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

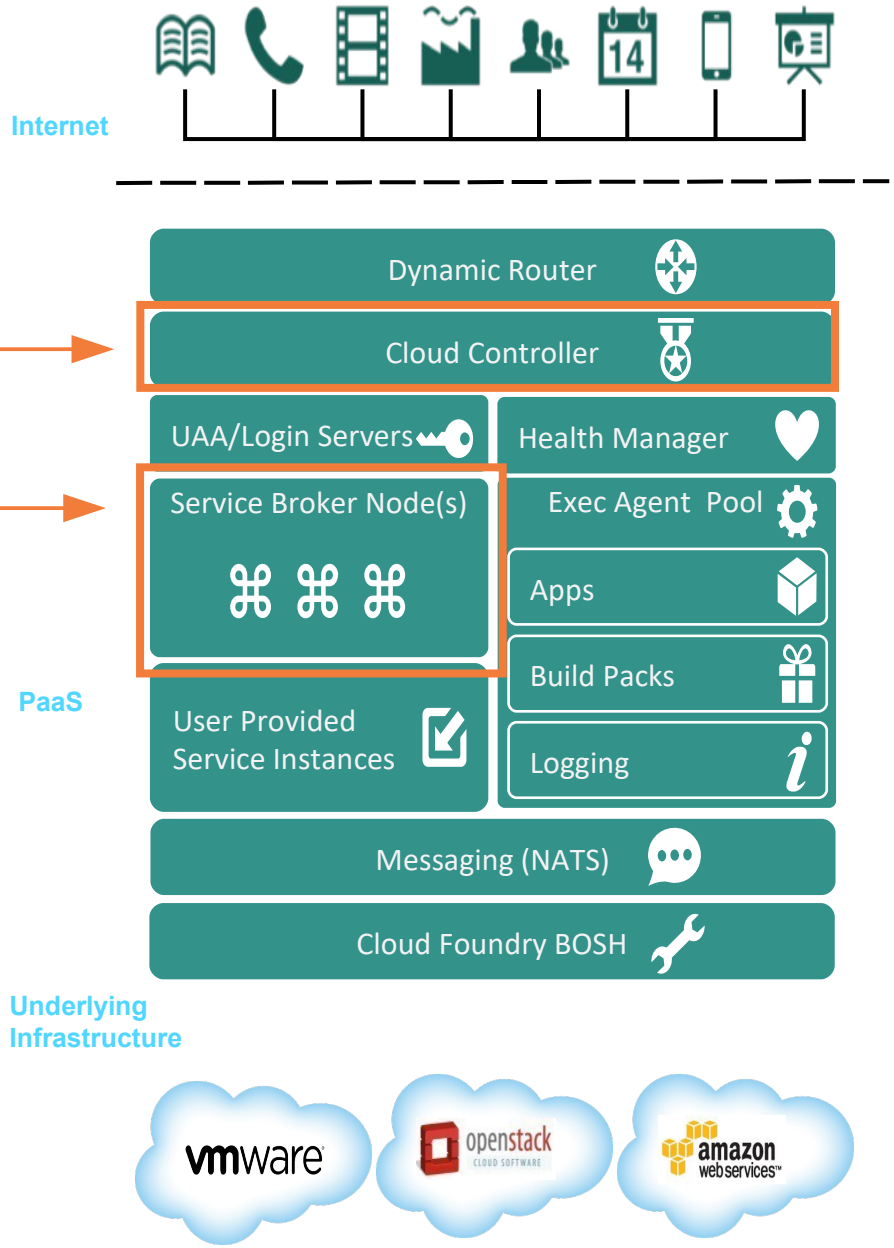
Custom (Managed) Services

