Technical Talk on

“Pivotal Cloud Foundry”

Pivotal
Cloud Foundry®

INTRODUCTION

Traditional Application Deployment

Specify and procure hardware

Configure hardware

Deploy middleware, database and message broker

Deploy application and settings

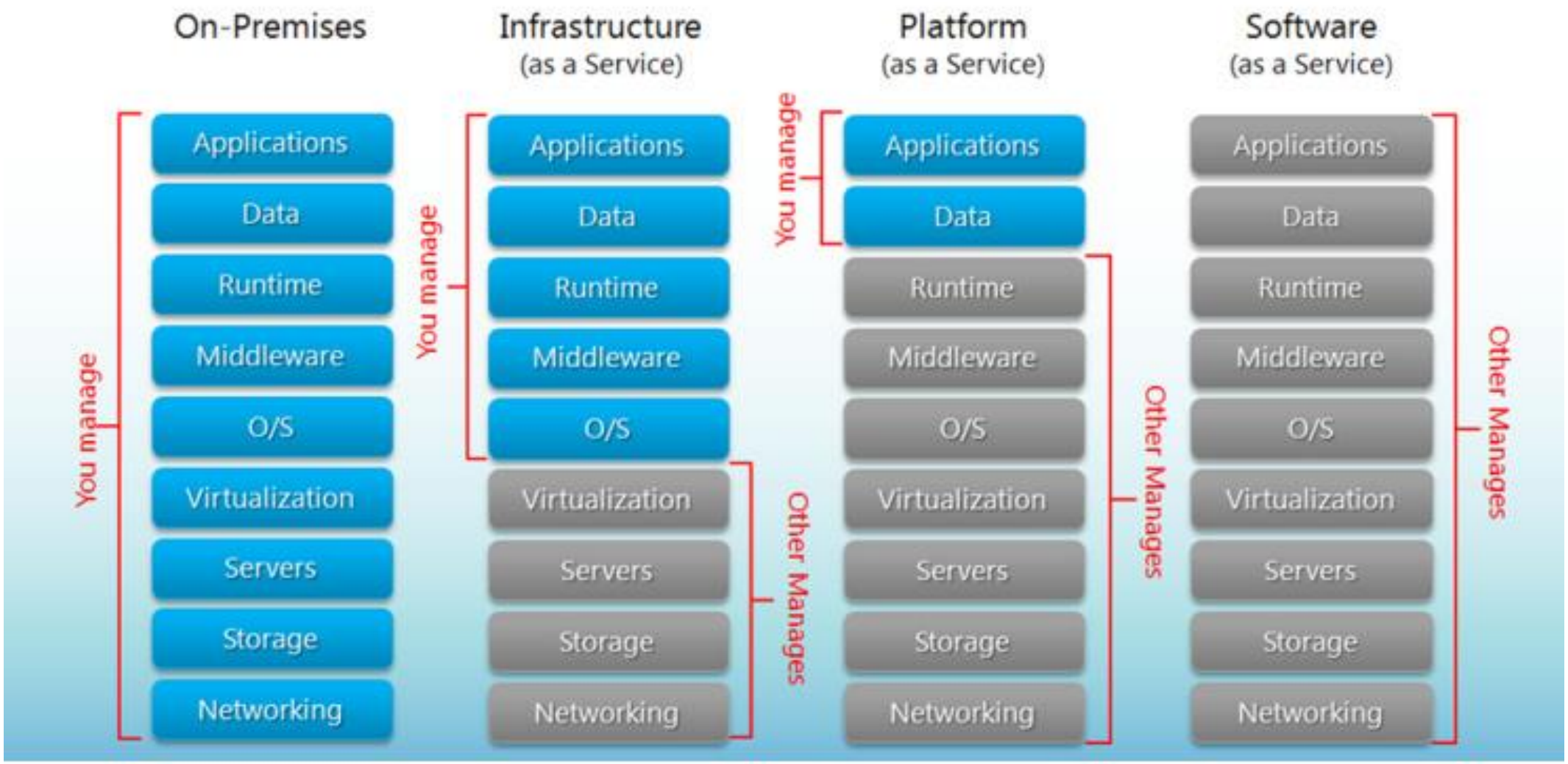
Traditional Application Deployment

Add hardware and reconfigure stack

Add hardware and reconfigure stack

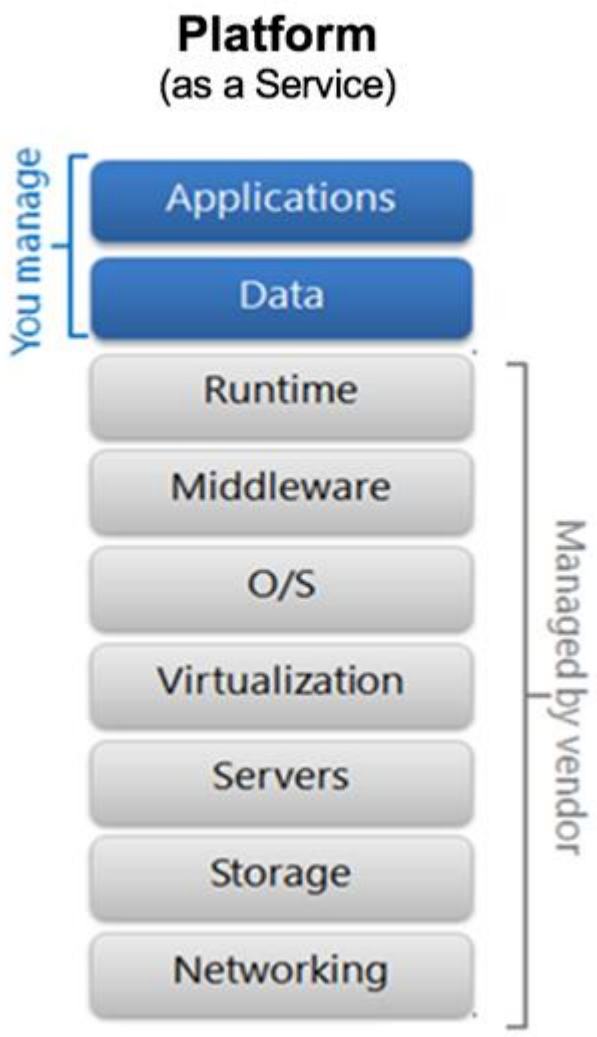
Add hardware and reconfigure stack

Add hardware and reconfigure stack


On-Premises
Infrastructure
(as a Service)
Platform
(as a Service)
Software
(as a Service)
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You manage
Other Manages

Application deployment in Cloud

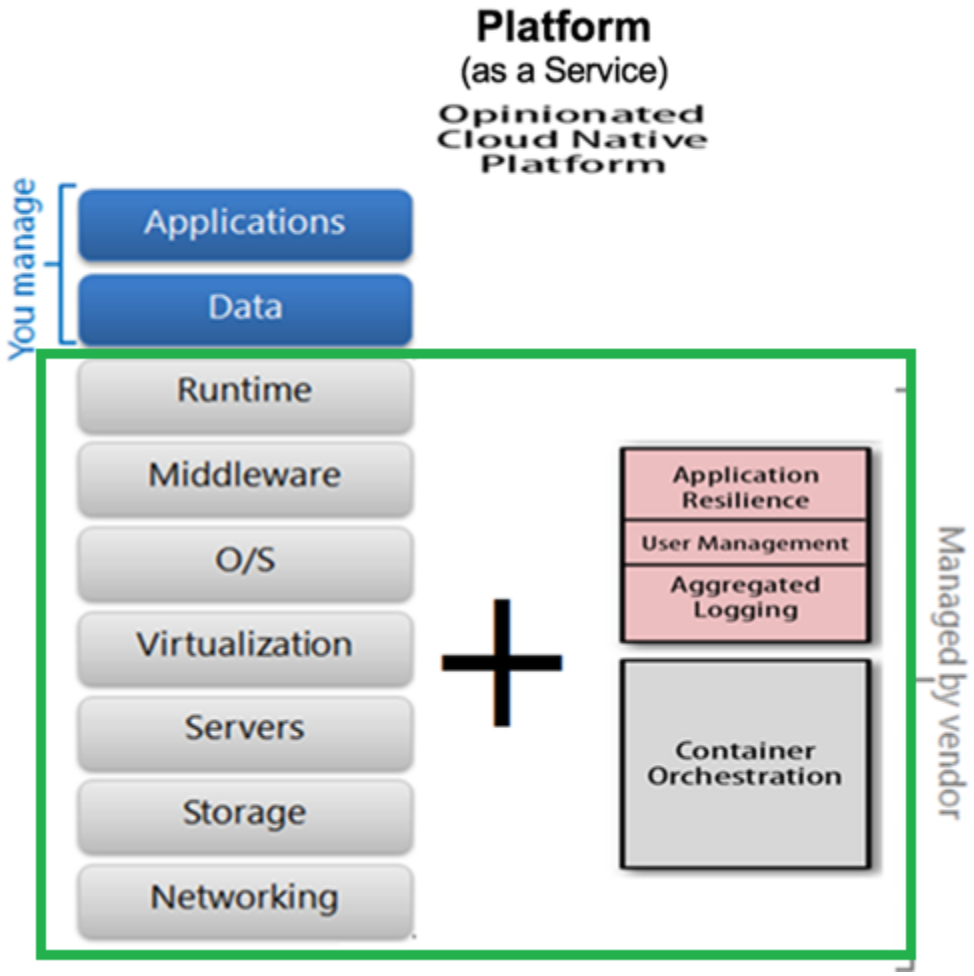
Cloud platform deployment time?	2-4 hours
Application deployment time?	1-5 minutes



Characteristics of a PaaS

- A PaaS solution should have an integrated stack.
- Developers focus on code.
- Operations focus on optimizing the way they mange the application.

PCF - The Opinionated, Cloud Native Platform

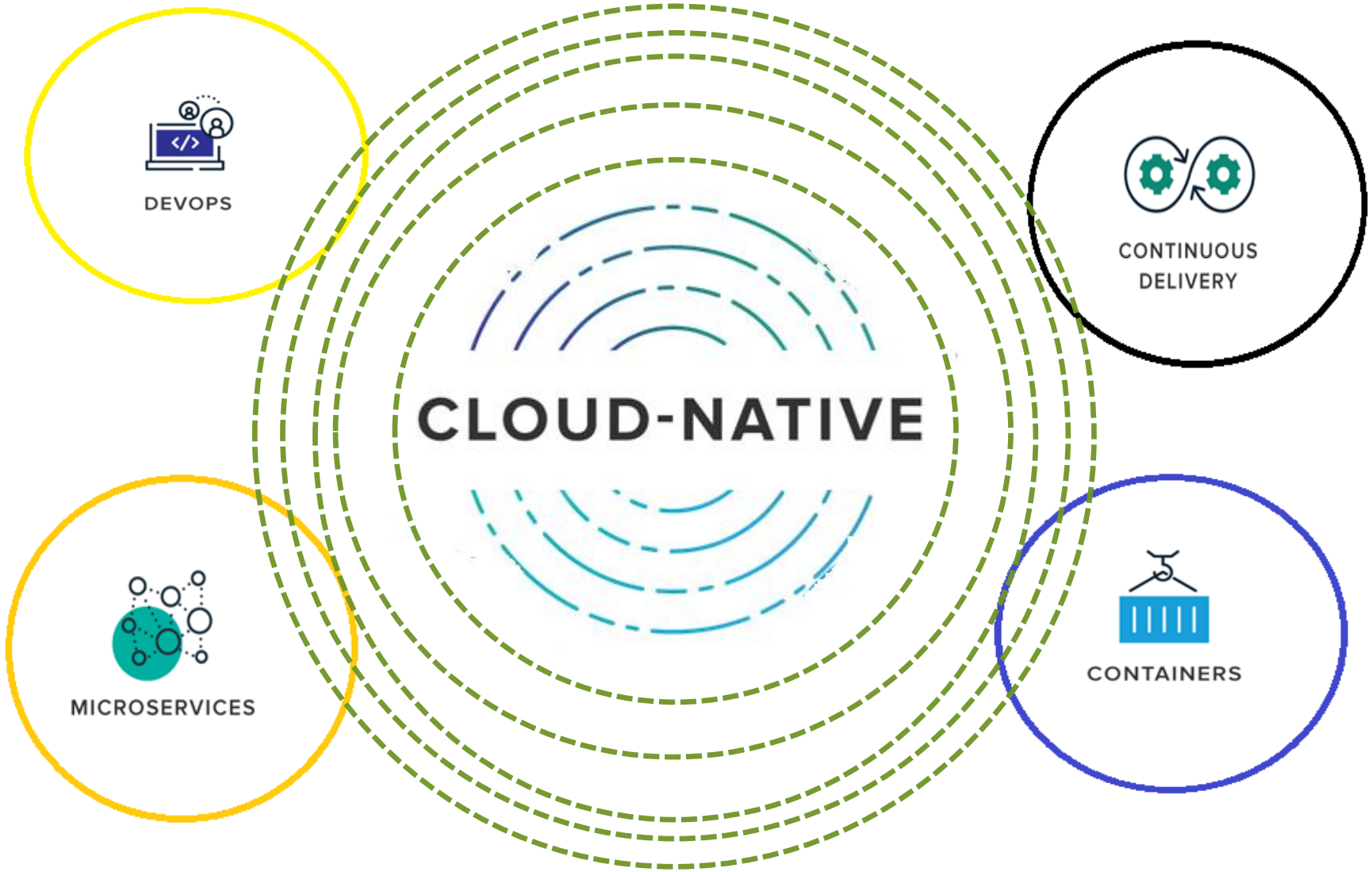


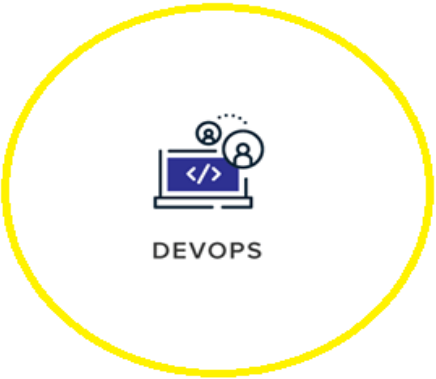
- Platforms are opinionated because they make **specific assumptions** and optimizations to remove complexity and pain from the user.
- Opinionated platforms are designed to be consistent across environments, with every feature working as designed out of the box.
- For example, the Cloud Foundry platform provides the **same user experience** when deployed over different IaaS layers and the same developer experience regardless of the application language.

- **Cloud-native** is an approach to building and running applications that exploits the **advantages** of the **cloud computing delivery** model.

- Cloud-native applications conform to a framework or **"contract"** designed to maximize resilience through predictable behaviours.

- Organizations require a platform for building and operating cloud-native applications and services that automates and integrates on **FOUR characteristics**.


DEVOPS
CONTINUOUS DELIVERY
CLOUD-NATIVE
MICROSERVICES
CONTAINERS



A DevOps culture helps developers and operations work together to deliver shared value to the customer.

- Jenkins is a continuous integration (CI) and continuous delivery (CD) software — an orchestration system with hundreds of plugins to automate everything from building an application and testing it, to the final deployment.

