

# Pivotal Cloud Platform Deep Dive

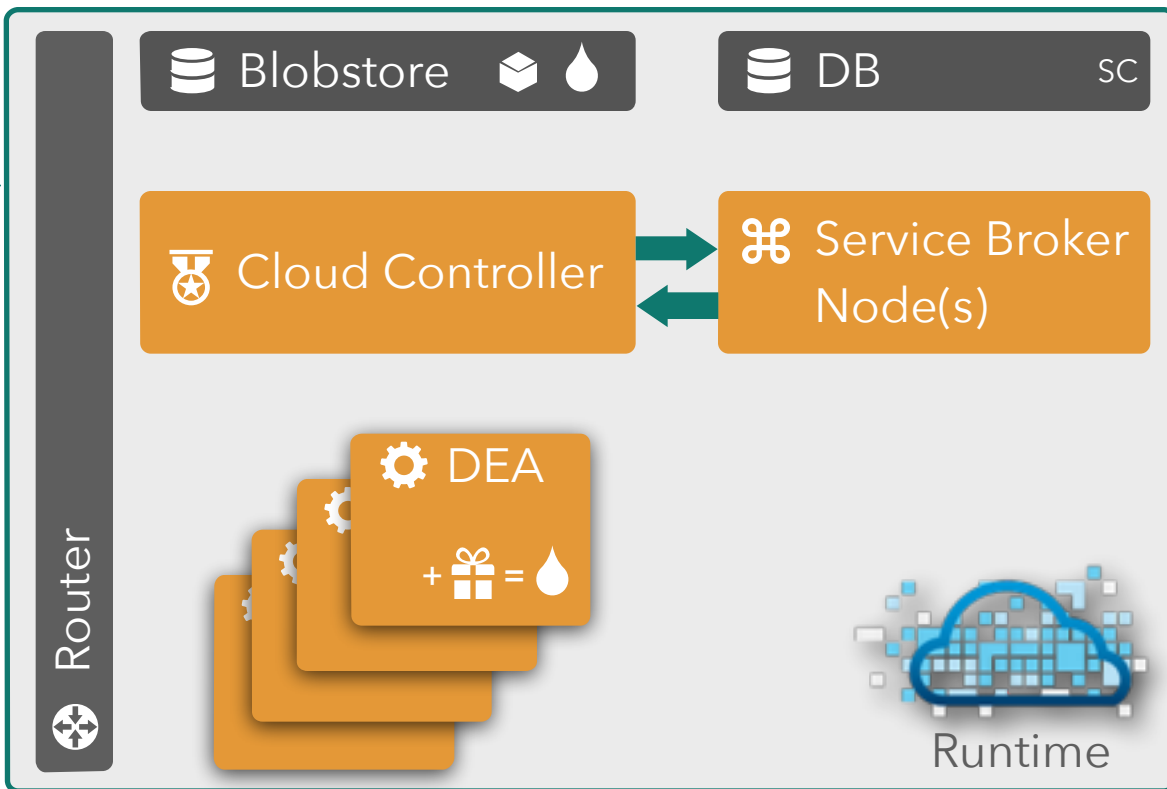
## Part 3: Custom Buildpacks and Data Services