- **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*

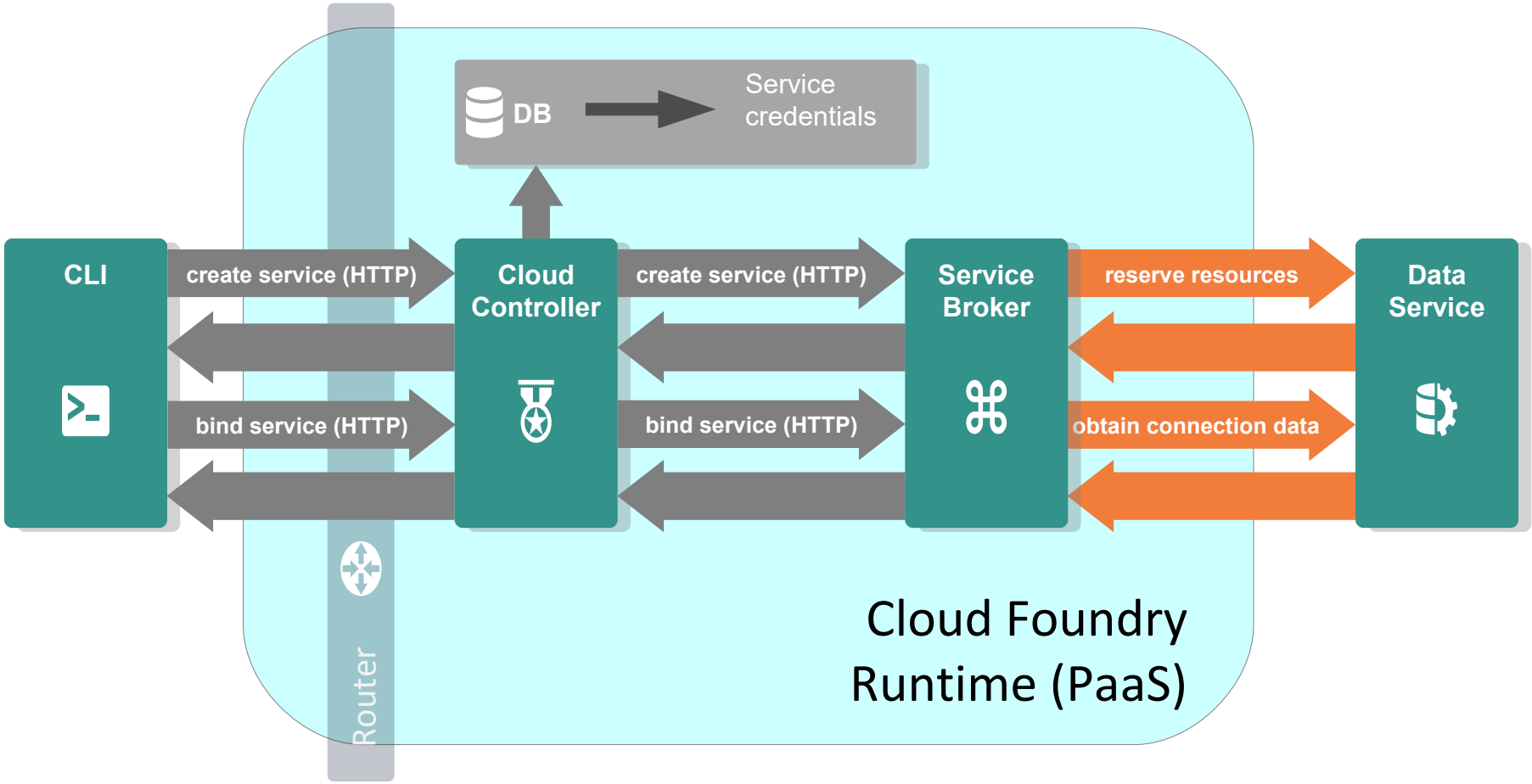
Pivotal

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

