoA - ?
  - NO -> It's SoA done right. Single function, single data domain
  - Interconnectivity using common transport but stateless communication
  - A microservice can run on it's own (low coupling) and provides it's own business value (high cohesion)
  - Anti-fragility

- Distributed Systems with defined responsibilities are better by design:
  - Individual services can be scaled separately
  - Individual services can be managed in separate runtimes
  - Interconnection is less rigid

- Code can be released faster but safely

Pivotal

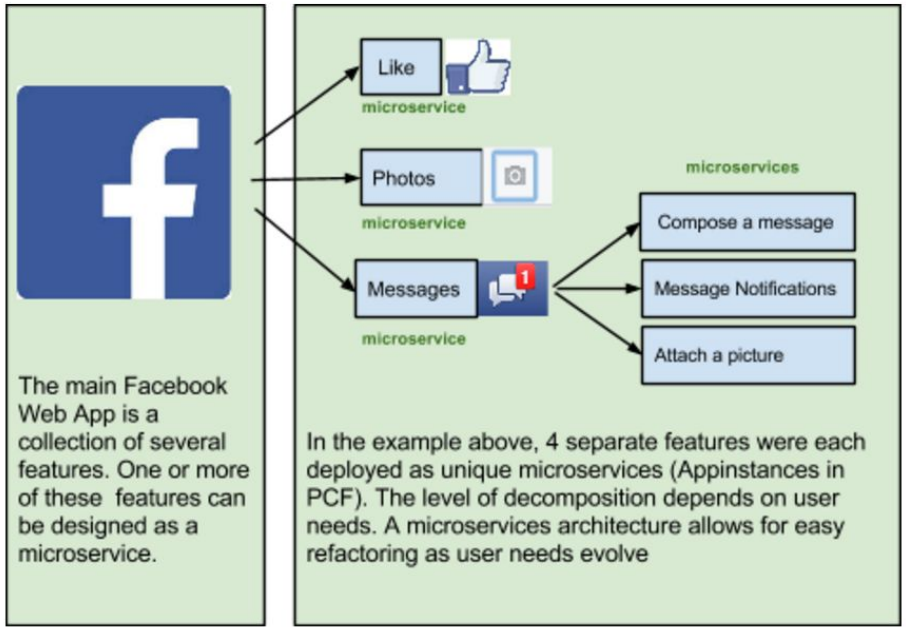
DEFINE: Microservice

If every service has to be updated in concert,
it's not loosely coupled!

Loosely coupled service oriented architecture with bounded contexts

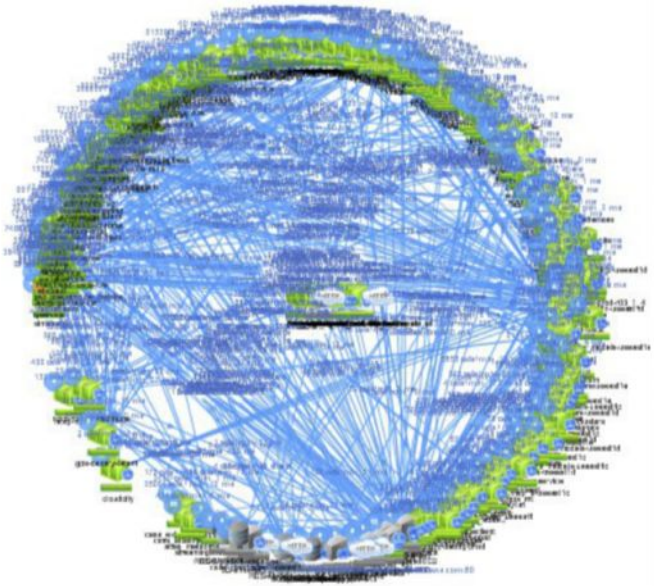
If you have to know about surrounding services you don't have a bounded context.

DEFINE: Microservice

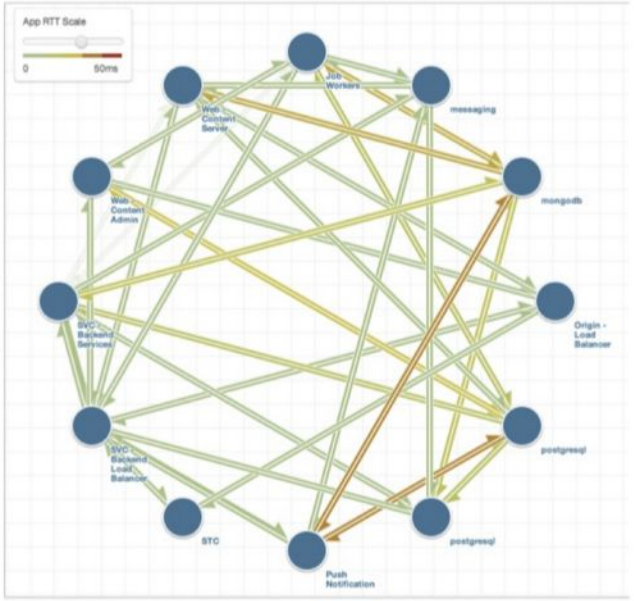


Pivotal

But Microservices!



Netflix



Gilt Groupe (12 of 450)

Twitter

Pivotal

Anatomy of a cloud native framework

Cloud

Runtime Platform, Infrastructure Automation boilerplate patterns (provision, deploy, secure, log, data services, etc.)

Pivotal Cloud Foundry

Desktop

Application coordination boilerplate patterns

Spring Cloud

Application configuration boilerplate patterns

Spring Boot

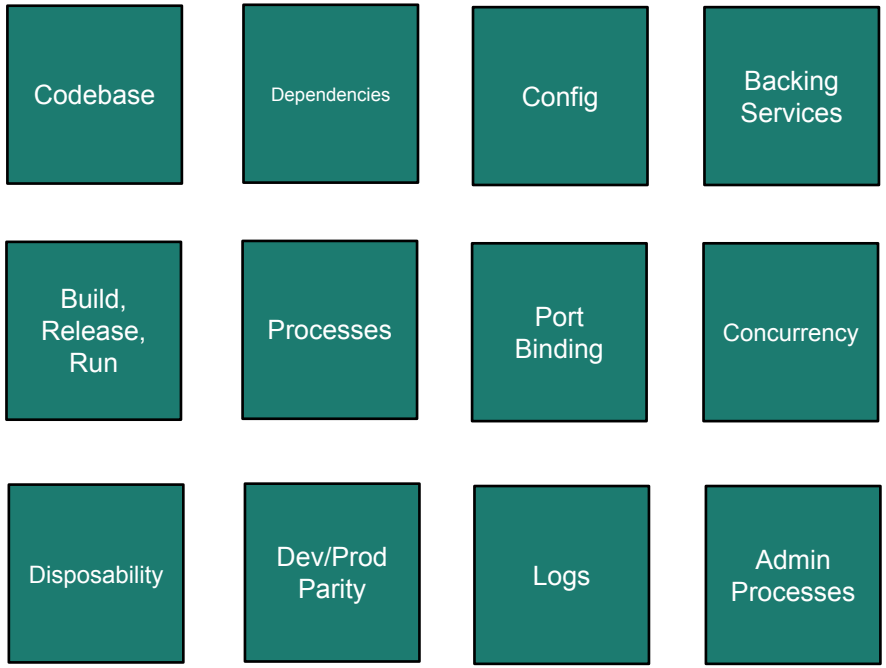
Enterprise application boilerplate patterns

Spring IO

Pivotal

Use 12 factor app principles to create cloud ready applications

- A set of best practices for developing and deploying cloud-native software.
- Practices translate into platform features and workflow requirements.



