

# Developing Applications with Cloud Foundry

Hands-On Workshop

Application Deployment using Pivotal Cloud  
Foundry

Version 1.5.a



Pivotal

# Copyright Notice

Copyright © 2015 Pivotal Software, Inc. All rights reserved. This manual and its accompanying materials are protected by U.S. and international copyright and intellectual property laws.

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

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

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



# Developing Applications with Cloud Foundry

## Introduction

Course Agenda and Pivotal Background

Pivotal

# Logistics

- Participants list
- Self introduction
- MyLearn registration
- Courseware
- Internet access
- Working hours
- Lunch
- Toilets
- Fire alarms
- Emergency exits
- Other questions?



# How You will Benefit

- Learn to use Pivotal CF to run applications
  - Introduction to PaaS and Cloud Foundry
  - How to use Pivotal CF
  - Running, debugging applications
  - How to use services
  - Buildpacks
  - And more ...
- Gain hands-on experience
  - 60/40 presentation and labs



# Covered in this section

- Topics
- Lab Setup
- Pivotal

# Day 1 – Pivotal CF Basics

- Introduction to Cloud Technologies
- Getting Started with Pivotal CF
- Cloud Foundry Concepts
- Using Command Line, Web Console, Eclipse Plugin
- Developing with Cloud Foundry
  - Logs, Troubleshooting, Manifests, Environment Variables



# Day 2 – Pivotal CF

- Using Services
- Using Buildpacks
- Managing Applications in Cloud Foundry
  - 3<sup>rd</sup> Party Log Management, Monitoring, Autoscaling, Zero-downtime deployments
- Cloud Foundry Architecture and Components

2

Pivotal™

# Day 3 – Management and Development

- Continuous Delivery
- Application Design Principles
- Spring for Cloud Applications
- Deploying Java and JVM Applications

3

# Covered in this section

- Topics
- **Lab Setup**
- Pivotal

# Lab Environment

- This course includes hands-on exercises
  - Designed to re-enforce presented concepts
- Exercises can be run on various environments
  - Windows, Mac, Linux
- There are some requirements:
  - Internet connection
  - Pivotal CF Installation / Account
    - Or Pivotal Web Services account
  - Cloud Foundry Command Line
  - GitHub Account (optional)
  - Eclipse or Spring Tool Suite

# Pivotal CF Installation / Pivotal Web Services Account

- Exercises will require a Pivotal CF environment
  - Your organization may already have one
- If not, use **Pivotal Web Services**:
  - Visit <https://run.pivotal.io>
  - Sign up for a trial account



# Cloud Foundry – Command Line Interface (CLI)

- Most exercises involve running commands to interact with Cloud Foundry
- Install Cloud Foundry Command Line Interface
  - Visit <https://console.run.pivotal.io/tools>
    - or <https://github.com/cloudfoundry/cli>
  - Download / Run installer for your platform
    - English, French, Mandarin, Portuguese, Spanish available



Pivotal

# Eclipse / Spring Tool Suite



- Some exercises involve inspecting / modifying existing program code
  - Emphasis is on *using* Cloud Foundry
  - Programming knowledge is not a strict requirement for this course, though many development topics are covered
  - *Do not install an IDE unless you normally use one*
- Install Eclipse or Spring Tool Suite
  - Visit <https://www.eclipse.org/downloads/> or <http://spring.io/tools>
  - Download / Run installer for your platform
  - Note: Java based examples use Java 8



- Course materials stored in GitHub
  - Visit <https://github.com>
  - GitHub account recommended, but not required
- Course Materials
  - Download from  
<https://github.com/S2EDU/CloudFoundryStudentFiles>
  - Download Zip or use Clone
    - Depending on if you have a GitHub account
    - (Advanced Git knowledge not required for this course)

# Covered in this section

- Topics
- Lab Setup
- **Pivotal**

# Pivotal

- A collaboration between
  - EMC
  - VMware
  - General Electric
- Born on April 1, 2013
  - Pre-IPO



Pivotal™

# Pivotal Platform

## Cloud Foundry

*Cloud Independence  
Microservices  
Continuous Delivery  
Dev Ops*



## Development

*Frameworks  
Services  
Analytics*



## Big Data Suite

*High Capacity  
Real-time Ingest  
SQL Query  
Scale-out Storage*



HAWQ



GREENPLUM



# Summary

- Agenda
- Pivotal

Let's get on with the course..!



Pivotal™

# Lab

## Setup Lab Environment



# What is Cloud Foundry?

## Overview

Introduction to Cloud Technologies

Pivotal

# What is Cloud Foundry?

- **Introduction to Cloud**
- Cloud Foundry
- Key Concepts
- Pivotal CF

# “The Cloud”



- Means many things to many people.
  - Distributed applications accessible over a network
    - Typically, but not necessarily, The Internet
  - An application and/or its platform
  - Resources on demand
  - Inherently virtualized
  - Can run in-house (private cloud) as well
  - Hardware and/or software sold as a commodity

# Types of Cloud Computing: IaaS

- Infrastructure as a Service
  - Replacement for physical hardware
  - Provides virtual hardware, OS, Network
  - Amazon Web Services (AWS), RackSpace, Microsoft Azure, VMware vCloud Air



Windows Azure

Pivotal

# Types of Cloud Computing: PaaS

- Platform as a Service
  - More than a raw machine with OS
  - Provides ready-made platform for running apps
  - CloudFoundry, Heroku, Google App Engine, Amazon Elastic Beanstalk



Windows Azure

Pivotal

# Types of Cloud Computing: SaaS

- Software as a Service
  - Complete software application
  - SalesForce.com, Google Apps, hundreds of examples



Windows Azure

Pivotal

# PaaS: Pizza as a Service!

You Manage

Provided

Traditional data-center (on-prem)

Dining Table

Soda

Electric/Gas

Oven

Fire

Pizza Dough

Tomato Sauce

Toppings

Cheese

**Make at Home**

Infrastructure as a Service (IaaS)

Dining Table

Soda

Electric/Gas

Oven

Fire

Pizza Dough

Tomato Sauce

Toppings

Cheese

**Frozen Pizza**

Platform as a Service (PaaS)

Dining Table

Soda

Electric/Gas

Oven

Fire

Pizza Dough

Tomato Sauce

Toppings

Cheese

**Home Delivery**

Software as a Service (SaaS)

Dining Table

Soda

Electric/Gas

Oven

Fire

Pizza Dough

Tomato Sauce

Toppings

Cheese

**Eat Out**

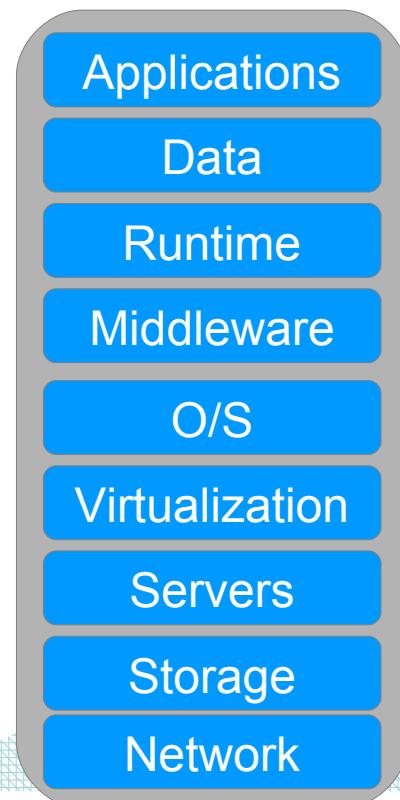
# PaaS: Platform as a Service

You Manage

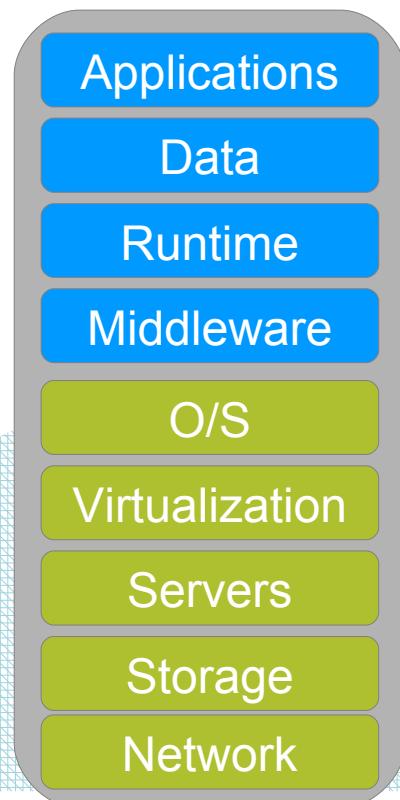
Provided

Agility  
And  
Cost  
Savings

*Traditional IT*



*IaaS*



*PaaS*



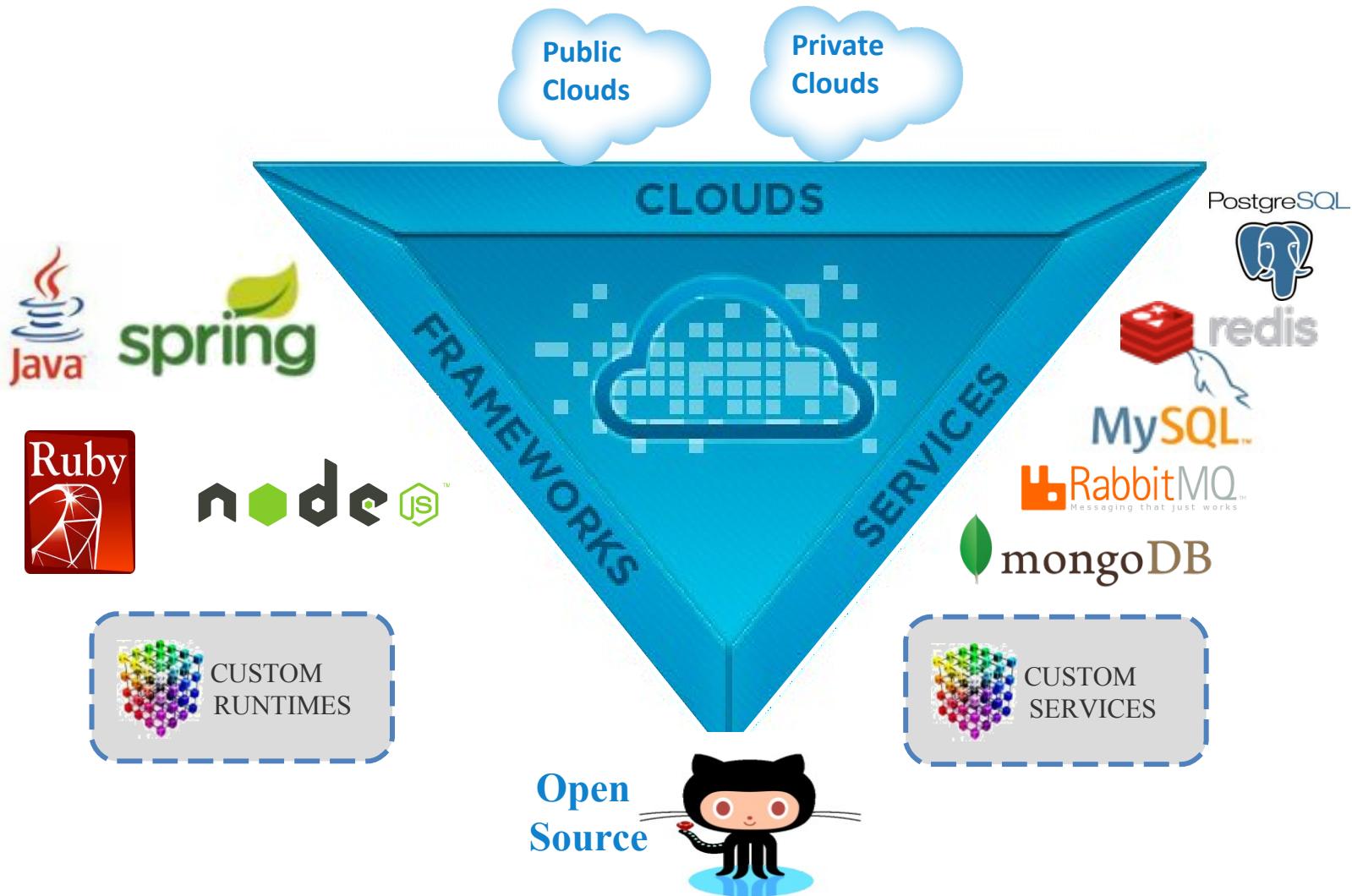
Cloud Enablement

Pivotal™

# What is Cloud Foundry?

- Introduction to Cloud
- **Cloud Foundry**
- Key Concepts
- Pivotal CF

# Cloud Foundry – The Open PaaS



# Why Cloud Foundry?

- Open Source
  - Reduce vendor-lock
- Public OR Private
- Language Independence
  - Via Buildpacks
- Wide, growing range of services



# Cloud Foundry: Foundation Open Governance

Platinum



Gold



Silver



Pivotal

# An Explanation of the Various Web Sites

- A web-search for “Cloud Foundry” can be confusing!

URL	Details
<a href="http://cloudfoundry.org">cloudfoundry.org</a>	<ul style="list-style-type: none"><li>• The open source project's home page</li><li>• No hosting of any kind here.</li><li>• Documentation</li></ul>
<a href="https://github.com/cloudfoundry">github.com/cloudfoundry</a>	<ul style="list-style-type: none"><li>• The location of the source</li><li>• Download, build, and run if you like!</li></ul>
<a href="http://cloudfoundry.com">cloudfoundry.com</a>	<ul style="list-style-type: none"><li>• Old commercial web-site</li><li>• Superseded by <a href="http://run.pivotal.io">run.pivotal.io</a></li><li>• Redirects to <a href="http://pivotal.io">pivotal.io</a> product page</li></ul>
<a href="http://blog.cloudfoundry.org">blog.cloudfoundry.org</a>	<ul style="list-style-type: none"><li>• Technical blog</li></ul>
<a href="http://run.pivotal.io">run.pivotal.io</a>	<ul style="list-style-type: none"><li>• Pivotal Web Services (PWS)</li><li>• Pivotal's hosted environment, runs PivotalCF</li></ul>

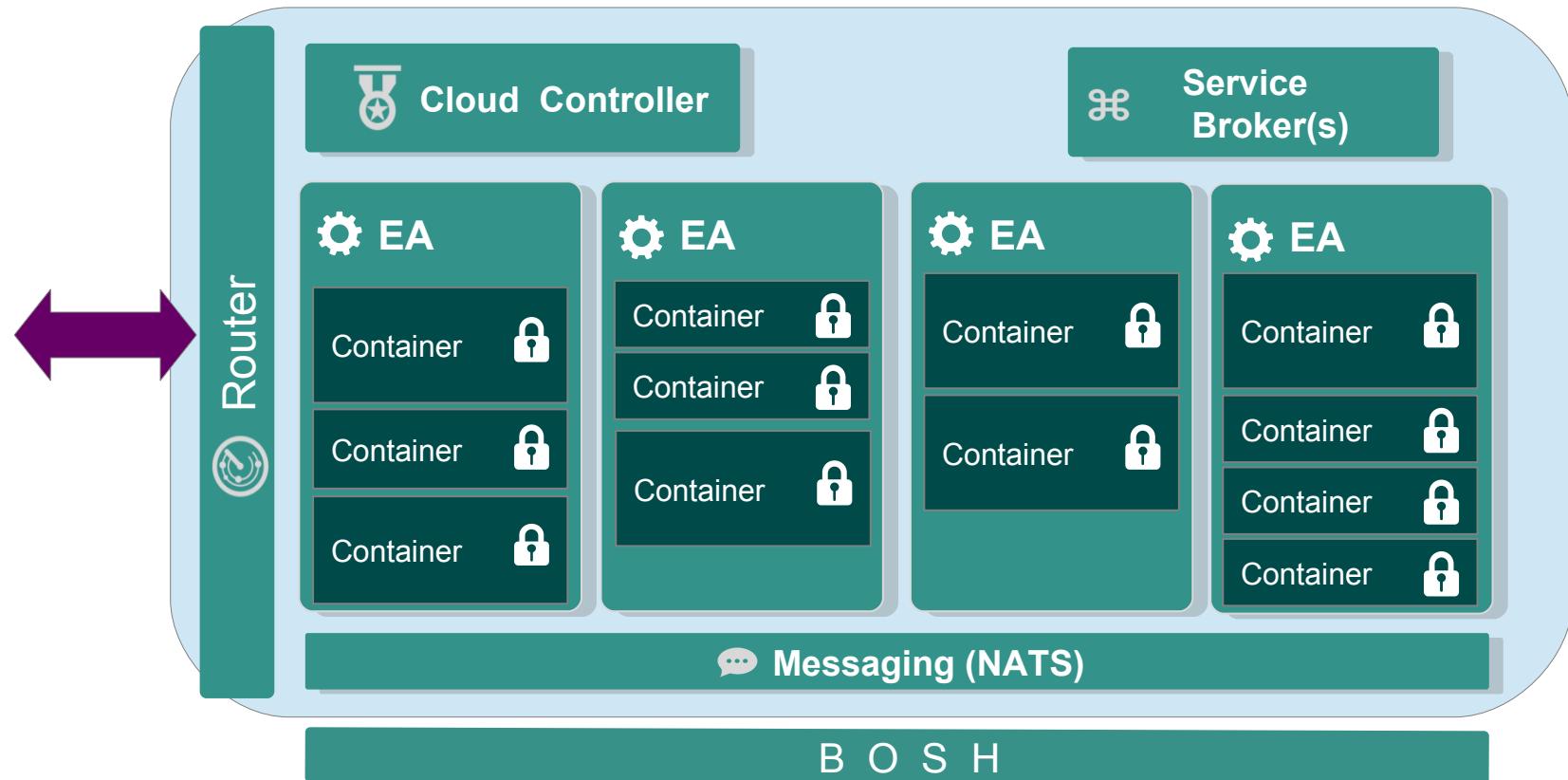
# A Major Cloud Foundry Site

- Largest Chinese Internet search site
  - One billion page views per day
  - Ad-Words facility powered by Cloud Foundry



# Cloud Foundry Architecture

## High-Level View – more details later



**EA** = Execution Agent  
= A VM where apps run

# DEA vs DIEGO

- Execution Agents
  - This course generically refers to Execution Agents
    - Virtual Machines (VMs) where applications run
    - Could be a DEA or a *Diego Cell*
  - Almost all course content applies to *either* implementation

Cloud Foundry Version	Execution Agent Type	Description
Since 1.0	Droplet Execution Agent or DEA	A VM running one or more “Droplets” - each Droplet is an application instance
Since 1.5 (Diego)	<i>Cell</i>	A VM running one or more Droplets, Docker images, ...

# What is Cloud Foundry?

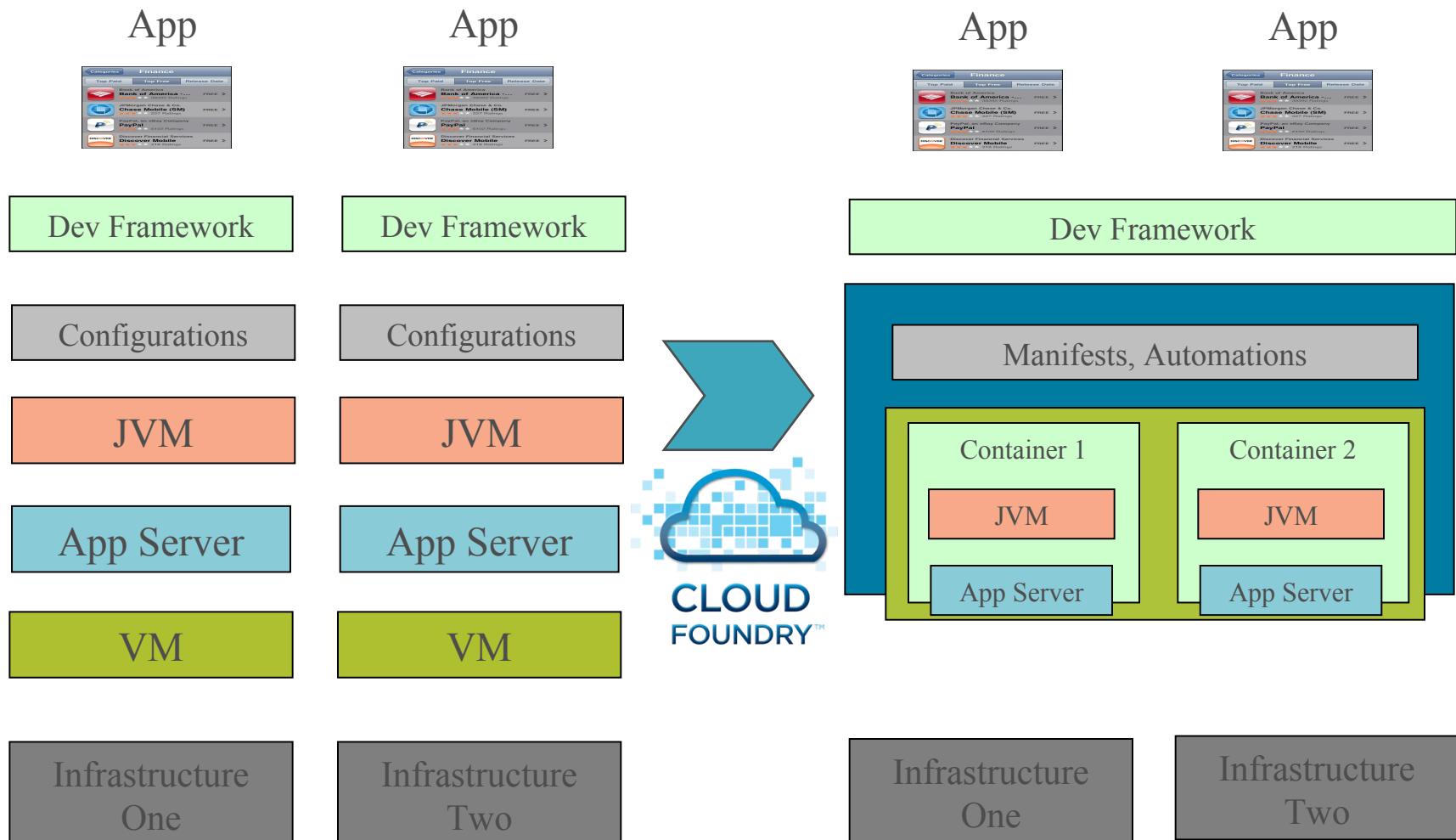
- Introduction to Cloud
- Cloud Foundry
- **Key Concepts**
- Pivotal CF

# Cloud Foundry Platform as a Service

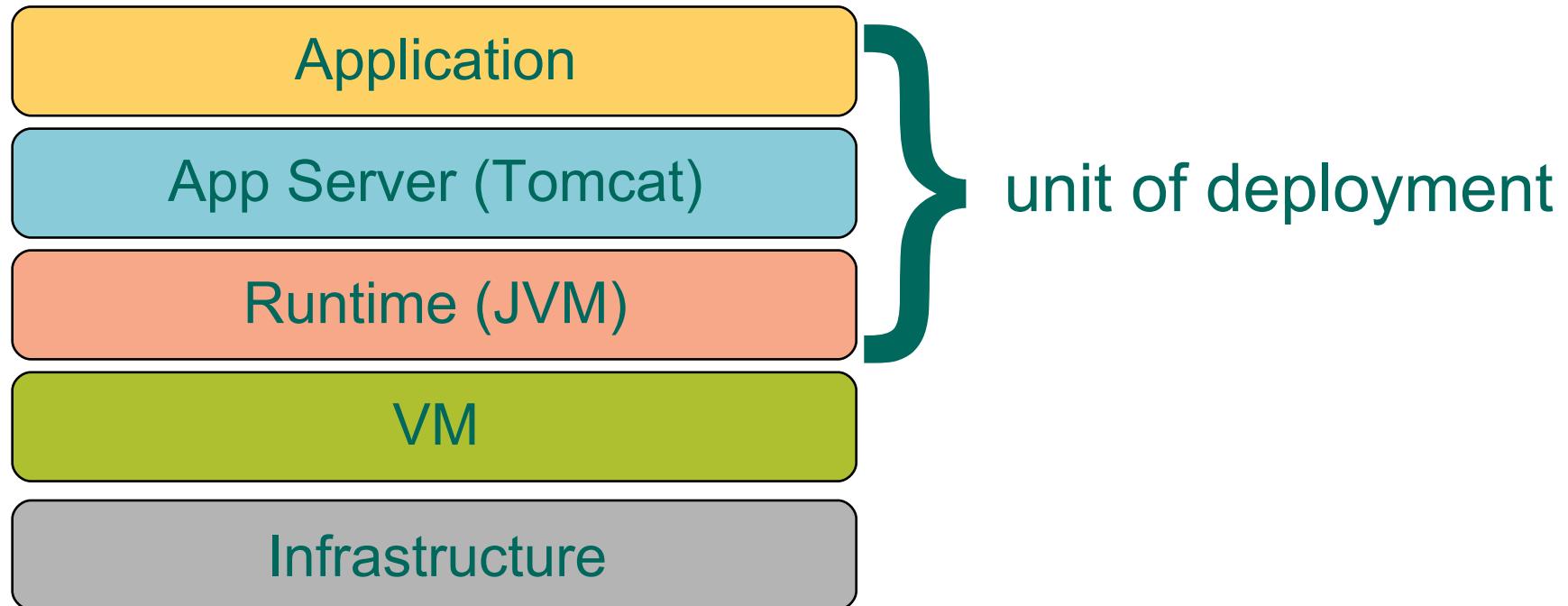
- **Application** is the new unit of deployment and control
  - Abstracting VMs and Middleware
  - Abstracting containers and processes
  - Data as a service
  - Eliminate bottleneck of provisioning & deployment



# From VM Centric to Application Centric

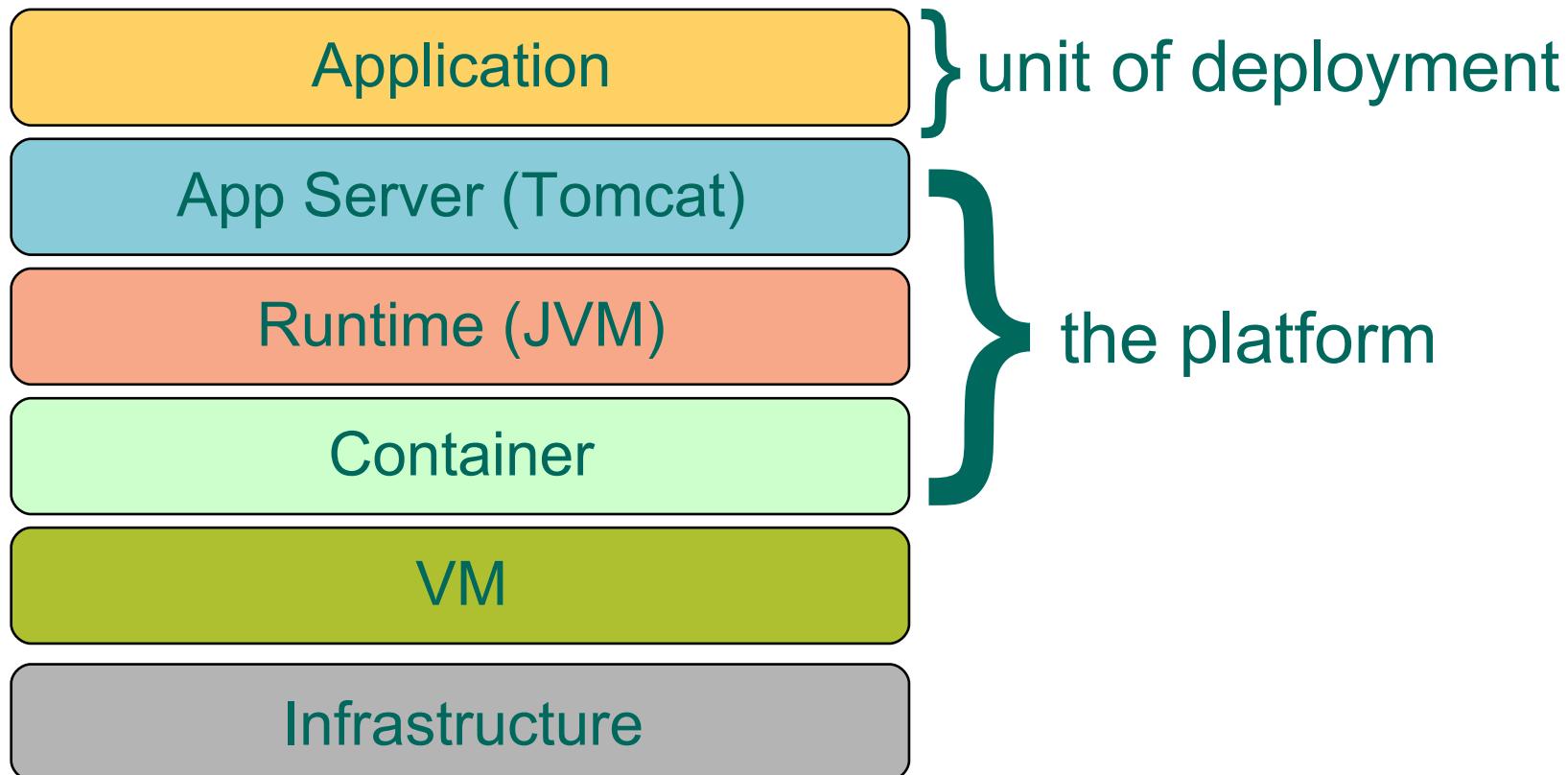


# IaaS: VM-centric Deployment



Scaling = creating VMs from blueprints/templates

# PaaS: App-centric Deployment



Scaling = creating containers in a VM pool

# Core Tenets of Pivotal CF

## Radically Simple, Developer Friendly

- Push an app – “it just works”
- Supporting wide ranging use cases
- Support new application patterns – microservices
- Easy to add/customize

## Broad Ecosystem of Services

Data Services

App Services

Mobile Services

## Operational Benefits for every Application

- Highly scalable
- Self healing
- Logging & audit trail
- Existing enterprise policies – user access & authorization
- Application monitoring
- Operational metrics
- App uptime SLAs

Deploy, Operate Update, Scale Platform on Any IaaS

Pivotal™

# What is Cloud Foundry?

- Introduction to Cloud
- Cloud Foundry
- Key Concepts
- **Pivotal CF**

# Which Cloud Foundry Product?



- Open Source
- Setup and run a PaaS for yourself
- No paid support
- No tools



## PWS

- Pivotal Web Services
- Public CF PaaS run/managed by Pivotal
- Hosted on AWS
- No guaranteed SLAs, not for production
- Support offered

## PWS-E

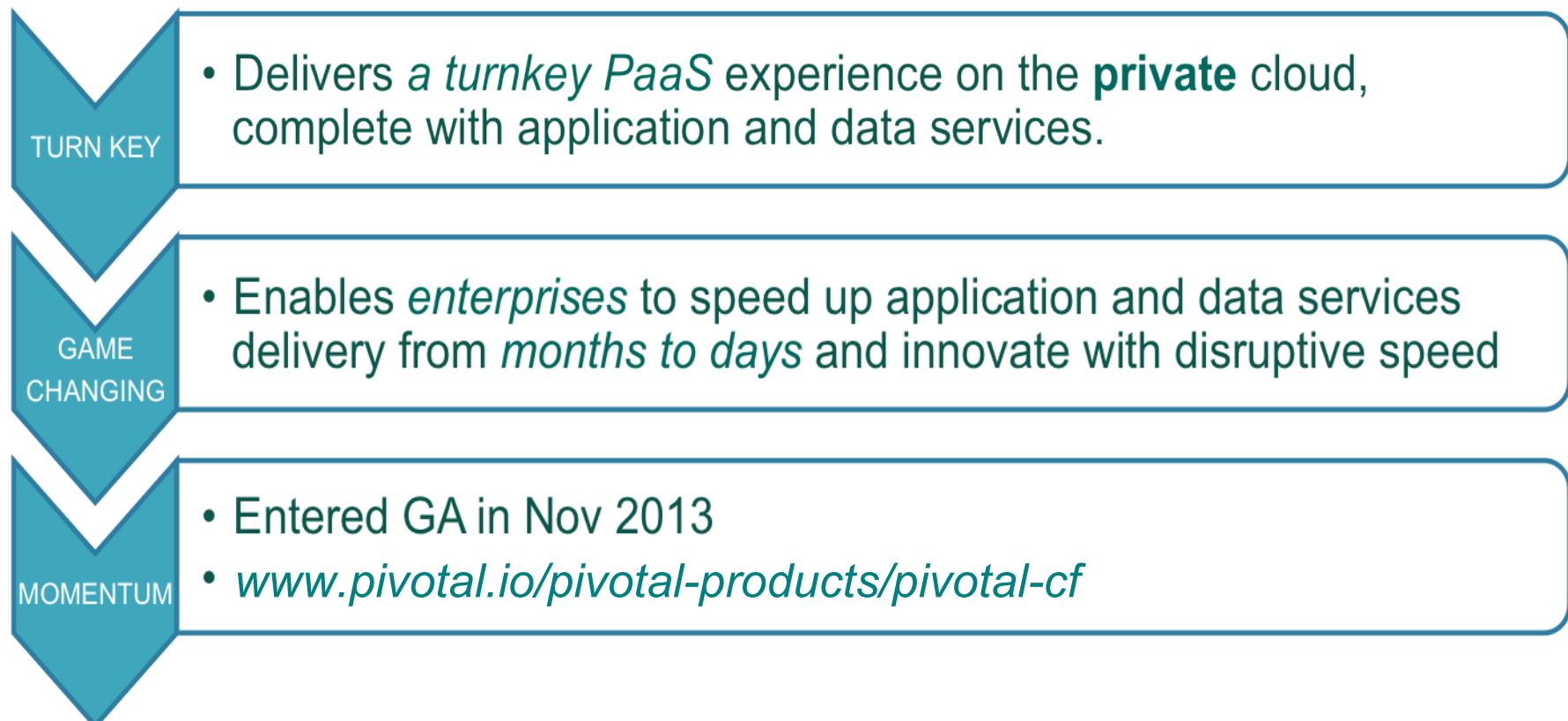
- PWS Enterprise
- For existing Pivotal customers to use PWS



- Commercial product
- Run private PaaS in-house, or
- Create public PaaS (ISP managed)
- Sophisticated Web Console (App Mgr)
- Additional Tools (Ops Mgr)
- Less hassle
- Support offered

# What is Pivotal CF?

- An Enterprise PaaS powered by Cloud Foundry



# Pivotal CF: Enterprise PaaS

## Commercially Supported

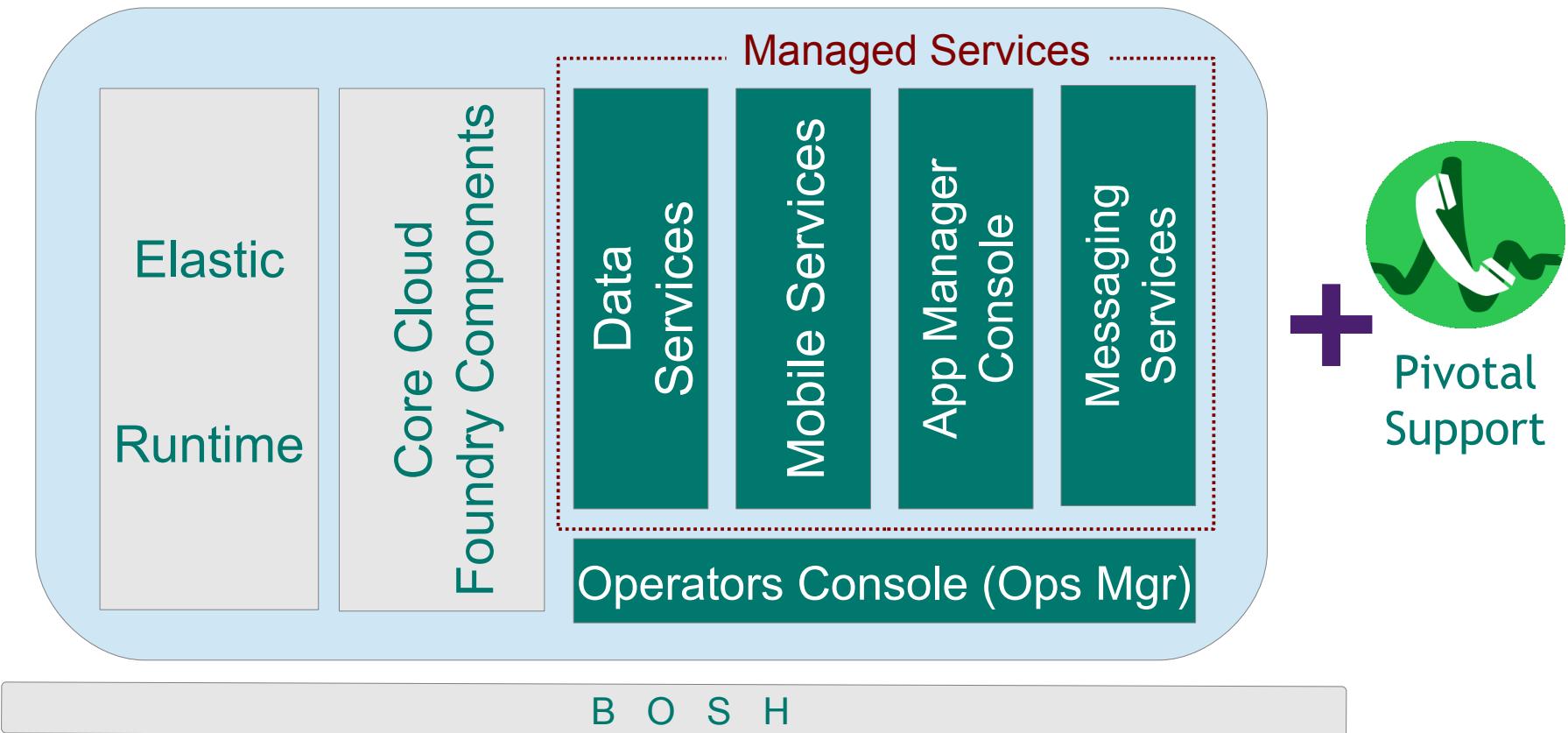
- Certified upgrades, curated & tested quarterly releases
- Operations Manager Cloud Administrative Console
- Elastic Runtime Application Manager \*
- Built-in Application Performance Monitoring (APM)
- Operational Metrics service
- Auto-Scaling service
- Easy LDAP/Active Directory Integration
- Historical Usage Data per Application Instance
- Declarative Availability Zones
- Behind the firewall download packages (do not require internet access)
- Enterprise documentation (PDF format)

\* Formerly Developer Console

OSS

PivotalCF

# OSS vs Pivotal CF



**Data Services:** RDB and NOSQL

**Messaging Services:** Rabbit/MQ

**Mobile Services:** Push Notifications, API Gateway, Data Sync, ...

Pivotal™

# Pivotal CF – Commercial Components

## *PCF Ops Manager*

- One Pivotal CF Operation Manager instance hosted across a cluster of VMs.
- Typical enterprise deployment contains 2-3 Pivotal CF Ops Manager instances (dev/test + production x2 for HA)

## *PCF Elastic Runtime Application Instances (AI)*

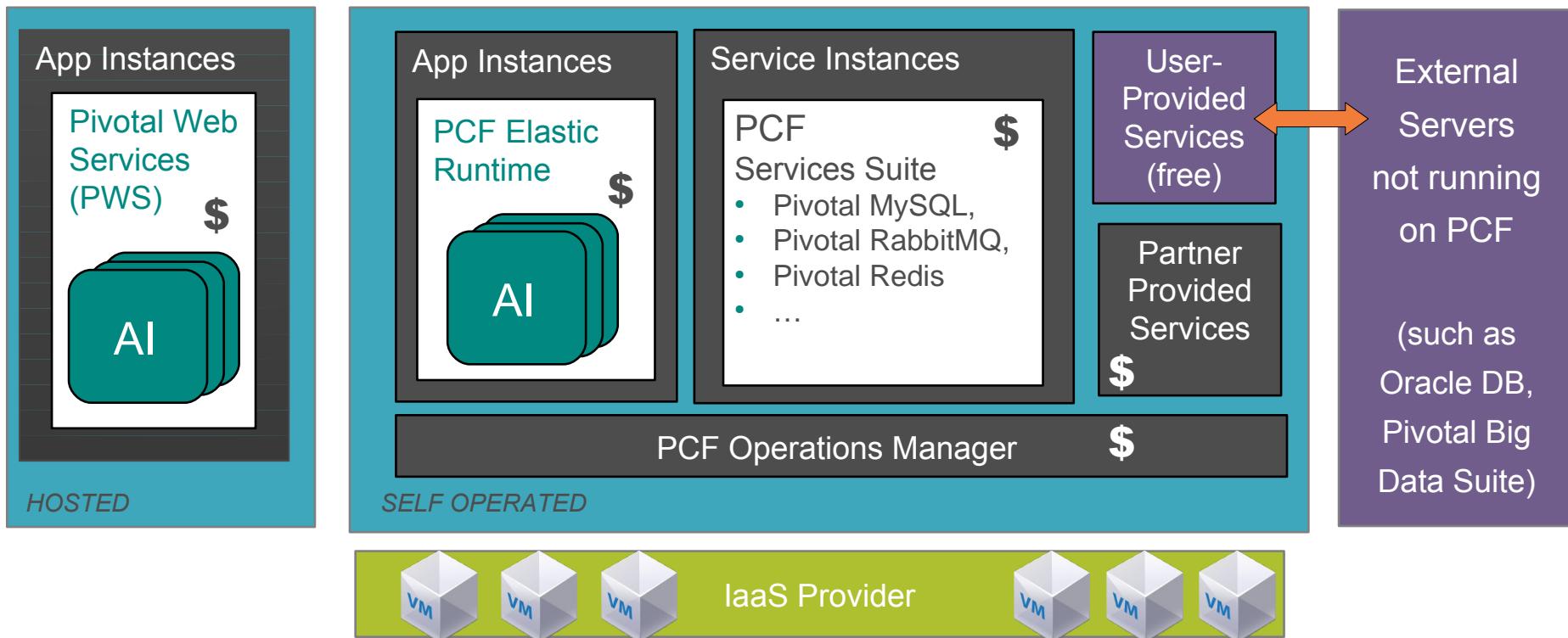
- Individual instances of deployed applications on PCF Elastic Runtime or on Pivotal Web Services (PWS)
- An application may scale up to multiple instances
- For a Java app: each AI is a Java Virtual Machine (JVM)

## *Service Instances (SI)*

- A single, unique configuration of a service (such as a database or other software or middleware) within a PCF Foundation (installation)
- Utilizes resources (such as CPU, cores, virtual machines, memory, messaging, development and/or data storage) within the same (or another) licensed Foundation

# What We Charge For

- Each App Instance, each Op Manager, each managed service



# Pivotal CF: Developer Benefits

- Application Manager
  - Start/stop
  - Scale
  - Services
  - Logs
  - etc.

The screenshot shows the Pivotal CF Application Manager interface. The top navigation bar displays the path: `sct-org > development > spring-music-sct`. The right side of the header shows the user's email: `struitt@gopivotal.com`.

The main content area is centered around the application `spring-music-sct`, which is highlighted with a large green circle. Below the circle, there are two small icons: a blue square and a white square with a blue border.

**INSTANCES** section:

INSTANCES	MEMORY LIMIT	DISK LIMIT
2	256 MB	1 GB

**SCALE APP** button: `SCALE APP`

**USAGE** section:

#	STATUS	CPU	MEMORY	DISK	UPTIME
1	Running	88%	98.0	.39	5d 23hr 59min
2	Running	85%	94.4	.39	5d 23hr 59min

**ABOUT** section:

BUILDPACK	liberty_buildpack
STACK	Ubuntu 10.04
ROUTE	spring-music-sct.example.com

**Activity** tab is selected. Other tabs include **Service Instances** and **Resources**.

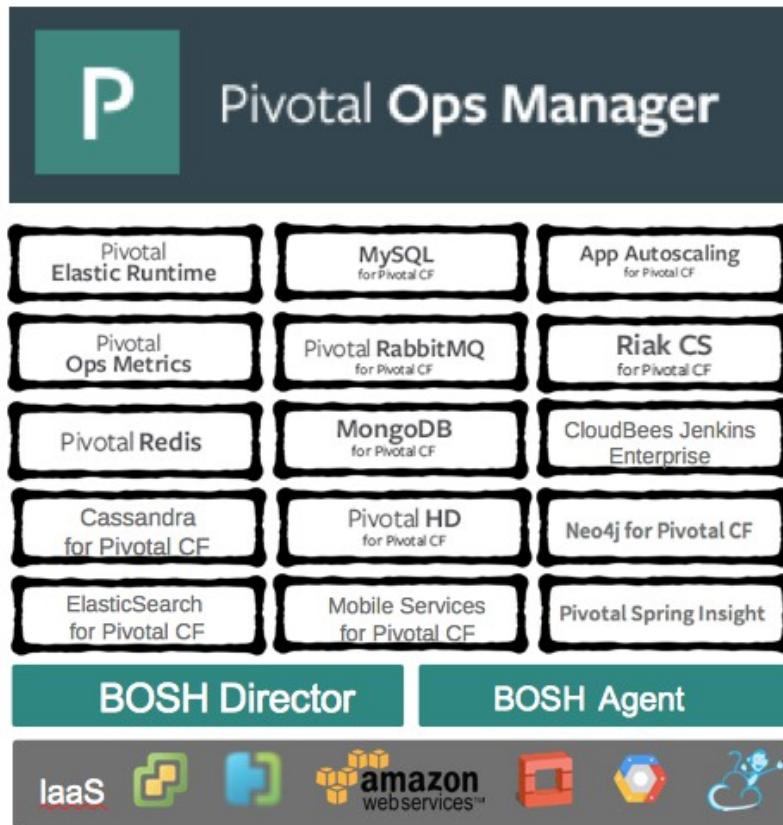
**EVENTS** section:

- `scaled instances (1 to 2)`  
struitt@gopivotal.com  
2014/02/21 at 14:29:21.31-800
- `started app`

**RECENT LOGS** section:

```
2014-02-21T14:29:21.55-0800 [App/0] OUT at org.apache.catalina.loader.WebappClassLoader.loadClassInternal (WebappClassLoader.java:2932)
2014-02-21T14:29:22.31-0800 [App/0] OUT 22:29:21,304 INFO ContextLoader:272 - RootWebApplicationContext: initialization started
```

# Pivotal CF: Operator Benefits



- Click to Install
- No downtime updates
- Explore install logs
- Click to scale platform
- Built-in high availability
- Built-in platform monitoring
- Integrated services

# Pivotal CF: Operators View



Install platform



Target multiple clouds



Handle live upgrades/updates



Setup High Availability



Scale and plan capacity

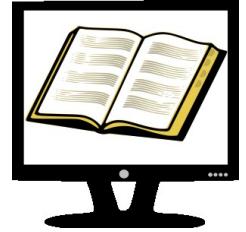


Install and manage services

The central graphic features a dark teal rounded rectangle containing the "Pivotal CF" logo at the top left. Below it is the "CLOUD FOUNDRY" logo, which includes a stylized white cloud icon composed of a grid of squares and the words "CLOUD FOUNDRY™". To the right of these are six circular icons, each containing a white outline of a different service or tool: a network of circles labeled "CF", a dolphin labeled "CF", a leaf labeled "CF", a smiling face labeled "CF", a starburst labeled "CF", and a large letter "H" labeled "CF". Above these circular icons is a dark teal box containing a white "P" icon and the text "Pivotal Ops Manager". At the bottom of the central graphic is a dark grey horizontal bar containing several logos: "IaaS" (green square with yellow plus), "Amazon web services™" (orange cube icon), "Google Cloud Platform" (red square icon), and "Microsoft Azure" (blue cloud icon with a cartoon monkey).

# How can I use Pivotal CF?

- Pivotal CF is the commercial PaaS based on CF
- Many options:
  - Public – Pivotal Web Services: [run.pivotal.io](http://run.pivotal.io)
  - Basis of other public PaaS offerings
  - Run it in-house as the basis of your private cloud
  - Run it yourself on third-party cloud provider
    - Open Stack, AWS ...



# Documentation

- CF docs can be found on three different sites, depending on the context:
  - <http://docs.cloudfoundry.org>
    - Open-source CF project docs
  - <http://docs.pivotal.io/pivotalcf>
    - Setting up and running Pivotal CF
  - <http://docs.run.pivotal.io>
    - Information about running on Pivotal Web Services
- Generic information about Cloud Foundry can be found on all three sites
  - In general pick one and stay with it

# Pivotal Support

- Support is available for Pivotal Cloud Foundry
  - Options: <http://pivotal.io/support/offers>
  - Register: <http://tinyurl.com/piv-support>
  - Support Portal: <https://support.pivotal.io>
    - Community forums, Knowledge Base, Product documents
  - PWS Support:
    - <https://support.run.pivotal.io> or [support@run.pivotal.io](mailto:support@run.pivotal.io)
- Community forums
  - <https://groups.google.com/a/cloudfoundry.org/forum/#!forumsearch/>



# Summary

- Cloud Foundry is “the Open PaaS”
- Key concepts: Organizations, Spaces, Services
  - The *Application* is the unit of deployment.
  - Simpler for Developers
- Pivotal CF is Pivotal's Cloud Foundry distribution
  - Supported, easier to install / manage
- Various documentation resources available



# Lab

## Explore Cloud Foundry & Pivotal CF documentation



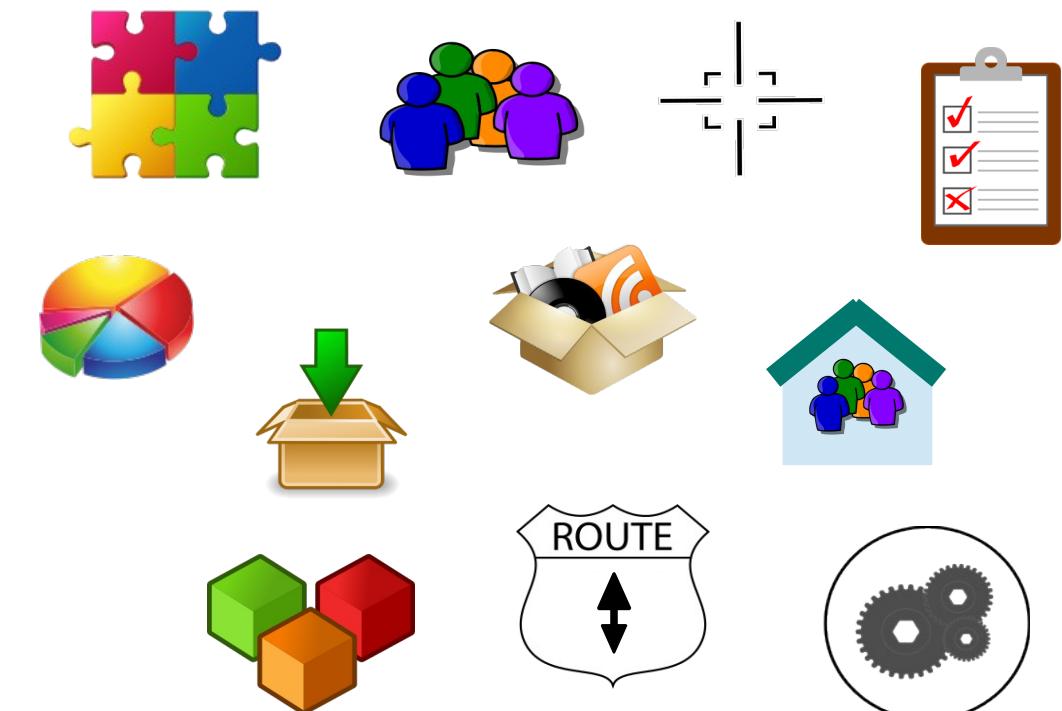
# Cloud Foundry Concepts

## Terms We Use

Organization, Space, Role, Route, Application

# Overview

- After completing this lesson, you should understand:
  - Applications
  - Buildpacks
  - Manifests
  - Organizations
  - Spaces
  - Users and Roles
  - Quotas
  - Domains
  - Routes
  - Services



# Roadmap

- **Applications, Buildpacks, Manifests**
- Organizations, Spaces, Users, and Roles
- Domains and Routes
- Services



# Applications

- PaaS exists to deploy *applications*
  - In Cloud Foundry, the application is the unit of deployment
  - Developers focus on apps, not runtimes or services
- Cloud Foundry is development *agnostic*
  - Not limited to specific language
  - Doesn't mandate the runtime environment you get
  - Some are “out-of-the-box”
  - You can add more

# Buildpacks and Manifests



- Buildpacks
  - Allow CF to support multiple languages and deployment environments
    - Buildpacks for Java, Ruby, JavaScript ...
    - Buildpacks for Tomcat, Rails, Node.JS ...
- Manifests
  - A deployment “blueprint” for an application
  - *Repeatable*: redeploy using same manifest

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

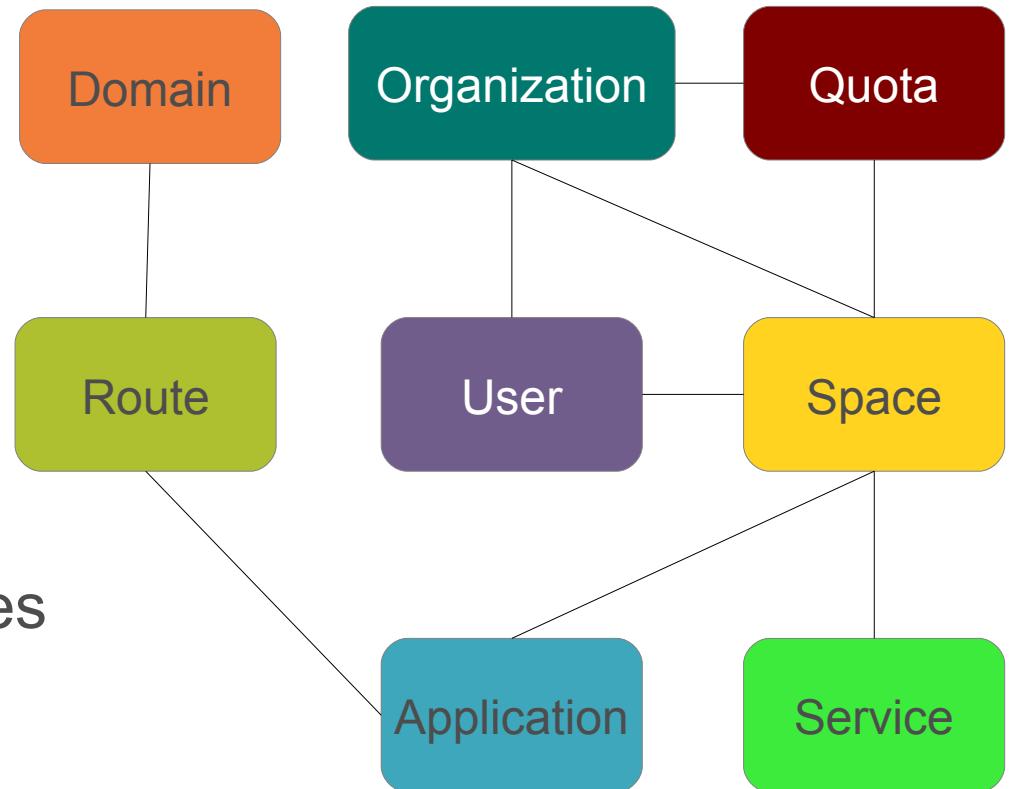
# Roadmap

- Applications, Buildpacks, Manifests
- **Organizations, Spaces, Users, and Roles**
- Domains and Routes
- Services



# Organizations

- Top-most administrative unit
- Contains spaces and users
  - Which own routes, applications and services
- Quotas restrict resources
  - For orgs and spaces
- Domain(s)
  - Define routes to apps

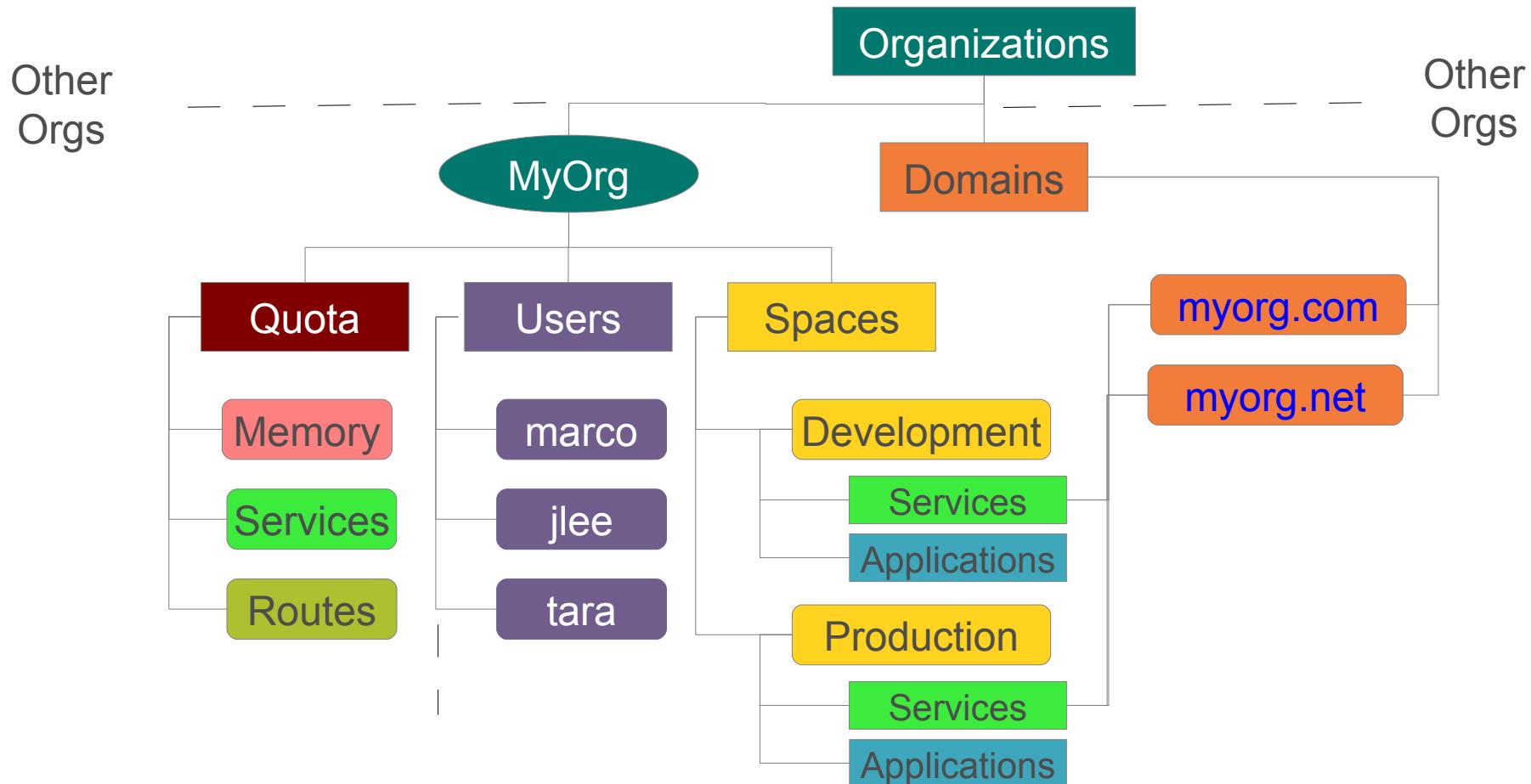


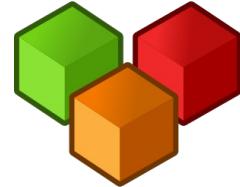


# Organizations

- CF is designed for use by Organizations
  - Typically a company, department, application suite or large project
- Designed to support collaboration
  - Potentially many users
- Defines one or more domains
  - Cloud Foundry instance defines default domain for all organizations
    - For PWS: [cfapps.io](http://cfapps.io)
    - You may add additional domains
- Defines Security, Quotas

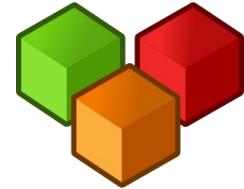
# Example Organization





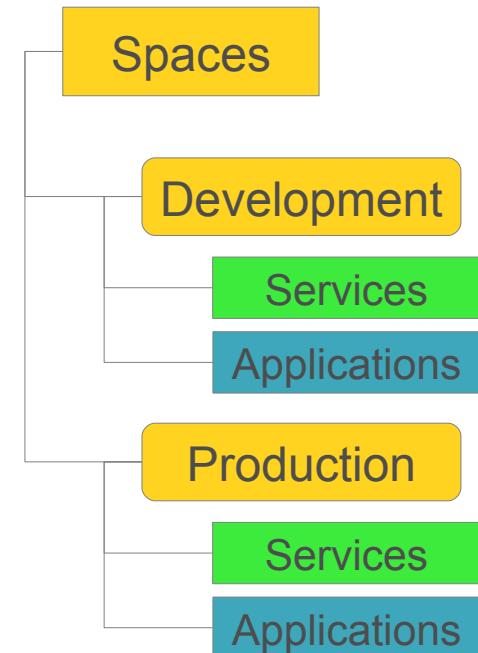
# Spaces

- Organization contains multiple spaces
  - Default PWS space: *development*
  - Users can create additional spaces
- Applications and services scoped to a space
  - Many applications can run/scale within a space
- Provides set of users access to a shared location
  - Application development
  - Functionality and/or Performance Testing
  - Quality Assurance
  - Deployment to production
  - Maintenance



# Spaces

- Organization contains multiple spaces
  - Default PWS space: *development*
  - Users can create additional spaces
- Provides set of users access to shared location
  - Application development
  - Functionality and/or Performance Testing
  - Quality Assurance
  - Deployment to production
  - Maintenance
- Applications and services scoped to a space
  - Many applications can run/scale within a space





# Users and Roles

- Members of an organization
  - You can invite users to share your “cloud”
- Have specific roles
  - Roles control access to domains and spaces
  - Roles therefore control who has permission
    - To manage routes (see later slides)
    - To deploy applications
    - To add/bind/remove services
- ***Don't*** need to be CF User to access deployed apps
  - Each application does its ***own*** user-management



# Organization Roles

- Organization Manager
  - Can invite/manage users, select/change the plan, establish spending limits
- Organization Auditor
  - View only access to all org and space info, settings, reports



# Application Space Roles

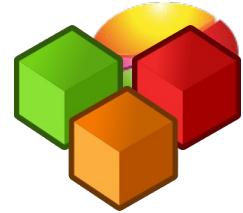
- Space Manager
  - Can invite/manage users, enable features for a given space
- Space Developer
  - Can create, delete, manage applications and services, full access to all usage reports and logs
- Space Auditor
  - View only access to all space information, settings, reports, logs



# Administrator User / Roles

- Special Administrator user / role defined
  - Defined for the Cloud Foundry installation
  - Separate from users defined at Organization / Space
- Several **cf** commands restricted to Administrator only
  - Setting organization and space quotas
  - Defining security groups
  - Administering services
  - Adding, modifying and removing user accounts
- Use without the right role, CLI returns:

```
Server error, status code: 403, error code: 10003, message: You are not  
authorized to perform the requested action
```

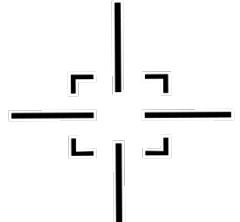


# Quotas

- Restrictions on available resources
  - Total memory available to all applications
  - Total number of routes
  - Max application instance size
  - Total number of services
- Can view
  - Using the CLI
  - Using App Manager (on org homepage)

# Roadmap

- Applications, Buildpacks, Manifests
- Organizations, Spaces, Users, and Roles
- **Domains and Routes**
- Services



# Domains

- Deployed applications are associated with a URL
  - All requests to that URL redirect to the application
- Each Cloud Foundry instance has a default app domain
  - PWS has [cfapps.io](#)
- Custom Domains
  - Register your own domain
  - Give it to Cloud Foundry to use and manage
- Subdomains
  - Each application has a *unique* sub-domain
    - Its URL is therefore *sub-domain.domain*
    - Example: deploy an app to <http://myapp.cfapps.io>

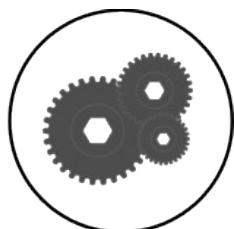


# Routes

- Define how to get to an application
  - A unique *route* exists to each *application* in every *space*
  - Behind the scenes CF uses a router
    - maps incoming requests to the right application
- Domain can be mapped to *multiple* spaces
  - Route can *only* be mapped to *one* space
  - Same application *can* be deployed in *multiple* spaces
    - Each must have a *different, unique* URL
    - Development space route: <http://myapp-test.cfapps.io>
    - Production space route: <http://myapp.cfapps.io>

# Roadmap

- Applications, Buildpacks, Manifests
- Organizations, Spaces, Users, and Roles
- Domains and Routes
- **Services**

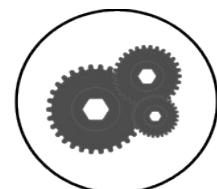
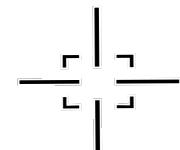
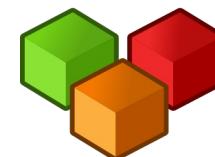


# Services

- Any type of add-on that can be provisioned along side your apps.
  - database, messaging, mail, third-party SaaS provider
- Services are usually “bound” to 1 or more applications
  - Connection info and credentials are put in an environment variable: **VCAP\_SERVICES**
  - **Note:**
    - All configuration data to CF applications should be passed via environment variables
    - Can't use configuration files: *no file system*

# Summary

- After completing this lesson, you should have learned about:
  - Applications, Buildpacks, Manifests
  - Organizations, Users, Roles ,Quotas
  - Domains, Spaces, Routes
  - Services





# Getting Started with Cloud Foundry

Deploying your First Application

Setup, Deploy and Manage

Pivotal

# Overview

- After completing this lesson, you should be able to:
  - Deploy an application to CloudFoundry using CLI
  - Manage application instances using online Dashboard

# Roadmap

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

# The Command Line Interface

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

# Test the CLI Utility

- Version *must* be **6** or more
- *Must* remove any earlier (Ruby) version

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

The image contains two screenshots of command-line interfaces. The top screenshot is from a Mac OS X terminal window titled "paulchapman — bash — 87x10". It shows the command "/usr/local/bin(cf --version)" being run, with the output "/usr/local/bin(cf version 6.0.2-0bba99f" displayed. The bottom screenshot is from a Windows Command Prompt window titled "Administrator: Command Prompt". It shows the command "cf --version" being run, with the output "cf version 6.0.2-0bba99f" displayed.

```
paulchapman — bash — 87x10
Last login: Thu Mar 20 22:01:51 on ttys005
localhost:~ paulchapman$ /usr/local/bin(cf --version
/usr/local/bin(cf version 6.0.2-0bba99f
localhost:~ paulchapman$ 
```

```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\chapmanp>cf --version
cf version 6.0.2-0bba99f

C:\Users\chapmanp>
```

# Getting Help

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

# DO NOW – Get Help on a Command

- Perform these steps on your computer:
  - Open a command prompt
  - Issue the **cf help** command
  - Get help on the login command: **cf help login**
- Answer these questions:
  - What option do you use to specify username?
  - Is specifying the password option encouraged?

# Roadmap

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

# Login to Cloud Foundry

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

Color highlighting  
MacOS, Linux only

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

Targeted org Cloud Foundry Course
Targeted space development

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

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

# Cloud Foundry URLs

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

# Cloud Foundry URLs

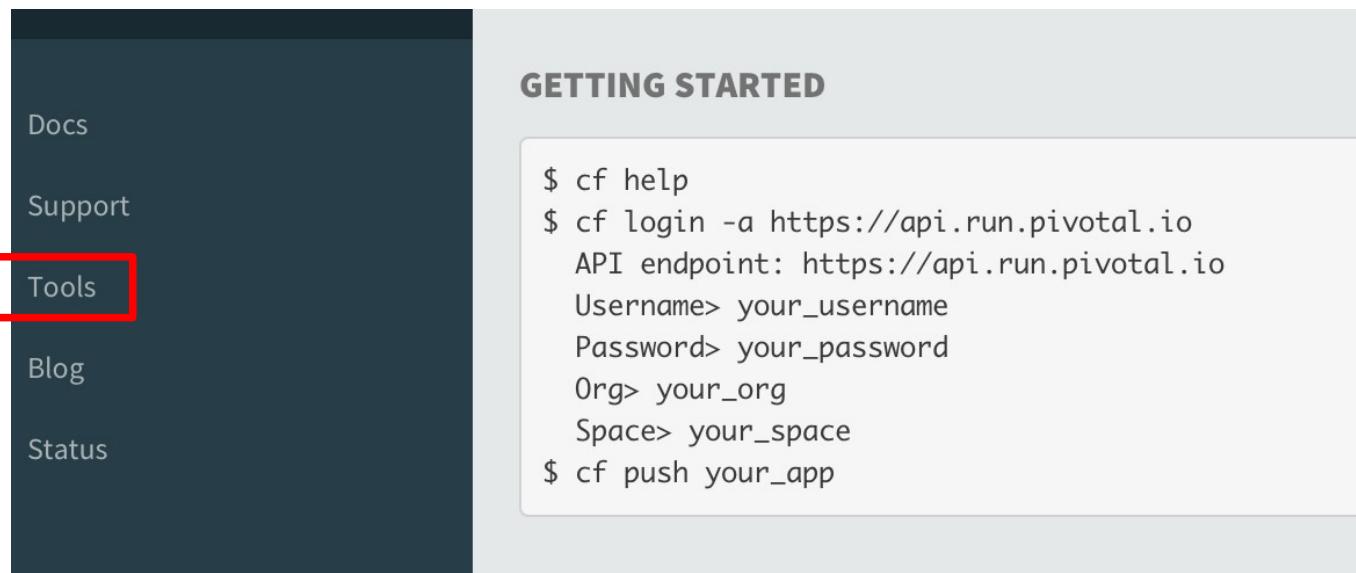
For simplicity, most examples in this section show PWS URLs

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



# Finding the API Endpoint URL

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



# DO NOW – Login

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

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

```
...
```

- Firewall issues?  
<http://docs.cloudfoundry.org/devguide/installcf/http-proxy.html>

# The .cf Directory

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

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

# DO NOW – .cf folder

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

# Current Targets

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

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

- By default your organization only has one space
  - Development
- **Note:** On PWS you are setup as your own organization



# Viewing Organization

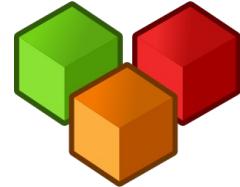
- Commands

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

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

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

>$
```



# Managing Spaces

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

# Roadmap

- Get Setup
- Login
- **Deploying an Application**
- Managing Application Instances

# Deploy Using the CLI

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

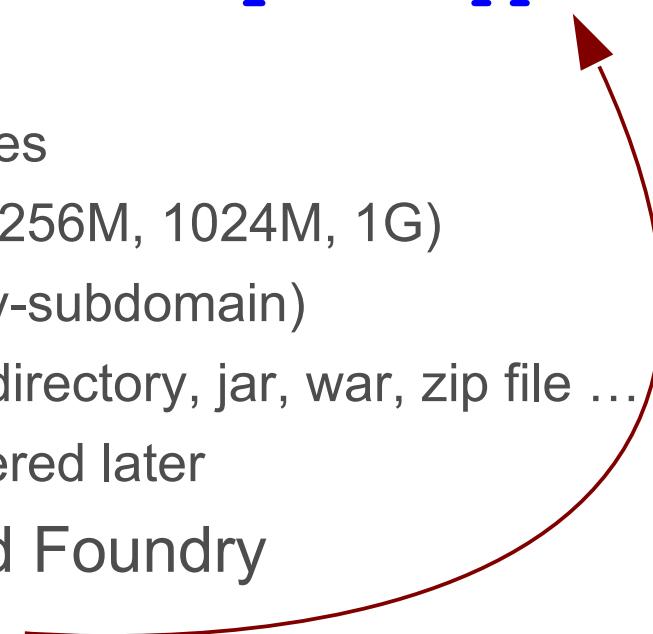
# The cf push Philosophy

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



*Haiku*

# Deploy (*push*) to Cloud Foundry

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

# Domains and URLs

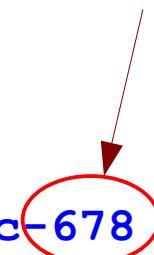
- Every CF instance is assigned a domain at installation
  - Known as the *Apps Domain*
  - For PWS this is ***cfapps.io***
- When you deploy, your application gets a unique route (URL) to access it: **hostname + app domain name**
  - By default, hostname = application name
  - Make sure hostname is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

# Examples of Using cf push

- Fully specified (recommended)

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

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



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

# What Happens ?

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

# What Happens - 1

URL: `spring-music-678.cfapps.io`

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

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

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

Updates CF metadata  
(app name, instances, memory)

```
Using route spring-music-678.cfapps.io
```

```
Uploading spring-music...
```

```
Uploading app files from: pre-built/spring-music.war
```

```
Uploading 574.8K, 95 files
```

```
Done uploading
```

```
OK
```

Establish route

Uploads war

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

```
...
```

Next...

# What Happens - “Staging”

CF must prepare the app before its first run

```
...  
Starting app spring-music in org your-org / space development as your-id@company.io...  
OK  
  
-----> Downloaded app package (21M) ← "Buildpack" selected and executed  
-----> Java Buildpack Version: v2.7.1 | https://github.com/cloudfoundry/java-buildpack#fee275a  
-----> Downloading Open Jdk JRE 1.8.0_40 from https://download.run.pivotal.io/openjdk/lucid/x86_64/openjdk-1.8.0_40.tar.gz (6.1s)  
      Expanding Open Jdk JRE to .java-buildpack/open_jdk_jre (1.3s)  
-----> Downloading Spring Auto Reconfiguration 1.7.0_RELEASE from https://download.run.pivotal.io/auto-reconfiguration/auto-reconfiguration-1.7.0_RELEASE.jar (0.2s)  
-----> Downloading Tomcat Instance 8.0.20 from https://download.run.pivotal.io/tomcat/tomcat-8.0.20.tar.gz (1.1s)  
      Expanding Tomcat to .java-buildpack/tomcat (0.1s)  
-----> Downloading Tomcat Lifecycle Support 2.4.0_RELEASE from https://download.run.pivotal.io/tomcat-lifecycle-support/tomcat-lifecycle-support-2.4.0_RELEASE.jar (0.0s)  
-----> Uploading droplet (73M)  
...  
Next...  
Buildpack creates “Droplet”  
Buildpack obtains Tomcat  
Buildpack configures Java  
Reconfigure Spring for cloud environment
```

# What Happens - start

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

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

App started

OK

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

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

OK

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

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

Health Check

Done! 1 application instance running on [spring-music-678.cfapps.io](http://spring-music-678.cfapps.io)

# Application State and Logs

- Run `cf apps`

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

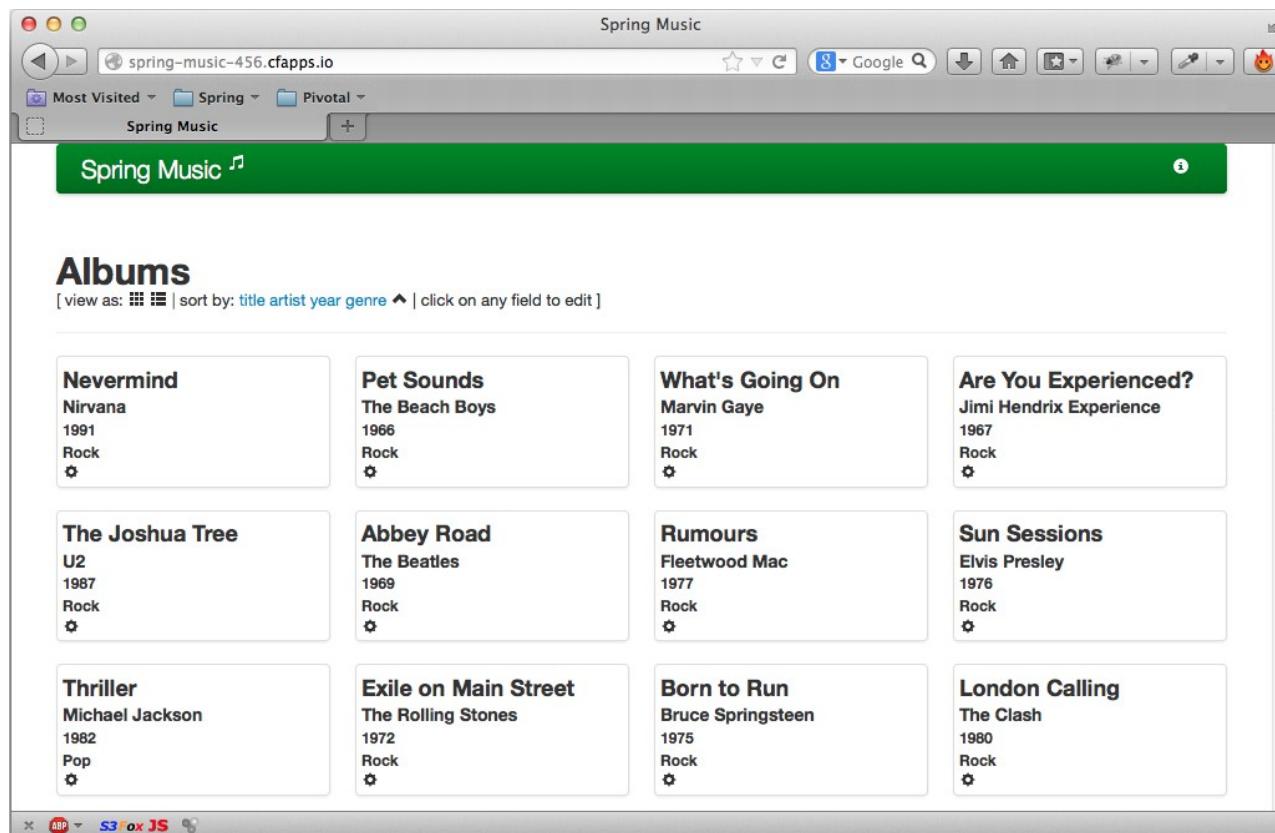
name      requested state    instances   memory   disk    urls
spring-music  started        1/1        512M     1G      spring-music-678.cfapps.io
```

- `cf logs spring-music`

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

# See The Application Running

- Open a browser window to [spring-music-678.cfapps.io](http://spring-music-678.cfapps.io)



# Configuring a Deployed Application

- Change the number of instances
  - `cf scale <app> -i <new-value>`
  - Two instances: `cf scale spring-music -i 2`
  - New instances added, or some existing instances stopped
- Change the memory allocation
  - `cf scale <app> -m <new-value>`
  - 1024M: `cf scale spring-music -m 1024M`
  - Requires a restart to take effect

# Stopping and Starting

## `cf stop`

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

## `cf start`

- Starts existing application

## `cf restart`

- `cf stop` followed by `cf start`

## `cf restage`

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



# Adding / Removing Routes

- Add a new domain mapping
  - `cf map-route <app> <domain> -n <hostname>`
  - `cf map-route spring-music cfapps.io -n mymusic`
    - `mymusic.cfapps.io` also maps to spring-music
- Remove mapping
  - `cf unmap-route <app> <domain> -n <hostname>`
  - `cf unmap-route spring-music cfapps.io -n spring-music-678`
    - `spring-music-678.cfapps.io` no longer maps to spring-music

# Cleaning Up Unused Routes



- Routes tend to accumulate over time
  - Applications in other Orgs / Spaces cannot use these routes
- Find all other routes used in a space:
  - `cf routes`
- Remove route:
  - `cf delete-route`
- Very Useful! Remove unused routes:
  - `cf delete-orphaned-routes`

# Roadmap

- Get Setup
- Login
- Deploying an Application
- **Managing Application Instances**

# Apps Manager

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

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

# Apps Manager Home Page

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

The screenshot shows the Pivotal Apps Manager interface. On the left, there's a sidebar with a 'P' logo and 'Pivotal Web Services'. It has sections for 'ORG' (pivotaledu) and 'SPACES' (development, production, staging, Marketplace). A red arrow points from the text 'Click to select different space' to the 'Marketplace' button in the sidebar. The main area shows the 'development' space with a title 'SPACE development'. Below it is a table titled 'APPLICATIONS' with columns for STATUS, APP, INSTANCES, and MEMORY. It lists two applications: 'classfeedback-dev' (STOPPED, 1 instance, 1GB memory) and 'spring-music' (100% healthy, 1 instance, 512MB memory). A red arrow points from the text 'URL of your application' to the URL part of the 'spring-music' row ('spring-music-678.cfapp...'). Another red arrow points from the text 'Click application name to see its dashboard (next slide)' to the 'development' space title.