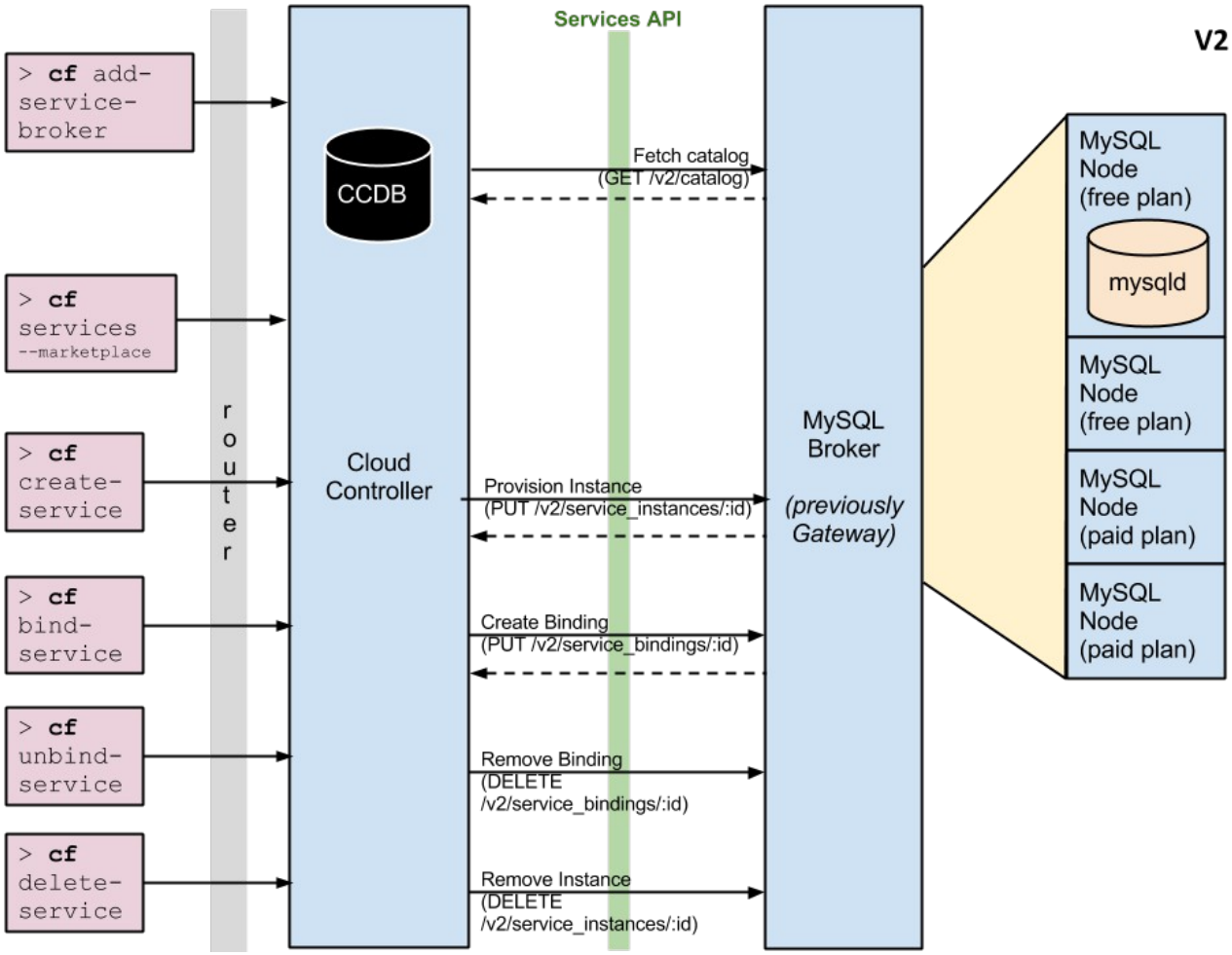


Creating and Binding a *Service*



Service Broker Example (MySQL)