The Twelve-Factor App

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

esignation-matisse.cfapps.io

otal Personal Spring Coding Microservices Docker FE EMEA Useful Stuff - FE EMEA

spring io

Postal Code	City	County	State Code	Map
GU7 2	Aaron's Hill	Surrey	SU9543	
KT16 8	Abbey Mead	Surrey	TQ0467	
SO23 7	Abbots Worthy	Hampshire	SU4932	
SO24 9	Abbotstone	Hampshire	SU5634	
GU1 1	Abbotswood	Surrey	TQ0051	
SP11 7	Abbotts Ann	Hampshire	SU3243	
RH5 6	Abinger Common	Surrey	TQ1145	
RH5 6	Abinger Hammer	Surrey	TQ0947	
PO14 4	Abshot	Hampshire	SU5105	
B27 7	Acock's Green	West Midlands	SP1183	
KT15 1	Addlestone	Surrey	TQ0464	
KT15 2	Addlestonemoor	Surrey	TQ0465	
GU5 9	Albury	Surrey	TQ0547	
GU5 9	Albury Heath	Surrey	TQ0646	
B13 0	Alcester Lane's End	West Midlands	SP0780	
CV2 1	Alderman's Green	West Midlands	SP3583	
SO16 5	Aldermoor	Hampshire	SU3915	
GU11 1	Aldershot	Hampshire	SU8650	
WS9 8	Aldridge	West Midlands	SK0500	
GU6 8	Alfold	Surrey	TQ0334	

Displaying items 1 through 20 of 2036 in 102 pages.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62

Pivotal

Demo

Two Apps, each with their own codebase:

https://github.com/skazi-pivotal/spring-boot-cities-service

https://github.com/skazi-pivotal/spring-boot-cities-ui

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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.
  - Don't rely on implicit dependencies from the deployment environment

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

12-Factor Application

I. Codebase

One codebase tracked in SCM, many deploys

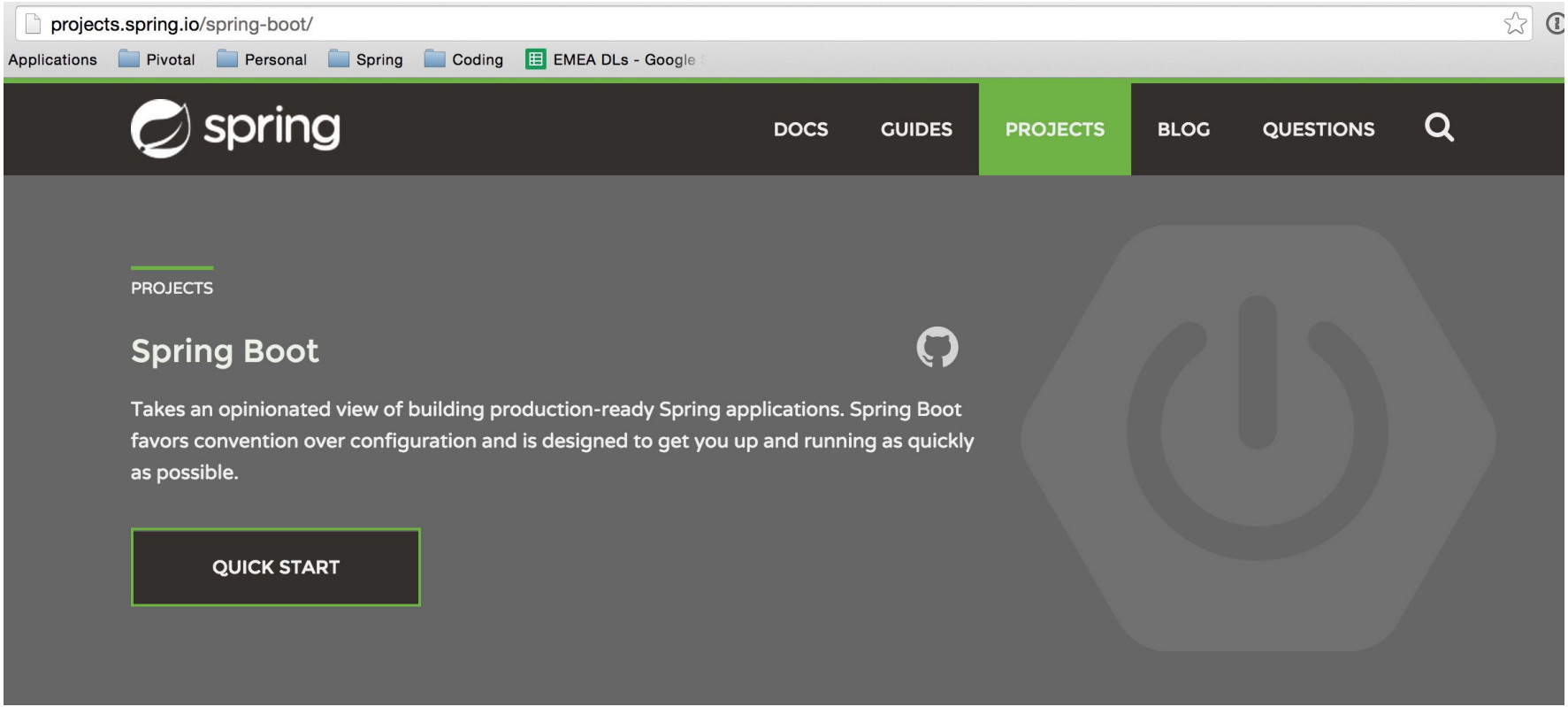
II. Dependencies

Explicitly declare and isolate dependencies

III. Configuration

Store config in the environment

- Configuration
  - Anything that varies by deployment should not be hard-coded
  - Environment variables or configuration server recommended



Pivotal

Why Spring Boot?

- “MICRO” doesn’t mean small
  - It means to decompose
  - Separate distinct components (sort, tail …..)

- Easy to build, test and DEPLOY
  - Microservices should perform one thing and perform it well
  - Typically concise codebase
  - Ship everything they need
  - Can deployed standalone or in the cloud, singly reducing operations overhead

- Spring Boot:
  - Is Opinionated
  - Gets you up and running quickly

Pivotal

Enhanced Application with Spring Boot

MAGIC!!

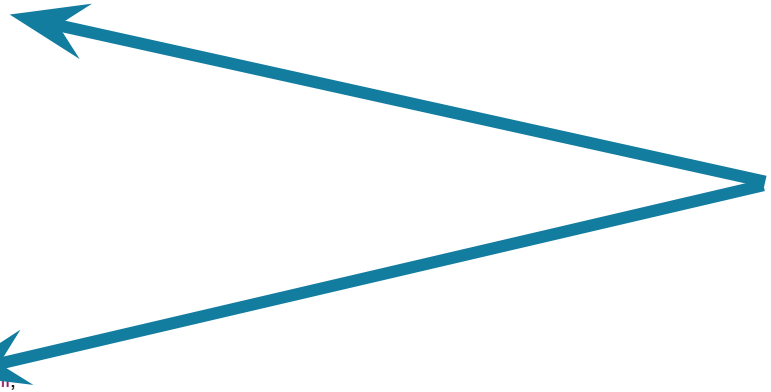
- Tags the class as a source for Spring Beans
- Asks Boot to automatically add beans based on classpath
- Tell Spring to look for other components, configs etc. in the same package

```
package hello;

import java.util.Arrays;

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

@SpringBootApplication
public class Application {

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

}
```

Pivotal

DB Connection: From Local to Bean

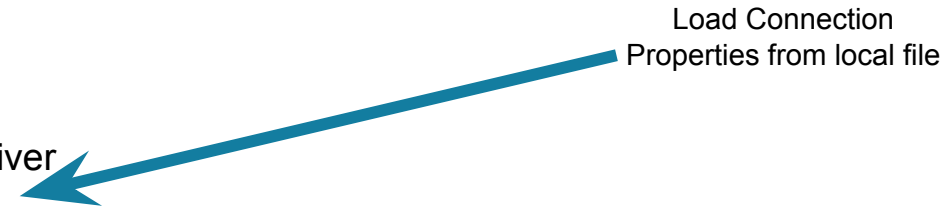
```
@Configuration
@ConfigurationProperties(prefix="spring.datasource")
public class DataSourceConfig {
        private String driverClassName;
        private String url;
        private String username;
        private String password;

        public DataSourceConfig() {
                super();
        }

        @Bean
        public DataSource dataSource() {
                SimpleDriverDataSource dataSource = null;
                try {
                        Class.forName(this.driverClassName);
                        dataSource = new SimpleDriverDataSource();
                        dataSource.setDriver(DriverManager.getDriver(this.url));
                        dataSource.setUrl(this.url);
                        dataSource.setUsername(this.username);
                        dataSource.setPassword(this.password);
                        Logger.INSTANCE.log("Connected to: " + this.url);
                } catch (Exception e) {
                        throw new IllegalStateException("An Exception occurred initialising datasource", e);
                }

                return dataSource;
        }
```