APPLICATIONS			
STATUS	APP	INSTANCES	MEMORY
STOPPED	classfeedback-dev classfeedback-dev.cfapp...	1	1GB
100%	spring-music spring-music-678.cfapp...	1	512MB

# Application Dashboard

The screenshot shows the Pivotal Application Service (PaaS) Application Dashboard for an application named "smkk". The dashboard is divided into several sections:

- APP**: Shows the application icon (blue square with white circle) and status (green circle with "C" indicating it's running).
- ABOUT**: Displays buildpack information (java-buildpack v2.1.2), stack (Ubuntu 10.04), and route (smkk.cfapps.io).
- CONFIGURATION**: Allows changing the number of instances (1) and memory limit (512 MB). It also includes disk limit (1024 MB) and a "Scale App" button.
- STATUS**: Shows instance statistics for one instance: Status (Running), CPU (0%), Memory (370 MB), Disk (122 MB), and Uptime (1 min).
- Activity**: A tabbed section showing recent activity. The "Events" tab is selected, displaying a log entry for "started app" by kkrueger@gopivotal.com on 08/23/2014 at 06:09 PM UTC.
- Services**: Shows no services listed.
- Env Variables**: Shows no environment variables listed.
- Routes**: Shows no routes listed.
- RECENT LOGS**: Displays a log history with entries from 2014-08-23:

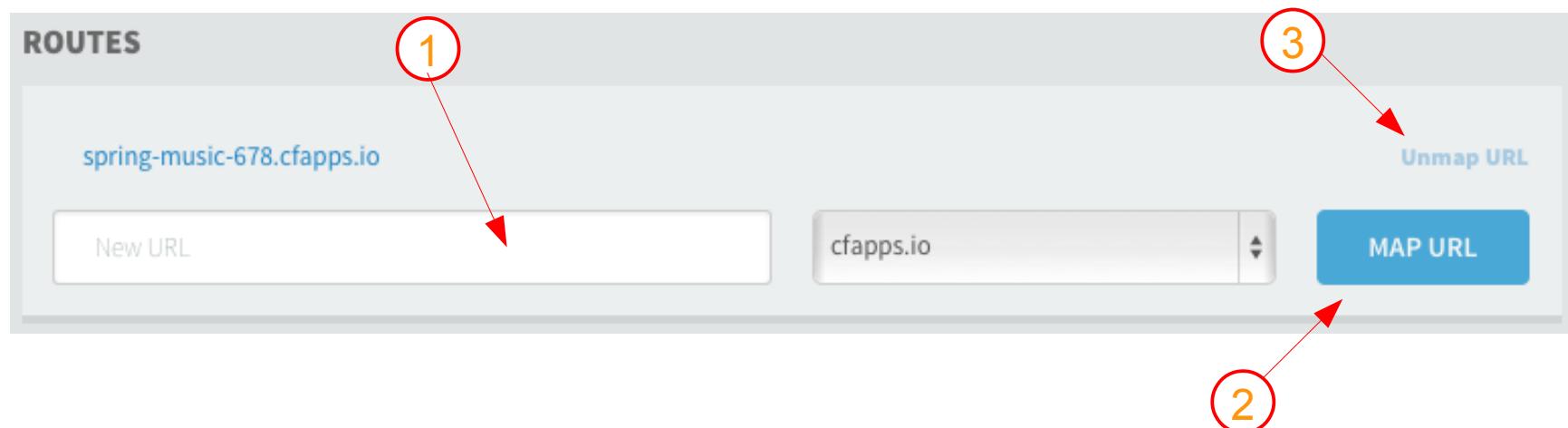
  - 2014-08-23 18:10:20 +0000 [App/0] OUT [CONTAINER] org.apache.coyote.http11.Http11Protocol INFO Starting ProtocolHandler ["http-bio-62799"]
  - 2014-08-23 18:10:19 +0000 [App/0] OUT 18:10:19,725 INFO SimpleUrlHandlerMapping:315 - Mapped URL path [/\*\*] onto handler of type [class org.springframework.web.servlet.resource.DefaultServletHttpRequest
  - 2014-08-23 18:10:19 +0000 [App/0] OUT 18:10:19,719 INFO SimpleUrlHandlerMapping:315 - Mapped URL path [/assets/\*\*] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandle

Annotations with red arrows point to specific sections:

- An arrow points from the "Change instances, memory" annotation to the "INSTANCES" and "MEMORY LIMIT" fields in the Configuration section.
- An arrow points from the "Application state" annotation to the "Status" section.
- An arrow points from the "Instance Statistics" annotation to the table in the Status section.
- An arrow points from the "Logs and Events" annotation to the "RECENT LOGS" section.

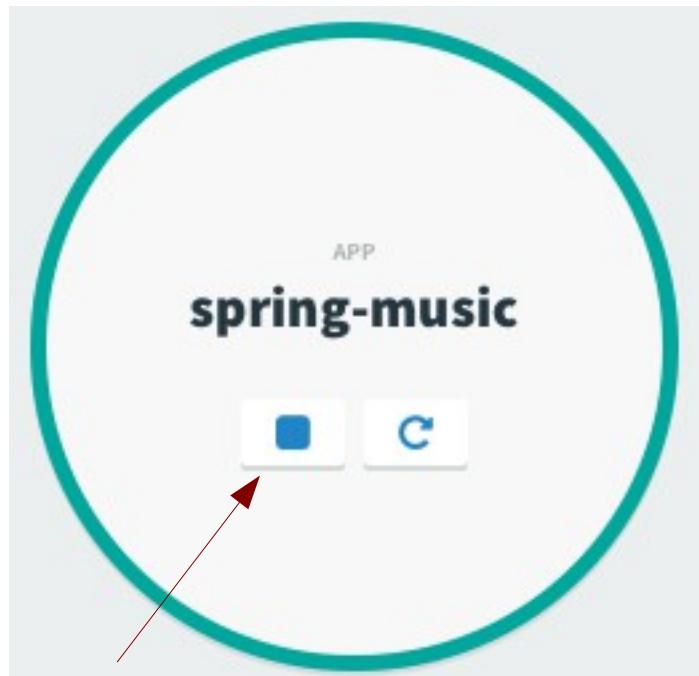
# Change Mapped URL

- Enter new domain (1)
  - Remember to make it *unique*
- Click MAP URL (2)
- To remove a mapping (3)
  - Click UNMAP URL at far right of same line

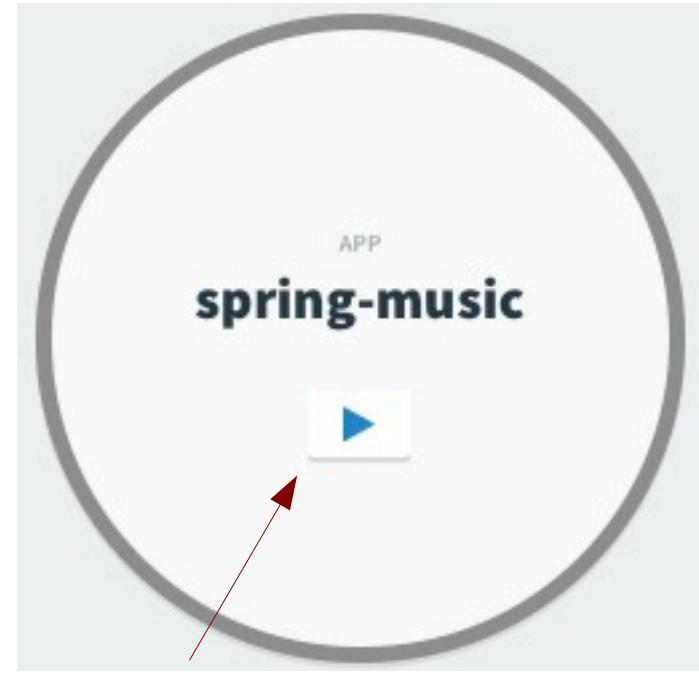


# Stopping and Starting

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



Click to Stop



Click to Start

# Monitoring Instances

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

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

# Summary

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



Pivotal™

# Lab

Push an existing application to Pivotal CF



# Getting Started with Cloud Foundry

Deploying via Eclipse or Spring Tool Suite

Build, deploy, run, monitor

Pivotal

# Overview

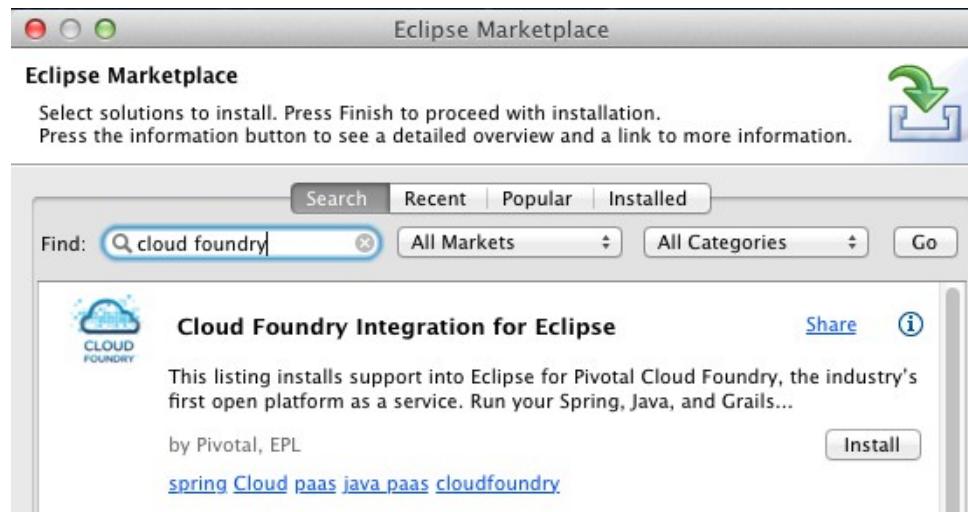
- After completing this lesson, you should be able to:
  - Setup Eclipse or Spring Tool Suite to use Cloud Foundry
  - Deploy an application to CloudFoundry
  - Manage application instances

# Roadmap

- **Get Setup**
- Deploying an Application
- Managing Application Instances

# Setting up Eclipse / Spring Tool Suite

- Select Help → Eclipse Marketplace
- Search for *Cloud Foundry* plugin and install

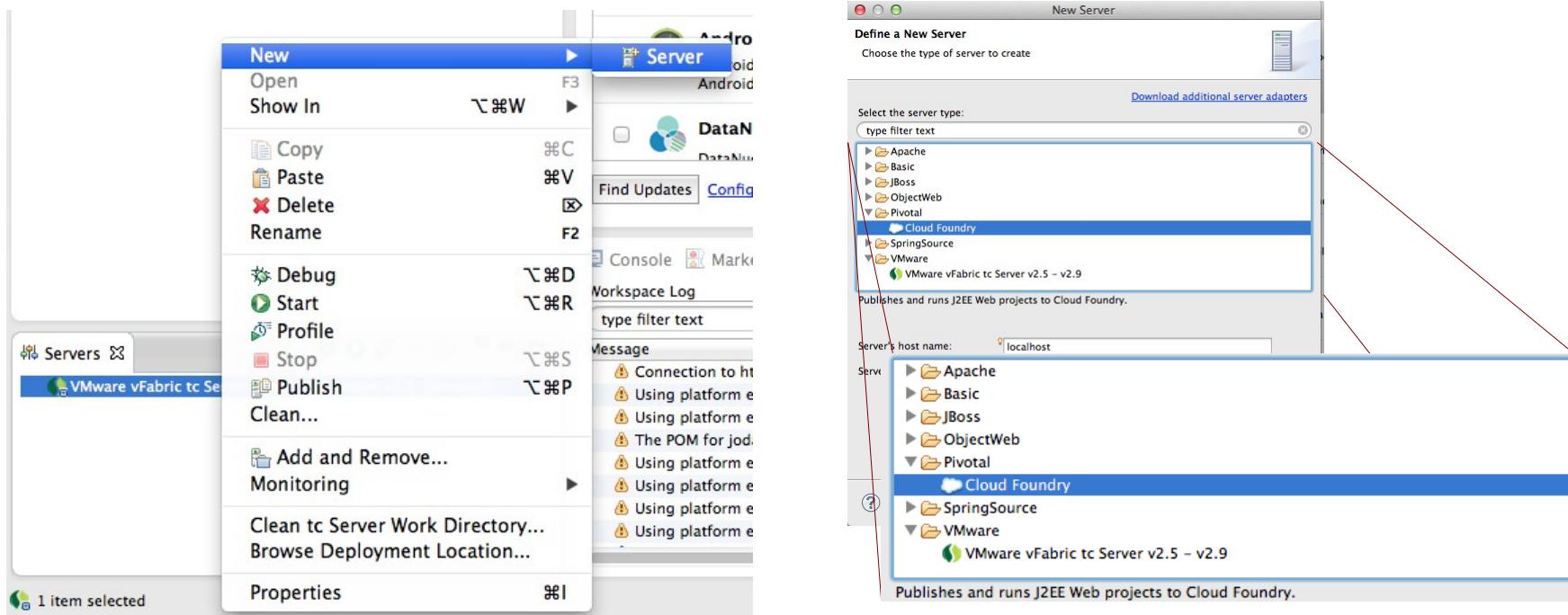


Do this now...

- The install takes time to run.

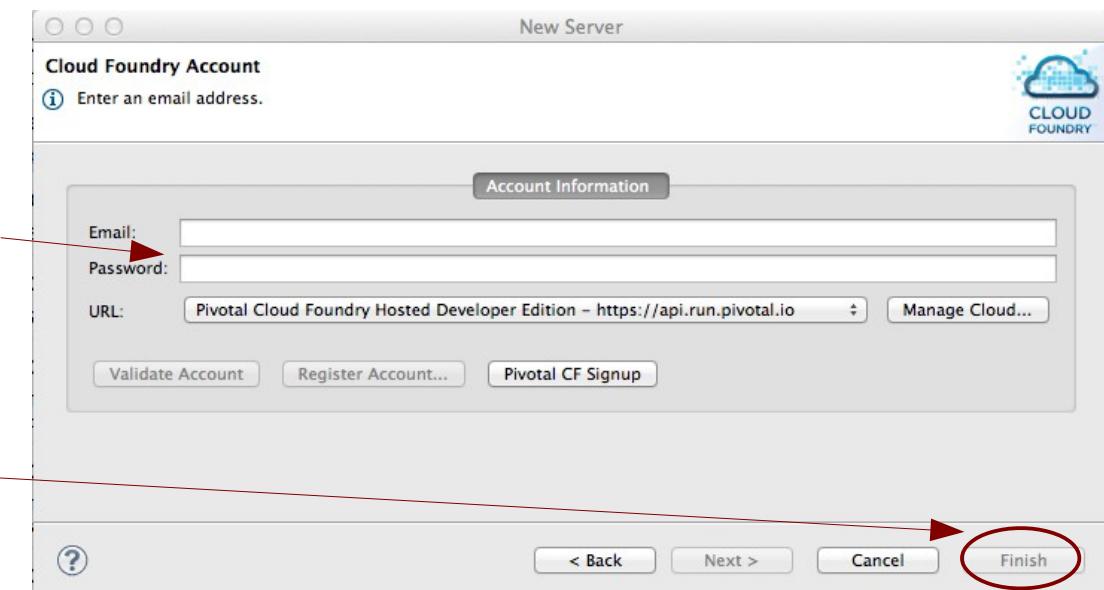
# Setting up a New Server

- In white-area of Servers panel, right click New → Server
- In popup, under Pivotal select Cloud Foundry



# Fill in Details of your CF Account

- Fill in registration dialog
  - Use Manage Cloud to specify a different URL
    - Another public PaaS or for your private cloud
  - Note there is a Signup button here



Same email and password  
That you signed-up with

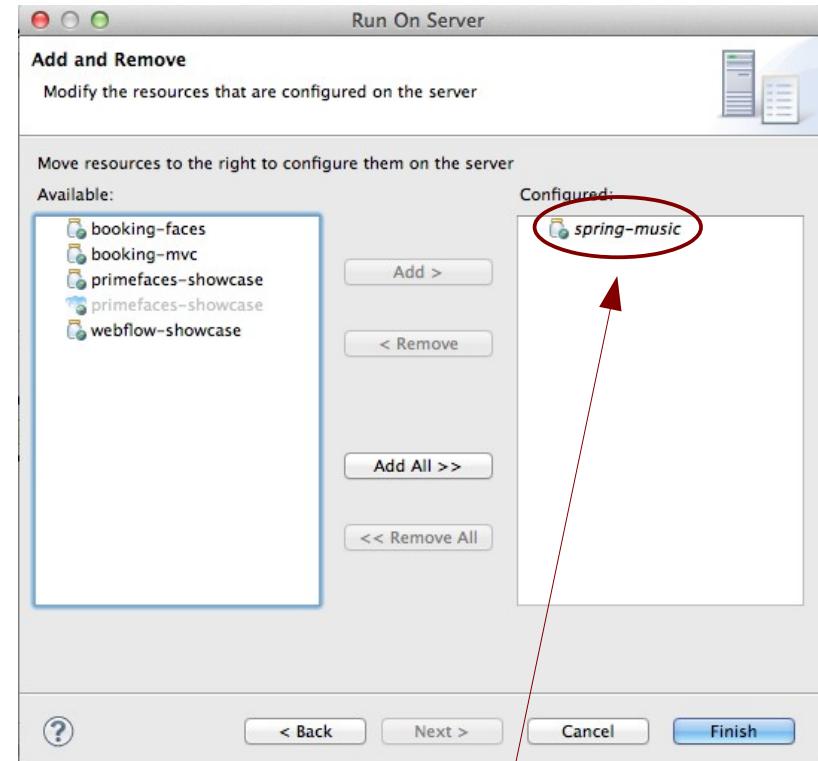
Click Finish when done

# Roadmap

- Get Setup
- **Deploying an Application**
- Managing Application Instances

# Deployment

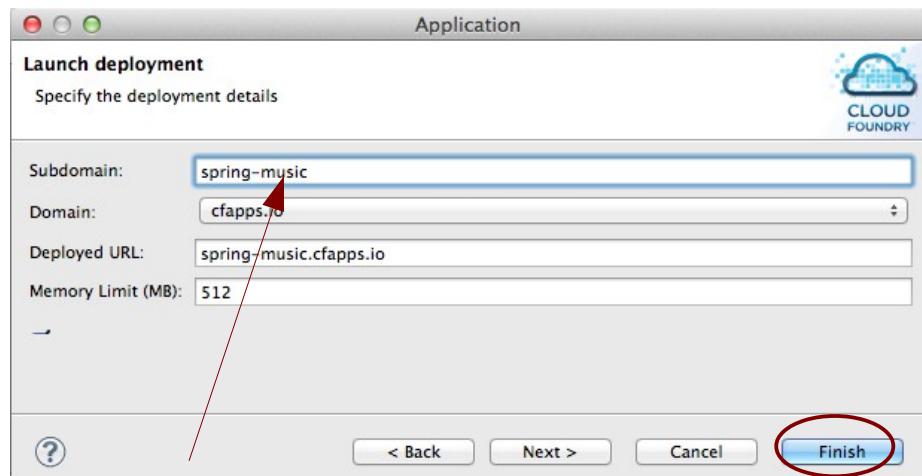
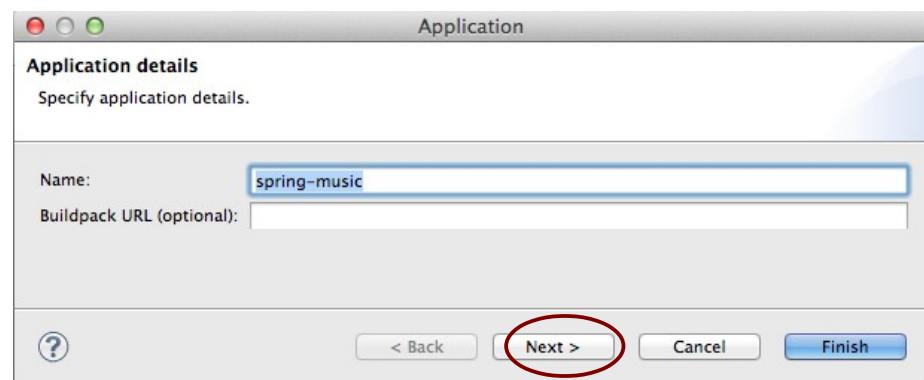
- We are now ready to deploy an application
- Select a project in Eclipse
  - Right click and select
    - Run As ... → Run on Server
    - *Just like any other server*
  - Select Cloud Foundry server
  - Click Next
  - In next dialog make sure your project is in the RHS list
    - Just as you would normally
  - Click Finish to deploy



**Now things get different ...**

# Application details

- Two dialogs appear
  - Application details ... just click next for now
  - Launch deployment ... Pick a *unique* URL
    - All apps deploy to <appname>.cfapps.io
  - For now, just click *Finish* to deploy



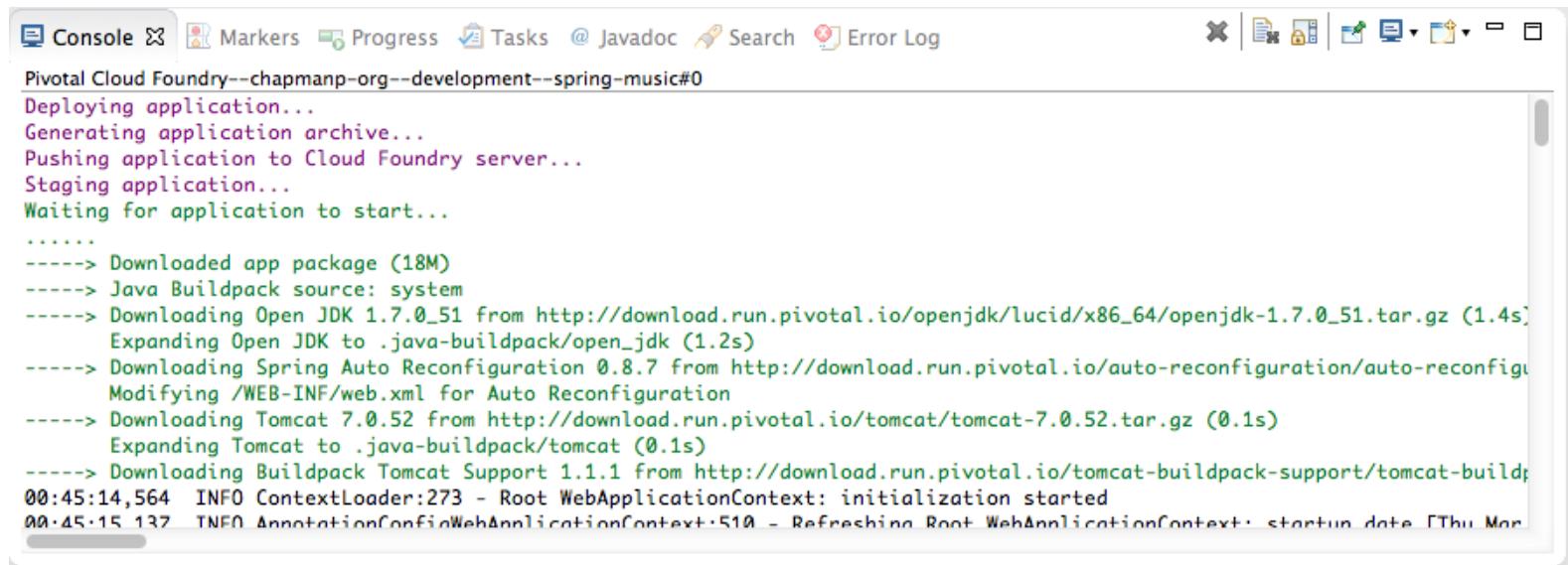
*Will this be unique? Change sub-domain to make sure*

# What just happened?

- Eclipse Connected to Cloud Foundry using your credentials
- It 'pushed' your application to CF and told it to deploy it
  - The whole application is uploaded – takes a while
  - CF “staged” your application
    - Recognized Java / WAR, prepared a “droplet” containing JRE and Tomcat server
    - “Droplet” was deployed to a container and began running
    - All requests to the *Deployed URL* route to your application
- Same process as when using the CLI

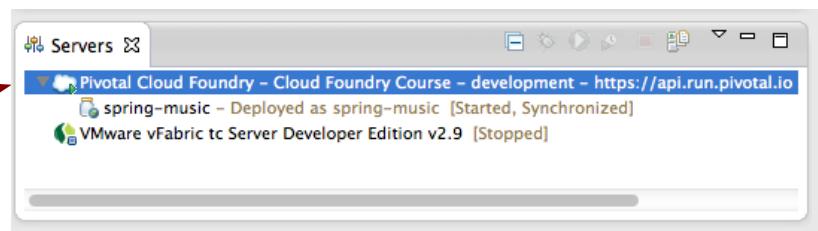
# Watching it Run

- In the Console View



Pivotal Cloud Foundry--chapmanp-org--development--spring-music#0  
Deploying application...  
Generating application archive...  
Pushing application to Cloud Foundry server...  
Staging application...  
Waiting for application to start...  
.....  
----> Downloaded app package (18M)  
----> Java Buildpack source: system  
----> Downloading Open JDK 1.7.0\_51 from http://download.run.pivotal.io/openjdk/lucid/x86\_64/openjdk-1.7.0\_51.tar.gz (1.4s)  
    Expanding Open JDK to .java-buildpack/open\_jdk (1.2s)  
----> Downloading Spring Auto Reconfiguration 0.8.7 from http://download.run.pivotal.io/auto-reconfiguration/auto-reconfiguration  
    Modifying /WEB-INF/web.xml for Auto Reconfiguration  
----> Downloading Tomcat 7.0.52 from http://download.run.pivotal.io/tomcat/tomcat-7.0.52.tar.gz (0.1s)  
    Expanding Tomcat to .java-buildpack/tomcat (0.1s)  
----> Downloading Buildpack Tomcat Support 1.1.1 from http://download.run.pivotal.io/tomcat-buildpack-support/tomcat-buildpack-support-1.1.1.tar.gz (0.0s)  
00:45:14,564 INFO ContextLoader:273 - Root WebApplicationContext: initialization started  
00:45:15.137 TNFO AnnotationConfigWebApplicationContext-510 - Refreshing Root WebApplicationContext: startup date [Thu Mar 12 00:45:15 CDT 2015]; parent [Root WebApplicationContext]

- In the Dashboard
  - Double click



# Overview Dashboard Tab

The screenshot shows the Pivotal Cloud Foundry interface with the "Spring Music" application selected. The "Overview" tab is active, displaying general information, account details, and server status.

**General Information:**  
Specify the host name and other common settings.

- Server name: Pivotal Cloud Foundry
- Host name: localhost
- Runtime Environment: Cloud Foundry (Runtime) v1.0

**Account Information:**

- Email: your email address here
- Password: [REDACTED]
- URL: Pivotal Cloud Foundry Hosted Developer Edition – <https://api.run.pivotal.io>
- Organization: Cloud Foundry Course
- Space: development

Buttons: Clone Server..., Change Password..., Validate Account, Pivotal CF Signup

**Server Status:**  
Pivotal Cloud Foundry: Connected

Buttons: Connect, Disconnect

Navigation tabs at the bottom: Overview (selected), Applications and Services

# Applications Tab

The screenshot shows the Pivotal Cloud Foundry Applications tab interface. At the top, there's a header with the Pivotal Cloud Foundry logo and the application name "Spring Music". Below the header, the main area is titled "Applications" and contains a message: "Select a currently deployed application to see details." A blue box highlights the application "spring-music". A red arrow points to this box with the text "Select app first". To the right of the application list, there's a "General" section with fields for Name (spring-music [Started]), Mapped URLs (spring-music-123.cfapps.io), Instances (1), and Manifest (Save). Below this is a "General (Application Restart Required)" section with Memory Limit (512 MB) and Environment Variables (Edit...). Under "Application Operations", there are buttons for Start, Stop, Restart, and Update and Restart. The "Services" section below the application list has a message: "Drag a service to the right hand side to bind it to an application." It includes a table for managing services. The "Instances" section shows one instance with ID 0, Host 10.10.81.12, CPU 7.7237..., Memory 440388M (5...), Disk 132M (1024M), and Uptime 0h:14m:55s. At the bottom, there are tabs for Overview and Applications and Services, with Applications and Services selected.

Application status here

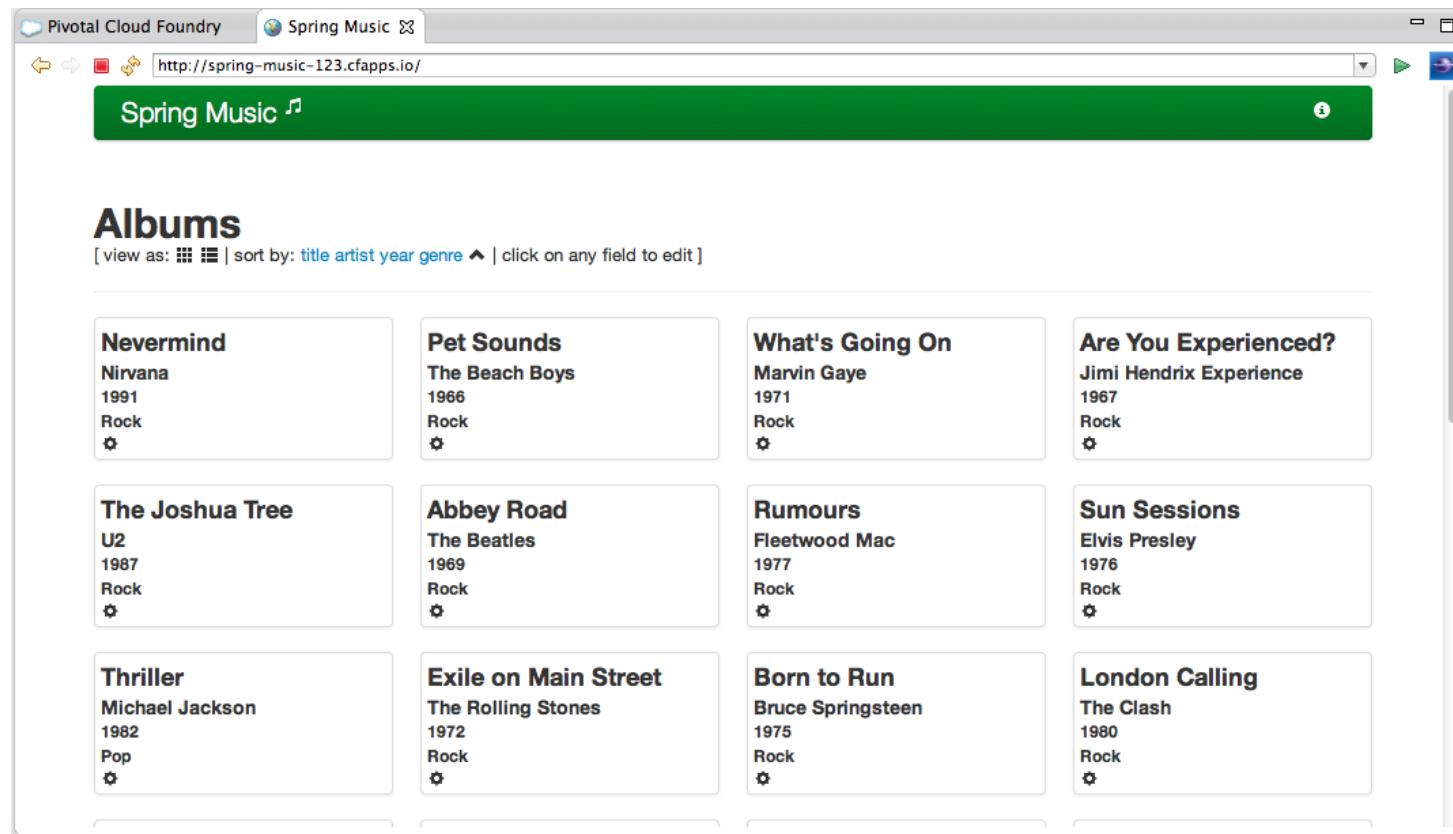
Select app first

Click URL once running

Note URL: added -123 to make it unique

# See Your Application Running in Eclipse

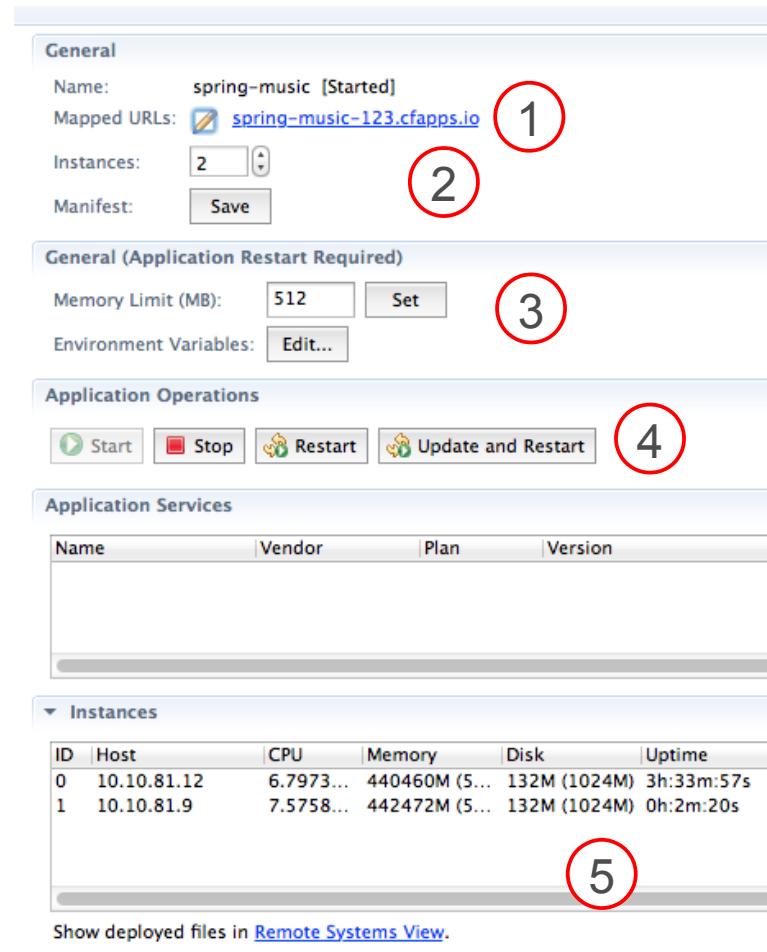
- Eclipse pops up a browser window open at your URL
  - Or use the browser of your choice



# Roadmap

- Getting Setup
- Deploying an Application
- **Managing Application Instances**

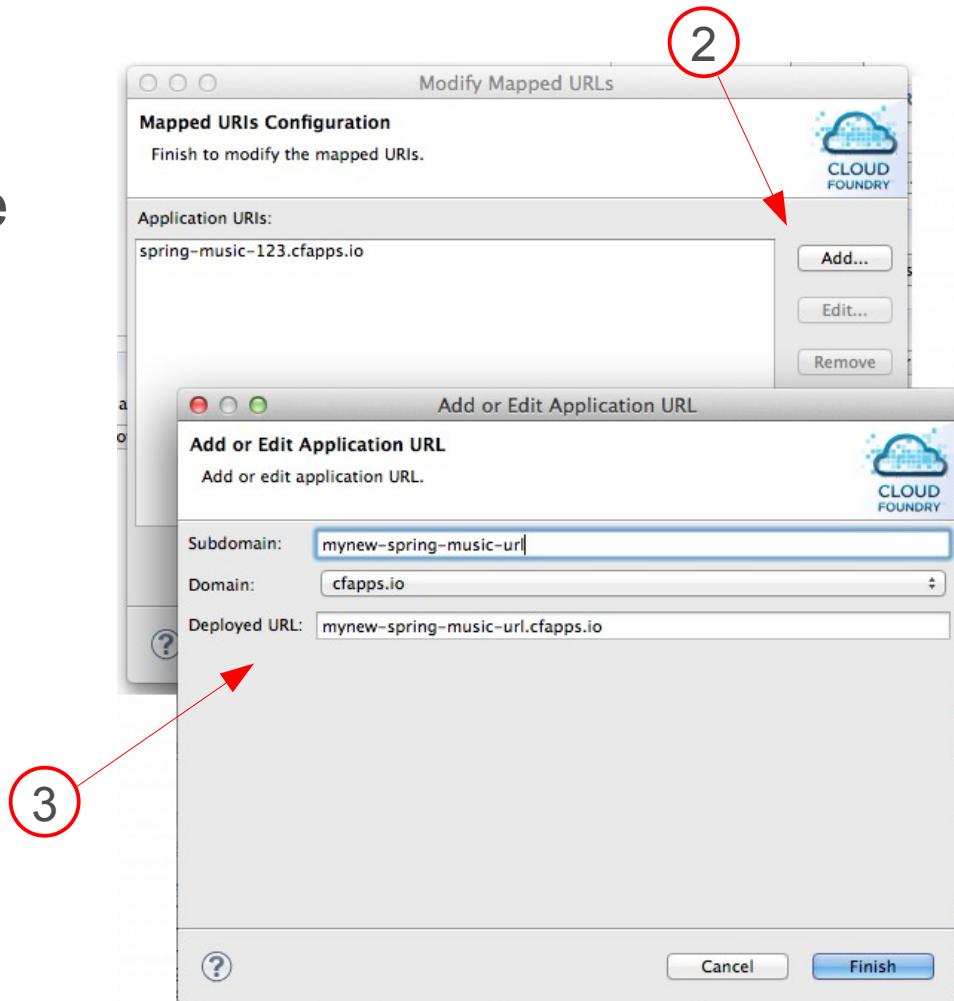
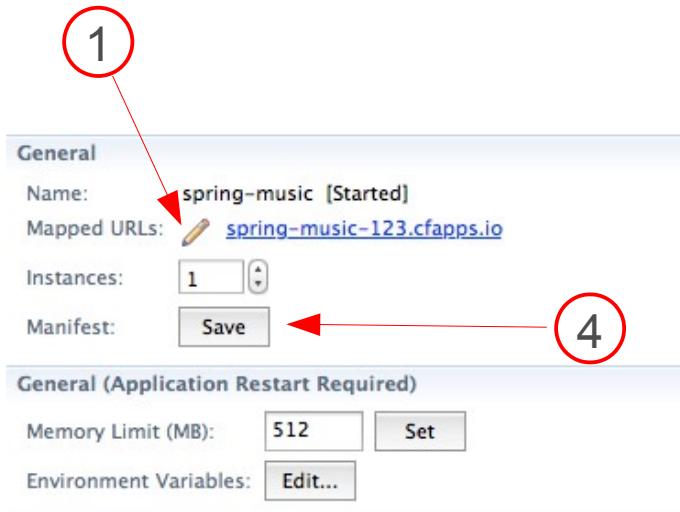
# Cloud Foundry Dashboard



- The right-side panel of Applications and Services tab
    - Below *General*
  - Control your application
    1. Change mapped URL
    2. Add/remove instances
    3. Change memory
    4. Stop/start
    5. Monitor instances

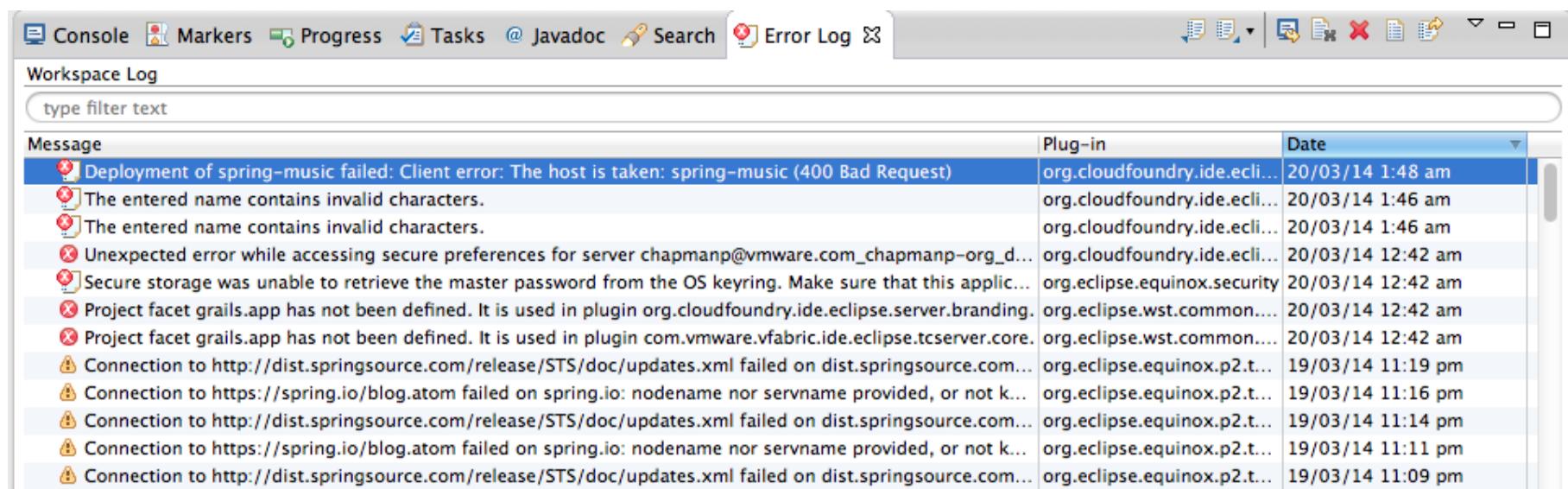
# Change Mapped URL

- Click pencil (edit) icon
- Can add, edit or remove URLs
- Save when done



# Choosing Your URL

- All applications mapped to **cfapps.io** domain
- Your URL must be unique
  - Get a Bad Request 400 if you try to use an existing URL



The screenshot shows the Eclipse IDE's Error Log view. The title bar includes tabs for Console, Markers, Progress, Tasks, Javadoc, Search, and Error Log. Below the title bar is a toolbar with various icons. The main area is titled "Workspace Log" and contains a search bar with the placeholder "type filter text". A table lists deployment errors:

Message	Plug-in	Date
Deployment of spring-music failed: Client error: The host is taken: spring-music (400 Bad Request)	org.cloudfoundry.ide.ecli...	20/03/14 1:48 am
The entered name contains invalid characters.	org.cloudfoundry.ide.ecli...	20/03/14 1:46 am
The entered name contains invalid characters.	org.cloudfoundry.ide.ecli...	20/03/14 1:46 am
Unexpected error while accessing secure preferences for server chapmanp@vmware.com_chapmanp-org_d...	org.cloudfoundry.ide.ecli...	20/03/14 12:42 am
Secure storage was unable to retrieve the master password from the OS keyring. Make sure that this applic...	org.eclipse.equinox.security	20/03/14 12:42 am
Project facet grails.app has not been defined. It is used in plugin org.cloudfoundry.ide.eclipse.server.branding.	org.eclipse.wst.common....	20/03/14 12:42 am
Project facet grails.app has not been defined. It is used in plugin com.vmware.vfabric.ide.eclipse.tcserver.core.	org.eclipse.wst.common....	20/03/14 12:42 am
Connection to http://dist.springsource.com/release/STS/doc/updates.xml failed on dist.springsource.com...	org.eclipse.equinox.p2.t...	19/03/14 11:19 pm
Connection to https://spring.io/blog.atom failed on spring.io: nodename nor servname provided, or not k...	org.eclipse.equinox.p2.t...	19/03/14 11:16 pm
Connection to http://dist.springsource.com/release/STS/doc/updates.xml failed on dist.springsource.com...	org.eclipse.equinox.p2.t...	19/03/14 11:14 pm
Connection to https://spring.io/blog.atom failed on spring.io: nodename nor servname provided, or not k...	org.eclipse.equinox.p2.t...	19/03/14 11:11 pm
Connection to http://dist.springsource.com/release/STS/doc/updates.xml failed on dist.springsource.com...	org.eclipse.equinox.p2.t...	19/03/14 11:09 pm

# Instances

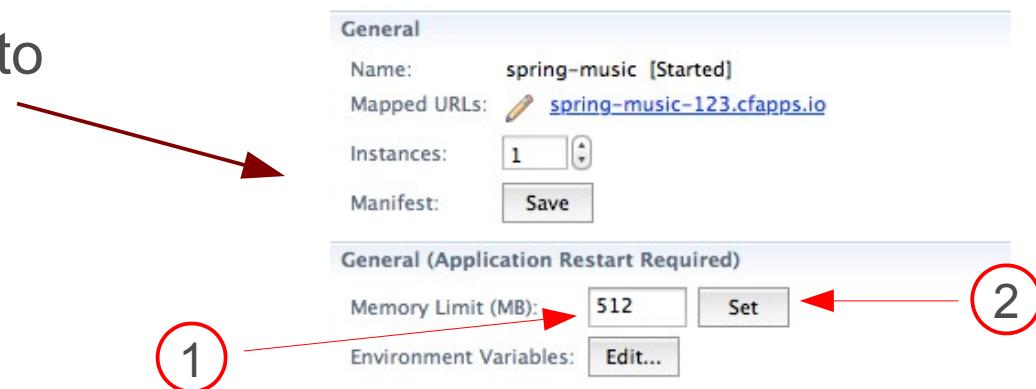
- By default one instance of your application runs up
  - Typically a Tomcat server
- To handle large loads you need multiple servers
  - Known as instances
  - Run behind load balancer
- How many instances do I need?
  - Design issue – covered later
- Modify as shown



The screenshot shows the 'General' configuration page for an application named 'spring-music [Started]'. The 'Mapped URLs' field contains 'spring-music-123.cfapps.io'. The 'Instances' field is set to '1'. The 'Manifest' section includes a 'Save' button. Below the main configuration, there is a note 'General (Application Restart Required)' followed by 'Memory Limit (MB): 512' and 'Environment Variables: Edit...'. Red annotations include a circle labeled '1' around the 'Instances' input field and another circle labeled '2' around the 'Save' button.

# Memory Allocation

- Define how much memory our process gets to run in
  - 512 is the default
    - good for a typical application under test (1 or 2 users)
  - How much memory do I need in production
    - Another good question for later!
- Easy to configure ...
  - But the server has to be *restarted*



# Stopping and Starting

- Normally this happens in the Servers view
  - Those buttons are greyed out
- Instead use the Dashboard

The screenshot shows the Cloud Foundry Dashboard for an application named "spring-music". The application status is listed as "[Started]". The dashboard includes sections for "General", "Application Operations", and "Logs".

**General** section:

- Mapped URLs: [spring-music-123.cfapps.io](http://spring-music-123.cfapps.io)
- Instances: 1
- Manifest: Save

**Application Operations** section:

- Memory Limit (MB): 512 (Set button)
- Environment Variables: Edit...

**Buttons at the bottom:**

- Start (green play icon)
- Stop (red square icon)
- Restart (green recycling icon)
- Update and Restart (green recycling icon)

A large red arrow points from the text "Those buttons are greyed out" in the slide content down towards the "Stop" and "Restart" buttons in the screenshot.

# Monitoring Instances

- The very bottom panel shows all your instances
  - Provides statistics
  - *Not* real-time
- To refresh
  - Click refresh icon on the application list

Application Services																							
Name	Vendor	Plan	Version																				
<b>Instances</b>																							
<table border="1"><thead><tr><th>ID</th><th>Host</th><th>CPU</th><th>Memory</th><th>Disk</th><th>Uptime</th></tr></thead><tbody><tr><td>0</td><td>10.10.81.12</td><td>9.7046974537506...</td><td>440484M (5...</td><td>132M (1024M)</td><td>4h:44m:48s</td></tr><tr><td>1</td><td>10.10.81.9</td><td>7.5637518159724...</td><td>442484M (5...</td><td>132M (1024M)</td><td>1h:13m:11s</td></tr></tbody></table>						ID	Host	CPU	Memory	Disk	Uptime	0	10.10.81.12	9.7046974537506...	440484M (5...	132M (1024M)	4h:44m:48s	1	10.10.81.9	7.5637518159724...	442484M (5...	132M (1024M)	1h:13m:11s
ID	Host	CPU	Memory	Disk	Uptime																		
0	10.10.81.12	9.7046974537506...	440484M (5...	132M (1024M)	4h:44m:48s																		
1	10.10.81.9	7.5637518159724...	442484M (5...	132M (1024M)	1h:13m:11s																		

Show deployed files in [Remote Systems View](#).

# Summary: Cloud Foundry Dashboard

- In the Application and Services tab
  - Configure your application (below *General* on right-side)
  - Options
    - Modify mapped URL
    - Change number of instances
    - Change the amount of memory allocated
    - Start and stop the application
    - Monitor instances
  - You may have noticed we missed two options (later)
    - Add or remove services
    - Set environment variables

# Summary

- After completing this lesson, you should have learned:
  - Get setup to use Cloud Foundry
    - See Appendices for installing Cloud Foundry plug-in into Eclipse or Spring Tool Suite
  - Deploy an application to CloudFoundry using Eclipse
  - Manage application instances



# Lab

Deploy an existing application to  
Cloud Foundry using Eclipse

# Appendices

- Describe installation of Cloud Foundry plug-in into
  - Appendix A: STS
  - Appendix B: Standard Eclipse

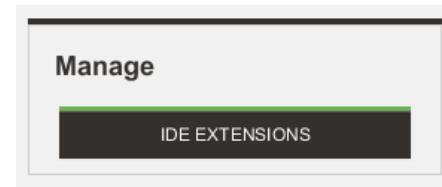


For full details see:

<http://docs.cloudfoundry.org/devguide/deploy-apps/sts.html>

# Appendix A: Installing CF Plugin into STS

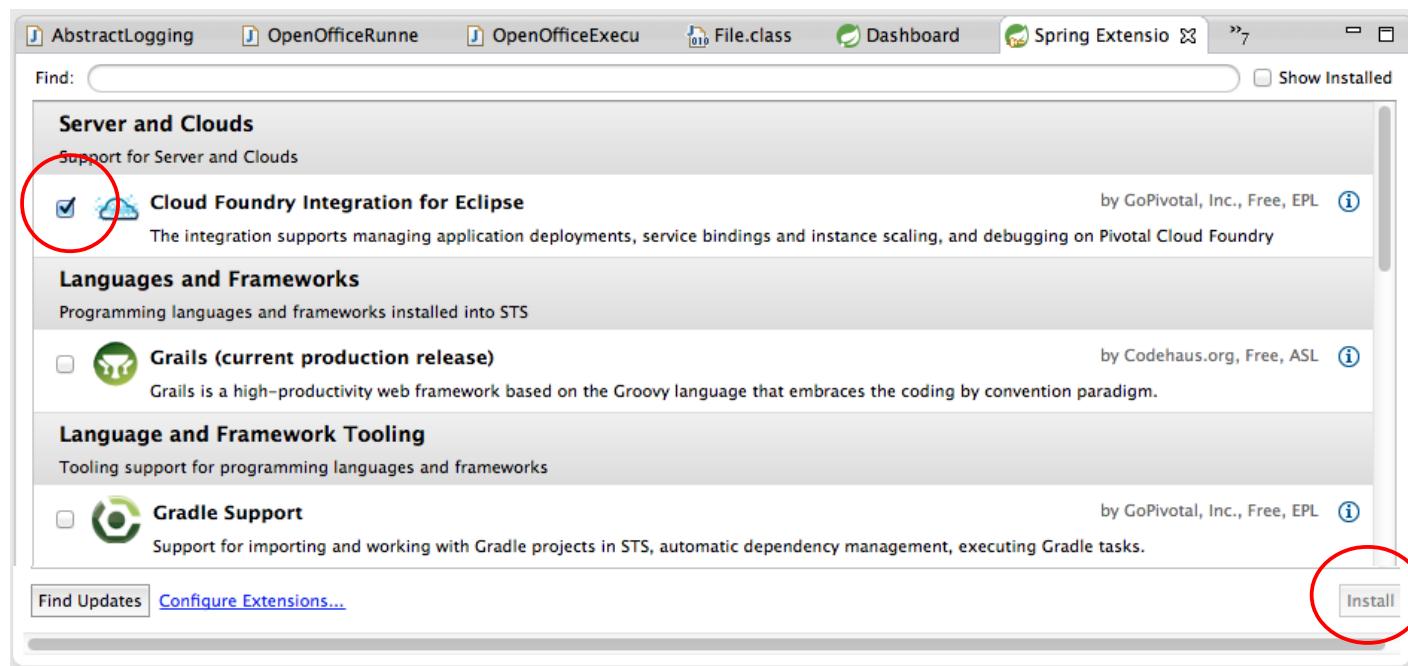
- Click Spring leaf icon in STS
  - Displays dashboard
- At bottom right under Manage
  - click “IDE EXTENSIONS”



See: <http://docs.cloudfoundry.org/devguide/deploy-apps/sts.html#install-to-sts>

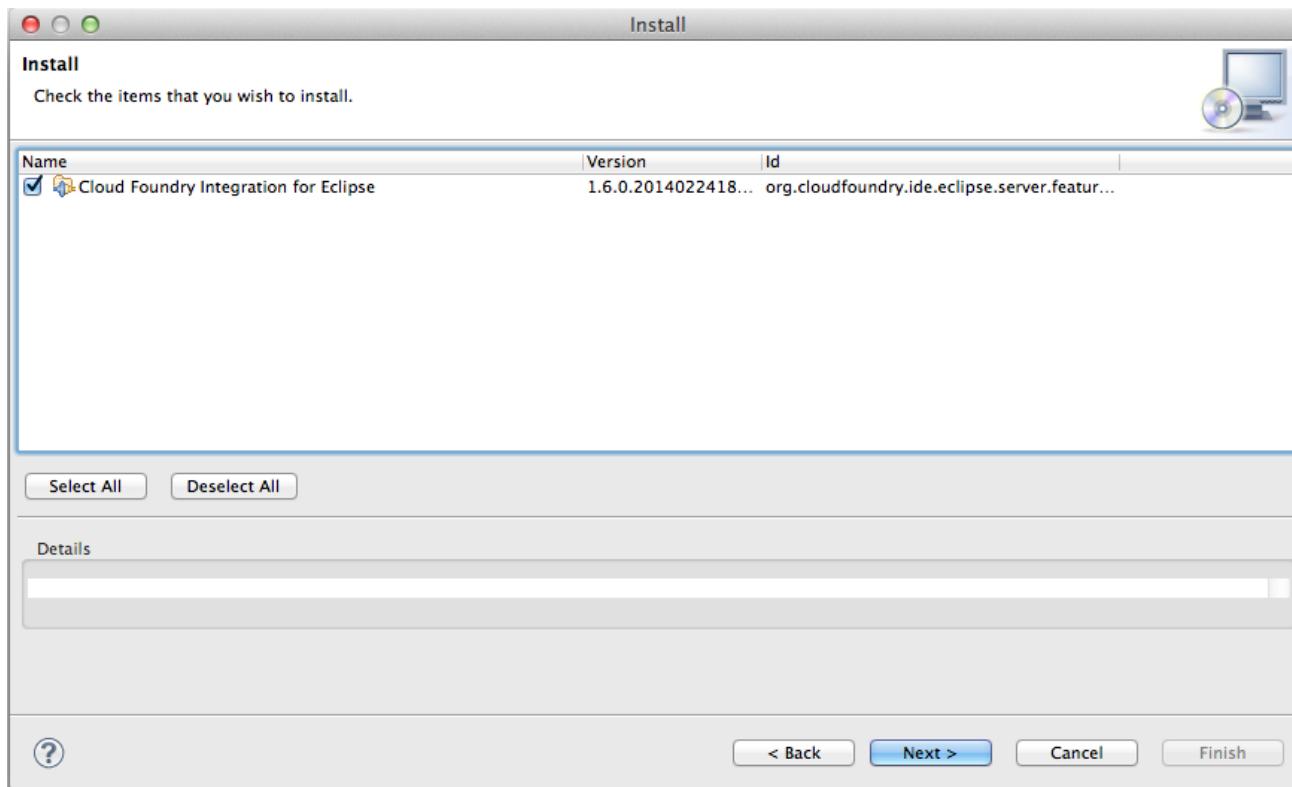
# Select Cloud Foundry Integration for Eclipse

- Select checkbox and click *Install* button
  - If not listed, enter “cloud foundry” in Find and hit enter



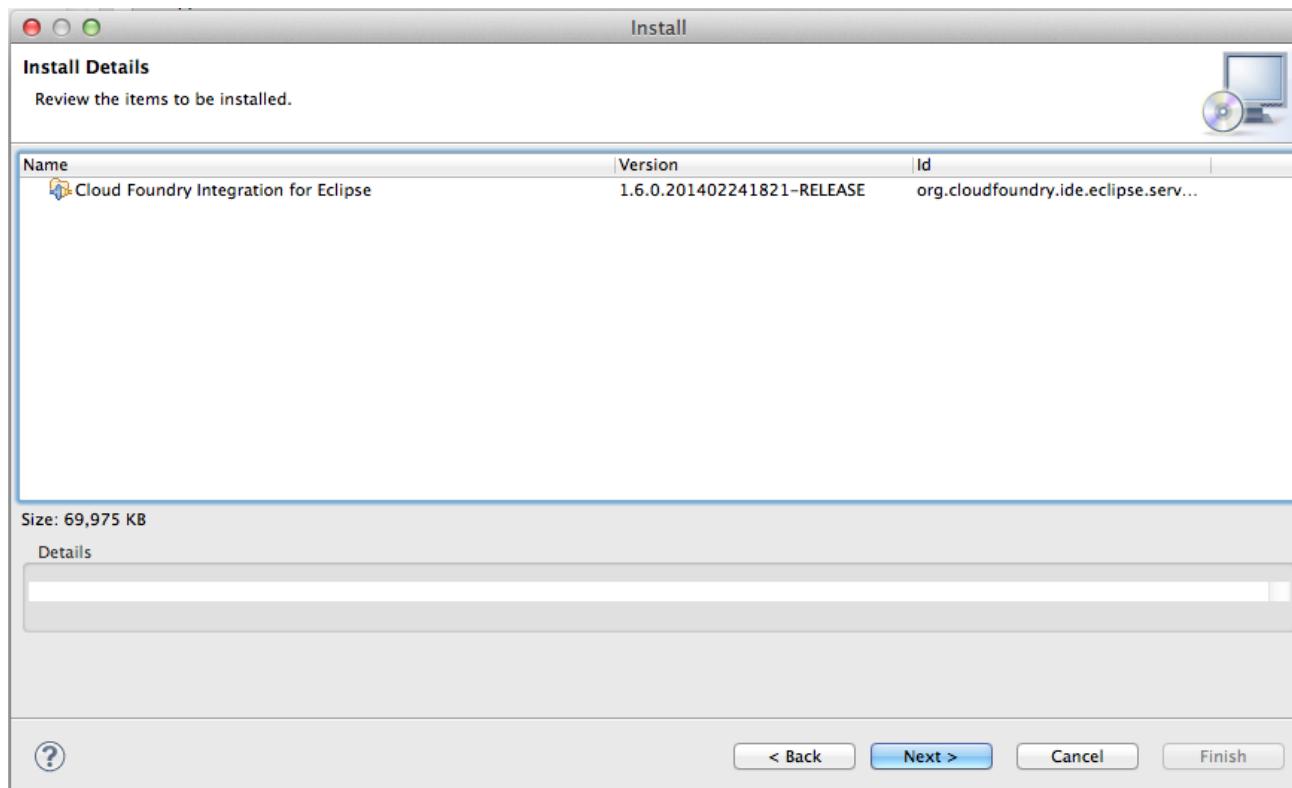
# Runs up a Wizard

- Click Next



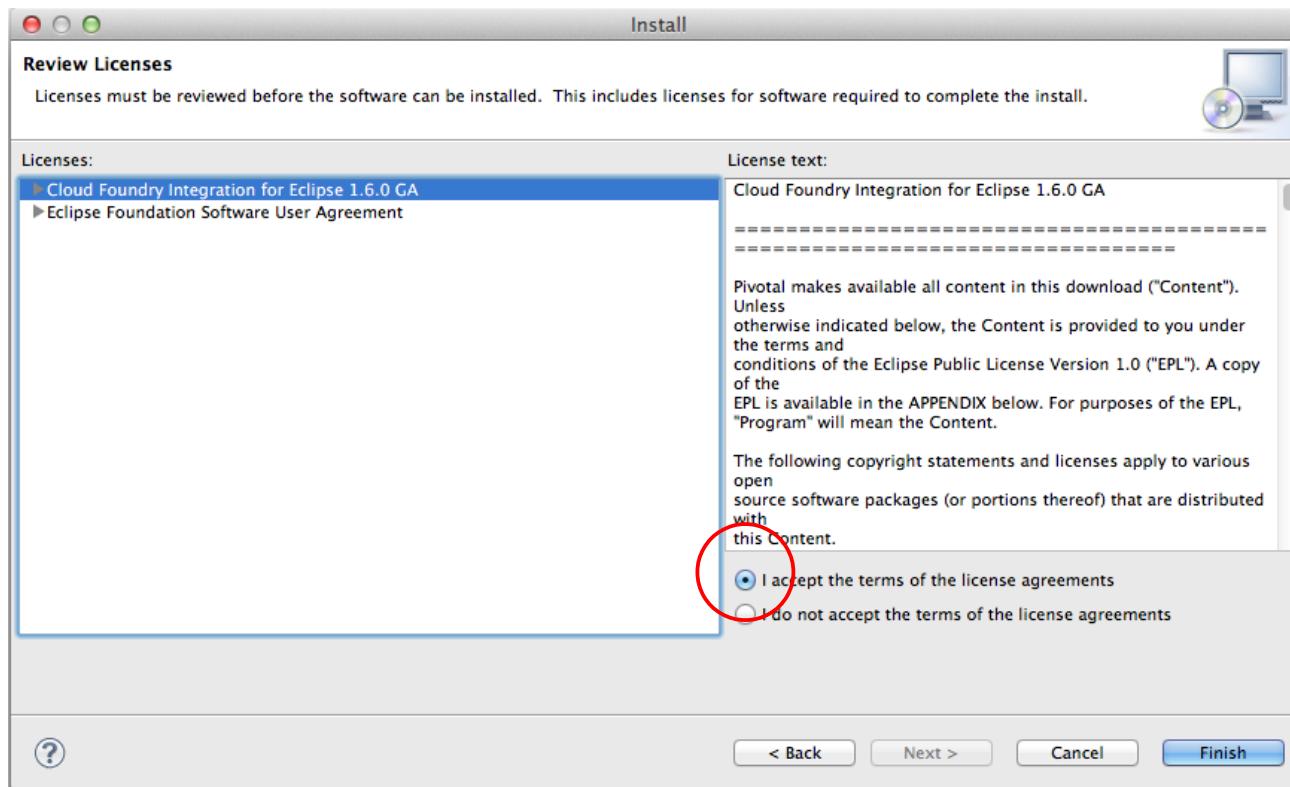
# Wizard – Step 2

- Click Next again



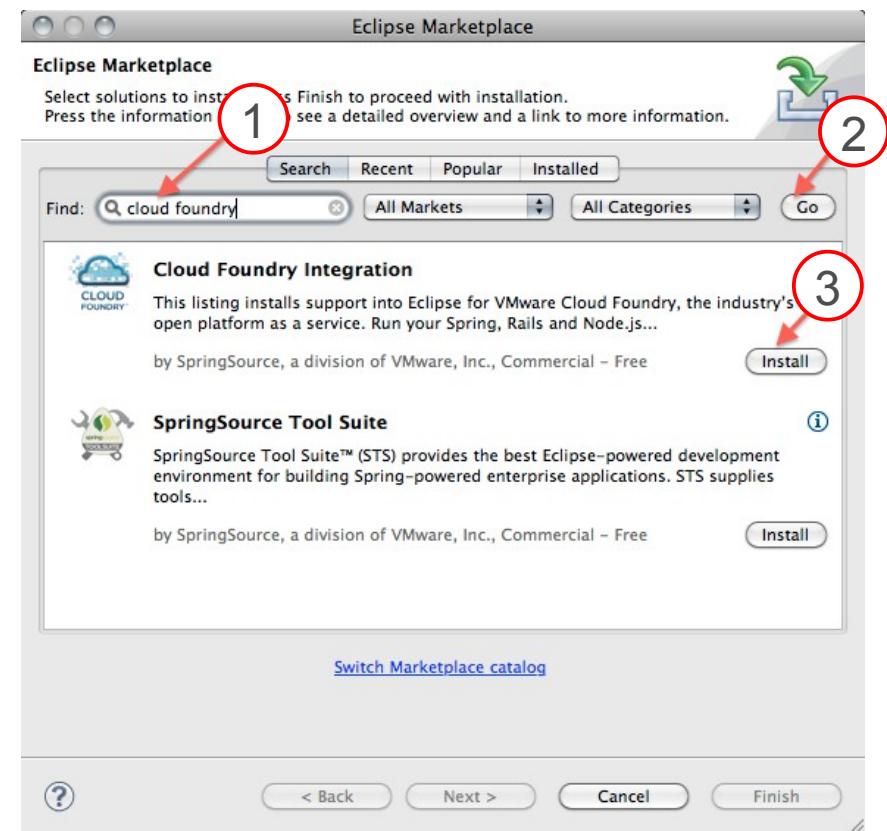
# Wizard – Step 3

- Accept license agreement, and the installer will run



# Appendix B: Installing into Eclipse

- From Help menu
  - Eclipse Marketplace
- Enter “cloud foundry” into Find box
- Click “Go”
- Click “Cloud Foundry Integration” *Install* button



See: <http://docs.cloudfoundry.org/devguide/deploy-apps/sts.html#install-to-eclipse>

# Confirm Selected Features

- Popup window lists what will be installed
  - “Cloud Foundry Integration for Eclipse”
  - “SpringSource UAA Integration” (optional)
    - Reports tool usage data, *anonymously*
    - Helps us track usage of free software
    - Deselect to stop plugin usage statistics being sent
  - Click Confirm.



Optional, deselect if you wish →

# Last Few Steps

- Accept the license agreement
- Click Finish
- Installer runs (takes a while)
- Eventually you are asked to restart Eclipse





# Logging and Troubleshooting in Cloud Foundry

A closer look at practical Cloud Foundry usage

Logging, Troubleshooting, and  
Understanding Application Deployment

Pivotal

# Overview

- After completing this lesson, you should be able to:
  - View Logs from your CF application
  - Troubleshoot your application
  - Understand how deployment works

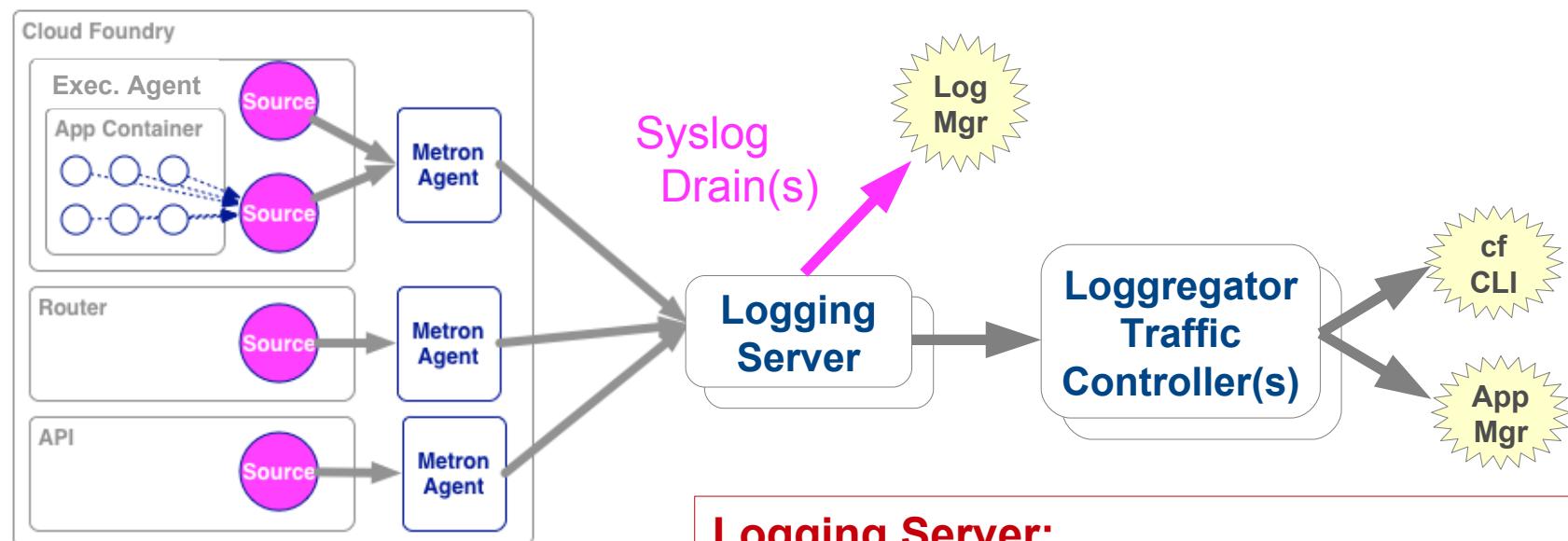
# Roadmap

- Logging
- Troubleshooting
- How Deployment Happens

# Log Aggregation Architecture

- Collects logging output from:
  - Application Instances, CF Components
- Aggregates into a consolidated log

**Execution Agent VM**  
**DEA: Droplet**  
Execution Agent  
Termed **Cells** in *Diego*



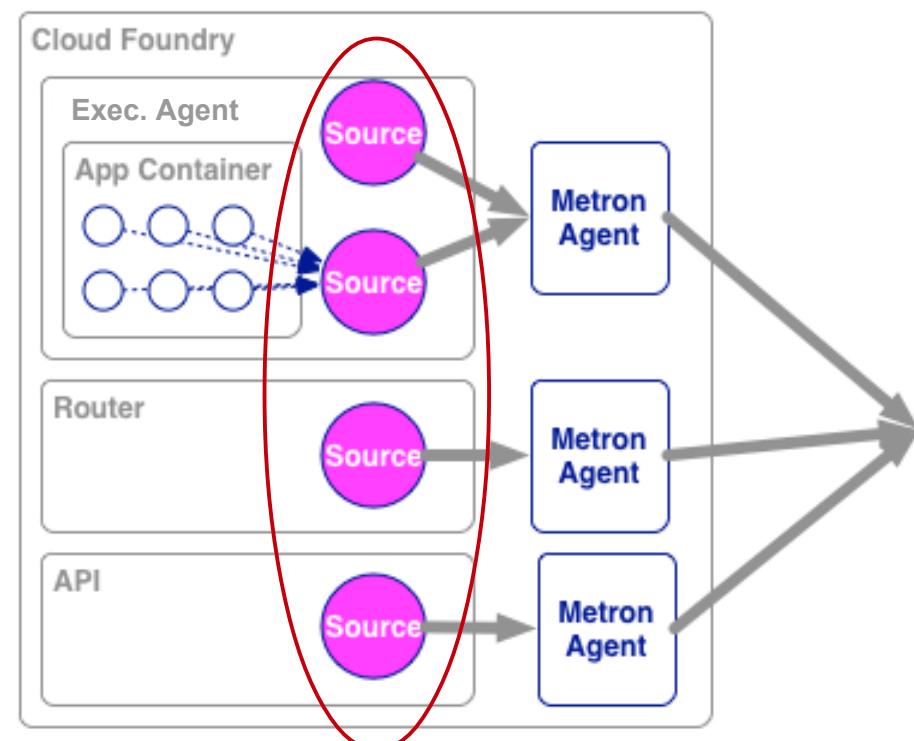
**Logging Server:**  
formerly *Loggregator*, now called *Doppler*

# Logging Terminology

- CF 1.3 or earlier
  - Apps & CF Components logged to a *Loggregator* process
  - It both accumulated and supplied logging information
  - *Loggregator* referred to this process
- CF 1.4 or later
  - **Loggregator:** *entire Log Aggregation Architecture*
    - Metric agents + logging servers + traffic controllers
  - Multiple components
    - Metron Agents: receive both logs *and* metrics
    - Doppler Server(s): accumulates data from agents
    - Traffic Controller(s): allow access to metrics and logs

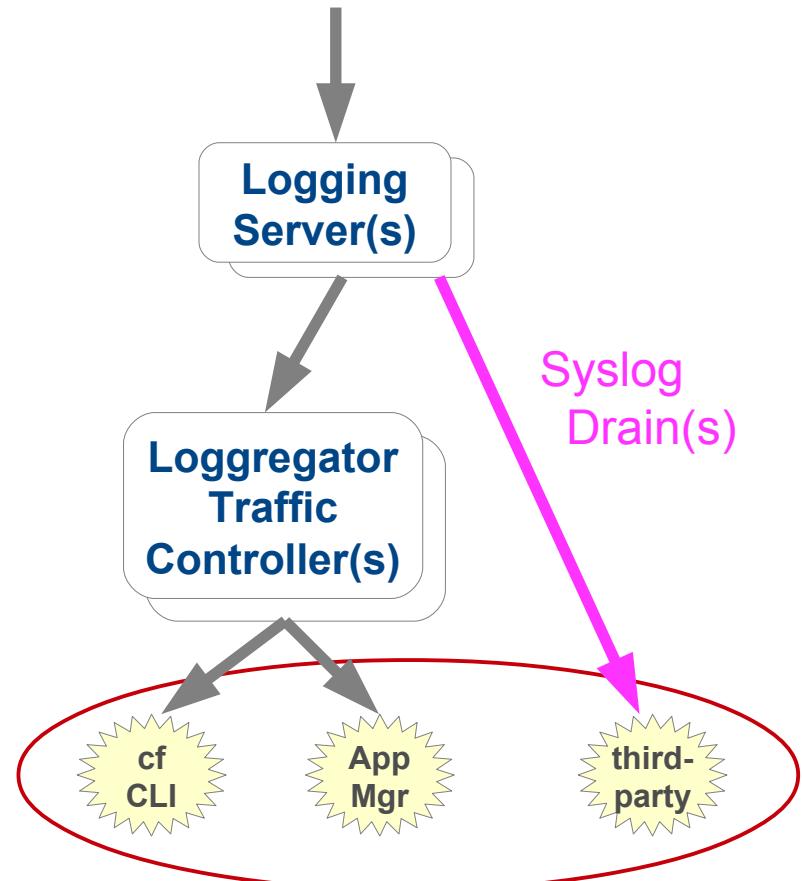
# Logging - Sources

- Application Instances
  - Logs collected from all application instances
    - From *sysout* or *syserr*
- Cloud Foundry Components
  - Router, Cloud Controller, Execution Agents ...
  - Useful to understand update / deployment issues



# Logging - Sinks

- CLI – Tail
  - Quick, direct way to obtain output
  - `cf logs <appname>`
- Third-party log managers
  - Splunk, Logentries, logstash, Papertrail, etc.



# A Tour of Log Output

Timestamp – Added by  
Loggregator

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO WebApplicationC  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO WebApplicationC  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...  
2014-04-11T14:58:54.47 [App/0] OUT ...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/0] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:01:09.44 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /info HTTP/1.1" 200 48  
2014-04-11T15:01:09.46 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /albums HTTP/1.1" 200 4669  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance (index 1) with guid 018.70d-e84f05375859  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52  
2014-04-11T14:58:54.47 [App/1] OUT ...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/1] ERR INFO: Initializing Spring Environment for Cloud Foundry  
2014-04-11T14:58:57.44 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/1] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances":>2}
```

# A Tour of Log Output

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO ...  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO ...  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO ...  
2014-04-11T14:58:54.47 [App/0] OUT ...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert int  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert int  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring Environment for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] OUT spring-music-567.cf  
2014-04-11T14:58:57.45 [App/0] OUT spring-music-567.cf  
2014-04-11T15:01:09.44 [RTR] OUT Starting app instance 1  
2014-04-11T15:01:09.46 [RTR] OUT Starting app instance 1  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance 1  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Server startup in 8713 ms  
2014-04-11T14:58:54.47 [App/1] OUT ...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert int  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert int  
2014-04-11T14:58:56.38 [App/1] ERR INFO: Initializing Spring Environment for Cloud Foundry  
2014-04-11T14:58:57.44 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/1] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances"=>2}
```

## Log Type / Origin

Where this message came from

- API – Cloud Controller
  - A change in application state
- App/# - Your Application / Instance number
- RTR – Router
- DEA – Droplet Execution Agent
- etc.

# A Tour of Log Output

2014-04-11T14:58:41.66 [API]	OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}
2014-04-11T14:58:53.32 [App/0]	OUT 18:58:53,323 INFO WebApplicationContext:244 - Found 1 ...
2014-04-11T14:58:53.32 [App/0]	OUT 18:58:53,324 INFO WebApplicationContext:229 - Successfully resolved ...
2014-04-11T14:58:53.52 [App/0]	OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...
2014-04-11T14:58:54.47 [App/0]	OUT ...]
2014-04-11T14:58:56.24 [App/0]	OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)
2014-04-11T14:58:56.25 [App/0]	OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)
2014-04-11T14:58:56.38 [App/0]	ERR INFO: Initializing Spring for Cloud Foundry
2014-04-11T14:58:57.44 [App/0]	ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start
2014-04-11T14:58:57.44 [App/0]	ERR INFO: Starting ProtocolHandler ["http-bio-62773"]
2014-04-11T14:58:57.45 [App/0]	ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start
2014-04-11T14:58:57.45 [App/0]	ERR INFO: Server startup in 8713 ms
2014-04-11T15:01:09.44 [RTR]	OUT spring-music-567.cfapps.io - [19:01:09 +
2014-04-11T15:01:09.46 [RTR]	OUT spring-music-567.cfapps.io - [19:01:09 +
2014-04-11T15:03:35.16 [DEA]	OUT Starting app instance (index 1) with guid
2014-04-11T15:03:40.43 [App/1]	ERR INFO: Starting Servlet Engine: Apache T
2014-04-11T14:58:54.47 [App/1]	OUT ...]
2014-04-11T14:58:56.26 [App/1]	OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)
2014-04-11T14:58:56.26 [App/1]	OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)
2014-04-11T14:58:56.38 [App/1]	ERR INFO: Initializing Spring Environment for Cloud Foundry
2014-04-11T14:58:57.44 [App/1]	ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start
2014-04-11T14:58:57.44 [App/1]	ERR INFO: Starting ProtocolHandler ["http-bio-62773"]
2014-04-11T14:58:57.45 [App/1]	ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start
2014-04-11T14:58:57.45 [App/1]	ERR INFO: Server startup in 8713 ms
2014-04-11T15:03:35.15 [API]	OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances"=>2}

## Channel

- Whether this output came from “SYSOUT” or “SYSERR”

# A Tour of Log Output

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO WebApplicationContext:244 - Found 1 ...  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO WebApplicationContext:229 - Successfully resolved ...  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...  
2014-04-11T14:58:54.47 [App/0] OUT [...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/0] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:01:09.44 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /info HTTP/1.1" 200 48  
2014-04-11T15:01:09.46 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /albums HTTP/1.1" 200 4669  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance (index 1) with guid 018.70d-e84f05375859  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52  
2014-04-11T14:58:54.47 [App/1] OUT [...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
                                         um (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
                                         Environment for Cloud Foundry  
                                         org.apache.coyote.AbstractProtocol start  
                                         andler ["http-bio-62773"]  
                                         org.apache.catalina.startup.Catalina start
```

## Message

- From the application / component

```
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances"=>2}
```

# A Tour of Log Output

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO WebApplicationContext:244 - Found 1 ...  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO WebApplicationContext:229 - Successfully resolved ...  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...  
2014-04-11T14:58:54.47 [App/0] OUT ...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/0] ERR INFO: Server startup in 8713 ms  
  
2014-04-11T15:01:09.44 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /info HTTP/1.1" 200 48  
2014-04-11T15:01:09.46 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /albums HTTP/1.1" 200 1660  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance (index 1) with guid 018f9a20-f0c9.5375859  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52  
2014-04-11T14:58:54.47 [App/1] OUT ...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/1] ERR INFO: Initializing Spring Environment for Cloud Foundry  
2014-04-11T14:58:57.44 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/1] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances"=>2}
```

Instance 0 startup activity  
(condensed)

# A Tour of Log Output

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO WebApplicationContext:244 - Found 1 ...  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO WebApplicationContext:229 - Successfully resolved ...  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...  
2014-04-11T14:58:54.47 [App/0] OUT ...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/0] ERR INFO: Server startup in 8713 ms  
  
2014-04-11T15:01:09.44 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /info HTTP/1.1" 200 48  
2014-04-11T15:01:09.46 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /albums HTTP/1.1" 200 4669  
  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance (index 1) with guid 018.70d-e84f05375859  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52  
2014-04-11T14:58:54.47 [App/1] OUT ...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/1] ERR INFO: Initializing Spring Environment for Cloud Foundry  
2014-04-11T14:58:57.44 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/1] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f053
```

User requests for a web page on instance 0

# A Tour of Log Output

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO WebApplicationContext:244 - Found 1 ...  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO WebApplicationContext:229 - Successfully resolved ...  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...  
2014-04-11T14:58:54.47 [App/0] OUT ...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/0] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:01:09.44 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /info HTTP/1.1" 200 48  
2014-04-11T15:01:09.46 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /albums HTTP/1.1" 200 4669  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance (index 1) with guid 018.70d-e84f05375859  
  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52  
2014-04-11T14:58:54.47 [App/1] OUT ...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId,  
2014-04-11T14:58:56.38 [App/1] ERR INFO: Initializing Spring Environment  
2014-04-11T14:58:57.44 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/1] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances"=>2}
```

Request to scale application

- `cf scale <appname> -i 2`

# A Tour of Log Output

```
2014-04-11T14:58:41.66 [API] OUT Updated app with guid 018f9a20-f0c9.5375859 {"state"=>"STARTED"}  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,323 INFO WebApplicationContext:244 - Found 1 ...  
2014-04-11T14:58:53.32 [App/0] OUT 18:58:53,324 INFO WebApplicationContext:229 - Successfully resolved ...  
2014-04-11T14:58:53.52 [App/0] OUT 18:58:53,527 INFO XmlBeanReader:316 - Loading definitions from class path ...  
2014-04-11T14:58:54.47 [App/0] OUT ...]  
2014-04-11T14:58:56.24 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.25 [App/0] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/0] ERR INFO: Initializing Spring for Cloud Foundry  
2014-04-11T14:58:57.44 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/0] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/0] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/0] ERR INFO: Server startup in 8713 ms  
2014-04-11T15:01:09.44 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /info HTTP/1.1" 200 48  
2014-04-11T15:01:09.46 [RTR] OUT spring-music-567.cfapps.io - [19:01:09 +0000] "GET /albums HTTP/1.1" 200 4669  
2014-04-11T15:03:35.16 [DEA] OUT Starting app instance (index 1) with guid 018.70d-e84f05375859  
  
2014-04-11T15:03:40.43 [App/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52  
2014-04-11T14:58:54.47 [App/1] OUT ...]  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.26 [App/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)  
2014-04-11T14:58:56.38 [App/1] ERR INFO: Initializing Spring Environment for Cloud Foundry  
2014-04-11T14:58:57.44 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.coyote.AbstractProtocol start  
2014-04-11T14:58:57.44 [App/1] ERR INFO: Starting ProtocolHandler ["http-bio-62773"]  
2014-04-11T14:58:57.45 [App/1] ERR Apr 11, 2014 6:58:57 PM org.apache.catalina.startup.Catalina start  
2014-04-11T14:58:57.45 [App/1] ERR INFO: Server startup in 8713 ms  
  
2014-04-11T15:03:35.15 [API] OUT Updated app with guid 018f9a20-f0c9-4b5a-970d-e84f05375859 {"instances"=>2}
```

Instance #1 startup activity

# Diego Specific Logging

- The *Execution Agent* under Diego is called a **Cell**
  - Equivalent to a DEA, more capable
- Logging identifiers change too
  - CELL = logging from a Cell
  - APP = logging from an application running in a Cell
  - HEALTH = new health info generated by Cell

```
2015-04-11T12:58:35.16 [CELL] OUT Starting app instance (index 1) with guid 018.70d-e84f05375859
2015-08-31T12:58:40.43 [APP/1] ERR INFO: Starting Servlet Engine: Apache Tomcat/7.0.52
2015-08-31T12:58:54.47 [APP/1] OUT ...
2015-08-31T12:58:56.26 [APP/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)
2015-08-31T12:58:56.26 [APP/1] OUT Hibernate: insert into Album (albumId, artist, trackCount, id) values (?, ?, ?, ?, ?, ?, ?)
2015-08-31T12:58:56.26 [HEALTH/1] OUT healthcheck passed
```

# Accessing Logs

- Tailing – provides *real time* tail of output from loggregator
  - `cf logs <app_name>`
  - Live feed
  - Use `<ctrl> c` to exit
- Recent activity
  - `cf logs <app_name> --recent`

# Direct Access to Files / Logs

- Log files can be accessed directly via CLI
  - `cf files <app_name> <path_to_directory_to_view>`

```
> cf files spring-music /  
app/  
logs/  
run.pid  
staging_info.yml  
tmp/
```

Second instance

- View contents of a specific file

```
> cf files spring-music /logs/stdout.log -i 1  
20:55:27,179  INFO ContextLoader:272 - Root started  
20:55:27,878  INFO :67 - Found serviceInfos:  
20:55:27,879  INFO :513 - Refreshing Root  
...
```

# Eclipse Remote Systems View

Applications

Applications

Select a currently deployed application to see details.

cf-profile-test  
 cf-service-broker

General

Name: cf-profile-test [Started]  
Mapped URLs: [cf-profile-test-dev.cfapps.io](#)  
Memory limit: 512M ▾ Change is not updated until application restarts  
Instances: 1 ▾  
 Stop Restart

Application Services

Name	Vendor	Plan	Version

▼ Instances

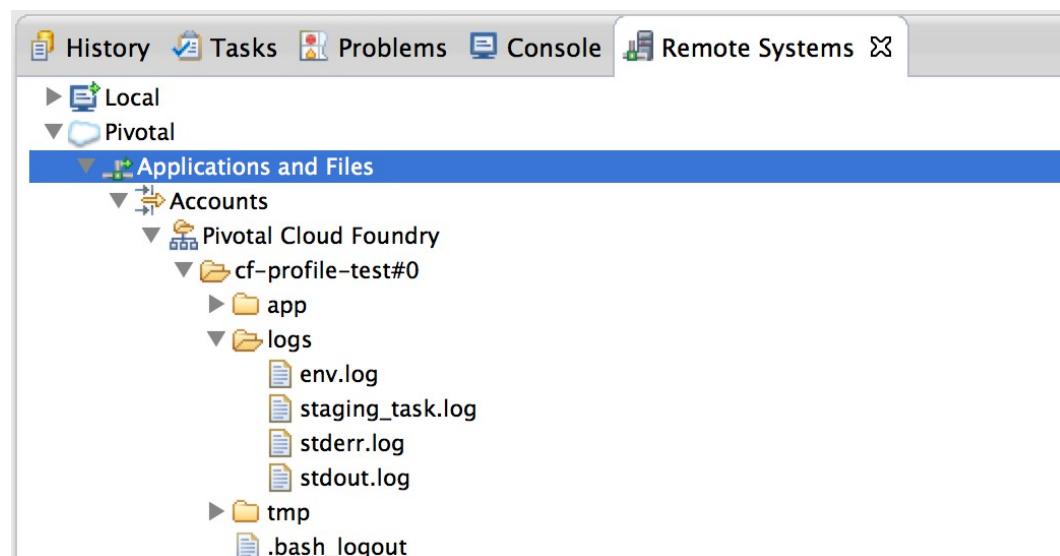
ID	Host	CPU	Memory	Disk	Uptime
0	10.10.17.11	0.0% (0)	0M (512M)	102M (1024M)	0h:0m:0s

Show deployed files in [Remote Systems View](#).