Pivotal CF Team

# Buildpacks and Services

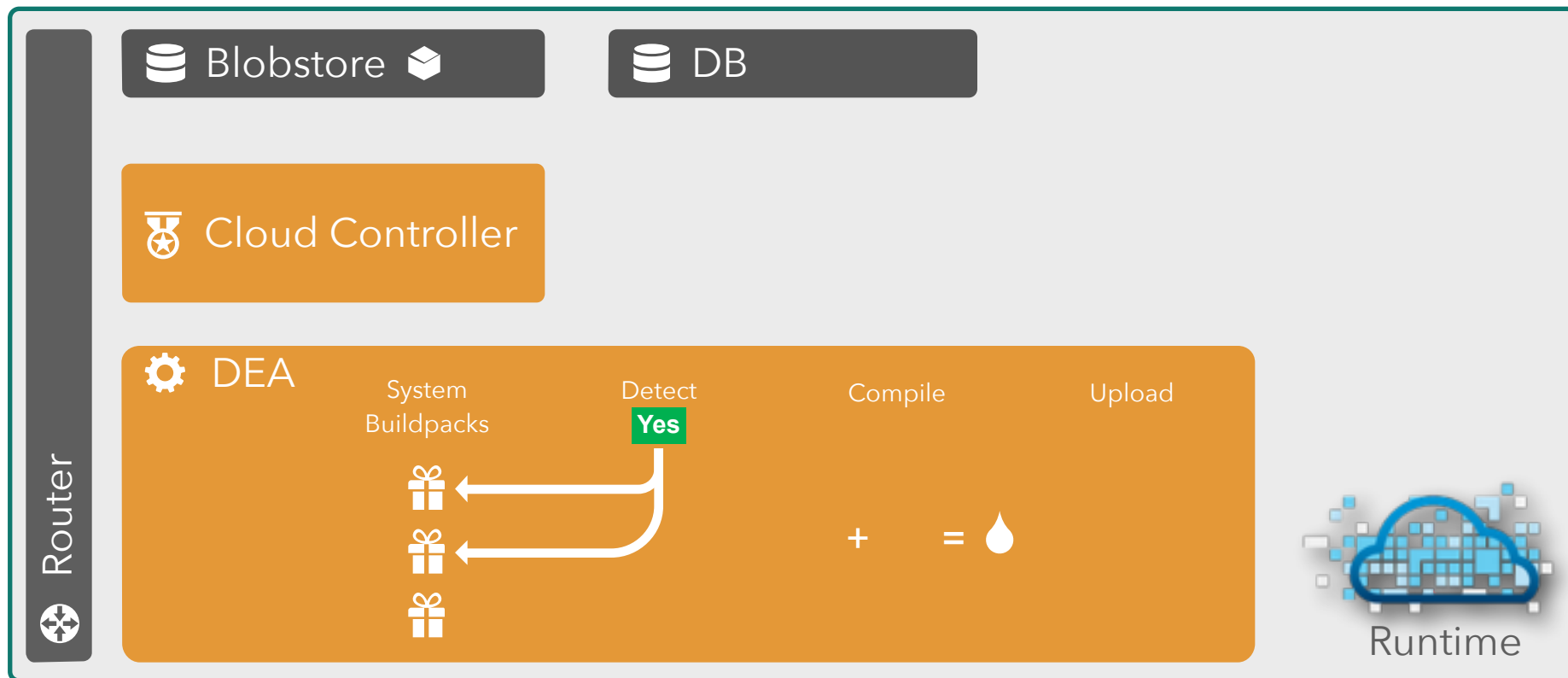
- Buildpacks
- Java Buildpack Deep Dive
- User-Provided Services
- Managed Services

# REVIEW: Deploying Applications to Cloud Foundry Runtime



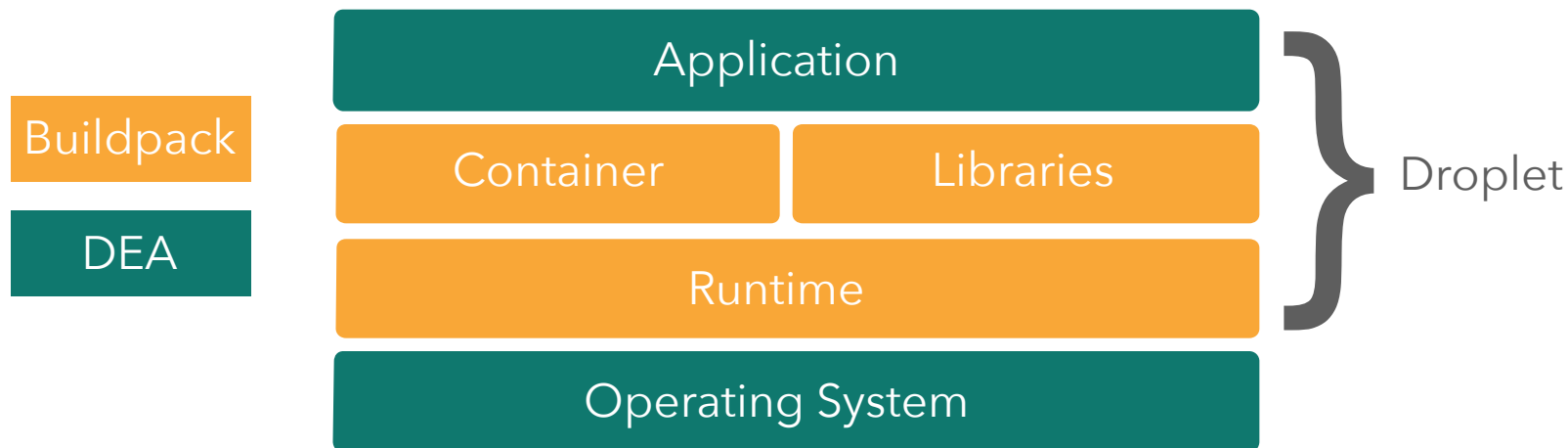
1. Upload bits/metadata
2. Create/bind services
3. Stage app
4. Deploy app