- These plugins can integrate building software from source code platforms such as Git to cloud services.

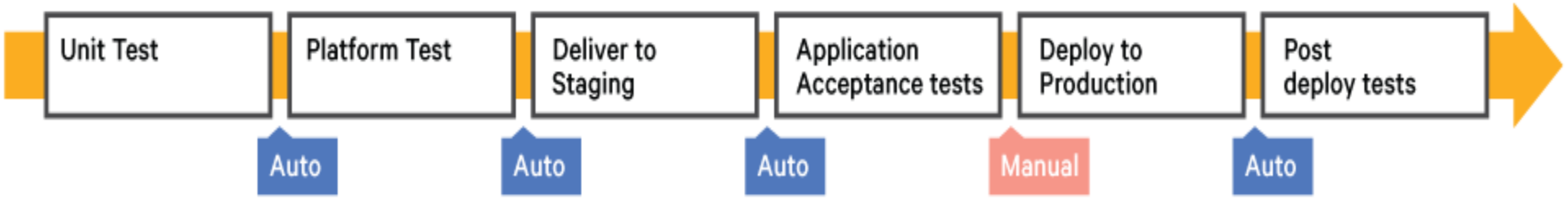
- It is, essentially, a pipeline from source to delivery that’s becoming the engine of DevOps.

Developers, QA and sysadmins work together on an effective and automated workflow:

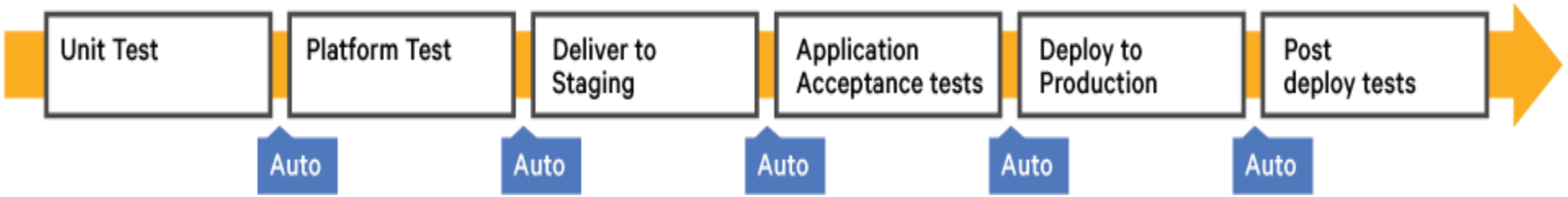
- Do developers need a new working environment? Jenkins will fire up Docker to launch it. Same Production and Development Environments with Docker
- Do sysadmins require a new QA test to pass before an update can go live? They just add it to the Jenkins pipeline.
- Do developers want to pull changes to a production site and launch new servers with it? No problem: sysadmins have already instructed Jenkins what tests to run, and if everything goes fine, how to launch the servers.

Continuous Delivery, enabled by Agile product development practices, is about shipping small batches of software to production constantly, through automation, at less risk, and get feedback faster from end users.

Continuous Delivery



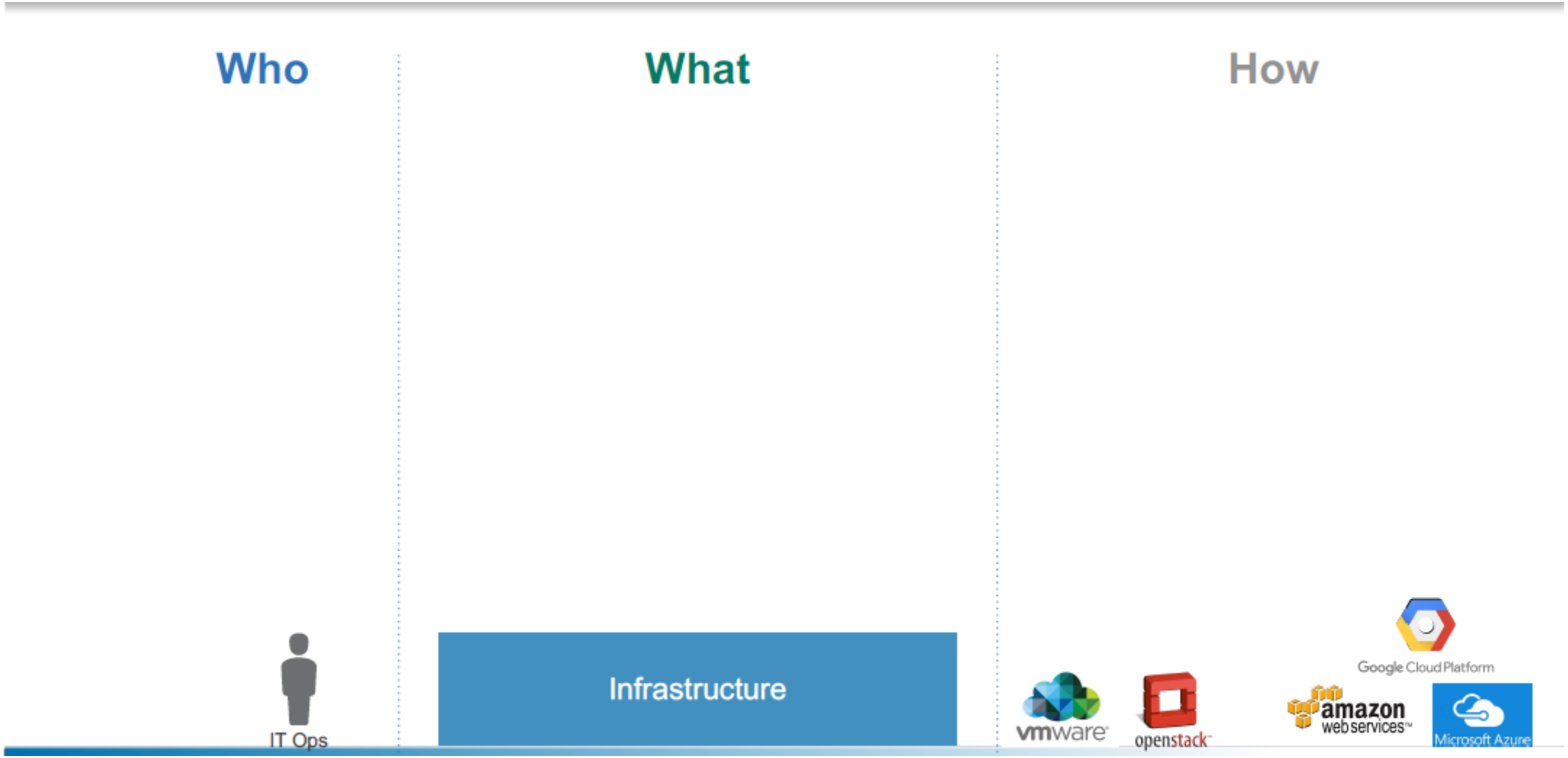
Continuous Deployment

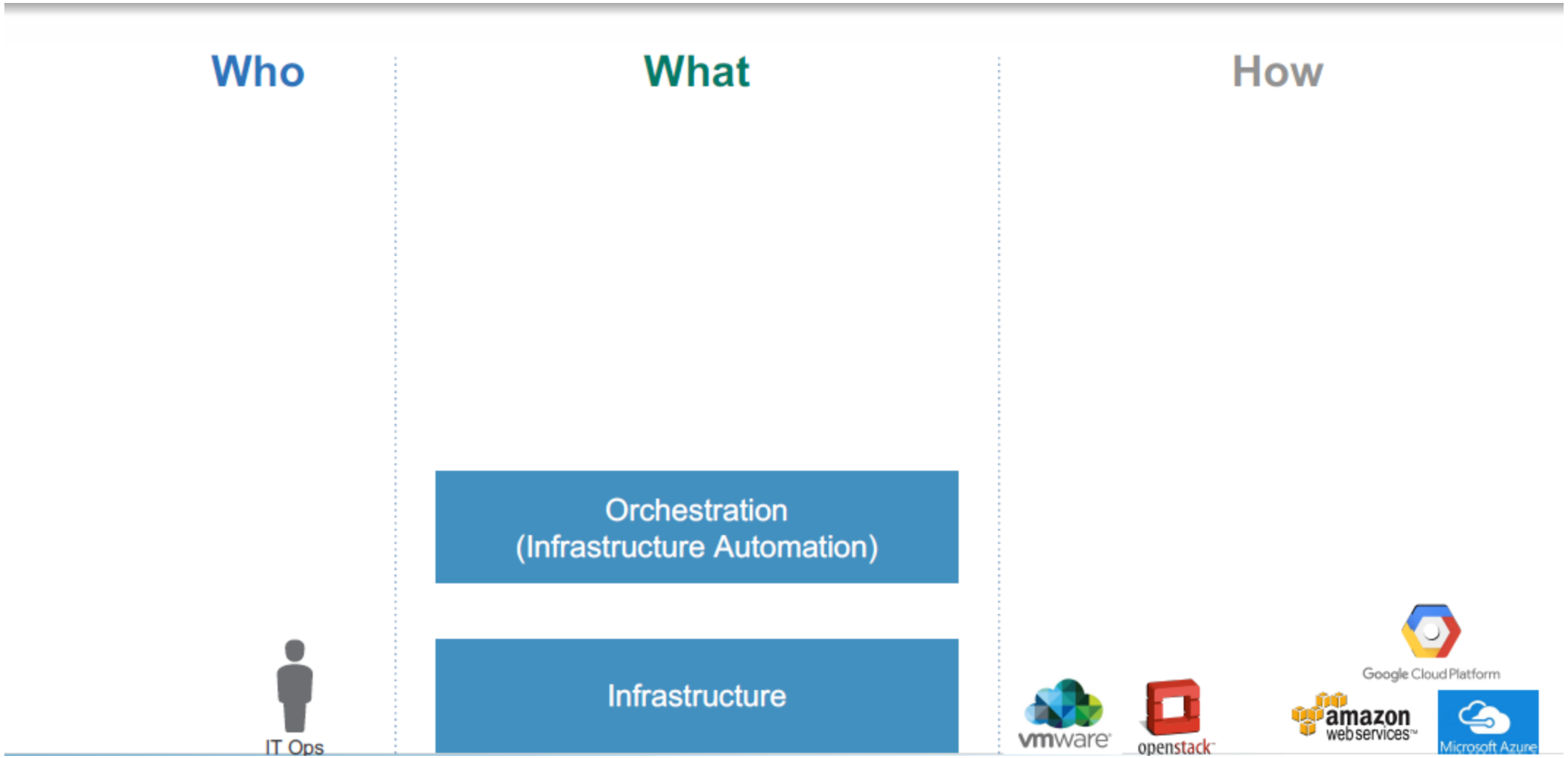


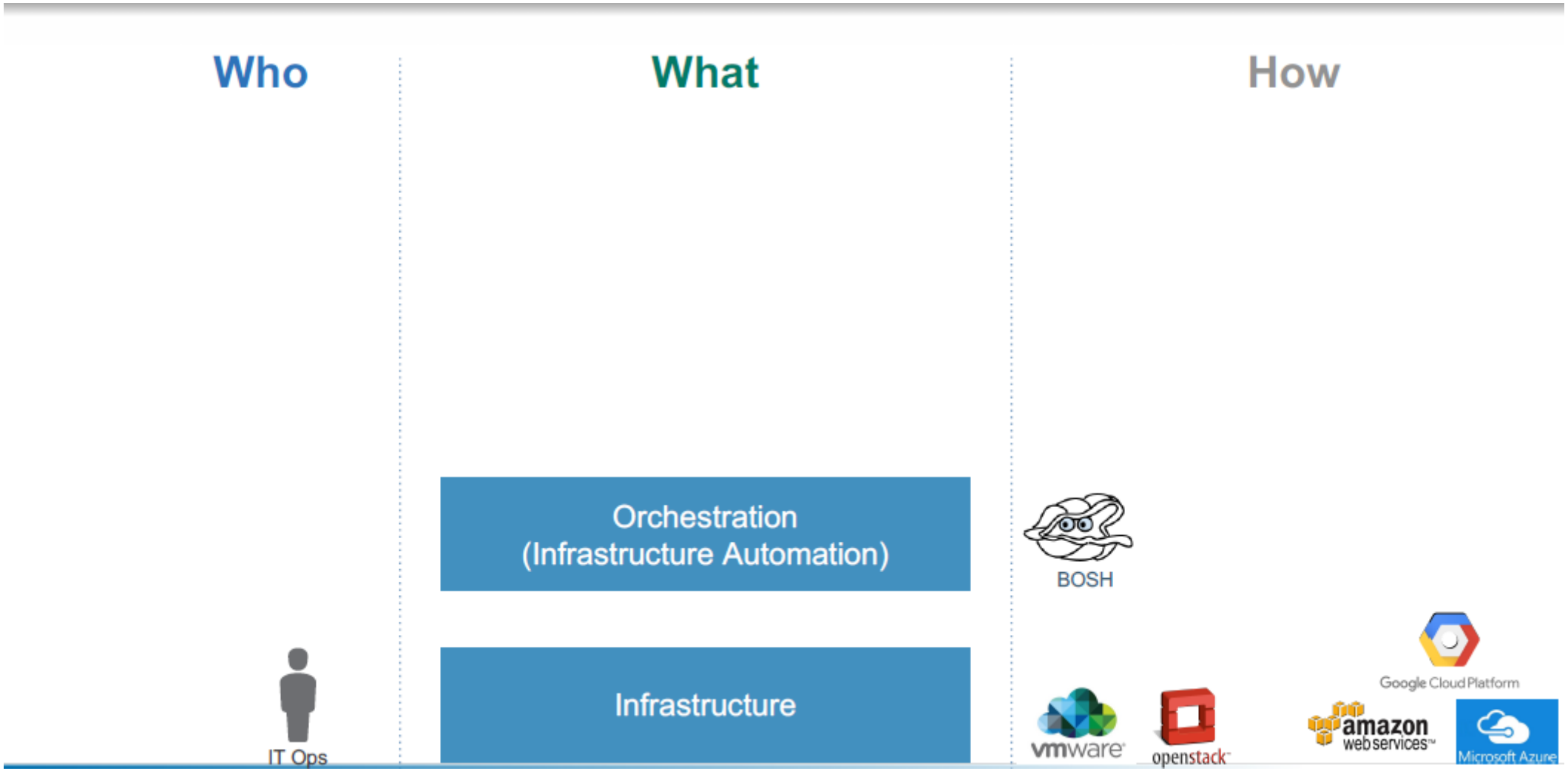
A microservices architecture is a method of developing software applications as a suite of:

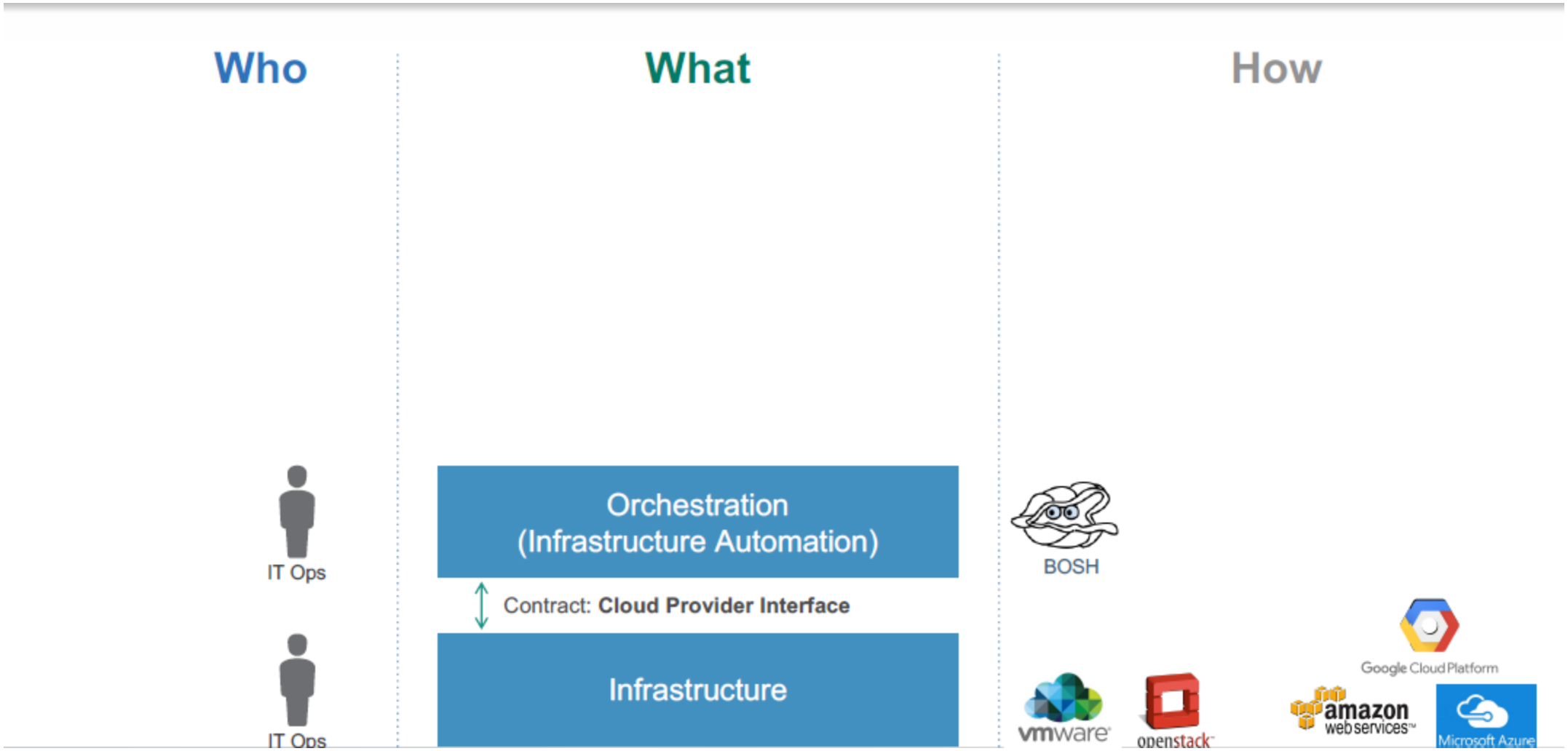
- independently deployable,
- small,
- Each service runs a unique process and
- communicates through a well-defined, lightweight mechanism to serve a business goal.

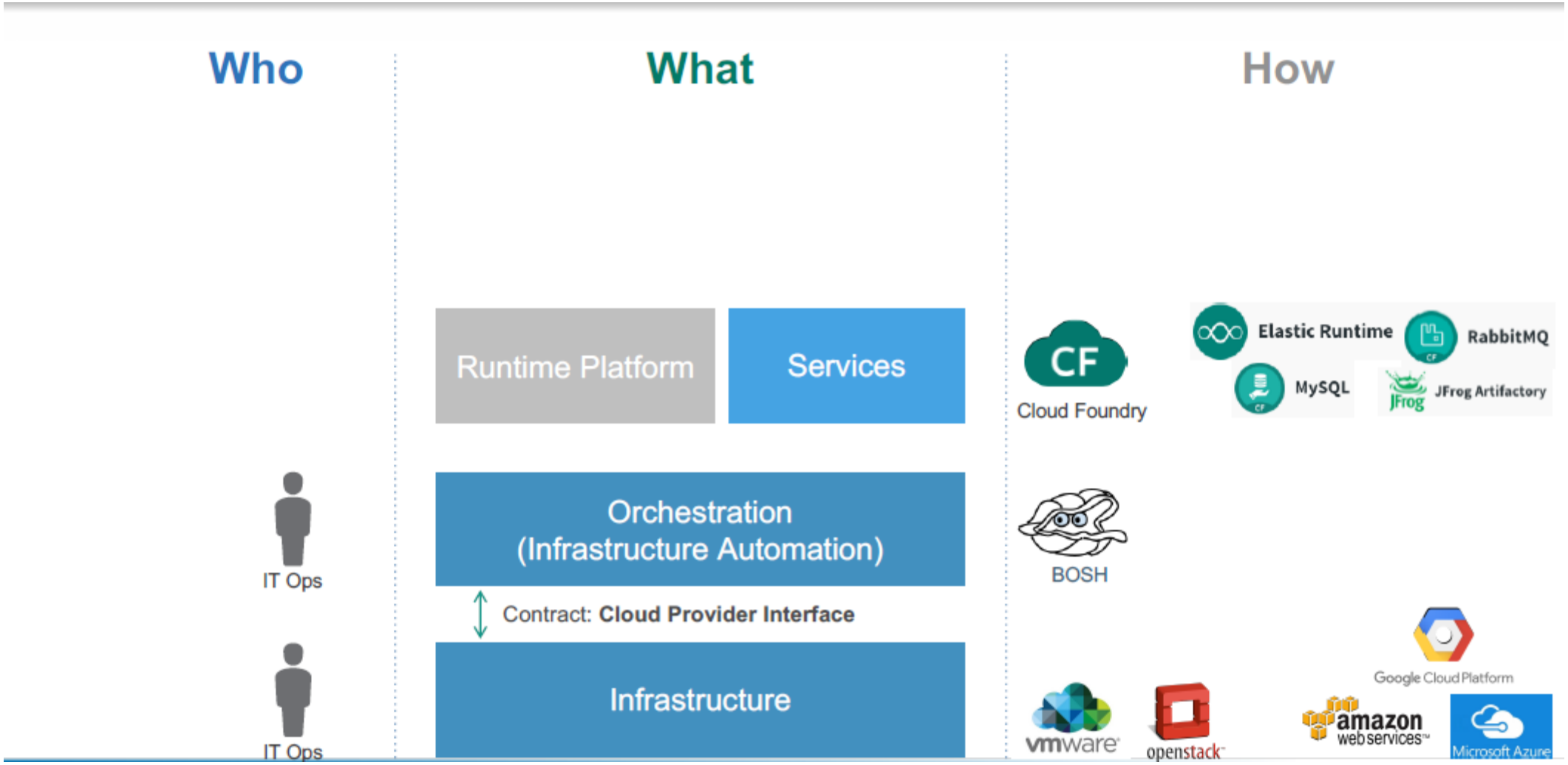
Containerization -- also called container-based virtualization and application containerization -- is an OS-level virtualization method for **deploying and running distributed applications** without launching an entire VM for each application.

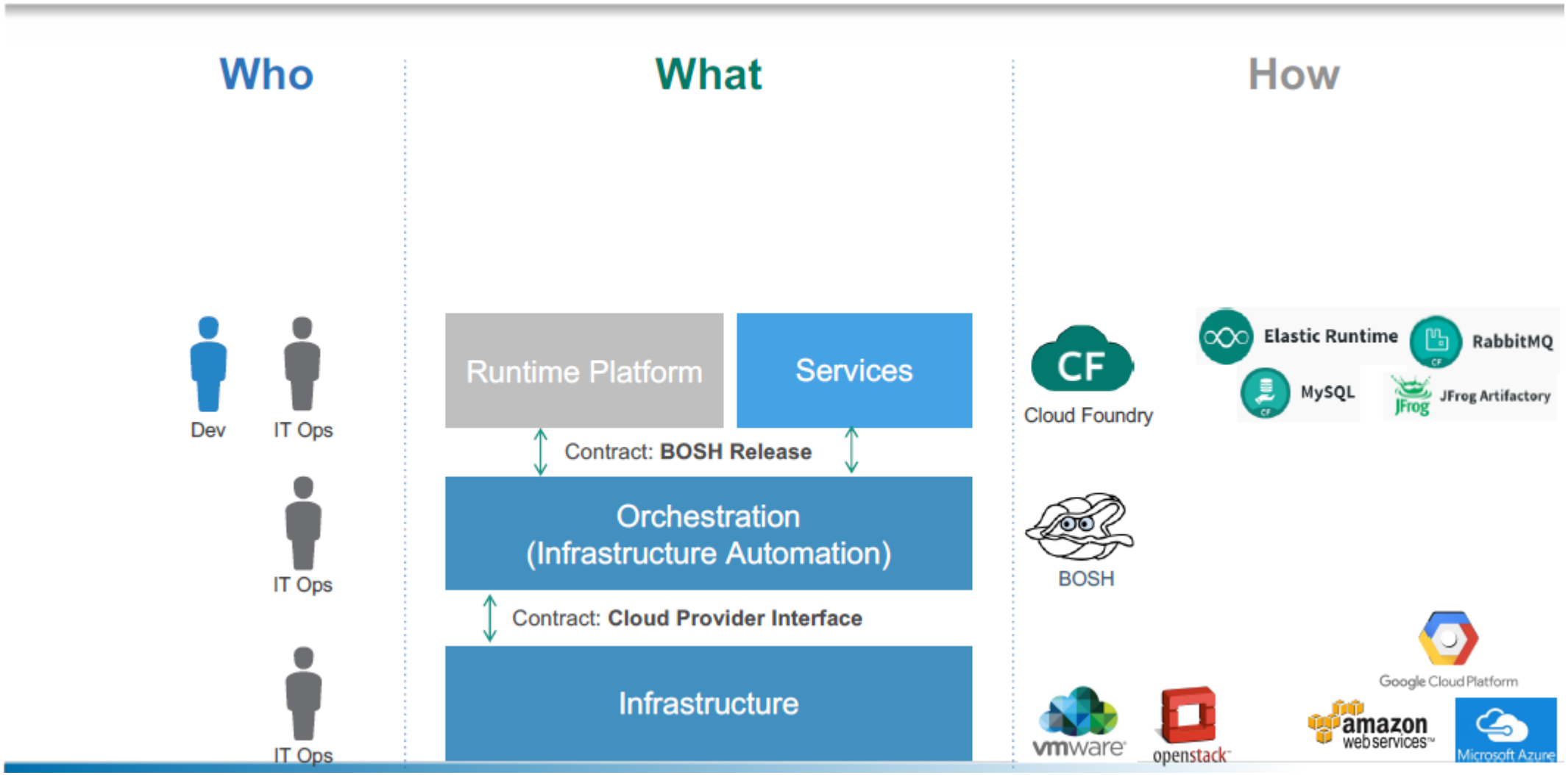

Who
What
How
IT Ops
Infrastructure
vmware
openstack
Google Cloud Platform
amazon web services
Microsoft Azure

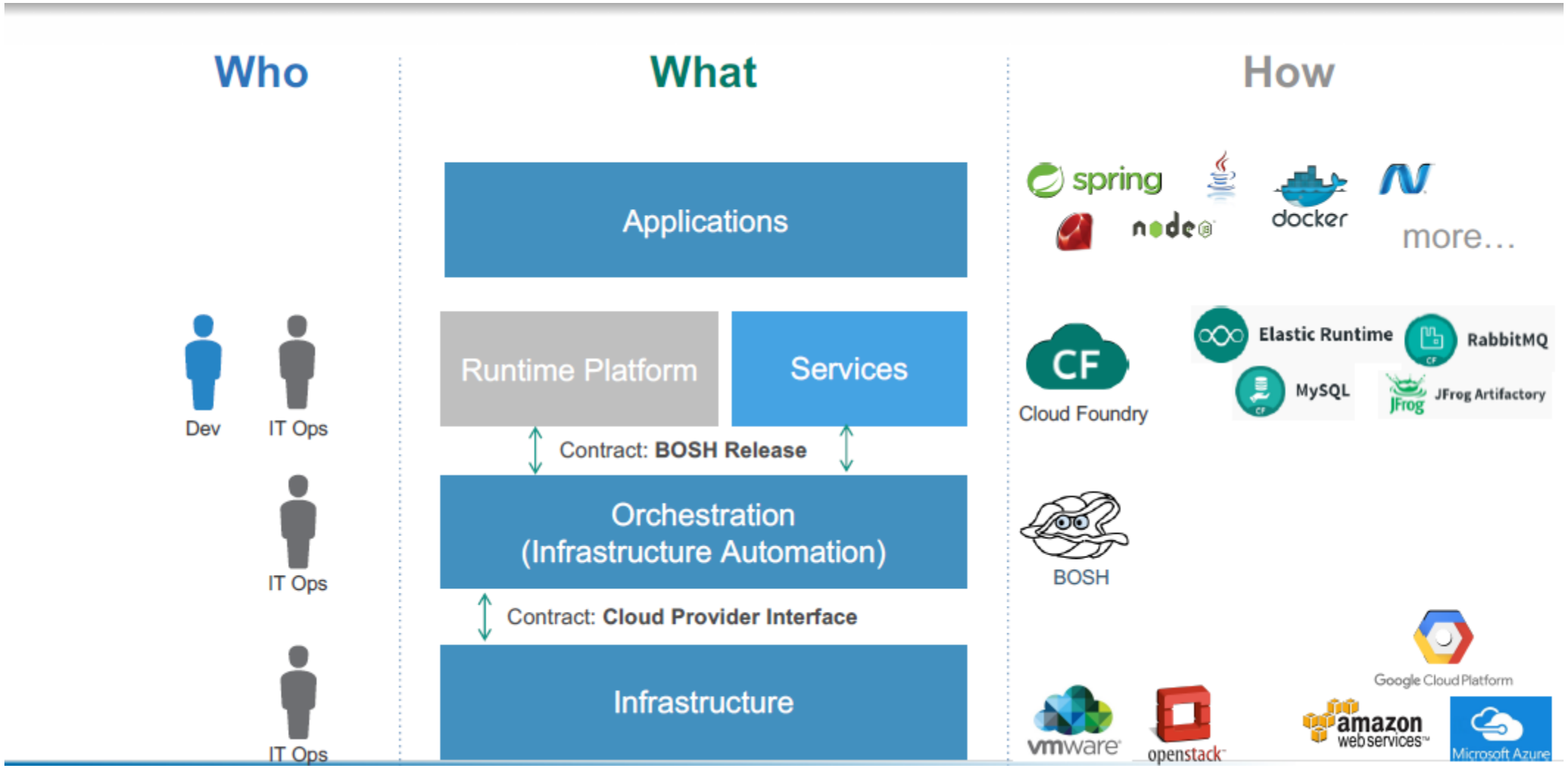

Who
What
How
Orchestration
(Infrastructure Automation)
Infrastructure
IT Ops
vmware
openstack
Google Cloud Platform
amazon
web services
Microsoft Azure