Pivotal™

# Eclipse Remote Systems View – cont.

- Eclipse plug-in for Cloud Foundry
  - Remote Systems View
  - Access logs



# Roadmap

- Logging
- **Troubleshooting**
- How Deployment Happens

# Troubleshooting applications in Cloud Foundry

- Troubleshooting Applications in Cloud Foundry can be hard
  - Deployment / push issues
  - Runtime issues
- Important to be familiar with the tools at your disposal

# cf apps command

- **cf apps** - Lists all applications within a space
  - ... and their *status*
- **cf app <app-name>** - Lists all instances of an application within a space
  - ... and their *status*

```
> cf app spring-music
...
   state      since            cpu    memory          disk
#0  running   2014-04-14 03:27:02 PM  0.1%  407.1M of 512M  134.8M of 1G
#1  starting  2014-04-14 09:12:37 PM  0.0%  0 of 0          0 of 0
#2  running   2014-04-14 09:12:55 PM  0.0%  415.9M of 512M  134.8M of 1G
#3  running   2014-04-14 09:12:54 PM  0.0%  415.7M of 512M  134.8M of 1G
```

# Events

- `cf events <app-name>`
  - Event log related to your application

```
> cf events spring-music
```

time	event	description
2014-04-01T16:33:37.00-0400	audit.app.create	instances: 1, state: STOPPED
2014-04-01T16:33:38.00-0400	audit.app.update	state: STARTED
2014-04-01T16:34:17.00-0400	audit.app.update	instances: 1, state: STOPPED
2014-04-01T17:03:45.00-0400	audit.app.update	state: STARTED
2014-04-01T19:30:09.00-0400	audit.app.update	state: STOPPED
2014-04-10T17:56:07.00-0400	audit.app.update	state: STARTED
2014-04-10T17:56:11.00-0400	audit.app.update	state: STOPPED
2014-04-10T17:58:48.00-0400	audit.app.update	state: STARTED
2014-04-11T14:58:41.00-0400	audit.app.update	instances: 2
2014-04-11T15:03:35.00-0400	audit.app.update	state: STOPPED
2014-04-11T15:16:28.00-0400	audit.app.update	state: STARTED
2014-04-11T16:55:17.00-0400	audit.app.update	instances: 4
2014-04-14T21:12:40.00-0400	audit.app.update	

# Recent Logs

- Obtain most recent subset of logs
  - Very useful when diagnosing a crash
  - `cf logs <app_name> --recent`

# CF\_TRACE

- Set this environment variable
  - On our *own* machine (*not* a Cloud Foundry variable)
  - Detected by the `cf` utility causing it to generate extra debugging output
  - Displays REST messages between `cf` and Cloud Foundry
- Usage

```
export CF_TRACE=true  
cf push ...
```

MacOS  
/ Linux

```
export CF_TRACE=/path/to/log/file  
cf push ...
```

MacOS/Linux

```
set CF_TRACE=true
```

MS Windows

```
set CF_TRACE=\path\to\log\file
```

MS Windows

# Common CF Deployment Issues - Java

- Memory set too low
  - Warden container fails due to memory, discarded, new container created, fails again, restarted, fails ...
- JAR / WAR too large, upload fails
  - Resolve by:
    - push the JAR / WAR only
    - Decompose application
- Slow start, health check fails
  - Specify Tomcat `BindOnInit = false`
  - Alter timeout settings (explained later...)

Known as  
FLAPPING

# Best Practices - Ruby

- Precompile assets
  - Recommended to reduce staging time
- Use `rails_serve_static_assets` gem (Rails 4)
  - Sets `config.serve_static_assets = true`
  - Allows populating of CDN, or serving files directly from application
- Don't push unnecessary files
  - List them in `.cignore`
  - `cf` should ignore `.cignore!`

```
.cignore  
.gitignore  
_darcs  
.DS_Store  
.git  
.hg  
.svn  
/manifest.yml  
/tmp  
/logs
```

# Best Practices – Node.JS

- **package.json mandatory**
  - For configuration
- Again. don't push unnecessary files
  - Use `.cfignore`
- Specify application's web start command
  - Using a Procfile
  - Or `cf push -command "node myapp.js"`
  - Or Cloud Foundry deployment manifest
    - (covered later)

```
{  
  "name": "myapp",  
  "version": "0.0.1",  
  "author": "Your Name",  
  "dependencies": {  
    "express": "3.4.8",  
    "consolidate": "0.10.0",  
    "express": "3.4.8",  
    "swig": "1.3.2"  
  },  
  "engines": {  
    "node": "0.12.2",  
    "npm": "2.7.4"  
  }  
}
```

# Controlling Timeouts



- When a new application is pushed, timeouts may occur
  - `cf push` after 60 secs
  - Application staging after 15 mins
  - Application start-up after 5 minutes
- Can specify push timeout in *seconds* using `-t` option
  - `cf push -t 120`
  - maximum timeout is 180s
- Use environment variables to control staging or startup
  - `CF_STAGING_TIMEOUT` and `CF_STARTUP_TIMEOUT`
  - specify time in *minutes*

# What about IDE Debugging?

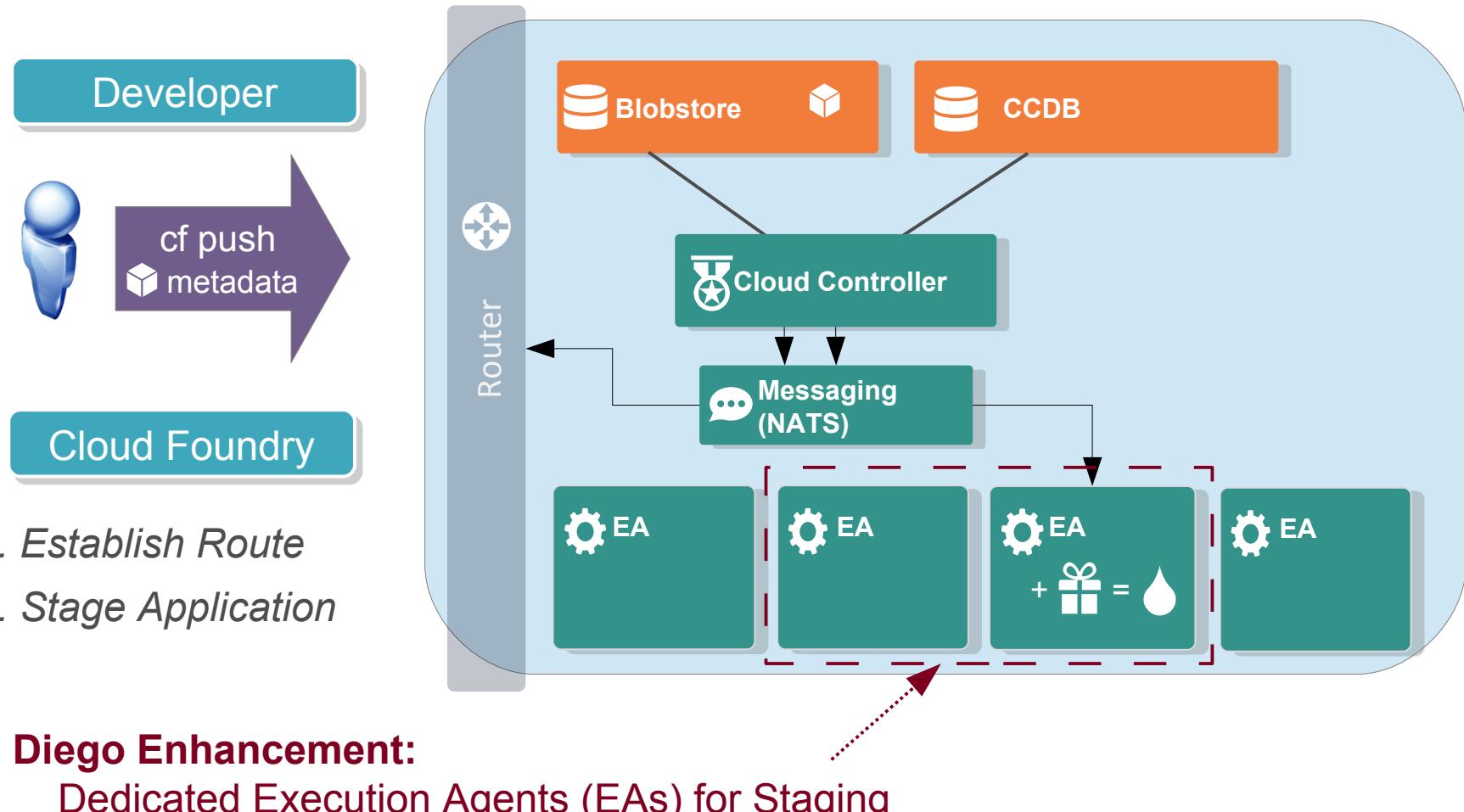
- Connecting an IDE to an application running on Cloud Foundry is basically not possible
- Cloud Foundry restricts incoming traffic
  - Only HTTP on port 80 & 443
- Work-arounds are possible
  - Described in later module
- Recommendation: debug applications locally, rely on techniques just described for troubleshooting.

# Roadmap

- Logging
- Troubleshooting
- **How Deployment Happens**

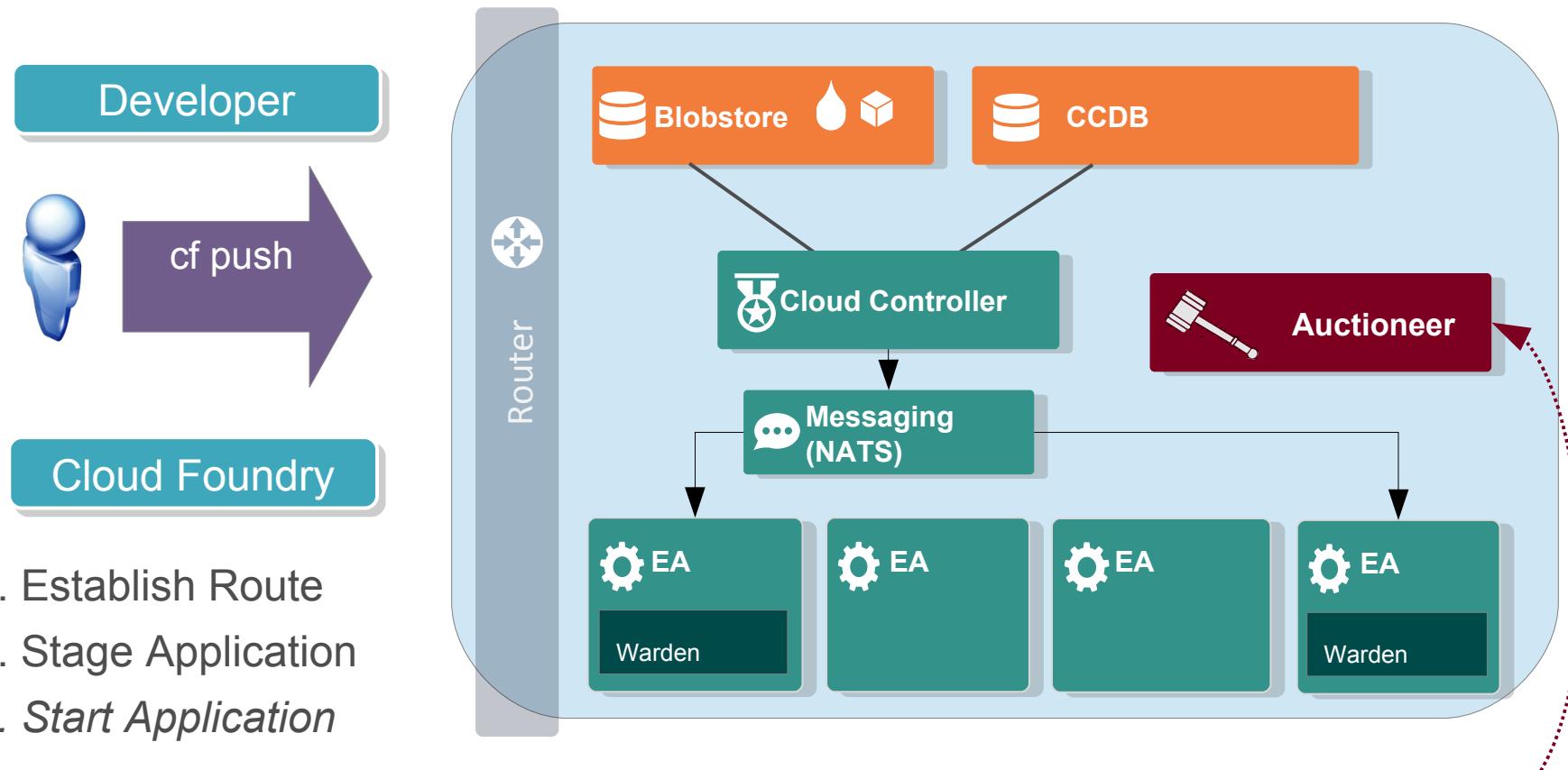
# Deploying App to Cloud Foundry

## Push & Staging



# Deploying App to Cloud Foundry

## Starting



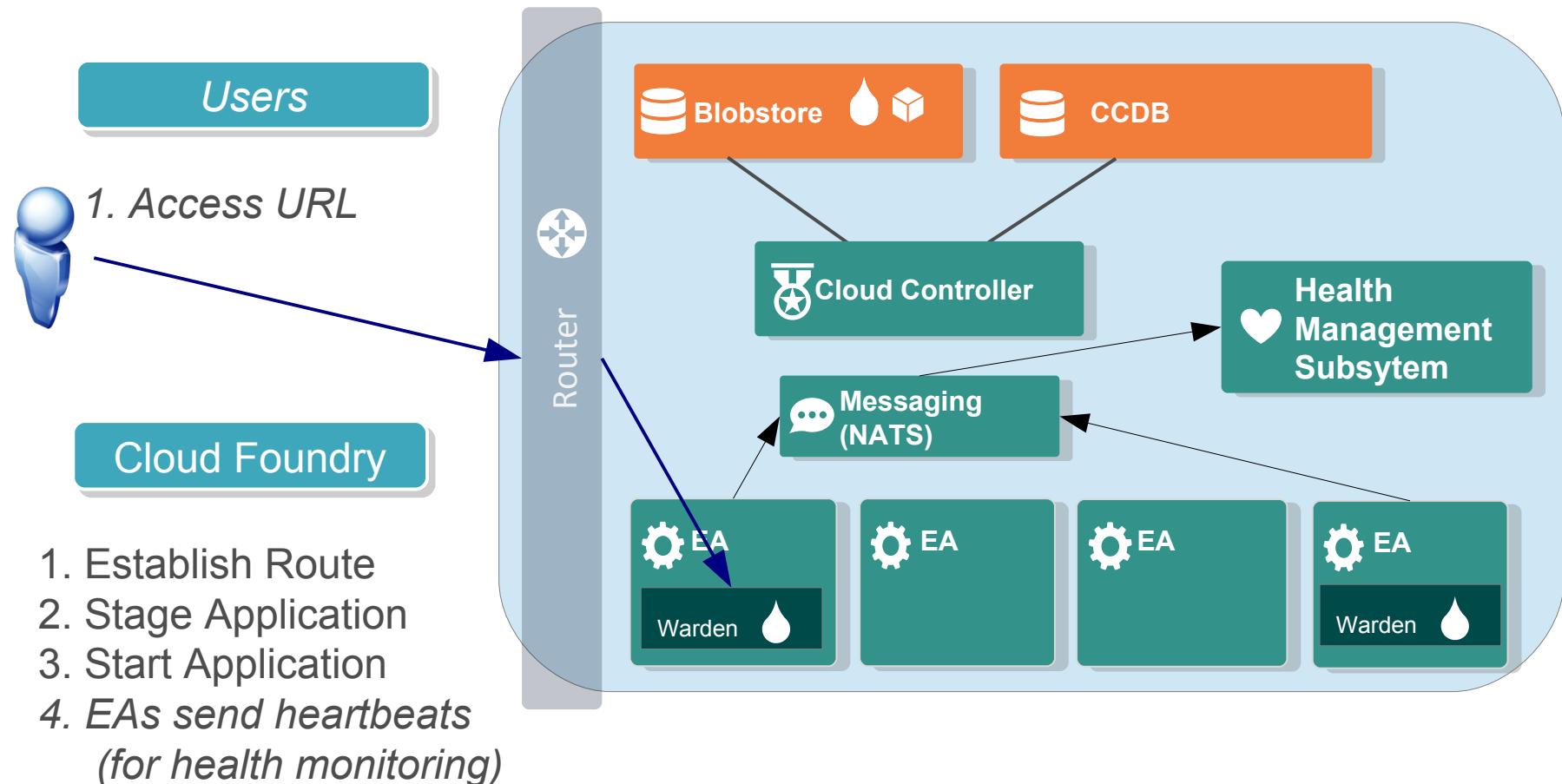
1. Establish Route
2. Stage Application
3. Start Application

### Diego Enhancement:

Adds *Auctioneer* component to pick “best” EAs for better load balancing

# Deploying App to Cloud Foundry

## Health Management and Running



# Summary

- After completing this lesson, you should have learned:
  - How to log and troubleshoot your application
  - How deployment happens

# Lab

Troubleshooting application problems  
with Cloud Foundry



# Manifests, Environment Variables, and Scaling

A closer look at practical Cloud Foundry usage

Manifests, Environment Variables,  
and Scaling

Pivotal

# Overview

- After completing this lesson, you should be able to:
  - Define a Manifest
  - Set Environment Variables
  - Understand how Scaling works

# Roadmap

- **Manifest Files**
- Environment Variables
- Scaling



# Cloud Foundry Manifest file

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

Create using  
your favorite  
text editor

\* *YAML Ain't Markup Language*



# Using a Manifest with Push

- `cf push` automatically detects manifest
  - In current directory or parent directories
  - Expects file named `manifest.yml`
  - Override with `-f` option
    - `cf push -f dev-manifest.yml`
  - Or ignore with `--no-manifest` option.
- No manifest found?
  - `cf push` will default all deployment options
    - Not the best choices
    - *Different* to previous version of `cf` (which prompted)



# YAML Format

- 3 dashes
  - Indicate start of document
- Indent with spaces, not tabs!
  - Determines hierarchy
  - Each indent 2 spaces
  - “-” defines “group”
- Syntax: **property: value** pairs
- # starts a one line comment
- See <http://www.yaml.org/spec/1.2/spec.html>

```
---  
applications:  
- name: nodetestdh01  
  memory: 64M  
  instances: 2  
  host: crn      # comment  
  domain: cfapps.io  
  path: .  
    # comment  
  
- name: nextapp    # group 2  
  memory: 256M  
  ...
```

# manifest.yml Example



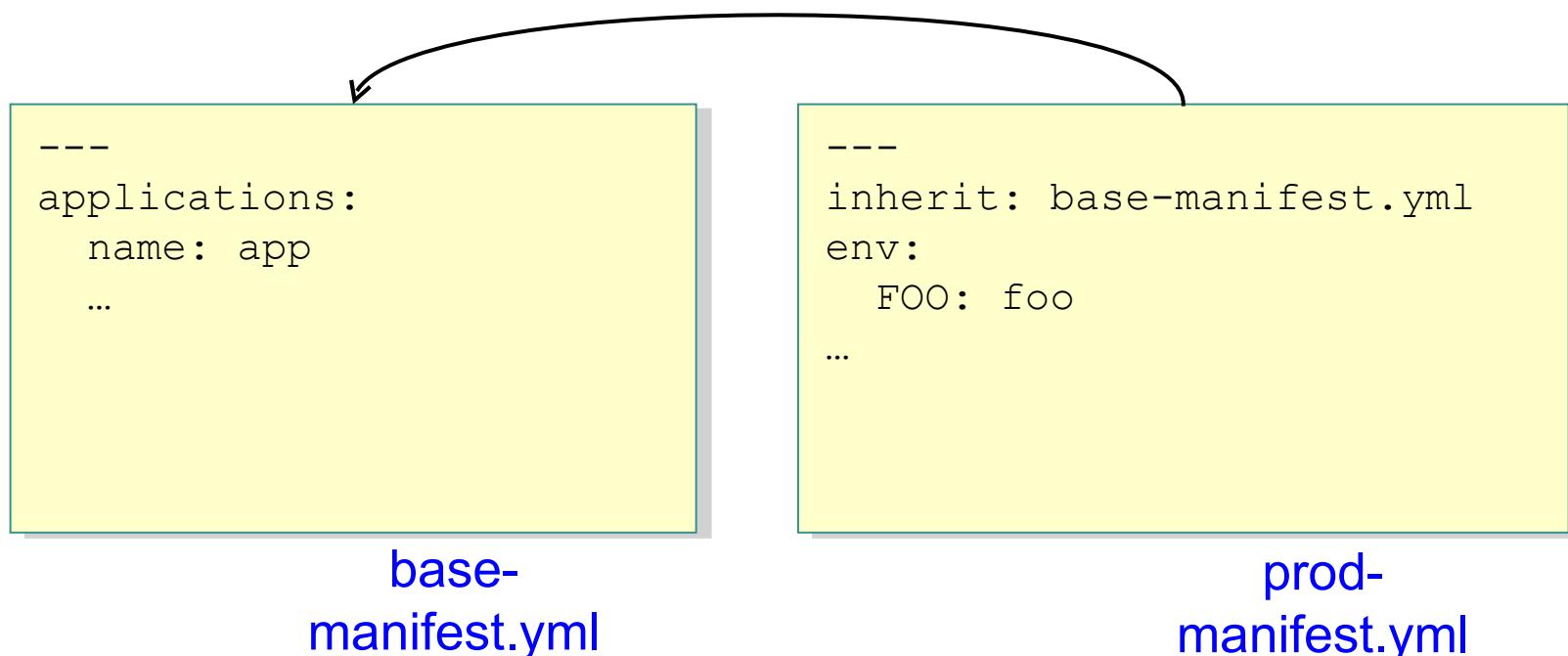
```
---  
applications:  
- name: cf-node-demo  
  command: node app.js  
  memory: 128M  
  instances: 1  
  host: demo-${random-word}  
  domain: cfapps.io  
  path: .
```

- Applications: can describe one or more applications
- Name: of the app – used in commands
- Command: command to run (optional)
- Memory ceiling / instances to run
- Host: your choice, must be unique (within domain)
  - Tip: \${random-word}
- Path: to executable



# Manifest Inheritance

- You may wish to have multiple manifests for an App
  - Different manifests for each space
- One manifest can “inherit” from another



# Specifying Timeouts

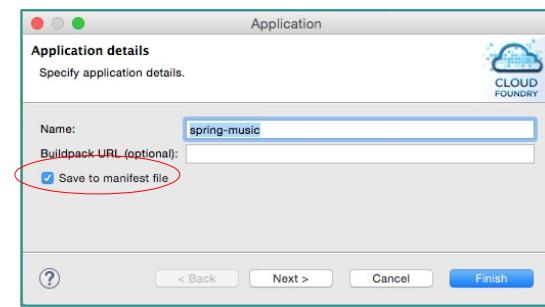


- Can use the manifest to specify timeouts for push, staging or startup
  - When you know your application is too large for the defaults

```
---  
applications:  
- name: nodetestdh01  
  memory: 64M  
  instances: 2  
  host: crn  
  domain: cfapps.io  
  path: .  
  timeout: 120 # 120 secs  
  env:  
    # Set to 20 and 10 mins  
    CF_STAGING_TIMEOUT: 20  
    CF_STARTUP_TIMEOUT: 10
```

# Manifests and STS – I

- STS users can create a manifest
  - Checkbox during push
  - Or after an application has been pushed
    - Applications tab of Server properties

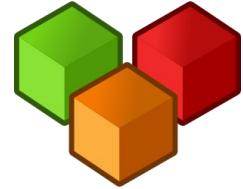


A screenshot of the STS interface showing the 'Applications' tab. The left pane lists deployed applications, with 'spring-music' selected. The right pane shows the application details. In the 'General' section, there is a 'Save' button which is circled in red. Other settings include 'Name: spring-music', 'Mapped URLs: spring-music-  
pc1.cfapps.io', 'Instances: 2', and 'Manifest:'. Below the general settings are sections for 'General (Application Restart Required)', 'Application Operations' (with 'Start', 'Stop', 'Update and Restart', 'Push', and 'Debug' buttons), 'Application Services' (empty table), and 'Instances' (empty table). A note at the bottom says 'Show deployed files in [Remote Systems View](#)'.

# Manifests and STS – II

- Manifest created using same info as original push
  - Note { } in syntax (valid but optional)
- **WARNING:**
  - **STS does not use manifest when pushing**
  - **Even if it has generated one**

```
---  
applications:  
- name: dblsmvc  
  memory: 512M  
  host: dbl-spring-mvc-demo  
  domain: cfapps.io  
  env: {}  
  services: {}
```



# Note on Spaces

- A Space cannot be specified in a manifest file
  - Set first: `cf target -s development`
- To determine the current space
  - Just run: `cf target`



# Manifest vs CLI

- A manifest reduces the amount of typing when deploying via CLI
  - Purpose is to make deployment easily *repeatable*
- Options specified via CLI override options specified via manifest
  - Example: **cf push my-app -i 8 -m 1024M**
  - Deploys 8 instances with 1024 M limit each, regardless of manifest settings

# Roadmap

- Manifest Files
- **Environment Variables**
- Scaling

# Environment Variables

- Key / value pairs
  - Used for anything you like
  - Specify via manifest:

```
---  
env:    # global, all apps  
  spring_profiles_active: dev  
  another_variable: foo  
applications:  
...
```

```
---  
applications:  
- name: myapp  
  memory: 256M  
  instances: 1  
  host: crn  
  domain: cfapps.io  
  env:    # this app only  
    spring_profiles_active: dev  
    another_variable: foo
```

- Or via command line
  - `cf set-env <app-name> <env-var-name> [<value>]`
  - Requires re-staging (i.e. `cf restage` or `cf push`) to take effect
- Or use App Manager or Eclipse plug-in

# Environment Variables - Precedent

- Environment variables via manifest take precedent over CLI
  - Opposite from the push options defined earlier!
- Example:
  - `cf set-env app FOO fromCLI`
  - `cf push`
  - Result? 'fromManifest'!
- Use `cf push app --no-manifest` to bypass manifest values.

```
---  
env:  
  FOO: fromManifest  
applications:  
  name: app
```

# Environment Variables - Persistence

- Environment variables retain their values
  - Whether application is running or not.
- To view use: **cf env <app>**
- Use **cf unset-env <app> <var>** to remove
- If changed while app is running
  - Use **cf restage <app>** to make change take effect

# Environment Variables - Accessing

- CF environment variables are available to applications
  - Appear like any other environment variable
- Access via...
  - Java: `System.getenv("some_variable");`
  - Ruby: `ENV['some_variable']`
  - Node.js: `process.env.some_variable`

# Environment Variables - VCAP\_APPLICATION

- Information on memory, instances
  - JSON formatted object (described later):

```
{"instance_id": "451f045fd16427bb99c895a2649b7b2a",
 "instance_index": 0,
 "host": "0.0.0.0",
 "port": 61857,
 "started_at": "2013-08-12 00:05:29 +0000",
 "started_at_timestamp": 1376265929,
 "start": "2013-08-12 00:05:29 +0000",
 "state_timestamp": 1376265929,
 "limits": {"mem": 512, "disk": 1024, "fds": 16384},
 ...
}`
```

# Environment Variables - VCAP\_SERVICES

- Information on all bound services
  - JSON formatted object (described later):

```
{ "elephantsql": [ { "name": "elephantsql-c6c60", "label": "elephantsql-n/a", "tags": [ "postgres", "postgresql", "relational" ], "plan": "turtle", "credentials": { "uri": "postgres://PHxTPn@babar.elephantsql.com:5432/selmbd" } }, { "name": "mysendgrid", "credentials": { "username": "QvsXMBJ3rK", "password": "HCHMOYluTv" } } ] }
```

# Integration with Spring

- **spring\_profiles\_active**
  - “Activates” a profile in a Java/Spring application.

```
---  
env:  
  spring_profiles_active: dev  
  another_variable: foo  
applications:  
  ...
```

# Environment Variables - Management

- Not all environment variables can be changed
  - Those set by CF runtime cannot
    - Such as **VCAP\_SERVICES** (set by service binding)
    - See URL below for list of variables set by runtime
- To view environment variables:
  - **cf env [app-name]**
    - Displays user defined and system defined variables
    - Some variables only available to running instances

<http://docs.pivotal.io/pivotalcf/devguide/deploy-apps/environment-variable.html>

# Roadmap

- Manifest Files
- Environment Variables
- **Scaling**

# Scaling Applications

- Allows updating application to adjust to changes in load
  - Update instances (horizontal)
  - Update memory (vertical)
- Done from CLI, Eclipse or Apps Manager
- From CLI
  - `cf scale <app-name>`
  - Reports current scaling
  - `cf scale <app-name> -i 4 -m 512M -k 1G`
  - Make 4 instances, each with 512 Meg memory, 1 Gig disk space.



# Applications and Resource Limits

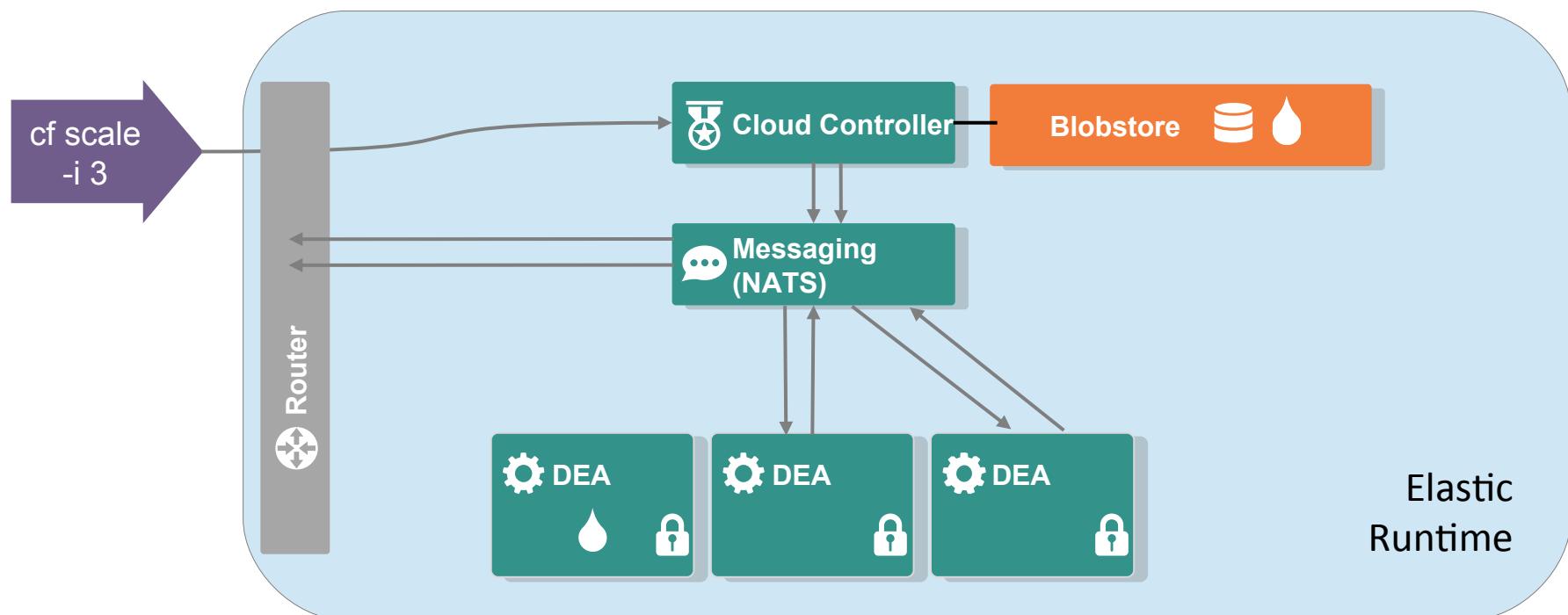
- When an application is pushed/scaled
  - Disk and memory limits are specified
- Application may not exceed its limits
  - Will fail instead
  - If limits exceeded at start-up: ***flapping***
    - Repeating sequence of fail & restart, until timed out
- Quotas limit disk and memory usage by *all* applications in a space and/or an organization
  - Attempt to push/scale-up refused

# How is CPU Allocated?

- Memory and local-disk limits can be set, what about CPU?
- “Fair-share” CPU allocation based on memory allocation
  - Each *Execution Agent* (EA) has 256 “shares” of CPU
    - Regardless of memory or cores
    - Allocation =  $256 * \text{container-memory-limit} / \text{total-EA-memory}$
    - 1 share is the minimum, regardless of memory
- Example
  - 1G application: `cf push myapp -m 1G`
  - Container size = instance size = 1G
  - EA size = 32G (set by Ops)
  - CPU shares = 8 ( $256 * 1 / 32 = 8$ )

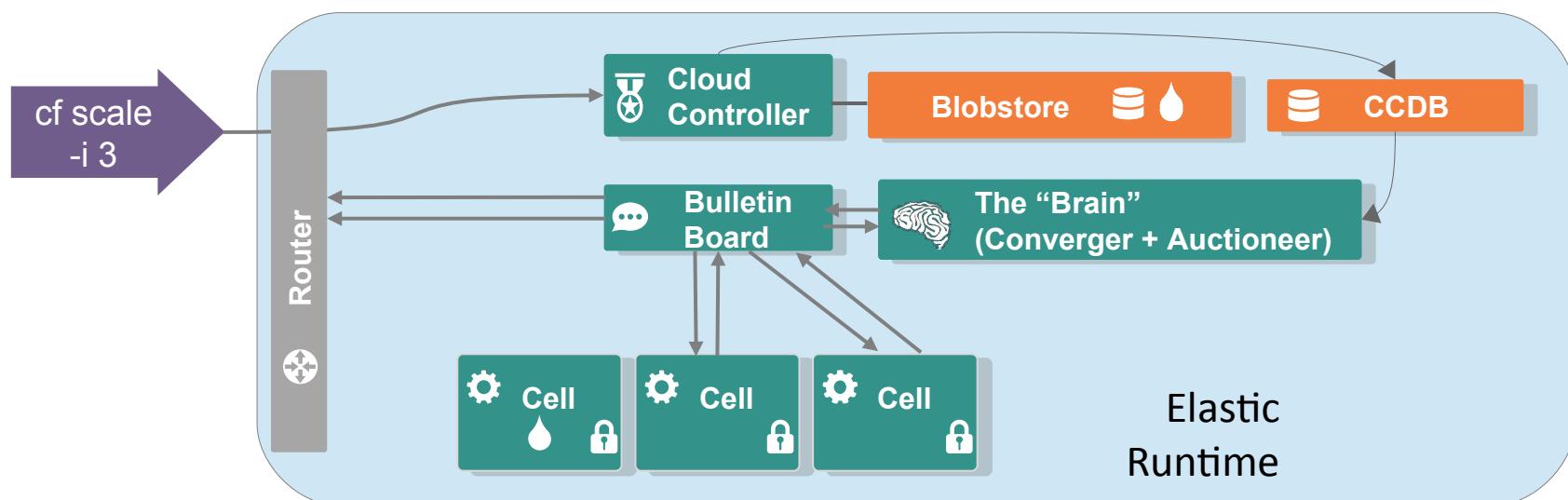
# Scaling an Application (DEA)

- Cloud Controller starts new instances or stops surplus instances
  - Load-balancing router shares load across all instances



# Scaling an Application (DIEGO)

- Cloud Controller indicates instances should be started or stopped
  - Converger adds tasks to start/stop instances to central Bulletin Board
  - Auctioneer determines the least busy Cells to use



# What if I re-push?

- Subsequent **cf push** commands override the number of instances
  - Based on 1) CLI and 2) manifest
  - If CLI/Manifest do not specify instances, previous values stand
  - Application restarted
- Recommendation:
  - Use manifest to store 'default' scaling settings
  - OR omit scaling settings from the manifest.



# File System Implications

- Application instances (Droplets) run in isolated Warden containers
  - Own resources, non-shared – such as files
  - *For safety and security*
- When an application is scaled up
  - New container, new isolated file-system
- When application ends (stopped, failed or scaled down)
  - Local file-storage is destroyed with container
- **Implication**
  - *Do not rely on files: transient and not shareable*

# Summary

- After completing this lesson, you should have learned:
  - Use of environment variables
  - All about application manifests
  - How scaling happens

# Lab

Using Environment Variables and  
Scaling Applications



# Cloud Foundry Services

## Introduction and Overview

What is a Service?

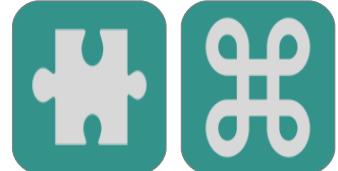
What Services are available?

# Roadmap

- **Services**
- Managed (Marketplace) Services
- Pivotal CF Standard Services
- PWS and App Direct Services

# Cloud Foundry Services

## What is a service?



- **Service** is an **external** application **dependency** or **component** such as
  - Database
  - Message Queue
  - Monitoring App
  - Security
  - Hadoop instance
  - Generic Service Endpoint (Web Services)
  - Other dependent applications



# Cloud Foundry Services

## Features and Functionality

- Provide *functionality* to your applications
- *External* to your applications
  - Add-on provisioned alongside an application.
- May be *shared* among many applications
  - Example – Relational DB, Messaging system
- Are *bound* to (associated with) an application
  - Using a “Service Broker”
- Provide connection information to application via environment variables
  - **VCAP\_SERVICES**



# Cloud Foundry Services

## Why use a service?

- Applications are the deployment unit
  - Must be self-contained
  - Anything else they need is provided by the PaaS
    - By a service
- Services in a PaaS are
  - One of the main possible charging units / elements
    - Instead of hardware resources like an IaaS
  - Make commercial PaaS possible
  - Enable charge-back in your organization

# Cloud Foundry Services

## Two Types of Services

CF Managed

- Managed Services (a.k.a. “Marketplace” Services)
  - Available 'out-of-the-box'
  - Selected from marketplace 'catalog'
  - Instances provisioned by PaaS, for use by application



User Managed

- User Defined Services

- Services running external to Cloud Foundry
- Connection information stored and used to connect
- PaaS does not provision resources, only supplies connection information.



Pivotal™



# Cloud Foundry Services

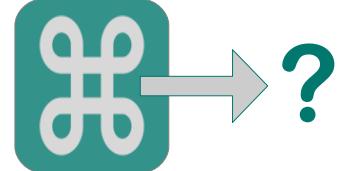
## Custom Services

- Custom Built Services
  - Custom service created and installed into Cloud Foundry
    - Looks like any other managed service once added
    - Appears in the Marketplace
  - Alternative to user-defined services
    - Full integration with CF
- Requirements
  - Custom development of *service broker*
  - And *operator* installation into Cloud Foundry

Also CF Managed

# Cloud Foundry Services

## Accessing External Services



- Typically these exist already
  - Your ERP and CRM systems (Oracle, SAP ...)
  - Mainframe and other legacy applications
  - Applications developed/running in-house
  - Cloud-based services such as sales, CRM or payroll
- Two options
  - User-defined Service
    - Your ops people continue to manage and provision
  - Custom Service
    - CF uses service-broker to provision and bind

# Services Summary

- Question: Within a Cloud Foundry App, How do I...
  - Connect to a relational database?
  - Connect to a messaging system?
  - Connect to an email system?
  - Utilize NoSQL databases?
  - Read and write files
  - Save and retrieve sessions
  - Access anything *not* coded in my application
- Answer: Via Services



# Roadmap

- Services
- **Managed (Marketplace) Services**
- Pivotal Cloud Foundry Standard Services
- PWS and App Direct Services

# Accessing Managed Services

- Easily available via *Marketplace*
  - Allow you to sign-up, select plans, etc
  - Once bound to application, can be used easily
- Many pre-packaged services for Pivotal CF
  - See <https://network.pivotal.io/products>

 <p>MongoDB for Pivotal CF MongoDB Data Store</p>	 <p>MySQL for Pivotal CF MySQL database-as-a-service for Cloud Foundry applications</p>	<p>??? for Pivotal CF More planned ...</p>
 <p>Pivotal CF RabbitMQ Service Give your applications a common platform to safely</p>	 <p>Pivotal HD for Pivotal CF® Gain insight from the massive data captured by apps, syst...</p>	 <p>Redis for Pivotal CF Cloud Foundry Redis service for application development a...</p>

# Managed Services

- Services preconfigured and made available to Cloud Foundry
  - Typically by operations personnel
- On-premise/private cloud
  - Your company controls what is available
  - Services typically run in your data-centre
- Public cloud
  - Cloud provider controlled
  - Services may run *anywhere* – locality considerations



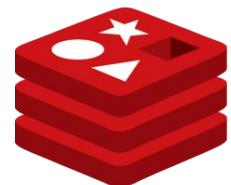
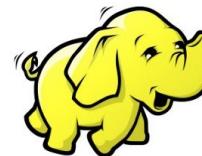
# Roadmap

- Services
- Managed (Marketplace) Services
- **Pivotal CF Standard Services**
- PWS and App Direct Services

# Pivotal CF Services

## What Services are Available

- Whatever your company chooses
  - Services added after Pivotal CF installation by Ops
  - Available as .pivotal files from Pivotal Network
    - See: <https://network.pivotal.io>
- This section discusses several services
  - They may not be available to your private cloud



Pivotal™

# SQL Databases - MySQL



- Free, Open Source Relational Database
  - GPL licensing
  - High-Available, clustered, synchronously replicated using MariaDB Galera Cluster
    - Each node has a copy of each DB
    - Writes to any DB are replicated to all copies
    - Client connections routed to primary, on failure proxy routes to a healthy node
  - Suitable for production use



Pivotal

# NoSQL Data Services



MongoDB for Pivotal CF  
MongoDB Data Store



Neo4j for Pivotal CF  
Neo4j Graph Database



Riak CS for Pivotal CF  
An S3-compatible object store  
for Cloud Foundry applications



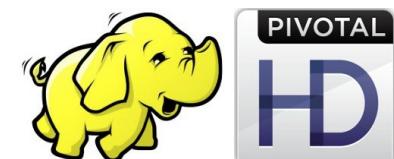
Redis for Pivotal CF  
Redis for Pivotal CF service for  
application development and  
testing.

- MongoDB – Popular Document store
  - Neo4j – Graph Database
  - Riak CS - “Cloud Store” for accessing Amazon S3-like file storage
  - Redis – Popular Key / Value store from Pivotal
- 
- Available Soon: Elasticsearch, Memcached, Gemfire, DataStax (Cassandra)
  - *Not intended for production use*



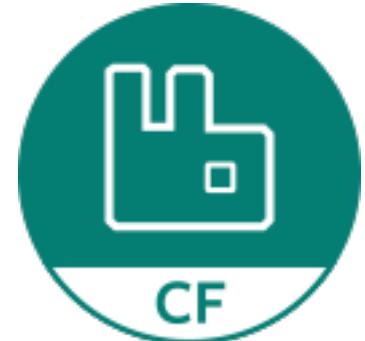
# Pivotal HD for Cloud Foundry

- Directly from Pivotal CF, provision Hadoop resources to power data-centric Cloud Foundry Apps
  - HDFS, Map-Reduce, Hive, HBase, YARN, Zookeeper ...
    - Or connect to existing Hadoop cluster
  - HAWQ: Pivotal's fast, distributed SQL engine
    - Perform deep, complex analytics in SQL (or R, Madlib, etc)
  - Run Hadoop jobs directly from Pivotal CF application
- Runtime service broker integration
  - Instant credential generation & binding to shared PHD cluster



Pivotal™

# RabbitMQ



- Single HA cluster deployment of RabbitMQ
  - Runtime service broker integration
    - Instant credential generation, binding to shared RMQ cluster
    - Unique virtual host per binding
  - Is intended for production use

# Roadmap

- Services
- Managed (Marketplace) Services
- Pivotal Cloud Foundry Standard Services
- **PWS and App Direct**

# Pivotal Web Services (PWS)



- Public Cloud Foundry instance
  - Hosted on AWS
  - Provides extensive marketplace of services via *App Direct*
    - Some free
    - Some pay-per use
  - Examples
    - Postgres DB, MySQL, MongoDB, Redis
    - Rabbit MQ
    - Blazemeter monitoring
    - Many, many more ...

# Pivotal Web Services (PWS)

## Marketplace Home Page in App Manager Console

The screenshot shows the Pivotal Web Services (PWS) Marketplace Home Page within the App Manager Console. The left sidebar displays the organization 'krueger-net' and its spaces: development, production, QA, staging, and Marketplace (which is currently selected). The main content area is titled 'Services Marketplace' and features a grid of six service cards:

- BlazeMeter**: The JMeter Load Testing Cloud. Icon: Yellow 'M' logo.
- ClearDB MySQL Database**: Highly available MySQL for your Apps. Icon: Purple cylinder with green checkmark.
- CloudAMQP**: Managed HA RabbitMQ servers in the cloud. Icon: White rabbit.
- CloudForge**: Development Tools In The Cloud. Icon: Blue cloud with 'CloudForge' logo.
- ElephantSQL**: PostgreSQL as a Service. Icon: Elephant.
- IronMQ**: Powerful Durable Message Queueing Service. Icon: Blue square with white arrow.
- IronWorker**: Scalable Background and Async Processing. Icon: Blue gear.
- Load Impact**: Automated and on-demand performance testing. Icon: Green circular logo.

- Commercial provider of services
  - Provide third-party service *market*
    - Teamed up with well-known providers, like Redis Labs
    - Various plans and fees
  - The provider of services offered by PWS
  - Your company *may* choose to use App-Direct as well
- Services run by *providers*
  - For example a Redis instance would run at Redis Labs
  - *External* to your data-centre
  - Locality issues: performance, connectivity, security, legal

# PWS Cloud-Foundry Services

- Marketplace services in PWS offered via App-Direct

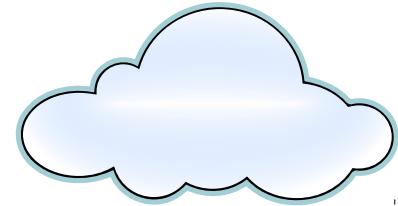


# Running Cloud Foundry *On Premise*

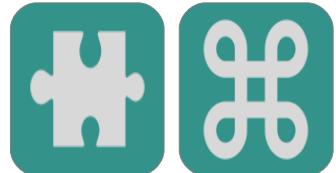


- All services typically run in your data-centre
  - Many may already exist
    - Databases, message-brokers, mail servers ...
  - CF ops decide what services to install and/or make available to applications
- Your company *may* choose to use third-party services
  - For services it does not wish to manage
  - Cloud-based services such as sales, CRM, payroll
  - And/or a Commercial provider like AppDirect

# Using a Public Service



- Services provided for you by your cloud provider
  - For example PWS
  - You have little control over what services are offered
  - Services may or may not be hosted by cloud provider
- Considerations around service location
  - Network reliability
  - Legal jurisdiction of host servers
  - Security



# Services – Summary

- External dependencies your application relies on
  - Connection information provided to application at runtime
- Two types
  - Managed
    - Either predefined (by Pivotal) or Custom (by you)
  - User-Defined
- Many predefined services available
  - MySQL, NoSQL, Mobile, Hadoop, ...
- App Direct provides Marketplace services for PWS
  - You *may* choose to use them in your own CF setup

# What You Have Learned

- Services
- Managed (Marketplace) Services
- Pivotal Cloud Foundry Standard Services
- PWS and App Direct Services



# Cloud Foundry Services

## Creating and Binding Services

Making services available to your applications

# Roadmap

- **Provisioning Services**
- Using the CLI
- Using the Pivotal CF App Manager Console
- Binding to a Service
- User Provided Services

# Service vs. Service Instance

- **Services provision services instances**
  - For example
    - ClearDB service provisions MySQL databases.
    - Offers different plans (fees, SLAs)
  - You may get a dedicated server, or share a multi-tenant server

# Provisioning – Operator View

- Available services depend on CF setup
  - Must be installed and configured by CF Ops
    - Either via Pivotal CF Operator's Console (*Ops Manager*)
    - Using **cf** CLI
    - Or using the BOSH provisioning tool
- Once Ops have deployed a service to your CF instance
  - It appears in the **marketplace**
  - Can be made available to your application = **provisioning**

OPERATOR

***cf create-service-broker***

***cf enable-service-access***

# Service “Tiles” in PCF Ops Manager

*Only installed services appear in the “marketplace”*

The screenshot shows the PCF Ops Manager interface. On the left, a sidebar lists available products: Ops Manager Director (v1.4.0.0), Pivotal Elastic Runtime (v1.4.0.0), Jenkins Enterprise by CloudBees for Pivotal CF (v1.3.8.3), MySQL for Pivotal Cloud Foundry (v1.5.0.0), and Pivotal Ops Metrics (v1.4.0.0). The main area is the Installation Dashboard, which displays five service tiles. From left to right: 1. Ops Manager Director for VMware vSphere® (v1.4.0.0) - icon shows a circle with a dot and a line. 2. Pivotal Elastic Runtime (v1.4.0.0) - icon shows a circle with an infinity symbol. 3. Jenkins Enterprise by CloudBees for Pivotal CF (v1.3.8.3) - icon shows a circle with a gear and a bar chart. 4. MySQL for Pivotal Cloud Foundry (v1.5.0.0) - icon shows a circle with a dolphin. 5. Pivotal Ops Metrics (v1.4.0.0) - icon shows a circle with a gear and a bar chart. Red arrows point from a callout box to the Jenkins and MySQL tiles. The callout box contains the text: "Operations staff import service ‘tiles’, configure them and **Apply Changes** to install."

PCF Ops Manager admin ▾

Available Products < Installation Dashboard No updates

Ops Manager Director  
No upgrades available

Pivotal Elastic Runtime  
No upgrades available

Jenkins Enterprise by CloudBees for Pivotal CF  
No upgrades available

MySQL for Pivotal Cloud Foundry  
No upgrades available

Pivotal Ops Metrics  
No upgrades available

Import a Product

Download PCF compatible products at [Pivotal Network](#)

Recent Install Logs ▾

Apply changes

Operations staff import service “tiles”, configure them and **Apply Changes** to install.

# Provisioning – Developer View

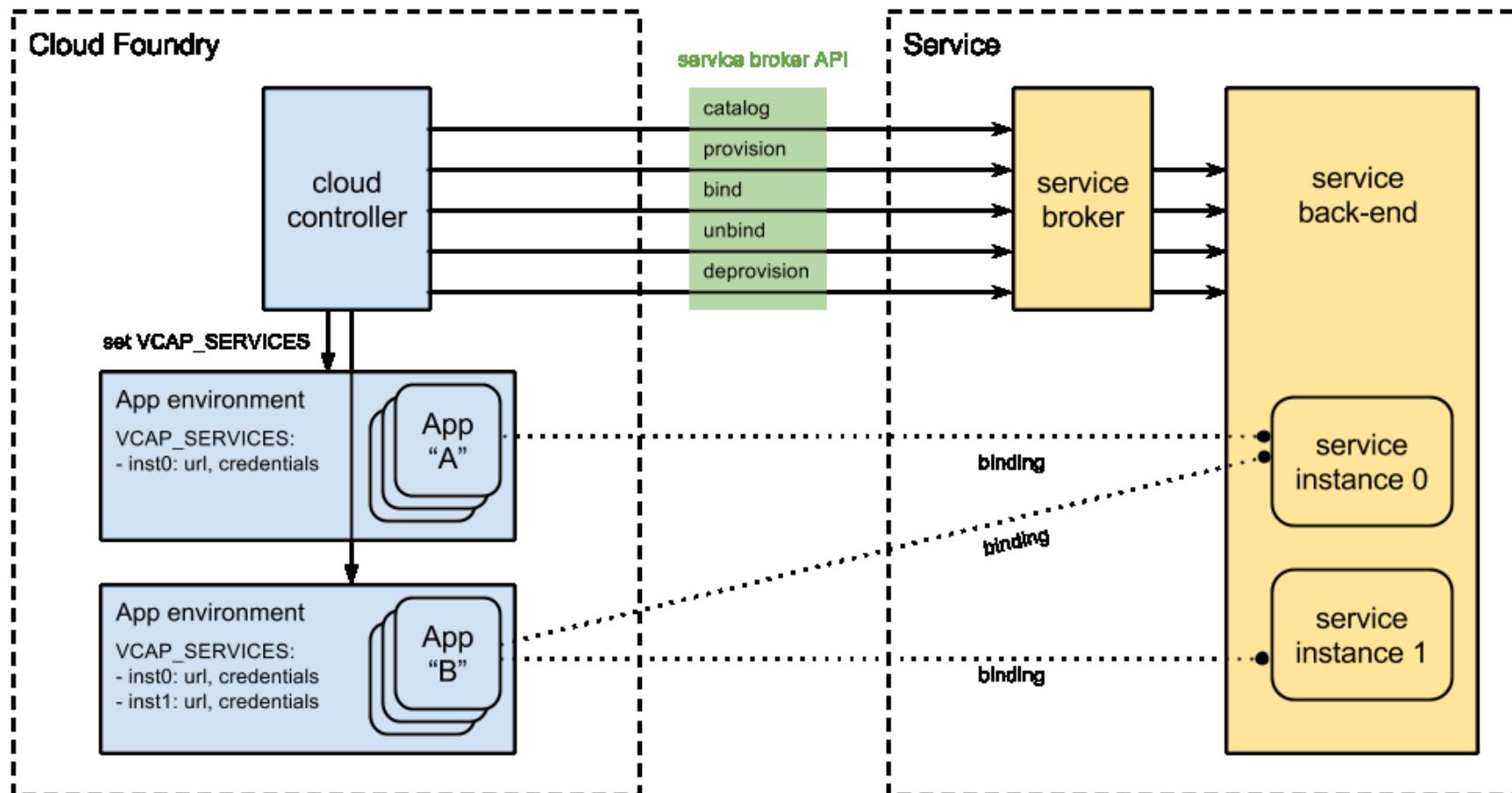
- Only concerned with what a developer has to do
  - Create (provision) a service
  - Bind it to your application

DEVELOPER

`cf create <myService>`

`cf bind <myApp> <myService>`

# Services Overview – Service Brokers



# Creating a Service Instance

- Actually we are *Provisioning* an *instance* of a service
  - It must already exist in CF marketplace
- Use App Manager or **cf create-service**
  - Allows selection of service and plan
- Service instance becomes available to *current space*
  - And any applications running in it
  - For multiple spaces, run **create-service** in each space

# Roadmap

- Provisioning Services
- **Using the CLI**
- Using the Pivotal CF App Manager Console
- Binding to a Service
- User Provided Services

# Finding Available Services

## Command Line Interface

- Check marketplace for available services
  - Essentially a service catalog

```
example$ cf marketplace
Getting services from marketplace in org pivotaledu / space development as user@domain...
OK

service      plans                                         description
blazemeter   free-tier, basic1kmr, pro5kmr, pp10kmr, hv40kmr
cleardb      spark, boost, amp, shock
cloudamqp    lemur, tiger, bunny, rabbit, panda
cloudforge   free, standard, pro
elephantsql  turtle, panda, hippo, elephant
ironmq       pro_platinum, pro_gold, large, medium, small, pro_silver
ironworker   large, pro_gold, pro_platinum, pro_silver, small, medium
...
...
```

# Finding Existing Service Instances

## Command Line Interface

- List existing services instance
  - In *current space*
- In this example: one service instance called mysql

```
example$ cf services
Getting services in org pivotaledu / space development as user@domain...
OK

name          service      plan      bound-apps
mydb          cleardb     spark      booking-app-123
```

- Remember, to change spaces
  - cf target -s [space-name]

# Provisioning a new Service Instance

## Command Line Interface

- Provision a new service instance
  - Added to *current space*
  - Give it a name
  - Choose the correct plan or contract
- Usage
  - `cf create-service [service-name] [plan-name] [instance-name]`

```
example$ cf create-service elephantsql turtle mypg
Creating service mypg in org pivotaledu / space development as user@domain...
OK
```

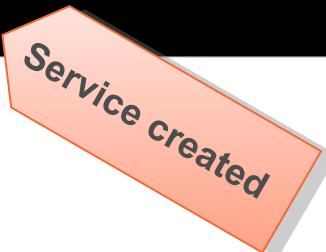
# Finding Existing Service Instances

## Command Line Interface

- List service instances again for *current* space
  - New service instance now appears

```
example$ cf services
Getting services in org pivotaledu / space development as user@domain...
OK

name      service    plan    bound-apps
mydb      cleardb   spark   booking-app-123
mypg      elephantsql  turtle
```



# Roadmap

- Provisioning Services
- Using the CLI
- **Using the Pivotal CF App Manager Console**
- Binding to a Service
- User Provided Services

# Provisioning Service Instances GUI

The screenshot shows the Pivotal Web Services interface for the 'development' space. The left sidebar includes links for Org (pivotaledu), Spaces (development, production, staging), Marketplace, Docs, Support, Tools, Blog, and Status. The main area displays the 'development' space with an applications table and a services table.

**APPLICATIONS**

STATUS	APP	INSTANCES	MEMORY
STOPPED	booking-app-123 <a href="#">booking-app-123.cfapp....</a>	1	1GB <small>[54]</small>

**SERVICES**

SERVICE INSTANCE	SERVICE PLAN	BOUND APPS
<a href="#">Manage</a>   Documentation   Support   Delete	ClearDB MySQL Database spark	1

**Marketplace**

**Services**

# Finding Available Services

## Service Selection

The screenshot shows the Pivotal Web Services Marketplace interface. On the left, a sidebar menu includes 'ORG' (pivotaledu), 'SPACES' (development, production, staging), and 'Marketplace' (selected). Below these are links for 'Docs', 'Support', 'Tools', 'Blog', and 'Status'. The main content area is titled 'Services Marketplace' and features a message: 'Get started with our free marketplace services. Upgrade to gain access to premium service plans.' It lists several services:

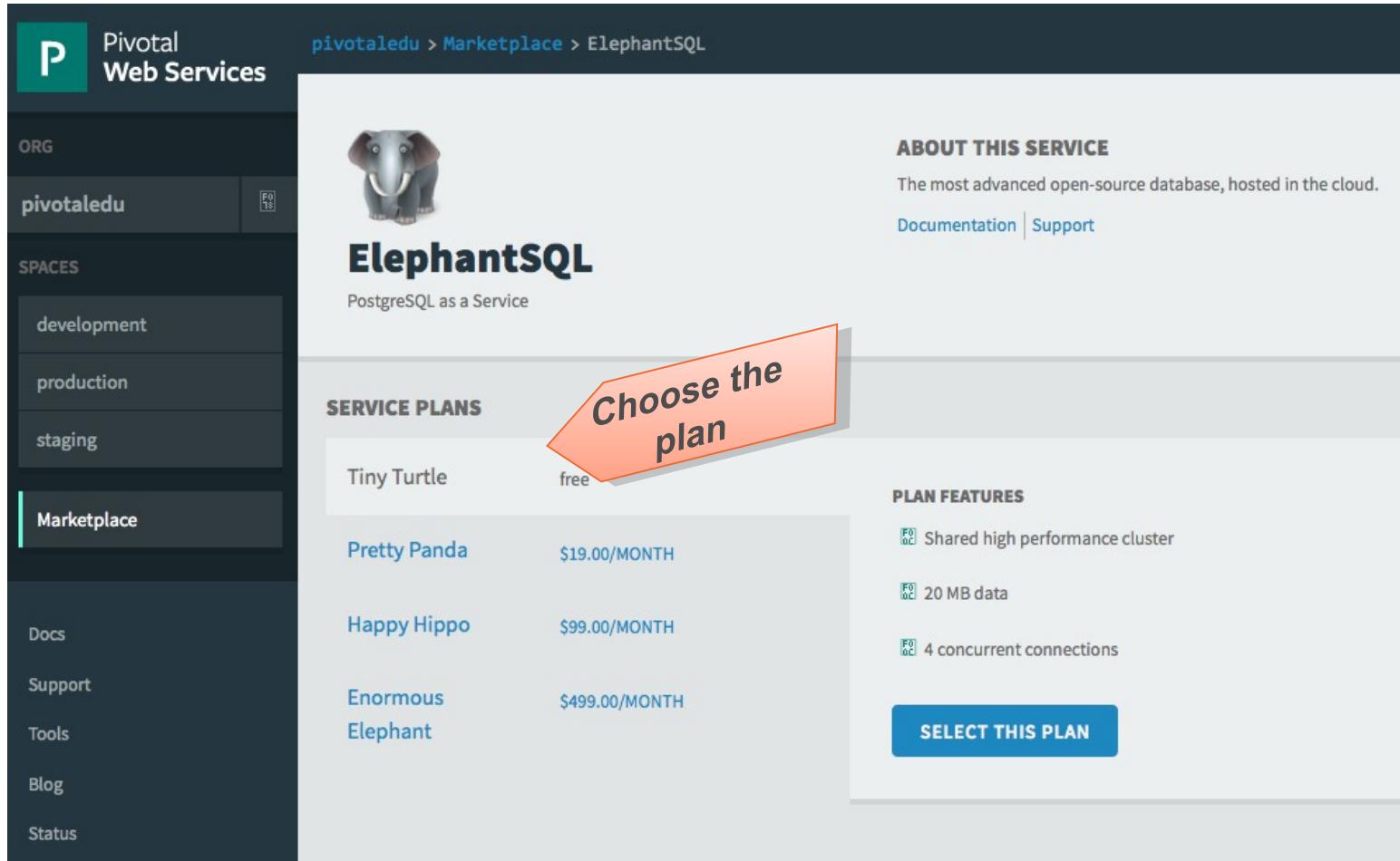
- BlazeMeter**: The JMeter Load Testing Cloud. Includes a yellow logo.
- CloudAMQP**: Managed HA RabbitMQ servers in the cloud. Includes a white rabbit logo.
- ElephantSQL**: PostgreSQL as a Service. Includes an elephant logo.
- IronWorker**: Scalable Background and Async Processing. Includes a blue gear logo.
- ClearDB MySQL Database**: Highly available MySQL for your Apps. Includes a purple database icon with a green checkmark.
- CloudForge**: Development Tools In The Cloud. Includes a blue cloud logo.
- IronMQ**: Powerful Durable Message Queueing Service. Includes a teal arrow logo.

A large orange callout box with the text "Choose to check available plans" points towards the 'VIEW PLAN OPTIONS' button for the ElephantSQL service.

Pivotal™

# Provisioning a new Service Instance

## Pick a Plan



The screenshot shows the Pivotal Web Services marketplace interface. On the left, a sidebar lists organizational units (ORG), spaces (development, production, staging), and links to Docs, Support, Tools, Blog, and Status. The 'Marketplace' link is currently selected. The main content area displays the ElephantSQL service page, which includes a logo of an elephant, the service name 'ElephantSQL' (described as 'PostgreSQL as a Service'), and an 'ABOUT THIS SERVICE' section stating it's 'The most advanced open-source database, hosted in the cloud.' with links to Documentation and Support. A large orange callout bubble with the text 'Choose the plan' points to the 'SERVICE PLANS' section. This section lists four plans: 'Tiny Turtle' (free), 'Pretty Panda' (\$19.00/MONTH), 'Happy Hippo' (\$99.00/MONTH), and 'Enormous Elephant' (\$499.00/MONTH). To the right of the plans is a 'PLAN FEATURES' section with icons for a shared cluster, 20 MB data, and 4 concurrent connections, and a 'SELECT THIS PLAN' button.

Service Plan	Cost
Tiny Turtle	free
Pretty Panda	\$19.00/MONTH
Happy Hippo	\$99.00/MONTH
Enormous Elephant	\$499.00/MONTH

# Provisioning a new Service Instance

## Provision (Create) Service

The screenshot shows the Pivotal Web Services interface. On the left, the sidebar includes sections for ORG (pivotaledu), SPACES (development, production, staging), Marketplace (selected), Docs, Support, Tools, Blog, and Status. The main content area displays the ElephantSQL service details: a cartoon elephant icon, the service name "ElephantSQL", the description "PostgreSQL as a Service", and the company "84codes AB". A "SERVICE PLAN" section shows "Tiny Turtle" as the plan and "free" as the cost. Below this is a "CONFIGURE INSTANCE" form with fields for "Instance Name" (set to "mypyg"), "Add to Space" (set to "development"), and "Bind to App" (set to "[do not bind]"). At the bottom of the form are "SUBSCRIPTION TERMS" with a bulleted list of service terms, and "CANCEL" and "ADD" buttons. Two orange callout boxes are overlaid on the right side: one pointing to the "Instance Name" field with the text "Specify instance name", and another pointing to the "ADD" button with the text "Create the service".

# Provisioning a new Service Instance

## Complete

The screenshot shows the Pivotal Web Services dashboard for the organization 'pivotaledu' and space 'development'. A green banner at the top indicates 'Service instance mypg created.' An orange 'Success' callout points to this message. Below, the 'development' space is selected. The 'APPLICATIONS' section lists one application: 'booking-app-123' (status: STOPPED, instances: 1, memory: 1GB). The 'SERVICES' section shows two service instances: 'spark' (plan: ClearDB MySQL Database spark, instances: 1) and 'mpgsql' (plan: ElephantSQL turtle, instances: 0). An orange 'Service available' callout points to the 'spark' service instance.

APPLICATIONS	LEARN MORE		
STATUS	APP	INSTANCES	MEMORY
STOPPED	booking-app-123 booking-app-123.cfapp...	1	1GB

SERVICES		ADD SERVICE
SERVICE INSTANCE	SERVICE PLAN	BOUND APPS
spark	ClearDB MySQL Database spark	1
mpgsql	ElephantSQL turtle	0

Pivotal™

# Roadmap

- Provisioning Services
- Using the CLI
- Using the Pivotal CF App Manager Console
- **Binding to a Service**
- User Provided Services

# Accessing Service Instances from an App?

## Traditional way

- Traditionally, for an application to access a service instance, connection properties are required
- For example: a database instance
  - Need to know service address / port, credentials
    - such as a JDBC connection
- May be hard-coded, provided through the environment or a configuration file
- Typically service-specific code is required

# Accessing Service Instances from an App?

## Traditional way

- Configuration files

```
development:  
  adapter: mysql2  
  encoding: utf8  
  database: pivotaldb  
  username: pivotal  
  password: pivotal  
  host: myDbHost  
  port: 3306
```

Ruby

```
datasource {  
  driverClassName = "com.mysql.jdbc.Driver"  
  username = "pivotal"  
  password = "pivotal"  
  url = "jdbc:mysql://myDbHost:3306/pivotaldb"  
}
```

Groovy

```
datasource.driverClassName="com.mysql.jdbc.Driver"  
datasource.username="pivotal"  
datasource.password="pivotal"  
datasource.url="jdbc:mysql://myDbHost:3306/pivotaldb"
```

Java

# Accessing Service Instances from an App?

## The CloudFoundry way

- In CloudFoundry, you **bind** the service instance to apps
  - Connection credentials are negotiated / defined for you
  - Application code only needs service name and type/kind
    - Example: a Postgres instance with name “**mypg**”
  - Service details injected into application by CF
    - **VCAP\_SERVICES**
  - Any changes (host/port/credentials) are managed external to the application.