Load Connection Properties from local file

DB Connection: From Local to Bean

```
spring:
  profiles: local
  datasource:
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://127.0.0.1/ccmdemo
    username: suf
    password: password
```



The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

12-Factor Application

IV. Backing Services

Treat backing services as attached resources

V. Build, Release, Run

Strictly separate build and run stages

VI. Processes

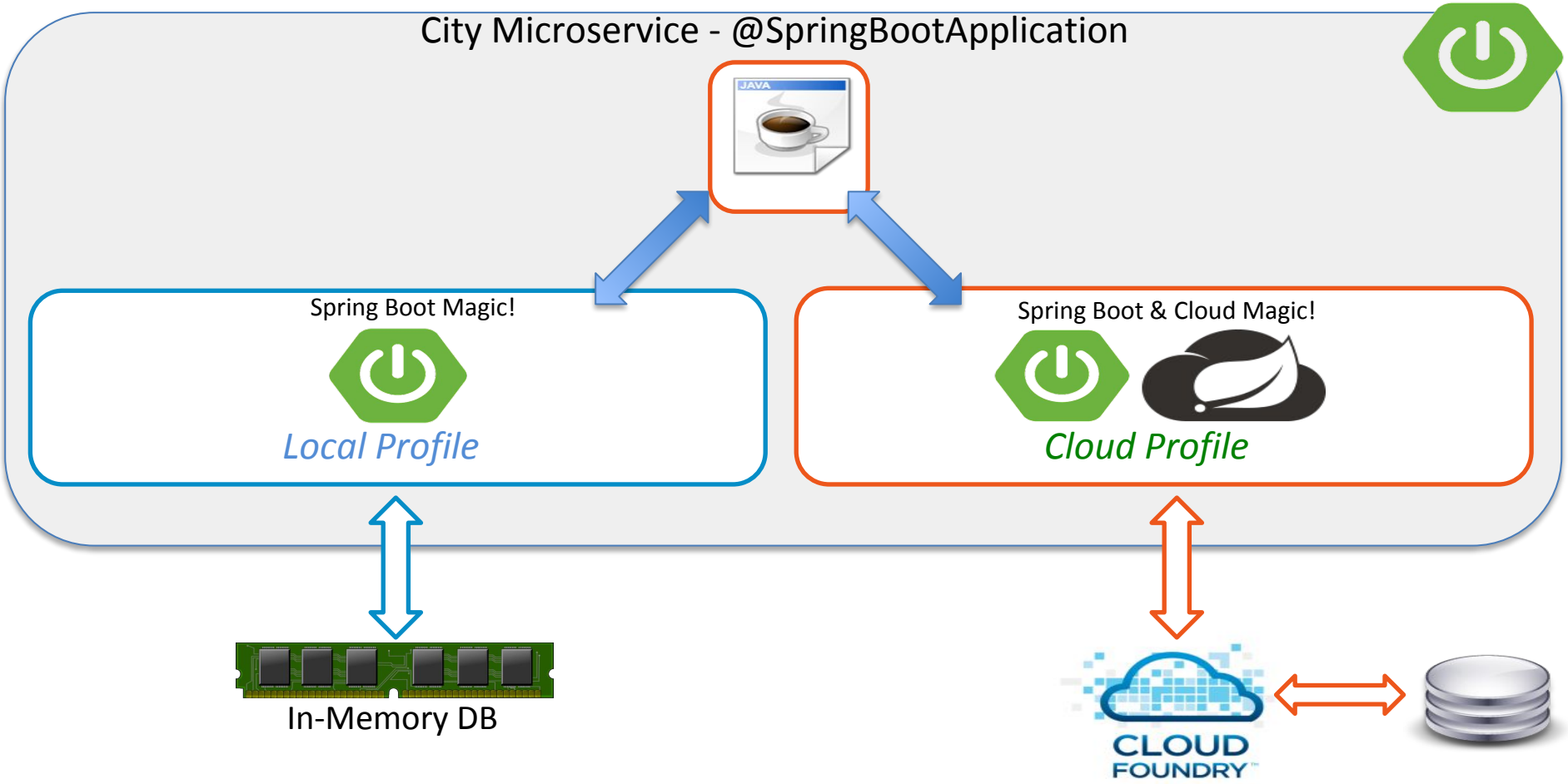
Execute app as stateless processes

- 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)

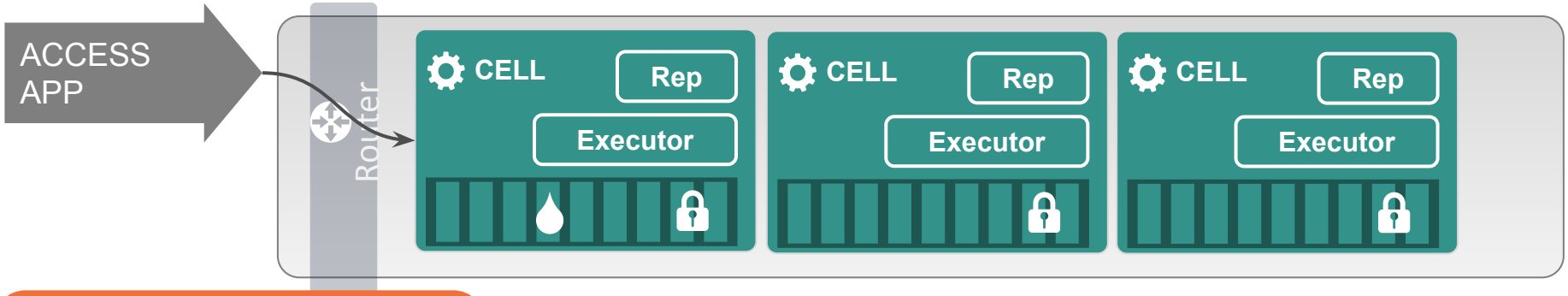
Cities - Service

/cities


```
{
  "_embedded" : {
    "cities" : [ {
      "name" : "Aaron's Hill",
      "county" : "Surrey",
      "stateCode" : "SU9543",
      "postalCode" : "GU7 2",
      "latitude" : "51.18277",
      "longitude" : "-0.63502",
      "_links" : {
        "self" : {
          "href" :
"http://cities-service-recognisable-wrongheadedness.cfapps.io/cities/2"
        },
        "city" : {
          "href" :
"http://cities-service-recognisable-wrongheadedness.cfapps.io/cities/2"
        }
      }
    }, {
      "name" : "Abbey Mead",
   ……….
```


City Microservice - @SpringBootApplication
JAVA
Spring Boot Magic!
Local Profile
Spring Boot & Cloud Magic!
Cloud Profile
In-Memory DB
CLOUD
FOUNDRY

Managed Service



Create Connection:

```
cf create-service
p-mysql 512mb MyDB
```

Expose Connection to App:

```
cf bind MyApp MyDB
```

Access a Connection:

```
"VCAP_SERVICES": {
  "MyDB": [
   {
    "credentials": {
     "hostname": "mysql.local.pcfdev.io"
     "jdbcUrl":
"jdbc:mysql://mysql.local.pcfdev.io:3306
    ]
   }
```