Service Broker



Pivotal

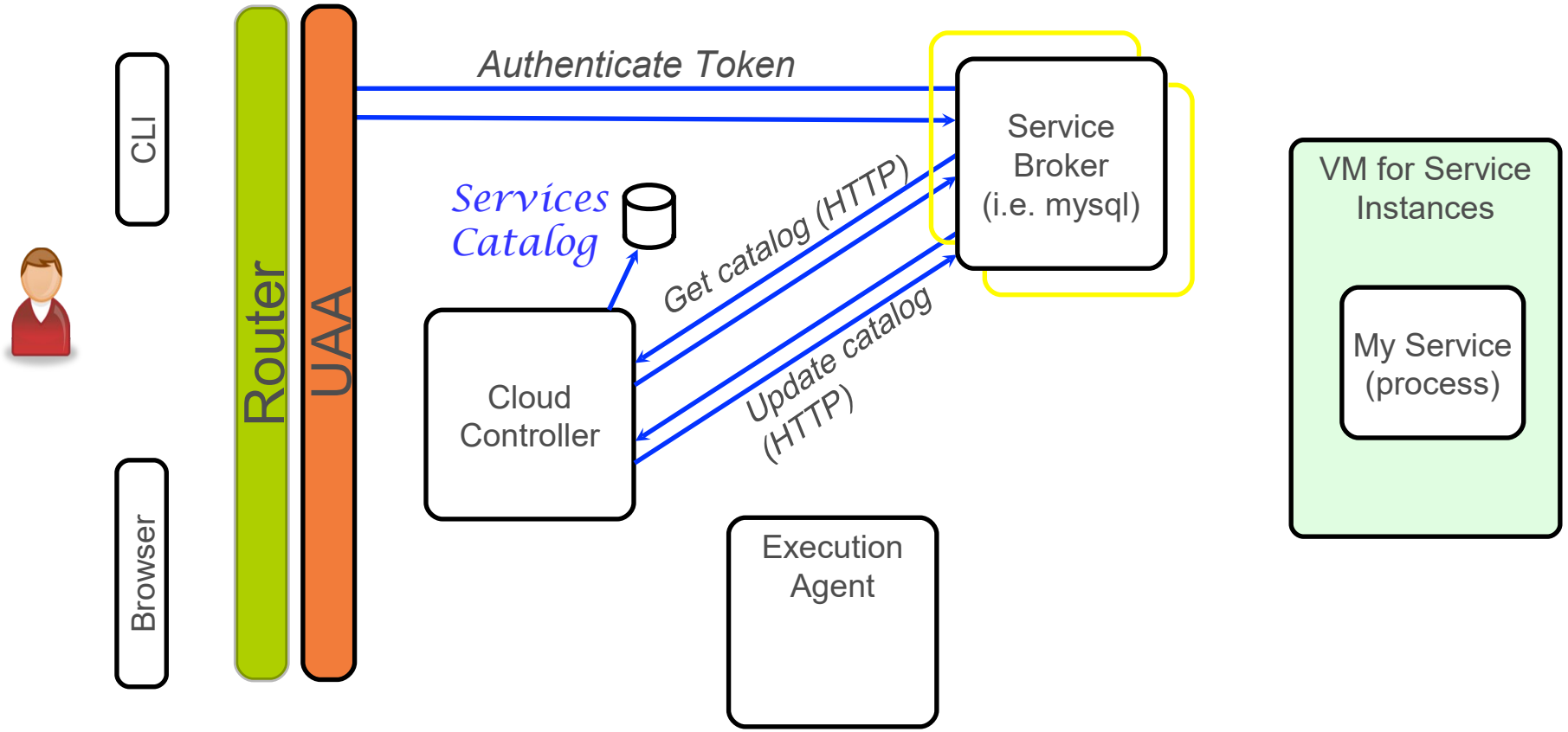
Server Broker API

- 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

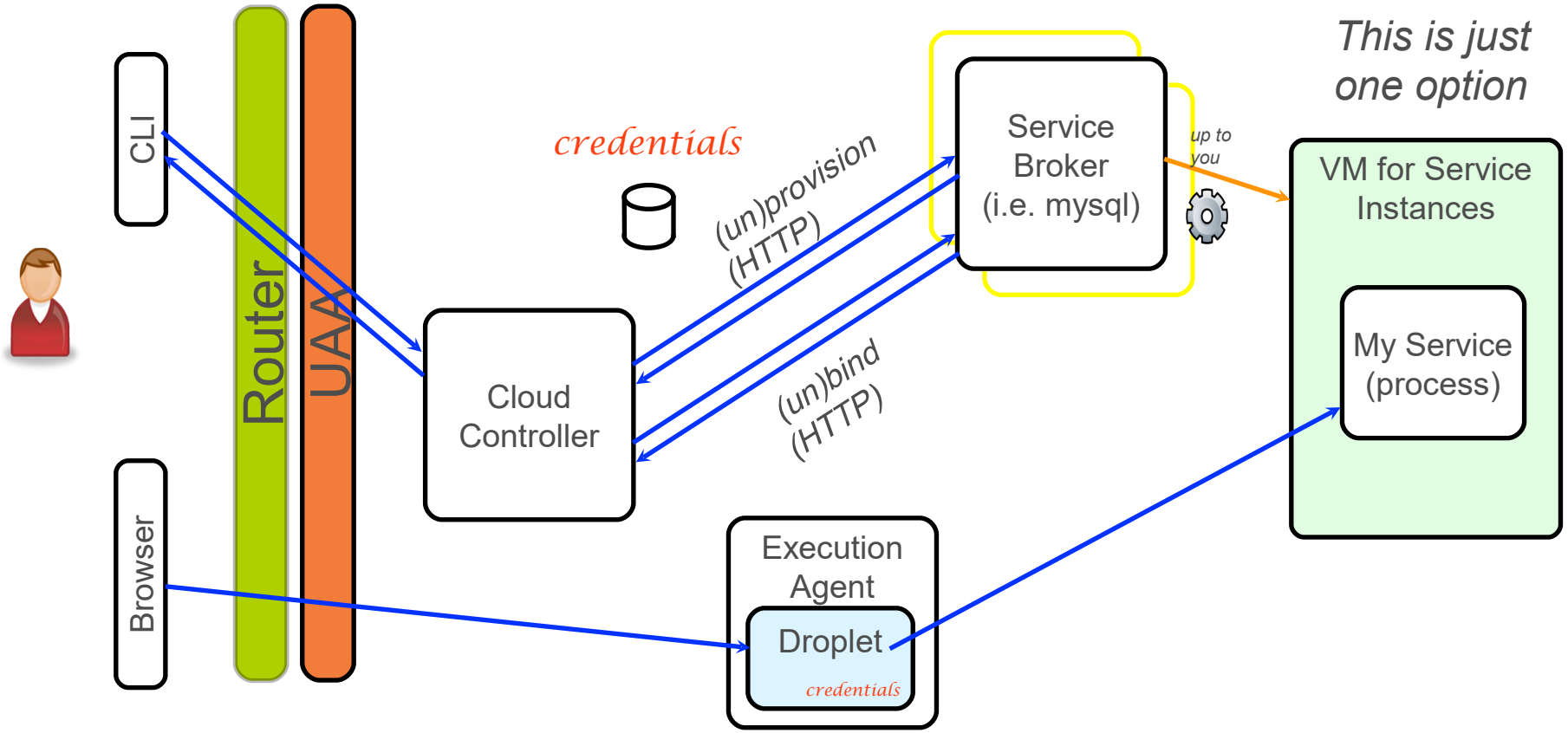
Pivotal

1. Broker Registers Offerings

- On Broker startup



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



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
- 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.

Pivotal

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 ...