# Example VCAP\_SERVICES Property

```
VCAP_SERVICES=
{
  cleardb-n/a: [
    {
      name: "cleardb-1",
      label: "cleardb-n/a",
      plan: "spark",
      credentials: {
        name: "ad_c6f4446532610ab",
        hostname: "us-cdbr-east-03.cleardb.com",
        port: "3306",
        username: "b5d435f40dd2b2",
        password: "ebfc00ac",
        uri: "mysql://b5d435f40dd2b2:ebfc00ac@us-cdbr-east-
              03.cleardb.com:3306/ad_c6f4446532610ab",
        jdbcUrl: "jdbc:mysql://b5d435f40dd2b2:ebfc00ac@us-
                  cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab"
      }
    ...
  ]
}
```

ClearDB is the MySQL instance offered through App Direct

# Using a Service – Cloud Foundry

## Binding using the CLI

- **Binding** associates an application to a service instance.
  - Use `cf bind-service`
  - Syntax
    - `cf bind-service [app_name] [service_name]`

```
example$ cf bind-service booking-app-456 mypg
Binding service mypg to booking-app-456 in org pivotaledu / space development
as user@domain...
OK
TIP Use 'cf restage' to ensure your env variable changes take effect ←— Note
```

# Using a Service – Cloud Foundry

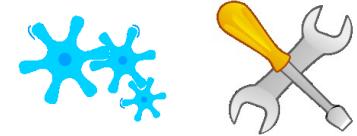
## Binding using a Manifest

- Add a *services* section to your application in the manifest
  - Example [manifest.yml](#)

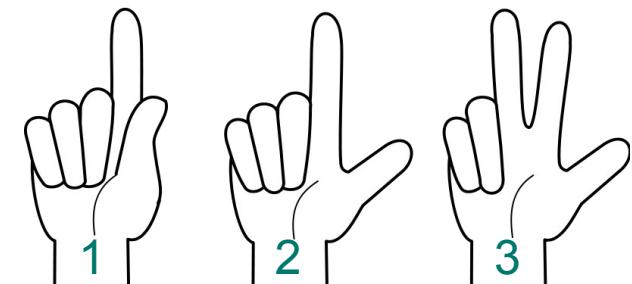
```
---
applications:
- name: booking-app-456
  memory: 256M
  instances: 2
  host: booking-app-456
  domain: cfapps.io
  path: target/booking-app.war
# services, one per line
services:
- mypg
- mydb
```

# Using a Service

## The CloudFoundry way



- Cloud Foundry provides Application with **VCAP\_SERVICES** environment variable
  - Which contains connection details / credentials in JSON.
- How can an application obtain the credentials?
- Three options:
  1. Manual
    - Explicit low-level code
  2. Custom library
    - Explicit code, higher level interface
  3. Auto configuration
    - CF does it for you



# Using a Service – Application View



## 1. Manually

- Manual configuration
  - Access **VCAP\_SERVICES** environment variable
  - In your code, parse the JSON (see next slide)
- Very low-level but works in most languages
  - Fall-back option when options 2 and 3 aren't possible

# Recall: VCAP\_SERVICES Property

```
VCAP_SERVICES=
```

```
{  
  cleardb-n/a: [  
    {  
      name: "cleardb-1",  
      label: "cleardb-n/a",  
      plan: "spark",  
      credentials: {  
        name: "ad_c6f4446532610ab",  
        hostname: "us-cdbr-east-03.cleardb.com",  
        port: "3306",  
        username: "b5d435f40dd2b2",  
        password: "ebfc00ac",  
        uri: "mysql://b5d435f40dd2b2:ebfc00ac@us-cdbr-east-  
              03.cleardb.com:3306/ad_c6f4446532610ab",  
        jdbcUrl: "jdbc:mysql://b5d435f40dd2b2:ebfc00ac@us-  
                  cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab"  
      }  
    }  
  ...  
}
```

Just a very long string in  
JSON format

Parse to extract  
these credentials



# Using a Service – Application View

## 2. Custom Library

- Avoid manual parsing using a cloud-aware library
  - Cloud foundry aware helper code
    - Language/framework dependent
    - Parses **VCAP\_SERVICES** for you
  - JVM: use Spring Cloud project
  - Node.js: use *cfruntime* object

Derived from  
**VCAP\_SERVICES**

```
for (ServiceInfo service : cloud.getServiceInfos() ) {  
    if (service instanceof MysqlServiceInfo)  
        connectionURI = ((MysqlServiceInfo)service).getJdbcUri();  
    } ...
```

Java Example



# Using a Service – Application View

## 3. Auto-configuration

- Cloud Foundry creates the service connection for you
  - Not always supported, depends on:
    - 1) The buildpack
      - Some buildpacks support auto-configuration, others do not.
    - 2) The framework
      - Spring, Grails, Lift, Rails currently supported.
      - NOT (currently) supported for Node.js, Sinatra, Rack ...
    - 3) The uniqueness of the service type
      - For example, can auto-configure ONE database connection
        - CF doesn't know which is which if there are two or more

# Accessing Connection Information

- Recall
  - Connection information once bound is in **VCAP\_SERVICES**
  - Every application's environment is logged at startup
- Once application is *staged*, view connection information using
  - *cf env [app-name]*
  - Look for **VCAP\_SERVICES** in the output

# Accessing Connection Information - 2

- Connection information also available via App Manager:

BOUND SERVICES [+ Bind a Service](#)

 mydb ClearDB MySQL Database Spark DB <a href="#">▼ Hide credentials</a>	<a href="#">Manage</a> <a href="#">Support</a> <a href="#">Docs</a>
JDBCURL	jdbc:mysql://b4eb8a837a231d:c3b28eec@us-cdbr-east-06.cleardb.net:3306/ad_d691ab771b6397e
URI	mysql://b4eb8a837a231d:c3b28eec@us-cdbr-east-06.cleardb.net:3306/ad_d691ab771b6397e?reco
NAME	ad_d691ab771b6397e
HOSTNAME	us-cdbr-east-06.cleardb.net
PORT	3306
USERNAME	b4eb8a837a231d
PASSWORD	c3b28eec

[View JSON](#)

# Roadmap

- Provisioning Services
- Using the CLI
- Using the Pivotal CF App Manager Console
- Binding to a Service
- **User Provided Services**

# User Provided Service Instances

- **User-provided service instances** are service instances
  - Already provisioned outside of Cloud Foundry
  - Behave like other service instances once created
  - Are little more than predefined configurations
    - A “*mock*” service for providing credentials
- When bound they provide service instance configuration (including credentials) to applications
  - Avoids hard coding service instance endpoints

<http://docs.cloudfoundry.org/devguide/services/user-provided.html>

# Use Cases

## User Provided Service Instances

- These are typically **legacy** or **existing instances** of a **service** (databases, queues, email, etc)
  - Applications connect to the *same* instance
    - With CF services, applications get *different* instances
  - Typically used with CF *on-premise*
    - Easy integration of your CF PaaS with your existing systems
- **Credential passing** used to inject the *same* credential set into each application

# Defining User Provided Services – 1

- Use `cf create-user-provided-service` command
  - Provide name and parameters/credentials
  - All applications bound to *same* instance in *same* way

```
$ cf cups mydb -p "hostname, port, username, password, name"  
hostname> db.example.com  
port> 1234  
username> dbuser  
password> dbpasswd  
name> mydb  
Creating user provided service mydb ... OK
```

Or use alias: `cf cups`

Specify any list of parameters here

Prompts for parameters values

# Defining User Provided Services – 2

- Or define within application's manifest.yml:

```
---
applications:
- name: spring-music
  memory: 512M
  instances: 1
  host: spring-music
  domain: cfapps.io
  path: build/libs/spring-music.war
  services:
    mydb:
      label: user-provided
      credentials:
        uri: postgres://dbuser:dbpass@db.example.com:1234/dbname
        username: pivotal
        password: pivotal
```

# User Provided Services - Accessing

- Bound service properties available in **VCAP\_SERVICES** environment variable
- In your code
  - Access variable
  - Parse JSON
  - Use to connect

```
{  
  user-provided: [  
    {  
      name: "mydb",  
      label: "user-provided",  
      tags: [ ],  
      credentials: {  
        hostname: "db.example.com",  
        port: "1234",  
        username: "dbuser",  
        password: "dbpasswd",  
        name: "mydb"  
      }  
    }  
  ]  
}
```

# Example: Application with Multiple Services

```
VCAP_SERVICES: {
  "rediscloud": [
    {
      "credentials": {
        "hostname": "redisvr...com",
        "password": "wU974wucDT45Jc",
        "port": "19016"
      },
      "label": "rediscloud",
      "name": "session-replication",
      "plan": "25mb",
      "tags": [
        "Data Stores",
        "Cloud Databases",
        "Developer Tools",
        "Data Store",
        "key-value",
        "redis"
      ]
    }
  ],
}
```

```
"user-provided": [
  {
    "credentials": {
      "uri": "http://review.cfapps.io"
    },
    "label": "user-provided",
    "name": "reviews",
    "syslog_drain_url": "",
    "tags": []
  },
  {
    "credentials": {
      "uri": "http://products.cfapps.io"
    },
    "label": "user-provided",
    "name": "products",
    "syslog_drain_url": "",
    "tags": []
  }
]
```

# What you have learned

- Provisioning Services
  - Using the CLI
  - Using the Pivotal CF App Manager Console
- Binding to a Service
- User Provided Services

# Lab

Creating a service and binding to it



# Services

Adding External Services

User Defined and Custom Services

Pivotal



# Topics Covered

- Creating new service providers as Custom (Managed) Services
- Using Service Brokers



# Roadmap

- **Custom Services using Service Brokers**
- Microservices
- Writing a Custom Service Broker
- Resources



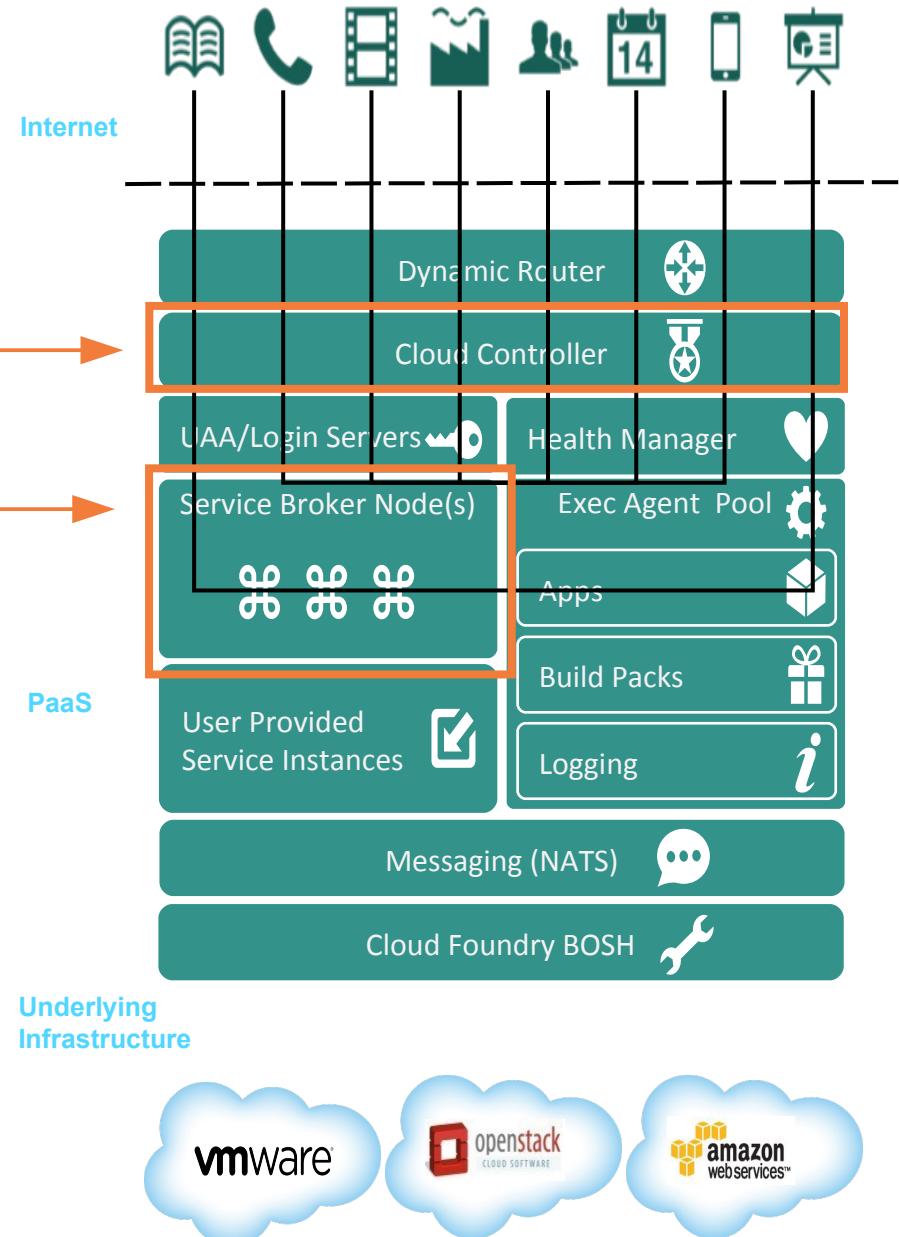
# Custom (Managed) Services



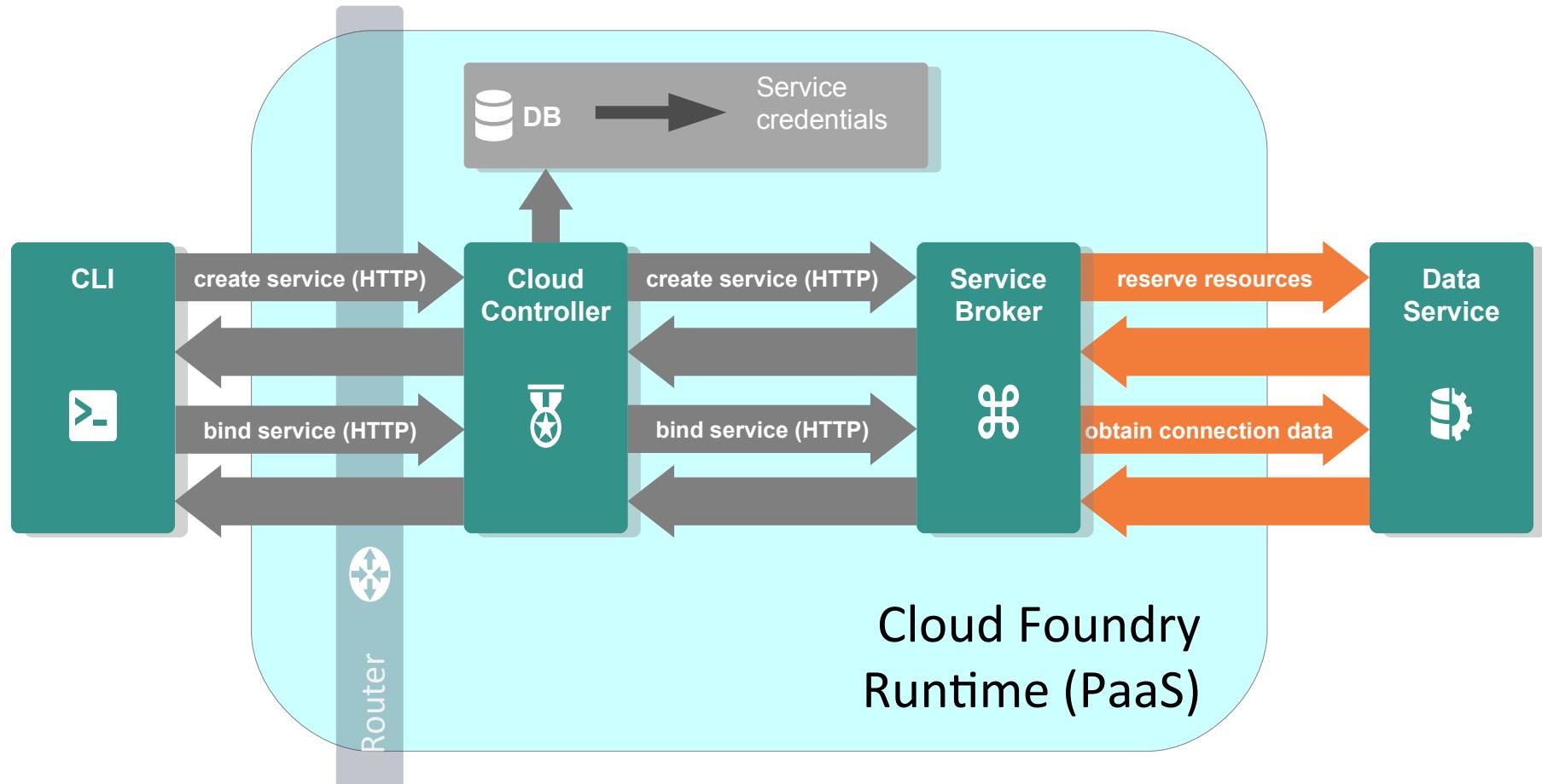
- **Custom or Managed Services**
  - Integrate with Cloud Foundry by implementing a documented API for which the cloud controller is the client
- **Service Broker**
  - Any component which implements the required API
  - Advertises catalog of service offerings or plans to CF
  - Handles calls from the Cloud Controller for five functions
    - *fetch catalog, create, bind, unbind, and delete*
  - In CF v1, Service Brokers were called *Service Gateways*

# Cloud Foundry Architecture

- Cloud Controller manages service via its service broker
- Broker runs on dedicated node
- A broker *must* define 5 HTTP RESTful endpoints
  - GET catalog
  - PUT new instance
  - PUT new binding
  - DELETE binding
  - DELETE instance

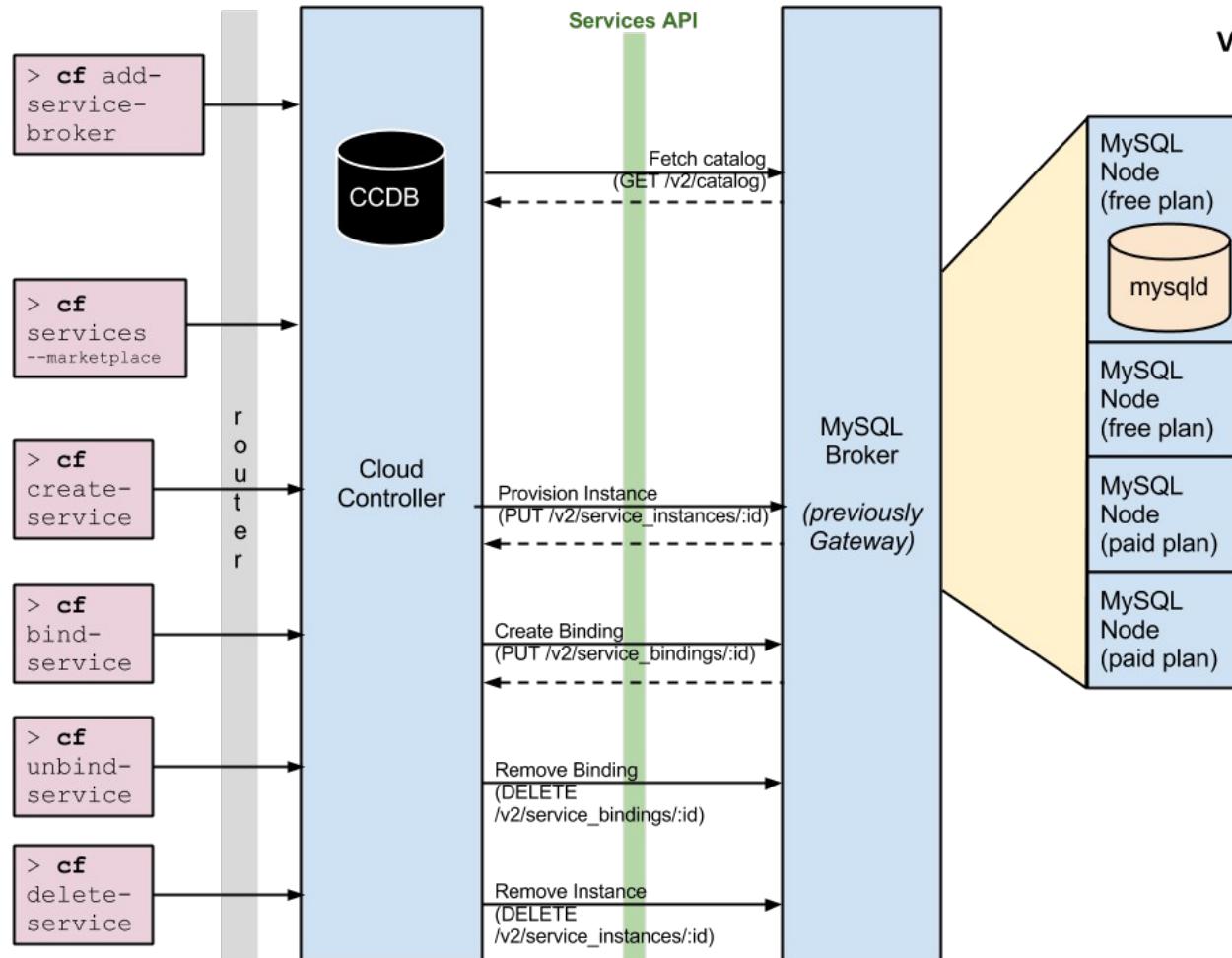


# Creating and Binding a Service



# Service Broker Example (MySQL)

Service Broker  
⌘ ⌘ ⌘



Pivotal™

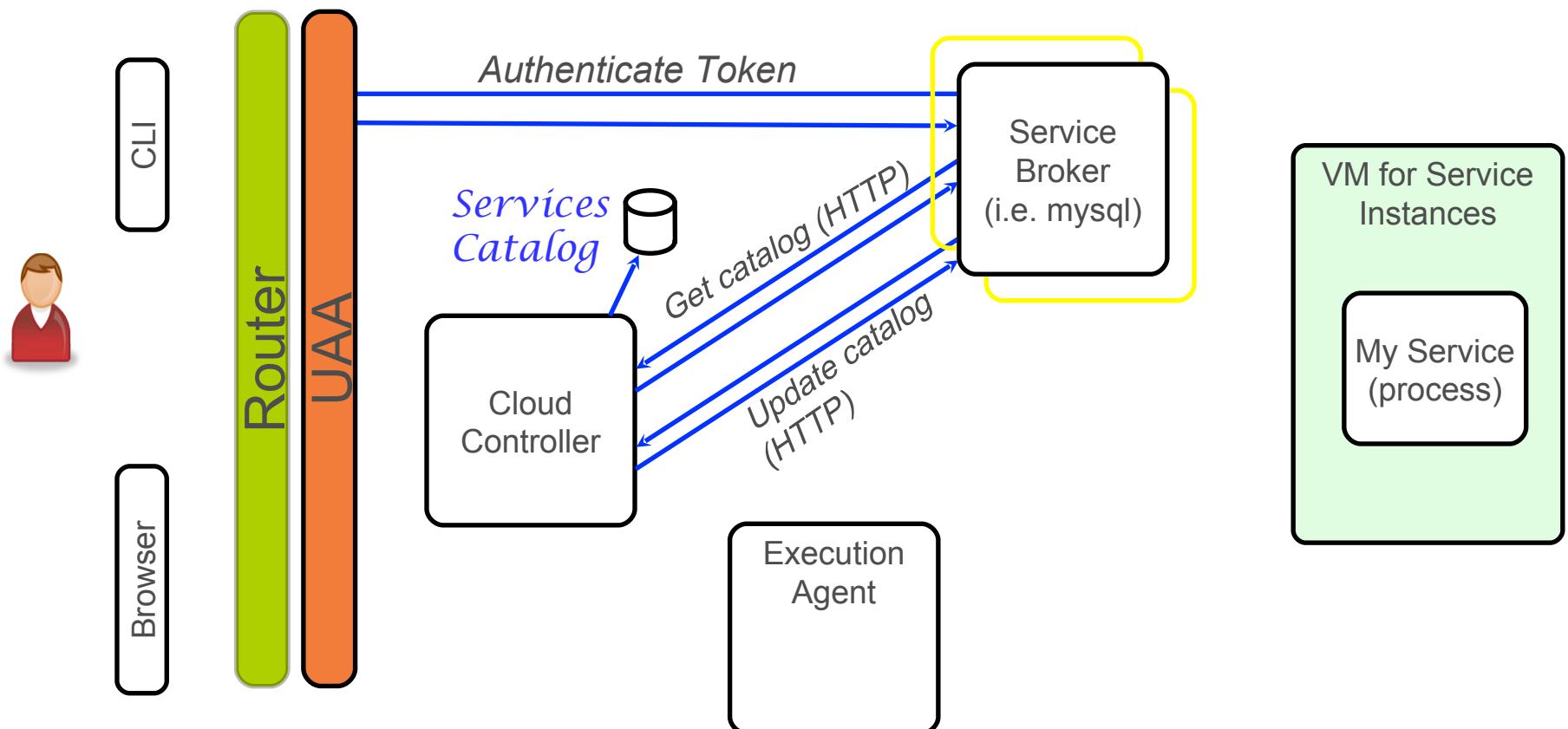
# Server Broker API



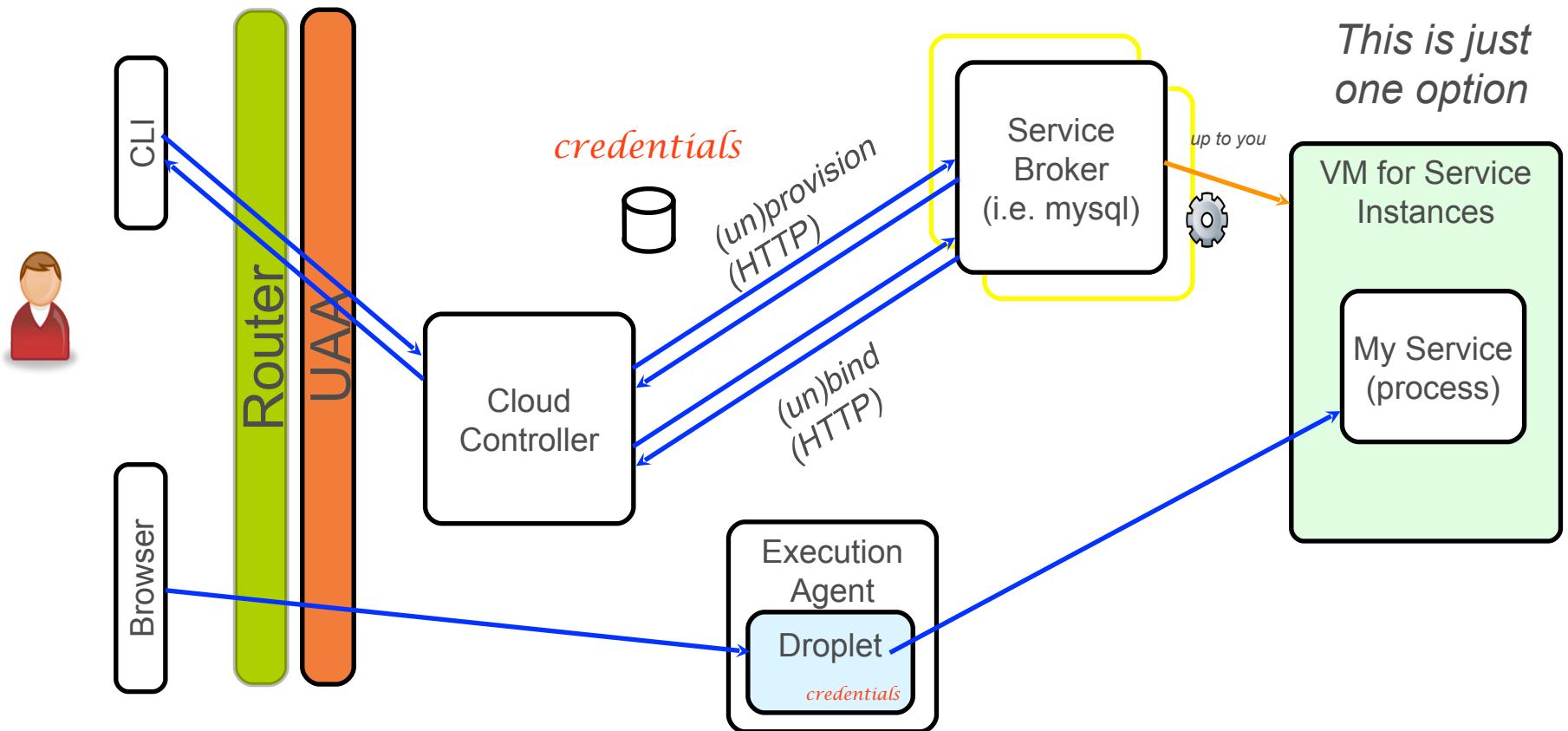
- [`GET /v2/catalog`](#)
  - List services and plans available from this broker.
- [`PUT /v2/service\_instances/:instance\_id`](#)
  - Create a new service instance.
- [`PUT /v2/service\_instances/:instance\_id/service\_bindings/:id`](#)
  - Create a new binding to a service instance.
- [`DELETE /v2/service\_instances/:instance\_id/service\_bindings/:id`](#)
  - Unbind from a service instance.
- [`DELETE /v2/service\_instances/:instance\_id`](#)
  - Delete a service instance
- [`PATCH /v2/service\_instances/:instance\_id`](#)
  - Update service to different plan

# 1. Broker Registers Offerings

- On Broker startup



## 2. Broker (un)provisions and (un)binds



# Service Broker Implementation

Service Broker  
⌘ ⌘ ⌘

- Service implementation is up to the service provider/developer
- Cloud Foundry only requires that the service provider implement the service broker API
- Broker can be implemented
  - As a separate application
  - By adding required http endpoints to an existing service
- Deploy using `cf add-service-broker`

# Service Instance Provisioning Examples

- Result of provisioning varies by service type
  - Some common actions that work for many services
- For a MySQL service, provisioning could result in:
  - An empty dedicated mysqld process running on its own VM.
  - An empty dedicated mysqld process running in a lightweight container on a shared VM.
  - An empty dedicated mysqld process running on a shared VM.
  - An empty dedicated database, on an existing shared running mysqld.
  - A database with business schema already there.
  - A copy of a full database, for example a QA database that is a copy of the production database.
  - For non-data services, provisioning could just mean getting an account on an existing system.

# Service Broker Deployment Models

- Many deployment models are possible:
  - Entire service packaged and deployed by BOSH alongside Cloud Foundry
  - Broker packaged and deployed by BOSH alongside Cloud Foundry
    - rest of the service deployed and maintained by other means
  - Broker (and optionally service) pushed as an application to Cloud Foundry user space
  - Entire service, including broker, deployed and maintained outside of Cloud Foundry by other means



# Roadmap

- Custom Services using Service Brokers
- **Microservices**
- Writing a Custom Service Broker
- Resources



# Microservices

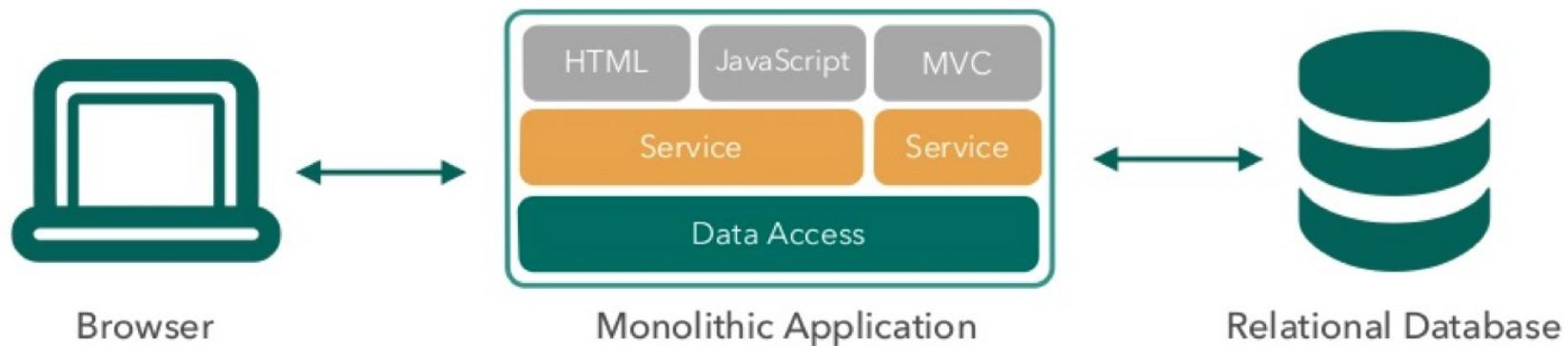
- What are they?
  - A software architecture design pattern
  - Complex applications are decomposed into small, independent processes communicating with each other
  - Language-agnostic APIs such as REST
  - Services are small, highly decoupled, focused on doing a small task
  - Ideally suited as Cloud Foundry apps
    - Made available as a service to other apps
  - **Terms:** anti-fragile, decomposition, agile ...

# Microservices Explained

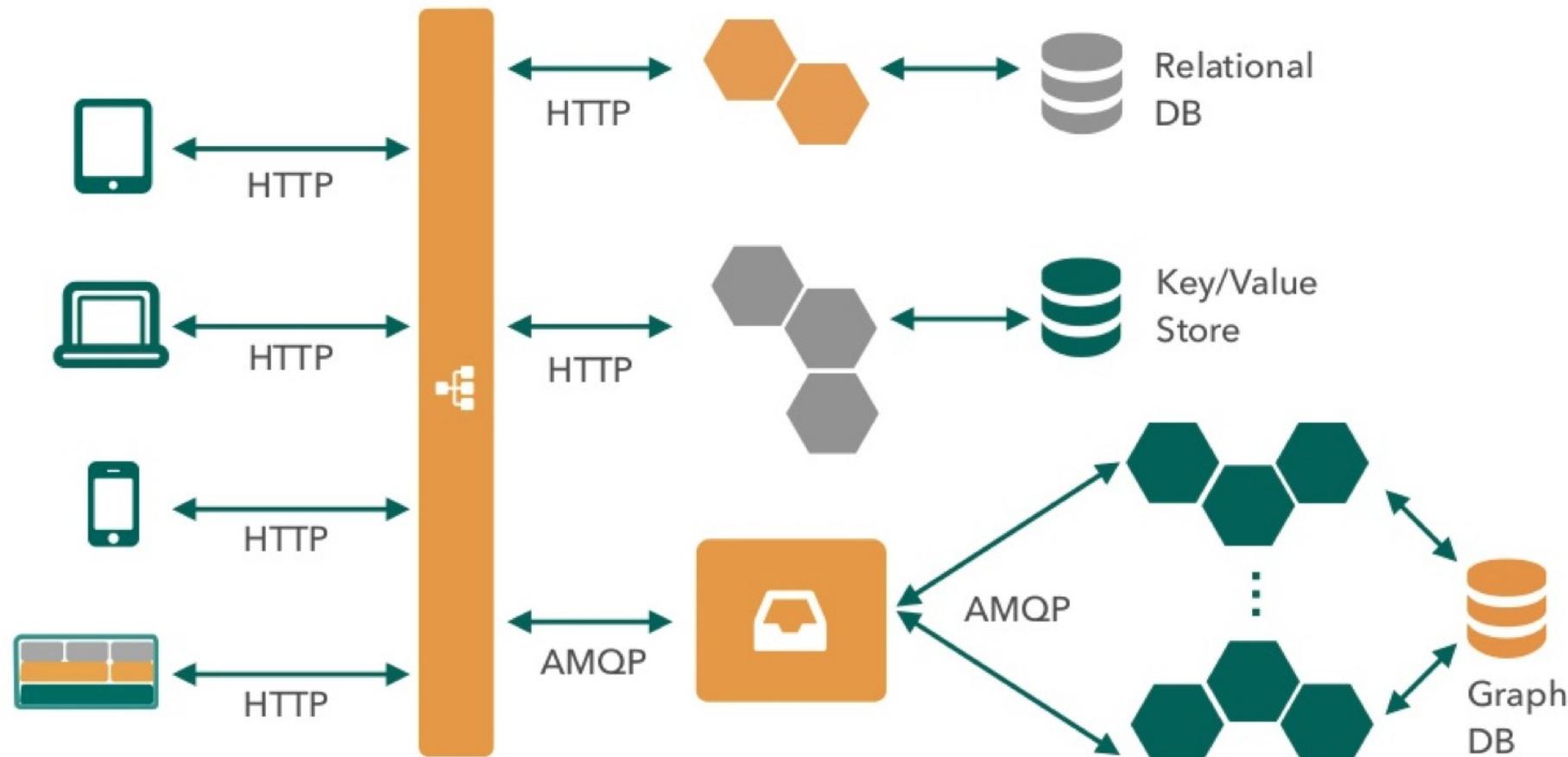
- Take a large monolithic application
  - Decompose into multiple services
  - Each can be maintained and enhanced independently
    - Effect of changes minimized
  - Main application fetched data from multiple microservices
  - SOA – lite!
- Implementation
  - Light weight, resilient
    - Run on Cloud Foundry
    - Natural fit for a Microservice – a Spring Boot Java app deployed to CF

# Monolithic Architecture

- Typical application today
    - Maintenance and change hard to organise



# Microservice Architecture





# Developing a Microservice

- Implement a Java application using
  - Spring Boot (minimal configuration)
  - Spring MVC (define a RESTful interface)
  - Run as CF application
    - CF benefits apply – automatic restart, run up new instances, scaling, availability
- Define a custom ServiceBroker
  - Make this application available as a Service



Pivotal



# Roadmap

- Custom Services using Service Brokers
- Microservices
- **Writing a Custom Service Broker**
- Resources



# Custom Service Broker – 1

- Several github projects exist
  - Help you define a Service Broker in Java
- Download generic project at
  - <https://github.com/cloudfoundry-community/spring-boot-cf-service-broker>
- Or rework Mongo DB example project (next slide)
  - <https://github.com/spgreenberg/spring-boot-cf-service-broker-mongo>

# Custom Service Broker – 2a

- Start with
  - <https://github.com/spgreenberg/spring-boot-cf-service-broker-mongo>
- Then customize...
  - Adjust `build.gradlefile`
  - Rename Mongo\*Service classes to appropriate names for your new service broker
  - Adjust code in service classes to implement appropriate backend processing for new service
  - Adjust/implement code in Repository classes
  - Adjust config code in CatalogConfig class to create a correct CF service catalog entry

# Custom Service Broker – 2b

- Continued ...
  - Adjust BrokerConfigComponentScan to exclude BrokerApiVersionConfig class
  - Update or disable test-code to ensure correct compilation
  - Make any final adjustments to *build.gradlefile*
  - Build and test
  - Define/adjust the CF manifest file
  - Deploy to CF and test



# Register New Service

- Register your new service-broker
  - `cf create-service-broker`
    - and other similar commands
- For more information see
  - <http://docs.cloudfoundry.org/services/managing-service-brokers.html>





# Roadmap

- Custom Services using Service Brokers
- Microservices
- Writing a Custom Service Broker
- **Resources**



# Resources

- MongoDB: <http://www.mongodb.org/downloads>
- Managed Services
  - <http://docs.cloudfoundry.org/services/managing-service-brokers.html>
- Spring Service Broker:
  - <https://github.com/cloudfoundry-community/spring-service-broker>
  - master branch is the REST framework application
  - mongodb branch is an implementation example using mongodb
- Spring Music App
  - <https://github.com/scottfrederick/spring-music>

# Roadmap

We have learned about

- Custom Services using Service Brokers
- Microservices
- Writing a Custom Service Broker
- Resources





# Introduction to BuildPacks

Deploying applications written in various languages

Java, Ruby, Groovy, JavaScript ... + Cloud

# Overview

- After completing this lesson, you should be able to:
  - Understand what a buildpack is
  - Deploy using a buildpack

# Roadmap

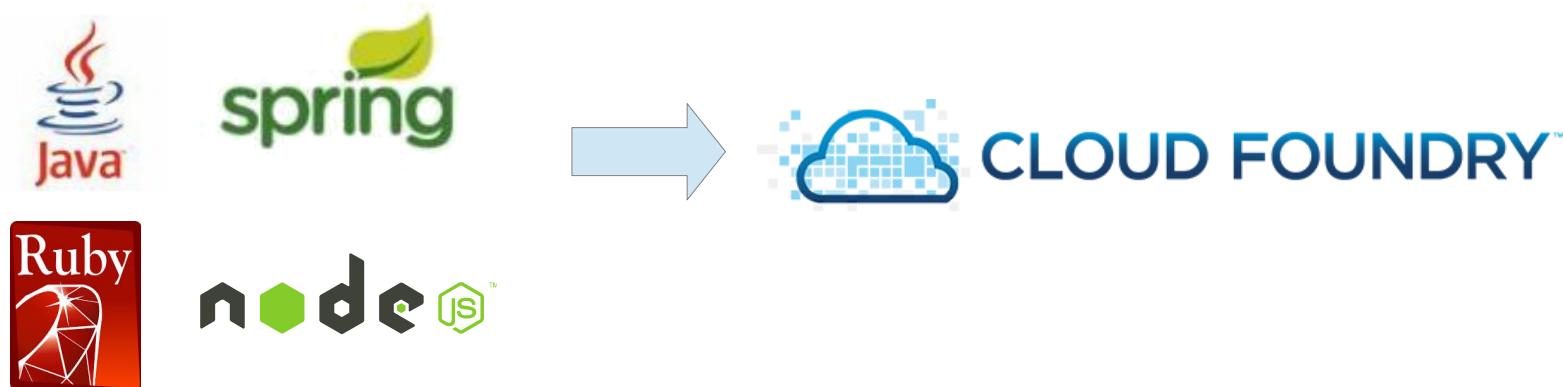
- **What are Buildpacks?**
- Deploying to Cloud Foundry
- Using Buildpacks

# Applications

- Consists of source code and application frameworks used by developers to create application
  - Java/Spring
  - Ruby/Rails
  - Java Script for Node.js
  - ...

# The Question:

- Applications can be written in many languages / frameworks:



- ...and yet each type can run in Cloud Foundry
- How is this possible?

# Configuring a Server from scratch

- If you were configuring a new server to run an application, what would you include / install?



- Operating system
  - Runtimes for your software (Java, Ruby, Python, etc.)
  - Containers as needed (e.g. Tomcat for Java, Apache HTTPD for PHP)
  - Frameworks as needed (APM tools)
  - Application binaries
- Good idea to write a script to do this.

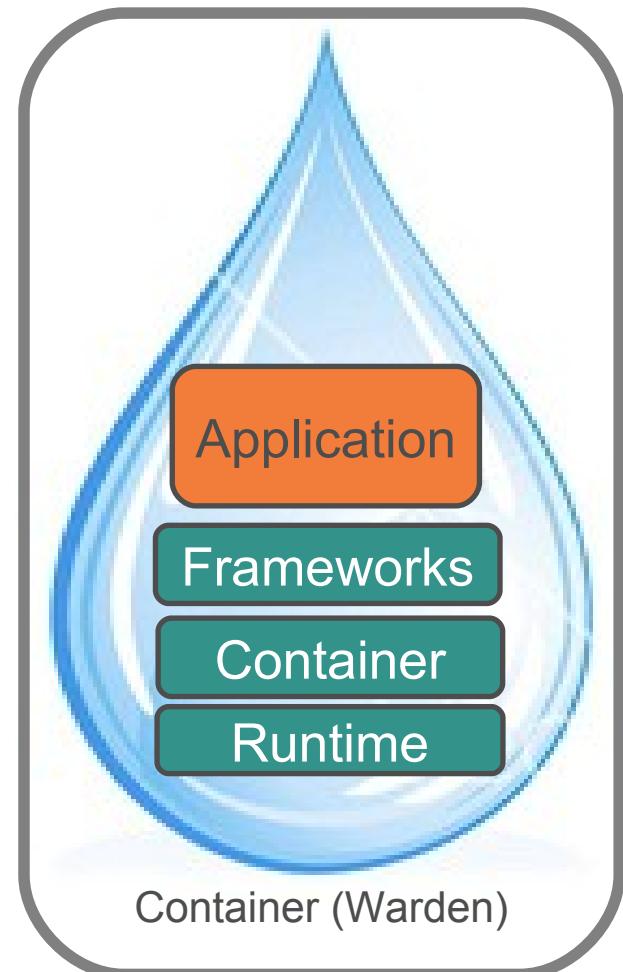
# A Buildpack Does the Same Thing

Except the goal is to run on Cloud Foundry

**Buildpack** – a combination of scripts that assembles runtimes, containers, frameworks, and your application into a *droplet*

**Droplets** – run inside Warden Containers

- Which run inside Execution Agents



# The Answer: Buildpacks

- **Buildpacks** define how assemble a droplet to run a specific kind of application
- Example:



- The Buildpack “builds” the “droplet” to run an app.
  - Called *staging* the application

# Buildpacks are Not...

- Buildpacks...
  - Are *not* a special build process for your application
    - Buildpacks build droplets
  - Do *not* run on your local machine
    - Buildpacks run on CF during the staging process

# Buildpack Structure

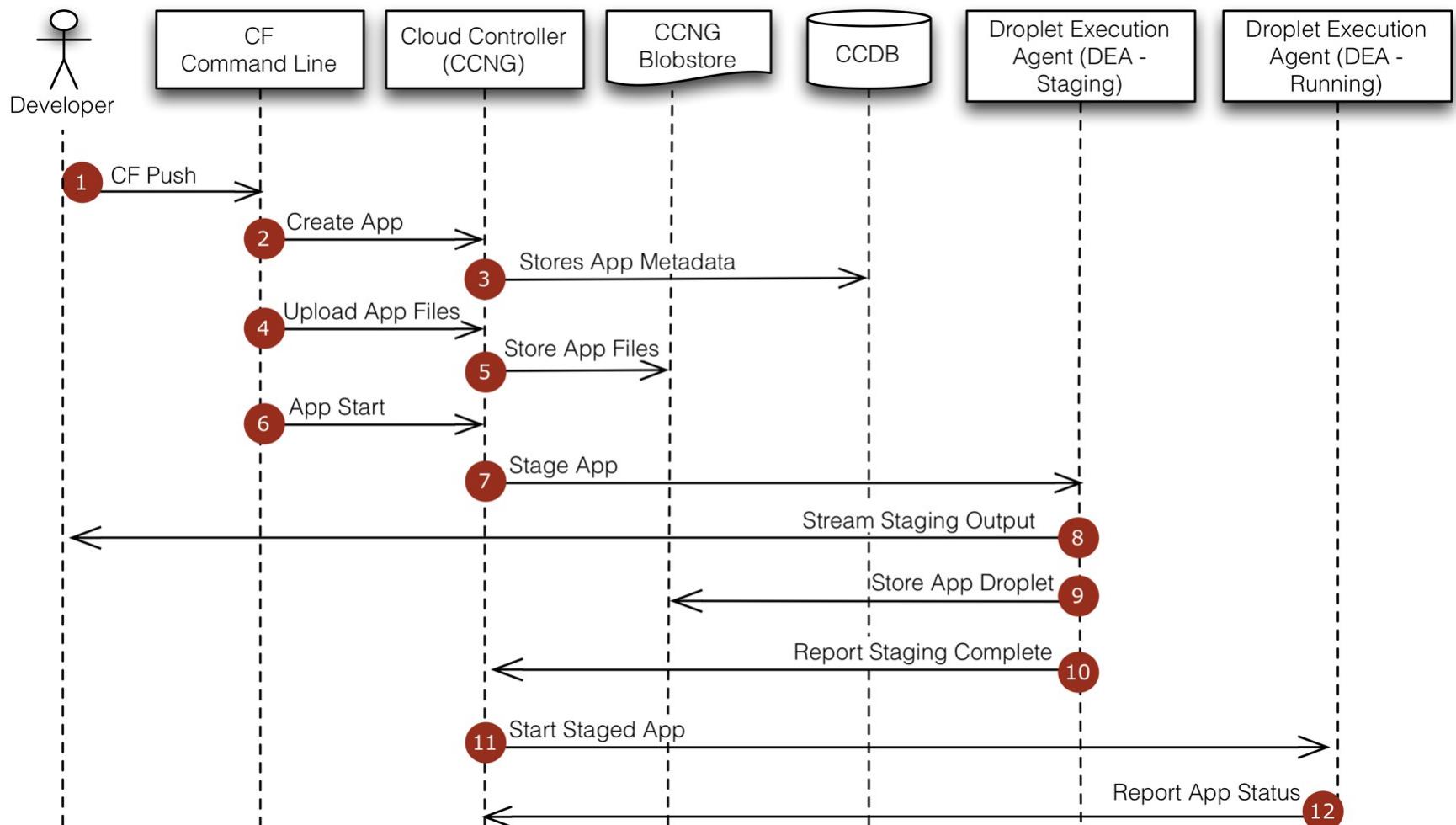
- Often written as Ruby scripts with three parts:
  - **Detect** if the buildpack should be applied
  - **Compile** (pack) the Droplet by combining the application code with runtimes, frameworks, plugins etc. necessary for the application
  - **Release** the app to be deployed to an assigned Execution Agent

**NOTE:** *Assemble* or *Pack* would be a better name than *Compile*  
**No** code compilation is happening

# Roadmap

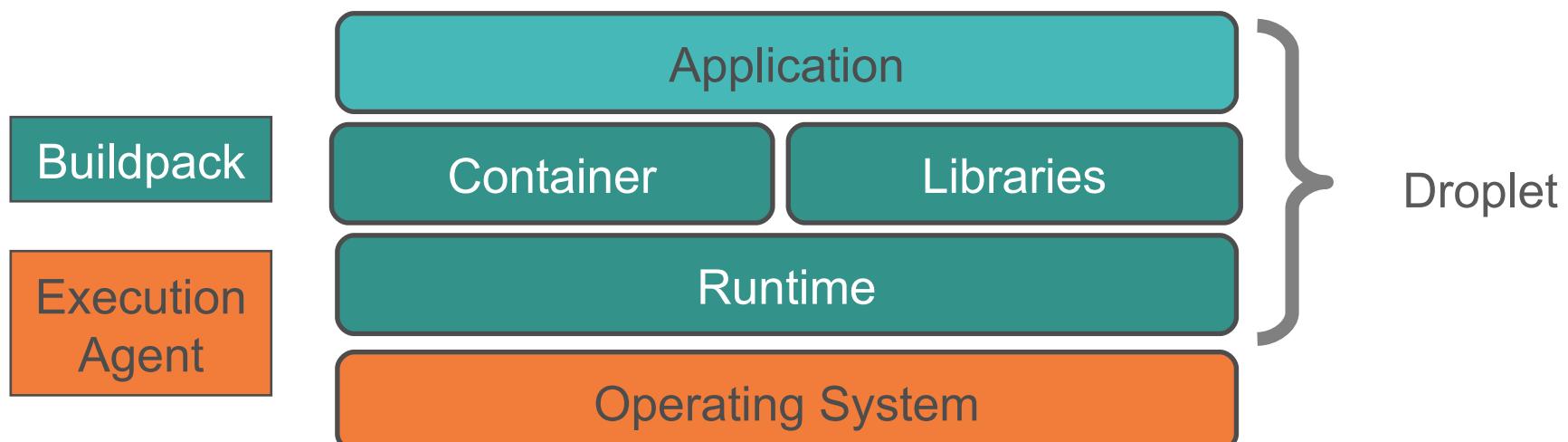
- What are Buildpacks?
- **Deploying to Cloud Foundry**
- Using Buildpacks

# Deploying to CF



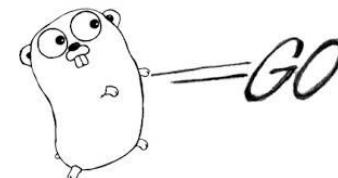
# Staging and Buildpacks

- Build packs are responsible for preparing the machine image for an application



# Available Buildpacks

- Buildpacks are either
  - Installed into a cloud foundry instance or
  - Loaded from an external location at push time
- Buildpacks provided by public Cloud Foundry
  - Note: This list expands over time!



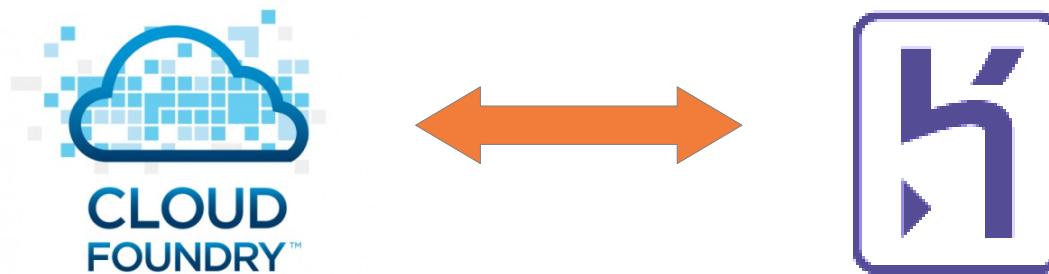
# Custom Buildpacks

- Cloud Foundry Community provides buildpacks for other languages
- Or write your own
  - Usually by forking / adapting an existing buildpack
- For list of CF Community Buildpacks
  - <https://github.com/cloudfoundry-community/cf-docs-contrib/wiki/Buildpacks>



# Compatibility

- Buildpacks can be compatible with multiple PaaS offerings
- CF buildpacks follow the Heroku buildpack design
  - CF and Heroku buildpacks are compatible (if you care to make them compatible)
  - Other PaaS offerings adopting the buildpack design



# Roadmap

- What are Buildpacks?
- Deploying to Cloud Foundry
- **Using Buildpacks**

# Built-In Buildpacks

- Use **cf buildpacks** to determine installed buildpacks

```
> cf buildpacks
Getting buildpacks...

buildpack      position  enabled  locked  filename
ruby_buildpack  1        true     false    ruby_buildpack-offline-v1.0.1.zip
nodejs_buildpack 2        true     false    nodejs-buildpack-offline-b29.zip
java_buildpack   3        true     false    java-buildpack-v2.4.zip
go_buildpack    4        true     false    go_buildpack-offline-v1.0.1.zip
liberty_buildpack 5        true     false    liberty_buildpack.zip
python_buildpack 6        true     false    python_buildpack-offline-v1.0.1.zip
php_buildpack   7        true     false    php_buildpack-offline-v1.0.1.zip
```

# Managing Built-In Buildpacks

```
$> cf create-buildpack <name> <path> <order>
```

- <path> – local directory / zip file / URL / URL to zip file
- <order> – relative order in buildpack list
- **--enable** / **--disable**

- Commands for update, delete, rename available
- Administrator permissions required

# Automatic Detection / Explicit Reference

**\$> cf push**

- Application checked against pre-defined buildpacks
- Matching buildpack invoked automatically

**\$> cf push -b <buildpack-name>**

- Desired buildpack specified (installed buildpack)

**\$> cf push -b <url>**

- The desired buildpack is referenced by a Git URL
  - Note: “disable custom buildpacks” disables this option

# Specify within manifest

- Use buildpack element
  - Specify name or URL

```
---  
applications:  
- name: cf-my-app  
  host: cf-my-app  
  domain: cfapps.io  
  path: target/my-war.war  
  buildpack: https://github.com/cloudfoundry/java-buildpack
```

- Remember precedence
  - Options specified in push command override manifest

# Pushing an Executable

- Suppose I have a binary executable?
  - Such as a script or statically compiled C, C++ application
  - *Must* be compiled for x86 Linux
- Then I can push and run it using the “null” buildpack

```
$> ./bin/hello
Hello World
$> cf push hello --no-route -p bin -m 256m -c "./bin/hello"
      -b http://github.com/ryandotsmith/null-buildpack.git
... usual push logging ...
Hello World
$>
```

# Summary

- After completing this lesson, you should have learned:
  - What a buildpack is
  - How to deploy using a buildpack

# Lab

Starting with Buildpacks,  
Using a third-party buildpack



# Customizing BuildPacks

Modifying a buildpack

# Overview

- After completing this lesson, you should understand:
  - The basic API of a Buildpack
  - The behavior of the Java Buildpack
  - How to configure / extend a Buildpack

# Roadmap

- **Buildpack API**
- Java Buildpack
- Configure / Extend Java Buildpack

# Buildpack API

- `/bin/detect app_directory`
  - Inspect application bits to determine buildpack applicability
- `/bin/compile app_directory cache_directory`
  - Download and install runtime, container, packages, libraries; install application bits as necessary
- `/bin/release app_directory`
  - Build application start command

# /bin/detect

- Inspect the app bits to determine if the buildpack knows how to handle the application

 <b>Ruby</b> <i>A Programmer's Best Friend</i>	Gemfile <b>exists?</b>
	package.json <b>exists?</b>
 <b>python</b> <sup>TM</sup>	setup.py <b>exists?</b>

# /bin/compile

- 'Builds' the Droplet
- Downloads and installs any necessary runtime
  - Java VM, Ruby interpreter, JavaScript interpreter ...
  - Container or web server
  - Support libraries, packages, modules
    - Java jars, Ruby gems, NPM packages
- Then install the app bits into the runtime or container

# /bin/compile caching

- Runtime, container, and support packages are often downloaded from sources external to Cloud Foundry
  - Depending on the buildpack
- Cloud Foundry provides a location (cache) for storing downloaded artifacts to speed subsequent staging operations

# /bin/release

- Builds a YAML-formatted hash with three possible keys
- On Cloud Foundry (currently) only the ***web:*** value is used to get the start command for the app

```
addons: []
config_vars: {}
default_process_types:
web: <start command>
```

# Roadmap

- Buildpack API
- **Java Buildpack**
- Configure / Extend Java Buildpack

# Java Buildpack

- Supports a variety of JVM languages, containers, and frameworks with a modular, configurable, and extensible design



# Java Buildpack Concepts

## Containers

How an application is run

## Frameworks

Additional application transformations

## JREs

Java Runtimes

# Java Buildpack Concepts

## Containers

Executable JARs, Groovy, Play,  
Servlet 2 & 3, Spring Boot CLI

## Frameworks

AppDynamics, New Relic,  
Spring Auto-reconfiguration

## JREs OpenJDK, Oracle JDK

# Container Detection Criteria

Java main()

META-INF/MANIFEST.MF exists with Main-class attribute set

Tomcat

WEB-INF directory exists

Groovy

- .groovy file with a main() method, or
- .groovy file with no classes, or
- .groovy file with a shebang (#!) declaration

Spring Boot CLI

one or more POGO .groovy files with no main() method, and no WEB-INF directory

Play

start and lib/play.play\_\*.jar exist

# Framework Detection Criteria

App Dynamics

App Dynamics service bound to app

New Relic

New Relic service bound to app

Spring  
AutoConfiguration

spring-core\*.jar in the application  
directory

# /bin/compile Output Example

- Output example:

```
-----> Downloaded app package (11M)
-----> Downloading Open Jdk JRE 1.8.0_20 from
http://download.run.pivotal.io/openjdk/lucid/x86_64/open
jdk-1.8.0_20.tar.gz (1.0s)

          Expanding Open Jdk JRE to .java-
buildpack/open_jdk_jre (1.1s)

-----> Downloading Tomcat Instance 7.0.53 from
http://download.run.pivotal.io/tomcat/tomcat-
7.0.53.tar.gz (0.5s)

          Expanding Tomcat to .java-buildpack/tomcat (0.1s)

-----> Uploading droplet (49M)
```

# See What's Going On

```
$> cf files <app-name> app
```

.java-buildpack.log

Log output from buildpack

.java-buildpack

Sandboxes for each component  
used during staging

open\_jdk\_jre

spring\_auto\_reconfiguration

tomcat

...

META-INF/

WEB-INF/

# Roadmap

- Buildpack API
- Java Buildpack
- **Configure / Extend Java Buildpack**
- Customization without Forking

# Customization

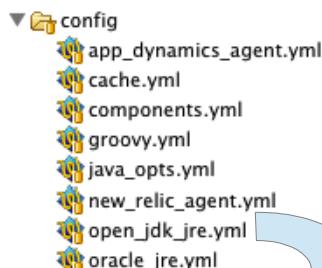
- You may alter Java buildpack
  - **Configure** artifacts used by standard JREs, Containers, and Frameworks
  - **Extend** the buildpack with your own JREs, Containers, and Frameworks
- Customization is done by forking the buildpack



- ...Or simply downloading, modifying, and zipping.

# Customizing Configuration

- Most configuration options found in /config
  - determine behavior of a JRE, Container, or Framework



```
---
repository_root: "{default.repository.root}/openjdk/{platform}/{architecture}"
version: 1.8.0_+
memory_sizes:
  metaspace: 64m..
memory_heuristics:
  heap: 75
  metaspace: 10
  stack: 5
  native: 10
```

*repository\_root* and *version* typically at the top of each file.

# Locating Downloads

- URLs derived from repository root
  - `{default.repository.root}/openjdk/{platform}/{architecture}`
  - `download.pivotal.io.s3.amazonaws.com/openjdk/lucid/x86_64`
  - **index.yml** holds location of each version

```
# http://download.pivotal.io.s3.amazonaws.com/openjdk/lucid/x86_64/index.yml
---
1.8.0_25: https://download.run.pivotal.io/.../x86_64/openjdk-1.8.0_25.tar.gz
1.7.0_71: https://download.run.pivotal.io/.../x86_64/openjdk-1.7.0_71.tar.gz
1.8.0_31: https://download.run.pivotal.io/.../x86_64/openjdk-1.8.0_31.tar.gz
1.7.0_75: https://download.run.pivotal.io/.../x86_64/openjdk-1.7.0_75.tar.gz
1.8.0_40: https://download.run.pivotal.io/.../x86_64/openjdk-1.8.0_40.tar.gz
...
```

# Customization by Configuration: Tomcat

- Example: customizing the Tomcat artifact for download

```
# cloudfoundry/java-buildpack/config/tomcat.yml
---
tomcat:
  version: 8.0.+
  repository_root: "{default.repository.root}/tomcat"
...
```

```
# http://files.example.com/tomcat-custom/index.yml
---
8.0.18: https://download.run.pivotal.io/tomcat/tomcat-8.0.18.tar.gz
8.0.17: https://download.run.pivotal.io/tomcat/tomcat-8.0.17.tar.gz
7.0.59: https://download.run.pivotal.io/tomcat/tomcat-7.0.59.tar.gz
8.0.20: https://download.run.pivotal.io/tomcat/tomcat-8.0.20.tar.gz
```

# Resource Configuration

- Tomcat container supports simple customization of **context.xml** and **server.xml**
  - Files will *overlay* sandbox provided values.

```
resources/tomcat/conf
  └── context.xml
  └── server.xml
```

- Not just for Tomcat
  - JDK, New Relic, etc.

# Extending the Buildpack - 1

- You can *extend* the Java Buildpack
  - To add different JRE, Container, or Framework
- Implement support class (Ruby) in the appropriate directory
  - with additional support classes as necessary

```
lib/java_buildpack
  └── jre
  └── container
  └── framework
```

# Extending the Buildpack - 2

- Support class types have similar interfaces, following the buildpack scripts naming conventions

```
# Return String or an Array<String> that identifies the component to be
# used in staging, or nil.
def detect

# Modifies the application's file system. Component is expected to
# transform the application's file system in whatever way is necessary
# (e.g. downloading files or creating symbolic links) to support the function
# of the component. Status output written to STDOUT is expected.
def compile

# Modifies the application's runtime configuration to support the function
# of the component. Create the command required to run the application,
# taking context values into account when creating the command. Container
# components are expected to return the command required to run the application.
def release
```

# Extending the Buildpack - 3

- Add new support class to `config/components.yml`

```
# Configuration for components to use in the buildpack
---
containers:
  - "JavaBuildpack::Container::DistZip"
  - "JavaBuildpack::Container::Groovy"
  - "JavaBuildpack::Container::JavaMain"
  - "JavaBuildpack::Container::PlayFramework"
  - "JavaBuildpack::Container::Ratpack"
  - "JavaBuildpack::Container::SpringBoot"
  - "JavaBuildpack::Container::SpringBootCLI"
  - "JavaBuildpack::Container::Tomcat"
  - "JavaBuildpack::Container::YOUR-CONTAINER-HERE"
```

```
jres:
  - "JavaBuildpack::jre::OpenJdkJRE"
#  - "JavaBuildpack::jre::OracleJRE"
```

...or here if JRE...

```
frameworks:
  - "JavaBuildpack::Framework::AppDynamicsAgent"
```

...or here if framework.

...

# More on Customization

- Much more information and documentation included in the GitHub repository

<https://github.com/cloudfoundry/java-buildpack>

# Roadmap

- Buildpack API
- Java Buildpack
- Configure / Extend Java Buildpack
- **Customization without Forking**

# Customization without Forking

- Simple customization of properties can be done without forking the buildpack
  - Set environment variables instead
    - Either using `cf set-env` or in the `env:` section of manifest
- Three options:
  - `JAVA_OPTS` variable
  - `JBP_CONFIG` variables

# Change JVM Runtime Options – I

- The `JAVA_OPTS` variable is recognized when app runs:

```
$> cf set-env spring-music JAVA_OPTS -showversion
Setting env variable JAVA_OPTS -showversion spring-music myorg
development jlee@pivotal.io
OK
TIP: Use 'cf restage' to ensure your env variable changes take effect

$> cf restage spring-music
... usual push output ...
2015-04-10T16:45:11.88 [App/0] ERR openjdk version "1.8.0_40-"
2015-04-10T16:45:11.88 [App/0] ERR OpenJDK Runtime Environment (build
1.8.0_40--vagrant_2015_03_26_09_03-b25)
2015-04-10T16:45:11.88 [App/0] ERR OpenJDK 64-Bit Server VM (build
25.40-b25, mixed mode)
...
$>
```

# Change JVM Runtime Options – II

- Most JVM options can be specified this way
  - Except some that govern memory sizing
    - Such as **-Xms**, **-Xmx**, **-Xss**, **-XX:MaxPermSize**,  
**-XX:MaxMetaspaceSize**, **-XX:MetaspaceSize**,  
**-XX:PermSize**
    - Most other **-XX** options can be used
    - For full details see:

[https://github.com/cloudfoundry/java-buildpack/blob/master/docs/framework-java\\_opts.md](https://github.com/cloudfoundry/java-buildpack/blob/master/docs/framework-java_opts.md)

# JBP\_CONFIG variables

- Use environment variable to override a buildpack configuration file
  - Naming convention used:
    - `my_file.yml` → `JBP_CONFIG_MY_FILE`
  - Variable must be set to valid *inline* YAML syntax
- To change default version of Java to 7
  - Override `open_jdk_jre.yml`

```
>$ cf set-env my-application JBP_CONFIG_OPEN_JDK_JRE '[version: 1.7.0_+, memory_heuristics: {heap: 85, stack: 10}]'
```



# Offline Buildpacks

- Wish to avoid downloading buildpacks from Internet
- Java Buildpack can be packaged as *Offline* Buildpack
  - Builds droplets *without* internet connection
- One-time build process
  - Internally packages latest version of each dependency within the buildpack
  - Disables remote downloads
  - About 180M in size
  - Install using **cf create-buildpack/update-buildpack**
  - <https://github.com/cloudfoundry/java-buildpack/blob/master/docs/buildpack-modes.md>
- **Note:** Pivotal CF ships with offline buildpacks!

# Summary

- After completing this lesson, you should have learned:
  - The basic API of a Buildpack
  - The behavior of the Java Buildpack
  - How to configure / extend a Buildpack

# Lab

## Forking and Customizing the Java Buildpack



# Managing Applications in Cloud Foundry

A closer look at practical Cloud Foundry usage

Log Management, APM Integration,  
Autoscaling, Zero-downtime  
deployments

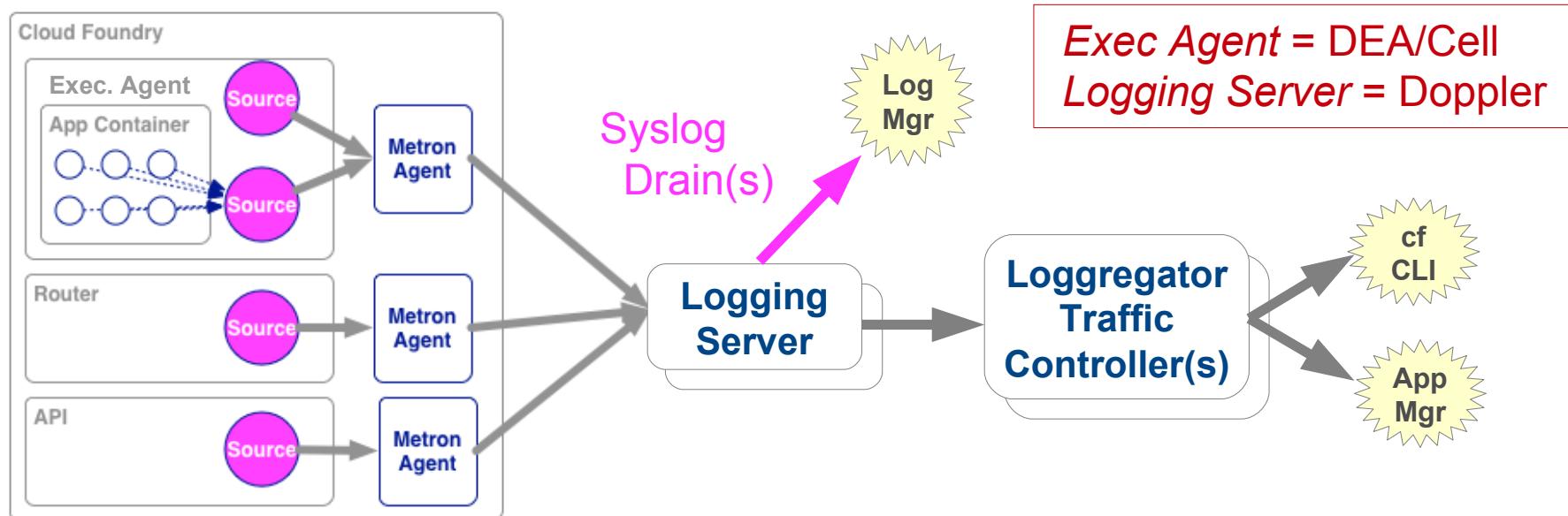
Pivotal

# Roadmap

- **Log Management**
- Application Performance Management
- Autoscaling
- Zero-Downtime Deployments

# Recall: Log Aggregation Architecture

- Collects log output from app instances, CF components
- Aggregates into a consolidated log
- Sinks to cf logs, App Mgr, *third-party log managers*



# Why Third-Party Log Managers?

- Recommended approach
  - Can store far more logging information than CF
  - Allow for persistence, storage, *searching, analyzing, metrics*
- Variety of third-party log managers supported:

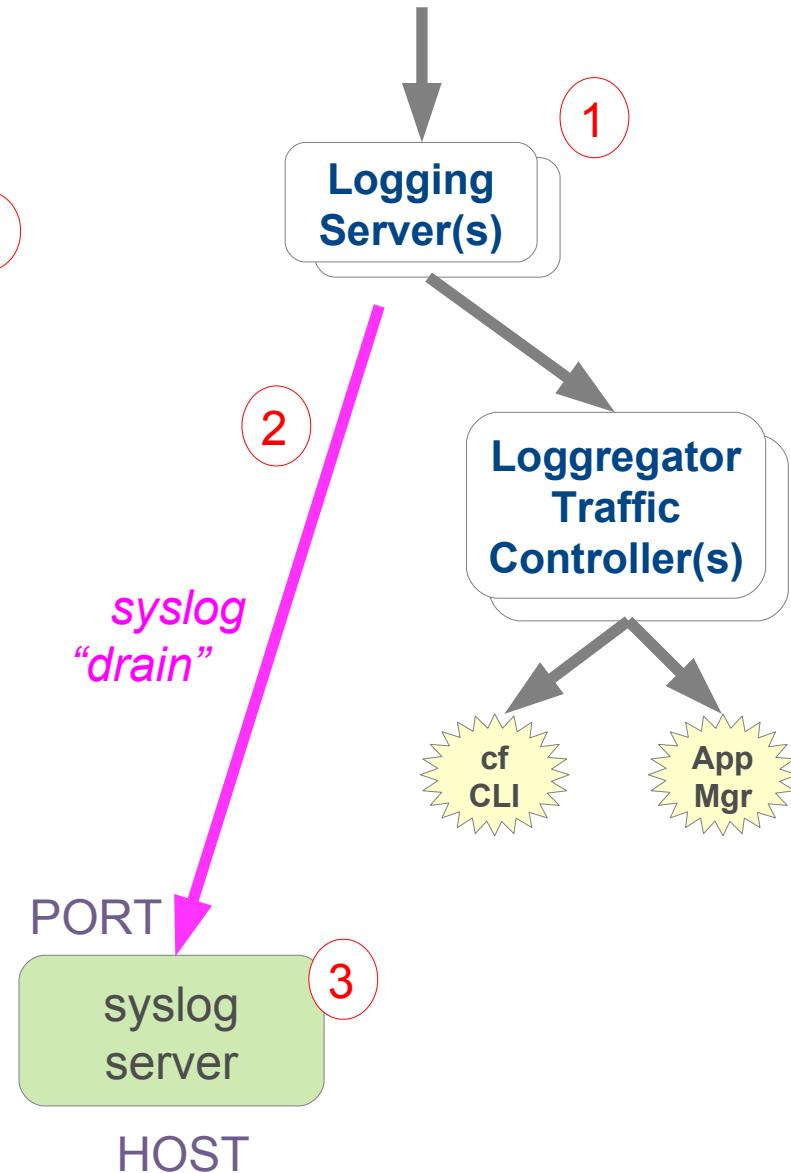


# Connecting to Third-Party Log Managers

- Setup Log Manager, determine **HOST** and **PORT**.
  - Process varies according to vendor
- Create User Provided Service with a Syslog drain:
  - `cf cups <SERVICE> -l syslog://<HOST>:<PORT>`
- Bind to application, restage.
  - Cloud Foundry sinks logger output to this drain for this application
  - See *next slide* ...

# How It Works

- All output for app collected by Logging (Doppler) server
- Loggregator opens socket to HOST:PORT  
    – Sends all log info for app to socket in syslog format
- Received by third-party syslog server



# Example: PWS and PaperTrail

- **PaperTrail:** Cloud-based Log Manager
  - a) Create account at <https://papertrailapp.com>
  - b) Use the “Add System” button
    - Papertrail will provide you the URL to use for your syslog drain
    - Example: logs2.papertrailapp.com:41846

# Example: PWS and Papertrail

c) Click the “Alternatives” link

d) Select the  
“*Heroku*”  
option

e) Name your  
system

Choose your situation:

A My syslogd only uses the default port  
GNU syslogd and some embedded OSes will only log to port 514. A few Linux distros shipped with GNU syslogd (mostly older CentOS and Gentoo).

B I use Heroku  
Register each app separately.

C My system's hostname changes  
In rare cases, one system may change hostnames frequently. For example, a roaming laptop which sets its hostname based on DHCP (and roams across networks).

Heroku uses TCP without TLS. We'll provide an app-specific Heroku syslog drain and [step-by-step setup](#).

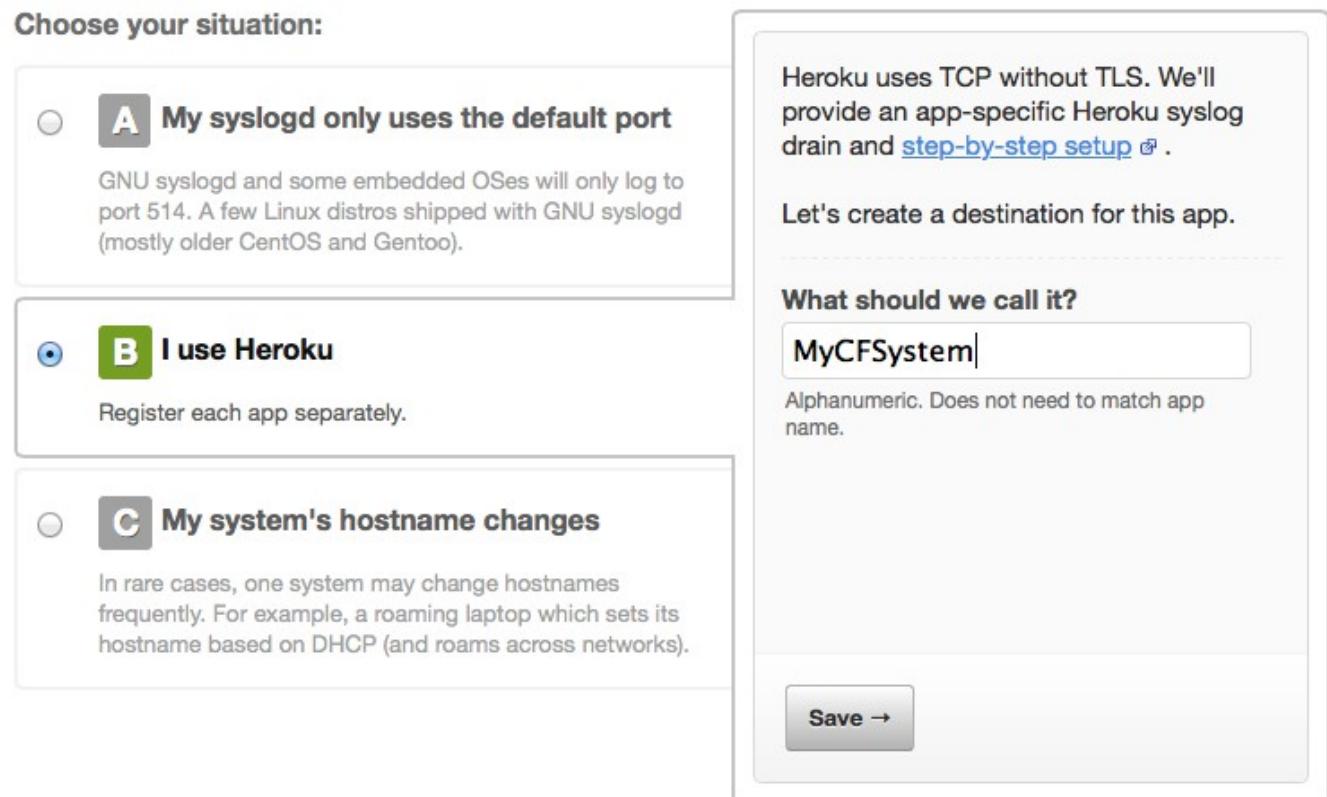
Let's create a destination for this app.

What should we call it?

MyCFSSystem

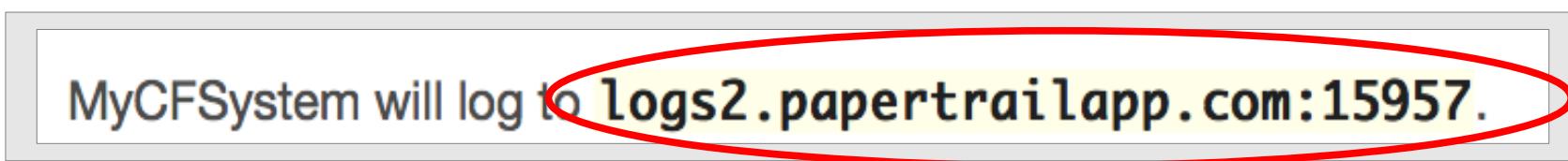
Alphanumeric. Does not need to match app name.

Save →



# Example: PWS and PaperTrail

- f) Setup user defined service using Papertrail's URL:



MyCFSSystem will log to **logs2.papertrailapp.com:15957**.

- g) Create User Provided Service with a Syslog drain:

```
cf cups the-drain -l  
syslog://logs2.papertrailapp.com:15957
```

- h) Bind to application, restart:

```
cf bind-service the-app the-drain  
cf restart the-app
```

# About Syslog

- De facto standard for logging on Unix/Linux
  - Can log to a file or a server *syslogd* (via a protocol)
  - Splunk, Papertrail and others provide syslog servers
- To log to syslog
  - Generate a TCP or UDP message in the right format
  - Open a socket to your syslog server and send
- Higher level logging options exist
  - <https://github.com/cloudfoundry-community/java-loggregator>
  - Output handlers for Java logging or log4j/logback

# Lab

Configuring Third-Party Log  
Management Tools with  
Cloud Foundry

# Roadmap

- Log Management
- **Application Performance Management**
- Autoscaling
- Zero-Downtime Deployments



# Application Performance Monitoring

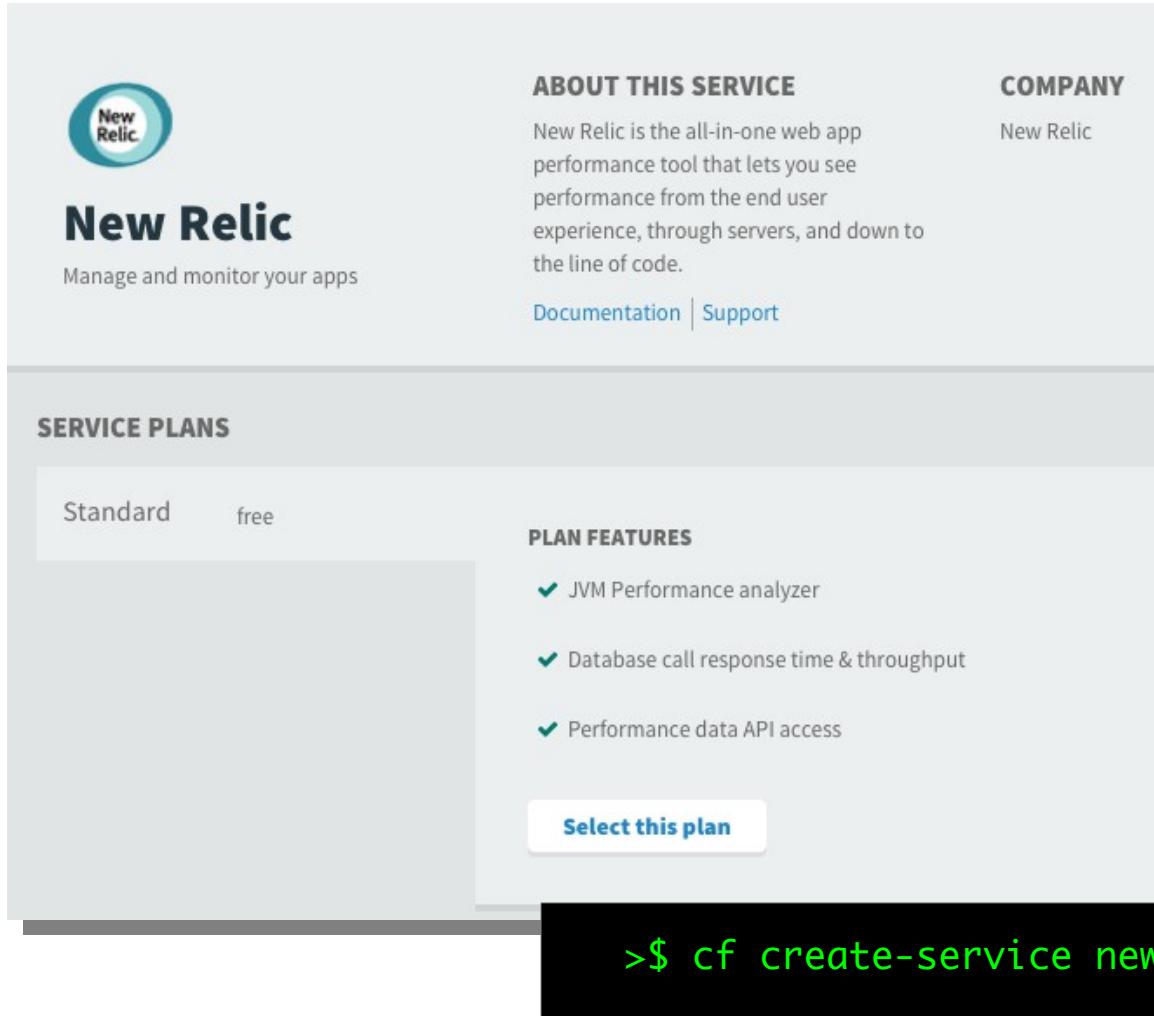
- Logs and analysis only takes you so far
- Important to have real-time monitoring of applications
  - Uptime, performance, etc.
- Application Performance Monitoring (APM) Tools
  - Monitor your application while running
  - Several choices available in Cloud Foundry
    - PWS – New Relic and AppDynamics
    - Pivotal Spring Insight

# Pivotal Web Services and New Relic



- PWS offers simple interface to New Relic
  - Available as Marketplace Service
  - Tracks different instances of application
  - Monitors down to the line of code
- How To Use:
  - Create New Relic service in desired space
  - Bind to desired Application(s)
  - Re-stage application
    - Java Buildpack includes New Relic Agent, others may not
  - APM available as a link from within PWS

# Creating the New Relic Service



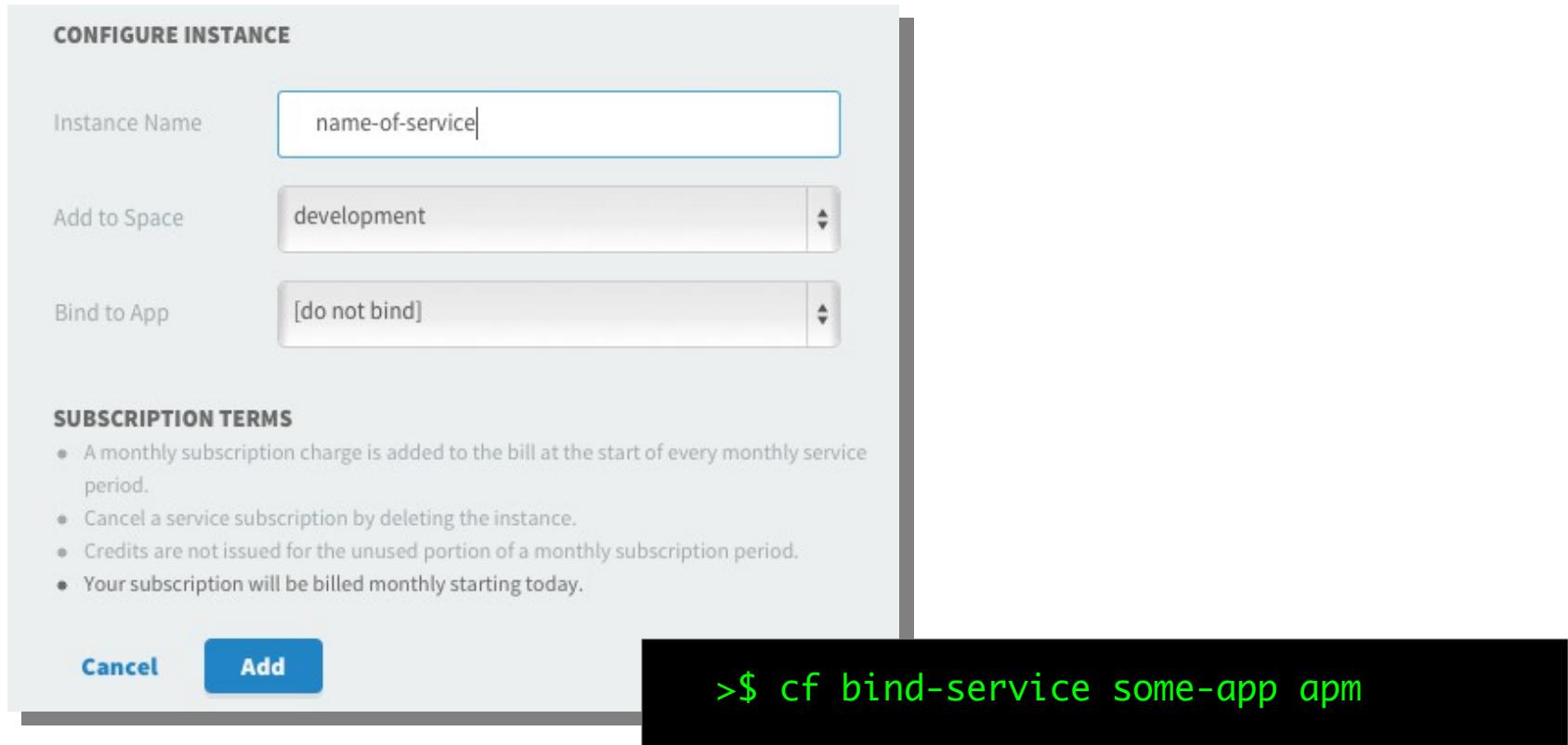
The screenshot shows the New Relic service creation interface. At the top left is the New Relic logo and the text "New Relic" followed by "Manage and monitor your apps". To the right is a section titled "ABOUT THIS SERVICE" which describes New Relic as an all-in-one web app performance tool. Below this is a "COMPANY" section with "New Relic". Under "ABOUT THIS SERVICE", there are links for "Documentation" and "Support". In the middle section, there's a "SERVICE PLANS" heading. A "Standard" plan is highlighted in light gray, indicating it's selected, and is described as "free". To the right of the plan details is a "PLAN FEATURES" list with three items: "JVM Performance analyzer", "Database call response time & throughput", and "Performance data API access". A blue button labeled "Select this plan" is located at the bottom of this section. At the bottom of the page, there's a black bar containing a green command-line text: ">\$ cf create-service newrelic standard apt".

- Use App Manager Console

- Use cf CLI

# Create Service / Bind Application

- Use cf CLI or App Manager Console:



The screenshot shows the 'CONFIGURE INSTANCE' dialog from the Pivotal App Manager. It includes fields for 'Instance Name' (set to 'name-of-service'), 'Add to Space' (set to 'development'), and 'Bind to App' (set to '[do not bind]'). Below this is a 'SUBSCRIPTION TERMS' section with a bulleted list of service subscription details. At the bottom are 'Cancel' and 'Add' buttons. To the right of the dialog is a black terminal window containing the command: `>$ cf bind-service some-app bpm`.

CONFIGURE INSTANCE

Instance Name: name-of-service

Add to Space: development

Bind to App: [do not bind]

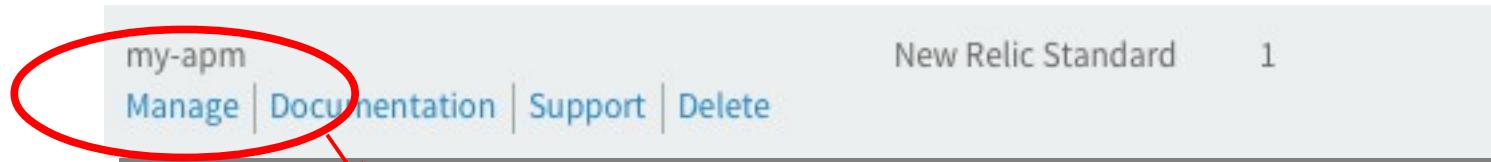
SUBSCRIPTION TERMS

- A monthly subscription charge is added to the bill at the start of every monthly service period.
- Cancel a service subscription by deleting the instance.
- Credits are not issued for the unused portion of a monthly subscription period.
- Your subscription will be billed monthly starting today.

Cancel Add

```
>$ cf bind-service some-app bpm
```

# Access via *Manage* Link in App Manager



A screenshot of the New Relic App Manager interface. At the top, there's a header with the application name "my-apm", plan level "New Relic Standard", and count "1". Below the header, a red circle highlights the "Manage" link in the navigation bar. An arrow points from this highlighted area down to the main monitoring dashboard below. The dashboard shows an application named "cf-session-demo" with tabs for Overview, Map, Transactions, External services, and JVMs. The Overview tab is selected, displaying a chart titled "Web transactions response time". The chart shows a sharp spike reaching up to 250 ms at approximately 11:19, labeled "13 ms" and "App server". The left sidebar contains links for Applications, Browser, Transactions, Mobile, Servers, Dashboards, Plugins, and Tools.

# Lab

Configuring Third-Party Application  
Performance Management Tools with  
Cloud Foundry

# Roadmap

- Log Management
- Application Performance Management
- **Autoscaling**
- Zero-Downtime Deployments

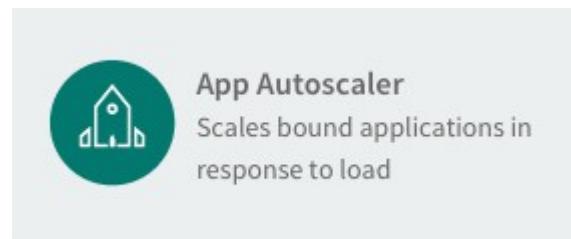
# Scaling Options



- CF allows horizontal scaling
  - Controlling the # of instances of an application running
  - All behind a common router (load balancer)
  - Controllable via the manifest, **cf** command line, or App Manager console
- All options require manual intervention

# AutoScaling

- CF can allow applications to be automatically scaled
  - “AutoScaling”
- System load can be used as a trigger in place of manual interaction.