Pivotal

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

12-Factor Application

IV. Backing Services

Treat backing services as attached resources

V. Build, Release, Run

Strictly separate build, release and run stages

VI. Processes

Execute app as stateless processes

- Build, Release, Run
  - Build stage – converts codebase into build (version)
    - Including managed dependencies
  - Release stage – build + config = release
  - Run – Runs app in execution environment
- In Cloud Foundry, these stages are clearly separated with **`cf push`**

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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

- Processes
  - The app executes as one or more discrete running processes
  - Stateless
    - Processes should not store internal state
  - Share nothing
    - Data needing to be shared should be persisted
  - Any necessary state is externalized as a backing service

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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
  - The app is self-contained
    - For example, Tomcat is included in the droplet
  - Apps are exposed via port binding (including HTTP)
    - Every app instance is accessed via a URI and port number
  - One app can become another app's service

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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

- Concurrency
  - Achieve concurrency by scaling out horizontally
    - Scale by adding more app instances
  - Individual processes are free to multithread

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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
    - Enhances scalability and fault tolerance
  - Should exit gracefully / finish current requests

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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
  - This enables high quality, continuous delivery

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

12-Factor Application

X. Dev/prod parity

Keep dev, staging, prod as similar as possible

XI. Logs

Treat logs as event streams

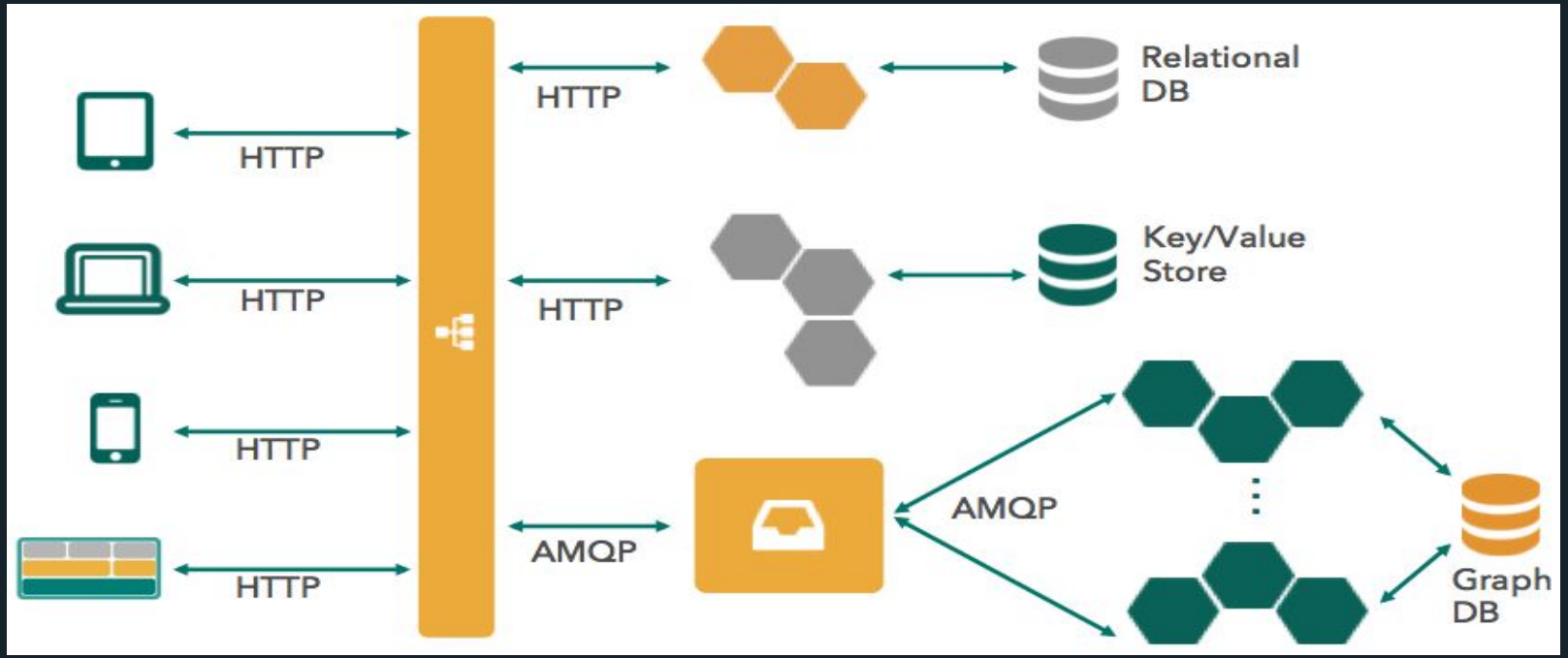
XII. Admin processes

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 stdout
  - Separate log managers handle management
    - Logging as a service
- Can be managed via tools like Papertrail, Splunk ...
  - Log indexing and analysis

Latency Analysis Of Microservices

Microservice architectures are often a graph of components distributed across a network



Pivotal

Dapper Paper By Google is where it started

This paper described Dapper, which is Google's production distributed systems tracing infrastructure

Design Goals :

- Low overhead
- Application-level transparency
- Scalability

Based on :

Concept of Traces and Spans

Pivotal

Distributed Tracing (Theory)

Distributed Tracing is a process of collecting end-to-end latency graphs in near real time.

As a request flows from one microservice to other in a system through ingress and egress, we need a way to trace it and provide info such as how much time it took for a request to complete? Which microservice was responsible for the delay?

Distributed Tracing Systems are often used for this purpose. An example of such a system is Zipkin.

Pivotal

Tracers (Theory)

- As a request is flowing from one microservice to another, **tracers** add logic to create unique trace ID.
- **Trace ID** is generated when the first request is made.
- As a request arrives at one microservice in it's journey, a new **Span ID** is assigned for that service and added to the trace.
- Tracers execute in your production apps! They are written to not log too much
- Tracers have instrumentation or sampling policy to manage volumes of traces and spans