# REVIEW: Stage an App



# Staging and Buildpacks

Buildpacks are responsible for preparing the machine image for an application.



# Types of Buildpacks

## System

- deployed with Cloud Foundry

## Admin

- uploaded to Cloud Foundry

## BYO

- specified at app push

# Managing Admin Buildpacks

```
$ cf buildpacks
$ cf create-buildpack <name> <path to bits> <position>
$ cf update-buildpack <name> [-p <path>] [-i <position>]
$ cf delete-buildpack <name>
```

# Buildpack selection

```
$ cf push
```

The application is tested against admin then system buildpacks.

```
$ cf push -b  
  <url>
```

The buildpack is referenced by a Git URL



```
$ cf push -b  
  <name>
```

The admin buildpack is referenced by name



# Tested Buildpacks

<https://github.com/cloudfoundry-community/cf-docs-contrib/wiki/Buildpacks>

## Containers



## Languages



# Buildpack API

**/bin/detect** app\_directory

Inspect app bits to determine buildpack applicability

**/bin/compile** app\_directory cache\_directory

Download and install runtime, container, packages, libraries;  
install app bits as necessary



**/bin/release** app\_directory

Build app start command



# /bin/detect

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

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

On match, return exit code 0 and write to STDOUT a string identifying the buildpack (often just the name of the language supported)

# /bin/detect

```
$ cf push
```

DEA iterates over admin and  
system buildpacks calling  
/bin/detect scripts  
until one of them returns exit  
code 0

```
$ cf push -b <url|name>
```

/bin/detect is  
not called

# /bin/compile

Download and install any necessary

runtime (Java VM, Ruby interpreter, JavaScript interpreter)

container (web server)

support libraries, packages, modules (Ruby gems, NPM packages)

... and then installing the app bits into the runtime or container

# /bin/compile Caching

Runtime, container, and support packages are downloaded from sources external to Cloud Foundry

DEA provides a location for storing downloaded artifacts to speed subsequent staging operations

# /bin/release

Build a YAML-formatted hash with three possible keys

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

On Cloud Foundry, currently only the web: value is used to get the start command for the app

# Buildpacks and Services

- ~~Buildpacks~~
- Java Buildpack Deep Dive
- User-Provided Services
- Managed Services



# 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

Java main()  
Tomcat  
Groovy  
Spring Boot CLI  
Play

## Frameworks

Spring config  
Play config  
Play JPA config  
New Relic agent  
AppDynamics agent

## JREs

OpenJDK



# Container Detection Criteria

Java <code>main()</code>	<code>META-INF/MANIFEST.MF</code> exists with <code>Main-class</code> attribute set
--------------------------	---

Tomcat	<code>WEB-INF</code> directory exists
--------	---------------------------------------

Groovy	<code>.groovy</code> file with a <code>main()</code> method, or <code>.groovy</code> file with no classes, or <code>.groovy</code> file with a shebang ( <code>#!</code> ) declaration
--------	--

Spring Boot CLI	one or more POGO <code>.groovy</code> files with no <code>main()</code> method, and no <code>WEB-INF</code> directory
-----------------	--

Spring Boot Embedded	<code>start</code> script and <code>lib/spring-boot-*.jar</code> exist
----------------------	--