- Autoscaling Service
  - Must be installed by administrator
  - Not available in PWS

# AutoScaling Service – Steps

## 1) Create the service

### 1) Select the desired plan

## 2) Bind to Application

## 3) Set desired scaling parameters

### 1) Add instance whenever high threshold is reached

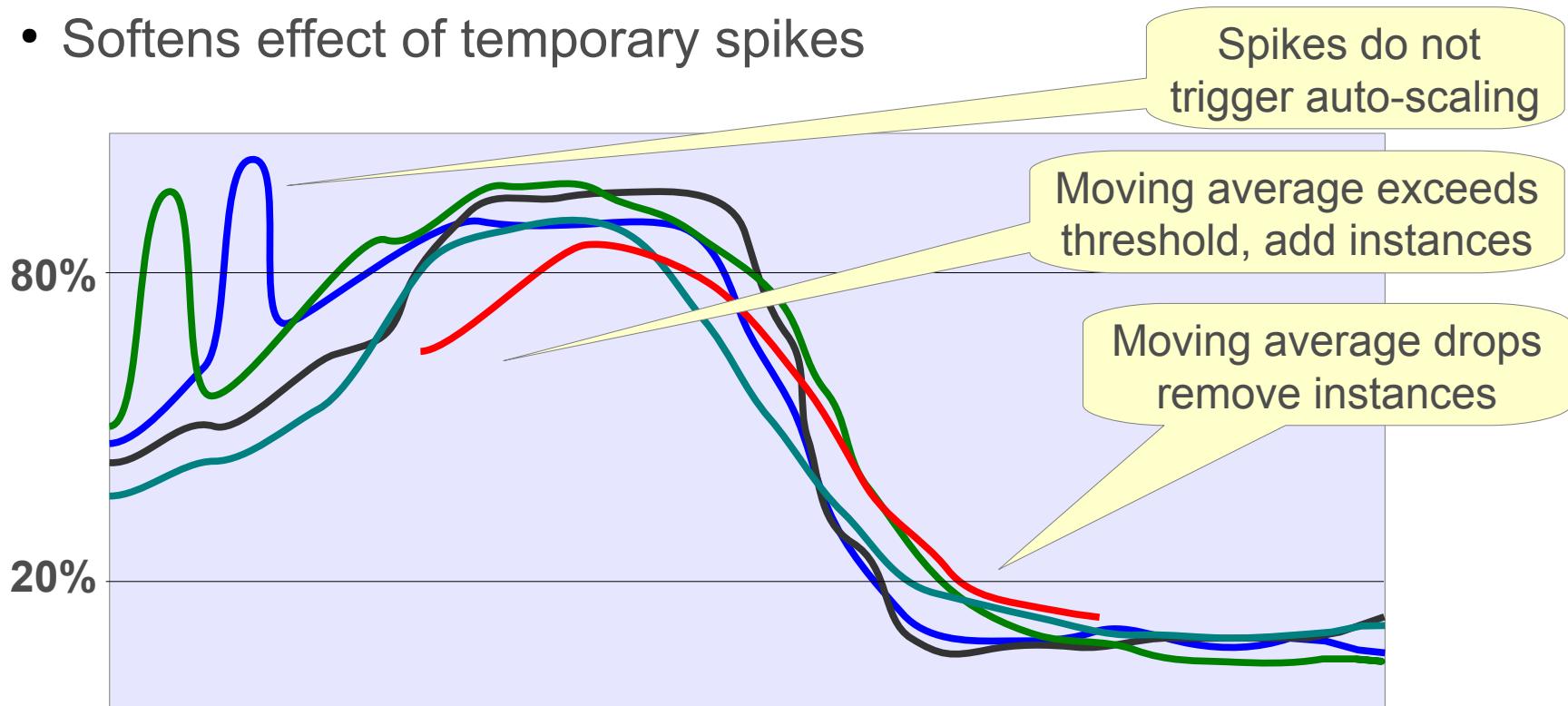
### 2) Subtract instance whenever low threshold is reached

The screenshot shows the Pivotal AutoScale interface with the following details:

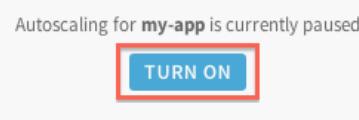
- my-app**: The application name.
- INSTANCES**:
  - min: 2
  - max: 5
- CPU THRESHOLDS**:
  - low: 20%
  - high: 80%
- LAST EVENT**: Scaled app from 1 to 2 instances on 09/11/14 @ 23:15:56 UTC.
- SCHEDULING**: 0 rules, Next: No Upcoming Events.

# AutoScaling – Moving Average

- Scaling activity based on moving averages
  - Softens effect of temporary spikes



- Manual scaling disables AutoScaling
  - Re-enable:



# AutoScaling – Scheduling

- Autoscaling events can be scheduled
- Changes autoscaling behavior on the given date / time.
- May be single event or recurring

SCHEDULING: MY-APP SERVER TIME: 09/12/14 @ 21:27:22 UTC X

+ New UNSAVED

11/21/2014 / 02:00 X  
5 to 10 instances

Mon, Fri / 04:00 II X  
10 to 20 instances

on  at   
\*All times are UTC

repeats every  S  M  T  W  T  F  S

min  low   
max  high

ADD

# Roadmap

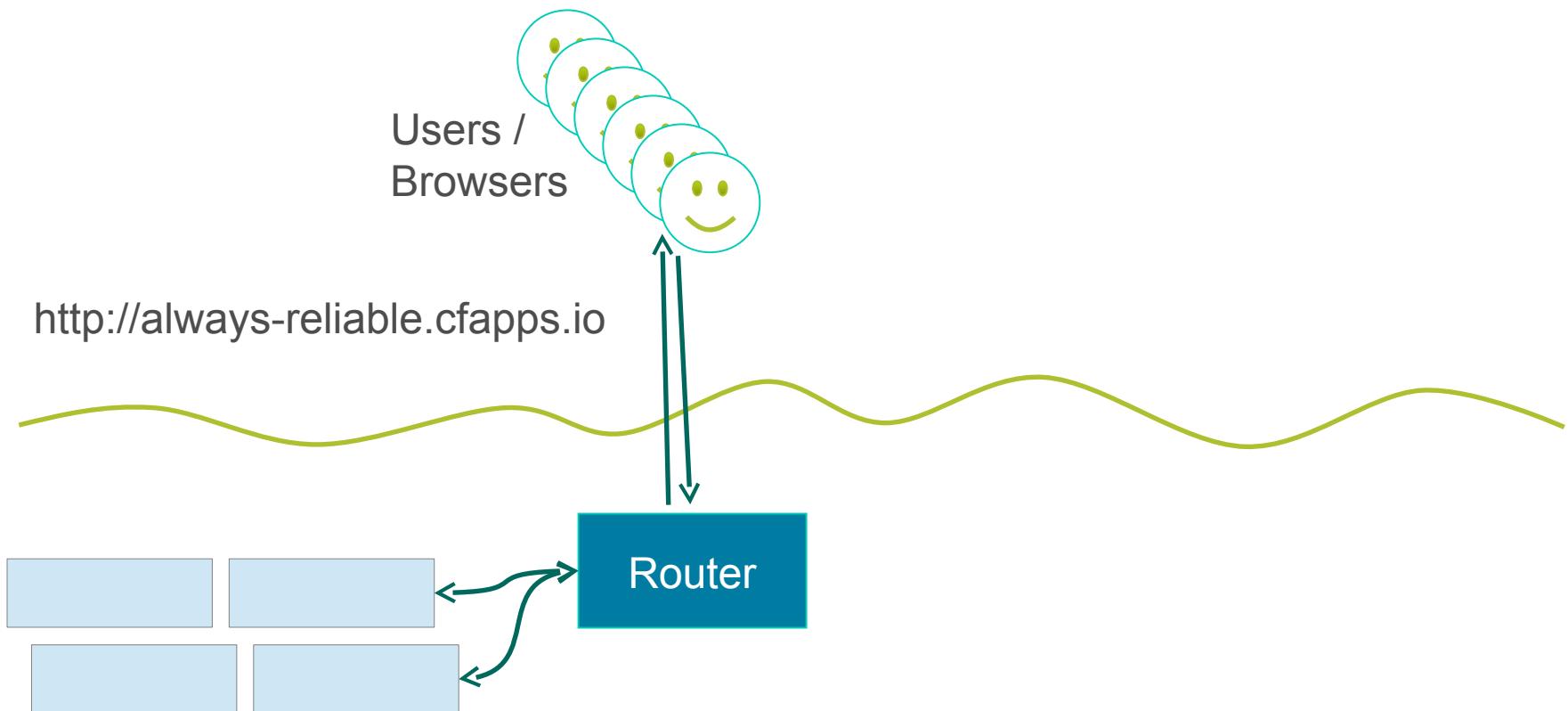
- Log Management
- Application Performance Management
- Autoscaling
- **Zero-Downtime Deployments**

# Blue / Green Deployments

- `cf push` causes CF to stop old instances, then start new
  - Bad news if you are a user
- Blue / Green Deployment eliminates user downtime
  - Also known as “zero-downtime” or “A / B” Deployment
  - Avoids “*Site Temporarily Down for Maintenance*”
- How it works:
  - Run 2 versions of an application (new / old)
    - NOT merely multiple instances.
  - Alter routes for applications to transfer traffic.
  - **Note:** Users can still experience session loss.

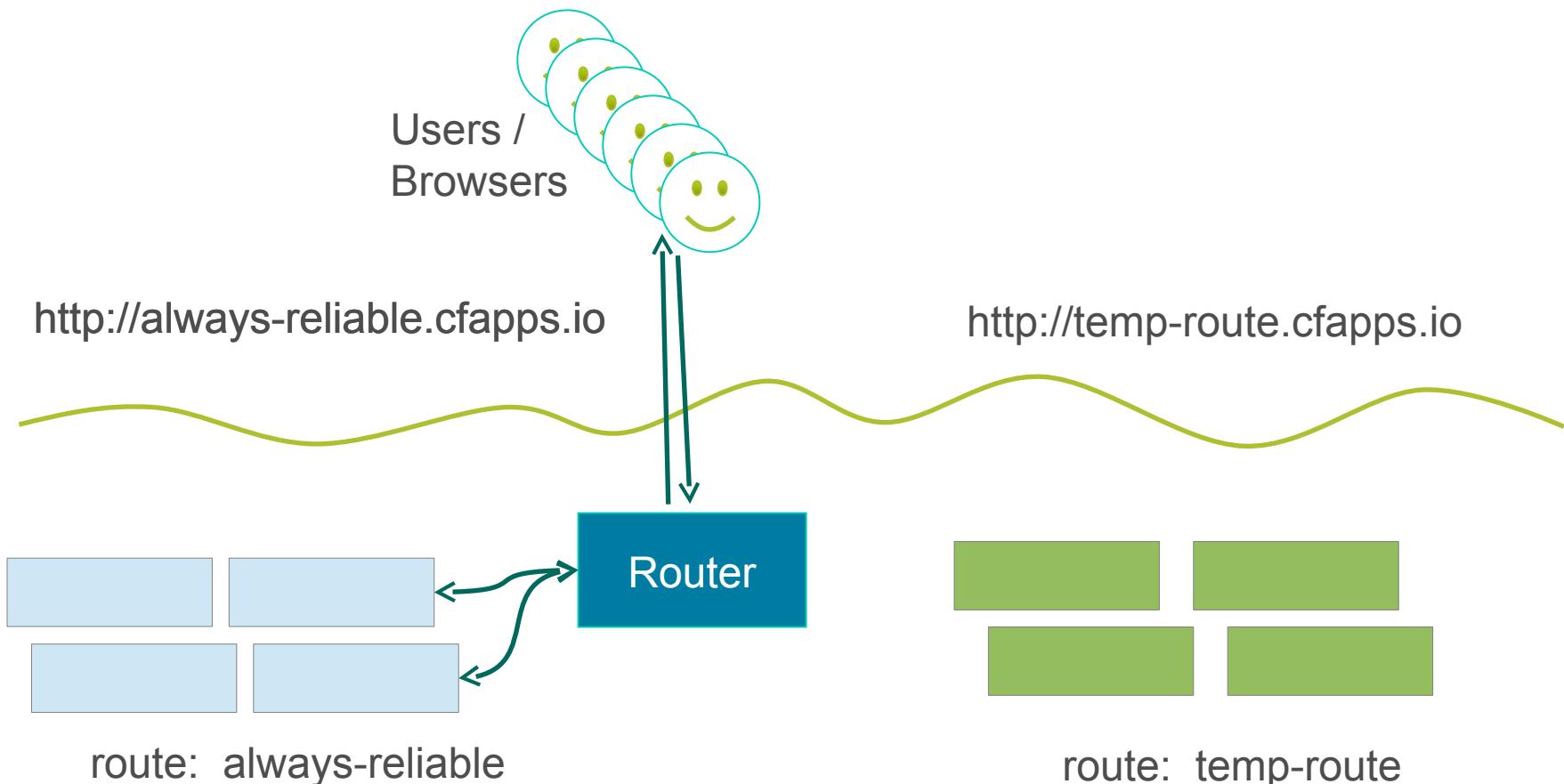
# Blue Green Deployment – Existing App

```
cf push blue -p app.war -n always-reliable -i 4
```



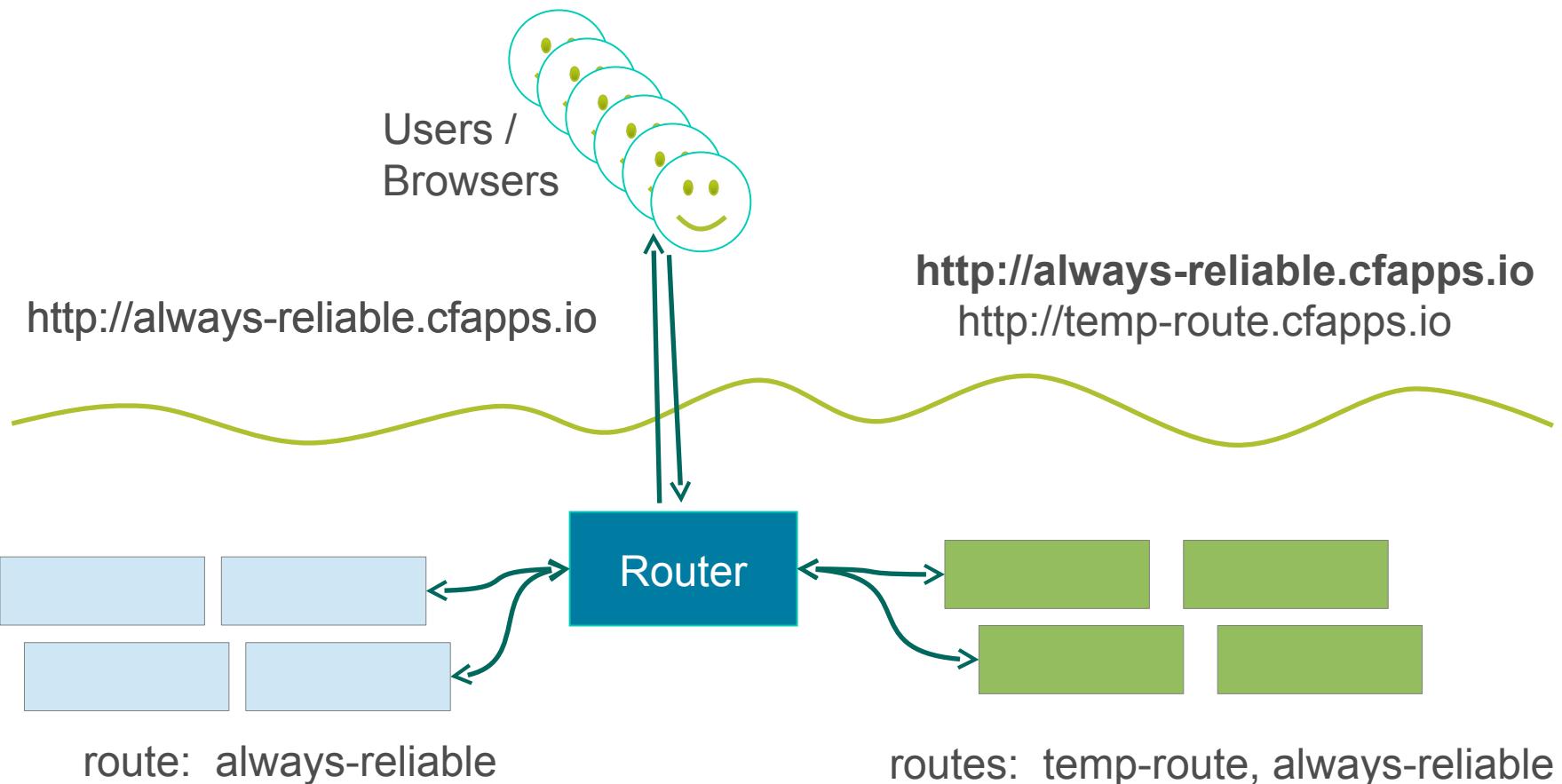
# Blue Green Deployment – New Version

```
cf push green -p app.war -n temp-route -i 4
```



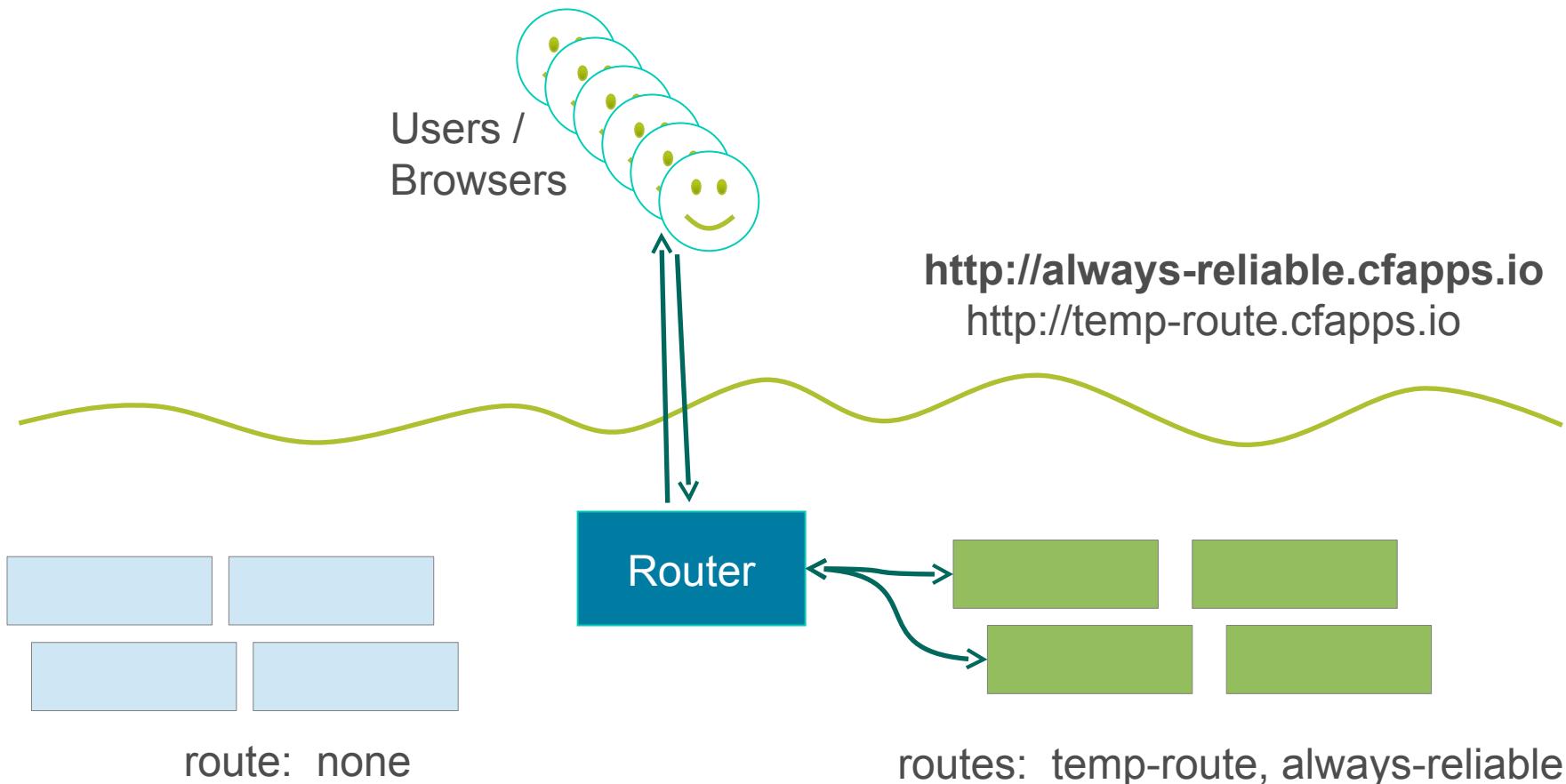
# Blue Green Deployment – Duplicate Route

```
cf map-route green cfapps.io -n always-reliable
```



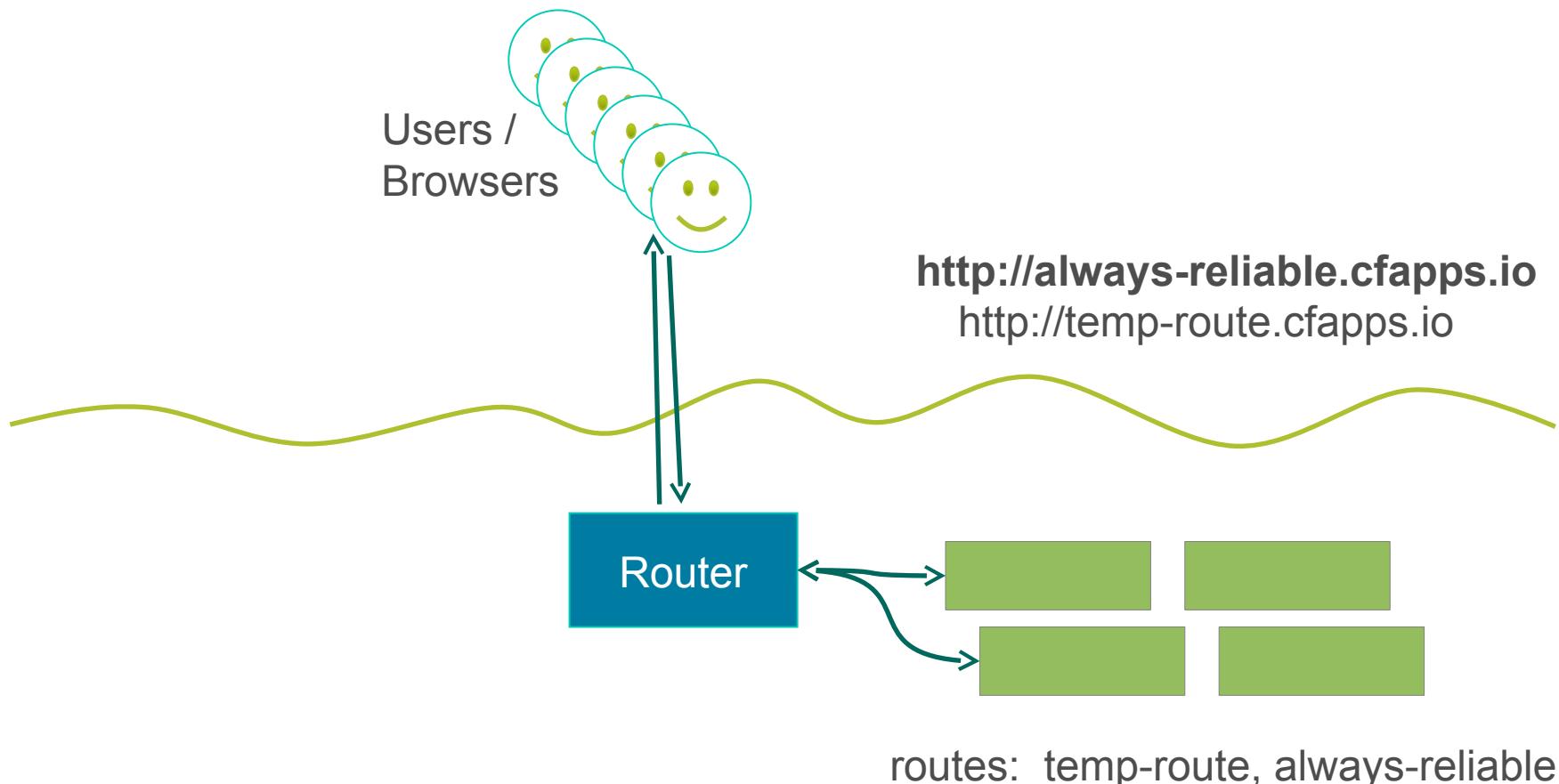
# Blue Green Deployment – Disconnect Blue

```
cf unmap-route blue cfapps.io -n always-reliable
```



# Blue Green Deployment – Remove Blue

`cf delete blue`

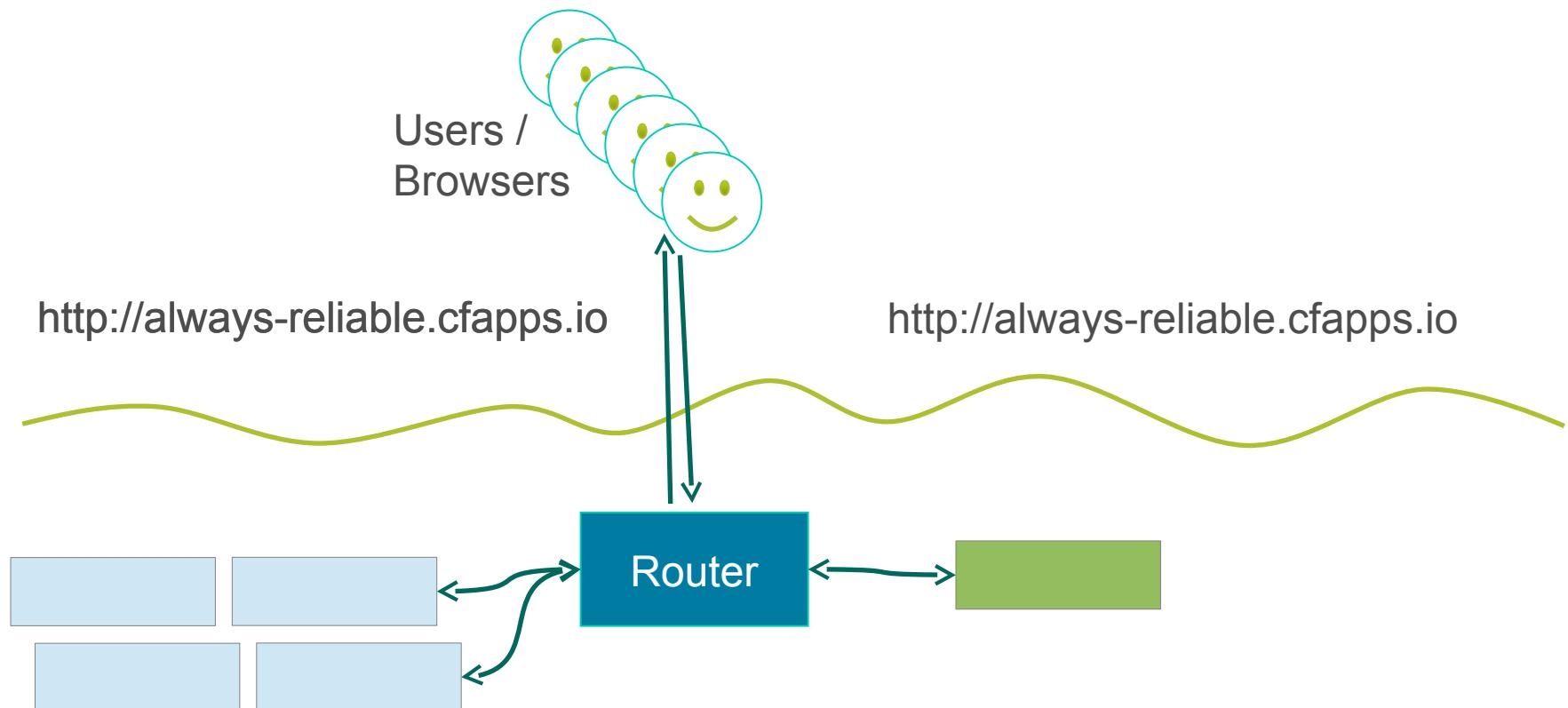


# Canary Deployments

- Variation on the Blue/Green Deployment.
    - “Canary in a coal mine”
1. Start with many 'blue' instances
  2. Start a single 'green' instance, route traffic to both
    - Green instance is the 'Canary'
  3. Watch the Canary
    - If it behaves, scale 'green' up / scale 'blue' down.
  4. Continue monitoring and scaling until zero blue instances

# Canary Deployment – Push The Canary

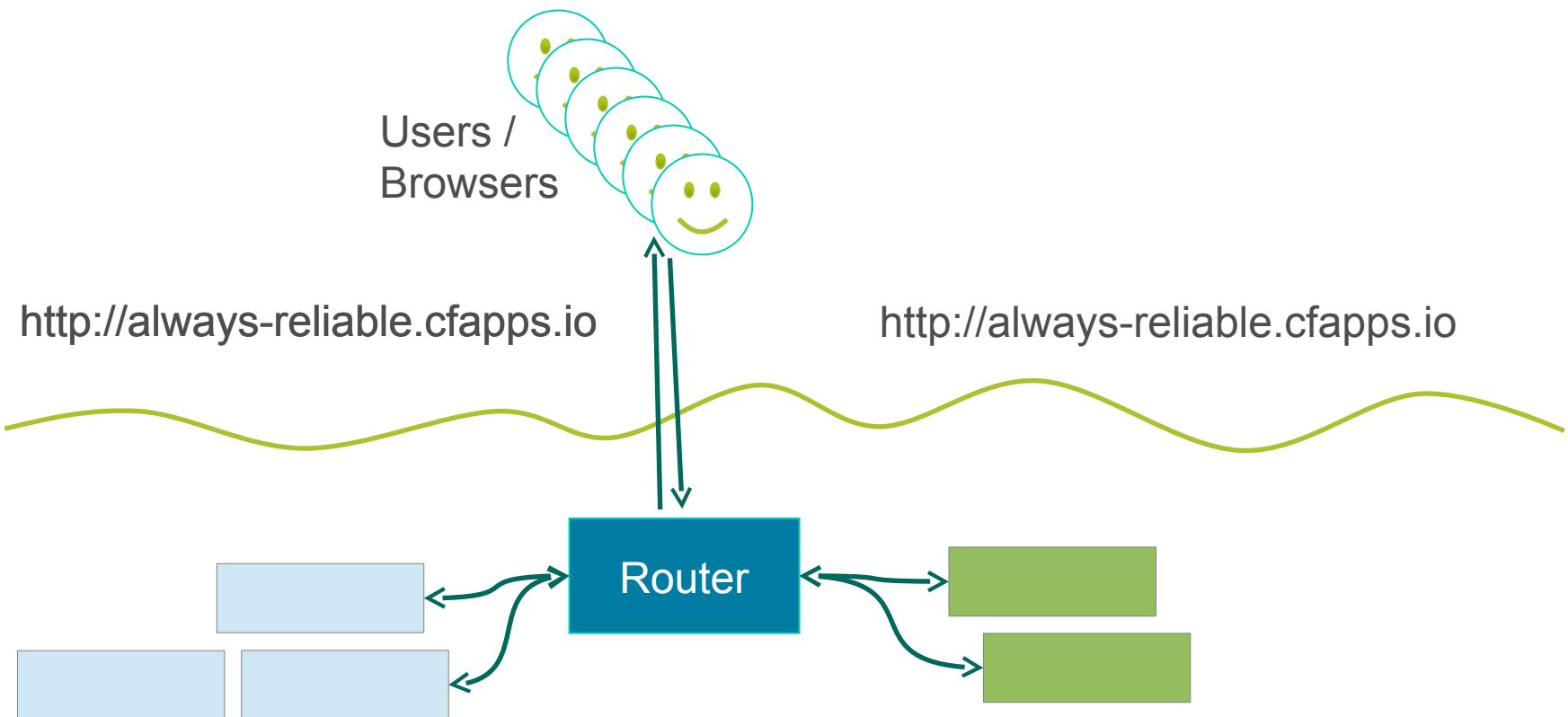
```
cf push green -p app.war -n always-reliable -i 1
```



# Canary Deployment – Scale Traffic

```
cf scale green -i 2
```

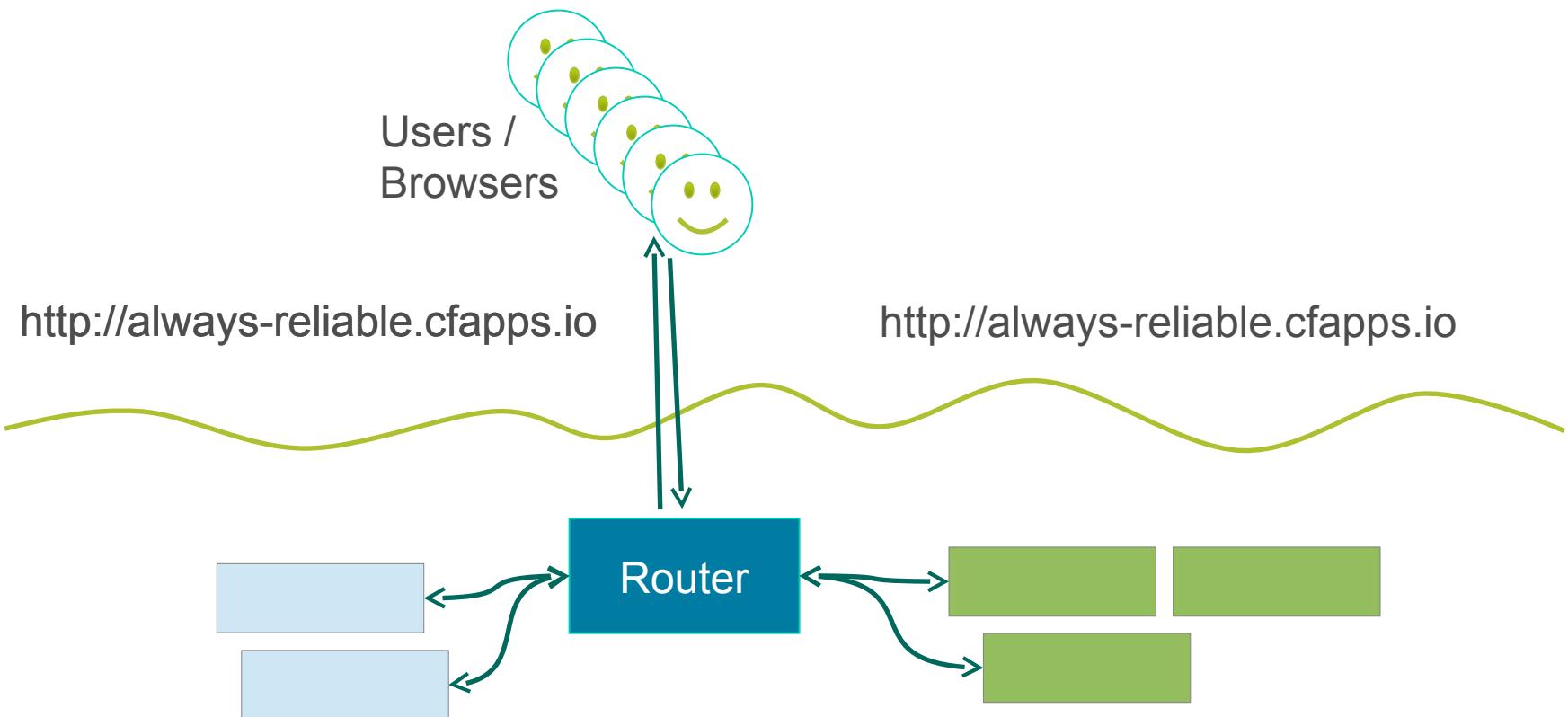
```
cf scale blue -i 3
```



# Canary Deployment – Scale Traffic

```
cf scale green -i 3
```

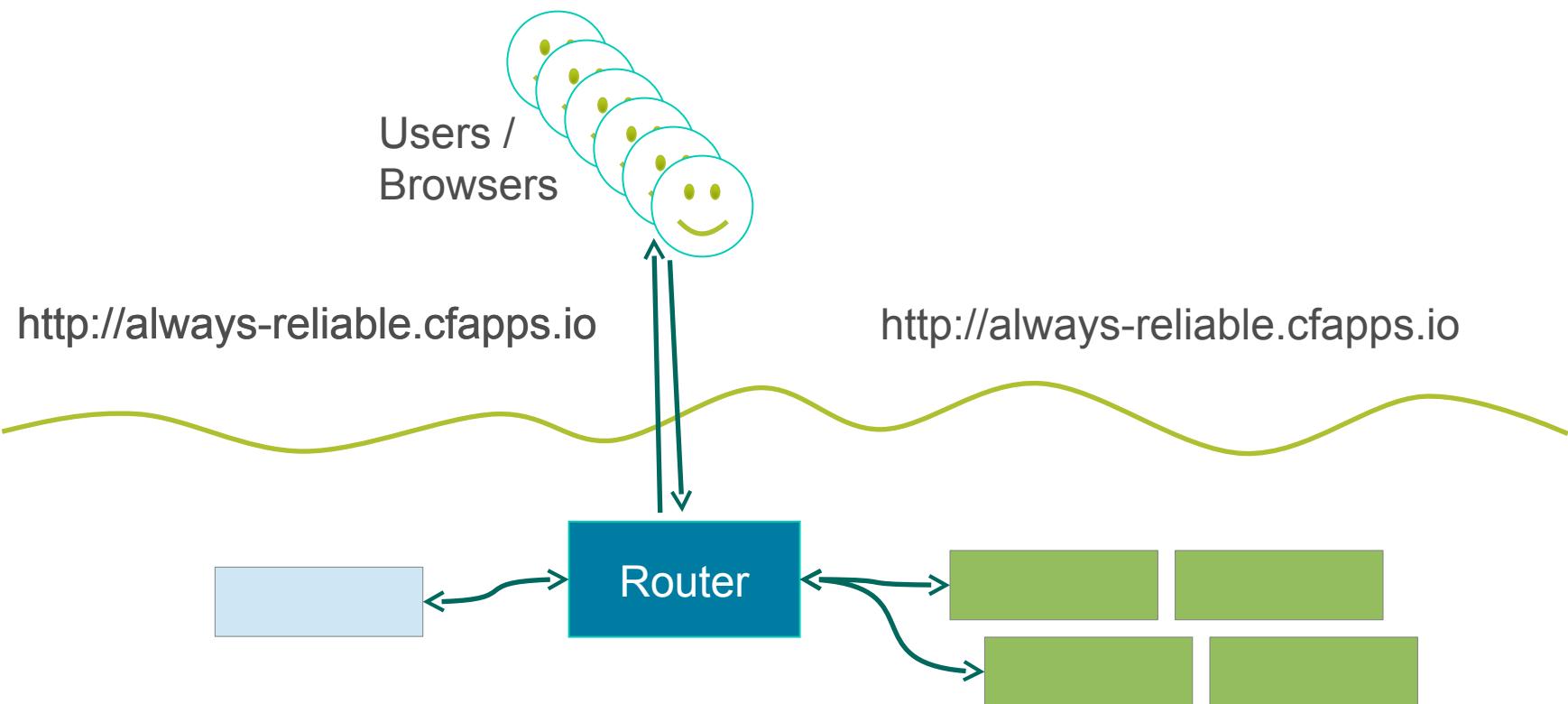
```
cf scale blue -i 2
```



# Canary Deployment – Scale Traffic

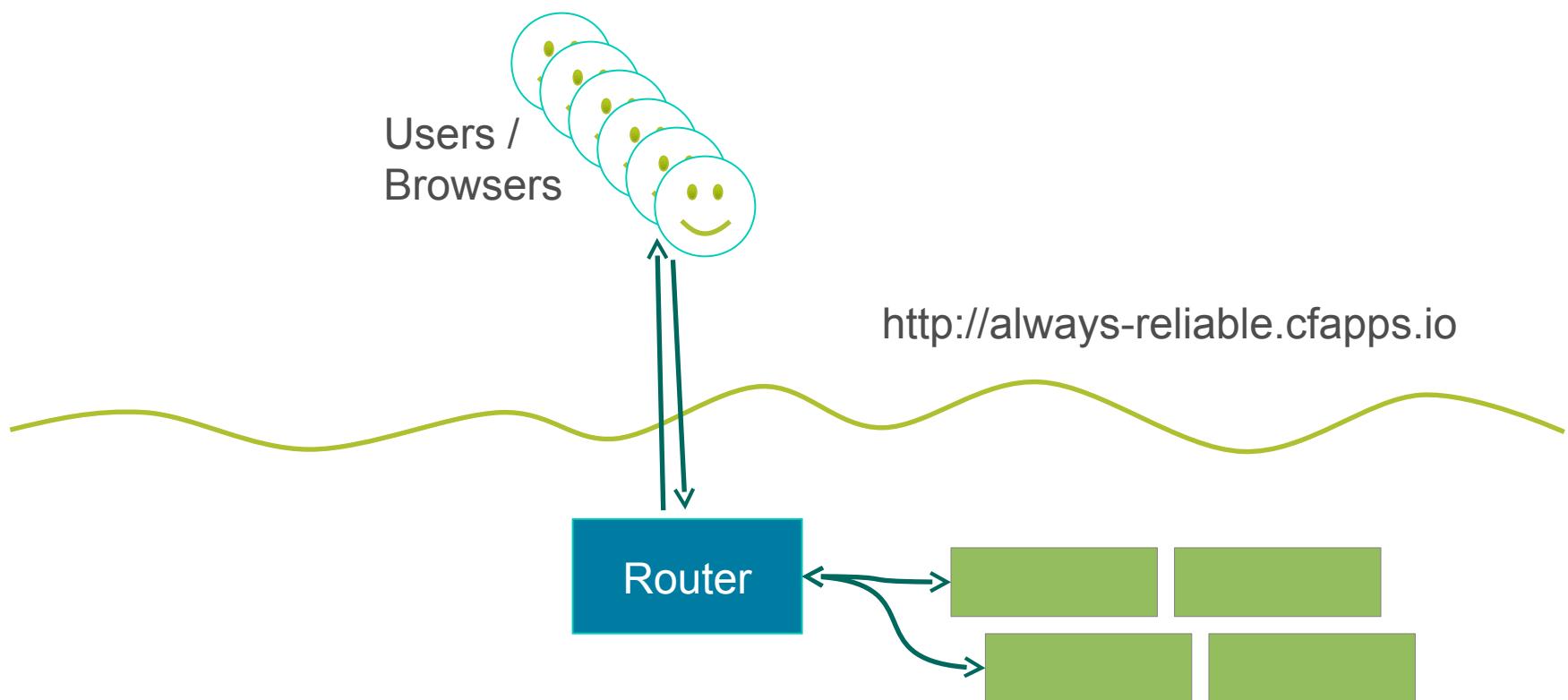
```
cf scale green -i 4
```

```
cf scale blue -i 1
```



# Canary Deployment – Scale Traffic

`cf delete blue`



# Lab

## Blue / Green Deployment

# Summary

- After completing this lesson, you should have learned:
  - How to integrate with third-party log manager
  - How to integrate with APM services
  - How to employ App Autoscaling
  - How to deploy with zero downtime.



# Cloud Foundry Architecture

## What's Inside the Box?

Overview of Architectural Components  
DEA and DIEGO

Pivotal

# Overview

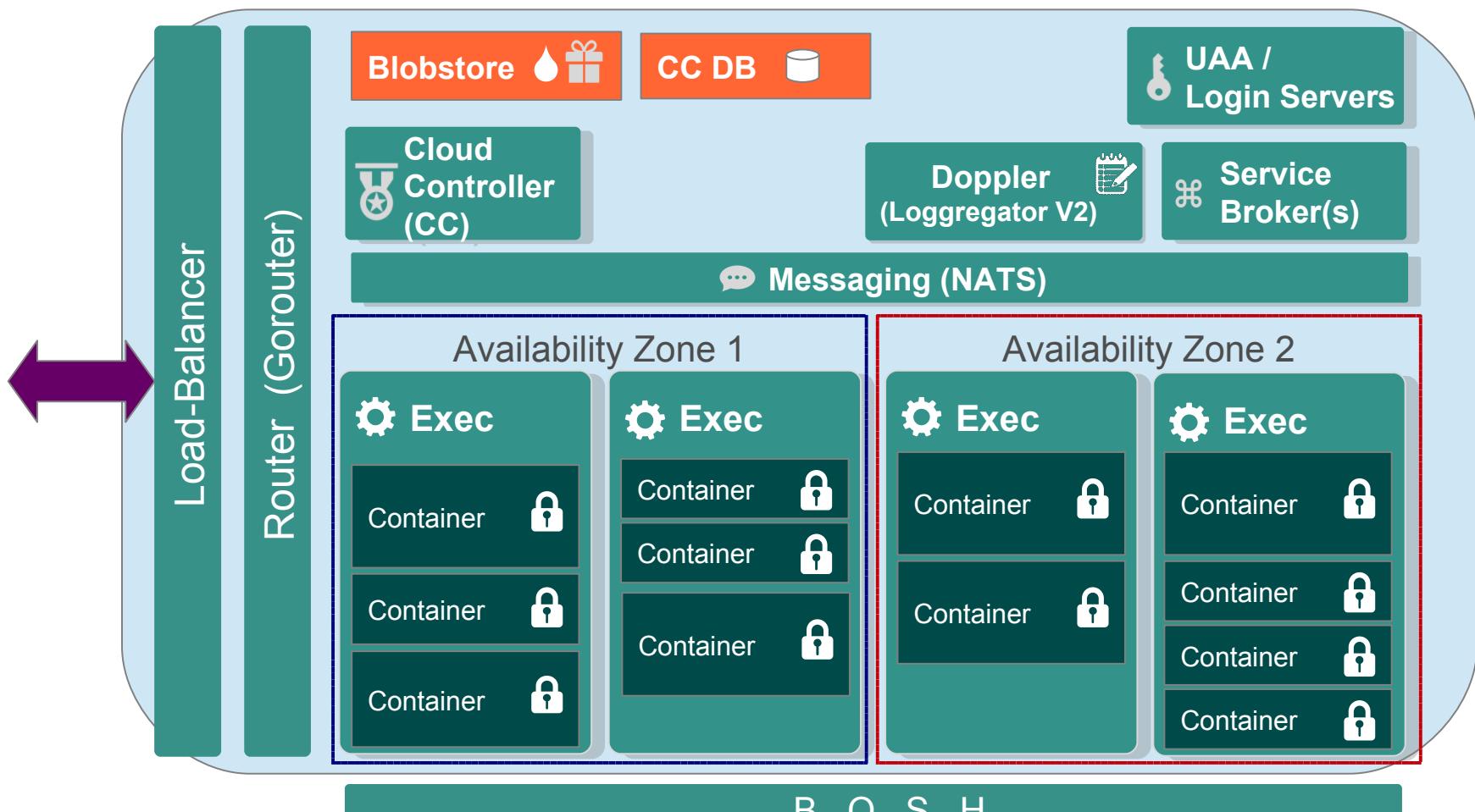
- After completing this lesson, you should understand:
  - The components that work together inside Cloud Foundry
  - The different levels of application containment
    - And what they do
  - How failed application instances can be automatically restarted

# Roadmap

- **Architectural Components**
  - DEA Architecture (up to 1.4)
  - Diego Architecture (from 1.5+)
- Handling Application Instance failure

# Pivotal CF Components

HA Zone 1  
HA Zone 2



Exec – Execution Agent

UAA – User Authorization & Authentication



Pivotal™



# Load Balancer (HA Proxy)

## Description

- HAProxy is separate component from the Router
  - Sits in front of the router
  - Only one instance allowed
    - Single-point of failure
- Cloud Foundry can only have one IP address visible to outside world
  - This is the Load Balancer
  - It passes requests to the Router(s) to send to the right CF component

## Responsible For

- Load-balancing across multiple routers (if used)
- SSL Termination
- **Note:** *Typically need to configure your own HAProxy or other load-balancer (such as F5 or NSX) for improved performance, SSL handling and high availability,*



# Router

## Description

- Routes all incoming HTTP traffic
  - System traffic (cf commands)
  - Application traffic
- Maintains dynamic routing table for each load-balanced app instance
  - Knows IP addresses and ports.
- Multiple routers possible
  - Configured by admin in Ops Manager
- From CF 1.4 rewritten in Go – the *GoRouter*

## Responsible For

- Load balancing across application instances
- Maintaining an active routing table
- Access logs
- Supports web-sockets
- In CF 1.6 will *also* do SSL termination



# Cloud Controller

## *Description*

- Command and Control
  - Responds to clients (CLI, Web UI, Spring STS)
  - Account and provisioning control.
- Provides RESTful interface to domain objects
  - Apps, services, organizations, spaces, service instances, user roles, and more ...
- Multiple Cloud Controllers possible
  - Configurable by Admin

## *Responsible For*

- Expected Application state, state transitions, and desired convergence
- Permissions/Authorization
- Organizations/Spaces/Users
- Services management
- Application placement
- Auditing/Journaling and billing events
- Blob storage



# BlobStore

## *Description*

- Storage for binary large objects
- Eliminates need for upload / re-staging when scaling applications
- Currently NFS mounted storage
  - Or Amazon S3 store

## *Responsible For*

- Stores uploaded application packages (cf push)
- Stores Droplets



# Cloud Controller Database (CCDB)

## *Description*

- Storage for application metadata
- Used exclusively by the Cloud Controller

## *Responsible For*

- Stores information on
  - Application name
  - # of instances requested
  - Memory limit
  - Routes
  - Bound services
- A Postgres DB instance



# Availability Zones

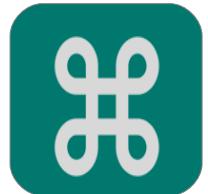
## *Description*

- Separate shared points of failure such as a power, network, cooling, roof, floor.
- Allows Cloud Controller to locate instances on separate zones to boost redundancy
- Marked on diagram as:

 HA Zone 1  
 HA Zone 2

## *Responsible For*

- Enhancing application redundancy
- Provides one of the layers of High Availability



# Service Broker

## Description

- Provide an interface for native and external 3<sup>rd</sup> party services
  - Service processes run on Service Nodes
  - Or with external as-a-service providers
- *Examples*
  - Mail server
  - Database,
  - Messaging
  - Many more ...
- Typically one service broker for each marketplace service.

## Responsible For

- Advertising service catalog
- Makes create/delete/bind/unbind calls to service nodes
- Requests inventory of existing instances and bindings from cloud controller for caching, orphan management
- SaaS marketplace gateway



# Messaging (NATS)

## Description

- Fast internal messaging bus
- Manages system-wide communication
- Uses a publish-and-subscribe mechanism
- *Not-Another Transport System*
- Single NATS per Cloud Foundry installation

## Responsible For

- Non-Persistent messaging
- Pub/Sub
- Queues (app events)
- Directed messages (INBOX)



# Loggregator

## Description

- Master logging process
  - Accepts logs from application instances
  - Accepts logs from other CF components
- Make log data available to external *sinks*
- Uses Diego's *Doppler* server since CF 1.4

## Responsibilities

- Non-Persistent, temporary log-storage
- Accumulates logs from multiple sources and aggregates by application
- Provides logs to external sources such as **cf logs** or App Manager console
- Supports log “drains” to syslog servers



# UAA and Login Services

## *Description*

- UAA = “User Authorization and Authentication”
- Provides identity, security and authorization services.
- Manages third party OAuth 2.0 access credentials
- Can provide application access & identity-as-a-service for CF apps
- Composed of
  - UAA Server
  - Command Line Interface
  - Library.
- Multiple UAA/Login Servers possible

## *Responsible For*

- Token Server
- ID Server (User management)
- OAuth Scopes (Groups) and SCIM
- Login Server
  - UAA Database
  - SAML support (for SSO integration) and Active Directory support with the VMWare SSO Appliance
- Access auditing



# Cloud Foundry BOSH

## Description

- Tool chain for managing large scale distributed systems
  - Release engineering
  - Deployment and lifecycle management
- Continuous and predictive updates with minimal downtime
- Control primitives (CPI) written for each underlying infrastructure provider
  - The 'glue' between CF and underlying IaaS

## Responsible For

- IaaS installer
  - Currently vSphere
- VM creation and management
- Continuous and predictive updates with minimal downtime
- High Availability
  - Restarts failed CF *internal* processes
  - Restarts Execution Agent VMs
- CPI (Cloud Provider Interface) to control underlying infrastructure (IaaS) primitives

# Roadmap

- **Architectural Components**
  - DEA Architecture (up to 1.4)
  - Diego Architecture (1.5+)
- Handling Application Instance failure

# DIEGO vs DEA

- Cloud Foundry up to 1.4 had a different architecture
  - Apps ran inside Warden containers in Droplet Execution Agents (DEAs)
  - Health Manager handled restart of failed app instances
  - Code written in Ruby, running on Linux
- DIEGO = DEA in Go
  - Rewrite in Google GO language
  - Major refactoring of the Cloud Controller and internals
  - Available since 1.5
- *We cover both in this section*
  - But from the outside, CF works the same way

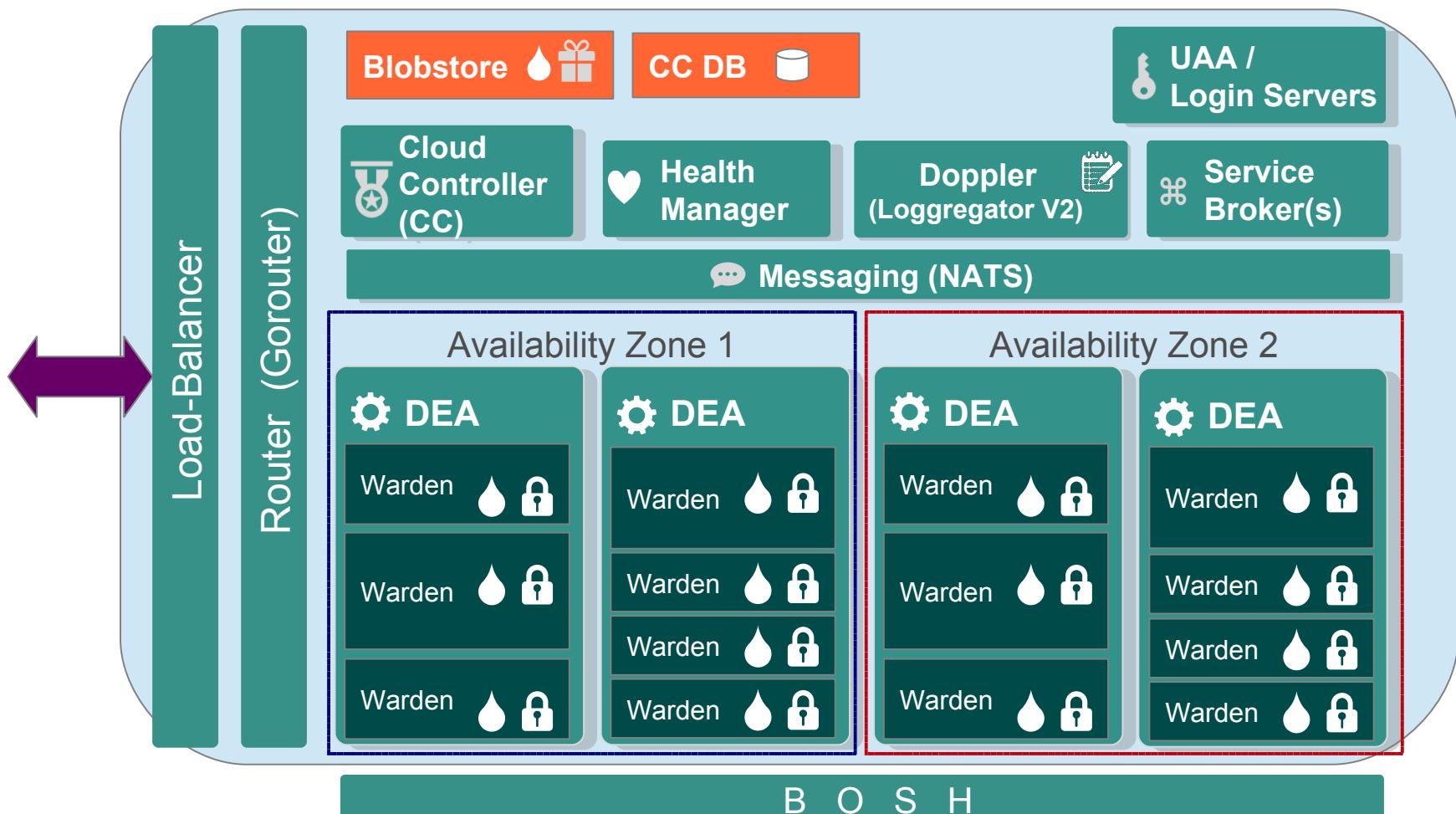
# DEA Architecture

- Everything so far applies to *all versions of PCF*
- This section covers
  - DEAs and Warden containers
    - Where applications run
  - Health Manager
    - Ensures instance recovery
    - *More on this later*

# Pivotal CF Components – V1.4

HA Zone 1

HA Zone 2



DEA – Droplet Execution Agent

UAA – User Authorization & Authentication



Pivotal™



# Droplet Execution Agents (DEA)

## *Description*

- Secure and fully isolated containers
  - Actually a Linux VM
- Responsible for an Apps lifecycle
  - Building, starting and stopping Apps as instructed
- Periodically broadcast messages about their state
  - Via the NATS message bus.
- Typically many DEAs in a Cloud Foundry installation

## *Responsible For*

- Managing Linux containers
  - Pivotal's Warden containers
- Monitoring resource pools
  - Process
  - File system
  - Network
  - Memory
- Managing app lifecycle
- App log and file streaming
- DEA heartbeats
  - via NATS to Cloud Controller & Health Manager

# Warden Container

## Description

- Isolated Process
  - Safe, lightweight alternative to full VM
  - Runs a Droplet
- Pivotal's secure implementation of LXC (Secure Linux Container)
- Isolates applications from each other
- Allows multiple applications running on each VM

## Responsible For

- Isolates applications running on the same VM
  - Individual failures does not affect other applications on the VM
  - Uses kernel namespaces to isolate network, disk, memory and CPU
  - Uses Linux *cgroups* to do resource management
- Secures applications from environment
- Runs Droplets



# Health Manager

## *Description*

- Monitors application uptime
- Listens to NATS message bus for mismatched application states (expected vs. actual)
  - Cloud Controller publishes expected state
  - DEAs publish actual state
- State mismatches are reported to the Cloud Controller.
- Multiple Health Managers possible

## *Responsible For*

- Maintains the **actual state** of apps
- Compares to **expected state**
- Sends suggestions to make actual match expected
  - Cannot make state changes itself – only Cloud Controller can do that!

# Roadmap

- **Architectural Components**
  - DEA Architecture (up to 1.4)
  - **Diego Architecture (1.5+)**
- Handling Application Instance failure

## Disclaimer:

DIEGO is *work in progress* (Sep 2015)

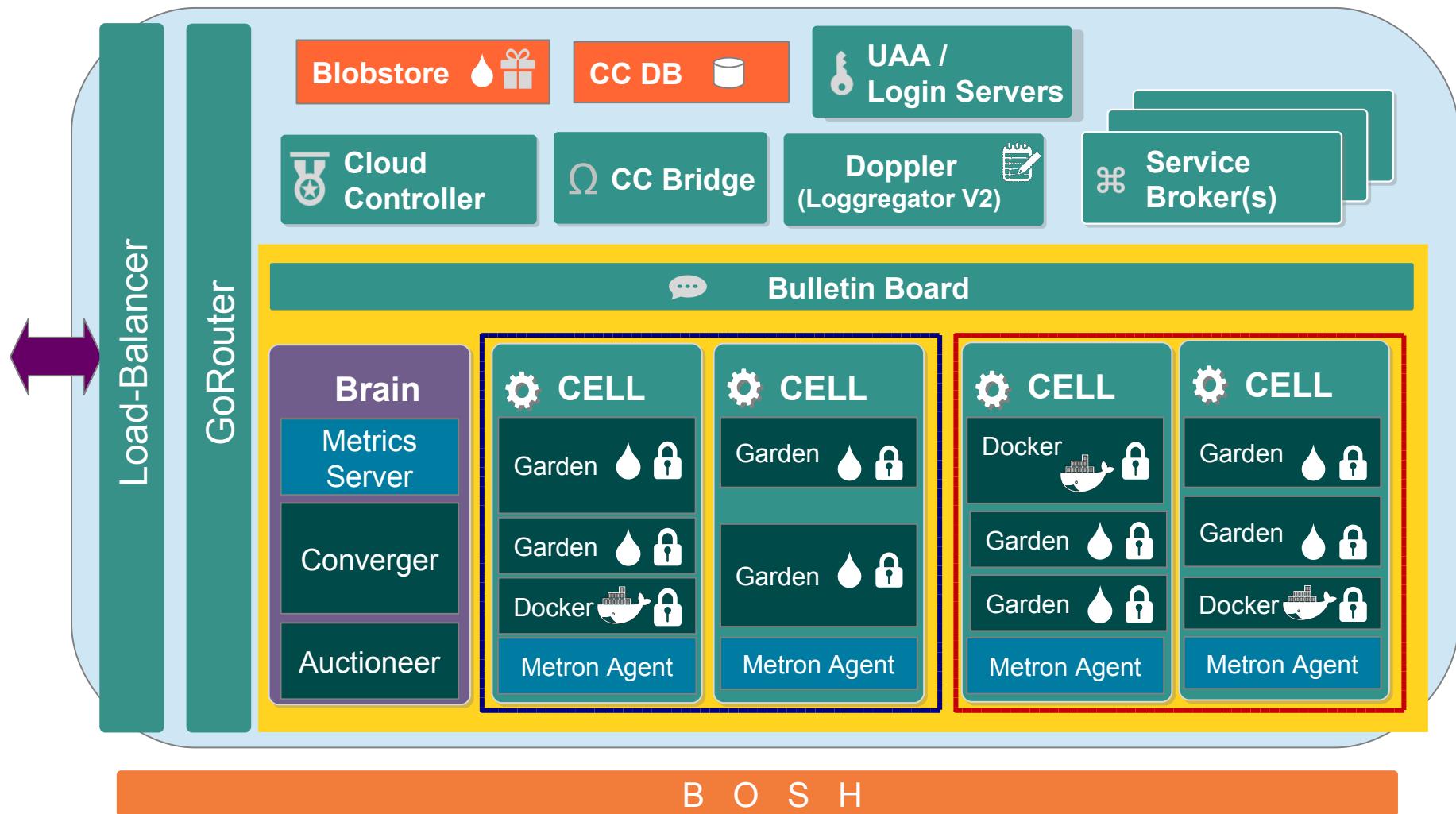
What is described here is just an overview and *subject to change*

# DIEGO

- Available in Cloud Foundry since V1.5
  - Refactors much of the functionality that used to be in the Cloud Controller
    - Cloud Controller still looks the same to the outside world
  - New components in the “Brain”
    - Auctioneer, Converger, Metrics Server
  - DEAs replaced by Cells
  - Warden containers can have multiple implementations
    - Not just for running Droplets
- From a CF user perspective – *nothing changes*

 DIEGO  
 HA Zone 1  
 HA Zone 2

# Cloud Foundry – Diego





# CC Bridge

## Description

- Translate app-specific requests into the generic language of LRP and Task
  - For example requests to stage, start, stop, scale applications
- Allows existing CF components to interact with Diego

## Responsibilities

- Places requests (actions) onto the Bulletin Board to be actioned by a Cell's "Executor"
- Ensures that these actions take place (eventually)
- Tracks running processes
  - Provides info on running tasks and LRPs to Cloud Controller
  - Accepts desired state of tasks and LRPs *from* Cloud Controller



# Bulletin Board (BB)

## Description

- Holds CF “actions”
  - Typical CF actions are to start/stop application instances or one-off task processes
- Essentially an action queue, currently backed by *etcd*
  - Other implementations likely
- Decouples control components (like the CC) from Diego
  - CC manages applications
  - Diego talks tasks and LRPCs

## Responsibilities

- Central co-ordination for activities (actions) within CF

# etcd



## Description

- A fast distributed open-source key-value store
- Written in Go and uses the Raft protocol\*

## Responsibilities

- Cluster coordination and state management
- The *consistent* store at the heart of diego

\* [https://www.youtube.com/embed/06cTPhi-3\\_8](https://www.youtube.com/embed/06cTPhi-3_8)

# Metrics Server



## Description

- Central co-ordination for system and application metrics

## Responsibilities

- Reads metrics from the *Metron* agents and publishes them to Loggregator (Doppler)



# Converger

## Description

- Makes sure that the system is *eventually consistent*
- Refactoring of instance recovery functionality that used to be in the old Heath Manager and the Cloud Controller

## Responsibilities

- Compare desired system state with actual system state
  - Take remedial action if they are different so desired and actual converge
- Place start (scale up), stop (scale down) and restart (instance recovery) requests on the Bulletin Board
- Spots if requests sit on BB too long
  - Forces BB to try them again



# Auctioneer

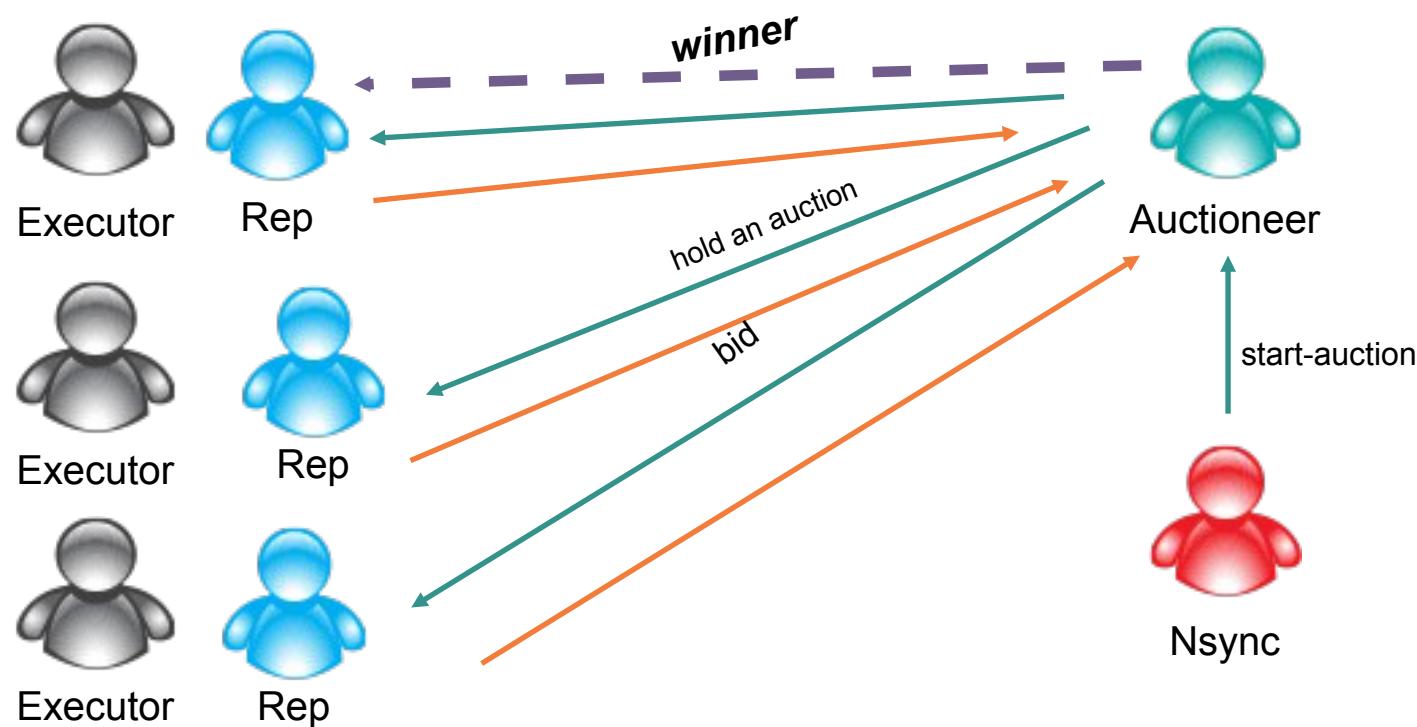
## Description

- Determining which Cell to run an application instance
  - Decision uses an Auctioning placement algorithm
  - Complex because distributed and asynchronous
- Also determines which instance to stop when scaling down
- One-off tasks not managed by auction yet
  - *But will in the future*

## Responsibilities

- Gather state from the appropriate sources of truth at decision-time
  - Run algorithm to decide
- Tell the cell that wins the auction to either start or stop the requested application instance
  - By placing the action on the BB

# Auctioneering Algorithm





# Cells

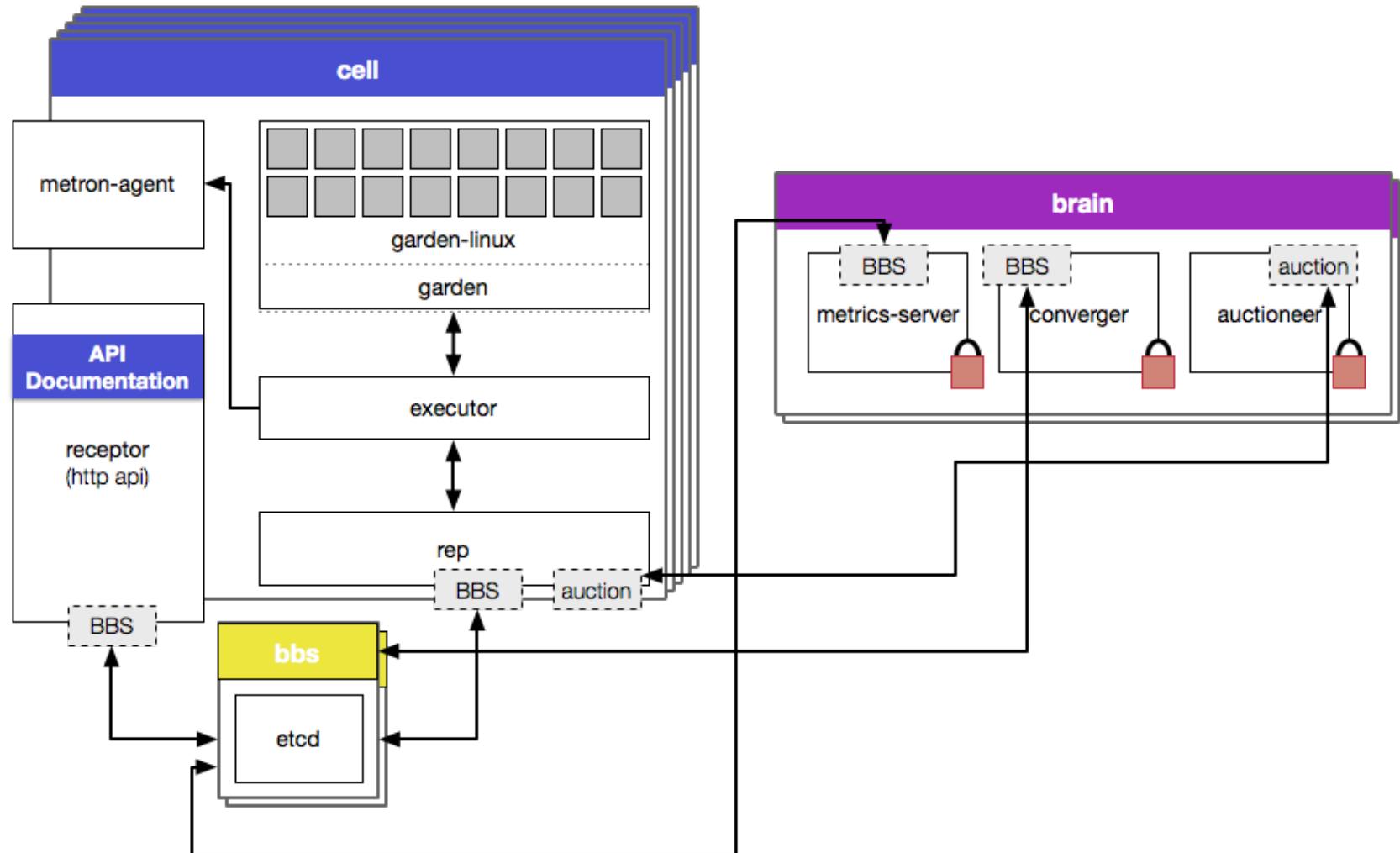
## Description

- Secure and fully isolated container
  - Typically a Linux VM
  - But *could* be a Windows VM
- Replace DEAs
- Periodically broadcast messages about their state
  - Via the NATS message bus.
- Typically *many* Cells in a Cloud Foundry installation
- Runs an *Executor* internally to manage containers

## Responsibilities

- Participate in app auctions
- Manage apps lifecycle (Executor)
  - Building, starting and stopping Apps as instructed
- Manage app containers
  - Pivotal's secure containers
  - Or any other container with same interface
    - For example to run Docker
- Monitor resource pools
  - Process, File System, Network, Memory

# Diego Internal Architecture





# Cell Internal Components

- Rep
  - Represents Cell
  - Mediates BBS comms
  - Bids on tasks
  - Schedules them on the Executor
- Receptor
  - Respond to request for tasks and desired LRPCs
  - Fetch information about currently running tasks and LRP instances
- Executor
  - Responsible for executing work (actions) given to it
  - Spins up a Garden container and executes the work encoded in Task/LRP
- Metron Agent
  - Monitors all containers on Cell
  - Provides logging and metrics to Loggregator for all apps/tasks running on Cell

# Roadmap

- Architectural Components
  - DEA Architecture (up to 1.4)
  - Diego Architecture (1.5+)
- **Handling Application Instance failure**

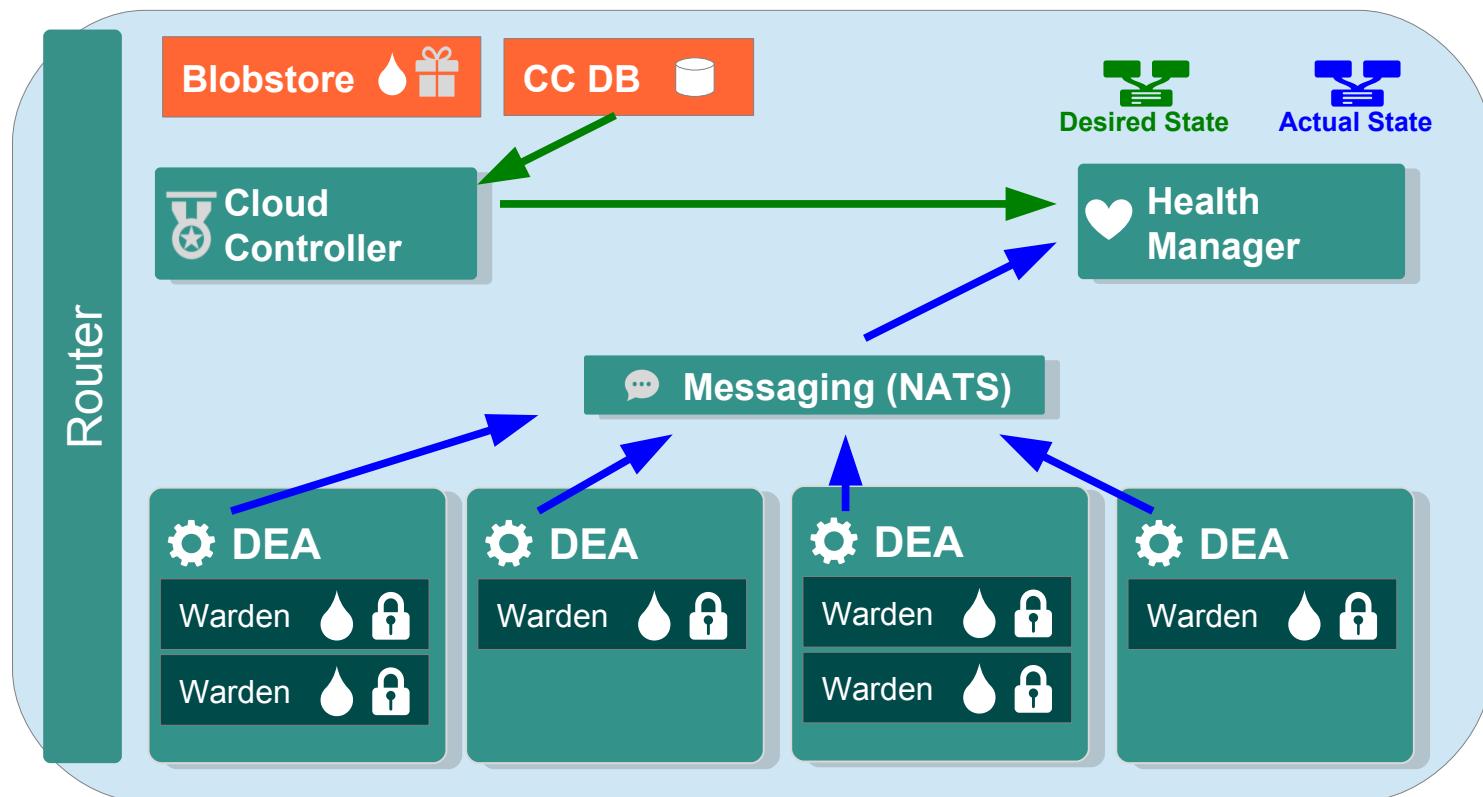


# Health Manager (DEA)

- Monitor applications to determine version, current state and number of instances
  - Actual state of an application is based on heartbeats from DEAs running the application
- Determine applications' desired state, version, and instances and compare to actual state
  - Desired state based upon the Cloud Controller database
- Direct Cloud Controller to take action to correct any discrepancies in the state of applications

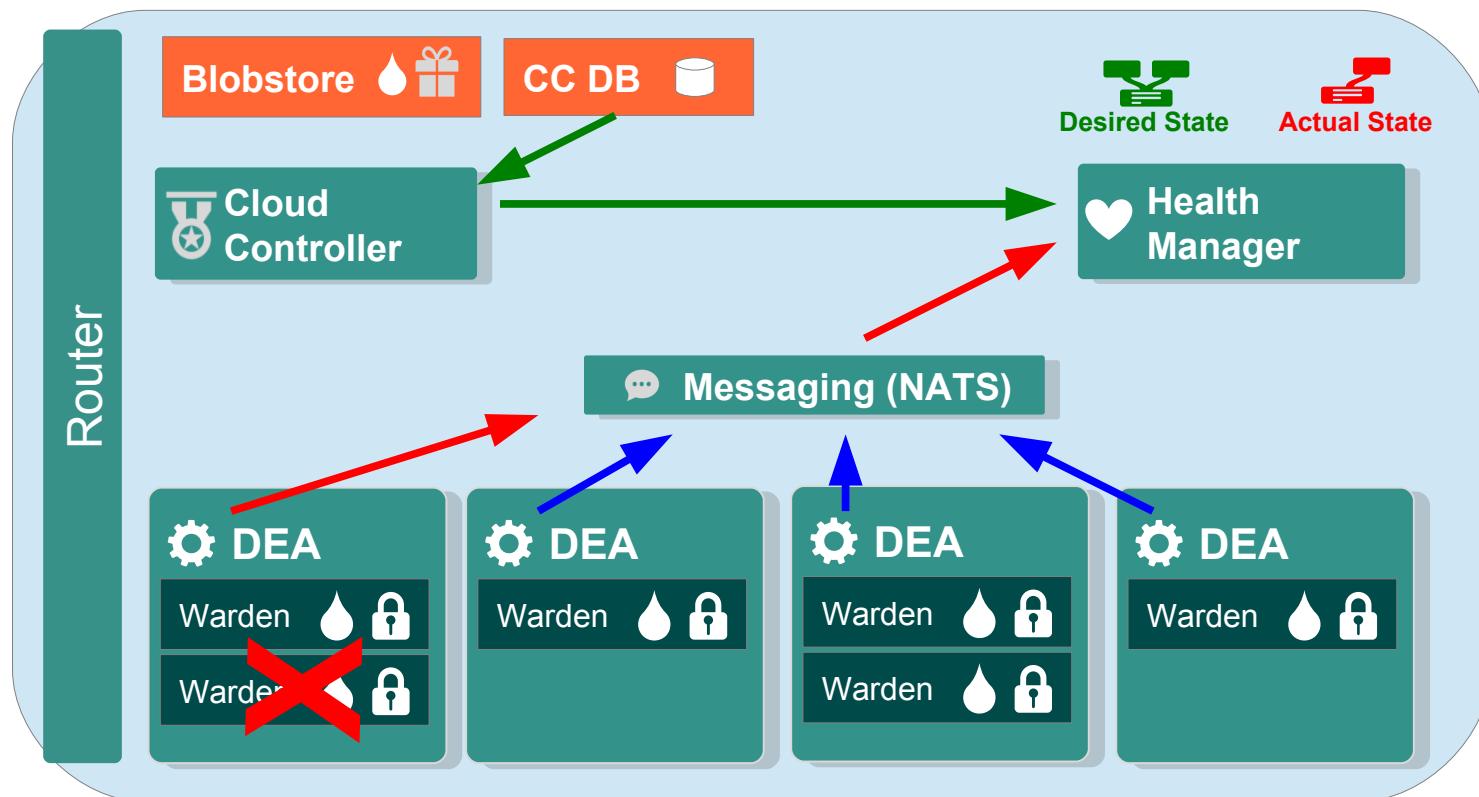
# Health Management – DEAs

1. Applications run within containers on DEAs
2. DEAs send heartbeat messages, messages sent to Health Manager
3. Health Manager obtains “desired state” from Cloud Controller
4. Does “desired state” = “actual state”? Yes



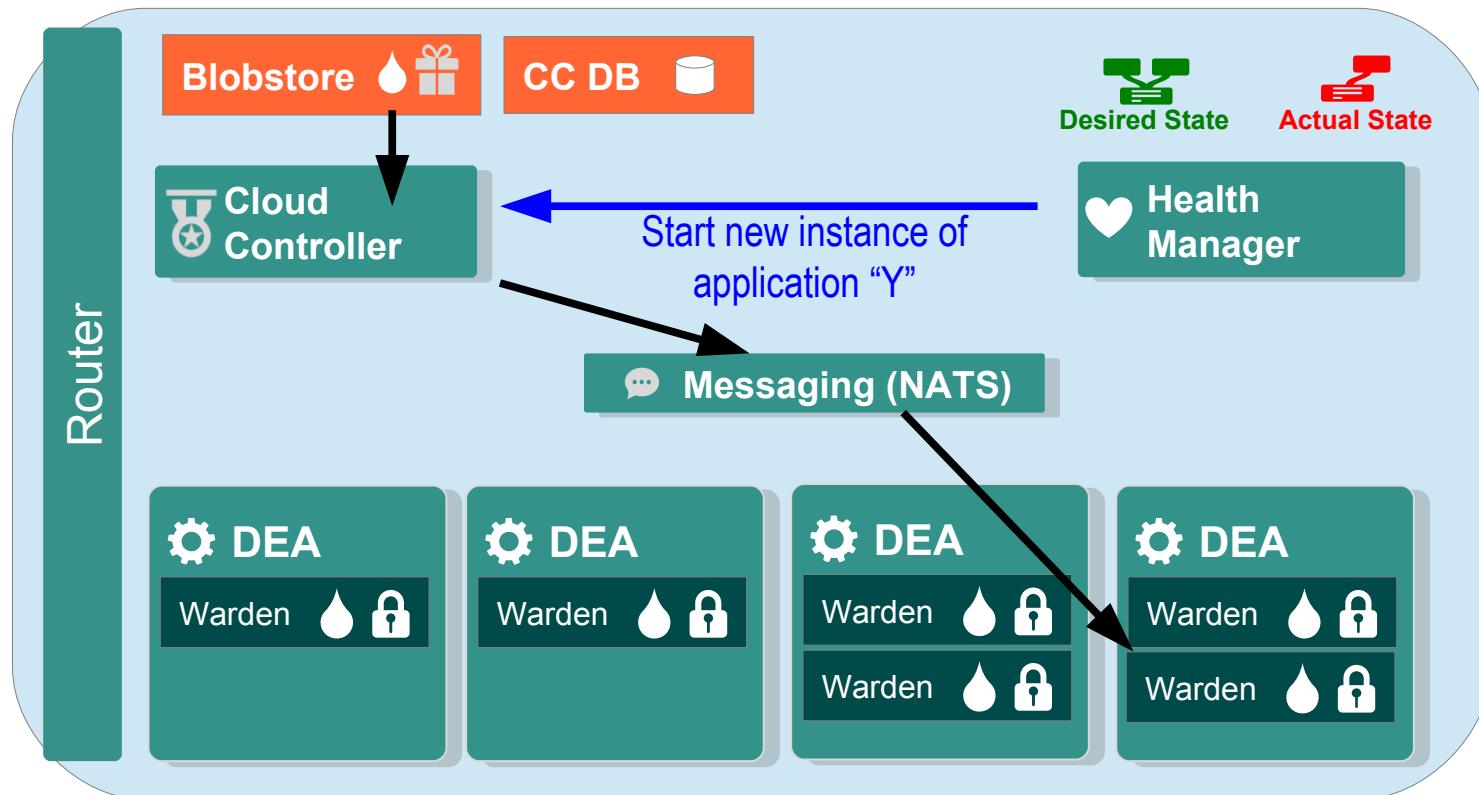
# Health Management – Detecting Failures