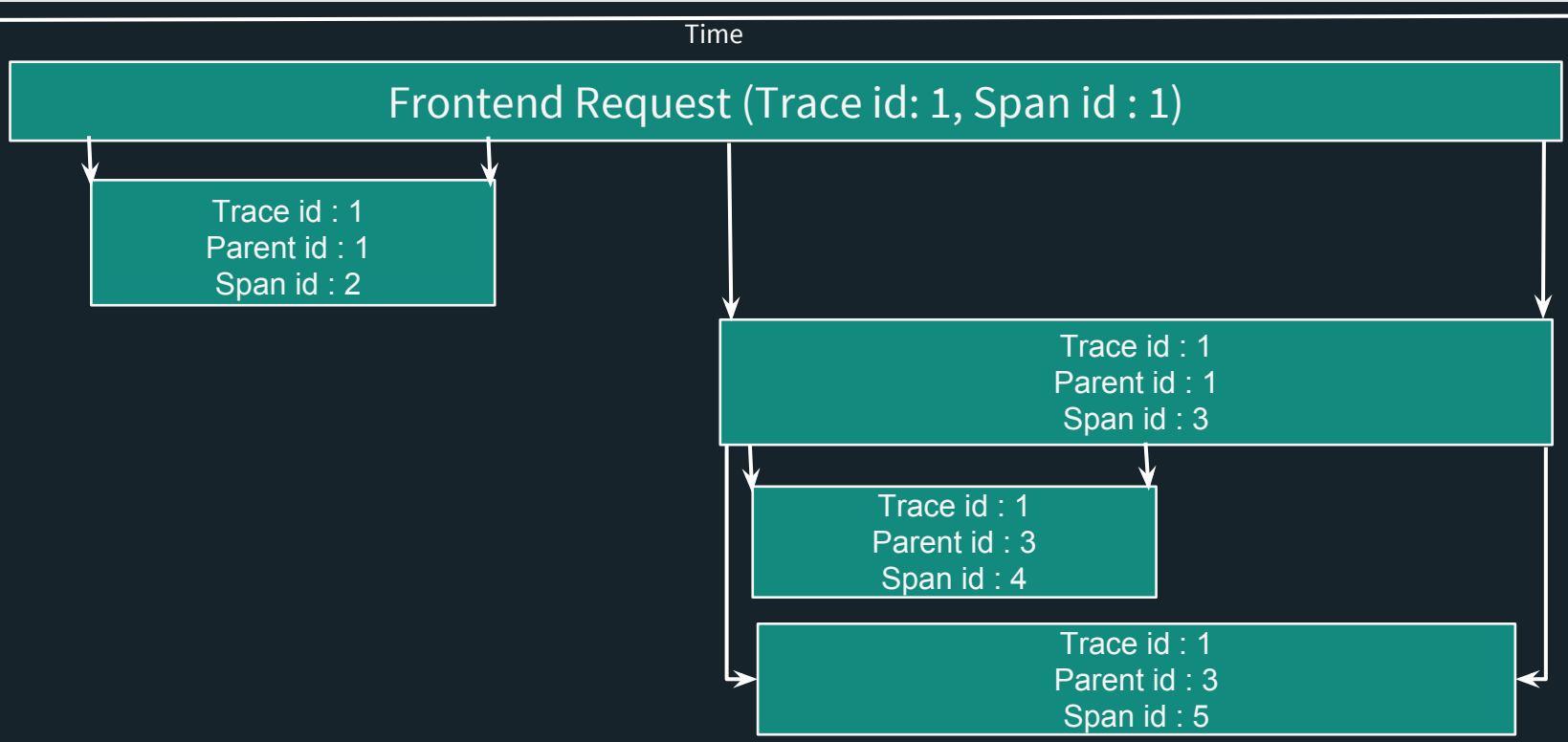
Pivotal

Traces and Spans (Theory)

- A trace represents the entire journey of a request, from one microservice to another
- A span represents a single operation call to a microservice
- If a microservice calls another microservice, then the calling span is known as parent span
- A span contains timestamped records which encode the span's start and end time, any RPC timing data, and zero or more application-specific annotations

Pivotal

Example Trace – Graph of spans

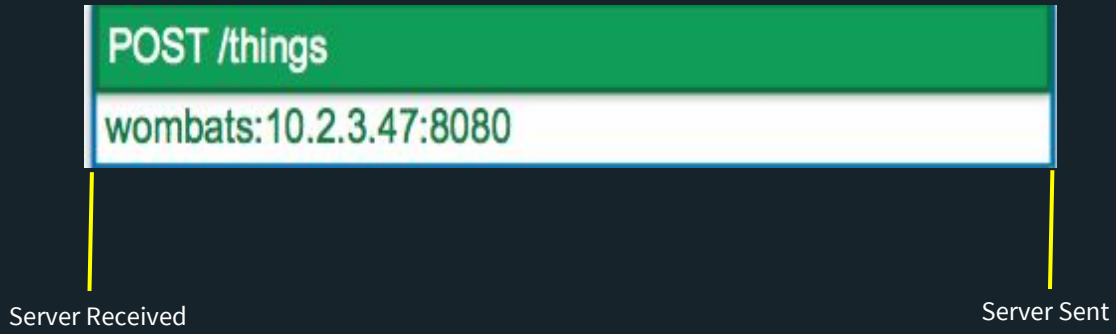


Pivotal

Example Span – An individual operation

Operation

Events



Tags

peer.ipv4	1.2.3.4
http.request-id	abcd-ffe
http.request.size	15 MiB
http.url	…&features=HD-uploads

Pivotal

Tracing Systems

Tracing systems collect, process and present data reported by tracers.

- aggregate spans into trace trees
- provide query and visualization for latency analysis
- have retention policy (usually days)

Pivotal

Zipkin is a distributed tracing system(Theory)

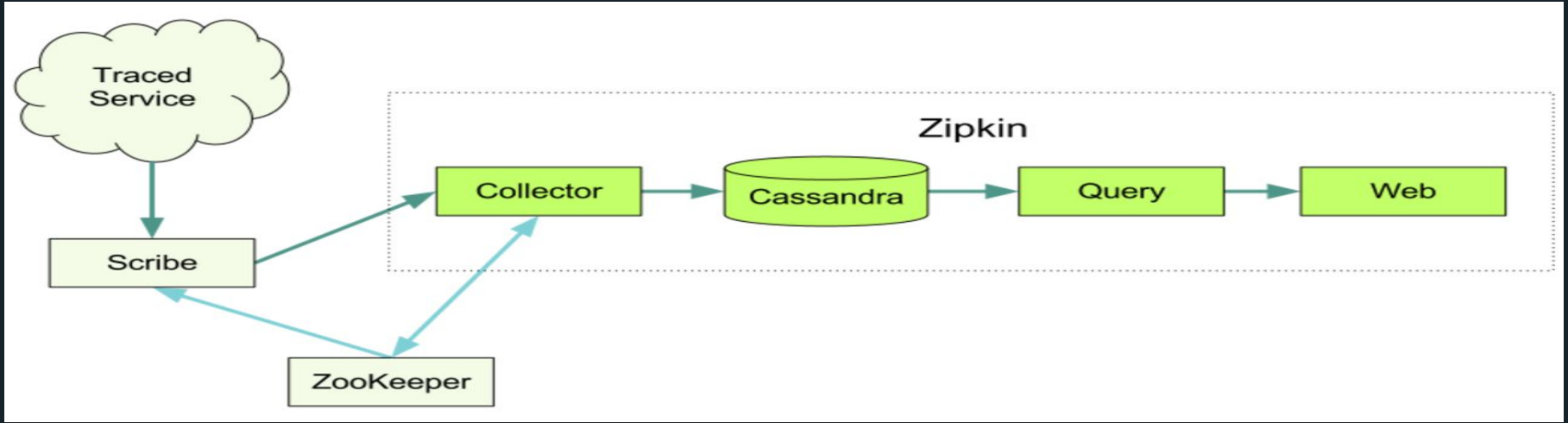
- Zipkin is a tracing system and was created by Twitter in 2012
- It is open sourced and it's implementation is based on Dapper Paper by Google
- Adrian Cole (Spring Cloud) is the main contributor
- In 2015, OpenZipkin became the primary fork

Goals :
Aggregate spans into trace trees
Provide query and visualization for latency analysis
Have retention policy