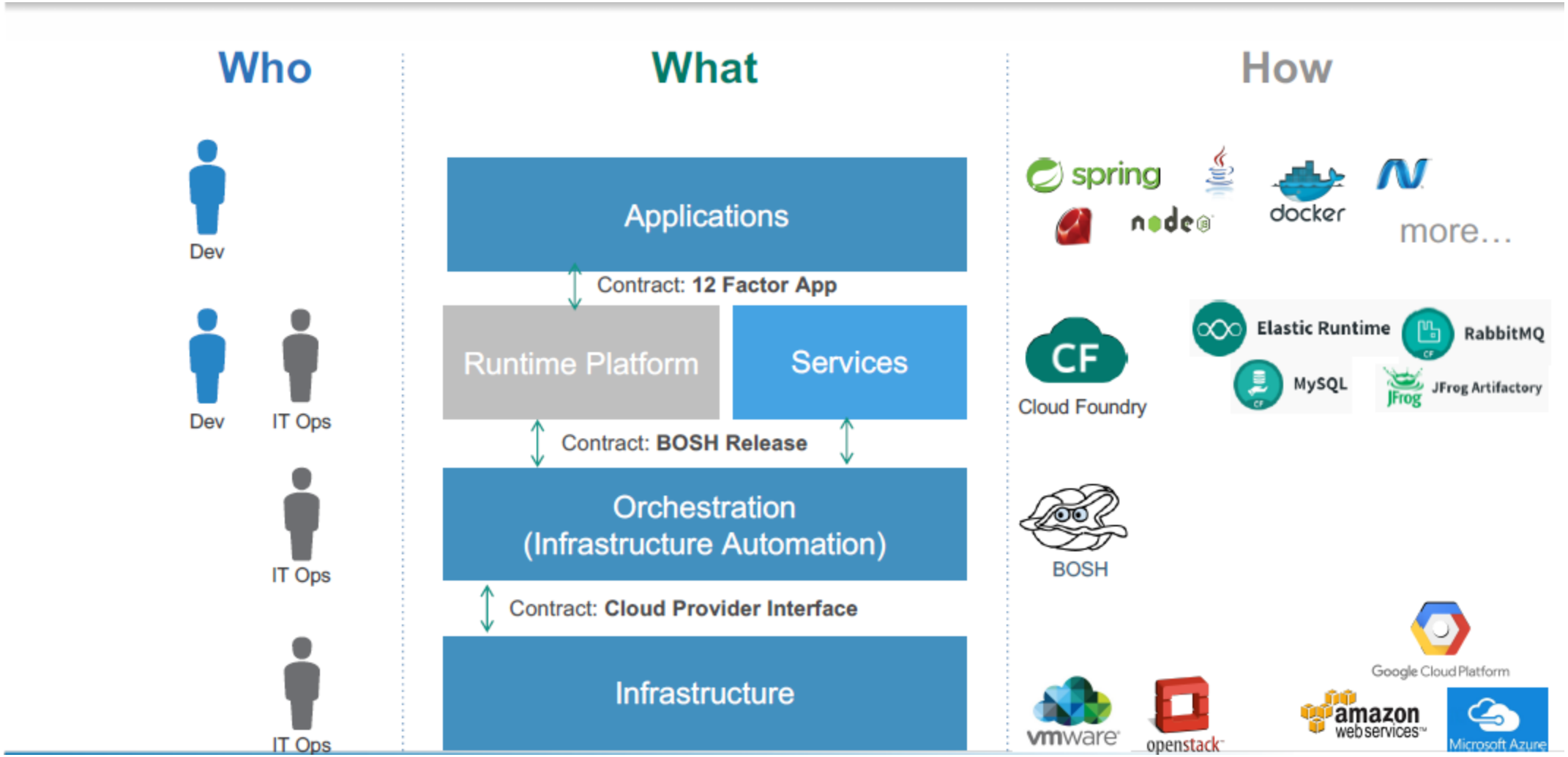

Who
What
How
Orchestration
(Infrastructure Automation)
BOSH
Infrastructure
IT Ops
vmware
openstack
Google Cloud Platform
amazon
web services
Microsoft Azure


Who
What
How
IT Ops
IT Ops
Orchestration
(Infrastructure Automation)
Contract: Cloud Provider Interface
Infrastructure
BOSH
vmware
openstack
Google Cloud Platform
amazon
web services
Microsoft Azure


Who
What
How
Runtime Platform
Services
Orchestration
(Infrastructure Automation)
Contract: Cloud Provider Interface
Infrastructure
IT Ops
IT Ops
CF
Cloud Foundry
Elastic Runtime
RabbitMQ
MySQL
JFrog Artifactory
BOSH
Google Cloud Platform
vmware
openstack
amazon web services
Microsoft Azure


Who
What
How
Dev
IT Ops
IT Ops
IT Ops
Runtime Platform
Services
Contract: BOSH Release
Orchestration
(Infrastructure Automation)
Contract: Cloud Provider Interface
Infrastructure
CF
Cloud Foundry
Elastic Runtime
RabbitMQ
MySQL
JFrog Artifactory
BOSH
Google Cloud Platform
vmware
openstack
amazon web services
Microsoft Azure


Who
What
How
Applications
Runtime Platform
Services
Contract: BOSH Release
Orchestration
(Infrastructure Automation)
Contract: Cloud Provider Interface
Infrastructure
Dev
IT Ops
IT Ops
IT Ops
spring
docker
node
more…
CF
Cloud Foundry
Elastic Runtime
RabbitMQ
MySQL
JFrog Artifactory
BOSH
Google Cloud Platform
vmware
openstack
amazon web services
Microsoft Azure


Who
What
How
Dev
Dev
IT Ops
IT Ops
IT Ops
Applications
Contract: 12 Factor App
Runtime Platform
Services
Contract: BOSH Release
Orchestration
(Infrastructure Automation)
Contract: Cloud Provider Interface
Infrastructure
spring
docker
more…
CF
Cloud Foundry
Elastic Runtime
RabbitMQ
MySQL
JFrog Artifactory
BOSH
Google Cloud Platform
vmware
openstack
amazon web services
Microsoft Azure

Cloud Native Platform

Developer — 1. Application Framework

Contract – 12 Factor App

Dev+Ops — 2. Container Runtime

Contract – BOSH Release

IT Ops — 3. Infrastructure Automation

Contract – Cloud Provider Interface

IT Ops — 4. Infrastructure

"Contracts" between Applications, opinionated frameworks like Spring Boot and Spring Cloud and opinionated Cloud Native Platforms like Cloud Foundry help significantly accelerate the development of Cloud Native applications

12 factors (solid principle for Cloud Software Architecture)

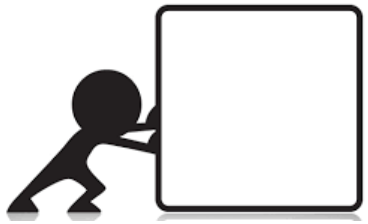
Codebase	One codebase tracked in revision control, many deploys
Dependencies	Explicitly declare and isolate dependencies
Config	Store configuration in the environment
Backing Services	Treat backing services as attached resources
Build, release, run	Strictly separate build and run stages
Processes	Execute the app as one or more stateless processes
Port binding	Export services via port binding
Concurrency	Scale out via the process model
Disposability	Maximize robustness with fast startup and graceful shutdown
Dev/prod parity	Keep development, staging, and production as similar as possible
Logs	Treat logs as event streams
Admin processes	Run admin/management tasks as one-off processes

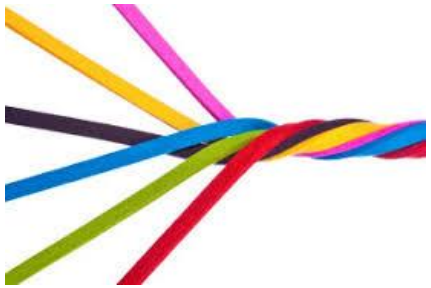
Pivotal Cloud Foundry®, powered by Cloud Foundry, delivers a turnkey PaaS experience on multiple infrastructures with leading application and data services.

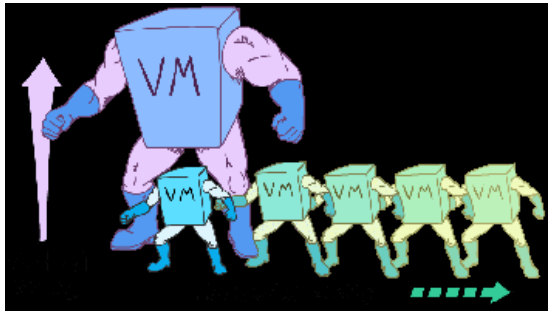
DevOps is a culture, not a role!


Pivotal
Cloud Foundry
Developer

Identify the pictures given below

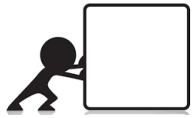






Cloud Foundry Command Line Interface (cf CLI).