1. What if an application crashes **X**?
2. DEAs send heartbeat messages, messages sent to Health Manager
3. Health Manager obtains “desired state” from Cloud Controller
4. “Desired state” = “actual state”? **No**



# Health Management - Replacing an Application

1. Health Manager instructs Cloud Controller
2. Cloud Controller clones Droplet into container on DEA



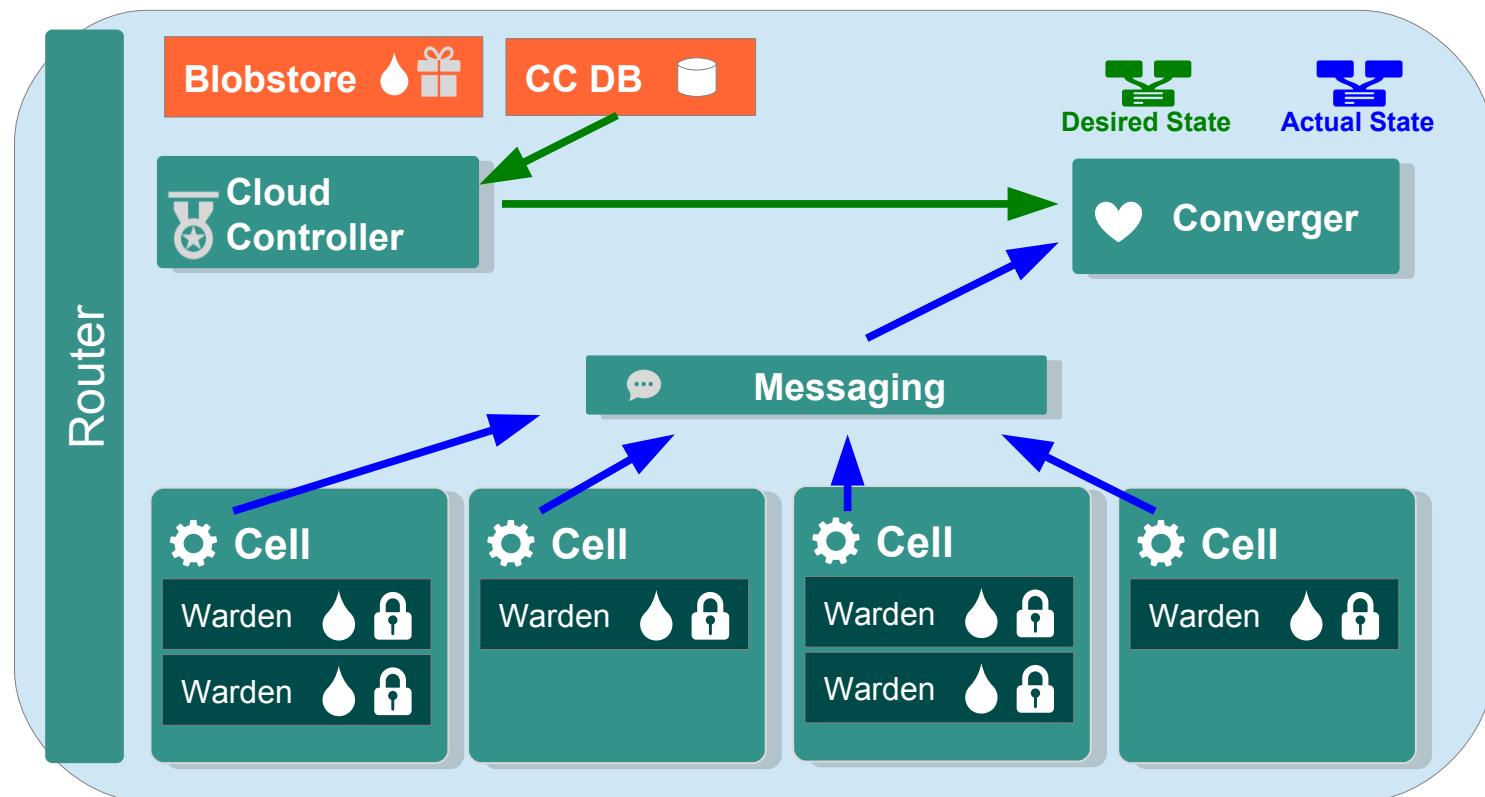


# Converger (DIEGO)

- Performs the same function in the DIEGO system
  - Any discrepancies cause a new instance request to be placed on the Bulletin Board
  - Cloud Controller is *no longer* involved in restart
- Instance started in the usual way
  - The auctioneer requests a Cell to use
  - The instance is started in a container on that Cell

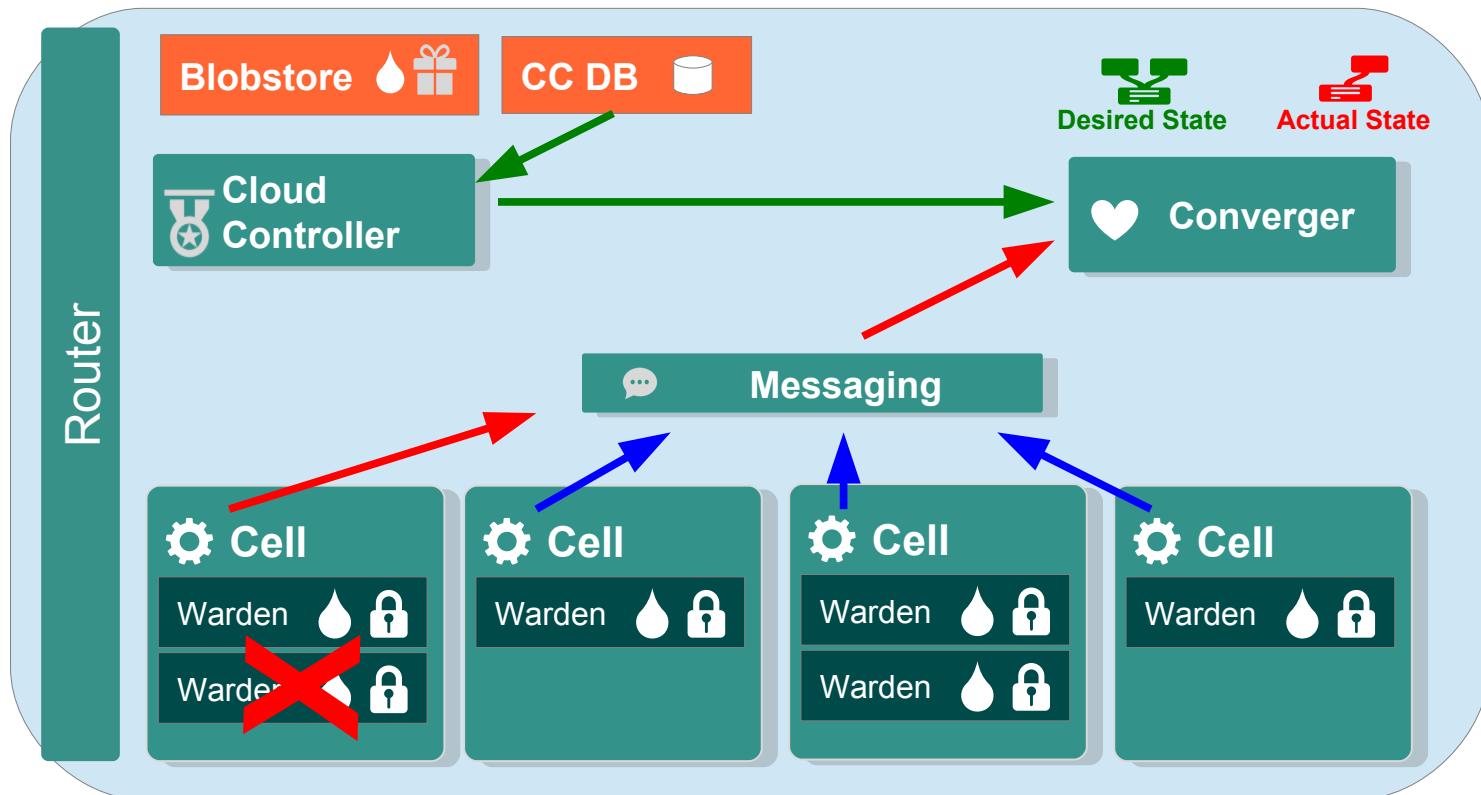
# Health Management – DIEGO

1. Applications run within containers on Cells
2. Cells send heartbeat messages, messages sent to Converger
3. Converger obtains “desired state” from Cloud Controller
4. Does “desired state” = “actual state”? Yes



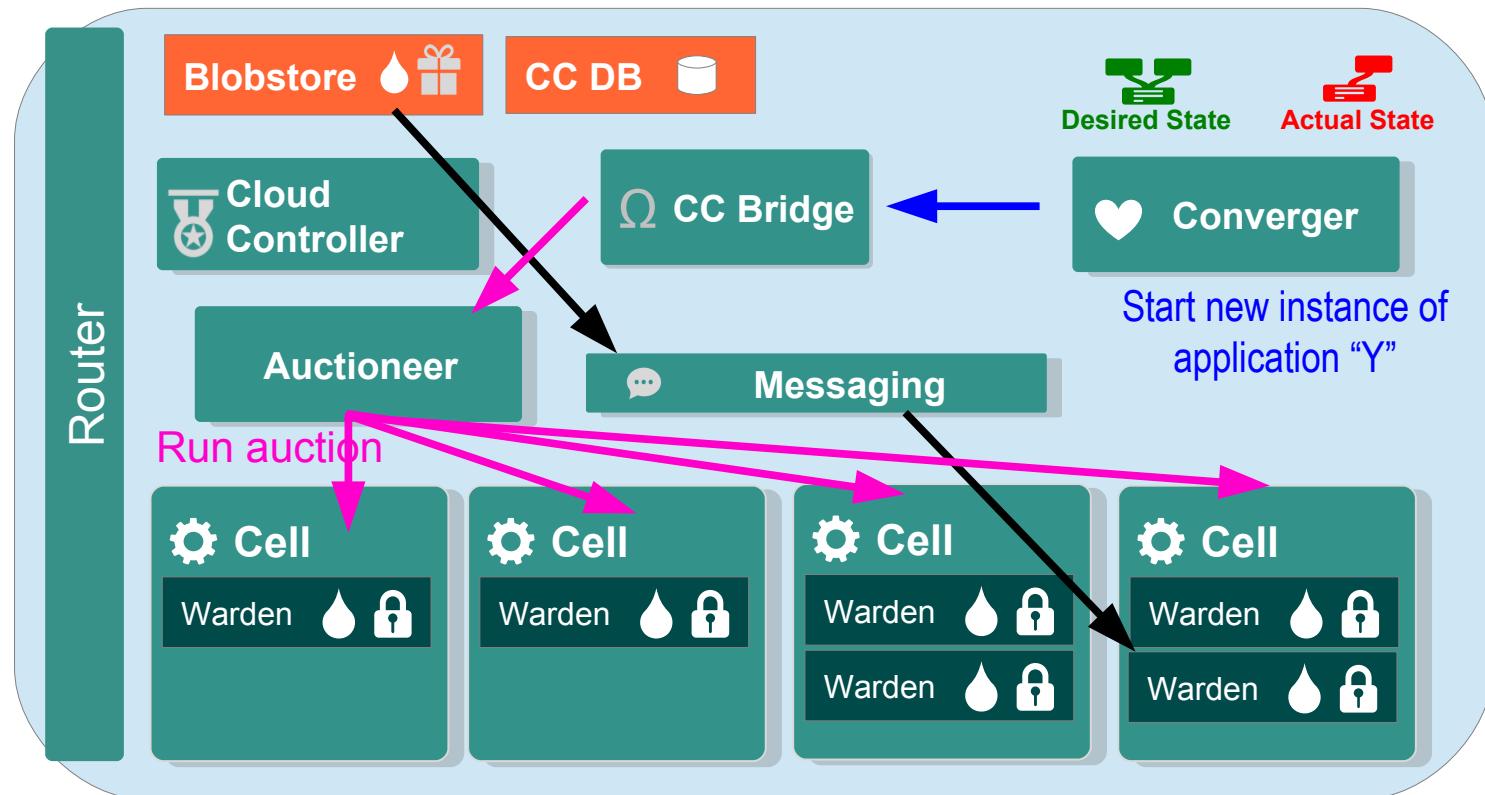
# Health Management – Detecting Failures

1. What if an application crashes **X**?
2. Cells send heartbeat messages, messages sent to Converger
3. Converger obtains “desired state” from Cloud Controller
4. “Desired state” = “actual state”? **No**



# Health Management - Replacing an Application

1. Converger puts *new instance* request on Bulletin Board
2. Auctioneer accepts request and conducts an auction
3. Auctioneer tells winning Cell to run Droplet in a new container



# Summary

- After completing this lesson, you should have learned about:
  - Cloud Foundry's internal architecture
  - How application instances are managed within various containers
  - How the Cloud Foundry enables application restart



# Continuous Delivery and Cloud Foundry

Combining SCM, CD, and CF

Pivotal

# Roadmap

- Defining Terms: SCM, CI, CD
- Continuous Integration with CloudBees and Pivotal Cloud Foundry

# Defining Terms

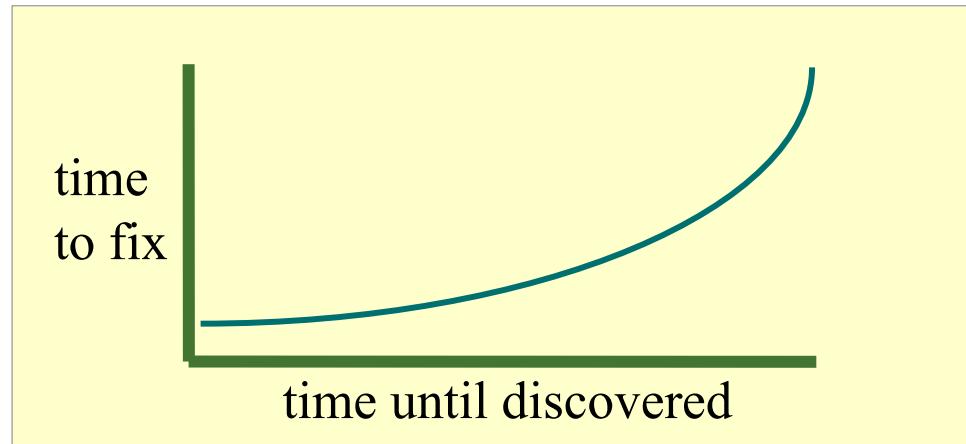
- **SCM** – Source Control Management
  - i.e. Version Control
  - Tool Examples: Git, SVN, CVS, Clear Case, Perforce, etc.
- **CI** – Continuous Integration
  - Running automated compilation, checks, and tests on a frequent basis
  - Tool Examples: Bamboo, Hudson/Jenkins, Cruise Control, Team City, Concourse
- **CD** – Continuous Delivery
  - CI + automated deployment

# Continuous Integration

- First seen as part of XP (eXtreme Programming)
- Goal: prevent integration / deployment problems by discovering them early
- Automatically run tests and packaging activities on periodic basis
  - Once per day
  - Each time a commit is made to SCM

# Benefits of Continuous Integration

- The cost to fix a bug grows exponentially in proportion to the time before it is discovered



- Continuous Integration (CI) focuses on reducing the time before the bug is discovered
  - Effective CI requires automated tests

# Continuous Delivery

- Continuous Integration + Automated Deployment
- Continuously run through the “Deployment Pipeline”

Compile

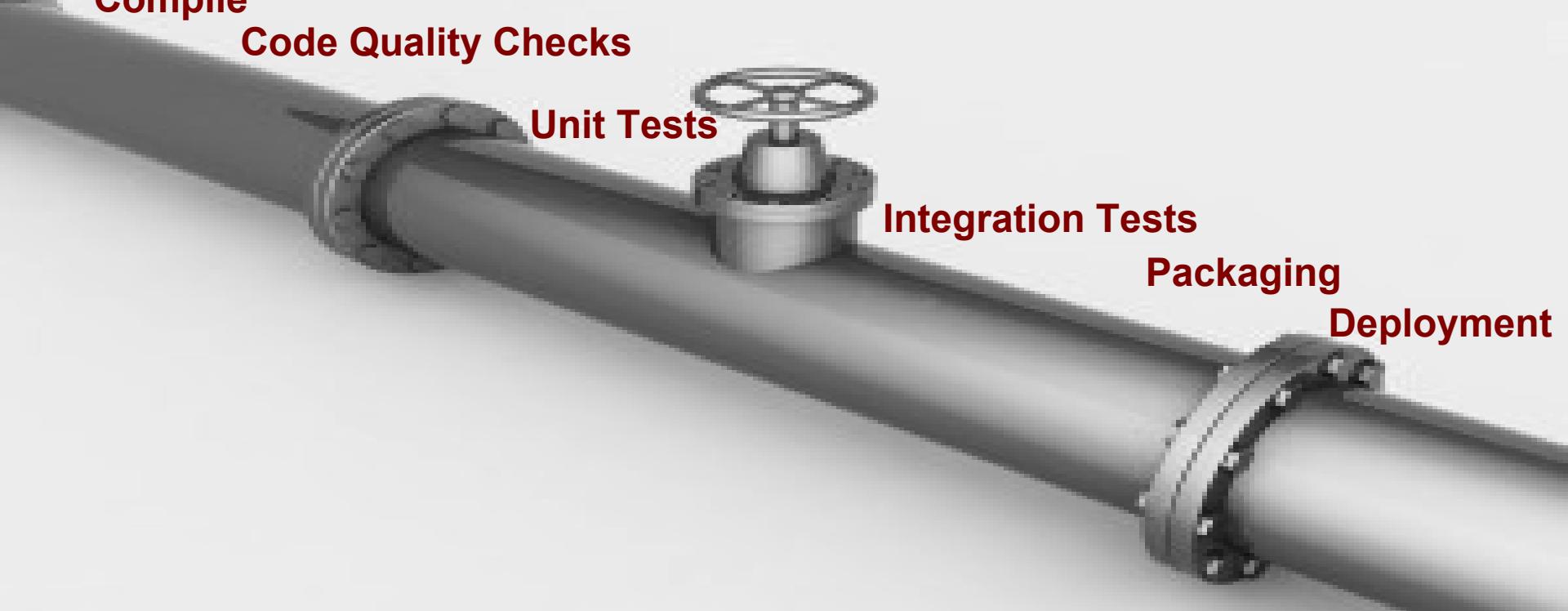
Code Quality Checks

Unit Tests

Integration Tests

Packaging

Deployment



Pivotal™

# Continuous Delivery

You're doing continuous delivery when:

- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments of any version of the software to any environment on demand

Martin Fowler, May 2013

# Roadmap

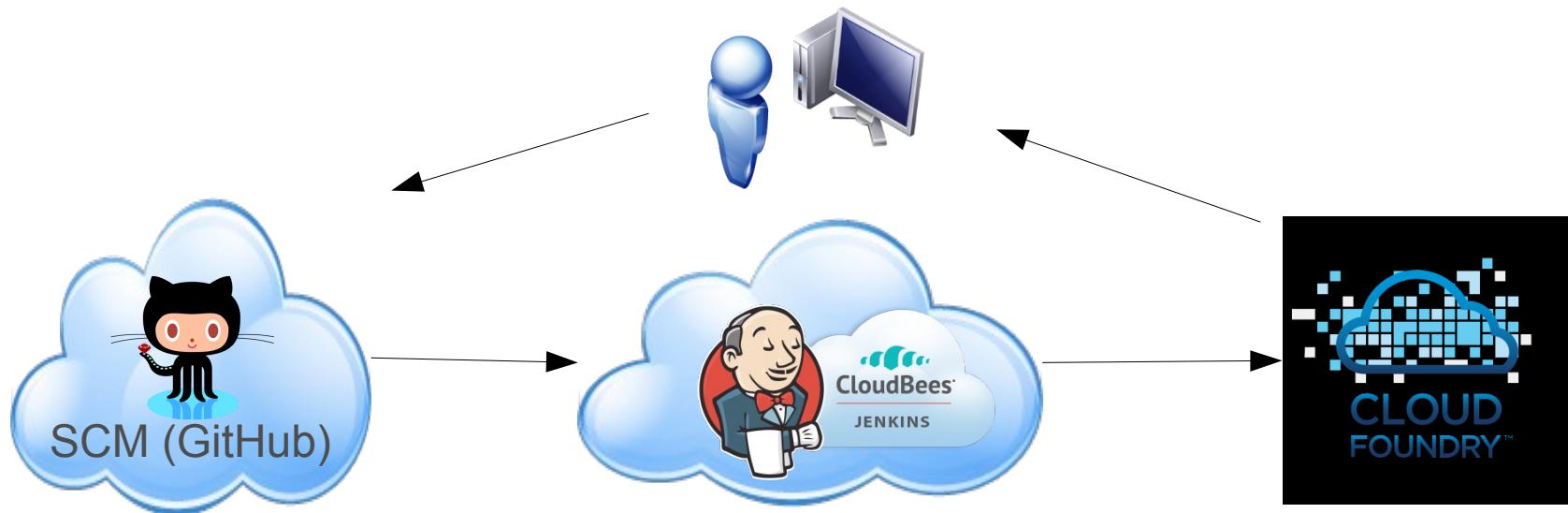
- Defining Terms: SCM, CI, CD
- **Continuous Integration with CloudBees and Pivotal Cloud Foundry**

# What Is CloudBees?

- Public Cloud-Based provider of Continuous Delivery services
  - Based on open-source Jenkins
- Features
  - Good integration with lots of SCM tools
  - Ability to run many build tools
    - Maven, Gradle, Ant, etc.
  - Great integration with PWS
- Good basis for our upcoming lab!
  - Or you can use your own Jenkins installation



# Example Continuous Delivery Pipeline Featuring SCM, CloudBees, and Pivotal CF



- 1) Developer commits code to SCM (GitHub)
- 2) CloudBees monitors repository for commit, launches build process
- 3) CloudBees pushes application to Cloud Foundry (PWS)

Note: CloudBees uses OAuth to connect to GitHub and PWS

# Establish Accounts

- You will need accounts for
  - GitHub
  - Pivotal Web Services
  - CloudBees
- Simply visit [cloudbees.com](http://cloudbees.com) to establish an account.
  - Use the “Free” Plan

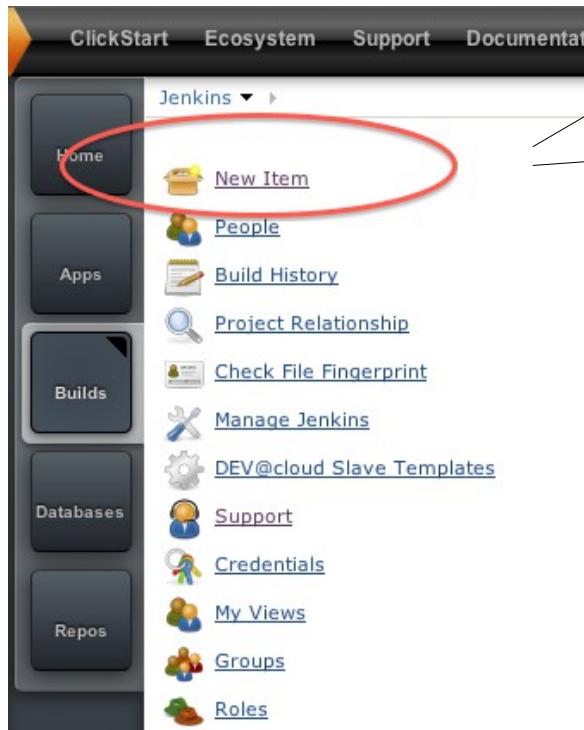
# CloudBees / Jenkins – Get Started with Builds

- Look for this item to setup CloudBees to do builds:



- This can take several minutes to provision a new account to do builds.

# CloudBees / Jenkins - New Build



Item name

**Build a free-style software project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Build a maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Build multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

**Monitor an external job**  
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

**Copy existing Item**  
Copy from

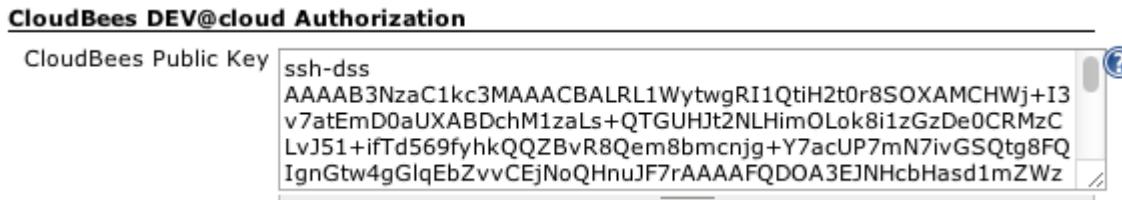
- Many other options available
  - See CloudBees documentation

# CloudBees / Jenkins - Configuration Options

- Many options available, these are just the highlights
  - See CloudBees documentation
- Set the name:

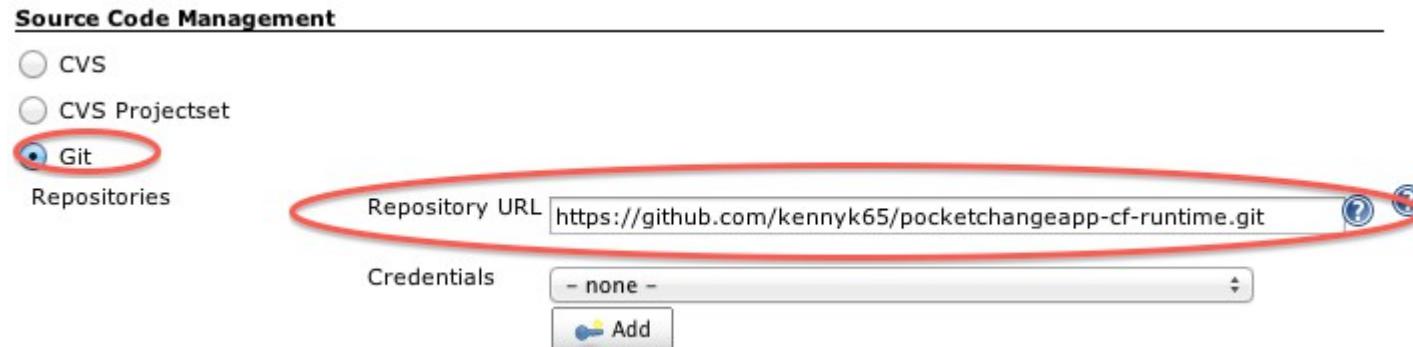
Maven project name

- Copy the public key
  - IF your GitHub repository is private



# CloudBees / Jenkins - SCM Integration

- Tell CloudBees about your SCM repository
  - We will use GitHub in our example:

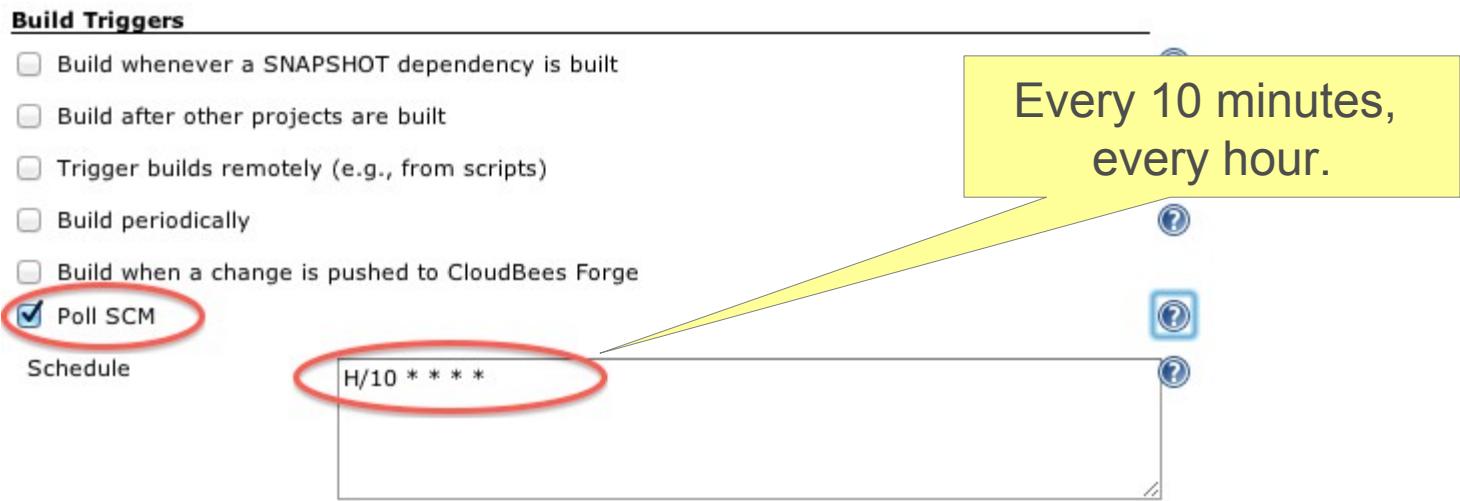


- Many options available here depending on SCM type.

# CloudBees / Jenkins

## Identify Build Triggers

- Under what circumstances should CloudBees launch the build process?



- Polling SCM – Can be expensive.
  - SCM push option available with more configuration.
- Schedule – Based on Cron syntax
  - “H” symbol introduces random ‘hash’ for better load handling

# CloudBees / Jenkins

## Add Post-Build actions to push to Cloud Foundry

- Multiple post actions available
  - i.e. deploy Maven artifacts
- CF will receive a request for authorization
- Similar options to CLI / Manifest

**Post-build Actions**

**Deploy applications**  
When build is  Stable  
 Unstable

**Host service**

**Cloud Foundry**  
Target API end-point <https://api.run.pivotal.io>

Credentials

Organization

Space

Applications

Application Name: pc-test

Deployment URL: pc-test.cfapps.io

Buildpack URL:

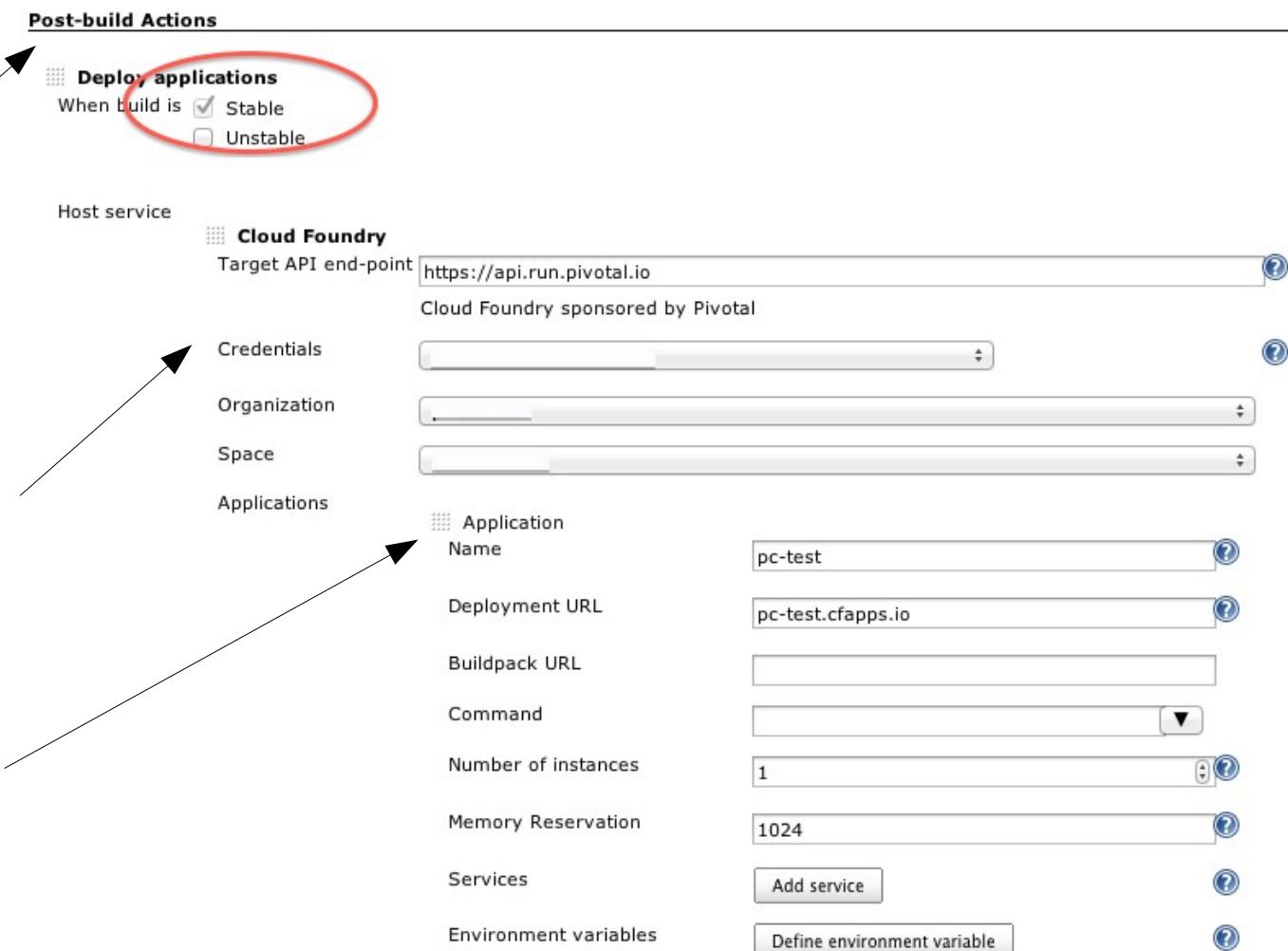
Command:

Number of instances: 1

Memory Reservation: 1024

Services: Add service

Environment variables: Define environment variable



# Pivotal CF – CloudBees as a Service

- Pivotal CF 1.3 includes CloudBees as a Service
  - Installed by administrator
- Advantages over external CloudBees SaaS:
  - Installed within enterprise (private)
  - Authentication integrated with Pivotal CF
  - Easy to scale

# Lab

Deploy to PWS via CloudBees



# Designing Applications for Cloud Foundry

## Writing Applications that Scale

Getting it right *By Design*

Pivotal

# Overview

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

# Roadmap

- 12-Factor Applications
- Design Guidelines

# Developer Topics / Application Architecture

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

# Roadmap

- **12-Factor Applications**
- Design Guidelines

# 12-Factor Application

- <http://12factor.net>
- Outlines architectural principles for modern apps
  - Focus on scaling, continuous delivery, portable, and cloud ready

# 12-Factor Application

## I. Codebase

One codebase tracked in SCM, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Configuration

Store config in the environment

## IV. Backing Services

Treat backing services as attached resources

## V. Build, Release, Run

Strictly separate build and run stages

## VI. Processes

Execute app as stateless processes

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity

Keep dev, staging, prod as similar as possible

## XI. Logs

Treat logs as event streams

## XII. Admin processes

Run admin / mgmt tasks as one-off processes

# 12-Factor Application

## I. Codebase

One codebase tracked in SCM, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Configuration

Store config in the environment

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

# 12-Factor Application

## I. Codebase

One codebase tracked in SCM, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Configuration

Store config in the environment

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

# 12-Factor Application

## I. Codebase

One codebase tracked in SCM, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Configuration

Store config in the environment

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

# 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)

# 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

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

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

# 12-Factor Application

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

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

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

# 12-Factor Application

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

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

# 12-Factor Application

## X. Dev/prod parity

Keep dev, staging, prod  
as similar as possible

## XI. Logs

Treat logs as event  
streams

## XII. Admin processes

Run admin / mgmt tasks  
as one-off processes

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

# 12-Factor Application

## X. Dev/prod parity

Keep dev, staging, prod  
as similar as possible

## XI. Logs

Treat logs as event  
streams

## XII. Admin processes

Run admin / mgmt tasks  
as one-off processes

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

# 12-Factor Application

## X. Dev/prod parity

Keep dev, staging, prod  
as similar as possible

## XI. Logs

Treat logs as event  
streams

## XII. Admin processes

Run admin / mgmt tasks  
as one-off processes

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

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



# 12-Factor Application

## I. Codebase

One codebase tracked in SCM, many deploys

## II. Dependencies

Explicitly declare and isolate dependencies

## III. Configuration

Store config in the environment

## IV. Backing Services

Treat backing services as attached resources

## V. Build, Release, Run

Strictly separate build and run stages

## VI. Processes

Execute app as stateless processes

## VII. Port binding

Export services via port binding

## VIII. Concurrency

Scale out via the process model

## IX. Disposability

Maximize robustness with fast startup and graceful shutdown

## X. Dev/prod parity

Keep dev, staging, prod as similar as possible

## XI. Logs

Treat logs as event streams

## XII. Admin processes

Run admin / mgmt tasks as one-off processes

# Roadmap

- 12-Factor Applications
- **Design Guidelines**

# Application Architecture

- Application architecture concerns:
  - Load Balancing / Session Management
  - Local file system
  - Port Limitations

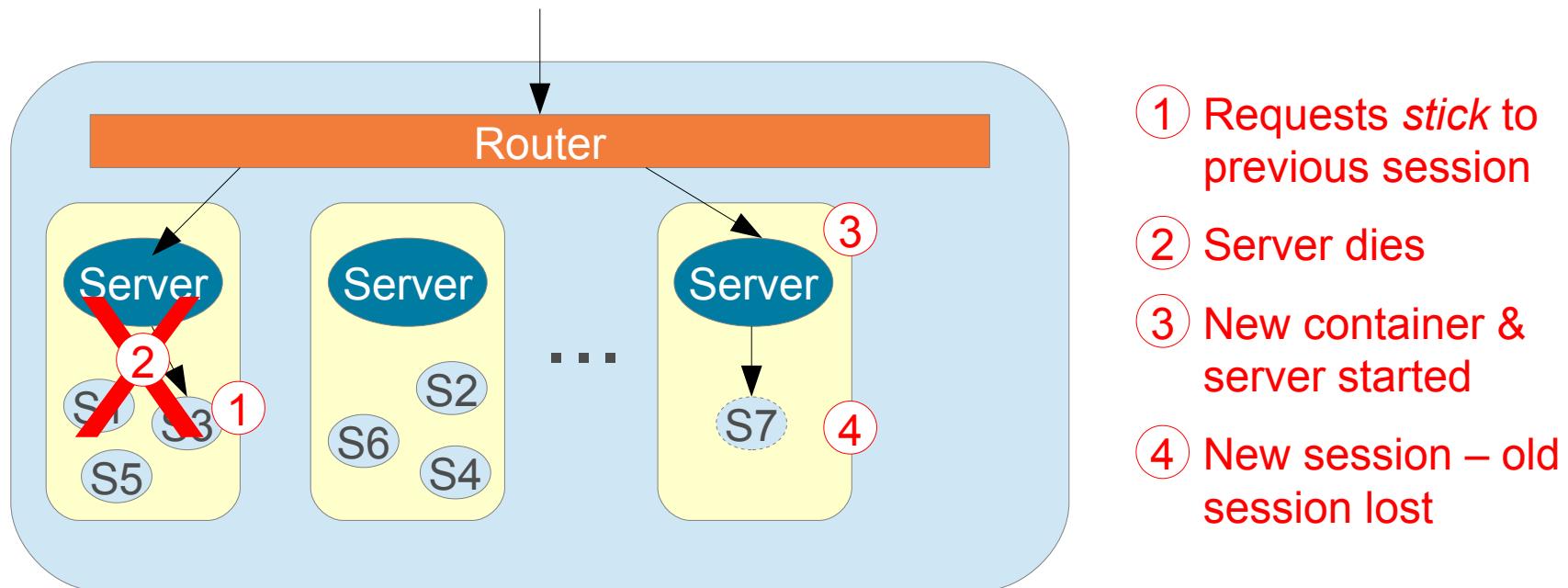
# Load Balancing Router

- CF Router provides automatic load balancing
  - When > 1 instance
  - Uses HAProxy <http://www.haproxy.org>
- Sticky Sessions - based on `JSESSIONID` parameter
  - Works automatically for Java Web apps
  - Other technologies need extra steps.



# Session Management

- Based on sticky sessions, managed by Router
- Session is NOT persisted between instances.
  - If an instance fails, those sessions are lost.



# Session Management

- Session use best avoided
  - In order to achieve massive scaling
  - Easy for RESTful servers
- If Sessions are essential
  - Add persistent session management
    - For example: Gemfire cache
  - Move session-data to a light-weight persistent store
    - Such as Redis key-value store

# Local File Access

- Apps should not attempt to access the local file system
  - Short lived, not shared
- Instead, use Service abstraction when flat files are needed
  - Amazon S3, Google Cloud Storage, Dropbox, or Box
    - Examples: file-uploading
  - File Storage as a Service is coming
- Or consider using a database
  - Redis: Persistent, in-memory data
  - Mongo DB: JSON document storage

# Logging

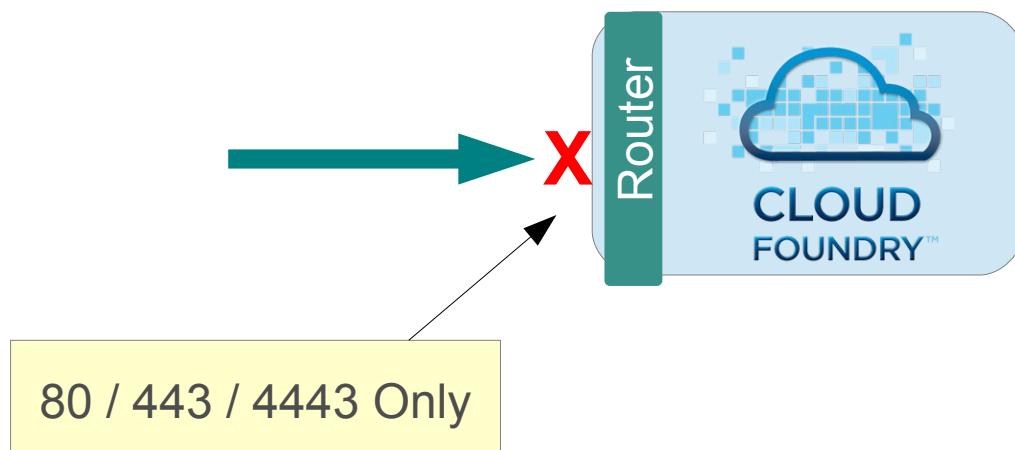
- Loggregator will automatically handle all output logged to sysout or syserr
- Don't use log-files
  - Local file system is generally not available
  - Loggregator will NOT handle log files made to the file system or other sources
  - Write to sysout instead
  - Or consider writing log records to a fast, NoSQL database
    - Can now be queried

# Resources

- All needed resources should be available via classpath
  - Example: use **classpath**: resource in Spring
- File resources not available
  - short lived / not shared
- Place configuration in **classpath**: resources
  - Spring MVC supports static web-resource in jars
    - Such as CSS, HTML, images, ...

# Port Limitations

- Port usage currently limited to HTTP and HTTPS
  - Only 80, 443 and 4443\* open to *incoming* traffic
    - Outgoing traffic controlled by Security Groups
  - Cloud Foundry *Router* only supports these protocols



\* 4443: for secure websockets

# Summary

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



# Spring and Cloud Applications

Making JVM cloud applications easier

Spring preliminaries for Cloud Applications

Pivotal

# Overview



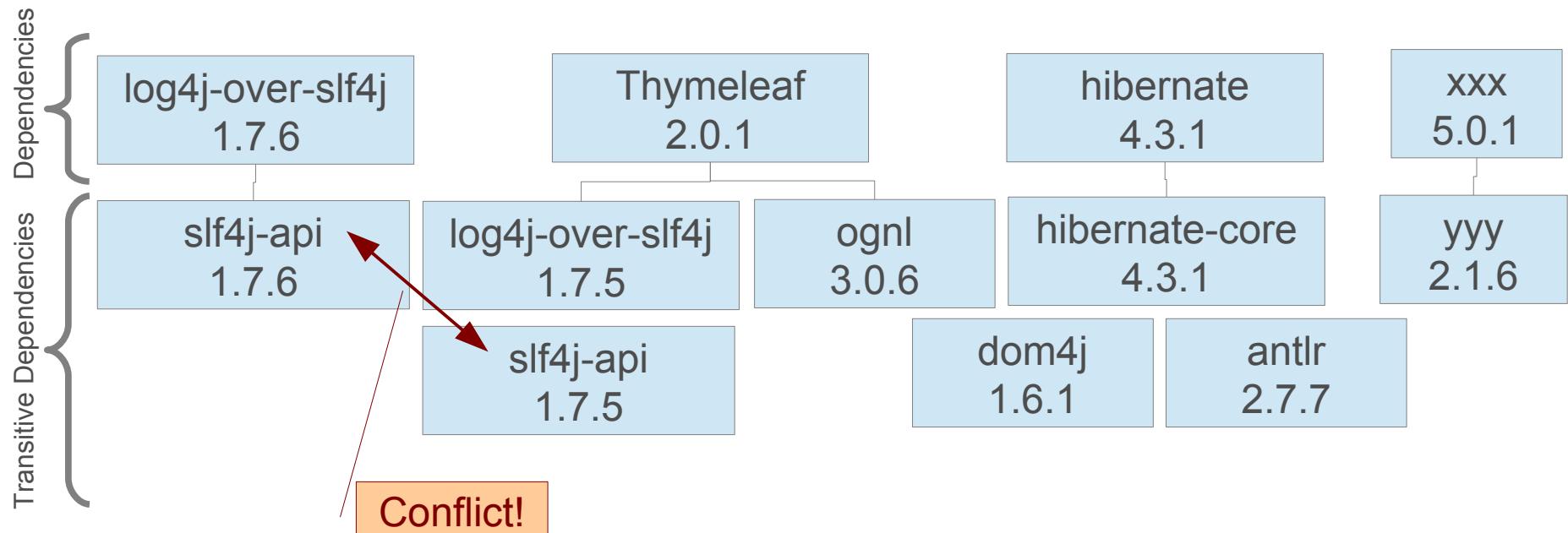
- After completing this lesson, you should be able to:
  - Use Spring IO and Spring Boot
  - Configure and run applications using Profiles
- **Note**
  - Spring IO, Spring Boot are **not essential** for a Cloud application
    - But they *simplify* code and dependency management
    - So cloud deployment is *much easier*
  - We will rely on them in later sections and labs

# Roadmap

- **Spring IO**
- Spring Boot
- Profiles
- Spring Cloud

# What is Spring IO?

- Working with open source libraries can be challenging
  - Transitive Dependencies* – OS libraries that depend on other libraries
  - Need Maven, Gradle, etc. to manage, but still difficult





# Spring IO

- Defines a set of maven library dependencies
  - For Spring and other commonly used JARs
  - Version-set known to work together

“Bill of Materials”

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.spring.platform</groupId>
      <artifactId>platform-bom</artifactId>
      <version>1.0.1.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Use Gradle if you prefer



# Using Spring IO

- Now define Maven dependencies in usual way
  - No need for <version>, Spring IO will decide

```
<dependencies>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-commons</artifactId>
    </dependency>

    <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
    </dependency>
    ...
</dependencies>
```

No version needed!



# What Dependencies are Available?

- *Many, many* of the JARs commonly used in Spring JVM applications
  - Spring, Groovy core and project JARs, ...
  - Apache Commons, Tiles, Solr, Velocity, Tomcat, ...
  - JPA, Hibernate, EclipseLink, MyBatis, NoSQL DBs ...
  - Logging, OXM, JSON, Metrics, ...
  - Web, Servlets, Jetty, Thymeleaf ...
- Presently, 480+ JAR versions are managed.

<http://docs.spring.io/platform/docs/current/reference/htmlsingle/#appendix-dependency-versions>

# Roadmap

- Spring IO
- **Spring Boot**
- Profiles
- Spring Cloud





# What is Spring Boot?

- A quick way to start building a Spring project
- An opinionated runtime for Spring Projects
- Supports different project types, like Web and Batch
- Can be used to create containerless apps
- It is not:
  - A code generator
  - An IDE plugin



# Opinionated Runtime?

- Spring Boot uses sensible defaults, mostly based on the classpath contents.
- For example:
  - Sets up a JPA Entity Manager Factory if a JPA implementation is on the classpath
  - Creates a default Spring MVC setup, if Spring MVC is on the classpath
- Everything can be overridden very easily
  - But most of the time not needed
  - Relies heavily on Spring 4 *@Conditional* annotation



# Spring Boot Demo

Pivotal™



# Spring Boot Dependency Management

- How it works (Maven Example):

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.3.0.RELEASE</version>
</parent>

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

Parent POM specifies dependency versions

Resolves JARS:

- spring-boot
- spring-core
- spring-context
- spring-aop
- aopalliance
- spring-beans
- logback-core
- plus ~10 more ...

- Eliminates need to document standard dependencies
  - See <http://projects.spring.io/spring-boot/> for latest version



# Spring Boot – Adding Dependencies

- To add capabilities, add additional “starter” dependencies:

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

Adds JARS:

- spring-web
- spring-webmvc
- jackson-databind
- hibernate-validator
- tomcat-embed
- *plus transitive dependencies*

For full list of Spring Boot starter dependencies see:

<https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters>



# Spring Boot JPA Example

- Dependencies

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

    <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
    </dependency>
</dependencies>
```

*Starter dependency set  
for RDBMS application:  
JDBC, JPA, Hibernate*

HSQldb Database



# Programming Spring Boot

- Run application using a `main()` method ①
  - Like we used to before containers!
- `@SpringBootApplication` ②
  - Enables automatic configuration: `@EnableAutoConfiguration`
    - Spring Boot scans classpath, setting up typical defaults
  - Marks class as a `@Configuration` class
  - Enabled `@ComponentScan` from current base package
- `SpringApplication` class ③
  - Initiates Spring Boot
  - Tells Spring Boot which class to start with
    - Usually *this* class ① ② ③ see next slide ....



# Example Spring Boot Application

2

```
@SpringBootApplication  
@ImportResource("classpath:config/db.xml")  
public class AccountMain {  
  
    @Autowired AccountService accountService;  
  
    public static void main(String[] args) {  
        SpringApplication.run(AccountMain.class, args);  
  
        // Do something ...  
        accountService.someMethod();  
    }  
}
```

@Configuration +  
@EnableAutoConfiguration +  
@ComponentScan

Database setup

Get Spring bean

Run Spring Boot

**Note:** Integrates with Java Config, annotated DI and/or XML configuration



# Web Application using Spring Boot

- Define Spring Controllers in usual way
  - Add **spring-boot-starter-web** dependency

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

- The **main()** program starts up embedded Tomcat

```
2 @SpringBootApplication
public class WebMain {
1  public static void main(String[] args) {
    SpringApplication.run (WebMain.class, args);
  }
}
```

- Spring Boot detects web-artifacts, starts Tomcat
- This method *only* returns when Tomcat shuts down

# Run as a WAR

- Two Steps:
  - Change Packaging to war: <packaging>war</packaging>
  - Extend Spring Boot's Servlet Initializer

```
@SpringBootApplication
public class WebMain extends SpringBootServletInitializer {

    // Can still have main() if you like
    public static void main(String[] args) { ... }

    @Override
    protected SpringApplicationBuilder
        configure(SpringApplicationBuilder application) {
        return application.sources(WebMain.class);
    }
}
```

No WEB.XML! (Unless you want)

# Overriding Defaults

- Spring Boot takes an *opinionated* approach to application decisions
  - Example Opinion: Web applications should use Tomcat
  - Example Opinion: JPA applications should use Hibernate
- What if you have different opinions?
  - Use Jetty, use EclipseLink
- No Problem!
  - Simply override the dependencies (see next)

# Overriding Defaults – Option 1

- Use Jetty instead of Tomcat
  - Just add its starter dependency

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

    <!-- Use Jetty as embedded servlet container not Tomcat -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jetty</artifactId>
    </dependency>

    ...
</dependencies>
```

# Overriding Defaults – Option 2a

- Use *EclipseLink* instead of *Hibernate*
  1. Change maven dependency
  2. Override the default *entityManagerFactory* bean
    - Normally created by Spring Boot, will use yours instead

```
<!-- Minimal persistence.xml, not needed with Hibernate, but  
EclipseLink adheres to the specification. -->

persistence.xml



```
<!-- Add EclipseLink dependency -->org.eclipse.persistenceorg.eclipse.persistence.jpa

Maven


```


```

# Overriding Defaults – Option 2b

```
@Configuration
public class JpaConfig {
    @Bean
    public EntityManagerFactoryBean emf (DataSource dataSource) {
        JpaVendorAdapter adapter = new EclipseLinkJpaVendorAdapter();
        // Set desired properties.

        Properties props = new Properties();
        // Set desired properties

        LocalContainerEntityManagerFactoryBean emfb =
            new LocalContainerEntityManagerFactoryBean();
        emfb.setDataSource(dataSource);
        emfb.setPersistenceUnitName("account");
        emfb.setJpaProperties(props);
        emfb.setJpaVendorAdapter(adapter);
        return emfb;
    }
}
```

Spring Java Config – remember  
to enable component-scanning

# Spring Boot Maven Properties

- Can override product versions using *properties* in your Maven POM or Gradle build file

```
<properties>
    <!-- Get Spring Boot to set Java version -->
    <java.version>1.7</java.version>

    <!-- Class containing main() -->
    <start-class>io.pivotal.cf.Application</start-class>

    <!-- Servlet version – downgrade to 2.x if you prefer -->
    <servlet-api.version>3.0.1</servlet-api.version>

    <!-- Tomcat version -->
    <tomcat.version>7.0.55</tomcat.version>
</properties>
```

<https://github.com/spring-projects/spring-boot/blob/master/spring-boot-dependencies/pom.xml>

# Spring Boot Application Properties

- Spring Boot automatically looks for **application.properties** in the classpath root
  - Or use **application.yml** if you prefer YAML
- Use predefined properties to control Spring Boot

```
spring.datasource.url=jdbc:mysql://localhost/test
```

```
spring.datasource.username=dbuser
```

application.properties

```
spring.datasource.password=dbpass
```

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

```
spring:
```

```
  datasource:
```

```
    url: jdbc:mysql://localhost/test
```

```
    username: dbuser
```

```
    password: dbpass
```

```
    driver-class-name: com.mysql.jdbc.Driver
```

application.yml

No tabs!

# Roadmap

- Spring IO
- Spring Boot
- **Profiles**
- Spring Cloud



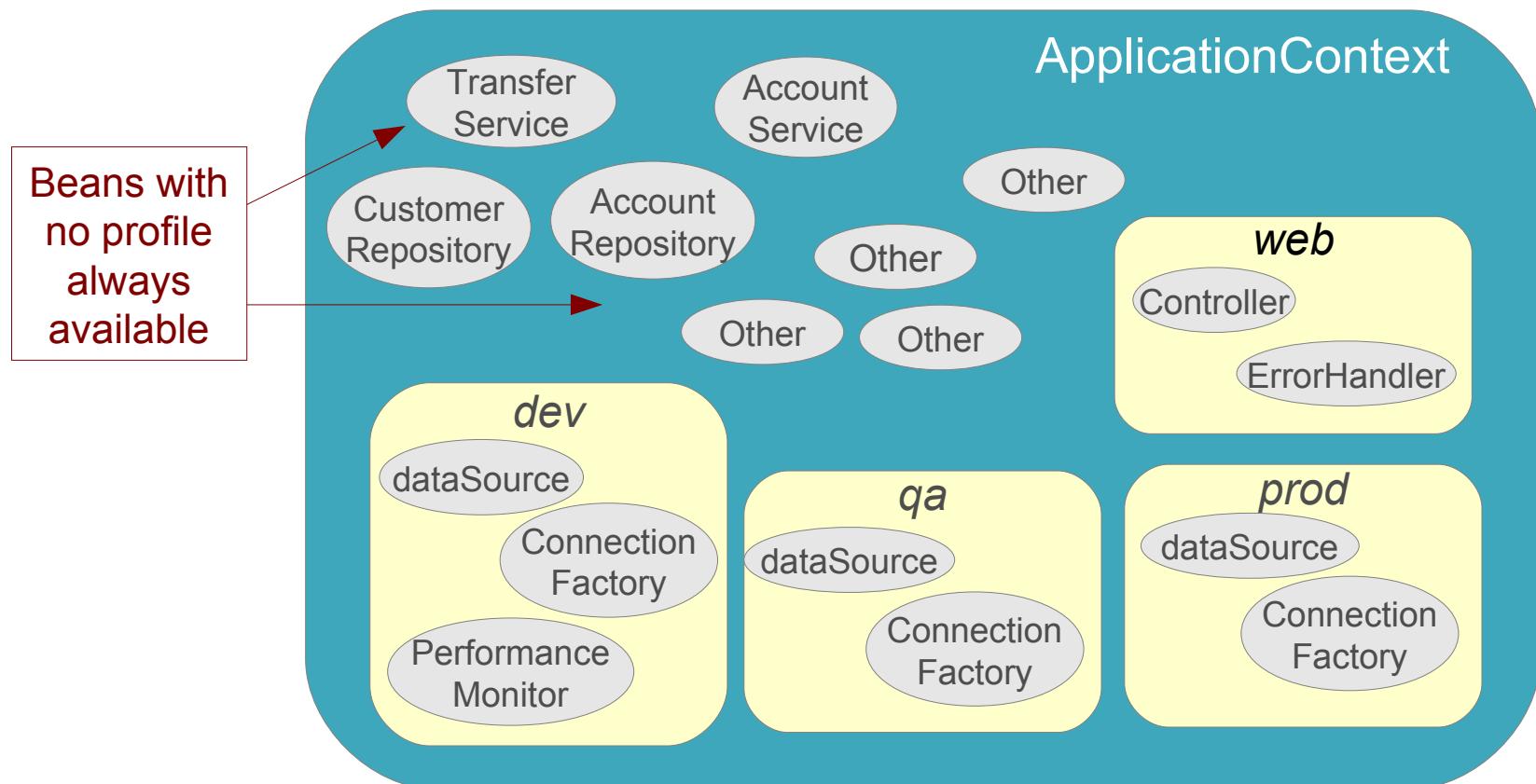


# Spring Profiles

- Part of Spring Framework since 3.0
  - Allow multiple different configurations
  - Select the ones you want by selecting one or more profiles
  - Integrated into Spring Testing framework also
- Beans can be grouped into Profiles
  - Profiles can represent purpose: “web”, “offline”,
  - Or environment: “dev”, “qa”, “uat”, “prod”, “cloud”
  - Or implementation: “jdbc”, “jpa”
  - Beans included / excluded based on profile membership



# Example Profiles





# Defining Profiles

- Add **@Profile** annotation to component or configuration
- Or qualify **<beans>** in XML

```
@Configuration  
@Profile("dev")  
public class DevConfig {  
  
    @Bean  
    public DataSource dataSource() {  
        ...  
    }  
  
    @Repository  
    @Profile("jdbc")  
    public class  
    JdbcAccountRepository  
    { ... }  
}
```

```
<beans xmlns=...>  
    <!-- Available to all profiles →  
    <bean id="transferService" ... />  
    ...  
    <beans profile="jdbc">  
        <bean id="dataSource" ... />  
    </beans>  
    <beans profile="jpa"> ... </beans>  
</beans>
```



# Quiz:

## Which of the Following is/are Selected?

-Dspring.profiles.active=jpa

?

```
@Configuration  
public class  
Config { ...}
```

?

```
@Configuration  
@Profile("jpa")  
public class  
JpaConfig  
{ ...}
```

?

```
@Configuration  
@Profile("jdbc")  
public class  
JdbcConfig  
{ ...}
```



# Activating Profiles For a Test

- **@ActiveProfiles** inside Spring-driven test class
  - Define one or more profiles
  - Beans associated with that profile are instantiated
  - Also beans not associated with *any* profile
- Example: Two profiles activated – *jdbc* and *dev*

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=AppConfig.class)
@ActiveProfiles( { "jdbc", "dev" } )
```

```
public class TransferServiceTests { ... }
```



# Activate Profiles

- Profiles may be activated at execution-time
  - System property

```
-Dspring.profiles.active=dev,jpa
```
- Profiles activated via Cloud Foundry environment variable
  - CLI or manifest

```
cf set-env <app> spring.profiles.active dev
```
- Java buildpack automatically activates “cloud” profile
  - You can activate additional profiles as needed

# Roadmap

- Spring IO
- Spring Boot
- Profiles
- **Spring Cloud**





# Spring Cloud

- Umbrella project for several sub-projects
  - Implement useful patterns required when building distributed, cloud-based applications
  - *Cloud Connectors*: access bound service information
  - *Cloud Starters*: for Spring Boot support
  - *Cloud Config*: centralized configuration management
  - *Cloud Netflix*: integration with Netflix OSS components
    - Eureka, Hystrix, Zuul, Archaius, ...
  - *Cloud Bus*: distributed messaging for services instances
  - *Spring Cloud for Cloud Foundry*
  - *Spring Cloud for Amazon Web Services*



# Spring Cloud Connectors Summary

```
// Obtain the Cloud abstraction:  
Cloud cloud = new CloudFactory().getCloud();  
  
// Obtain a bound service (no JSON parsing of VCAP_SERVICES):  
DataSource ds1, ds2;  
ds1 = cloud.getSingletonServiceConnector(DataSource.class, null);  
  
// Obtain a specific bound service by name:  
ds2 = cloud.getServiceConnector("mydb", DataSource.class, null);  
  
// Information about this instance:  
ApplicationInstanceInfo info = cloud.getApplicationInstanceInfo();  
logger.info("App id=" + info.getAppId()  
          + ", instance=" + info.getInstanceId());  
  
// Cloud properties:  
Properties p = cloud.getCloudProperties();
```

# Summary

- After completing this lesson, you should have learnt:
  - How to reduce configuration with Spring IO and Boot
  - How to use Spring Profiles
- We have all the Spring needed to deploy to the Cloud





# Java Applications and the Cloud

## Building Applications for Cloud Deployment

Handling Sessions, Configuration and Debugging

Pivotal

# Overview

- After completing this lesson, you should understand:
  - Using Spring to overcome Cloud application design Issues
  - How to handle HTTP sessions
  - On and off cloud configuration using Profiles
  - Debugging options
  - Testing on your local machine
  - Running the buildpack locally



# Roadmap

- **Cloud-Friendly Applications**
- Session Handling
- Configuration
- Debugging
- Local Development



This section applies to any language deployable by the *Cloud Foundry Java Buildpack* – Groovy & Grails, Scala & Play, Java & Spring ...

*Java Buildpack:* see <https://github.com/cloudfoundry/java-buildpack>

Pivotal™

# Cloud-Friendly Applications



- Many (most ?) applications will eventually run in the cloud
  - But not all applications are well-suited to it
  - Continuum of suitability from “*runs like it always has*” to “*thrives at cloud-scale*”
- How to create Java applications that *thrive* in the cloud?
  - Design guidelines (12 Factor)
  - Frameworks (Spring)
  - Cloud-specific code (Profiles)



# Leveraging Spring

- You don't have to use Spring, but it's easier
  - Spring Cloud
  - Spring Session
  - Spring Data
  - Spring IO and Spring Boot
- Design patterns covered here are applicable to all applications, Spring, Java or otherwise
  - Implementation details are Java & Spring-specific
  - Replicate functionality in other languages/environments

# Tuning Java for Cloud Foundry

- Memory divided between Heap, Metaspace (replaces Perm Gen from Java 8) and Native space
  - Thread memory uses Native space – for default run  
`java -XX:+PrintFlagsFinal -version | grep ThreadStackSize`
  - Use `-Xss` to change the default

Memory Flag	Heap	Metaspace	Native	Threads
Typical	75%	10% (64M min)	15%	~1M/thread
<code>-m 512M</code>	375M	64M	75M	~75 max
<code>-m 1G</code>	768M	102M	154M	~154 max
<code>-m 2G</code>	1536M	204M	308M	~308 max

<http://support.run.pivotal.io/entries/80755985-How-do-I-size-my-Java-or-JVM-based-applications->

# Cloud-Friendly Java Applications



- What are the considerations?
  - Managing State and Sessions
  - Application Configuration
  - Accessing Files and Logs
  - Application Debugging
  - Monitoring and Management
  - Local Development: test before pushing
- We've looked at these in the abstract
  - How do we implement them in JVM applications?

# Roadmap

- Cloud-Friendly Applications
- **Session Handling**
- Configuration
- Debugging
- Local Development





# Managing State is Hard

- State, especially shared state is the enemy of scalability
  - **12 Factors #6:** *Prefer stateless processes*
  - Any coordination of state impacts performance
  - Coordination of state across peer instances *really* impacts performance
  - Put shared state in data-stores – have concurrency built-in
- In-memory state won't survive instance failure
  - HTTP Sessions are a common example of this problem
  - But many JEE apps use/need sessions



# Option 1: Manual Session Management

## Replace HTTP Sessions by Redis

- Old code

```
session.setAttribute("shoppingCart", shoppingCart);
...
Cart shoppingCart = (Cart) Session.getAttribute("shoppingCart");
```

- New code: Spring's Redis Template
  - Supports jedis, JRedis, SRP or Lettuce

```
RedisTemplate template = ...; // Or @Autowire as a Spring Bean
template.convertAndSet(userName + ":shoppingCart", shoppingCart);
...
Cart shoppingCart =
    template.getAndConvert(userName + ":shoppingCart", Cart.class);
```

**BUT:** Requires code change to existing apps



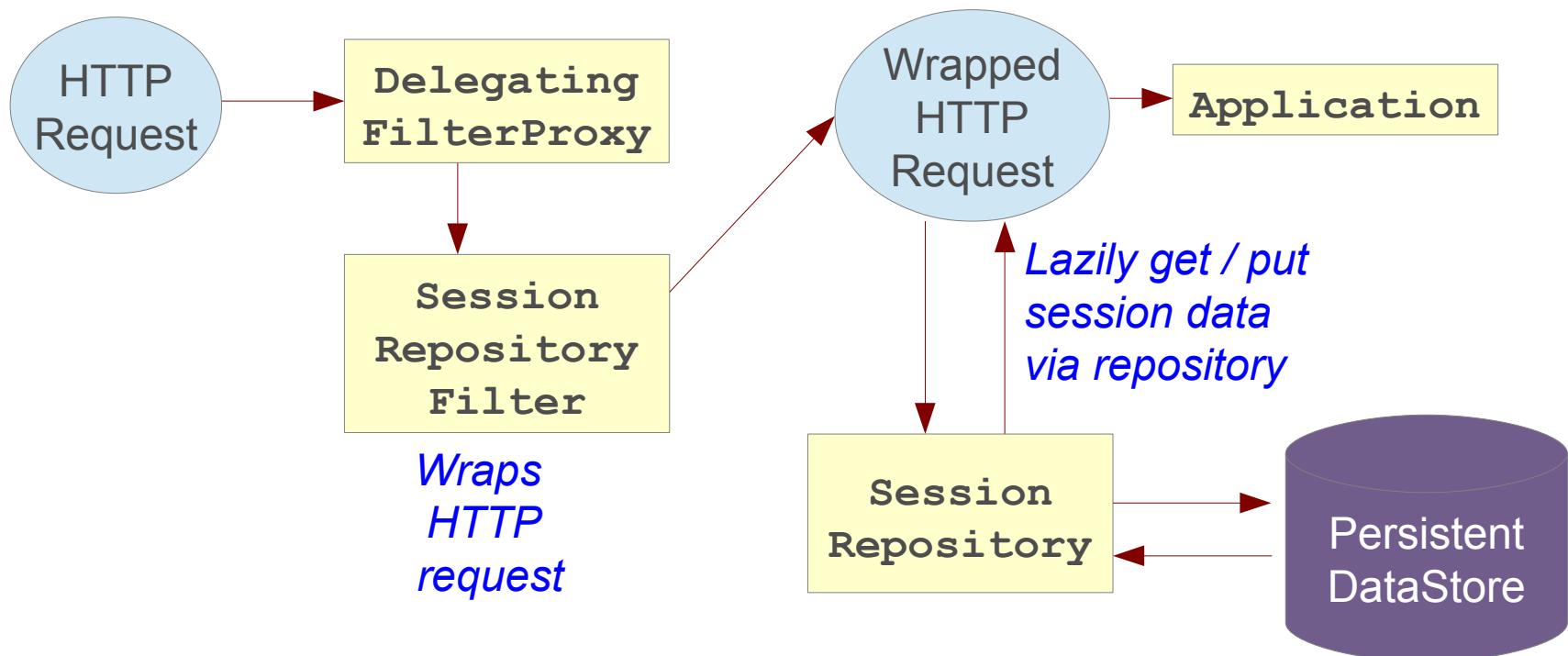
# Option 2: Spring's Session Abstraction

- What?
  - HTTP request is wrapped
    - All methods using session data overridden to use the session from `SessionRepository` instead
  - Any session changes automatically persisted
- Why?
  - If process fails, session data safe in Session Repository
  - Servlet code unchanged
- How?
  - Define a Redis connection (only implementation so far)
  - Use `@EnableRedisHttpSession`



# How It Works

- Typical scenario – current implementation uses Redis





# Define a Session Filter – XML 1

- Using **web.xml**

```
<filter>
    <filter-name>sessionRepositoryFilter</filter-name>
    <filter-class>...DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>sessionRepositoryFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

The “classic” XML Configuration

- All requests to the filter passed in turn to a Spring bean called **sessionRepositoryFilter**

# Spring Configuration – XML 2

- Java Config class for `@EnableRedisHttpSession`
  - Easier than manual configuration in XML

```
@EnableRedisHttpSession // <1>
public class RedisHttpSessionConfig {}
```

```
<beans>
    <context:annotation-config/>

    <bean class="org.springframework.data.redis.connection.
                           jedis.JedisConnectionFactory"/>
    <bean class="sample.RedisHttpSessionConfig"/>

</beans>
```

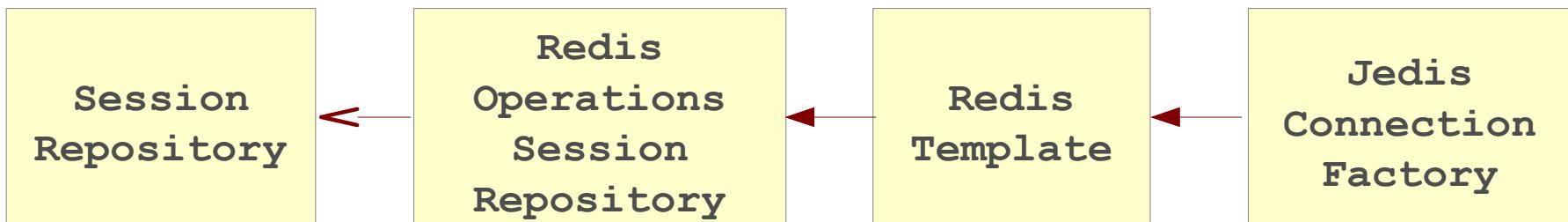
# Configuration – Java

```
@Configuration  
@EnableRedisHttpSession  
public class Config {  
    @Bean  
    public JedisConnectionFactory connectionFactory() {  
        return new JedisConnectionFactory();  
    }  
}  
  
import org.springframework.session.web.  
    context.AbstractHttpSessionApplicationInitializer;  
  
public class Initializer  
    extends AbstractHttpSessionApplicationInitializer {  
  
    public Initializer() {  
        super(Config.class); // Can list multiple classes  
    }  
}
```

Extends **WebApplicationInitializer** – runs  
a Spring Web app using Servlet 3 configuration

# How Does It Work?

- `AbstractHttpSessionApplicationInitializer` sets up a `DelegatingFilterProxy`
  - Only do this when *not* using `web.xml`
- `@EnableRedisHttpSession` creates
  - The `sessionRepositoryFilter`
  - A `RedisOperationsSessionRepository`
    - Implements `SessionRepository` using Redis



# Configuring Redis



- A Redis store needs to be available as well
  - If using Cloud Foundry, this would be a Redis Service bound in the usual way
    - Using Spring Cloud Connectors for example
  - In the online examples an embedded in-memory Redis store is used for demo/testing
    - <https://github.com/spring-projects/spring-session>
- Or you will need to configure Redis for your application
  - Spring Data Redis is helpful here:
  - <http://docs.spring.io/spring-data/data-redis/docs/current/reference>



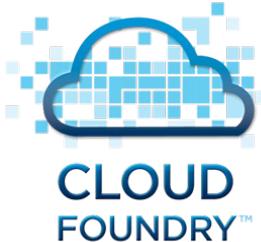
# Servlet Code

- Looks like normal session code!