https://github.com/openzipkin https://gitter.im/openzipkin/zipkin

Pivotal

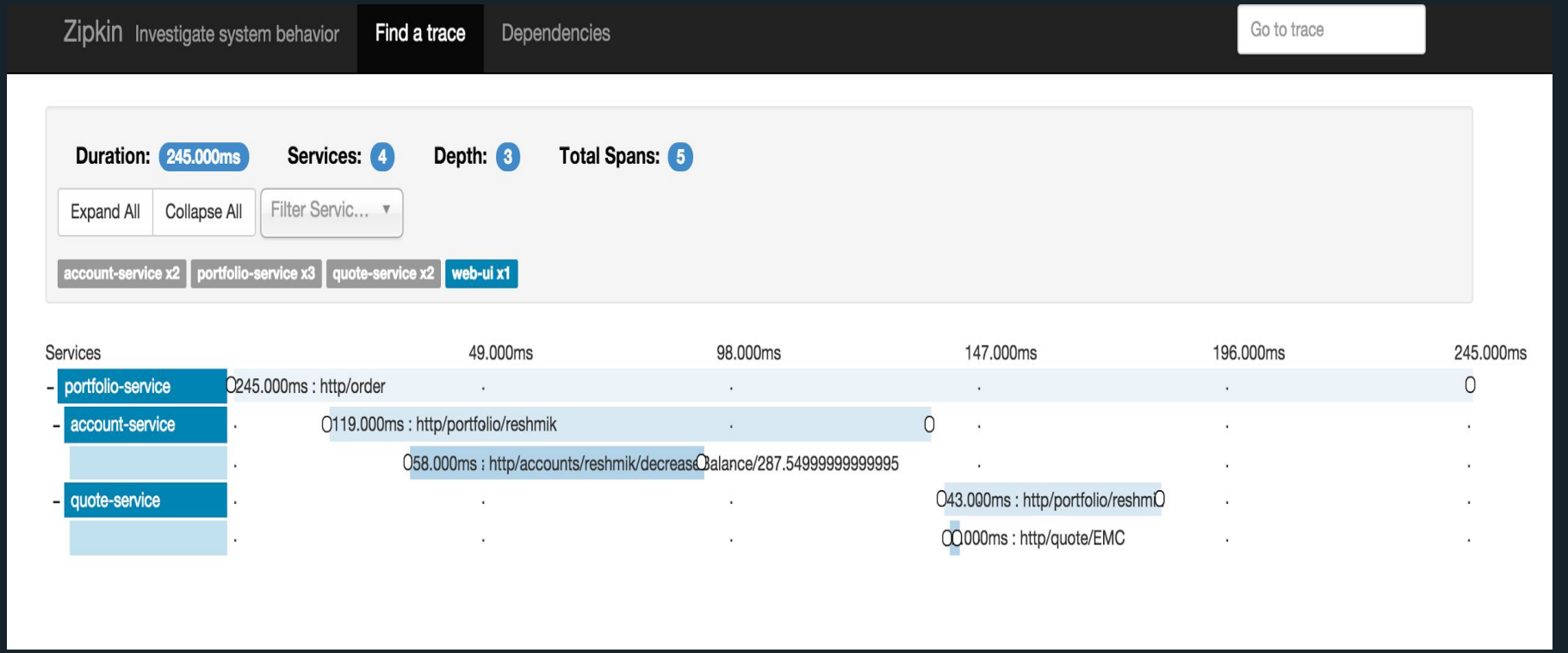
Zipkin Architecture



- Tracers collect timing data and transport it over HTTP or Kafka
- Collectors store spans in MySQL or Cassandra
- Users query for traces via Zipkin's Web UI or Api

Pivotal

Zipkin UI Example from Spring Trader app



Spring Cloud Sleuth is a Zipkin-Compatible tracer

- Spring Cloud Sleuth sets up useful log formatting for you that prints the trace ID and the span ID
- It includes instrumentation for Spring Boot, and a streaming collector
- Reports to zipkin via HTTP by depending on spring-cloudsleuth-zipkin
- Transparent to application developer
  1. Include starter BOM on app build system

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

  2. Specify “sampler”

```
@Bean
public AlwaysSampler defaultSampler() {
  return new AlwaysSampler();
}
```

Pivotal

The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

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

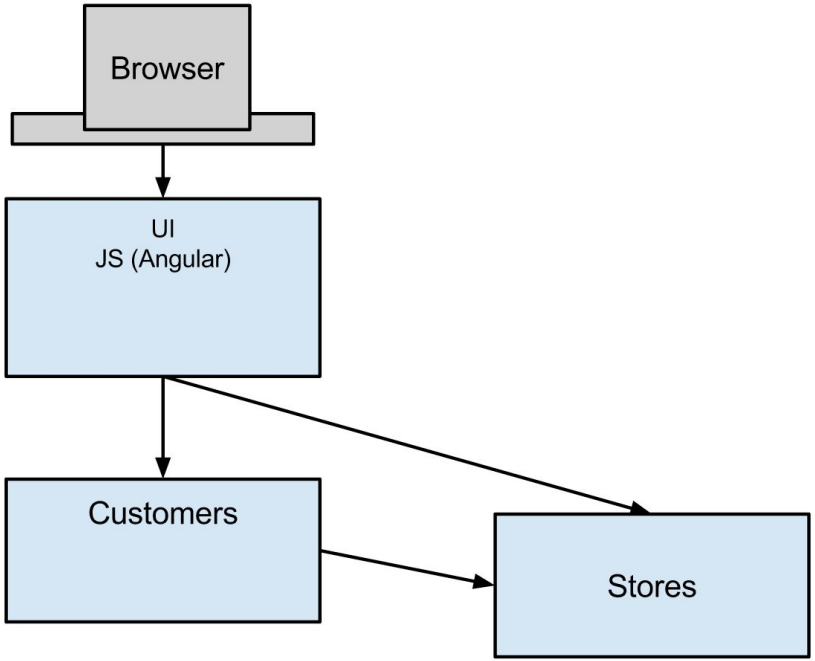
- Admin processes / management tasks run as one-off processes
  - Run admin processes on the platform
    - Leverages platform knowledge and benefits
  - DB migrations, one time scripts, etc.
  - Use the same environment, tools, language as application processes