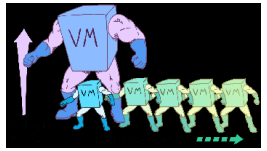
cf target



cf push

cf create-service

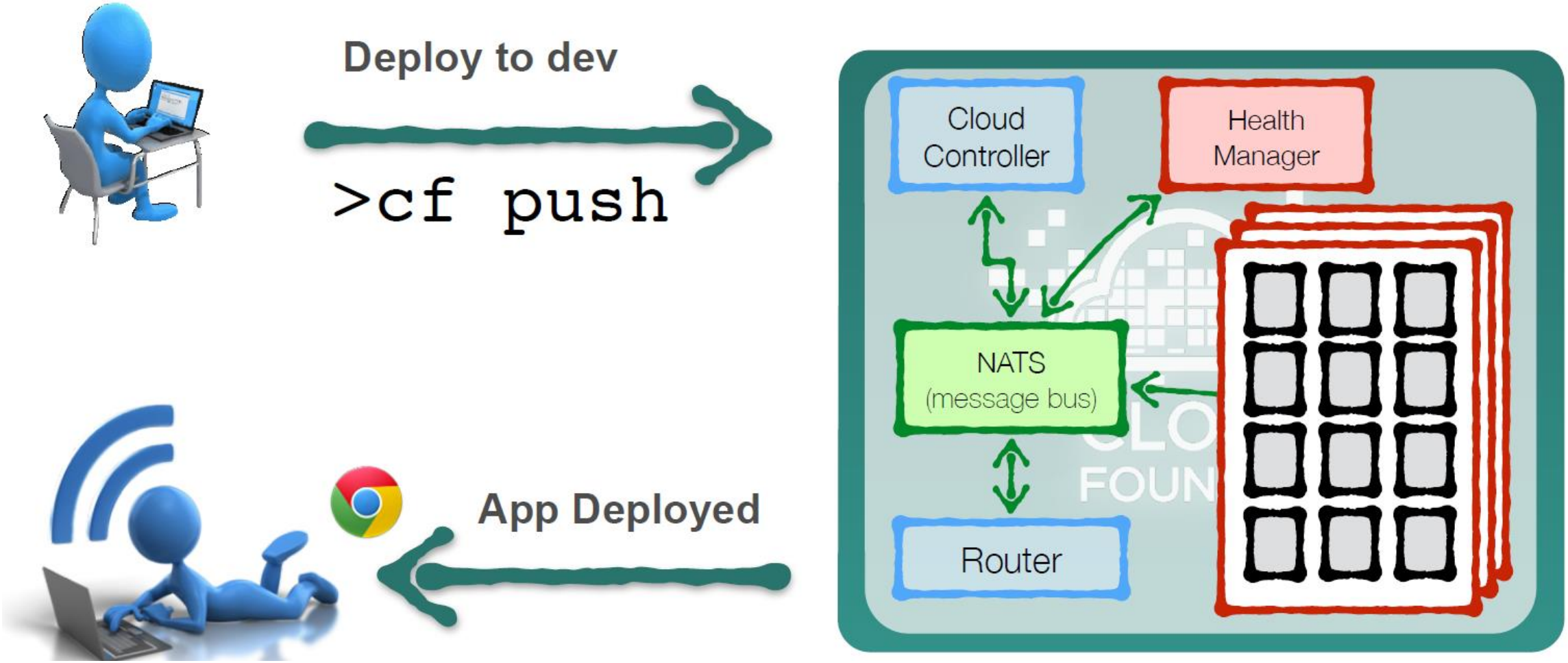
cf bind-service



cf scale

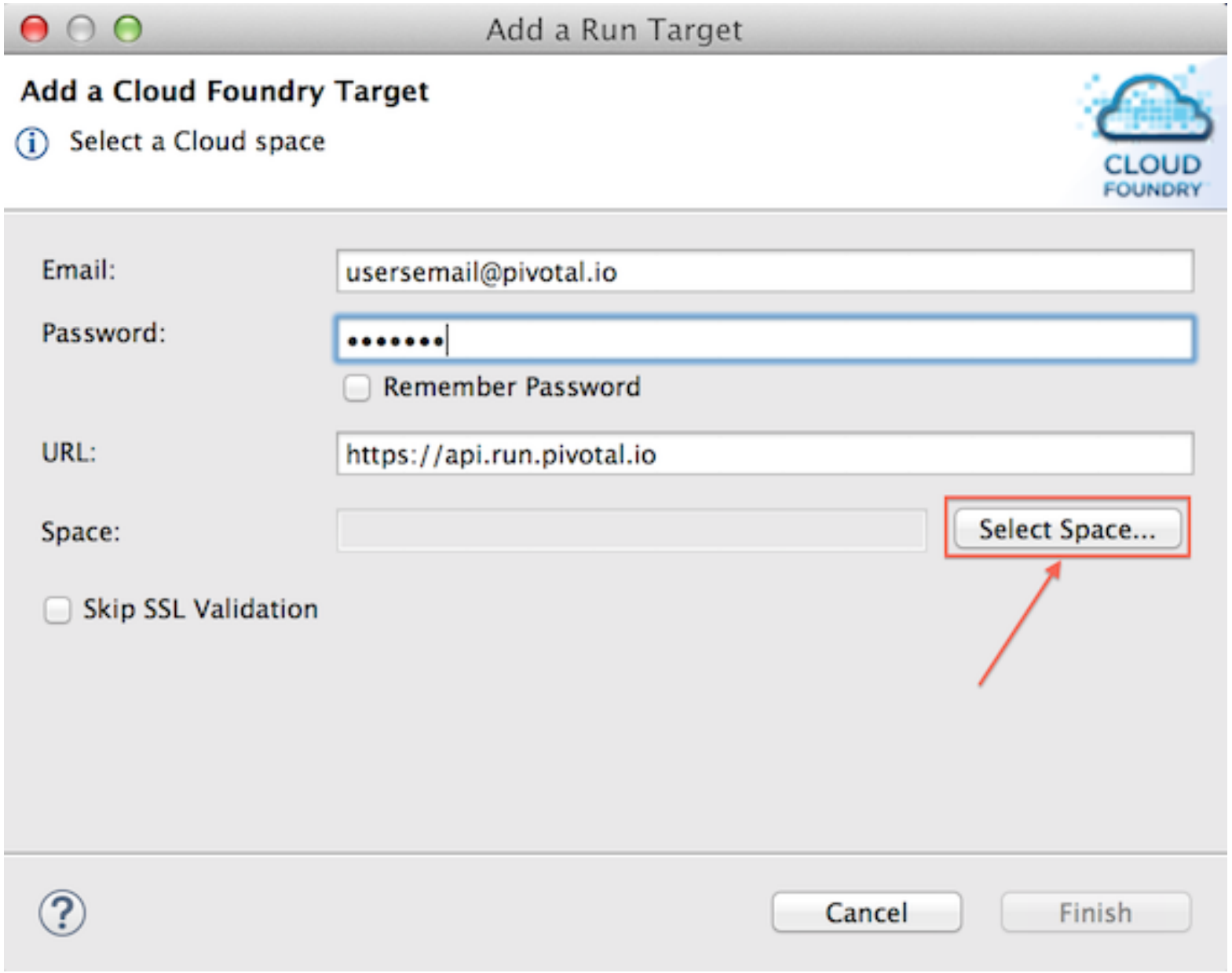
Pivotal
Cloud Foundry®

Developer-Friendly


Deploy to dev
>cf push
Cloud Controller
Health Manager
NATS (message bus)
Router
App Deployed

```
C:\Users\K V Ramana Rao>cf help
cf version 6.32.0+0191c33d9.2017-09-26, Cloud Foundry command line tool
Usage: cf [global options] command [arguments...] [command options]

Before getting started:
  config    login,l      target,t
  help,h    logout,lo

Application lifecycle:
  apps,a        run-task,rt    events
  push,p        logs           set-env,se
  start,st      ssh            create-app-manifest
  stop,sp       app
  restart,rs    env,e
  restage,rg    scale

Services integration:
  marketplace,m        create-user-provided-service,cups
  services,s           update-user-provided-service,uups
  create-service,cs    create-service-key,csk
  update-service       delete-service-key,dsk
  delete-service,ds    service-keys,sk
  service              service-key
  bind-service,bs      bind-route-service,brs
  unbind-service,us    unbind-route-service,urs

Route and domain management:
  routes,r        delete-route    create-domain
  domains         map-route
  create-route    unmap-route
```

Spring Tool Suite (STS)



Pivotal Web Services

Search by App Name press /

adityahrudayam777@gmail.com

ORG

MST

SPACES

development

Marketplace

Docs

Support

Tools

Blog

Status

SPACE

development

RUNNING 0 STOPPED 8 CRASHED 0

Apps (8) Services (3) Routes (15) Members (2) Settings

Apps

Status	Name	Instances	Memory	Last Push	Route
Stopped	agency	1	1 GB	4 months ago	https://agency777.cfapps.io
Stopped	company	1	1 GB	4 months ago	https://company777.cfapps.io
Stopped	cook	1	1 GB	4 months ago	https://cookex111.cfapps.io
Stopped	day4.springboot	1	1 GB	2 months ago	https://day4ex.cfapps.io
Stopped	greeter	1	1 GB	4 months ago	https://greeter777.cfapps.io
Stopped	message-generation	1	1 GB	4 months ago	https://message-generation777.cfapps.io
Stopped	simpleExample	1	1 GB	2 months ago	https://simpleExample777.cfapps.io

Operator

Install runtime and container

Install services (db, messaging, hadoop, ...)

Setup load-balancing, SSL termination and dynamic routing

Operator

Setup / config High Availability

Setup APM

Is it Ops Team friendly??

Operator

Click to install

No downtime updates

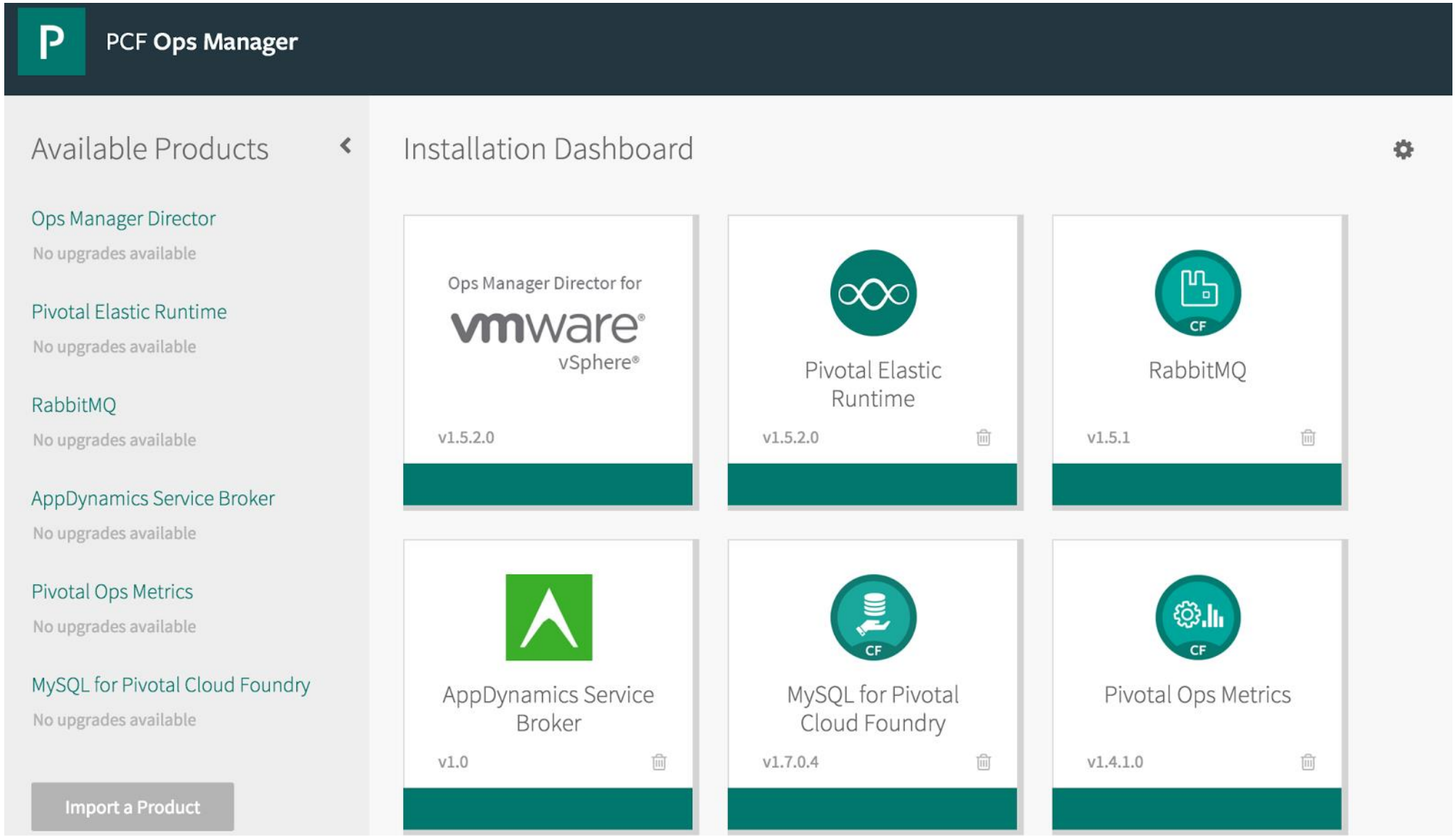
Explore install logs

Click to scale the platform

Built-in High Availability

Built-in Platform Monitoring

Integrated services


PCF Ops Manager
Available Products
Ops Manager Director
No upgrades available
Pivotal Elastic Runtime
No upgrades available
RabbitMQ
No upgrades available
AppDynamics Service Broker
No upgrades available
Pivotal Ops Metrics
No upgrades available
MySQL for Pivotal Cloud Foundry
No upgrades available
Import a Product
Installation Dashboard
Ops Manager Director for
vmware
vSphere
v1.5.2.0
Pivotal Elastic Runtime
v1.5.2.0
CF
RabbitMQ
v1.5.1
AppDynamics Service Broker
v1.0
CF
MySQL for Pivotal Cloud Foundry
v1.7.0.4
CF
Pivotal Ops Metrics
v1.4.1.0

pcf Command Line Utility

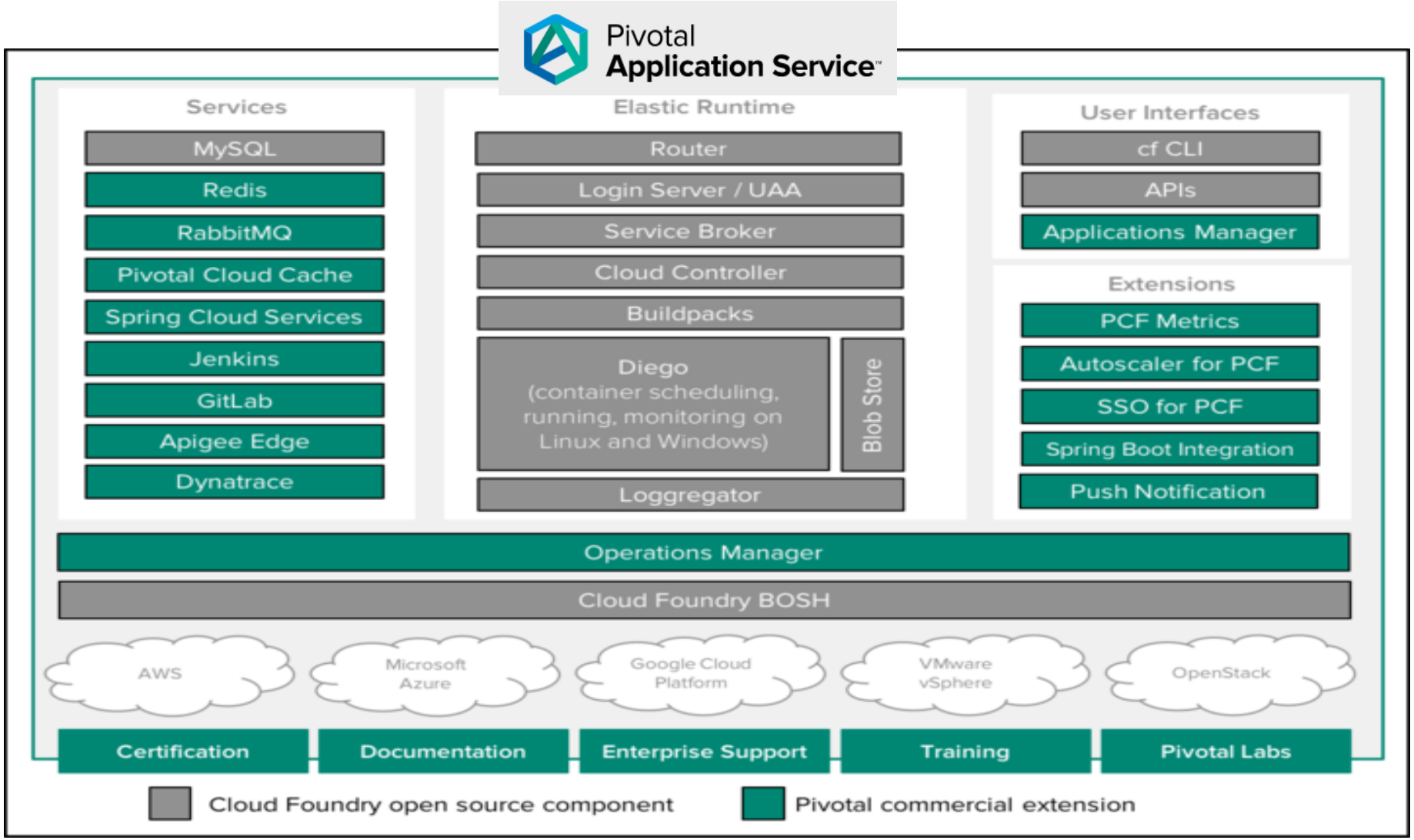
```
$ pcf --help

Usage: pcf [OPTIONS] COMMAND [ARGS]...

Options: --help Show this message and exit.

Commands:
apply-changes
cf-info
changes
configure
delete-unused-products
import
install
is-available
is-installed
logs
products
settings
target
test-errand
uninstall
```

PCF – Architecture or Commercialization Model

Pivotal
Cloud Foundry®

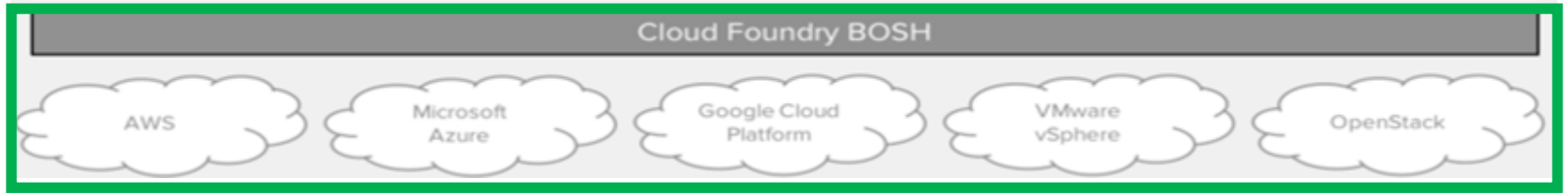
Continuously deliver any app to every major private and public cloud with a single platform.