**NO** code change  
to existing apps

```
session.setAttribute("shoppingCart", shoppingCart);  
...  
Cart shoppingCart = (Cart) Session.getAttribute("shoppingCart");
```

- SessionRepository** currently uses Redis
  - But other implementations may follow
  - Or implement your own
    - 4 methods: createSession, getSession, save, delete



# Option 3: Buildpack Does It For You

- Java Buildpack can setup session replication for you
  - Define a suitable service instance: Redis or Gemfire
  - Bind service instance to your Java application
  - Push your application
  - That's it!
- Internally builds on *Tomcat*
  - Session replication for Jetty *not supported*

<https://github.com/cloudfoundry/java-buildpack/blob/master/docs/container-tomcat.md>

# Option 3: Session Replication

Auto-Configured by Java Buildpack



- How it works
  - HTTP Session data automatically persisted to service
  - If application instance fails, new instance fetches session data from replication service automatically
  - Same as option 2, but *no* configuration needed
- Implementations
  - **Redis:** Normal Redis service
    - Service *must* be called *session-replication*
  - **Gemfire Session State Caching**
    - Custom Gemfire service specifically for session replication
    - [http://docs.pivotal.io/ssc-gemfire/using\\_the\\_data\\_service.html](http://docs.pivotal.io/ssc-gemfire/using_the_data_service.html)

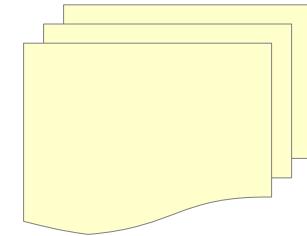


# Roadmap

- Cloud-Friendly Applications
- Session Handling
- **Configuration**
- Debugging
- Local Development



# Shared Files are *also* Shared State



- Don't rely on files
  - Many applications persist data to the filesystem
  - Even if you *can* use/share a filesystem, such as S3, **don't**
    - Shared state (files) *impact performance*
- Avoid files with Cloud Foundry
  - App instances run in *different* Execution Agents
    - **Can't** share their files
  - File-system in Cloud Foundry is temporary
    - Lost when application (container) terminates
    - Won't survive restart (app instance fail-over)
  - Consider NoSQL store like Riak CS or Mongo/DB instead



# Configuration

- Java programs often read from myriad configuration files
  - Typically Properties or XML files
  - Problematic to deploy with application
    - Packaged within JAR/WAR? Violates **12 Factors #3**
    - Deployed outside JAR/WAR? Ephemeral local file system.
    - Contents may be *environment* specific
- Classic example: Database Connection information
  - *On cloud*: Need to access database service
  - *Off cloud*: Ops need to define database URLs and other connection properties



# Use Classpath Resources

- File resources not available
  - short lived / not shared
- *Platform-independent* resources can be supplied via classpath
  - Package in your jar, war, ...
  - Consider Spring's **classpath:** resources
  - Spring MVC supports static web-resource in jars
    - Such as CSS, HTML, Script, images, ...
  - Use **@Profile** to setup environment-specific settings



# Configuration from Environment

- Dynamic (platform dependent) configuration should come from “*the environment*”
  - Java developers use **System** class
- *Recall:* to set in Cloud Foundry
  - **cf set-env app-name TEST\_PROP test-value**
- To access
  - **System.getenv("TEST\_PROP")**
  - May be unexpected for some Java developers



# Configuration and Spring Boot

- Spring Boot Relaxed Binding
  - No need for an exact match between desired properties and names
  - Intuitive mapping between system properties and environment variables
    - test.property equivalent to TEST\_PROPERTY

- Access via SpEL

Or without Spring Boot:

```
@Value("${TEST_PROPERTY}")
```

```
@Configuration  
class AppConfig {  
  
    @Value("${test.property}")  
    String testProperty;  
  
    @Bean SomeObject myBean() { ... }  
}
```



# Review: Accessing Configuration

- *Recall:* To view environment:
  - `cf env [app-name]`
    - User defined and system defined variables
- *Recall:* not all environment variables can be changed
  - Those set internally by Cloud Foundry cannot
    - Such as **VCAP\_SERVICES** (set by service binding)
    - See URL below for list of variables set by runtime

<http://docs.pivotal.io/pivotalcf/devguide/deploy-apps/environment-variable.html>



# JAVA\_OPTS

- Many Java configuration changes require buildpack modification
  - Java version
  - Heap, Perm Gen/Metaspaces sizes
- However, **JAVA\_OPTS** may be specified by environment variable
  - Combined with values set by Java buildpack
  - Examples:
    - `-agentlib`, `-enableassertions`, `-Dproperty=value`
  - *Note: memory settings are excluded*

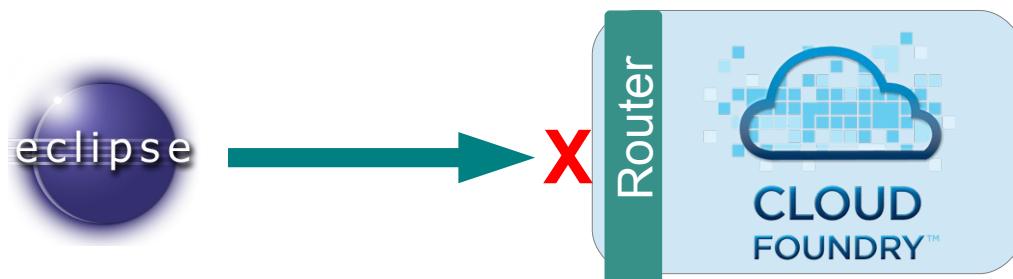
# Roadmap

- Cloud-Friendly Applications
- State Considered Evil
- Configuration
- **Debugging**
- Local Development



# Java Debugging via Eclipse

- Not possible to connect Eclipse debugger to Java application running in Cloud Foundry
  - Router only allows incoming HTTP on ports 80 / 443



- However, Java application running on Cloud Foundry can connect to Eclipse debugger
  - Fully explained:
  - <http://mikusa.blogspot.com/2014/08/debugging-java-applications-on.html>

# Java Debugging via Eclipse

- General Process:
  - Debug your Eclipse project using a Remote Java Application, connection type as standard socket listener
  - Push app with environment variable:
    - `JAVA_OPTS: -agentlib:jdwp=transport=dt_socket,address=<IP>:<PORT>`
- The hard part: making your Eclipse available over IP address, dealing with port / firewall issues.
  - Use port forwarding
  - Use SSH Tunnel



# Java Buildpack Troubleshooting

- Automatically logs to a file
  - To access:

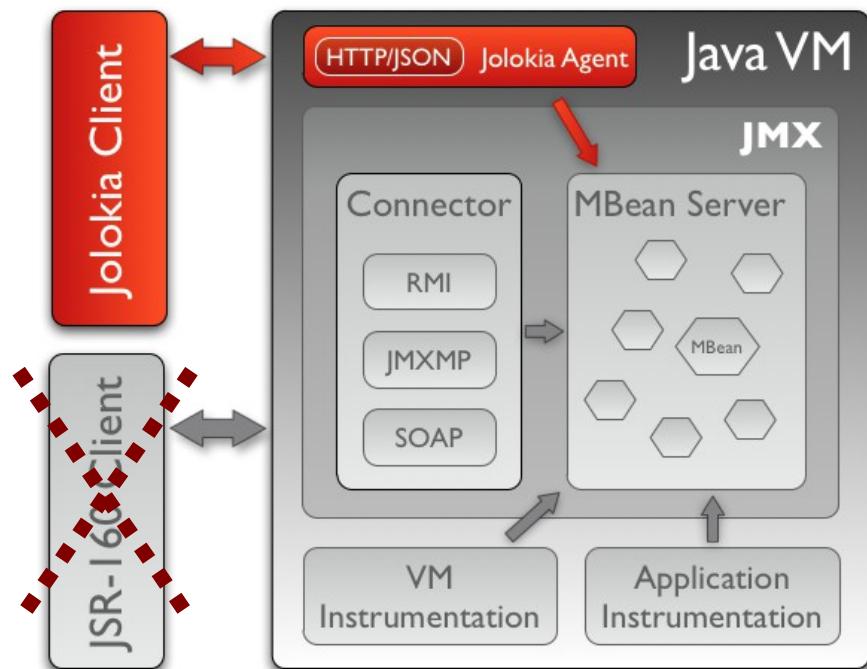
```
> cf files <APP> app/.java-buildpack.log
```

- If app fails to deploy, log file no longer exists
  - Configure for console logging instead
  - Increase buildpack logging level: **JBP\_LOG\_LEVEL**

```
> cf set-env <APP> JBP_LOG_LEVEL DEBUG
```

# Jolokia: JMX over HTTP

- Runs as a JMX client
- Responds to requests received over HTTP
  - Useful since CF *only* supports HTTP & HTTPS
- Configure as a Java servlet
- Or run as JVM agent (if no container)



# Jolokia: JMX over HTTP

- Deploy the `org.jolokia.http.AgentServlet` and map it
  - Maven dependency: `jolokia-core`
  - Managed by `spring.io`
- If using Spring Boot just specify  
`@EnableConfigurationProperties(value=JolokiaProperties.class)`
  - Still need `jolokia-core` dependency
- To access
  - `http://<your-app-url>/jolokia/read/<OBJECT-NAME>`



# Pivotal Spring Insight

- Automatic instrumentation & Java application monitoring
  - Uses same *Spring Insight* as Pivotal's *tc Server*
    - Included when you download *Spring Tool Suite (STS)*
  - Now a CF service
    - GA expected mid-2015
- Procedure
  - Bind *Insight* service to your application
  - Push app using customized Java Buildpack
    - Uses *tc Server* and detects *Insight* service
  - See <https://network.pivotal.io>

# Roadmap

- Cloud-Friendly Applications
- State Considered Evil
- Configuration
- Debugging
- **Local Development**





# Local Deployment, Cloud Production

- No matter how fast your cloud is, it's not as fast as running locally
  - Developers should do lots of work locally before pushing applications.
  - Not authoritative, but a good quick-turnaround option
- Therefore, applications must run both locally and in the cloud without modification
  - Use Spring Profiles (next slide)



# Local Deployment, Cloud Production

- Use Spring profiles to encapsulate conditional application configuration
  - Specifically the “cloud” profile
  - Auto-set in Cloud Foundry by Java Buildpack
- Can set additional profiles in CF via environment
  - `cf set-env SPRING_PROFILES_ACTIVE cloud,dev,jpa`

```
@Profile("cloud")
@Configuration
public static class DbConfig {
    @Bean
    DataSource dataSource() {...}
}
```

# Local Deployment, Cloud Production

- Run the buildpack *locally*
  - It's just a Ruby application
- Three scripts do all the work
  - `bin/detect <APP-DIR>`
  - `bin/compile <APP-DIR> <CACHE-DIR>`
  - `bin/release <APP-DIR>`
- Procedure
  - Install Ruby
  - Download buildpack
  - Look for bin directory



# Running and Debugging

- Use aliases (or a script)

```
$ alias detect='$BUILDPACK_ROOT/bin/detect .'  
$ alias compile='$BUILDPACK_ROOT/bin/compile . $TMPDIR'  
$ alias release='$BUILDPACK_ROOT/bin/release . |  
  ruby -e "require \"yaml\";  
  puts YAML.load(STDIN.read)[\"default_process_types\"][\"web\"]"'
```

- Set **JBP\_LOG\_LEVEL** variable *locally* for debug logging

```
export JBP_LOG_LEVEL=DEBUG
```

- For more details see:

<https://github.com/cloudfoundry/java-buildpack/blob/master/docs/debugging-the-buildpack.md>

# Summary



- Statelessness leads to scalability
- No filesystem access
- Configuration via the environment
- Console-based logging
- Monitoring via services and frameworks
- Smart Debugging
- Local development, cloud production



# Lab

Using sessions and environment variables



# Using Spring Cloud Connectors

## Building Applications for Cloud Deployment

Binding to Services in Multiple Environments

Pivotal

# Overview

- After completing this lesson, you should understand:
  - What Spring Cloud provides
  - How Spring Cloud simplifies service binding
  - How to extend and customize Spring Cloud, in particular to use new types of bound service



# Roadmap

- **Introduction**
- Service Binding: Auto-Configuration
- Service Binding: Manually
- Extending Spring Cloud



Pivotal™



# Spring Cloud

- Provides tools to implement common patterns in distributed systems
  - configuration management, service discovery
  - circuit breakers, intelligent routing, micro-proxy
  - control bus, one-time tokens, global locks
  - leadership election, distributed sessions, cluster state
- Umbrella project
  - This section only covers *Spring Cloud Connectors*
    - Primarily used for accessing bound services



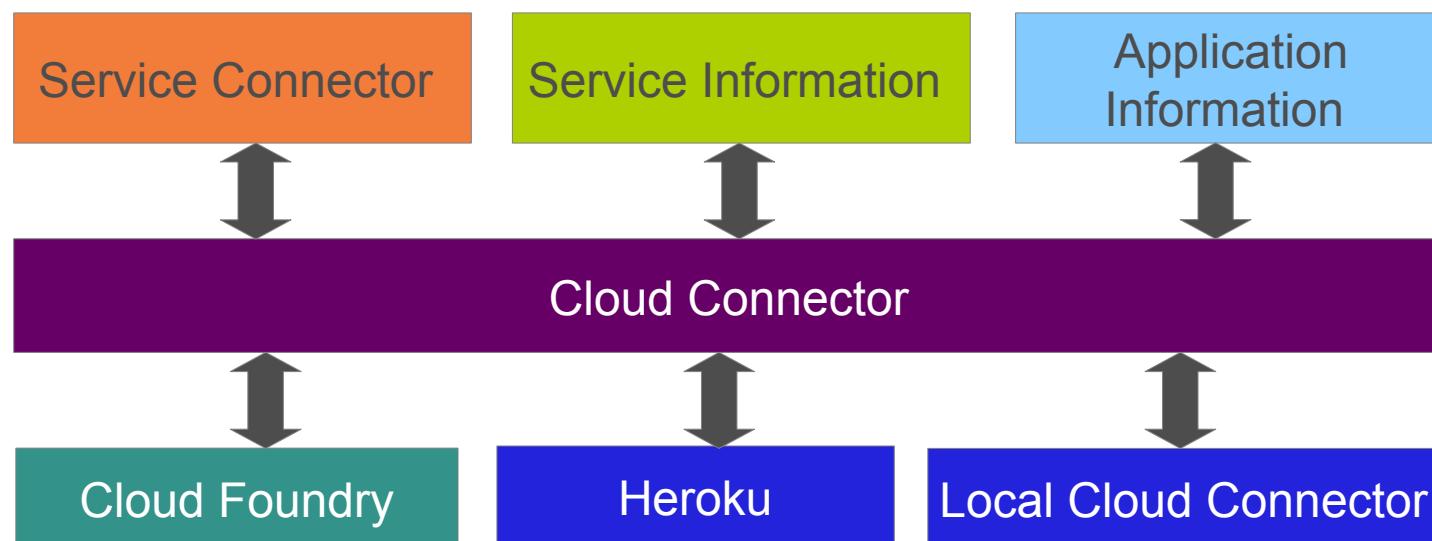
# Spring Cloud Connectors

- Cloud independent access to cloud and application specific data
  - Environment information
  - Services
  - Application information
- Other features
  - Implemented for Cloud Foundry and Heroku
    - Extensible: can support other clouds
  - Supports auto-configuration in Java Buildpack



# Spring Cloud Connectors

- Isolates application from cloud specifics
  - Provides information about the cloud and the application
  - Connection to services
  - Service Information





# Spring Cloud Connectors – Concepts

- **Cloud Connector**
  - Allows Spring Cloud to work with any Cloud
  - Currently Cloud Foundry and Heroku implemented
- **Service Connector**
  - Represents a connection to a service
  - Such as a SQL DataSource
- **Service Information**
  - The underlying service such as host, port, and credentials.
- **Application information**
  - About the application and its running instance(s)

# Roadmap (continued)



- Introduction
- **Service Binding: Auto-Configuration**
- Service Binding: Manually
- Extending Spring Cloud



# Service Binding

- Most common use is for service endpoint information
  - External resources: database, message queue, email server ...
- *12-Factors Design Principle #4*
  - Applications should depend on pre-configured services
  - Available as attached resources
- But how to access those resources?
  - Different environments need a different configuration for the same service
  - How to package for deployment?
  - How to ensure package is same for each environment



# Auto Configuration

- When deploying to Cloud Foundry
  - Java Buildpack automatically re-configures relevant beans to bind them to service instances
    - *Example:* existing DataSource bean overridden to point to bound database service instance
  - ***Careful: You may not always want this!***
    - Manual (explicit) configuration will be covered soon
  - Any custom configuration to your beans *is lost*
    - Your beans are substituted for equivalent bound services
    - Uses configuration details from **VCAP\_SERVICES**

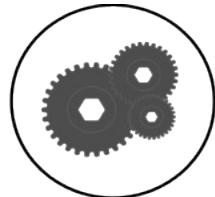
*Java Buildpack:* see <https://github.com/cloudfoundry/java-buildpack>

Pivotal™



# How Auto Configuration Works

- How it works
  - Buildpack adds Spring auto-reconfiguration JAR to Droplet
  - Modifies ApplicationContext at startup
    - Activates “cloud” Profile
    - Exposes CF Environment Variables as PropertySource
  - Bean Post-Processors find beans defining services
    - Transparently replaces bean definitions
  - Reconfigures JDBC, Hibernate, JPA, MongoDB, RabbitMQ, and Redis resources
  - Uses *Spring Cloud Connectors* project



# Auto Configuration Restrictions

Automatic configuration occurs **ONLY IF**:

- Exactly one service instance of a given type is bound
- You are using Spring beans
- Only one bean of a matching type exists
  - *Example:* one bound database service to swap for one DataSource bean



IBM's Liberty Server buildpack supports auto-configuration and Spring is *not* required

# Roadmap (continued)



- Introduction
- Service Binding: Auto-Configuration
- **Service Binding: Manually**
- Extending Spring Cloud



# Manual Configuration

- You may not want to use auto configuration
  - Do programmatically for more control
  - If you include Spring Cloud Connectors as a dependency, auto-configuration is disabled
- Use Spring Cloud Connector facility
  - Via **Cloud** class
    - Example: `cloud.getSingletonServiceConnector`
  - Or via the **cloud** XML namespace
    - Example: use `<cloud:data-source />` instead of regular data source



Spring Cloud Connectors can be used manually with *any* JVM application and using Spring and Spring Beans is optional



# Service Binding

- Using Spring Cloud
  - Java Config: extend `AbstractCloudConfig`

```
@Configuration  
@Profile("cloud")  
public class DbConfig extends AbstractCloudConfig {  
  
    @Bean(name = "dataSource")  
    public DataSource dataSourceRemote() {  
        return cloud().getServiceConnector  
            ("account-db", DataSource.class, null);  
    }  
}
```

No extra configuration – see next slide

- Without Spring configuration – use Cloud Factory

```
Cloud cloud = new CloudFactory().getCloud();  
DataSource ds =  
    cloud.getSingletonServiceConnector(DataSource.class, null);
```



# Configuring the Service

- Use a **ServiceConnectorConfig** instance
  - For example: using **DataSourceConfig** to define a connection pool

```
PoolConfig poolConfig = new PoolConfig(20, 200);
ConnectionConfig connectionConfig =
    new ConnectionConfig("characterEncoding=UTF-8");
DataSourceConfig serviceConfig =
    new DataSourceConfig(poolConfig, connectionConfig);

DataSource inventoryDataSource =
    cloud.getSingletonServiceConnector
        (DataSource.class, serviceConfig);
```



# Java Configuration

```
@Configuration  
public class DbConfig extends AbstractCloudConfig {  
  
    @Profile("cloud") ← cloud profile set by Java Buildpack  
    @Bean(name = "dataSource")  
    public DataSource dataSourceRemote() {  
        return cloud().getServiceConnector()  
            ("postgres-db", DataSource.class, null);  
    }  
  
    @Profile("!cloud") ← Profile when NOT running in the cloud  
    @Bean(name = "dataSource", destroyMethod = "close")  
    public DataSource dataSourceEmbedded() {  
        return new EmbeddedDatabaseBuilder()  
            . setName("accounts")  
            . setType(EmbeddedDatabaseType.HSQL)  
            . addScripts("classpath:scripts/init.sql").build();  
    }  
}
```

Previously: **cf create-service ... postgres-db**



# Accessing Service Information

- ServiceInfo Sub-classes for most common service types
  - RDB (MySQL, Oracle, Postgres), MongoDB, Redis
  - AMQP, SMTP, Monitoring

```
Cloud cloud = new CloudFactory().getCloud();

// Fully manual
List<ServiceInfo> serviceInfos = cloud.getServiceInfos();
for (ServiceInfo serviceInfo : serviceInfos) {
    if (serviceInfo instanceof RelationalServiceInfo) {
        RelationalServiceInfo info = (RelationalServiceInfo) serviceInfo;
        ds = new SimpleDriverDataSource(new Driver(),
            info.getJdbcUrl(), info.getUserName(), info.getPassword());
    }
}
```

Driver class from JDBC jar – must be part of jar/war



# XML Configuration

- Or use *cloud* namespace and profiles

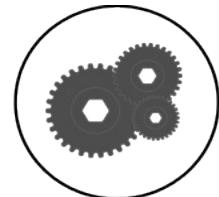
```
<beans xmlns="http://www.springframework.org/schema/beans" ... >

    <beans profile="!cloud">
        <jdbc:embedded-database id="dataSource" type="HSQL">
            <jdbc:script location="classpath:scripts/init.sql" />
        </jdbc:embedded-database>
    </beans>

    <beans profile="cloud">
        <cloud:data-source
                id="dataSource" service-name="postgres-db" />
    </beans>
</beans>
```

# Service-Scanning

Spring can inject services,  
if found (may be *null*)



```
@Controller  
public class HomeController {  
    @Autowired(required = false) DataSource dataSource;  
    @Autowired(required = false) RedisConnectionFactory redisCF;  
    @Autowired(required = false) MongoDbFactory mongoDbFactory;  
    @Autowired(required = false) ConnectionFactory rabbitConnFactory;  
  
    @Autowired ApplicationInstanceInfo appInfo;  
  
    ...  
}
```

```
@Configuration  
@ServiceScan  
@Profile("cloud")  
public class CloudConfig extends AbstractCloudConfig {  
    @Bean(name="applicationInfo")  
    public ApplicationInstanceInfo appInfo() {  
        return cloud().getApplicationInstanceInfo();  
    }  
}
```

Hunt for services



# Recall `VCAP_SERVICES` Property

```
VCAP_SERVICES=
{
  cleardb-n/a: [
    {
      name: "cleardb-1",
      label: "cleardb-n/a",
      plan: "spark",
      credentials: {
        name: "ad_c6f4446532610ab",
        hostname: "us-cdbr-east-03.cleardb.com",
        port: "3306",
        username: "b5d435f40dd2b2",
        password: "ebfc00ac",
        uri: "mysql://b5d435f40dd2b2:ebfc00ac@us-
              cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab",
        jdbcUrl: "jdbc:mysql://b5d435f40dd2b2:ebfc00ac@us-
                  cdbr-east-03.cleardb.com:3306/ad_c6f4446532610ab"
      }
    }
  ]
}
```

*Spring Cloud gets service information from the same place as any cloud application*



# Cloud and Application Information

- Reads **VCAP\_APPLICATION** and the Environment

```
// Obtain the Cloud abstraction:  
Cloud cloud = new CloudFactory().getCloud();  
  
// Information about this instance:  
ApplicationInstanceInfo info = cloud.getApplicationInstanceInfo();  
logger.info("App id=" + info.getAppId()  
          + ", instance=" + info.getInstanceId());  
  
// Cloud properties:  
Properties p = cloud.getCloudProperties();
```

# Roadmap (continued)



- Introduction
- Service Binding: Auto-Configuration
- Service Binding: Manually
- **Extending Spring Cloud Connectors**



# Extending Spring Cloud Connectors

- To support a new cloud
  - Implement `CloudConnector`
- Add a new type of bound service
  - Implement `ServiceInfo`, `ServiceInfoCreator` and `ServiceConnectorCreator`
- For more information
  - <https://spring.io/blog/2014/08/05/extending-spring-cloud>

# Cloud Connector

- Interfaces to a new cloud
  - Extracts information about the application and bound services

```
public interface CloudConnector {  
    boolean isInMatchingCloud();  
    ApplicationInstanceInfo getApplicationInstanceInfo();  
    List<ServiceInfo> getServiceInfos();  
}
```

- For example the Cloud Foundry connector parses the **VCAP\_APPLICATION** and **VCAP\_SERVICES** from the environment

# Defining a New Type of Service – 1a

- First define a **ServiceInfoCreator**
  - SI is a new **ServiceInfo** sub-type
    - You must also define this for your new service type
  - SD is the raw data about available services
    - Type depends on Cloud
    - In Cloud Foundry, SD is a Map based on the **VCAP\_SERVICES** environment variable

```
public interface ServiceInfoCreator<SI extends ServiceInfo, SD> {  
    public boolean accept(SD serviceData); // My service info?  
    public SI createServiceInfo(SD serviceData); // Process it  
}
```

# Defining a New Type of Service – 1b

- Can extend `CloudFoundryServiceInfoCreator`
  - Constructor accepts
    - Tags: list of strings
    - URI Scheme
- Implements `accept()` by iterating over all bound services
  - Returns true if any service name or label matches (starts with) one of the Tag strings
  - Or if any service has a URL whose scheme matches
    - In `http://localhost` the http is the *scheme*

# Defining a New Type of Service – 2

- Finally define a **ServiceConnectorCreator**
  - Provides access to your **ServiceInfo** sub-type
  - Easier to extend **AbstractServiceConnectorCreator**
  - SC is the actual type of your new service connection
    - For an RDB, SC would be a **DataSource**
  - SI is your custom **ServiceInfo** sub-class

```
public interface ServiceConnectorCreator<SC, SI extends ServiceInfo> {  
    SC create(SI serviceInfo,  
              ServiceConnectorConfig serviceConnectorConfig);  
    Class<SC> getServiceConnectorType();  
    Class<?> getServiceInfoType();  
}
```

Required to use `getServiceConnector` and `getSingletonServiceConnector`

# Example: SMTP Mail Service – 1

- Here is the implementation of the SMTP service connector
- The service info is very simple

```
public class SmtpServiceInfo extends UriBasedServiceInfo {  
  
    public SmtpServiceInfo(  
        String id, String host, int port,  
        String username, String password) {  
        super(id, "smtp", host, port, username, password, "");  
    }  
}
```

# Example: SMTP Mail Service – 2

```
public class SmtpServiceInfoCreator extends  
    CloudFoundryServiceInfoCreator<SmtpServiceInfo> {  
  
    public SmtpServiceInfoCreator() { // Define my type  
        super(new Tags("smtp"), "smtp");  
    }  
  
    public SmtpServiceInfo  
        createServiceInfo(Map<String, Object> serviceData) {  
        String id = (String) serviceData.get("name");  
        Map<String, Object> credentials =  
            (Map<String, Object>) serviceData.get("credentials");  
        String host = (String) credentials.get("hostname");  
  
        ... // Get port, username and password from serviceData  
  
        String uri = new UriInfo  
            ("smtp", host, port, username, password).toString();  
        return new SmtpServiceInfo(id, uri);  
    }  
}
```

**Data for a service matching smtp**

# Example: SMTP Mail Service – 2

- Creator uses service info to create an instance

```
public class MailSenderCreator extends  
    AbstractServiceConnectorCreator<MailSender, SmtpServiceInfo> {  
    @Override  
    public MailSender create(SmtpServiceInfo serviceInfo,  
                            ServiceConnectorConfig config) {  
        JavaMailSenderImpl mailSender = new JavaMailSenderImpl();  
  
        mailSender.setHost(serviceInfo.getHost());  
        mailSender.setPort(serviceInfo.getPort());  
        mailSender.setUsername(serviceInfo.getUserName());  
        mailSender.setPassword(serviceInfo.getPassword());  
  
        return mailSender;  
    }  
}
```

# Summary

- After completing this lesson, you should have learned:
  - What Spring Cloud Connectors provide
  - How Spring Cloud Connectors simplify service binding
  - How to extend and customize Spring Cloud Connectors



# Lab

Configuring and running a Java application locally and in the cloud

# Appendix



- Using Spring Cloud Connectors you can access
  - Environment variables
  - Cloud properties
  - Application/Instance name, id and properties
- This appendix shows a typical example of this data
  - You will notice that many Cloud and Application properties are the same

# Typical Environment Variables – I

OLDPWD	/home/vcap
VCAP_SERVICES	{ }
TMPDIR	/home/vcap/tmp
SHLVL	1
JAVA_HOME	/home/vcap/app/.java-buildpack/open_jdk_jre
VCAP_APP_PORT	61035
XFILESEARCHPATH	/usr/dt/app-defaults/%L/Dt
PATH	/bin:/usr/bin
VCAP_APP_HOST	0.0.0.0
USER	vcap
JAVA_OPTS	-Djava.io.tmpdir=/home/vcap/tmp -XX:OnOutOfMemoryError=/home/vcap/app/.java-buildpack/open_jdk_jre/bin/killjava.sh -Xmx382293K -Xms382293K -XX:MaxPermSize=64M -XX:PermSize=64M -Xss995K -Daccess.logging.enabled=false -Dhttp.port=61035

# Typical Environment Variables – II

PWD	/home/vcap/app
PORT	61035
HOME	/home/vcap/app
VCAP_APPLICATION	{"limits": {"mem":512,"disk":1024,"fds":16384}, "application_version":"3fa8051f-81b7-45df-b4aa-debdd2d71538", "application_name":"cf-cloud-info-solution", "application_uris":["cf-cloud-info-solution.cfapps.io"], "version":"3fa8051f-81b7-45df-b4aa-debdd2d71538", "name":"cf-cloud-info-solution", "space_name":"staging", "space_id":"1b39fe5b-c898-4080-ba33-cc8248588e28", "uris":["cf-cloud-info-solution.cfapps.io"]}, ...}
MEMORY_LIMIT	512m
NLSPATH	/usr/dt/lib/nls/msg/%L/%N.cat

# Cloud Properties – I

- All properties are `cloud.properties.xxx`, for example  
`cloud.properties.state_timestamp`

state_timestamp	1409208170
uris	[cf-cloud-info-solution.cfapps.io]
limits	{mem=512, disk=1024, fds=16384}
version	3fa8051f-81b7-45df-b4aa-debdd2d71538
name	cf-cloud-info-solution
application_id	32f6c4d3-ddd7-478a-b419-10b9f752872c
application_uris	[cf-cloud-info-solution.cfapps.io]
started_at_timestamp	1409208170
instance_id	d705cef2a0184ca29f360d3044cefb2b
cspace_name	staging

# Cloud Properties – II

application_name	cf-cloud-info-solution
application_version	3fa8051f-81b7-45df-b4aa-debdd2d71538
started_at	2014-08-28 06:42:50 +0000
start	2014-08-28 06:42:50 +0000
port	61035
instance-id	d705cef2a0184ca29f360d3044cefb2b
host	0.0.0.0
app-id	cf-cloud-info-solution
instance_index	0
space_id	1b39fe5b-c898-4080-ba33-cc8248588e28

# Application Information – I

<i>Application Name</i>	cf-cloud-info-solution
<i>Instance Id</i>	d705cef2a0184ca29f360d3044cefb2b
limits	{mem=512, disk=1024, fds=16384}
application_version	3fa8051f-81b7-45df-b4aa-debdd2d71538
application_name	cf-cloud-info-solution
application_uris	[cf-cloud-info-solution.cfapps.io]
version	3fa8051f-81b7-45df-b4aa-debdd2d71538
name	cf-cloud-info-solution
space_name	staging
space_id	1b39fe5b-c898-4080-ba33-cc8248588e28

# Application Information – II

```
space_id          1b39fe5b-c898-4080-ba33-cc8248588e28
uris              [cf-cloud-info-solution.cfapps.io]
users
application_id   32f6c4d3-ddd7-478a-b419-10b9f752872c
instance_id       d705cef2a0184ca29f360d3044cefb2b
instance_index    0
host               0.0.0.0
port               61035
started_at        2014-08-28 06:42:50 +0000
started_at_timestamp 1409208170
start              2014-08-28 06:42:50 +0000
state_timestamp   1409208170
```



# Microservices

## Challenges to Implementing Microservices

Microservices overview

# Overview

- After completing this lesson, you should be able to:
  - Describe Microservices
  - Discuss their advantages and challenges



# Roadmap



- **What are Microservices?**
- Challenges

Good overview:

<http://martinfowler.com/articles/microservices.html>



Pivotal™

# What are Microservices?

- *Loosely Coupled, Service Oriented Architecture (SOA) with Bounded Contexts*
  - Adrian Cockcroft (Netflix)
- Familiar concepts, reimagined
  - Loosely coupled
    - Deploy any time, no dependencies on anything else
    - SOA with an ESB is not loosely coupled
  - *Service Oriented Architecture*
    - Inherently distributed system
    - But using simpler components (services)
    - Designed that way, not a reuse mechanism for Silos



# Bounded Concepts



- From Eric Evan's "*Domain-Driven Design*" book
  - The setting in which a word or a statement appears that determines its meaning
- Given a central concept, each use is a separate context
  - Example: "reservation" in an airline booking system
- Difficult in a single “monolithic” application
  - Easier with microservices
    - Each can implement the same concept to suit their use of it
    - Each is free to have its own *independent* representation
- A *self-consistent* subset of the domain used by a micro-service

# Three Tenets of Microservices

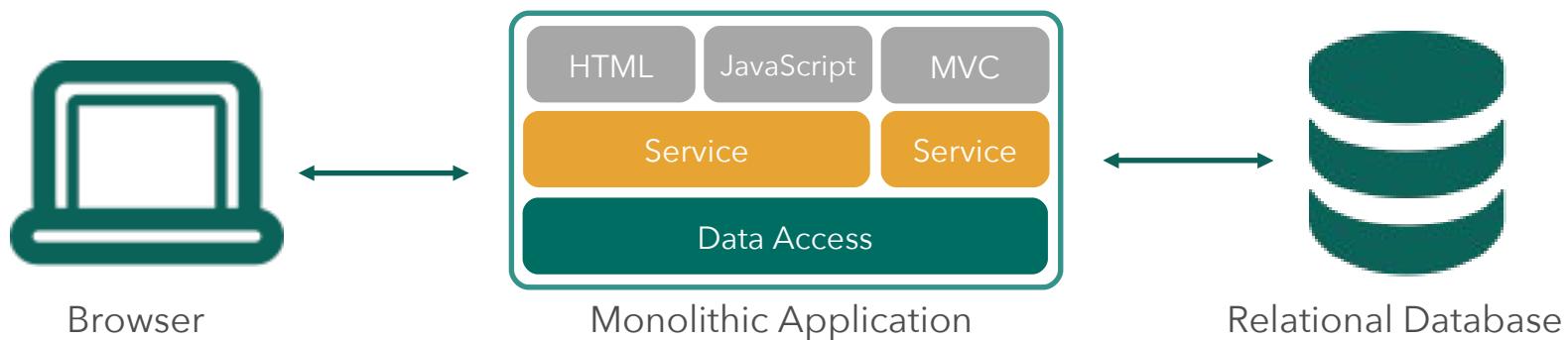


- Gary Olliffe, research director at Gartner
  - Consuming services separate from provisioning services
  - Separating infrastructure management from the delivery of the application capability
    - Using a PaaS like Cloud Foundry
  - Separating teams and decoupling services
    - Each can be built, enhanced and deployed separately
    - Embrace Dev Ops to do this successfully

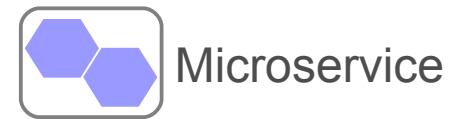
<http://sdtimes.com/digging-into-microservices>

# Monolithic Architecture

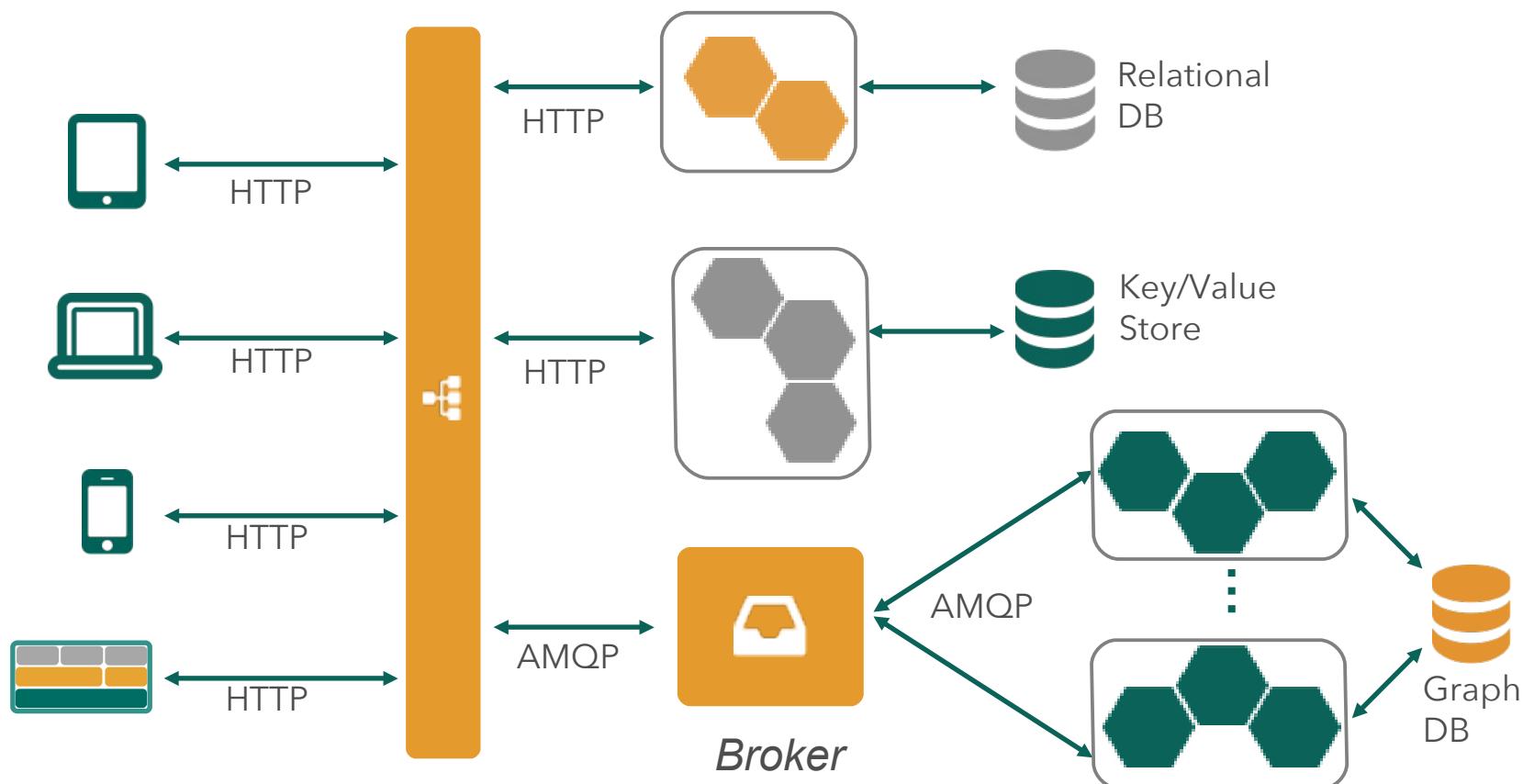
- How apps have traditionally been developed
  - Large, involved code-base
  - Infrequent updates
  - Risky to make small changes



# Microservices Architecture



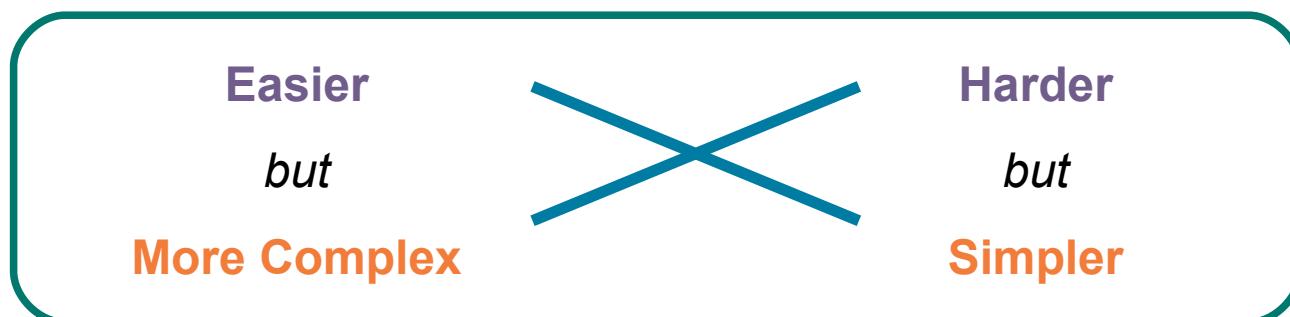
- Decompose into collaborating components



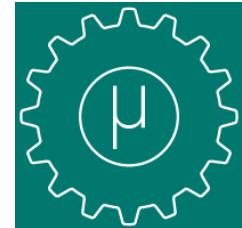
# Trade-Off



- **Monolith**
  - Easier to build
  - But *ultimately* more complex to enhance and maintain
  - Scaling Up (bigger processors) limited
- **Microservices**
  - Harder to build
  - But *ultimately* simpler to extend, enhance and maintain
  - Scaling Out (more processes) easier



# Roadmap



- What are Microservices?
- Challenges
- Spring Cloud
- Building a Microservices Application



Pivotal™

# Qualification Test

- Microservices are not for everyone
  - It's as much *how* you develop as *what* you develop
- "*You must be this tall*" to "ride" *Microservices*
  - Rapid provisioning
  - Basic monitoring
  - Rapid Application Development
  - Devops culture

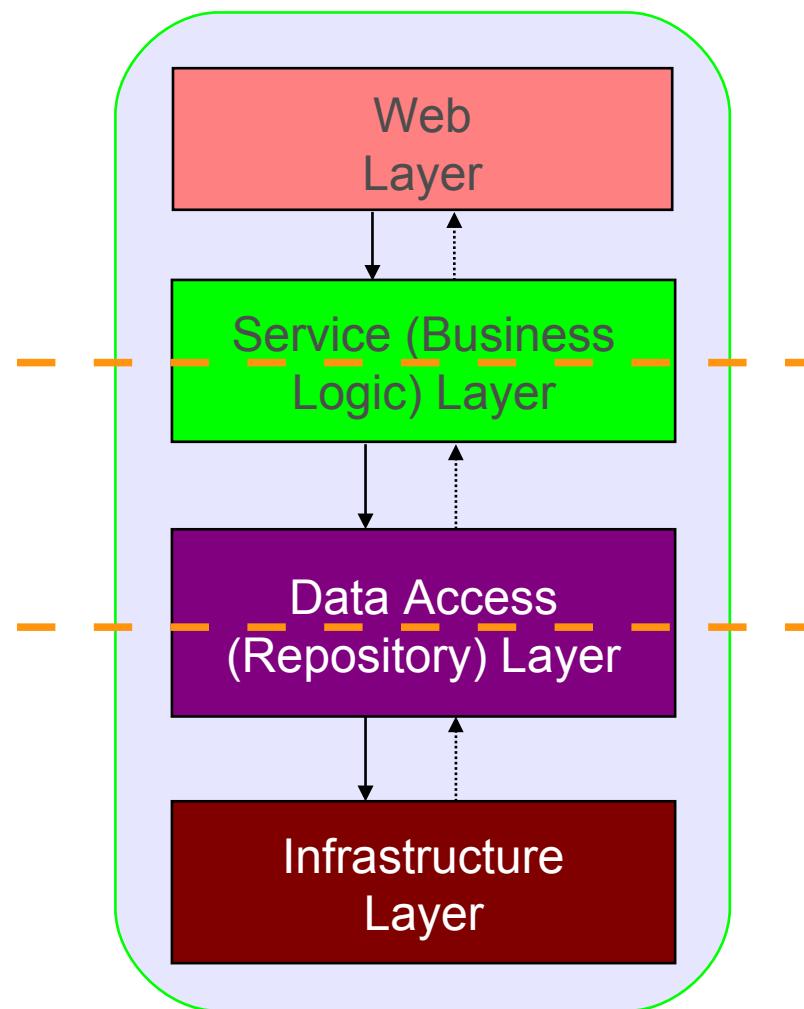


# Route To Microservices: Existing App

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

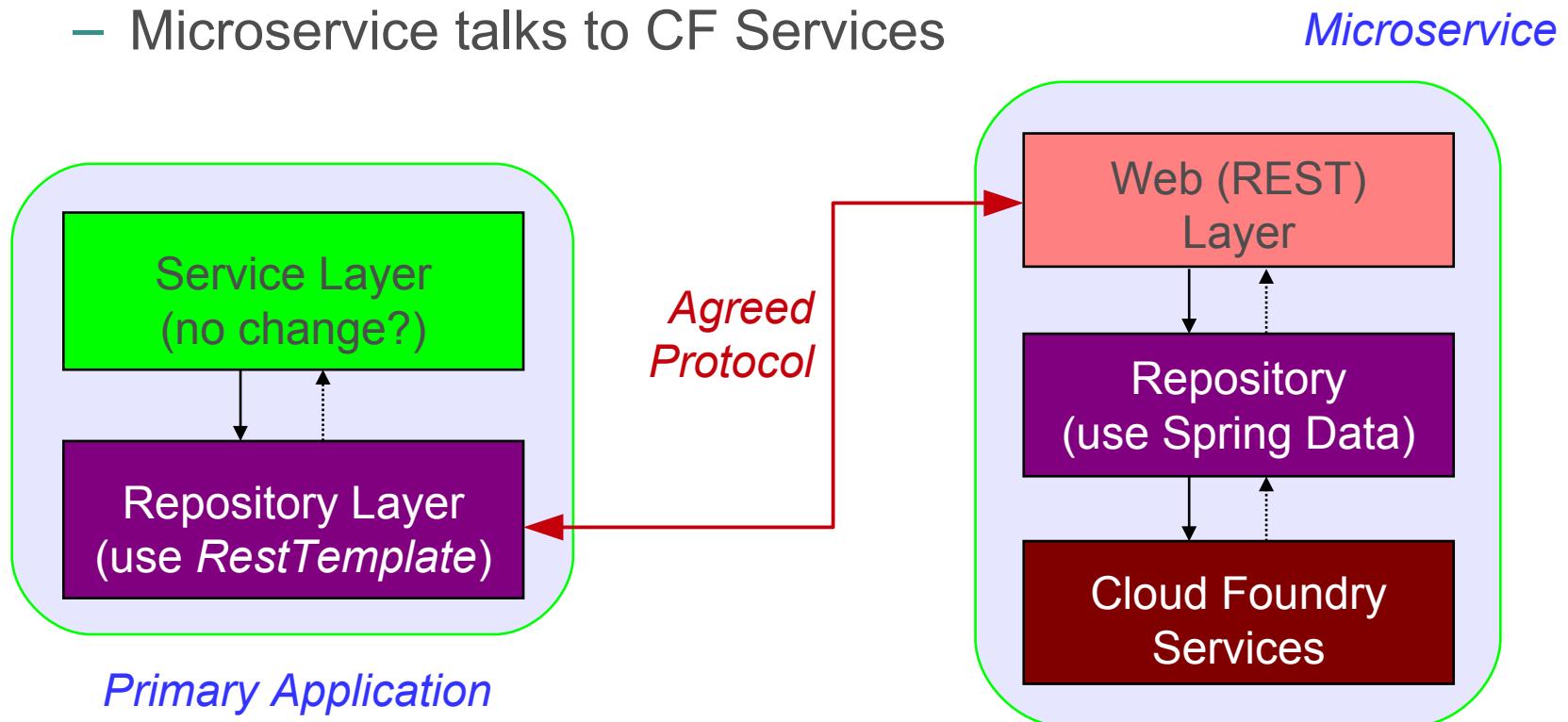
# Decomposing the Monolith

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



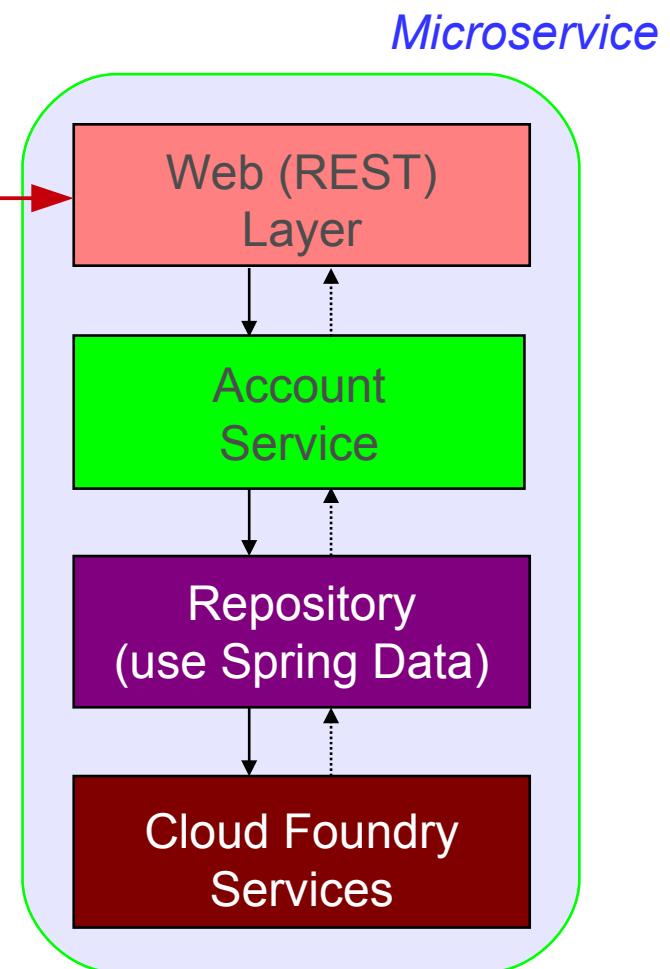
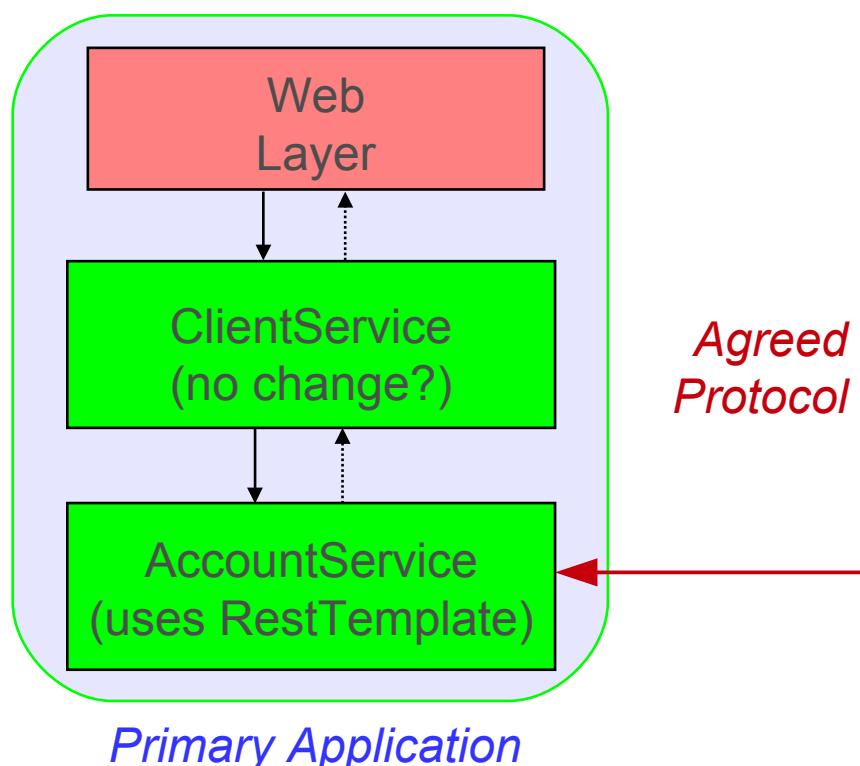
# Refactoring to Microservices Architecture

- Refactor the repository to talk to the microservice
  - Any protocol you like, here using REST
  - Microservice talks to CF Services



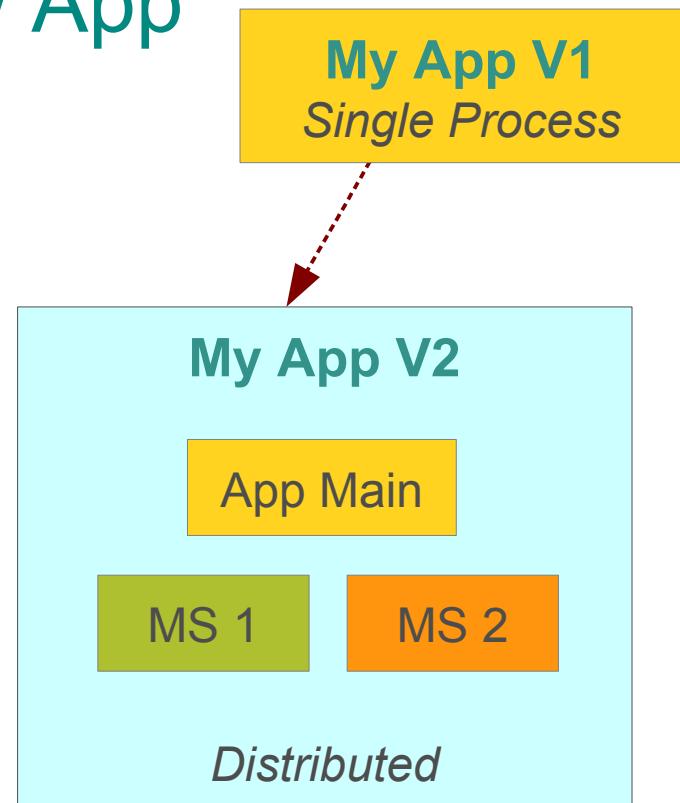
# Refactoring to Microservices Architecture

- The Microservice *is* a service
  - Refactor at Service layer



# Route to Microservices: New App

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



# Transactions - I

- What happens if I need to co-ordinate a change to multiple microservices at the same time?
  - Transactions help with consistency, but force significant temporal coupling
  - Distributed transactions are notoriously difficult to implement
- Microservice architectures emphasize transactionless coordination between services,
  - Eventual consistency is all you can guarantee
  - Problems are dealt with by “*compensating*” operations.
    - A transaction to undo a previous transaction

# Transactions – II

- Many businesses already do this
  - But it's new to developers
- Businesses may choose to handle a degree of inconsistency in order to respond quickly to demand
  - Define a reversal process to deal with mistakes
  - Trade-off is worth it *if* cost of fixing mistakes is less than the cost of lost business

See <http://martinfowler.com/articles/microservices.html>

# Deployment Challenges – I

- Inherently more complex, distributed architecture
- We need to support
  - Configuration management
  - Service registration and discovery
  - Routing and load balancing
  - Fault tolerance
  - Monitoring the individual components
  - *And also need a global/consolidated view (next slide)*

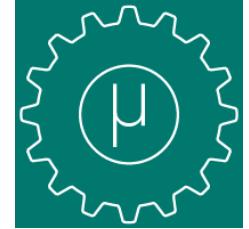


# Deployment Challenges – II

- *No microservice is an island – Dr Dave Syer (Pivotal)*
  - Must be part of an “archipelago”
- How to handle a whole (composite) system?
  - The “Big A” app
  - CF manifest does some of this, but its static
  - Static vs dynamic - need "BOSH for microservices" = PCF
  - Decentralized, autonomous capability required
    - Different teams can deploy at any time
    - You own it, you write it, you run it!

# Microservices and Cloud Foundry

- What does a microservice application require?
  - Environment provisioning
  - On-demand scaling
  - Failover and resilience
  - Routing and Load balancing
  - Data services ops (BOSH)
- Cloud Foundry gives you all these
  - Don't *have* to deploy to a PaaS, but it works well
  - A naturally symbiotic relationship



# What we have learnt

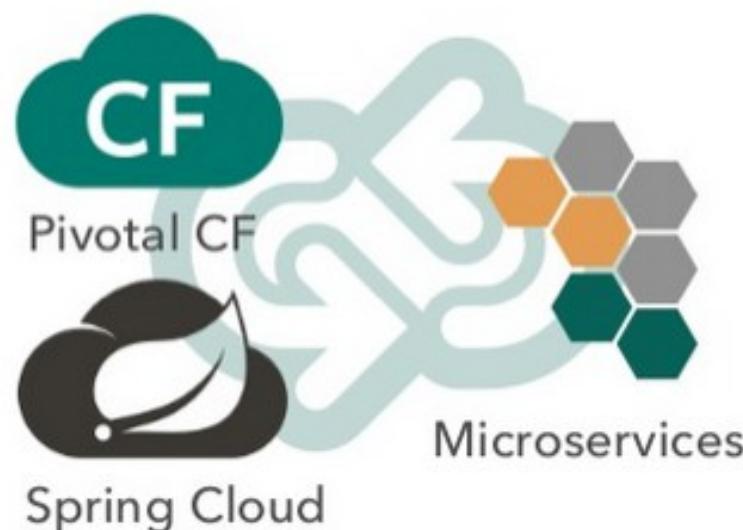
- What are Microservices?
- Challenges



Pivotal™

# Summary

- After completing this lesson, you should have learnt:
  - What is a Microservice Architecture
  - Advantages and Challenges of Microservices





# Microservices with Spring

Implementing Microservices with Spring and Cloud Foundry

Microservices overview, Spring Cloud

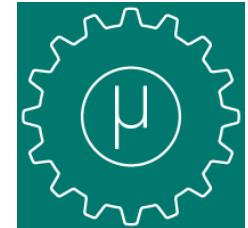
Pivotal

# Overview

- After completing this lesson, you should be:
  - Familiar with Spring Cloud projects
  - Using Spring Cloud to develop a Microservices application



# Roadmap



- **Spring Cloud**
- Building a Microservices Application



Pivotal™

# Building Blocks



- Netflix Open Source Software projects
  - They already did this and made much of the work OSS
- Spring Cloud
  - *Distributed System Patterns for the web*
  - Apply to the Netflix components
    - so you don't have to learn/understand them
- Complements the PaaS
  - Cloud Foundry ~ Deploy running apps
  - Spring Cloud ~ Compose a Fault Tolerant System



# About Spring Cloud

- Does *more* than just expose bound services in cloud applications
  - Wraps up and makes available useful services
    - Several based on Netflix OSS
    - Other OSS options (Consul) available
  - Microservices Infrastructure
    - Config Server / Cloud Bus
    - Service Registry (Eureka)
    - Load Balancer (Ribbon)
    - Circuit Breakers (Hystrix)
    - Intelligent Routing (Zuul)



# Using Spring Cloud

- Requires Spring Boot
  - Via starter dependencies (here using Maven)

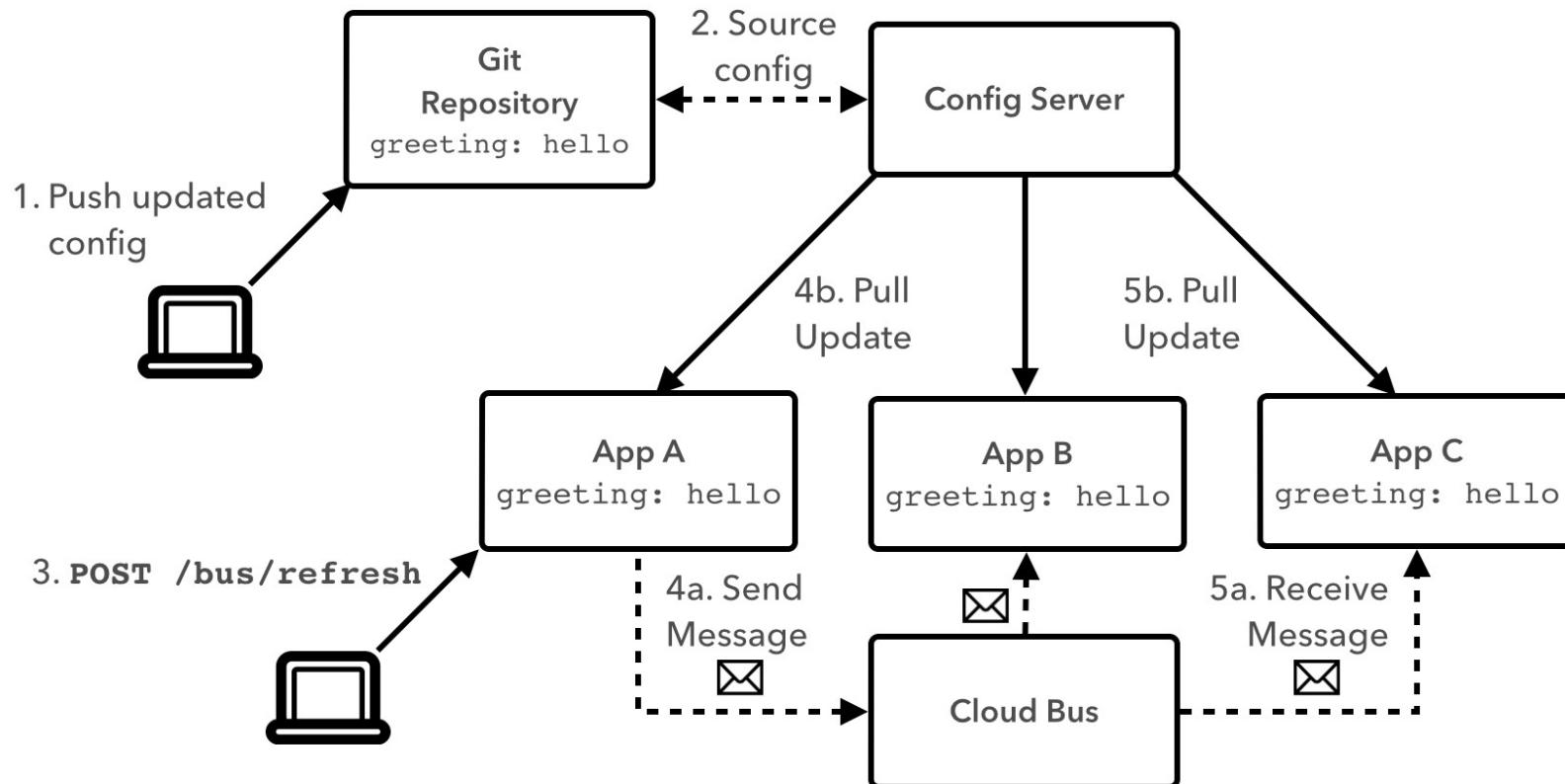
```
@SpringBootApplication  
public class BootApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(BootApplication.class, args);  
    }  
}  
  
<dependency>  
    <!-- Setup Spring MVC & REST, use Embedded Tomcat -->  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter</artifactId>  
</dependency>
```

main.java

# Pattern 1: Dynamic Reconfiguration

- Change system properties and configuration dynamically
  - Use a repository to maintain versions of configuration
    - Either a Java Properties or YML configuration file
    - Default implementation: uses Github
  - *Cloud Bus* pushes changes to running processes
    - Processes listen for changes: *EnvironmentChangedEvent*
    - Spring beans reconfigured and recreated
- To update processes
  - Spring Boot: each process has a `/refresh` endpoint
  - Or get `RefreshScope` Spring bean, invoke `refresh()`

# Config Server + Cloud Bus



# Config Server

- Responds to `http://cfhost/application/profile/label`
  - Application name, Spring profile, Label (=version)
  - Map to `<application>-<profile>.xxx` and
    - Where `xxx` is `properties`, `yaml` or `yml`
    - Also loads `application.xxx` if found
- By default downloads files from github repository
  - Then loads properties from files matching patterns above
    - Label (if used, optional) maps to a branch
  - Or set profile “*native*” and it loads from local dir instead
- Open <https://github.com/spring-cloud-samples/config-repo>
  - What does `http://cfhost/foo/development` load?

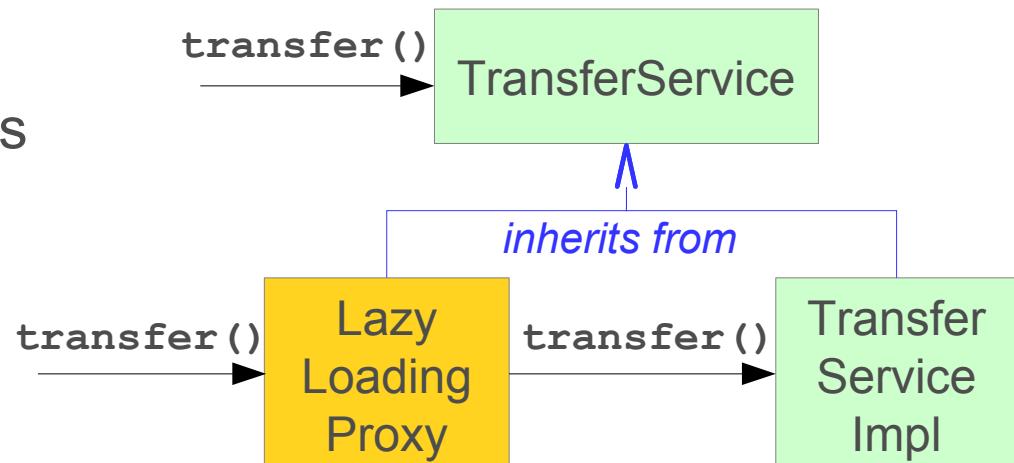
# Config Clients

- Automatically connect to Config Server if running
  - Dep: `org.springframework.cloud:spring-cloud-config-client`
  - Loads properties corresponding to their application name and profile(s).
  - Label defaults to “master” - git master branch
- Integrate with Spring Bus
  - If any client invokes refresh, they all do
  - Bus passes updated config to all clients
  - Clients may recreate beans in response – next slide

# Refreshable Beans

- **@RefreshScope**
  - Annotate reconfigurable Spring beans
    - Replaced by lazy-loading proxy at runtime
  - If underlying bean is recreated, next access gets *new* bean

```
@Configuration  
public class TransferConfig {  
  
    @Bean  
    @RefreshScope  
    // Actually returns a proxy  
    public TransferService transferService() {  
        ...  
    }  
}
```

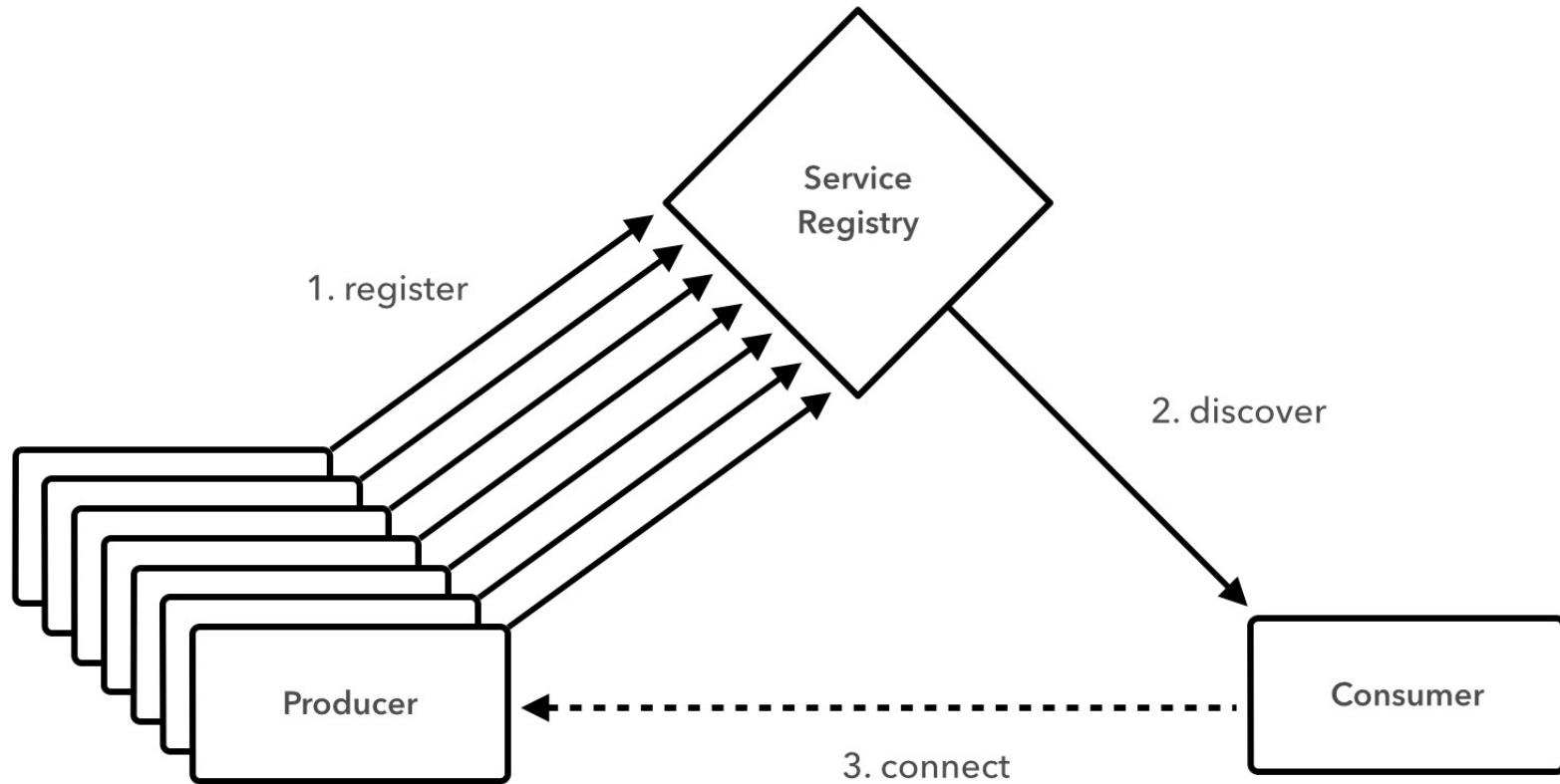


# Refresh Example

- Bean created
  - **dailyLimit** set from a PropertySource
  - PropertySource loaded from Config Server
    - Which has **transfers.daily.limit** in one of its config files
- Configuration updated
  - **transfers.daily.limit** modified and config file “pushed”
  - Refresh performed, new PropertySource loaded
  - Transfer Service bean recreated
    - Proxy picks up new bean with new **dailyLimit** set from updated **transfers.daily.limit**

```
public interface TransferService {  
    @Value("transfers.daily.limit")  
    protected BigDecimal dailyLimit;  
  
    public Accounts[] transfer  
        (String from, String to, BigDecimal amt);  
}
```

# Pattern 2: Service Registry (Eureka)



See also: <http://spring.io/blog/2015/07/14/microservices-with-spring>

# Deploy Eureka Server using Spring Cloud

- All you need to implement your own registry service!

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

main.java

```
server: application.yml  
port: 8761  
  
eureka:  
  instance:  
    hostname: localhost  
    client: # Not a client  
    registerWithEureka: false  
    fetchRegistry: false
```

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

pom.xml

# Service Registration

- Each microservice declares itself a discovery-client
  - using `@EnableDiscoveryClient`
  - Registered as “accounts” – the application name

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

spring:  
application:  
 name: accounts  
eureka:  
 client:  
 serviceUrl:  
 defaultZone: http://regsvc.cfapps.io/eureka/

Service name

Eureka URL

# Service Discovery Clients

- How do Microservice clients find the service?
  - **Explicit:** Inject a *DiscoveryClient* (next slide)
    - Fetch a service: `discoveryClient.getNextServerFromEureka`
    - Access a service: `serviceInstance.getHomePageUrl()`
  - **Implicit:** Inject a “smart” RestTemplate
    - Spring performs service lookup for you
    - Use *logical* service names in URLs

# Service Discovery Clients – Explicit

```
@RestController
public class AccountController {
    @Autowired
    protected DiscoveryClient discoveryClient;

    protected RestTemplate restTemplate = new RestTemplate();

    @RequestMapping("/{id}")
    public Account getAccount(String id) {
        // Lookup service
        InstanceInfo instance = discoveryClient.
            getNextServerFromEureka("accounts", false);

        // Fetch data
        return restTemplate.getForObject(
            instance.getHomePageUrl() + "/", Account.class, id);
    }
}
```

Service name

# Service Discovery Clients – Implicit

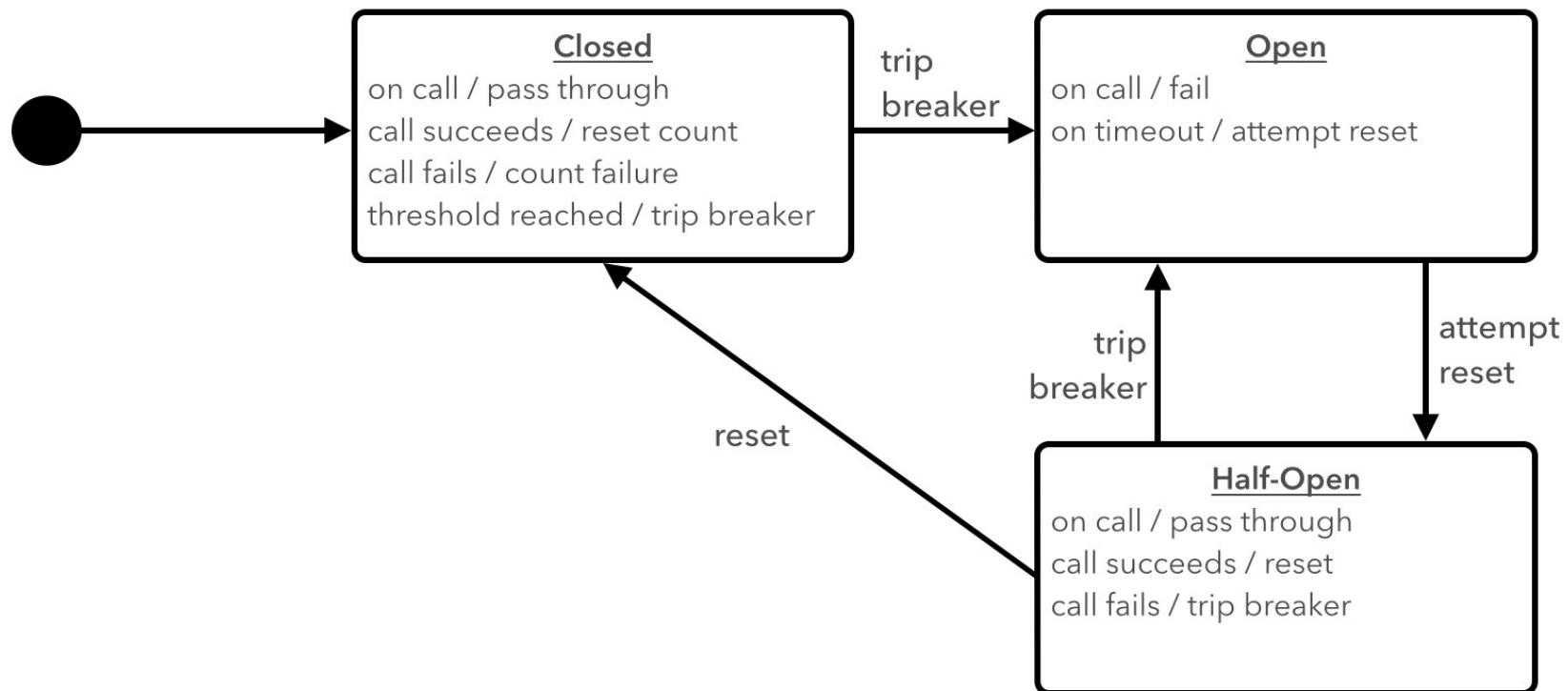
```
@RestController  
@EnableDiscoveryClient  
public class AccountsController {  
  
    // Spring injects a “smart” service-aware template  
    // injected with RibbonHttpRquestClient to do lookup  
    @Autowired  
    RestTemplate restTemplate;  
  
    @RequestMapping("/{id}")  
    public Account getAccount(String id) {  
        // Fetch data  
        return restTemplate.getForObject  
            ("http://accounts/", Account.class, id);  
    }  
}
```

Service name

# Pattern 3: Handling a Failed Service

- What happens if a service stops responding?
- Circuit Breaker
  - Defaults to closed, allows connections
  - If too many connections fail, switches to open
    - All connection attempts now fail (*with no-timeout*)
  - Periodically switches to half-open
    - If connections now succeed, switches to closed
    - Otherwise switches back to open
  - Keeps entering half-open state until normal service resumes
- Spring Cloud uses Netflix *Hystrix* internally

# Circuit Breaker – State Diagram



# Failover with Hystrix

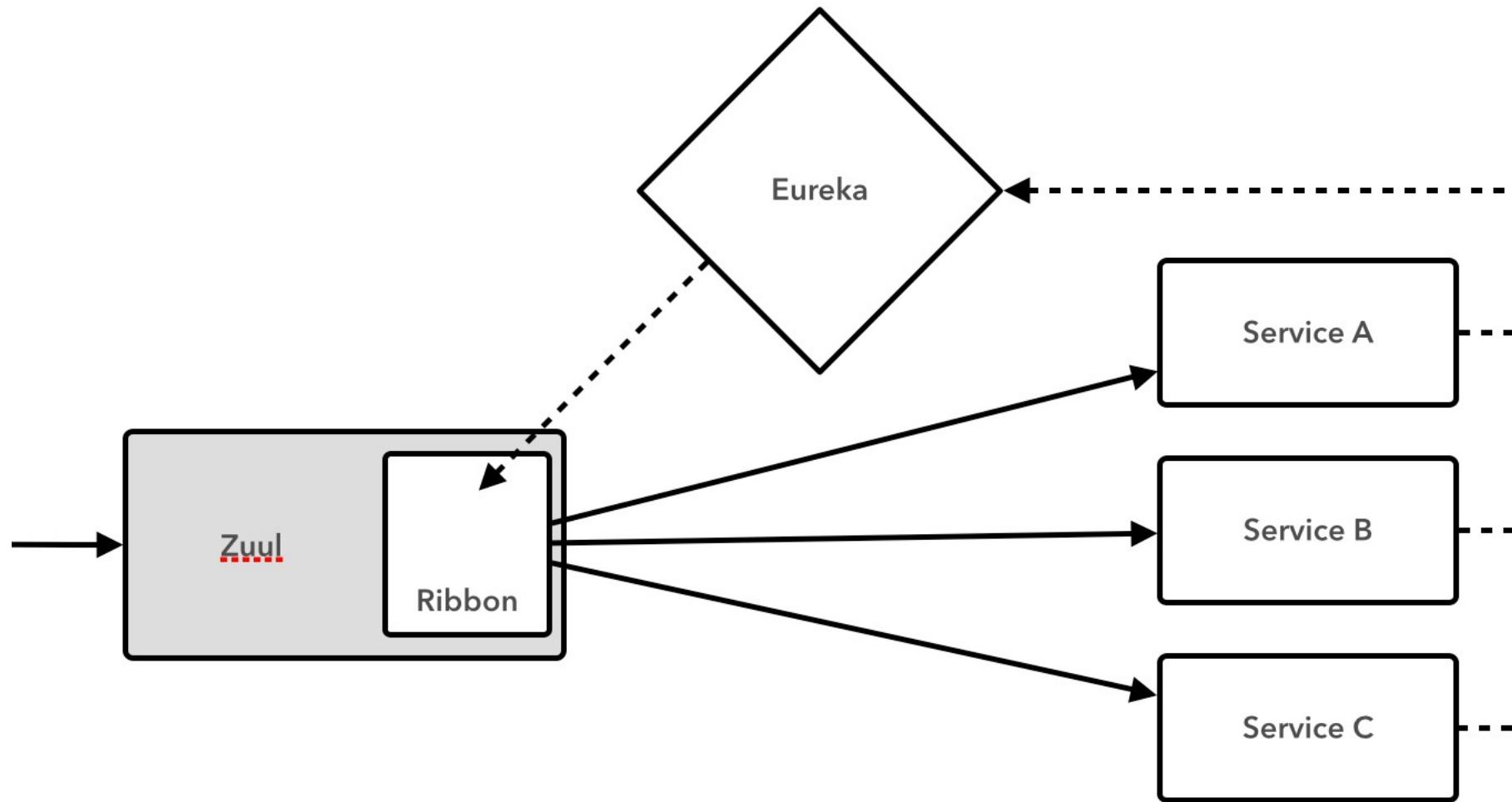
```
@RestController
@EnableCircuitBreaker
public class AccountController {

    @Autowired
    RestTemplate restTemplate;

    // This method wrapped in a Circuit-Breaker
    @HystrixCommand(fallbackMethod = "getAccountFallback")
    @RequestMapping("/{id}")
    public Account getAccount(String id) {
        return restTemplate.getForObject
            ("http://accounts/", Account.class, id);
    }

    // Account Service did not respond - this method called instead
    public Account getAccountFallback() {
        throw new AccountUnavailableException();
    }
}
```

# Pattern 4: Intelligent Routing (Zuul, Ribbon)



# Intelligent Routing

- Spring Cloud automatically integrates three Netflix utilities
  - “Ribbon” load-balancer
  - “Zuul” intelligent router
  - “Eureka” service-discovery
- End result:
  - Determines the best available service to use (when there are multiple, possible instances)
  - Just declare a load-balanced **RestTemplate**
    - Example in a moment
  - CF routes *to* instances, Zuul routes *between* instances

# Alternatives

- Spring Cloud is being extended to use non-Netflix components as an alternative
  - For example service registration using Consul



Pivotal™

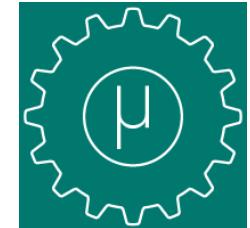
# Spring Cloud

- Project Home page
  - <http://projects.spring.io/spring-cloud>
- See Matt Stine's presentation from CF Summit
  - <https://github.com/mstine/2015-cfsummit-deploying-ms-to-cf>
  - The Netflix diagrams come from that presentation
- Demos using Spring Cloud and Spring Boot
  - <https://github.com/mstine/intro-spring-cloud-workshop>



Pivotal™

# Roadmap



- Spring Cloud
- **Building a Microservices Application**



Pivotal™

# Microservice Application

- Components
  - **Microservices** themselves
    - Typically RESTful applications
    - Use CF provided services
  - **Infrastructure Services**
    - Eureka Service Registration Service
    - Configuration Service and Cloud Bus
  - **Microservice Clients**
    - Typically full-blown Web-Applications
    - Visible to the outside world
    - Make requests of the microservices

# 1. Building a Microservice

- Use Java and Spring
  - Spring Boot
  - Spring MVC, Spring REST, Spring HATEOAS
  - Spring AMQP if using AMQP Messaging
  - Spring Data repositories
  - Spring Cloud for access to persistence services
  - Spring Integration provides many Gateways to interface to other systems
    - Email servers, SOAP applications, Twitter feeds ...
  - *Register with Service Registry* – `@EnableDiscoveryClient`



## 2. Pivotal CF's Spring Cloud Services

- Available as a service from PCF 1.4
  - Install as a “Tile” in usual way
  - No need to write your own
  - Just bind to your application to use
- Provides all the Infrastructure Services
  - Config Server (based on Spring Cloud Config Server)
  - Service Registry (based on Eureka & Spring Cloud Netflix)
  - Circuit Breaker Dashboard (based on Hystrix, Turbine & Spring Cloud Netflix)

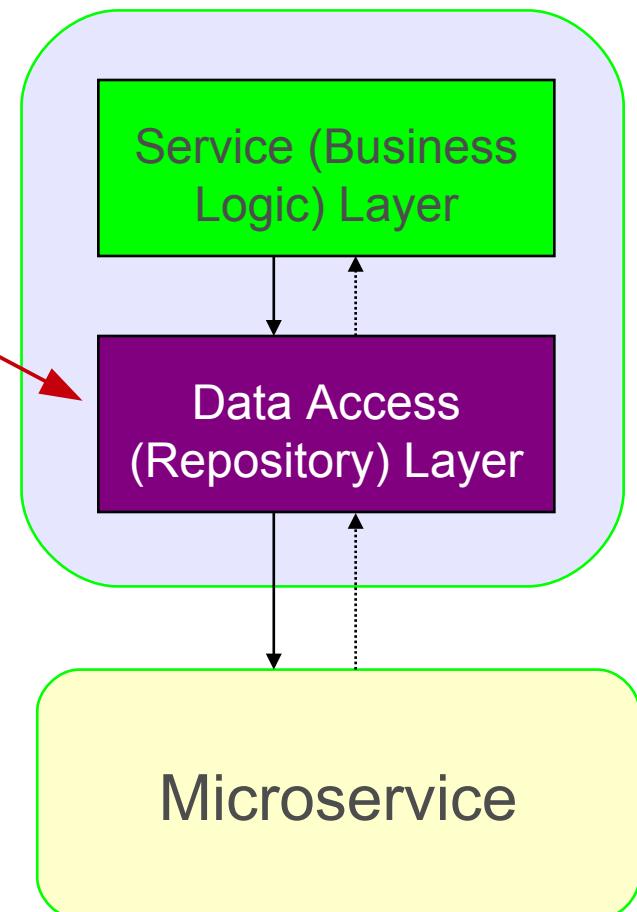
### 3. The Microservices Client

- Uses Spring Boot *and* Spring Cloud
  - Convenient annotation: `@SpringCloudApplication`

```
@SpringCloudApplication  
public class WebApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(WebApplication.class, args);  
    }  
}  
  
@SpringBootApplication // Enable Spring Boot  
@EnableDiscoveryClient // Discover the microservices  
@EnableCircuitBreaker  
public @interface SpringCloudApplication {  
}
```

# Recall: Layered Application Architecture

- Refactor to talk to microservice(s)
  - Either services as normal
    - Data-layer layer accesses *microservice(s)*
  - Or services wrap microservice access
- Any protocol can be used
  - Spring Cloud has convenient support for REST
    - Using *RestTemplate* and *Hystrix* circuit-breaker



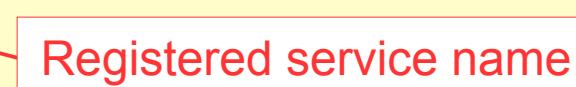
# Example Repository

```
@Repository
@EnableDiscoveryClient
public class UserRepository {

    @Autowired      // The “smart” template (see above)
    @LoadBalanced   // May be multiple microservice instances
    private RestTemplate restTemplate;

    @HystrixCommand(fallbackMethod = "noUserDetails")
    public void UserDetails getUserDetails(String id) {
        return restTemplate.getForObject
            ("http://users/user/{0}", UserDetails.class, id);
    }

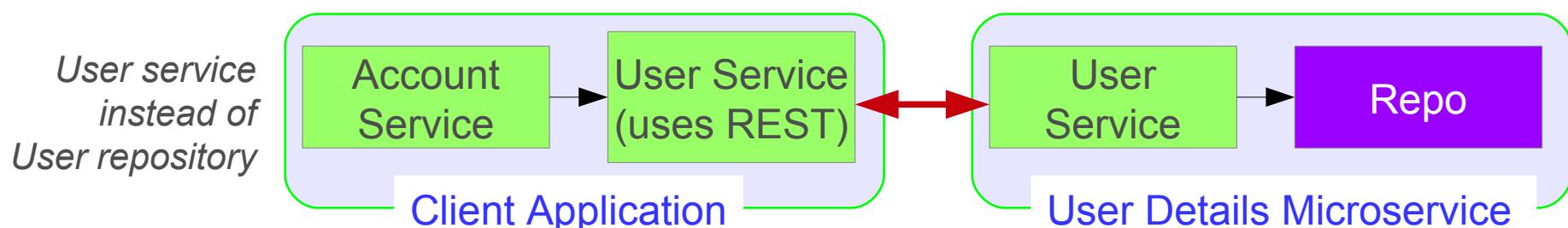
    public void UserDetails noUserDetails(String id) {
        return UserDetails.unavailable(id);
    }
}
```



Registered service name

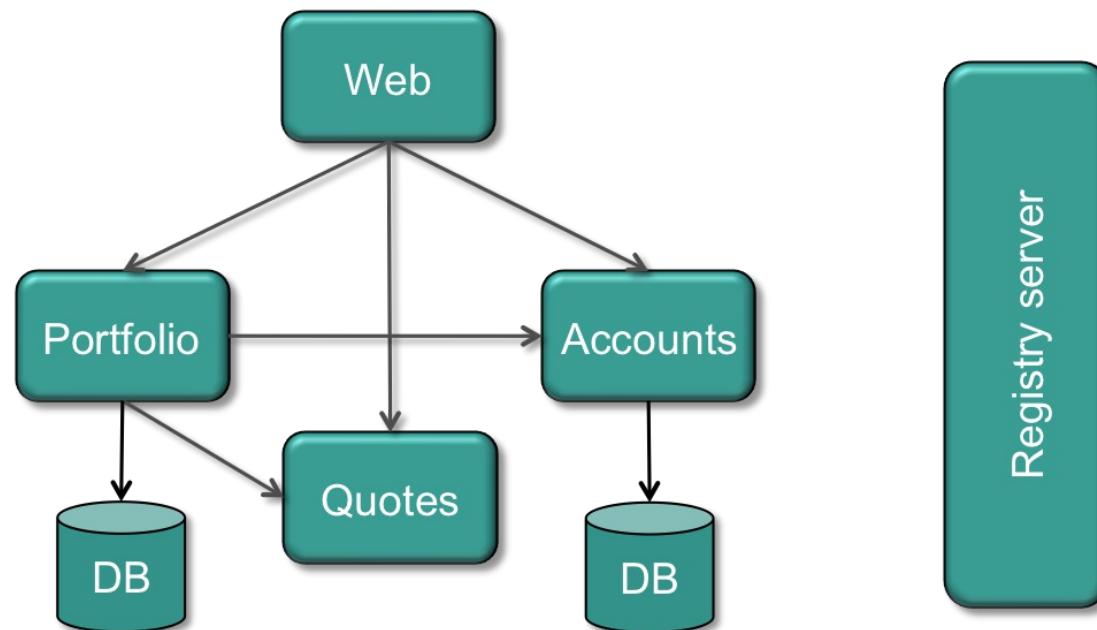
# Are Repositories Needed?

- Service layer *may* use RestTemplate directly
  - But how to test microservice clients standalone?
- Advantages of implementing a Repository layer
  - Repository dependency can be stubbed for testing
    - Stub may be reused by *any* service that needs it
  - May involve less change to an existing architecture
- If Microservices offer more than CRUD
  - Wrap access as a service instead, like this:



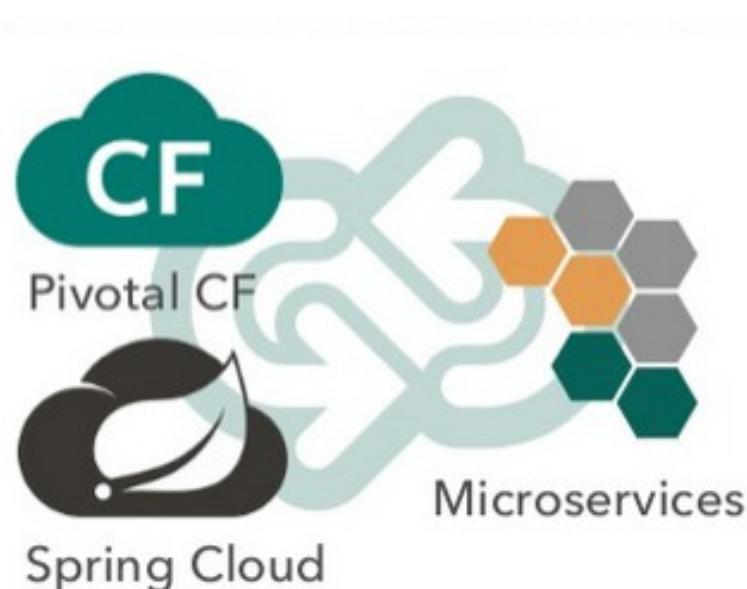
# Spring Trader

- Sample application
  - Web front-end uses three microservices
    - **Note:** Web process *does not* use repositories abstraction



# Summary

- After completing this lesson, you should have learnt:
  - The many projects in Spring Cloud
  - How to build Microservice Applications using Spring Cloud





# Finishing Up

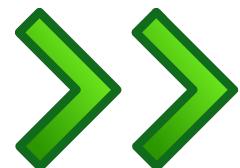
Course Completed

What's Next?