Play	<code>start</code> script and <code>lib/play.play_*.jar</code> exist
------	--

Ratpack	<code>start</code> script and <code>ratpack-core-*.jar</code> exist
---------	---

Choose zero or one



# Framework Detection Criteria

Spring

`spring-core*.jar` exists

Play config

Play application detected

Play JPA config

play-java-jpa plugin exists in app

Spring Insight

Insight service bound to app

New Relic agent

New Relic service bound to app

AppDynamics agent

AppDynamics service bound to app

Choose all that apply

Pivotal™



# /bin/compile Output Example

```
-----> Downloaded app package (18M)
-----> Downloading OpenJDK 1.7.0_21 JRE (17.5s)
      Expanding JRE to .java (1.4s)
-----> Downloading Auto Reconfiguration 0.7.1 (1.4s)
      Modifying /WEB-INF/web.xml for Auto Reconfig
-----> Downloading Tomcat 7.0.42 (3.5s)
      Expanding Tomcat to .tomcat (0.2s)
      Downloading Buildpack Tomcat Support 1.1.1 (0.0s)
-----> Uploading droplet (55M)
```

[DEA

[Buildpack

[DEA

# See What's Going On

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

**.buildpack-diagnostics/**

**.java/** — [ Buildpack-installed runtime

**.lib/** — [ Buildpack-installed support libraries

**.tomcat/** — [ Buildpack-installed container

META-INF/

WEB-INF/ — [ DEA-downloaded application files

assets/

# See What's Going On

```
$ cf files <app-name> staging_info.yml
```

```
detected_buildpack: openjdk=1.7.0_45 tomcat=7.0.47  
spring-auto-reconfiguration=0.7.2  
tomcat-buildpack-support=1.1.1
```

```
start_command: JAVA_HOME=.java
```

```
JAVA_OPTS="-Dhttp.port=$PORT
```

```
-Djava.io.tmpdir=$TMPDIR -XX:MaxPermSize=52428K
```

```
-XX:OnOutOfMemoryError=./.buildpack-diagnostics/killjava
```

```
-Xmx384M -Xss1M" .tomcat/bin/catalina.sh run
```



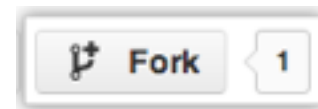
# Customization



Two ways to customize the 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



# Customization by Extension

Implement a JRE, Container, or Framework support class as one Ruby file in the appropriate directory

```
cloudfoundry/java-buildpack/lib/java_buildpack
```

```
└─ jre
```

```
└─ container
```

```
└─ framework
```

(with additional support classes as necessary)

# Use Cases: User Provided Service Instances

- Typically legacy or **existing instances of a service** (databases, queues, mail, etc) where applications connect to the **same instance**
- **Credential passing** when you need to inject the same credential set into an application