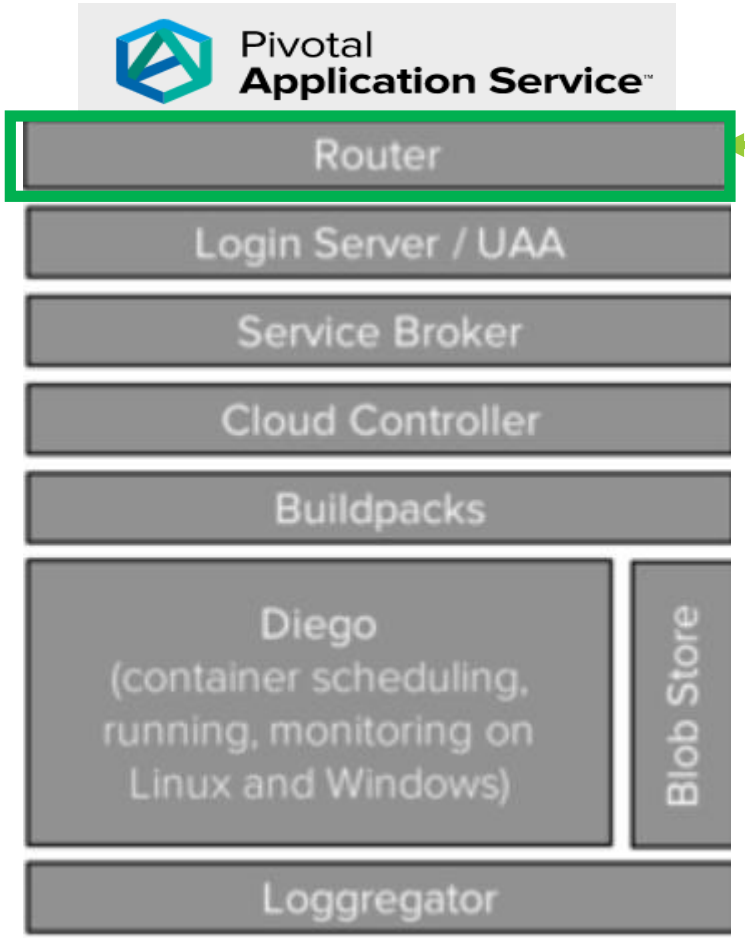
PCF - Commercialization Model



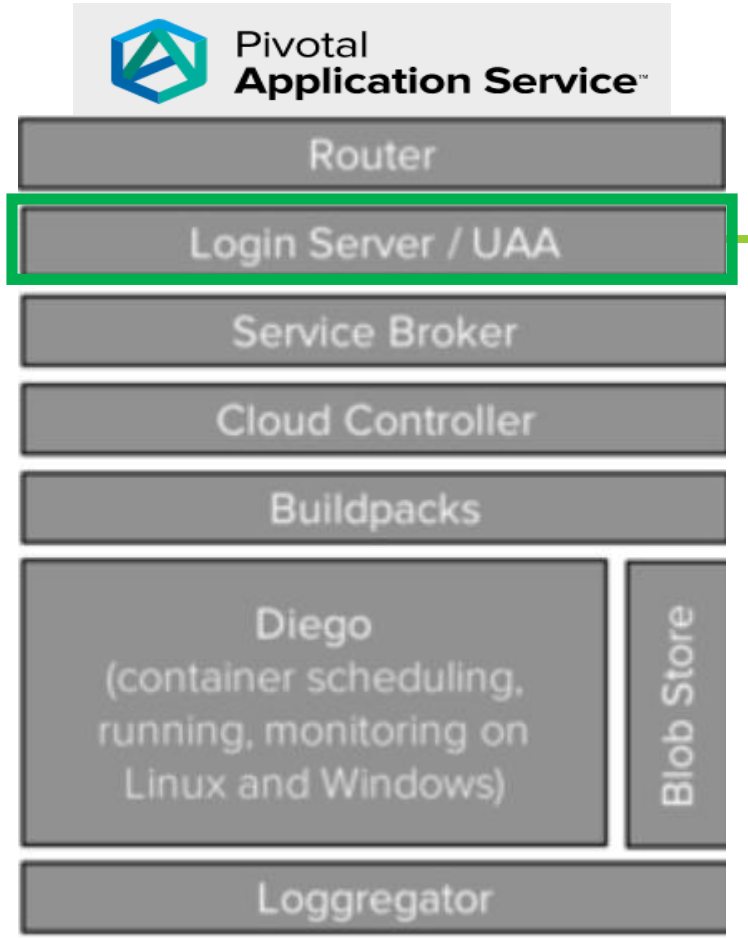
PCF Components:

BOSH creates and deploys virtual machines (VMs) on top of a physical computing infrastructure, and deploys and runs Cloud Foundry on top of this cloud. To configure the deployment, BOSH follows a manifest document.

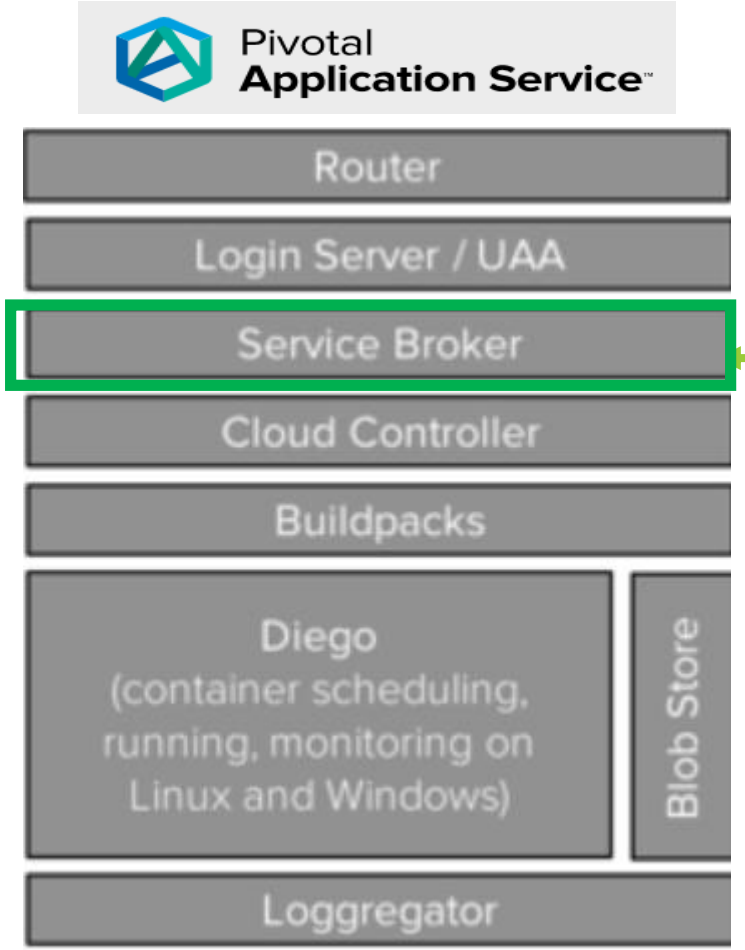




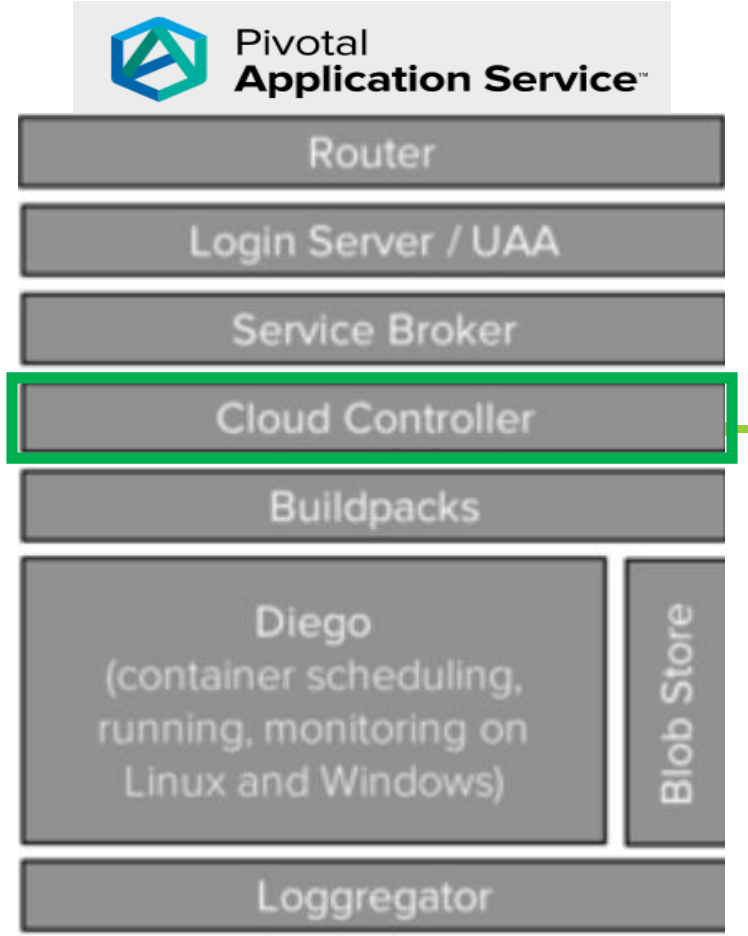
The **Gorouter** routes traffic coming into Cloud Foundry to the appropriate component, whether it is an operator addressing the Cloud Controller or an application user accessing an app running on a Diego Cell.



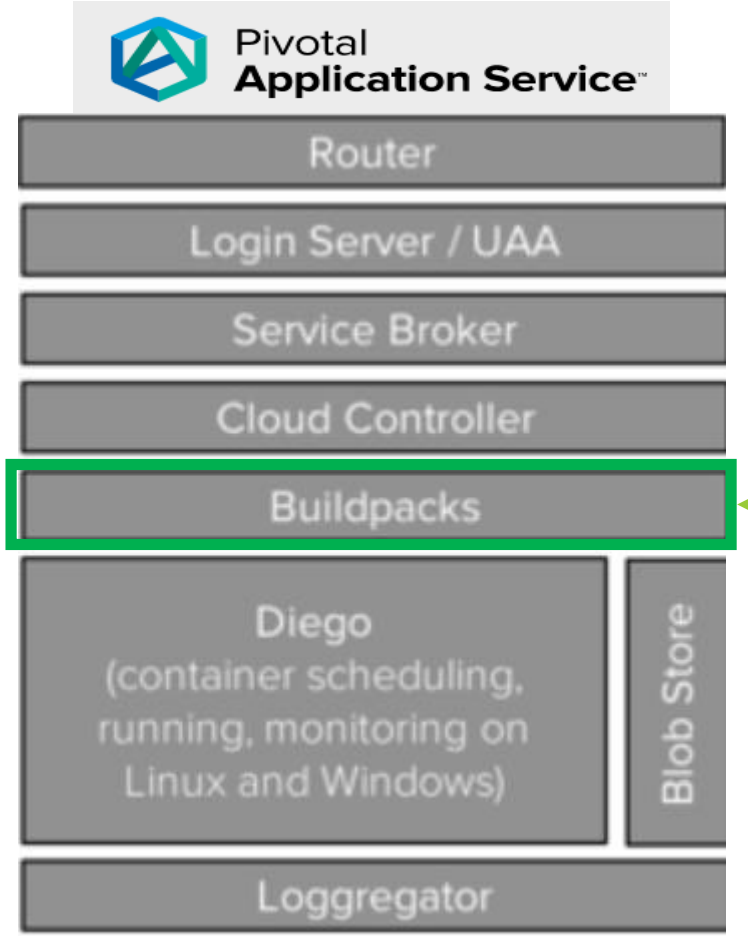
Login Server / "User Authorization and Authentication" provides identity, security and authorization services. It manages third party Oauth 2.0 access credentials and can provide application access and identity-as-a-service for apps running on Cloud Foundry.



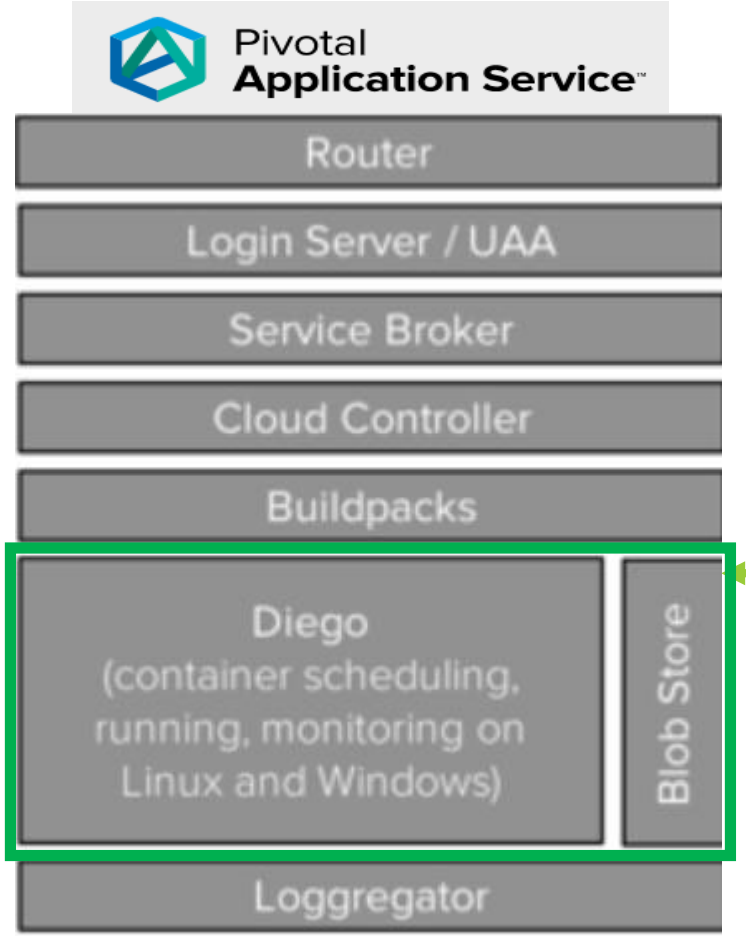
Service Brokers provide an interface for native and external 3rd party services. Service processes run on Service Nodes or with external as-a-service providers (e.g., email, database, messaging, etc.).



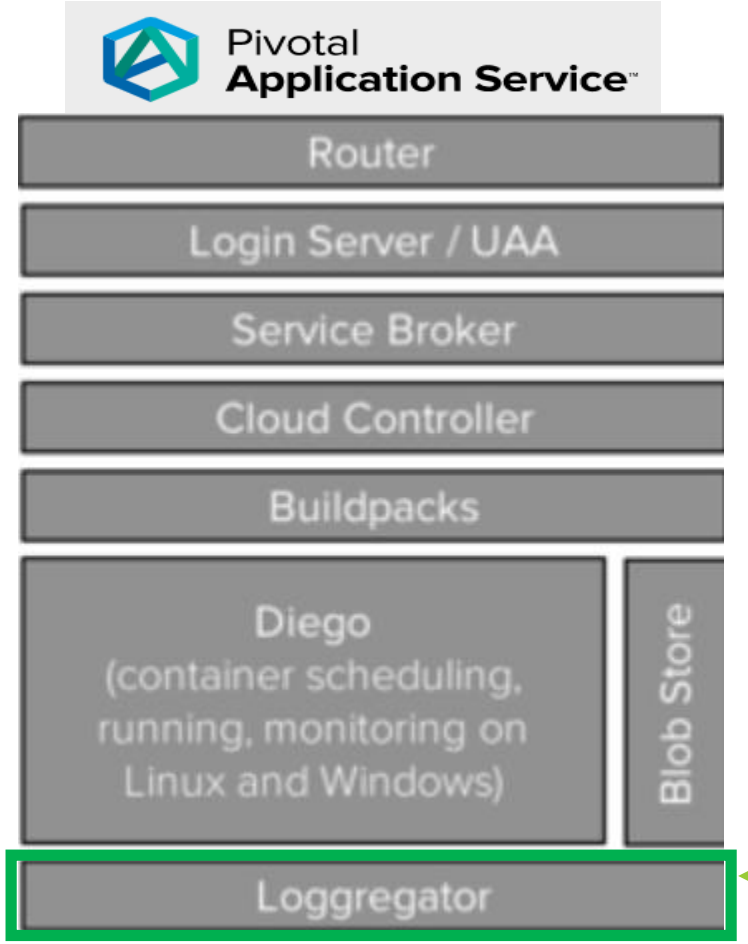
The CF **Cloud Controller** runs the apps and other processes on the cloud's VMs, balancing demand and managing app lifecycles.