Pivotal

# What's Next

- Congratulations, we've finished the course
- What to do next?
  - Other courses
  - Resources
  - Evaluation

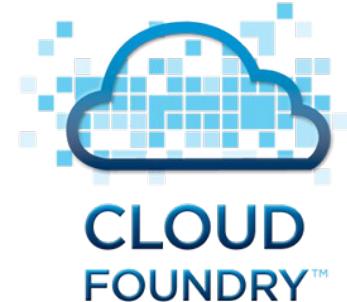


# Other courses



- Many courses available
  - Pivotal Cloud Foundry
  - Spring: Core Spring, Enterprise Spring, Spring Web, What's New in Spring
  - Rabbit/MQ
  - Hadoop, Greenplum ...
- More details here:
  - <http://www.pivotal.io/training>

# Cloud Foundry



- Pivotal CF Immersion
  - Administrators/Dev Ops course
  - Install, Configure and Manage a Cloud Foundry instance
  - Using the Pivotal CF operator's console
  - Introduction to BOSH
  - Organizations, Spaces,
  - Users, Roles, Security
  - Monitoring and Troubleshooting

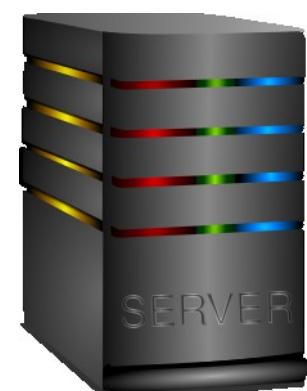
# Certified Spring Courses



- Core Spring
  - All the Spring basics: configuration, data-management, web-applications, messaging
- *Cloud Native Applications with Spring* 
  - *Using Spring Cloud to write cloud-ready applications*
- Enterprise Spring
  - Building loosely coupled, event-driven architectures
- Spring Web
  - Web and REST applications using Spring MVC
- What's New in Spring?
  - Spring 3.x and 4.x updates

# Big Data and Data Analytics

- We offer courses on
  - Pivotal Hadoop
  - Big Data Analytics
  - Pivotal Greenplum and HAWQ
  - Pivotal Gemfire
  - Spring XD
- Full list of courses:
  - <http://www.pivotal.io/training>



Pivotal™

# Pivotal Support

- Support is available for Pivotal Cloud Foundry
  - Options: <http://pivotal.io/support/offers>
  - Register: <http://tinyurl.com/piv-support>
  - Support Portal: <https://support.pivotal.io>
    - Community forums, Knowledge Base, Product documents
  - PWS Support:
    - <https://support.run.pivotal.io> or [support@run.pivotal.io](mailto:support@run.pivotal.io)
- Community forums
  - <https://groups.google.com/a/cloudfoundry.org/forum/#!forumsearch/>



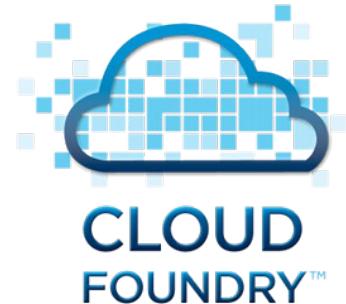
# Pivotal Consulting

- Custom consulting engagement?
  - Contact us to arrange it
    - <http://www.pivotal.io/contact/spring-support>
    - Even if you don't have a support contract!
- Pivotal Labs
  - Agile development experts
  - Assist with design, development and product management
    - <http://www.pivotal.io/agile>
    - <http://pivotallabs.com>



Pivotal™

# Resources



- Cloud Foundry reference documentation
  - <http://docs.cloudfoundry.org>
  - <http://docs.pivotal.io> (Pivotal CF Enterprise)
  - <http://docs.run.pivotal.io> (PWS)
- The official technical blog
  - <http://blog.pivotal.io/cloud-foundry-pivotal>
- Get involved
  - <http://cloudfoundry.org/get-in/index.html>

# Thank You!

We hope you enjoyed the course

- Please fill out the evaluation form





# Installation Overview

Basics of Pivotal CF Installation

Or take the *Pivotal CF Install Configure Manage* course

Pivotal

# Overview

- After completing this lesson, you should:
  - Understand how **Pivotal CF** is installed
    - But not necessarily do it yourself
  - Be familiar with the Ops Manager Console
  - Understand how the Elastic Runtime and CF Services are installed as “*Tiles*”

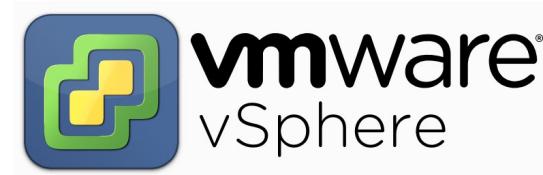
# Roadmap

- Prerequisites
- Configuration
  - Installing Pivotal CF Ops Manager VM
  - Configuring Ops Manager Director
  - Installing Elastic Runtime Tile
  - Installing Service Tile(s)



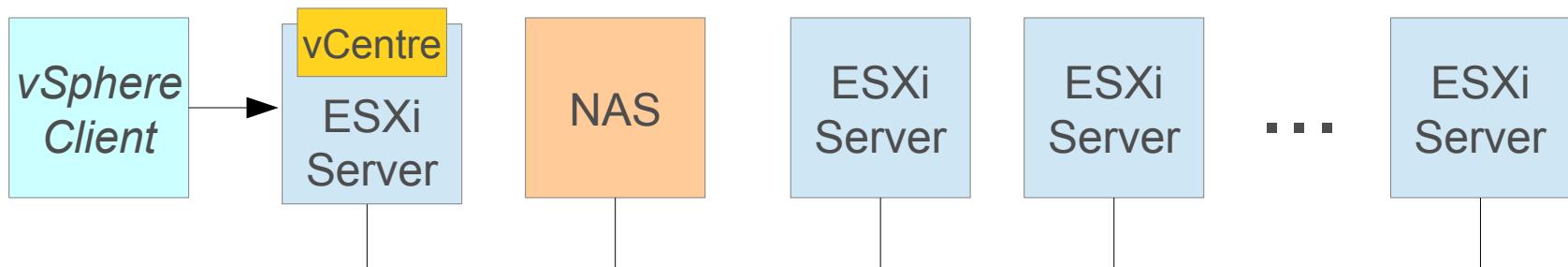
# Prerequisites

- A cluster running VMware virtualization software
  - Most common on-premise scenario
  - Other environments supported: AWS, Open Stack
    - Not covered here
- VMware experience is *very useful*
  - Pivotal CF setup via a prepackaged VM (appliance)
- CF OSS can also be installed on Open Stack, AWS, ...
  - But is harder to do
  - No appliance, you must configure it manually (BOSH scripting)



# Typical VMware Cluster

- The cluster needs the following nodes
  - One vCentre Server to co-ordinate the cluster
  - A vSphere client to configure the cluster (MS Windows)
    - Sends commands to vCentre
  - Network Storage (NAS)
  - ESXi servers (where vCenter & CF VMs actually run)



# Cluster Typical Resources

- vCentre
  - 6 CPUs, 6G RAM, 125G disk
- ESXi
  - 4+ CPUs, 16+ G RAM, 120+G
  - Scale proportionally
- vSphere
  - Windows only
  - 2-4 CPUs, 2G RAM, 20G disk
- NAS
  - 4 CPUs, 2G RAM, 1T disk

*Guidelines  
Only*

# Installation Steps

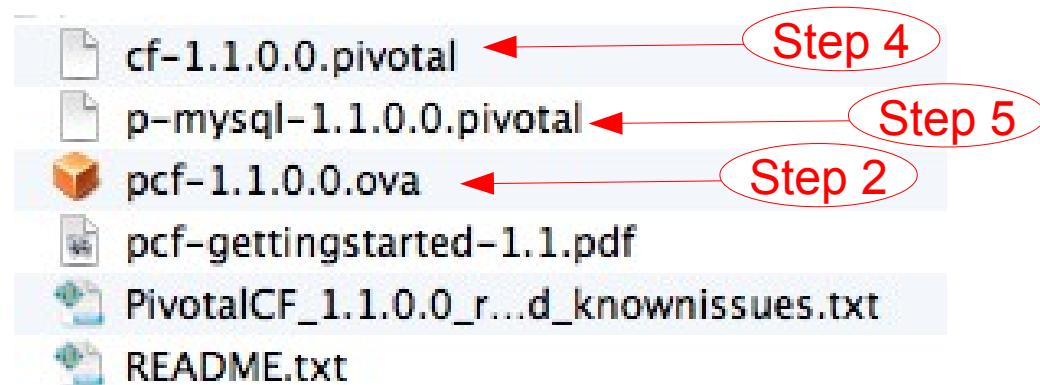
- Five steps
  1. Get the software from the *Pivotal Network*
  2. Install Pivotal Ops Manager appliance (VM)
    - Only Pivotal CF has this
  3. Configure vSphere Tile - in Ops Manager Director
  4. Configure Elastic Runtime Tile (= CF)
  5. Add Services Tiles

# Step 1. Get Software



Pivotal Network

- Download from <http://network.pivotal.io>
  - See also <http://docs.pivotal.io>
  -
- Typical files you might download
  - pcf-xxx.ova = Ops Manager VM
  - cf-xxx.pivotal = CF Elastic Runtime
  - p-mysql-xxx.pivotal = MySQL service
  - Documentation

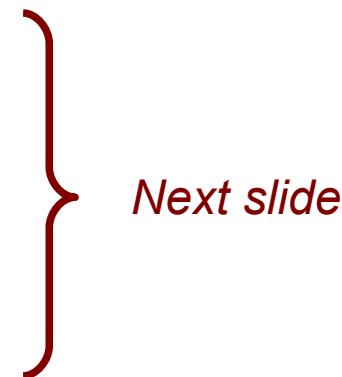


# Roadmap

- Prerequisites
- Configuration
  - **Installing Pivotal CF Ops Manager VM**
  - Configuring Ops Manager Director
  - Installing Elastic Runtime Tile
  - Installing Service Tile(s)

# Step 2. Install Ops Manager VM

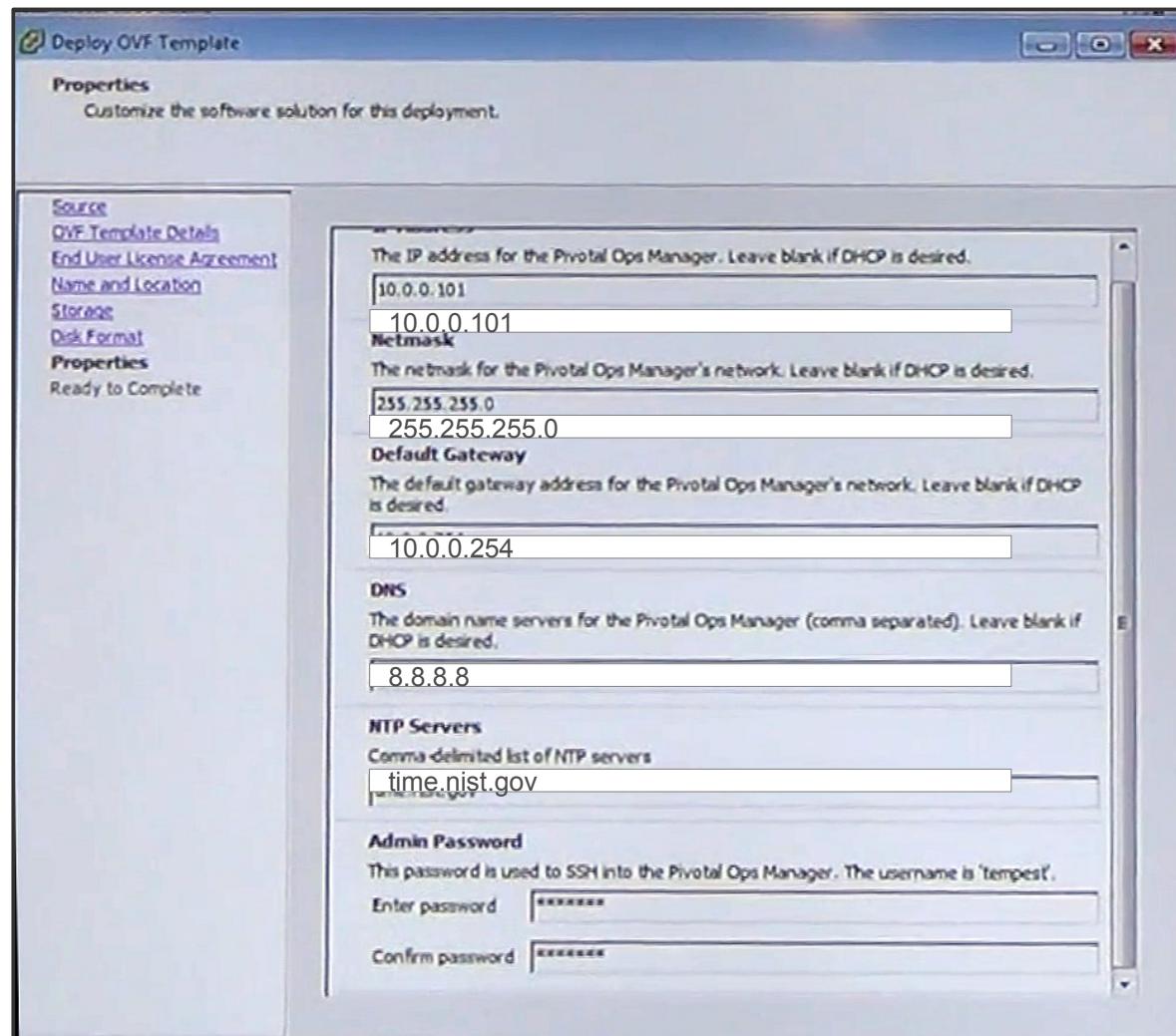
- Deploy OVA template
  - Use vSphere ... as you would to deploy any other VM
    - *Do not* use DHCP – requires a fixed IP address
  - Most important step, specify
    - IP Address of OPS Manager VM
    - Default gateway, DNS, NTP server
    - A username and password
      - allows *ssh* access to VM
      - for example to run BOSH commands
  - Deploy and start up the VM



*Next slide*

## 2b. Main Configuration Step

- A sys-admin/ops would *know* these values
  - IP address for Ops Mgr VM
  - Username and password to allow *ssh* into Ops Mgr VM



# Roadmap

- Prerequisites
- Configuration
  - Installing Pivotal CF Ops Manager VM
  - **Configuring Ops Manager Director**
  - Installing Elastic Runtime Tile
  - Installing Service Tile(s)

# What is *Ops Manager Director*?

- A Ruby on Rails web-application
  - Pre-installed on the Ops Mgr VM
  - Starts up whenever the VM is started
- Provides a GUI for configuring a Pivotal CF “foundation”
  - Behind the scenes it configures and runs *BOSH*

## BOSH



- An open-source deployment tool used to run up the VMs and processes that make up CF
- IaaS independent: runs on vCenter, AWS, vCloud Air ...
- Without Ops Mgr, OSS CF requires *manual* use of BOSH

# Step 3. Configure vSphere “Tile”

- Use browser to connect to IP address specified during OVA installation
  - Connects to Ops Manager Director (*a Rails web app*)
  - Asks you to setup an admin account: specify username and password you *won't forget*
- Most Pivotal CF Admin handled through Ops Manager
  - Not much need to use vSphere

## 3a. Open URL in Browser



Pivotal Ops Manager

admin

password

Confirm password

Create Admin User

Specify and confirm admin password

If you lose this password, you can *only* recover by reinstalling the VM

Pivotal Ops Manager v1.1.0.0 © 2014 GoPivotal Inc. All Rights Reserved

End User License Agreement

# Step 3. Configure vSphere “Tile”

- Installed features shown as “Tiles” in the main window
  - Color coding: **Orange**: not configured, **Blue**: configured, **Pivotal Green**: running
  - If anything goes wrong - click on Recent Install Logs to see what happened
  - There is a Revert option to return last good configuration
- First tile is always to configure Ops Mgr Director to use the underling IaaS
  - Here vCenter (Tiles exist for AWS, vCloud Air ...)

## 3b. Browser View

The screenshot shows the Pivotal Ops Manager Installation Dashboard. On the left, under 'Available Products', there is a card for 'Ops Manager Director for VMware vSphere' with version 'v1.1.0.0'. This card is highlighted with a yellow box labeled 'IaaS Tile'. In the top right corner, there is a 'Revert option' button. On the far right, there is a 'Pending Changes' section with a 'Revert' link, and below it, a 'Recent Install Logs' section with a 'Logs' link. A yellow box at the bottom right of the dashboard area contains the text 'Click here to see logs in case of errors'.

Pivotal Ops Manager

Available Products

Ops Manager Director for VMware vSphere  
No Upgrades Available.

Import a Product

Download Pivotal CF compatible products at Pivotal Network.

Installation Dashboard

Ops Manager Director for **vmware vSphere®**  
v1.1.0.0

IaaS Tile

Revert option

Pending Changes Revert

INSTALL Ops Manager Director for VMware vSphere

Install

Recent Install Logs

Click here to see logs in case of errors

Pivotal Ops Manager v3.0.0 © 2014 GoPivotal Inc. All Rights Reserved.

End User License Agreement

## 3c. Configure Ops Manager for vSphere

- vSphere knows about the Ops Manager VM
  - vSphere just installed and ran it
- But, *now* we need to tell Ops Manager about vSphere
  - Some configuration is green – defaults OK
  - Just need to setup the rest (orange)
- Also installs a single BOSH instance
  - Termed a *Micro-BOSH*
  - Enables Ops Manager to drive vSphere and create VMs
    - Uses BOSH to do all the hard work
    - Without Ops Manager, you'd have to use BOSH manually



# 3d. vCenter Credentials

The screenshot shows the PCF Ops Manager interface for configuring vCenter. On the left, a sidebar lists several options: vCenter Config (selected), Director Config, Create Availability Zones, Assign Availability Zones, Create Networks, Assign Networks, and Resource Config (which has a checked checkbox). A yellow callout box labeled "orange – yet to be done" points to the top right of the screen. The main form area is titled "vCenter Config". It contains fields for "vCenter IP Address\*" (set to "10.0.0.102"), "vCenter Username\*" (set to "root"), and "vCenter Password\*" (represented by five asterisks). Below these is a field for "Datacenter Name\*". A large yellow callout box labeled "vCenter IP Address" covers the "vCenter IP Address\*" field. Another yellow callout box on the right contains the text: "Real deployment: setup vCenter admin user with required access. For demos, just use root!".

PCF Ops Manager

admin ▾

Installation Dashboard

Ops Manager Director

Settings Status Credentials

vCenter Config

Director Config

Create Availability Zones

Assign Availability Zones

Create Networks

Assign Networks

Resource Config

orange – yet to be done

vCenter IP Address

vCenter Config

vCenter IP Address\*

10.0.0.102

The IP address of the vCenter that manages ESXI / vSphere

vCenter Username\*

root

vCenter Password\*

\*\*\*\*\*

Datacenter Name\*

Real deployment: setup vCenter admin user with required access.  
For demos, just use root!

green – defaults OK

PCF Ops Manager

ghts Reserved.

End User License Agreement

# 3e. Network Configuration

Installation Dashboard  
Ops Manager Director

Settings Status Credentials

vCenter Config Director Config Create Availability Zones Assign Availability Zones Create Networks Assign Networks Resource Config

green – now configured OK

Create Networks

Networks  
One or many IP ranges upon which your products will be deployed

LAN

Name\* LAN

vSphere Network Name\* LAN

Subnet (CIDR Range)\* 10.0.0.0/24

Excluded IP Ranges 10.0.0.1-10.0.0.102

DNS\* 10.111.111.222

Gateway\* 10.0.0.1

The network gateway IP to be used by VMs

Add

trash

Pivotal CF uses *all* addresses in subnet – *except* those excluded (already used by vSphere)

An ops person would know these values – because they setup The network originally

**No DHCP is used – VMs given explicit addresses by BOSH**

## 3g. Other Settings

- Availability Zones
  - Correspond to clusters in vCenter, supported directly in AWS, OpenStack
- Networks
  - Multiple networks possible
  - Define IP addresses available for use by CF VMs
- Resource sizes
  - Amount of resources available to Ops Manager
  - Increase in big system to deploy faster

# 3h. Do the Install!

- Click Install – takes 15-20 mins to run

The screenshot shows the PCF Ops Manager interface. On the left, there's a sidebar with 'Available Products' and 'Ops Manager Director' listed. The main area is the 'Installation Dashboard' for 'Ops Manager Director for VMware vSphere'. It shows the version 'v1.4.0.0'. To the right, there's a 'Pending Changes' section with a green checkmark next to 'INSTALL Ops Manager Director' and a large blue 'Apply changes' button. A red arrow points from a callout box at the bottom right towards the 'Apply changes' button. The callout box contains the text 'green – now configured OK'.

P CF Ops Manager admin ▾

Available Products

Ops Manager Director

No upgrades available

Import a Product

Download PCF compatible products at [Pivotal Network](#)

Installation Dashboard

Ops Manager Director for  
VMware vSphere®  
v1.4.0.0

Pending Changes Revert

INSTALLED Ops Manager Director

Apply changes

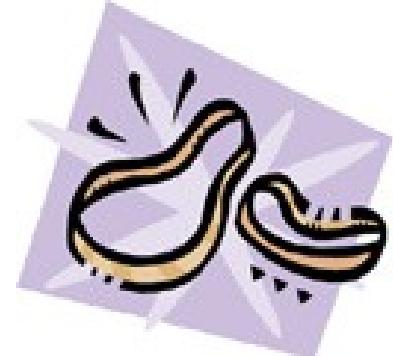
Recent Install Logs

green – now configured OK

# Roadmap

- Prerequisites
- Configuration
  - Installing Pivotal CF Ops Manager VM
  - Configuring Ops Manager Director
  - **Installing Elastic Runtime Tile**
  - Installing Service Tile(s)

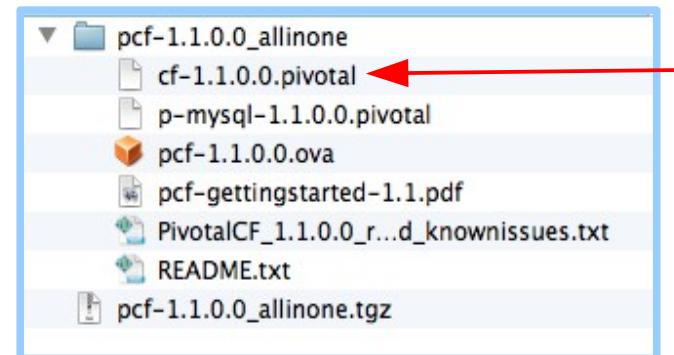
# Elastic Runtime



- A scalable runtime environment
  - This is what you actually deploy to
    - *To a developer, it **is** Cloud Foundry*
  - Uses the micro-BOSH to control the underlying IAAS
    - In our case: vSphere
- Operational control of Cloud Foundry is *really* management and configuration of the **Elastic Runtime**

# Step 4: Elastic Runtime Installation

- To install:
  - Click on “Import a Product” ①
  - Select “Elastic Runtime” = *cf-xxx.pivotal*
  - Click Add – Tile appears ②
  - Click on the *Elastic Runtime* tile to configure it



The screenshot shows the 'Available Products' section on the left and the 'Installation Dashboard' on the right.

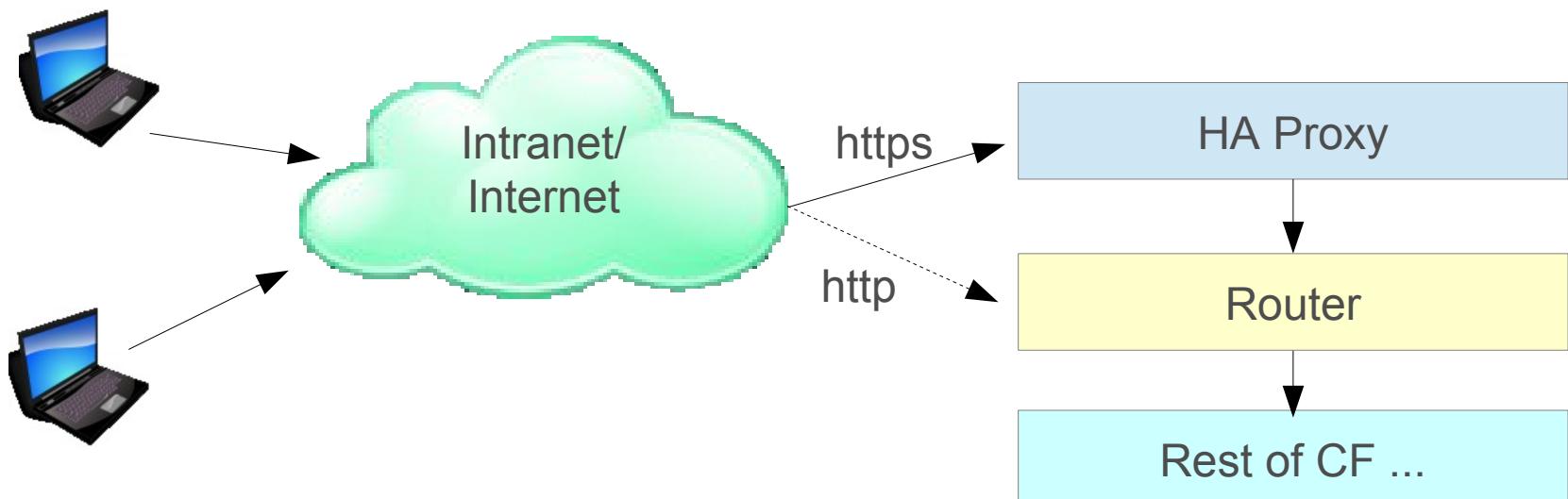
- Available Products:**
  - Ops Manager Director for VMware vSphere (No upgrades available)
  - Pivotal Elastic Runtime (No upgrades available)
    - Add »** button (circled with ②)
- Installation Dashboard:**
  - Operations Manager Director for **vmware vSphere** (v1.2.0.0)
    - Add »** button (circled with ①)

All services are added like this:  
MySQL, Rabbit/MQ ...  
– download *.pivotal* files from PivNet

Orange – not configured

# Accessing Your CF Instance

- Rest of company/outside world needs IP address to use
  - If public, setup DNS to point to that address
  - CF includes HA proxy load-balancer (or use your own)
    - HTTPS support (optional) via HA Proxy



# 4a. IPs and Ports – 1

PCF Ops Manager admin ▾

◀ Installation Dashboard

Pivotal Elastic Runtime

Settings Status Credentials Logs

Most options are green – ok defaults

Assign Networks

Assign Availability Zones

Root Filesystem

System Database Config

File Storage Config

IPs and Ports

MySQL Proxy Config

Cloud Controller

Router IPs

HAProxy IPs

Loggregator Port

10.0.0.121

Static address, visible to rest of company and/or outside world

Form continues on next slide ...

## 4a. IPs and Ports – 2

- Use real certificates if you have them, or generate self-signed

*... page continues from previous slide*

External Endpoints

SSO Config

LDAP Config

SMTP Config

Errands

Resource Config

Stemcell

SSL Certificate \*

```
-----BEGIN CERTIFICATE-----  
MIIDLjCCAhgAwIBAgIVAKYKJj8qMc5kyh3SEnqA  
Iet//ZeuMA0GCSqGSIb3DQEBr  
BQUAMD4XCZAJBgNVBAYTAIVTMRAwDgYDVQQK  
DAdQaXZvdGFsMR0wGwYDVQQDDBQa  
LJMweC5lZHUUucGl2b3Rhbc5pbzAeFw0xNTA2Mj  
AxMDI4MDMwMTExOvxtA3MtkvMDU4  
-----BEGIN RSA PRIVATE KEY-----  
MIIEpQIBAAKCAQEao+eEt2eakfyrlhurMgy3gre2a  
twmRDhi3Jur199/Q/PHqkvC  
nwuMMnzHSQDYzhBfxRZaaCvkRL2DDNp8-RJs1  
IA7AOcqKrkky-iBeTSBhDKP9tB  
BWZ0qGO0S5MCJ01Yb1C8ELLT1mzNPujA1VnK/  
6dcaadeRHcRqavuud1QWIMR5V6O  
Generate Self-Signed RSA Certificate
```

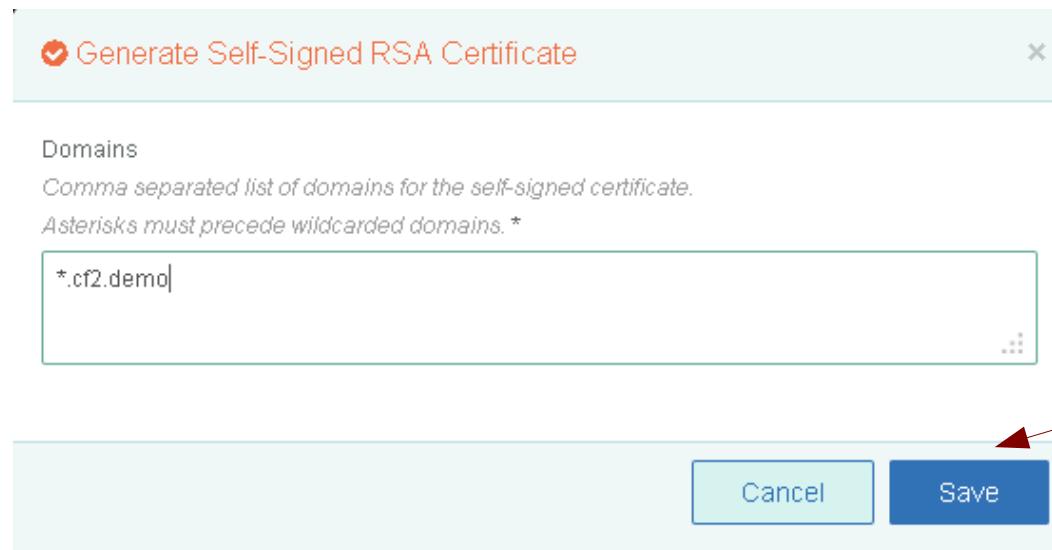
Trust Self-Signed Certificates

**Save**

**Click to auto-generate self-signed certs**

## 4b. Self-signed Certificates

- In popup window ...
  - Certificates should be generated for \*.[your-domain]



Click “Save” to  
populate  
(previous  
slide )

## 4c. Specify Domains

- Cloud Controller Tab
  - System and App domains
  - Same used for the HAProxy certificates, without the \* wildcard

- ✓ Assign Networks
- ✓ Assign Availability Zones
- ✓ Root Filesystem
- ✓ System Database Config
- ✓ File Storage Config
- ✓ IPs and Ports
- ✓ MySQL Proxy Config
- Cloud Controller

Coordinates Pivotal CF Elastic Runtime ap

System Domain \*

cf2.demo

Apps Domain \*

cf2.demo

Cloud Controller DB Encryption Key

Secret

Maximum File Upload Size (MB) ( min: 1024, max: 2048 ) \*

1024

- System domain is used to target and push apps to Elastic Runtime
  - In this case: `cf login -a api.cf2demo`
- Application domain is used to serve applications
  - `cf push spring-music` would create app at *spring-music.cf2.demo*

## 4d. Many Other Options

- Additional items
  - Networks, High Availability Zones
    - If you have a choice, tell ER which ones to use
  - SSO, LDAP
    - Used with SSO appliances and Active Directory integration
  - Mail Setup
    - Optional SMTP/email configuration to send email to users
      - for example: invites to use system, error/system messages
  - Errands
    - Tasks that run at end of Install – including creating the App Manager console

## 4e. Resource Sizing

- Configure how many instances of each component should be deployed
  - Can normally accept the defaults
- **Important:**
  - This Ops Manager console is still available *after* CF is installed
    - Normal way for ops to expand/configure system
    - Next slide ...

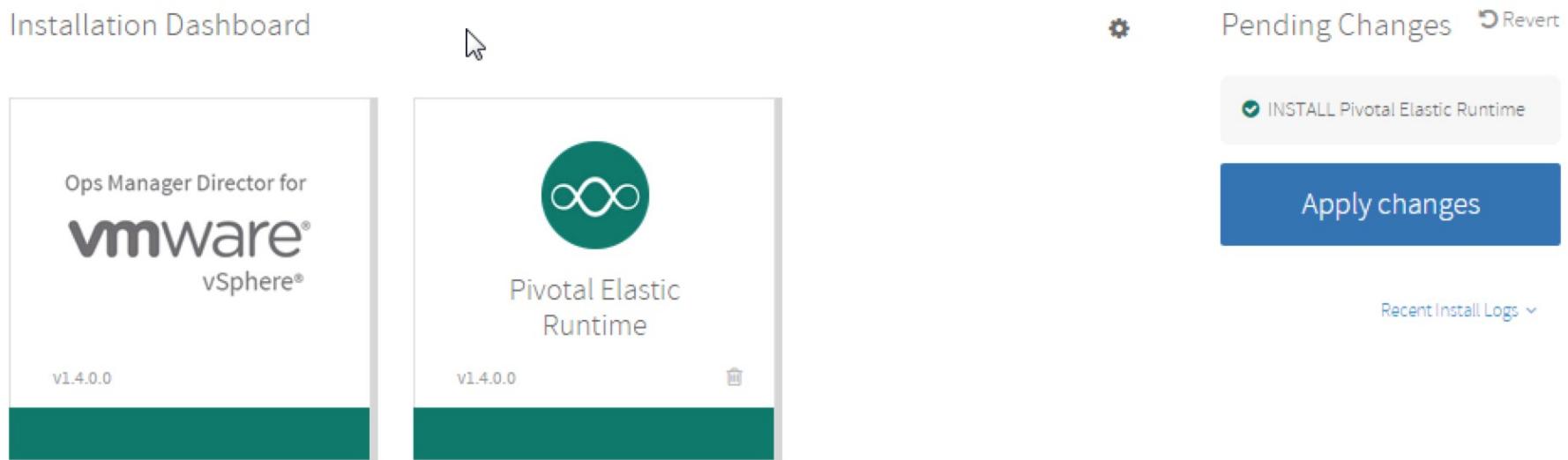
## Resource Config

---

JOB	INSTANCES	CPU	RAM (MB)	EPHEMERAL DISK (MB)	PERSISTENT DISK (MB)
NATS	1	1	1024	2048	0
etcd	1	1	1024	2048	1024
NFS Server	1	1	1024	2048	102400
Cloud Controller Database	1	1	1024	2048	2048
UAA Database	1	1	1024	2048	8192
Console Database	1	1	1024	2048	1024
Cloud Controller	1	1	4096	20480	0
HAProxy	1	1	1024	2048	0
Router	1	1	1024	2048	0
Health Manager	1	1	1024	2048	0
Clock Global	1	1	1024	2048	0
Cloud Controller Worker	1	1	1024	2048	0
Collector	0	1	1024	2048	0
UAA	1	1	1024	2048	0
Login	1	1	1024	2048	0
MySQL Proxy	1	1	1024	2048	0
MySQL Server	1	2	8192	30000	100000
DEA	1	2	16384	32768	0

## 4f. Install Elastic Runtime

- Click “Apply Changes”
  - This takes a while, up to 3 hours



Pivotal™

# 4f. Installation Takes Time!

**P** PIVOTAL  
**Operations Manager**

Installation Dashboard

## Installation in Progress

4%

- Setting Micro BOSH deployment manifest
- Installing Micro BOSH**
- Checking Micro BOSH status
- Logging into director
- Creating director user
- Removing director bootstrap user
- Targeting director for Pivotal Elastic Runtime
- Logging into director for Pivotal Elastic Runtime
- Uploading release for Pivotal Elastic Runtime
- Uploading stemcell for Pivotal Elastic Runtime
- Setting installation manifest for Pivotal Elastic Runtime

- Many steps
- Many VMs created

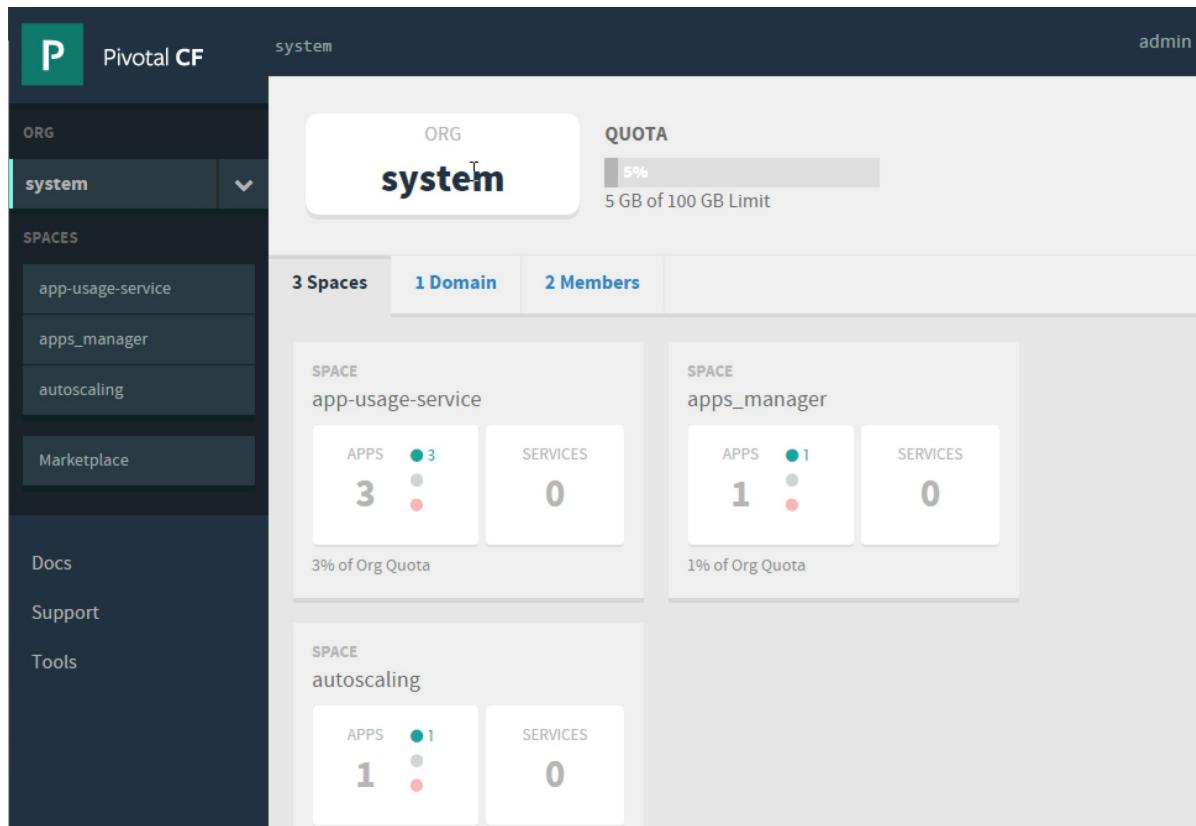
- Installing Pivotal Elastic Runtime
- Checking Pivotal Elastic Runtime
- Verifying app push
- Pushing Console app
- Verifying Console app
- Targeting director for Pivotal MySQL Dev
- Logging into director for Pivotal MySQL Dev
- Uploading release for Pivotal MySQL Dev
- Uploading stemcell for Pivotal MySQL Dev
- Setting installation manifest for Pivotal MySQL Dev
- Installing Pivotal MySQL Dev
- Checking Pivotal MySQL Dev
- Verifying MySQL service
- Storing settings in file system

Each item listed is a *BOSH task*

# Checking Installation

## Login to Pivotal CF App Manager Console

- Navigate to [http://console.<your\\_domain>](http://console.<your_domain>)



# Installing Services

- Many pre-packaged services for Pivotal CF
  - Supplied as **.pivotal** files at Pivotal Network
    - See <https://network.pivotal.io>

 <p><b>MongoDB for Pivotal CF</b> MongoDB Data Store</p>	 <p><b>MySQL for Pivotal CF</b> MySQL database-as-a-service for Cloud Foundry applications</p>	 <p><b>Pivotal CF</b> Rapidly deploy and scale applications on private clouds W...</p>
 <p><b>Pivotal CF RabbitMQ Service</b> Give your applications a common platform to safely</p>	 <p><b>Pivotal HD for Pivotal CF®</b> Gain insight from the massive data captured by apps, syst...</p>	 <p><b>Redis for Pivotal CF</b> Cloud Foundry Redis service for application development a...</p>

# Installing Services – MySQL



- For example: to install MySQL
  - Go back to Operations Manager home-page
  - Click “Add New Product”
  - Select MySQL
    - `p-mysql-1.1.0.0.pivot al` (your version may be different)
  - Appears as new Tile
  - Review options and install
    - Just like we did for Elastic Runtime
  - MySQL now appears as a service in the Pivotal CF Marketplace



# Summary

- After completing this lesson, you should have learnt:
  - How to setup Operations Manager VM
  - How to Install the CF Elastic Runtime
  - How to install other services

# Spring Introduction

Spring Background for the Course

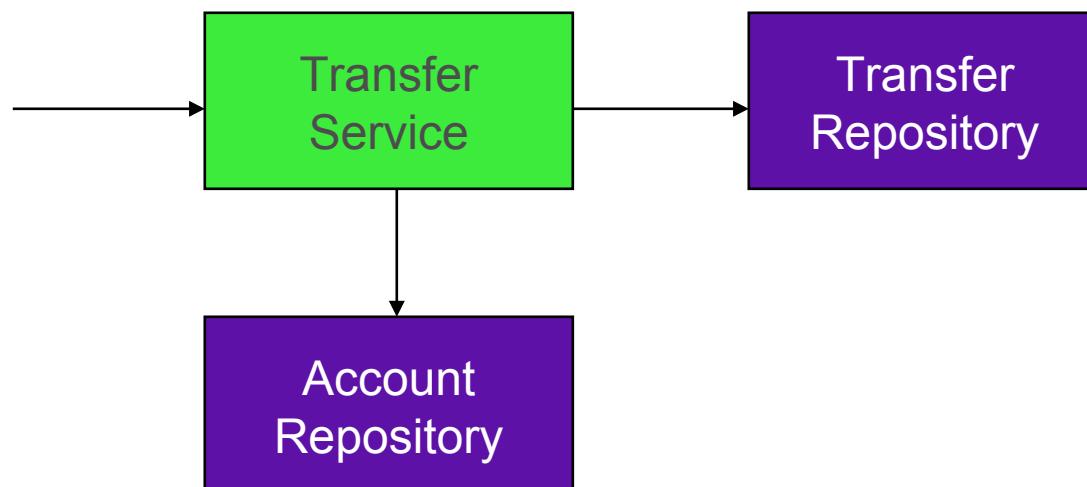
Spring Configuration, Dependency Injection,  
Bean Creation and Proxies

# Topics in this session

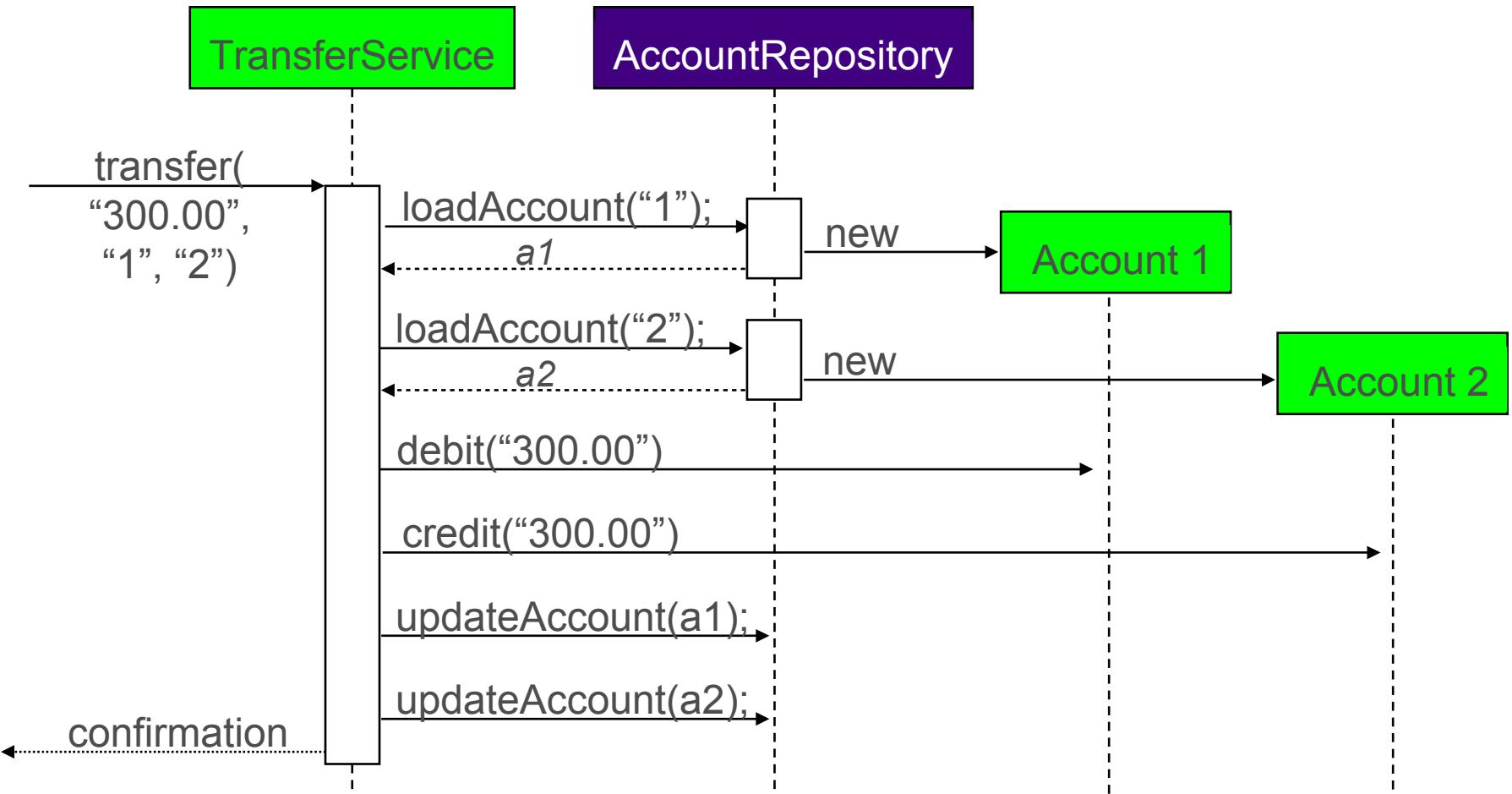
- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- Bean Creation

# Application Configuration

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



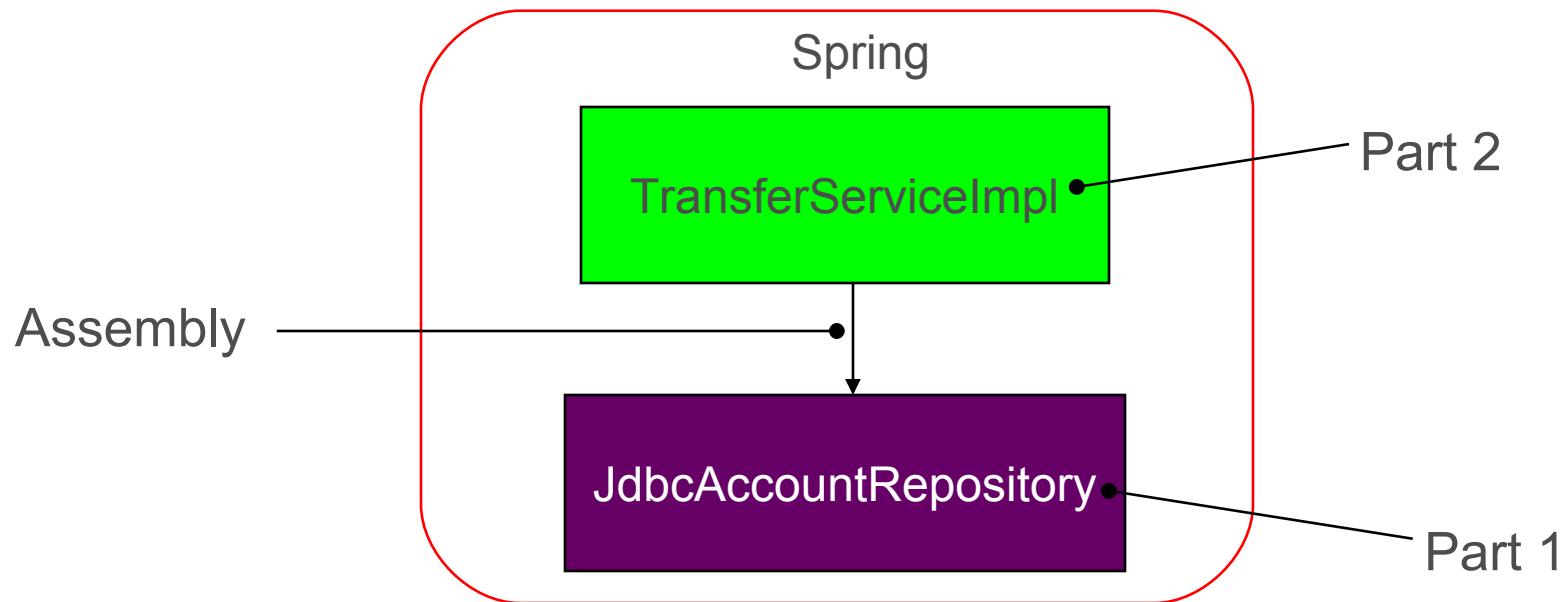
# Example: Money Transfer System



# Spring's Configuration Support

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

# Money Transfer System Assembly



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

# Parts are Just Plain Old Java Objects

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

Implements a service/business interface

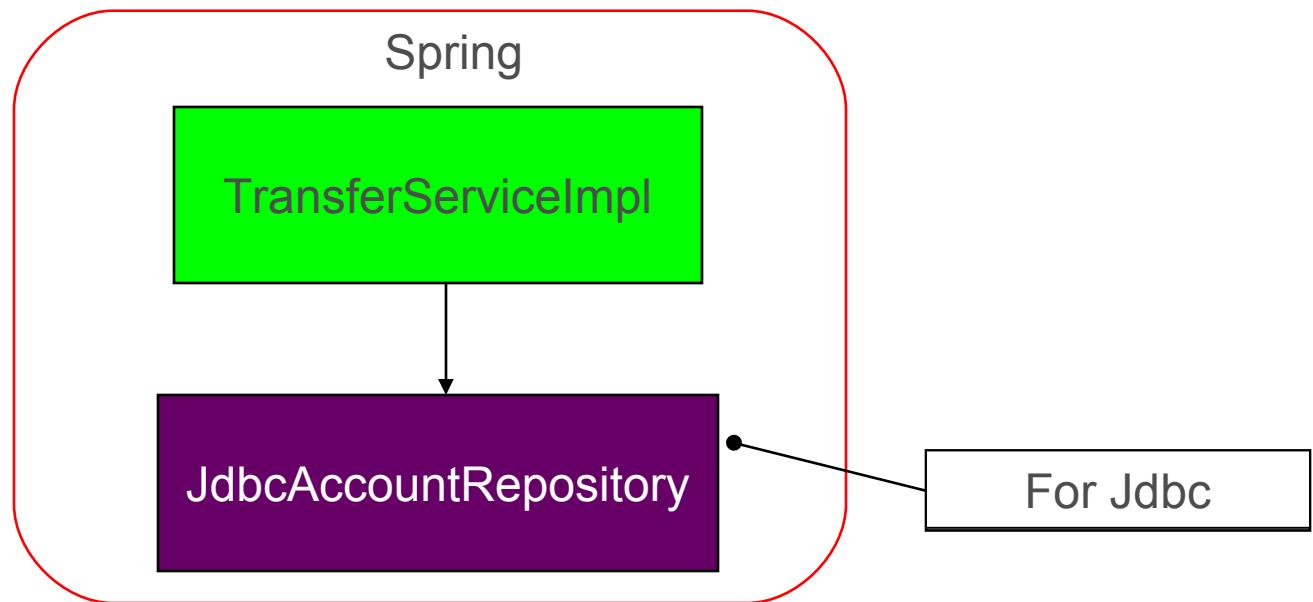
Part 1

```
public class TransferServiceImpl implements TransferService {  
    private AccountRepository accountRepository;  
  
    public void setAccountRepository(AccountRepository ar) {  
        accountRepository = ar;  
    }  
    ...  
}
```

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

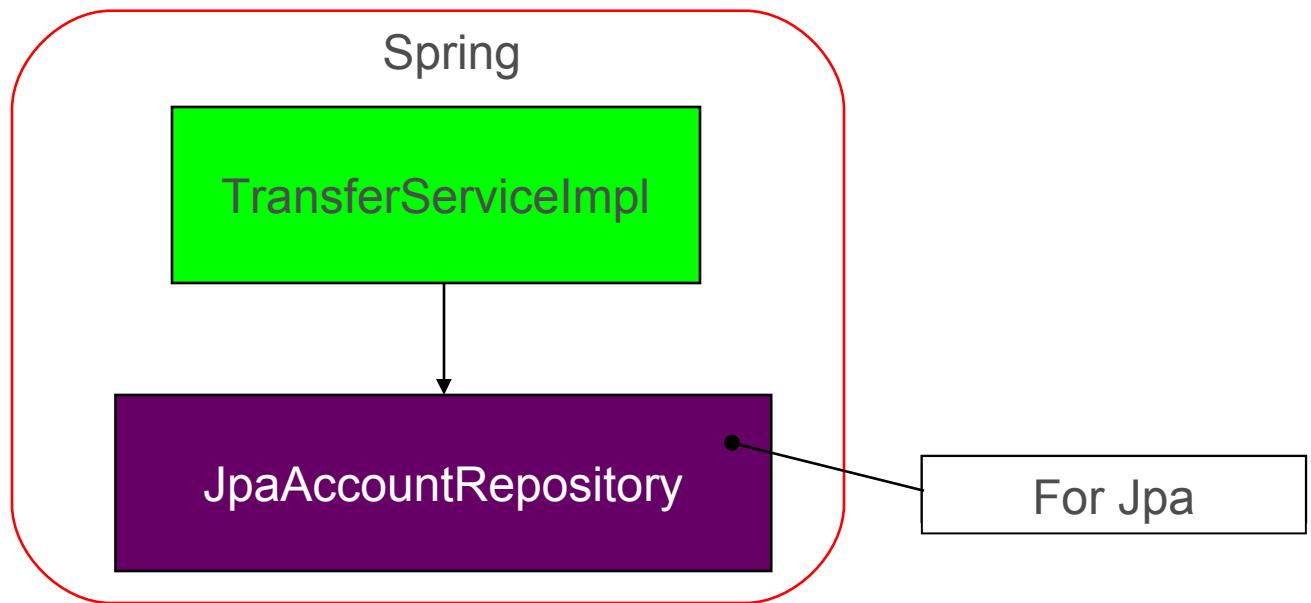
Part 2

# Swapping Out Part Implementations



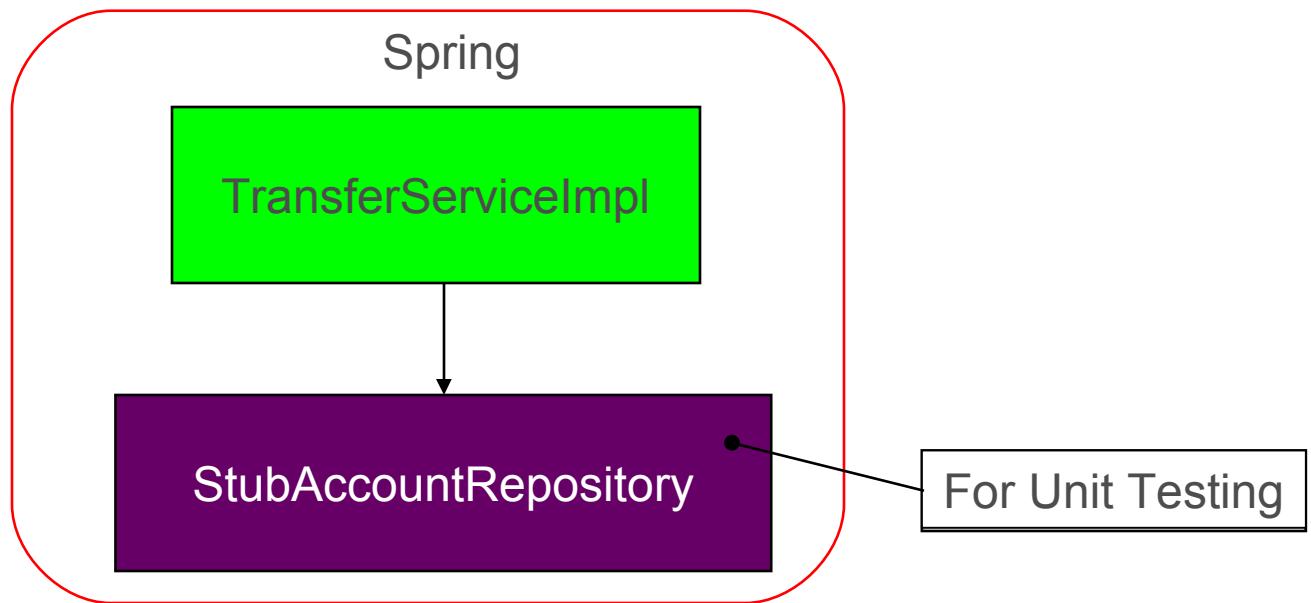
```
(1) new JdbcAccountRepository(...);  
(2) new TransferServiceImpl();  
(3) service.setAccountRepository(repository);
```

# Swapping Out Part Implementations



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

# Swapping Out Part Implementations

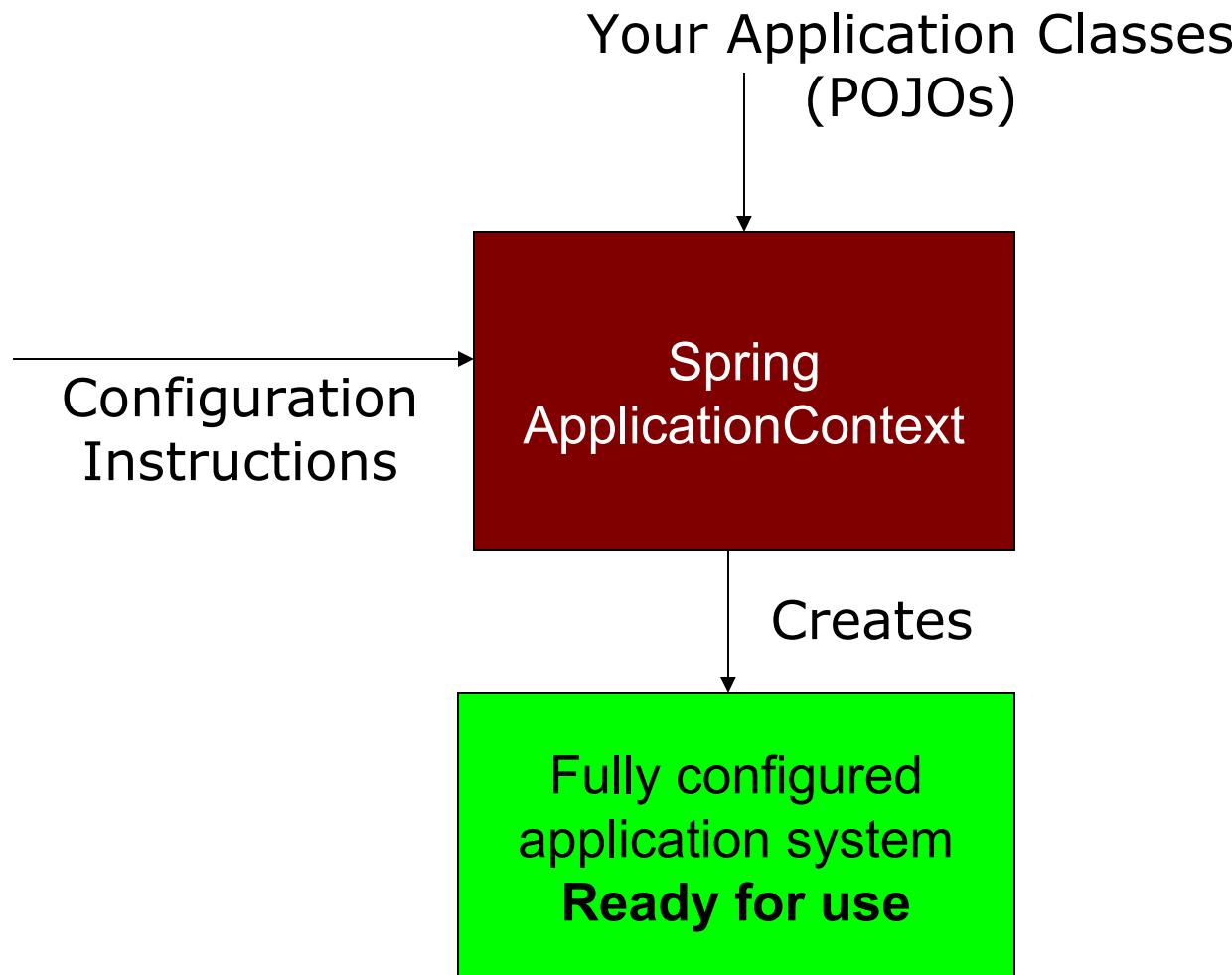


```
(1) new StubAccountRepository();
(2) new TransferServiceImpl();
(3) service.setAccountRepository(repository);
```

# Topics in this session

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

# How Spring Works



# Your Application Classes

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



Needed to perform money transfers between accounts

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



Needed to load accounts from the database

# Configuration Instructions – Java

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

**Dependency injection**

**Dependency injection**

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

# Configuration Instructions – XML

```
<beans>
```

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

Dependency injection

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

Dependency injection

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

Bean ID specified explicitly via id attribute

# Implicit Configuration using Annotations

- Annotation-based configuration *within* bean-class

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

Annotations embedded *within* POJOs

Bean ID

Dependency injection

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

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

Find @Component classes within designated (sub)packages

# Creating and Using the Application

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

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

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

Bean ID

**NOTE:** This code uses Spring Boot's *SpringApplication.run()*  
You may be used to seeing older code like this:

```
context = new ClassPathXmlApplicationContext("beans.xml");
```

# Topics in this session

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

# What Problem Does AOP Solve?

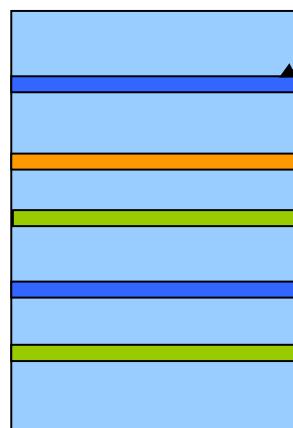
- Aspect-Oriented Programming (AOP) enables modularization of cross-cutting concerns
  - Generic functionality needed in many places in your application
- Examples
  - Logging and Tracing
  - Transaction Management
  - Security, Caching, Error Handling
  - Performance Monitoring
  - Custom Business Rules

# System Evolution Without Modularization

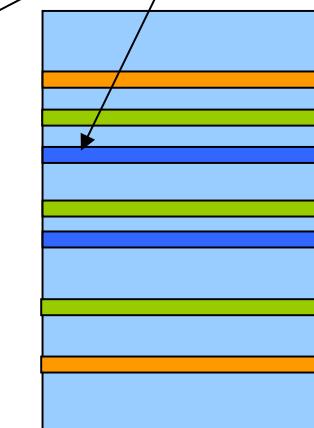
- Security
- Transactions
- Logging

Code scattering  
(duplication)

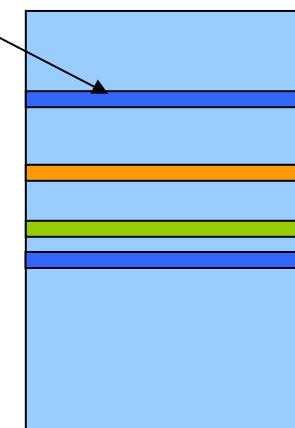
*You cut and paste  
(duplicate) the  
cross-cutting code  
everywhere – hard  
to maintain*



**BankService**

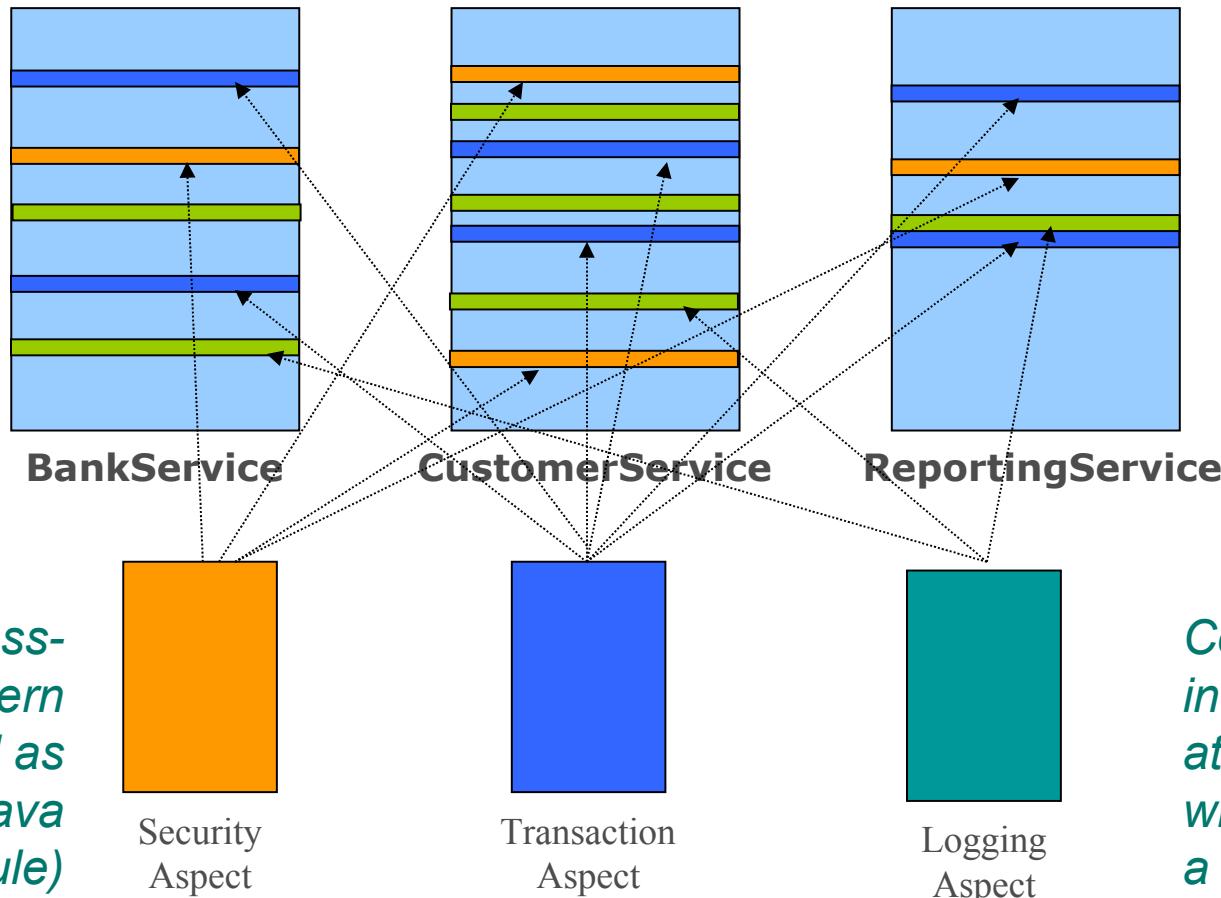


**CustomerService**



**ReportingService**

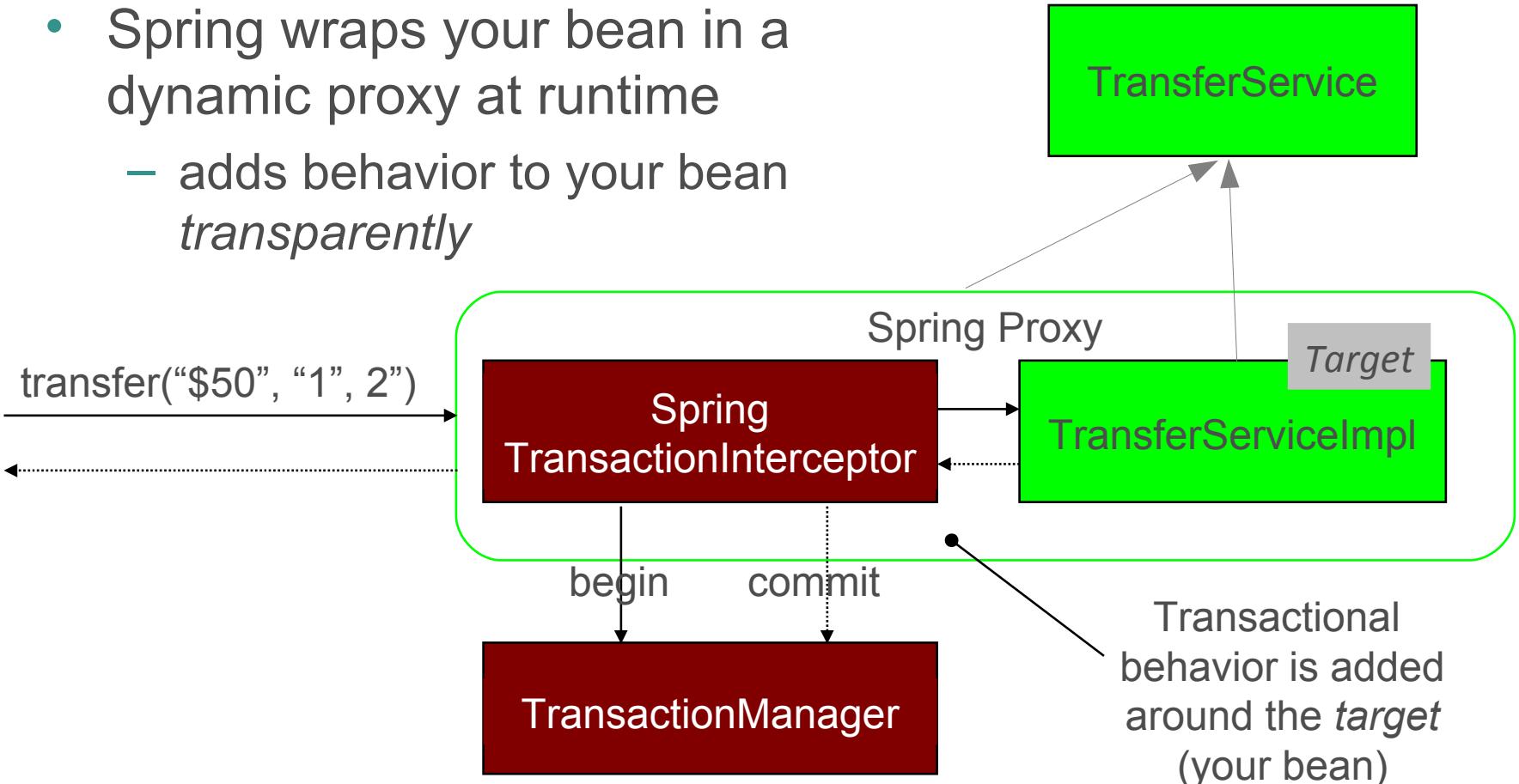
# System Evolution: AOP based



# Aspects Implemented Using a Proxy

## Transactional Example

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



# Kinds of Proxies

- Spring will create either JDK or CGLib proxies

## JDK Proxy

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

## CGLib Proxy

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

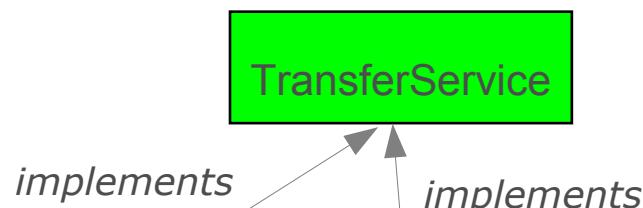


Recommendation: Code to interfaces / Use JDK proxies (default)

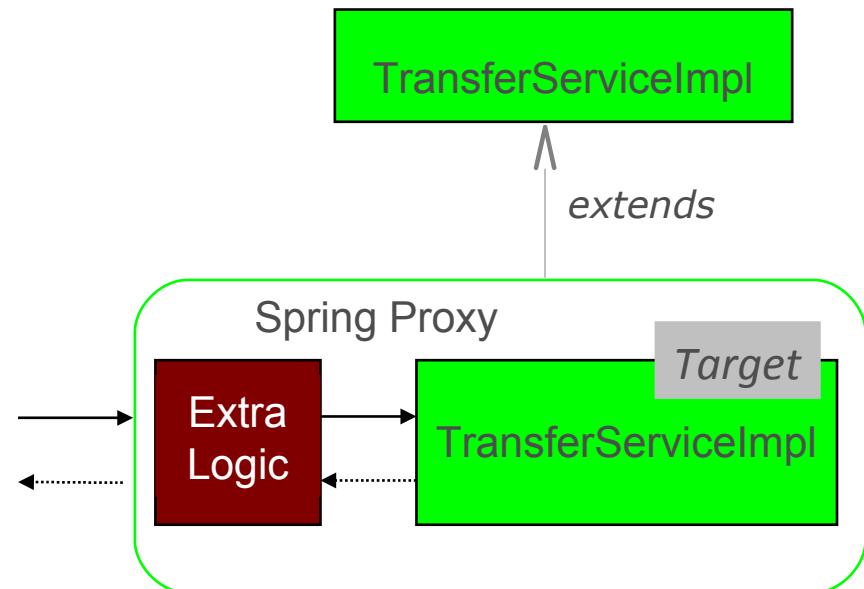
**See Spring Reference - 10.5.3 JDK- and CGLIB-based proxies**

# JDK vs CGLib Proxies

- JDK Proxy
  - Interface based



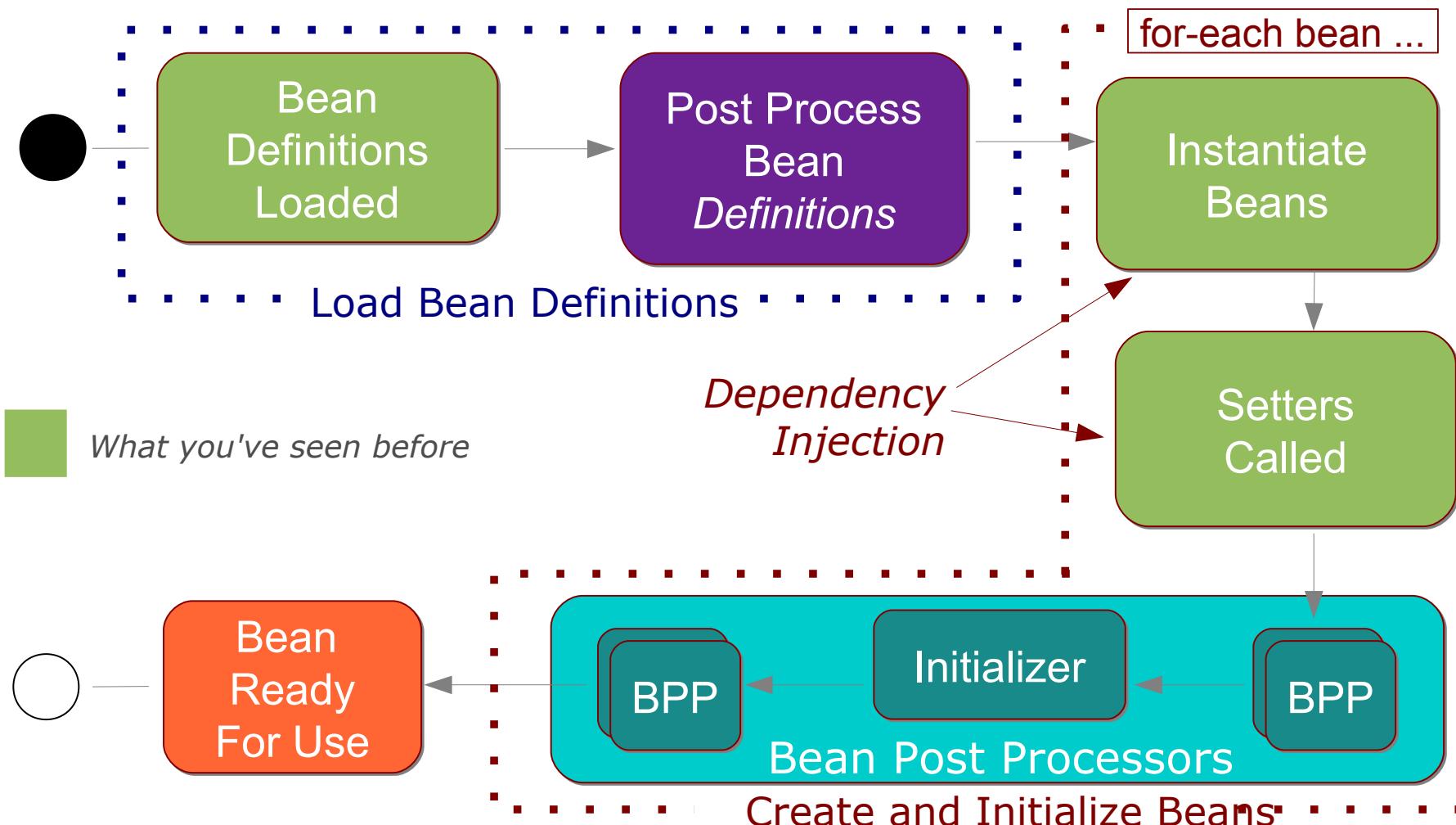
- CGLib Proxy
  - subclass based



# Topics in this session

- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- **Bean Creation**
  - **Bean Definition Post Processors**
  - Bean Post Processors

# Bean Initialization Steps



# BeanFactoryPostProcessor Extension Point

- Applies transformations to bean *definitions*
  - Before objects are actually created
- Several useful implementations provided in Spring
  - You can write your own (not common)
  - Implement **BeanFactoryPostProcessor** interface

```
public interface BeanFactoryPostProcessor {  
    public void postProcessBeanFactory  
        (ConfigurableListableBeanFactory beanFactory);  
}
```

# Most Common Example of BeanFactoryPostProcessor

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

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

Defines *PropertySourcesPlaceholderConfigurer*



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

Expands \$ variables  
from properties files



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

# Equivalent in Java Config

```
@Configuration  
@PropertySource ( "classpath:db-config.properties" )
```

```
public class ApplicationConfig {
```

```
    @Bean
```

```
    public DataSource dataSource(
```

```
        @Value("${db.driver}") String dbDriver,
```

```
        @Value("${db.url}") String dbUrl,
```

```
        @Value("${db.user}") String dbUser,
```

```
        @Value("${db.password}") String dbPassword) {
```

```
    DataSource ds = new BasicDataSource();
```

```
    ds.setDriverClassName( dbDriver);
```

```
    ds.setUrl( dbUrl);
```

```
    ds.setUser( dbUser);
```

```
    ds.setPassword( dbPassword ));
```

```
    return ds;
```

```
}
```

```
...
```

Requires a *PropertySourcesPlaceholderConfigurer*

# Loading Property Sources

- Property sources are loaded by a dedicated Spring bean
  - The `PropertySourcesPlaceholderConfigurer`
  - **Note: this is a *static* bean**
    - Such beans are created *first*
    - Ensures property-sources are read *before* any `@Configuration` bean using `@Value` is initialized

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

**@PropertySource *ignored unless* this bean declared**

# Spring Expression Language

## Annotation or Java Config Examples

```
@Repository  
public class RewardsDatabase {  
  
    @Value("#{systemProperties.databaseName}")  
    public void setDatabaseName(String dbName) { ... }  
  
    @Value("#{strategyBean.databaseKeyGenerator}")  
    public void setKeyGenerator(KeyGenerator kg) { ... }  
}
```

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



# Spring Expression Language

## XML Examples

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

Can refer a nested property

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

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

Equivalent to System.getProperty(...)



# Java Buildpack Auto-Configuration

- During staging of a Java Application
  - Buildpack scans Spring configuration
  - Looks for beans that define known services
    - Such as an RDBMS data-source
    - Or a Rabbit/MQ connection-factory
  - Swaps them for equivalent services from CF
    - Uses connection info from **VCAP\_SERVICES**
- How?
  - Using a custom **BeanFactoryPostProcessor**

# Topics in this session

- Why Spring?
- Configuration using Spring
- Aspects and Proxies
- **Bean Lifecycle**
  - Bean Definition Post Processors
  - **Bean Post Processors**

# Creating Bean Instances

- Each bean is eagerly instantiated by default
  - Created in right order with its dependencies injected
- After dependency injection each bean goes through a **post-processing** phase
  - Further configuration and initialization may occur
  - Initialization is a common special-case of a Bean Post Processor (BPP)



# Initializer BPPs



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

*By Annotation*

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

*Using XML*

```
@ComponentScan
```

```
<context:annotation-config/> OR  
<context:component-scan ... />
```

Declares several BPPs *including CommonAnnotationBeanPostProcessor*  
– which invokes the init-methods and/or post-construct methods

# The *BeanPostProcessor* Extension Point

- An important extension point in Spring
  - Can modify bean instances *in any way*
  - **Powerful enabling feature**
  - Spring provides many BPPs out-of-the-box



# BPPs Enable Proxies

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

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

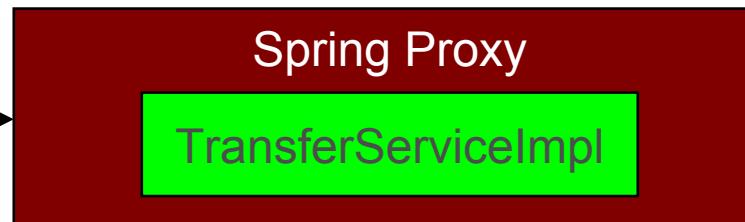
- Sometimes the bean is just your raw object

```
service.transfer("$50", "1", 2")
```



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

```
service.transfer("$50", "1", 2")
```



# Topics Covered in This Session

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