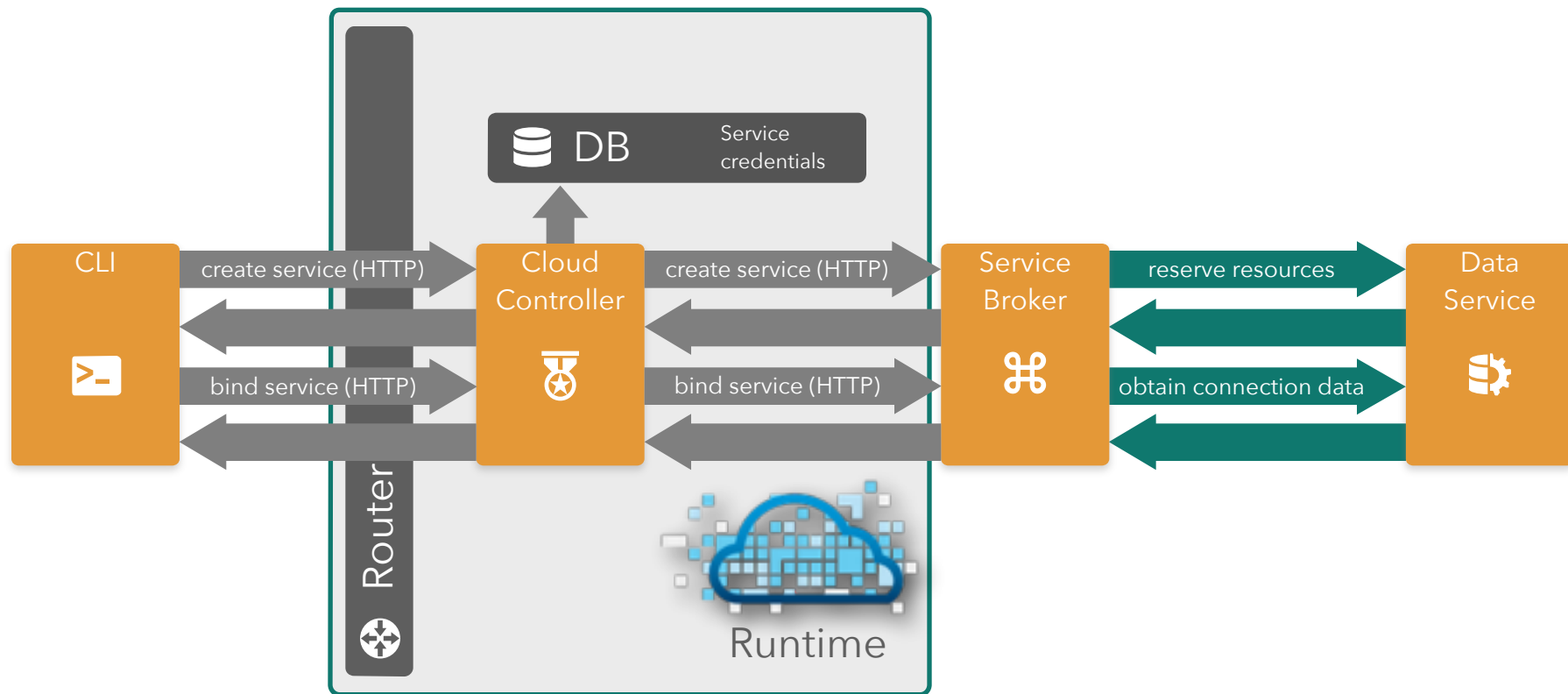
# Managed Services

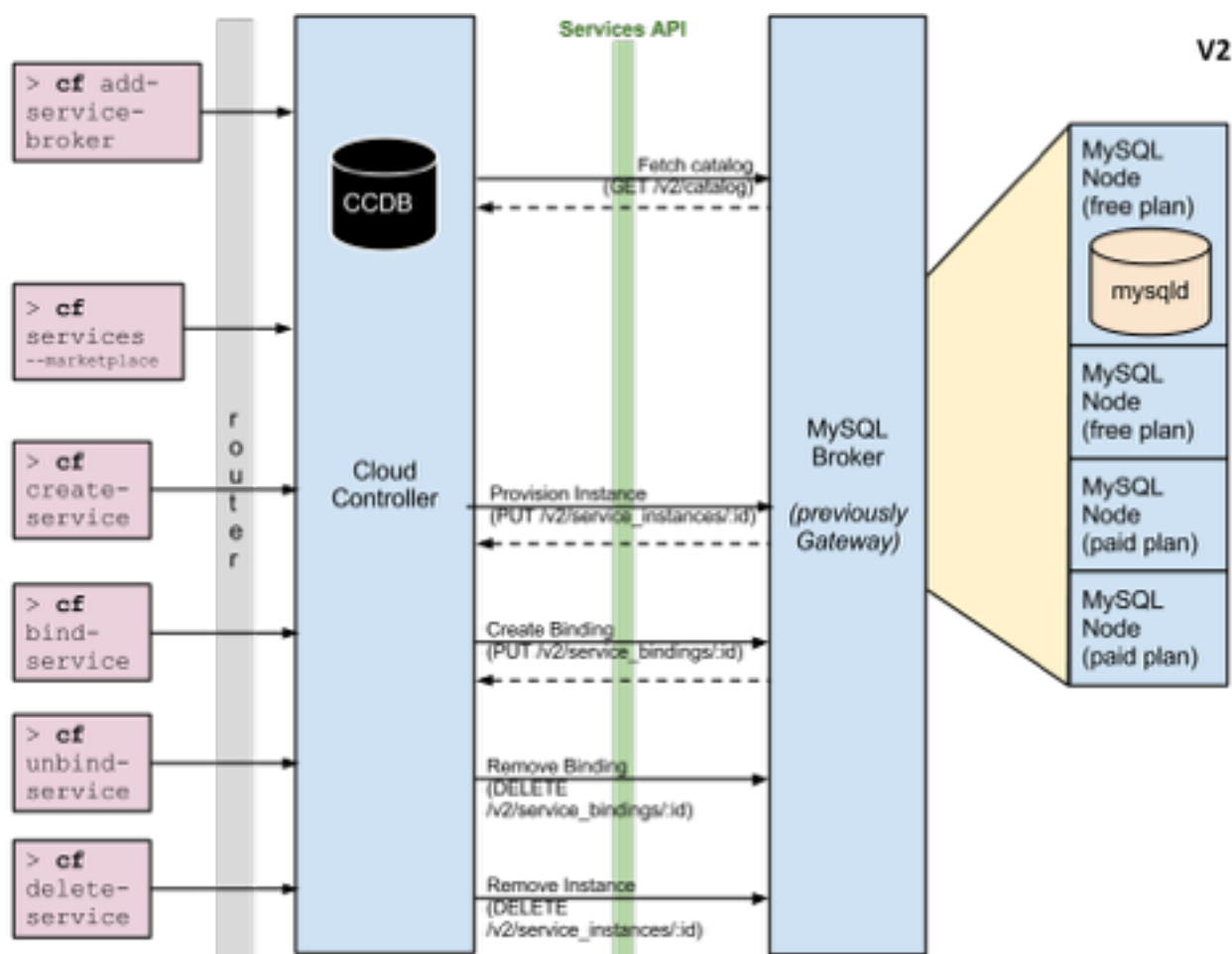
**Managed Services** are integrated with Cloud Foundry by implementing a documented API for which the cloud controller is the client