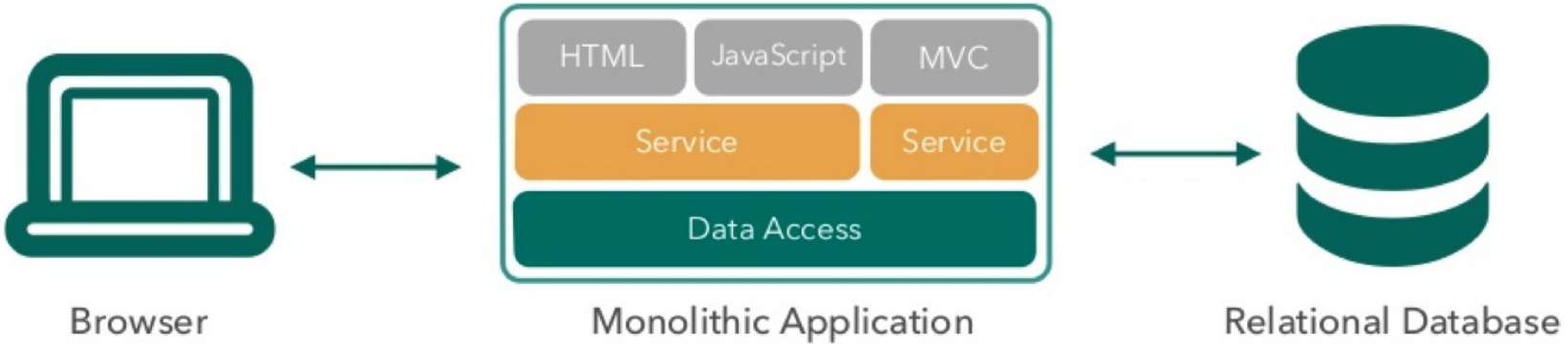
Pivotal

Microservices Explained

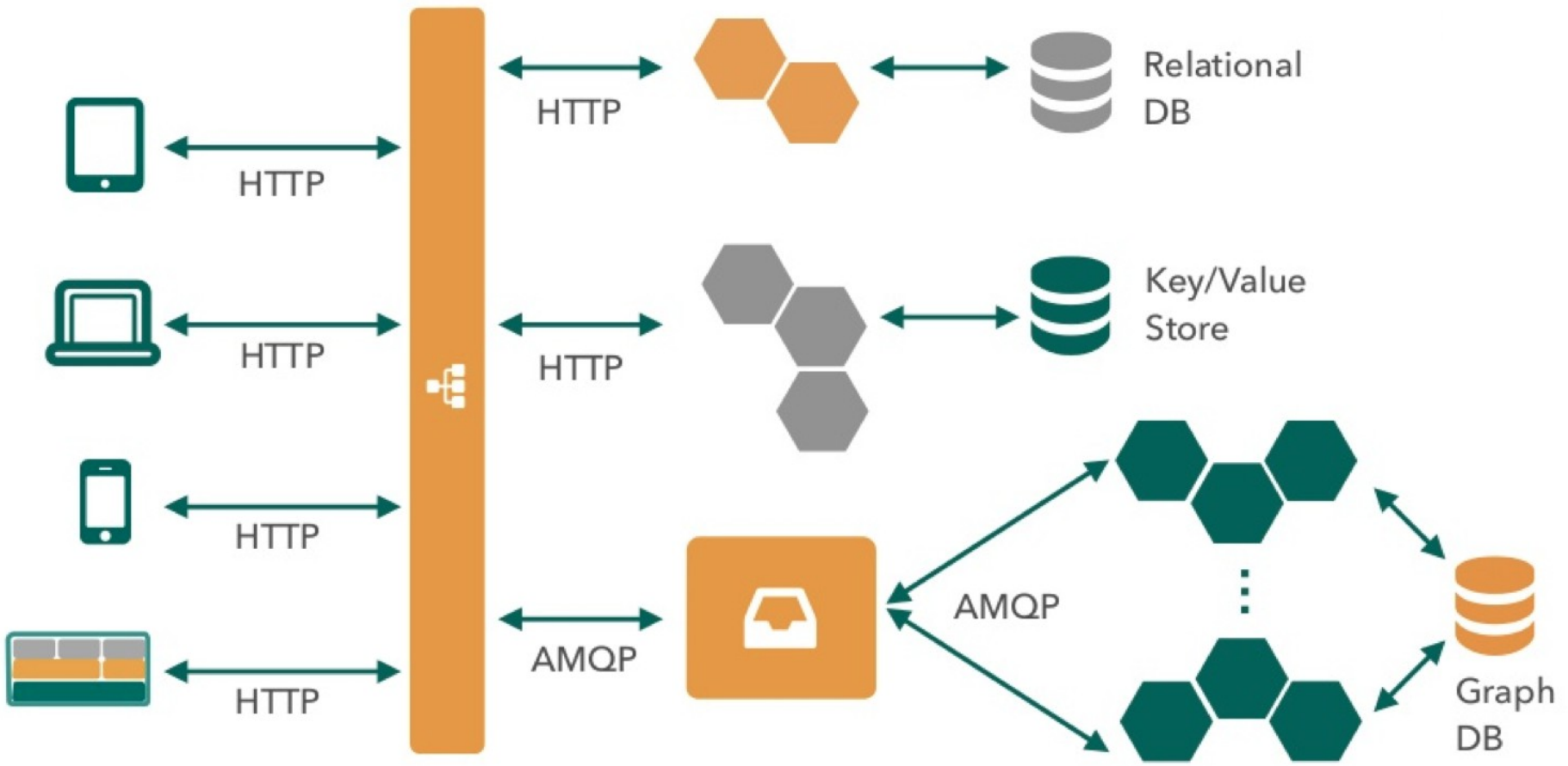
- Take a large monolithic application
  - Decompose into multiple services
  - Each can be maintained and enhanced independently
    - Effect of changed minimized
  - Main application fetched data from multiple microservices
  - SOA – lite!
- Implementation
  - Light weight, resilient
    - Run on Cloud 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



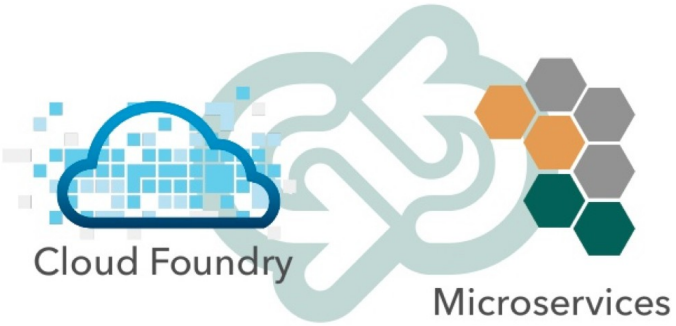
Microservice Architecture



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**

Pivotal

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

Pivotal

Roadmap

We have learned about

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

Pivotal