- Eureka
- Hystrix + Turbine
- Ribbon
- Feign
- Zuul

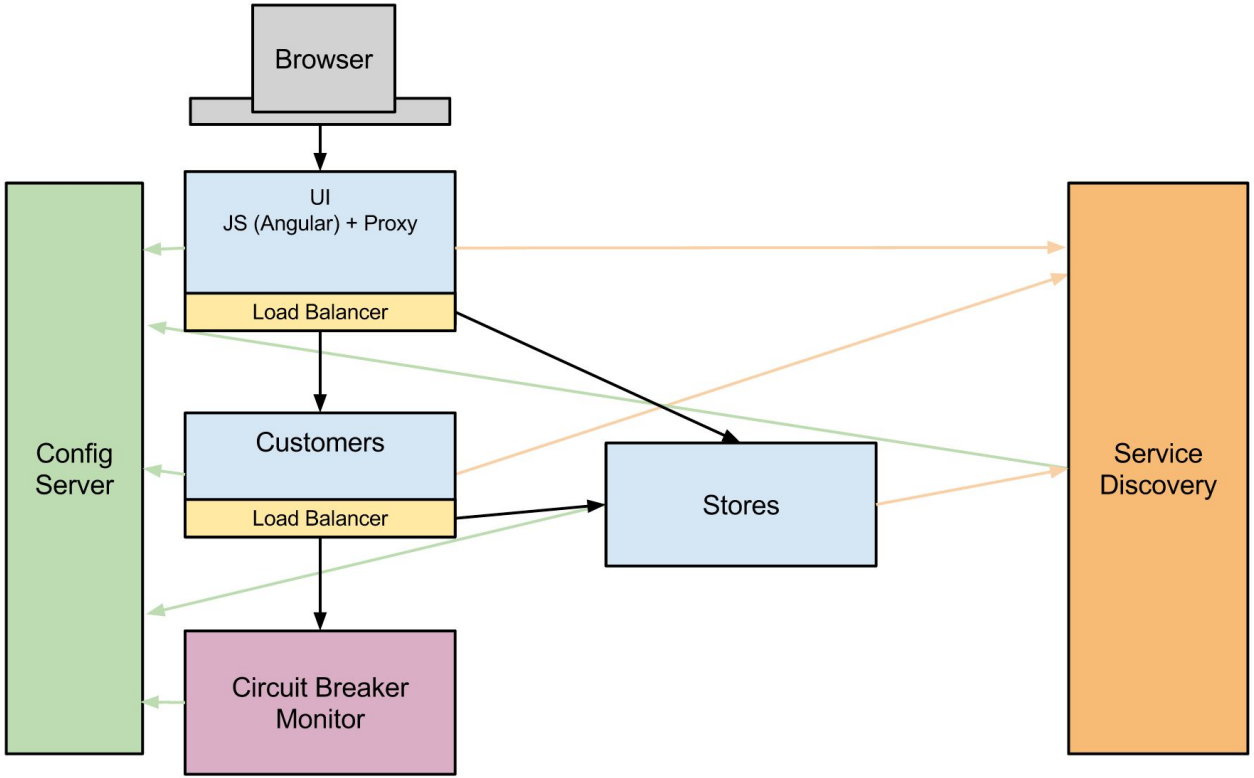
http://netflix.github.io

Pivotal

Example Distributed System: Minified

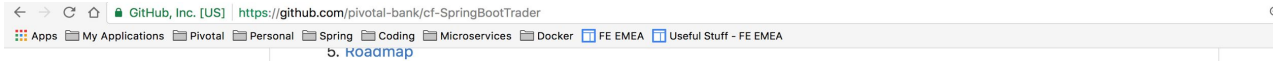


Example: Spring Cloud Services + Netflix OSS



Pivotal

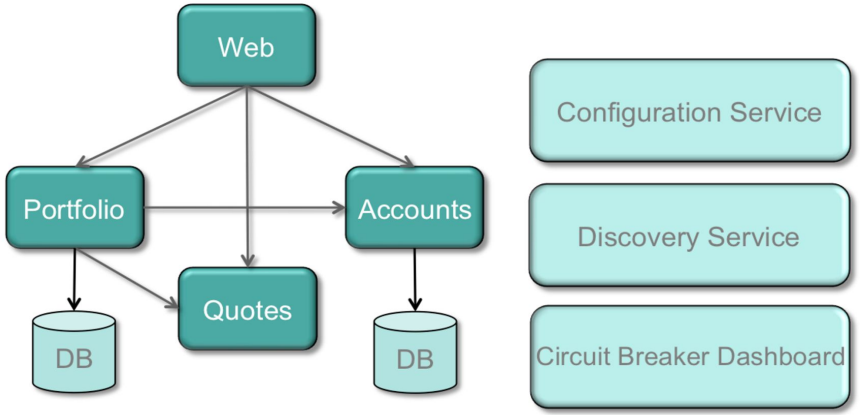
https://github.com/pivotal-bank/cf-SpringBootTrader



5. Roadmap
6. Contributing to the project

Architecture

The system is composed of 4 microservices. The relationship between the microservices is illustrated below.

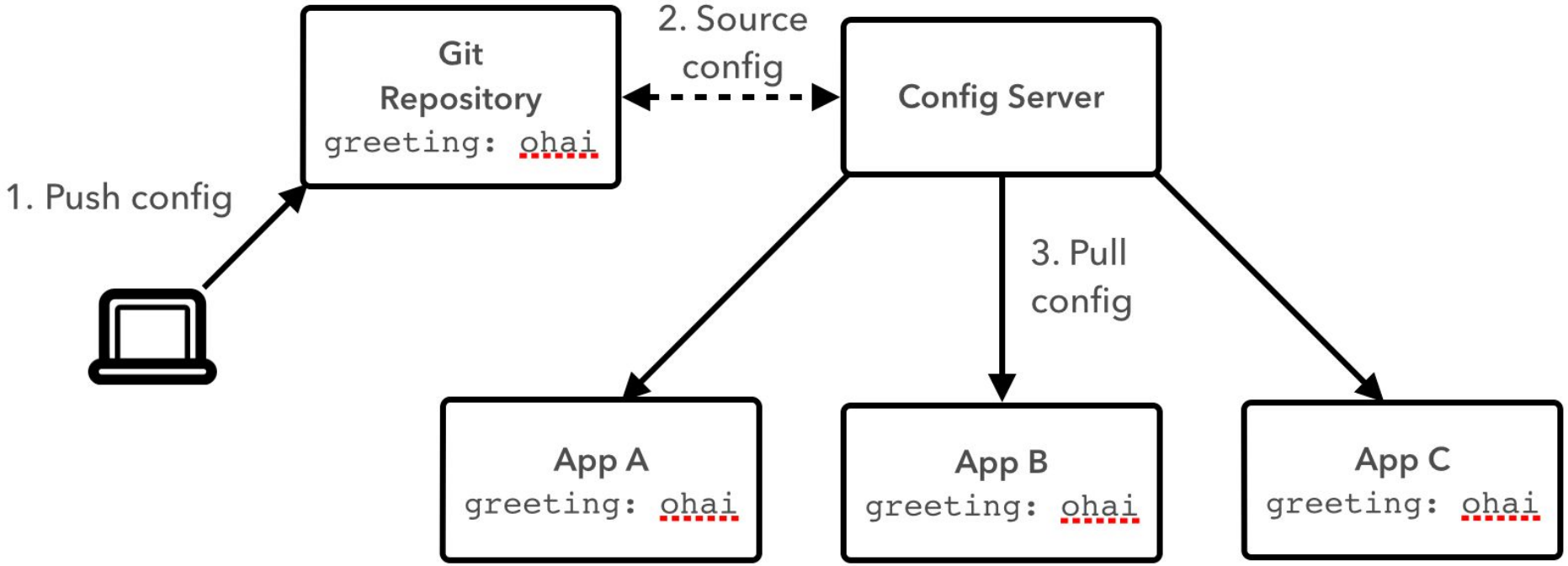


1. Quote Microservice

This service is a spring boot application responsible for providing up to date company and ticker/quote information. It does this by providing a REST api with 2 calls:

- `/quotes?q={symbol1,symbol2,etc}` Returns as up to date quote for the given symbol(s).
- `/company/{search}` Returns a list of companies that have the search parameter in their names or symbols.

Config Server



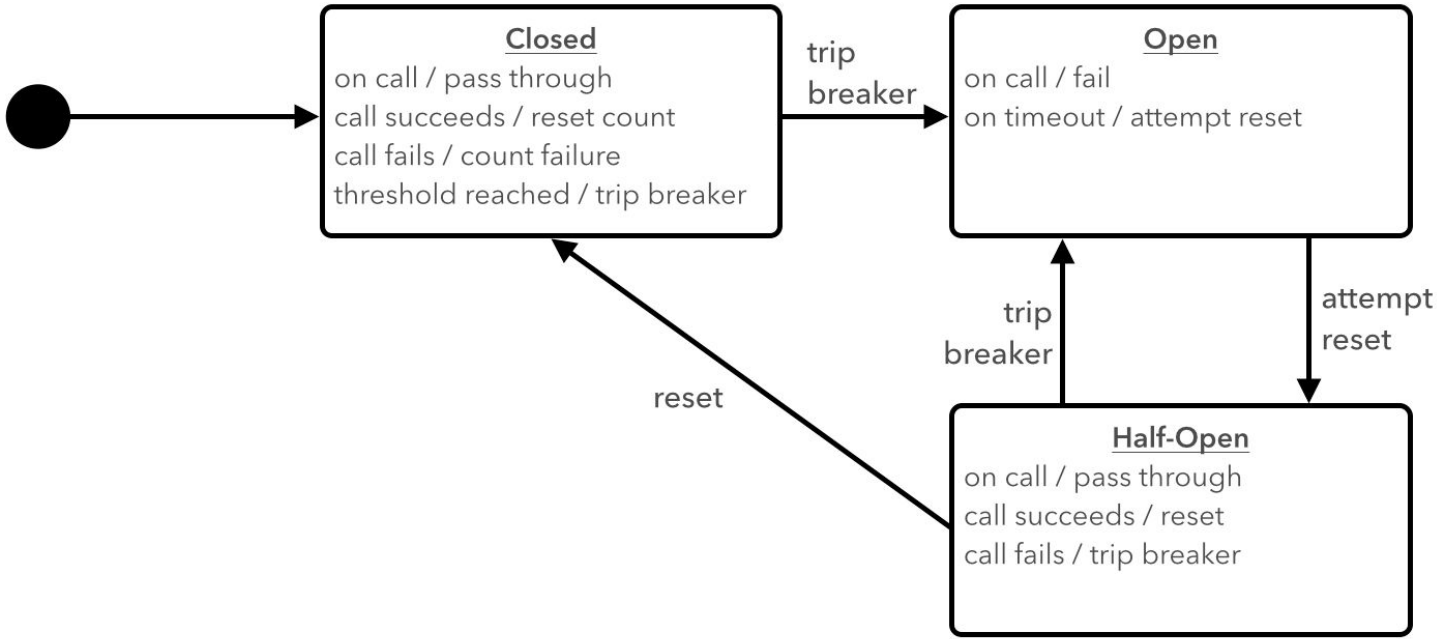
Pivotal

Fault Tolerance – Circuit Breakers

```
@SpringBootApplication
@EnableCircuitBreaker
@EnableDiscoveryClient
public class CustomerApp extends RepositoryRestMvcConfiguration {

    @Override
    protected void configureRepositoryRestConfiguration(RepositoryRestConfiguration
config) {
        config.exposeIdsFor(Customer.class);
    }

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

}
```