**Service Broker** is a component which implements the required API.

- Service brokers advertise a catalog of service offerings and service plans to Cloud Foundry, and receive calls from the Cloud Controller for five functions: fetch catalog, create, bind, unbind, and delete.

# REVIEW: Create and Bind Services





# Server Broker API

- GET /v2/catalog – List services and plans available from this broker.
- PUT /v2/service\_instances/: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/:id – Delete a service instance.

# Server Broker Registration

- Make the service broker known to the Cloud Controller
  - `cf create service-broker <broker name> <username> <password> <broker base URI>`
  - Broker should ONLY allow access to those requestors it shared its credential with (Basic Auth)
  - See: <http://docs.gopivotal.com/pivotalcf/services/managing-service-brokers.html#register-broker>
- Make 'plans' accessible to users in a specific org/space
  - Somewhat cumbersome: need to "hand parse" JSON to find service plan UUID as registered with the CC
  - See: <http://docs.gopivotal.com/pivotalcf/services/access-control.html#make-plans-public>

***Need admin creds/role in order to introduce a service broker to the Cloud Controller!***



# Service Broker Implementation

- Service implementation is up to the service provider/developer.
- Cloud Foundry only requires that the service provider implement the service broker API.
- A broker can be implemented as a separate application, or by adding the required http endpoints to an existing service.

# Service Broker Implementation

- **Best Practice:** Each binding is represented by its own *credentials* =>
  - T=1: create service instance
    - ***Neither App-1 or App-2 has access to the service instance***
  - T=2: bind App-1 to service instance
    - ***Only App-1 can access the service instance***
  - T=3: bind App-2 to service instance
    - ***Both App-1 and App-2 have access to the service instance***
  - T=4 unbind App-1 from service instance
    - ***Only App-2 can access the service instance***

# Service Instance Provisioning Examples

The result of provisioning varies by service type, although there are a few 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

Because Cloud Foundry only requires that a service implements the broker API in order to be available to Cloud Foundry end users, many deployment models are possible. The following are examples of valid deployment models:

- Entire service (service backend + broker) packaged and deployed by BOSH alongside Cloud 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 Foundry (**this is the approach we'll take in the lab...**)
- Entire service, including broker, deployed and maintained outside of Cloud Foundry by other means