Buildpacks provide framework and runtime support for apps. Buildpacks typically examine the apps to determine what dependencies to download and how to configure the apps to communicate with bound services.



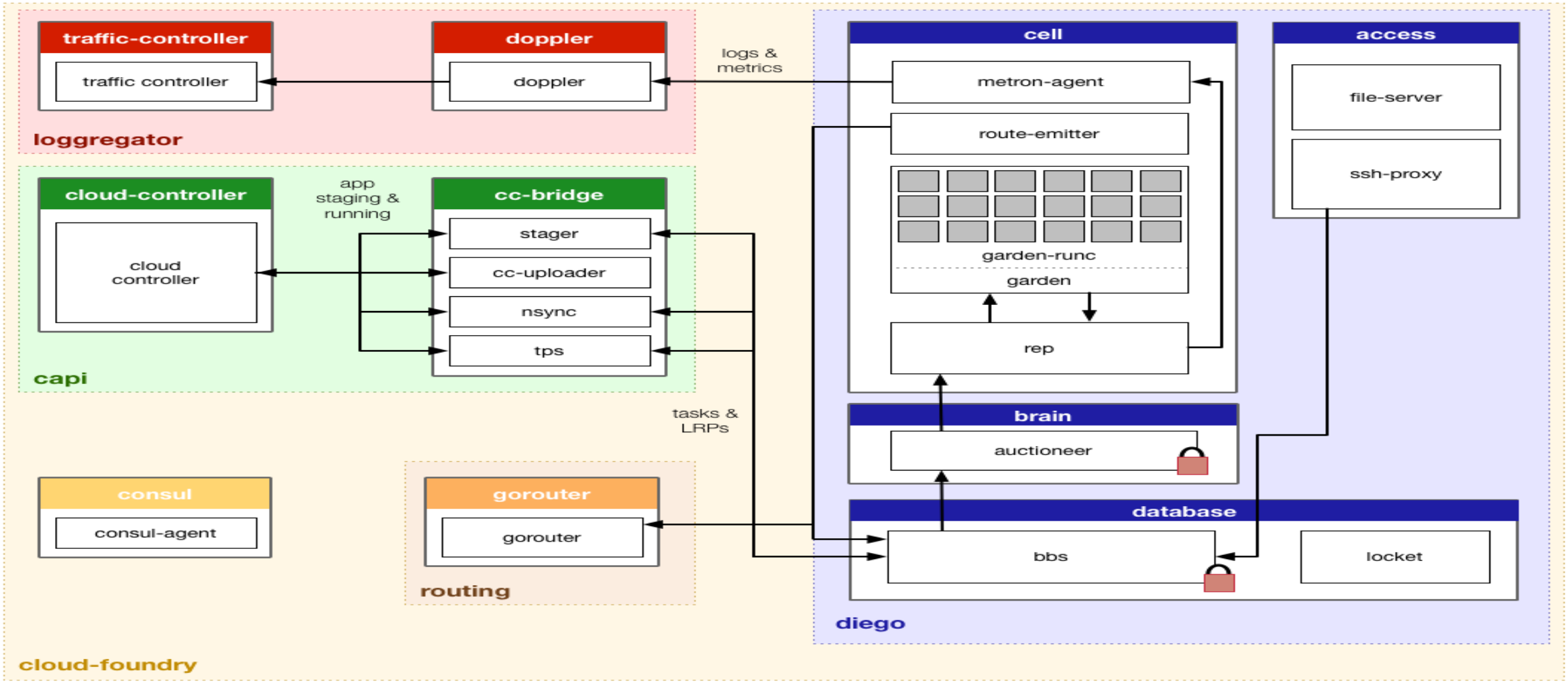
Diego are secure and fully isolated containers. It is responsible for an Apps lifecycle: building, starting and stopping Apps as instructed.



Loggregator gathers and streams logs and metrics from user apps in a Pivotal Cloud Foundry (PCF) deployment as well as metrics from PCF components.

Diego Components and Architecture

A distributed system that orchestrates containerized workloads



Inigo
Integration Tests

Vizzini
Diego Acceptance Tests

CATS
CF Acceptance Tests

buildpack-app-lifecycle

builder	launcher
health check	diego sshd

docker-app-lifecycle

builder	launcher
health check	diego sshd

Diego in Cloud Foundry

cf push

Cloud Controller

http://

Gorouter

Container Placement & Life Cycle

Diego Core

bbs

rep

auctioneer

Container Execution

garden

Consul/locket

Component Discovery & coordination

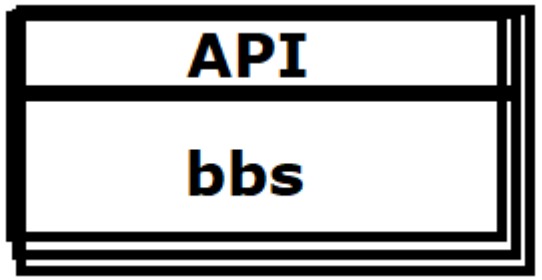
BBS uses Mysql

A Consul server stores longer-lived control data, such as component IP addresses and distributed locks that prevent components from duplicating actions.

BBS stores more frequently updated and disposable data such as cell and application status, unallocated work, and heartbeat message

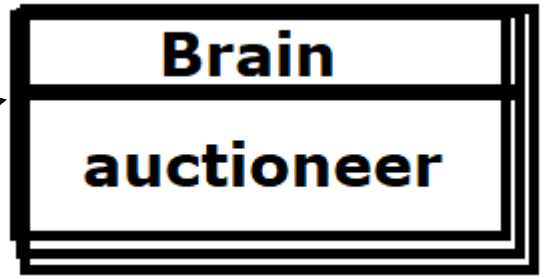
Diego Core Components

- Public API for clients
- Enforce lifecycle policy

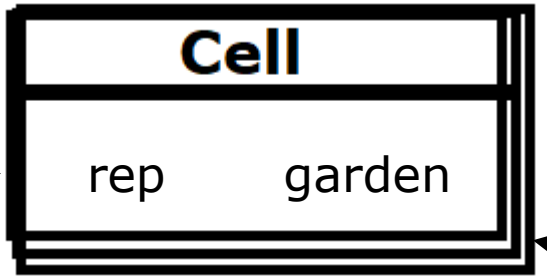


Maintains a real-time representation of the state of the Diego cluster, including all desired LRPs, running LRP instances, and in-flight Tasks