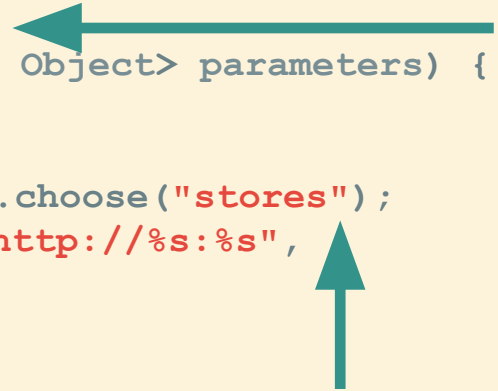
Pivotal

Fault Tolerance – Circuit Breakers



Enabling a Circuit Breaker

```
    @HystrixCommand(fallbackMethod = "defaultLink")
    public Link getStoresByLocationLink(Map<String, Object> parameters) {
        URI storesUri = URI.create(uri);
        try {
            ServiceInstance instance = loadBalancer.choose("stores");
             storesUri = URI.create(String.format("http://%s:%s",
instance.getHost(), instance.getPort()));
        }
        catch (RuntimeException e) {
            // Eureka not available
        }

        Traverson traverson = new Traverson(storesUri, MediaTypes.HAL_JSON);
        Link link = traverson.follow("stores", "search", "by-location")
                .withTemplateParameters(parameters).asLink();

        return link;
    }
```



Pivotal

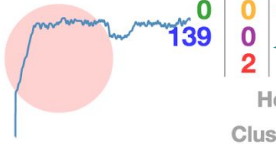
Hystrix Stream: http://pivotalcustomers.cfapps.io/hystrix.stream

HYSTRIX
DEFEND YOUR APP

Circuit Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

Success | Short-Circuited | Timeout | Rejected | Failure | Error %

getStoresByLocationLink



0 0 100.0 %
139 0
2

Host: 13.5/s
Cluster: 13.5/s
Circuit Open

Hosts	1	90th	0ms
Median	0ms	99th	0ms
Mean	0ms	99.5th	0ms

Thread Pools Sort: Alphabetical | Volume |

StoreIntegration

Host: 0.0/s
Cluster: 0.0/s

Active	0	Max Active	0
Queued	0	Executions	0
Pool Size	10	Queue Size	5

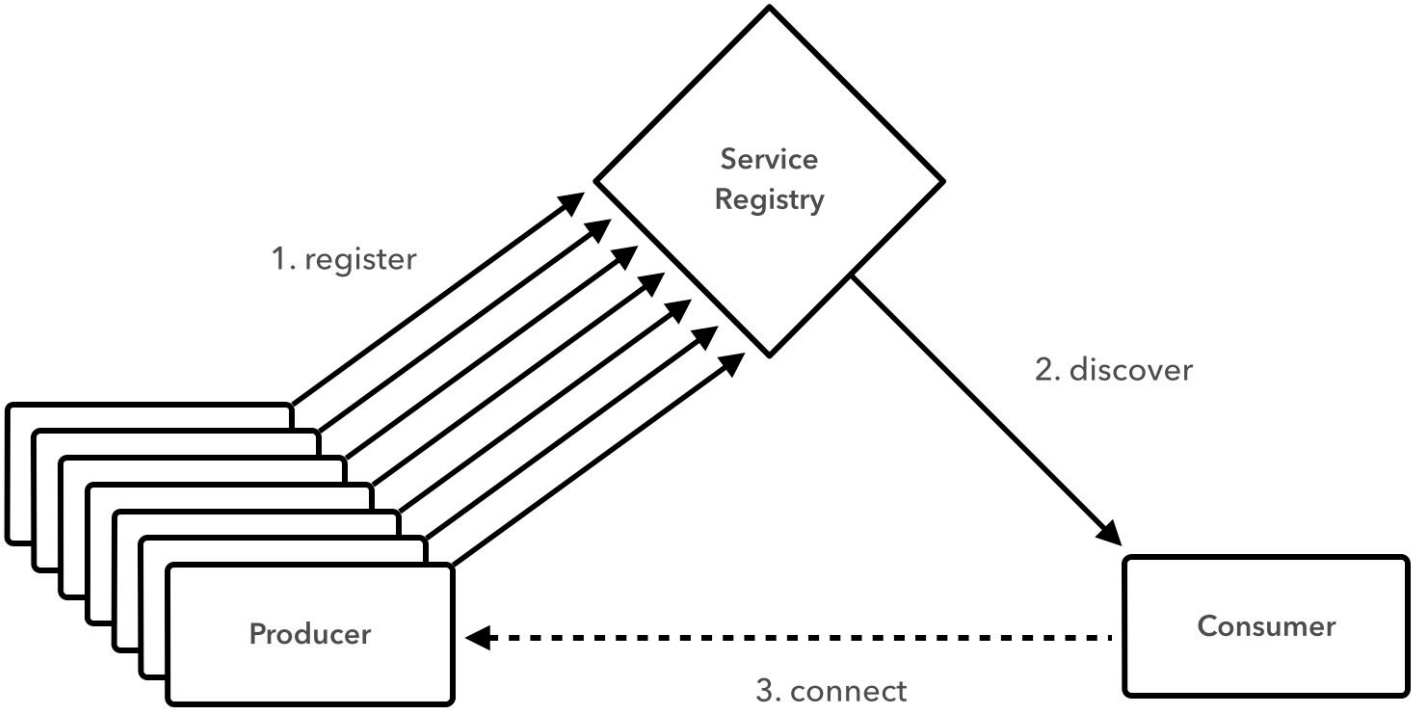
Pivotal

Service Registration/Discovery

```
@SpringBootApplication
@EnableCircuitBreaker
@EnableDiscoveryClient
public class CustomerApp extends RepositoryRestMvcConfiguration {

    @Override
    protected void configureRepositoryRestConfiguration(RepositoryRestConfiguration
config) {
        config.exposeIdsFor(Customer.class);
    }

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

}
```

Pivotal

Service Registration/Discovery



The Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
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 the app as one or more 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 development, staging, and production as similar as possible

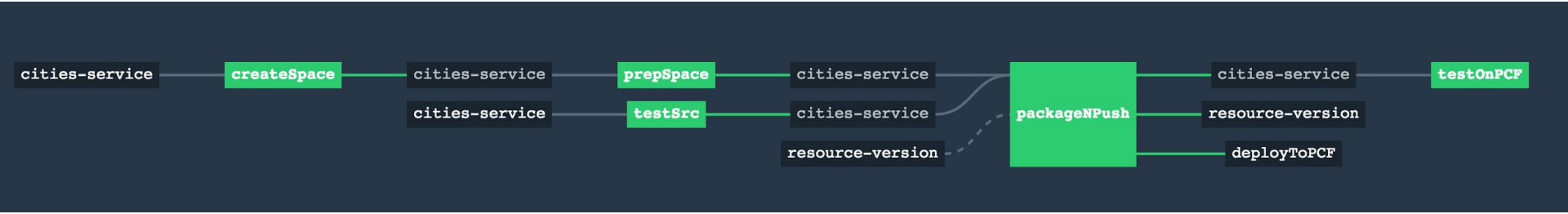
XI. Logs
Treat logs as event streams

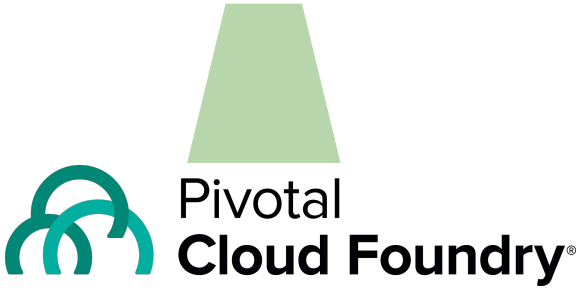
XII. Admin processes
Run admin/management tasks as one-off processes

http://12factor.net

Pivotal

CI/CD with Concourse (http://concourse.ci/)





Q&A

Pivotal


Pivotal®