- communicates with Cells
- decide optimal placement

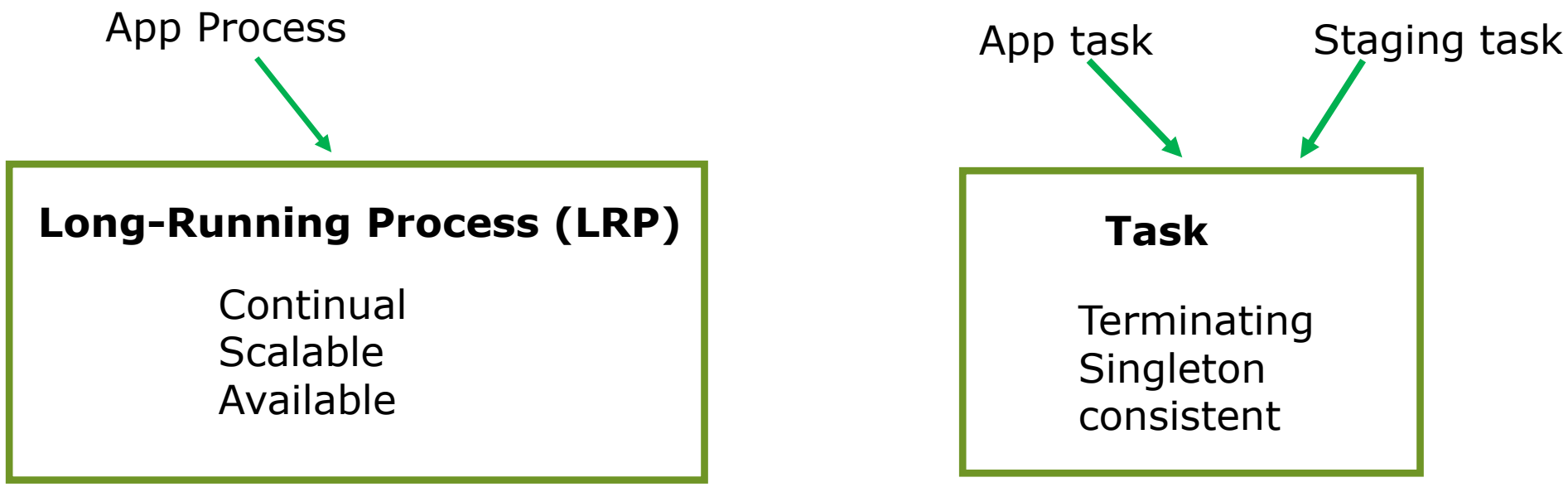


- Broadcasts cell presence
- Controls local garden
- Caches assets

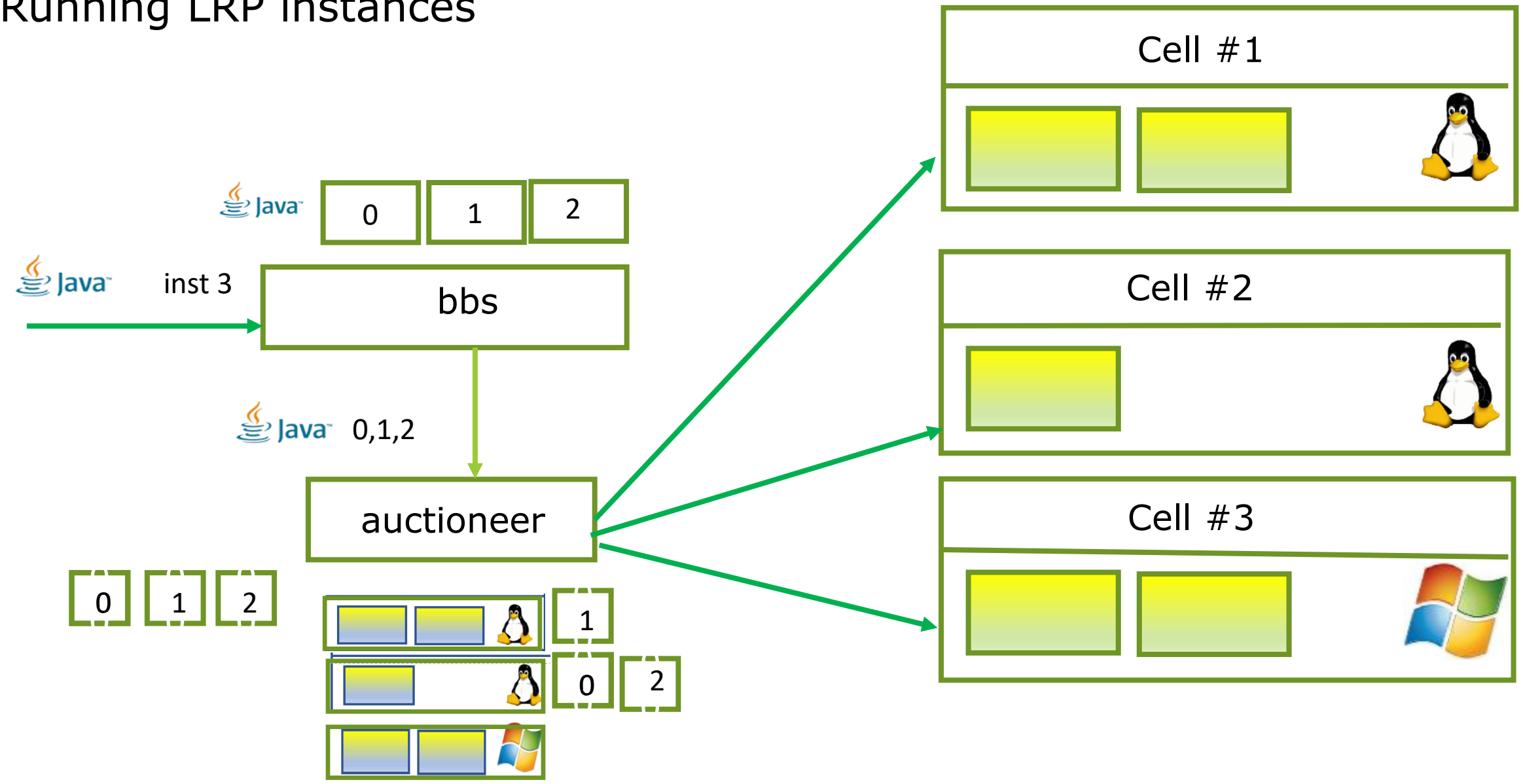


Create containers & run processes

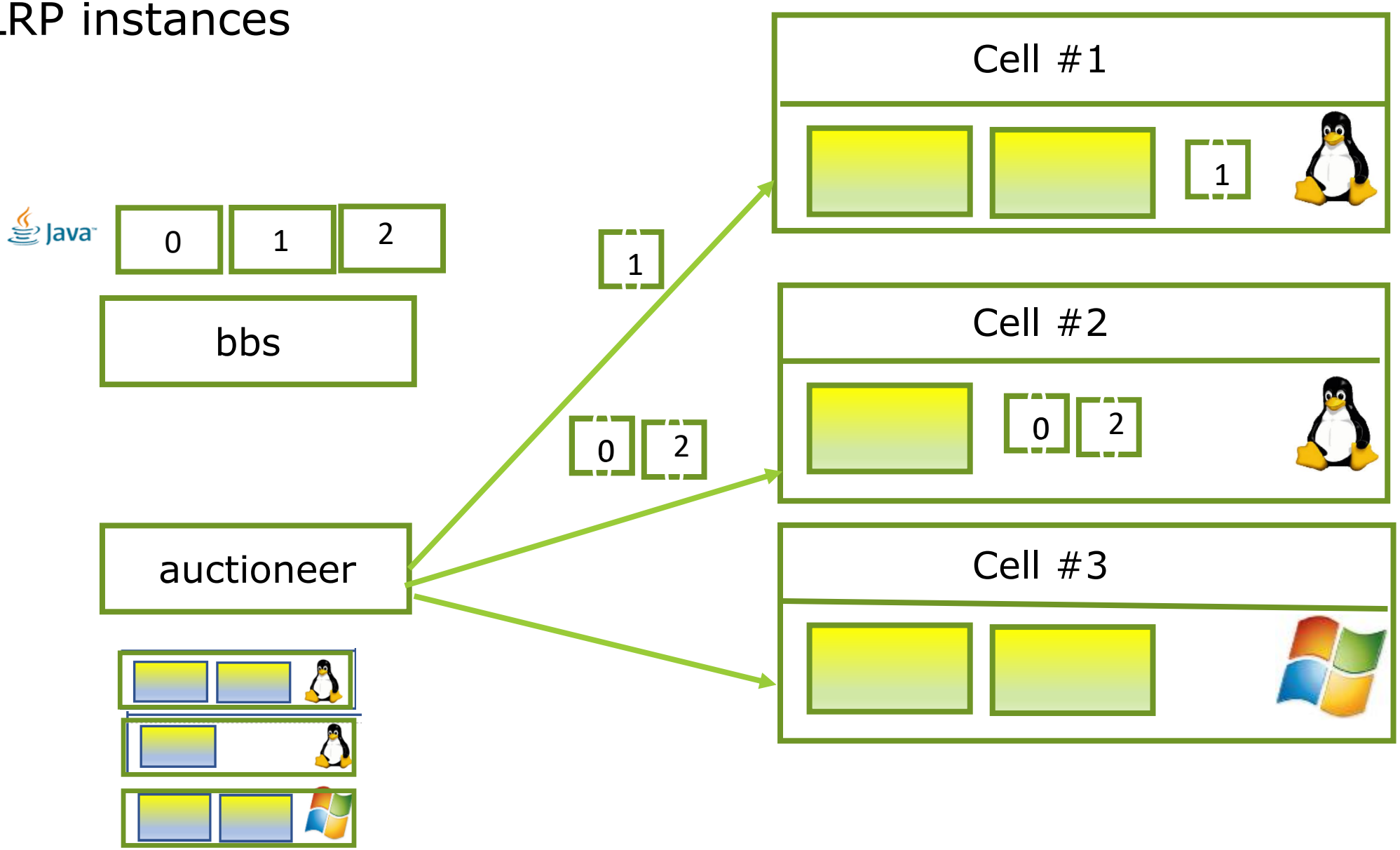
Diego Workload Types



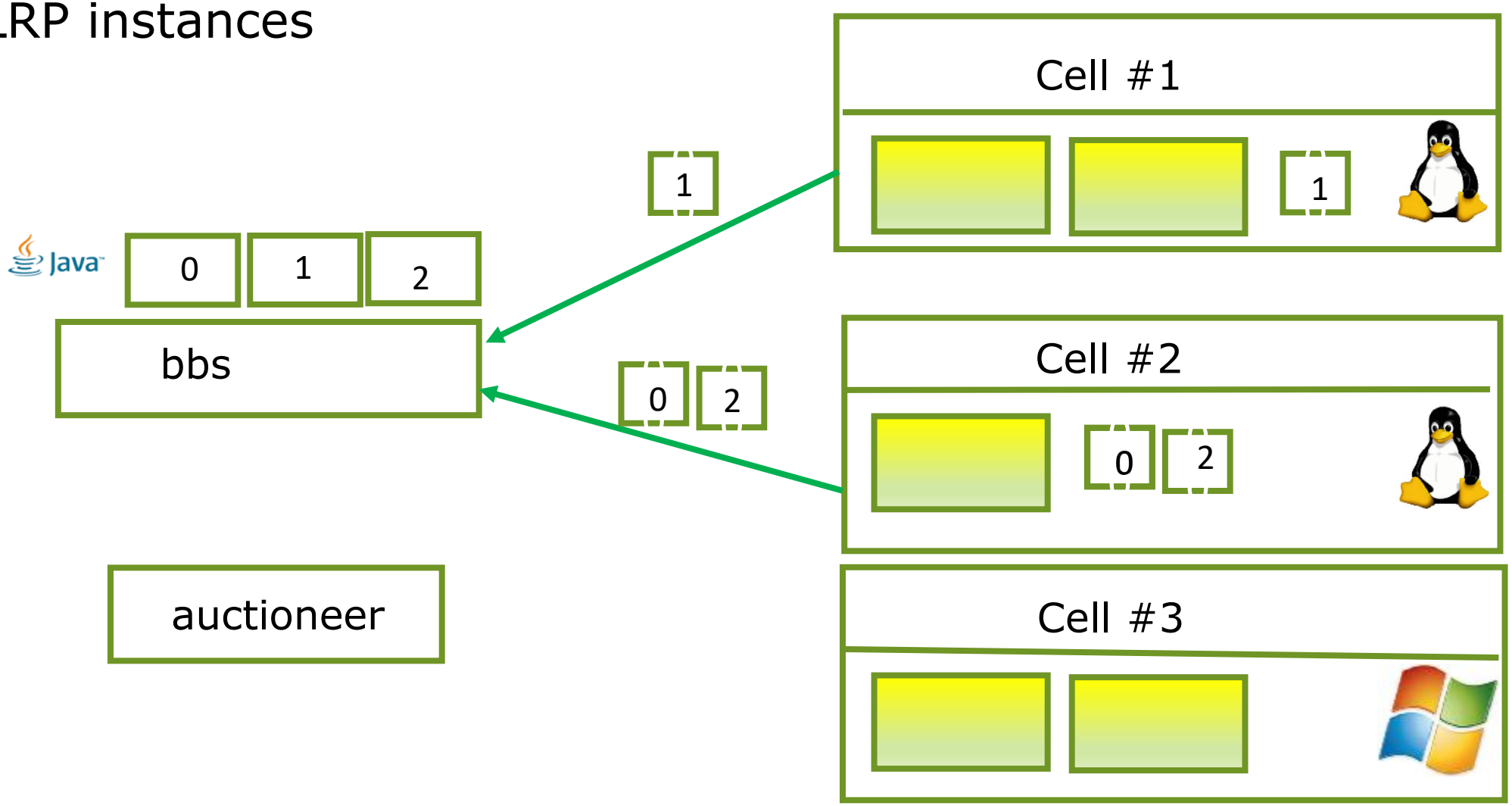
Running LRP instances



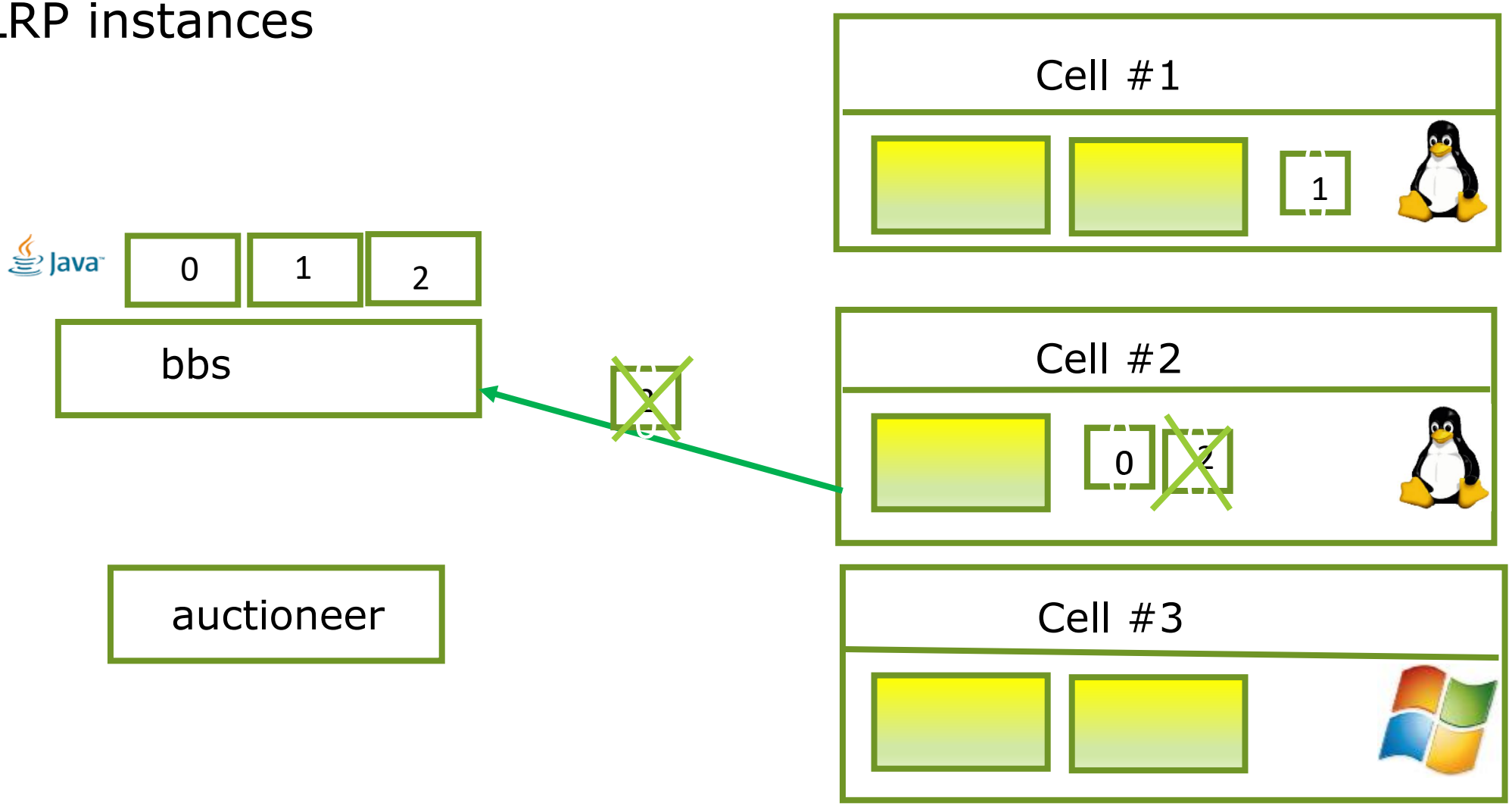
Running LRP instances



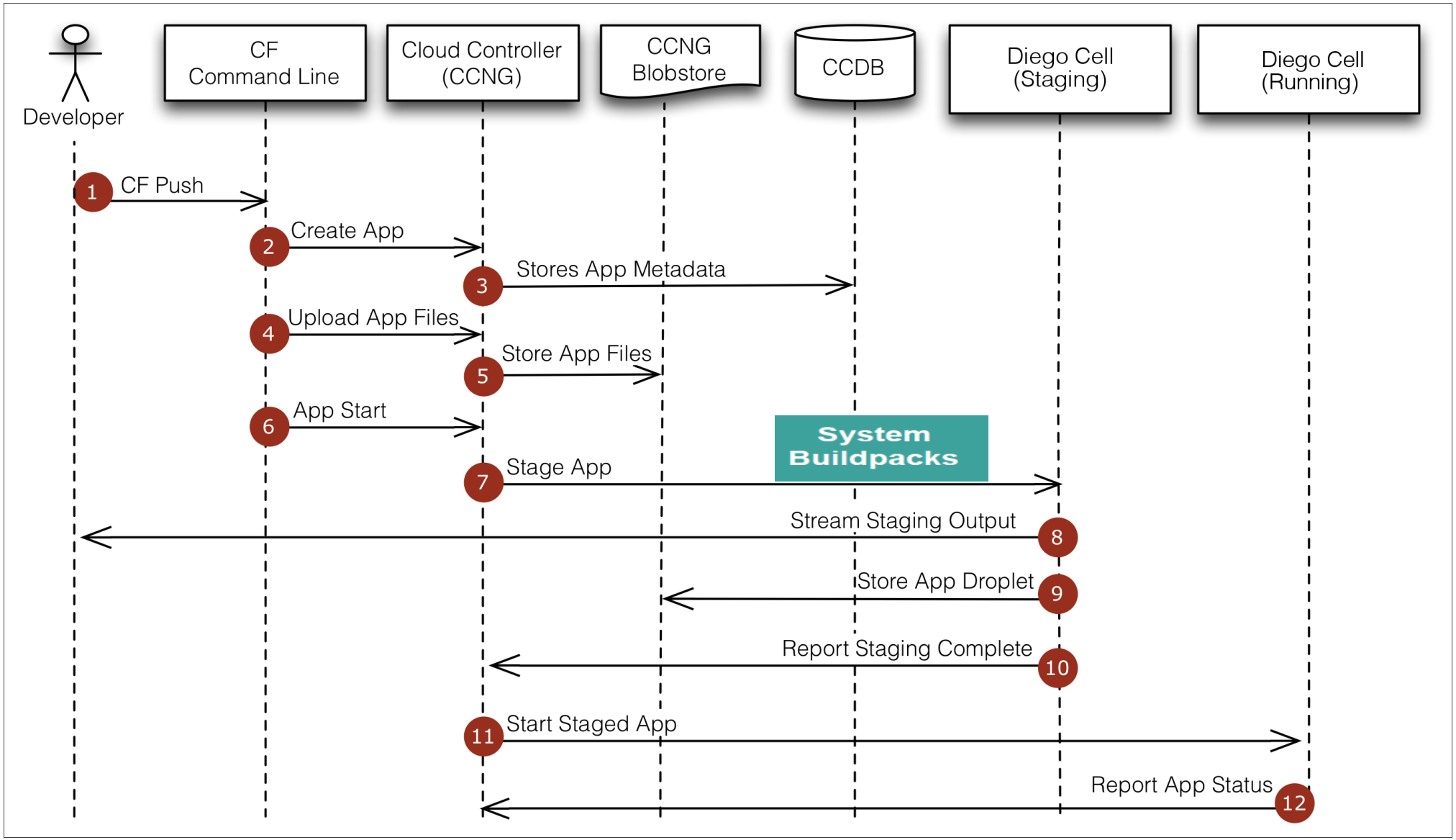
Running LRP instances



Running LRP instances



Staging the Applications


Developer
CF Command Line
Cloud Controller (CCNG)
CCNG Blobstore
CCDB
Diego Cell (Staging)
Diego Cell (Running)
1 CF Push
2 Create App
3 Stores App Metadata
4 Upload App Files
5 Store App Files
6 App Start
System Buildpacks
7 Stage App
8 Stream Staging Output
9 Store App Droplet
10 Report Staging Complete
11 Start Staged App
12 Report App Status

1). At the command line, the developer enters the directory containing the application source code and uses the Cloud Foundry Command Line Interface (cf CLI) to issue a push command.

2). The cf CLI tells the Cloud Controller to create a record for the application.

3). The Cloud Controller stores the application metadata. Application metadata can include the app name, number of instances the user specified, and the buildpack, and other information about the application.

4). Before uploading all the application source files, the cf CLI issues a resource match request to the Cloud Controller to determine if any of the application files already exist in the resource cache.
The uploaded application files are combined with the files from the resource cache to create the application package.

5). The Cloud Controller stores the application package in the blobstore.

6). The cf CLI issues an app start command.

7). The Cloud Controller issues a staging request to Diego, which then schedules a Diego cell (“Cell”) to run the staging task (“Task”). The Task downloads buildpacks and the app’s buildpack cache, if present. It then uses the buildpack that is detected automatically or specified with the -b flag to compile and stage the application.

8). The Cell streams the output of the staging process so the developer can troubleshoot application staging problems.

9). The Task packages the resulting compiled and staged application into a tarball called a “droplet” and the Cell stores the droplet in the blobstore. The Task also uploads the buildpack cache to the blobstore for use the next time the application is staged.

10). The Diego Bulletin Board System reports to the Cloud Controller that staging is complete. Staging must complete within 15 minutes or the staging is considered failed. Apps are given a minimum of 1GB memory to stage, even if the requested running memory is smaller.

11). Diego schedules the application as a Long Running Process on one or more Diego cells.

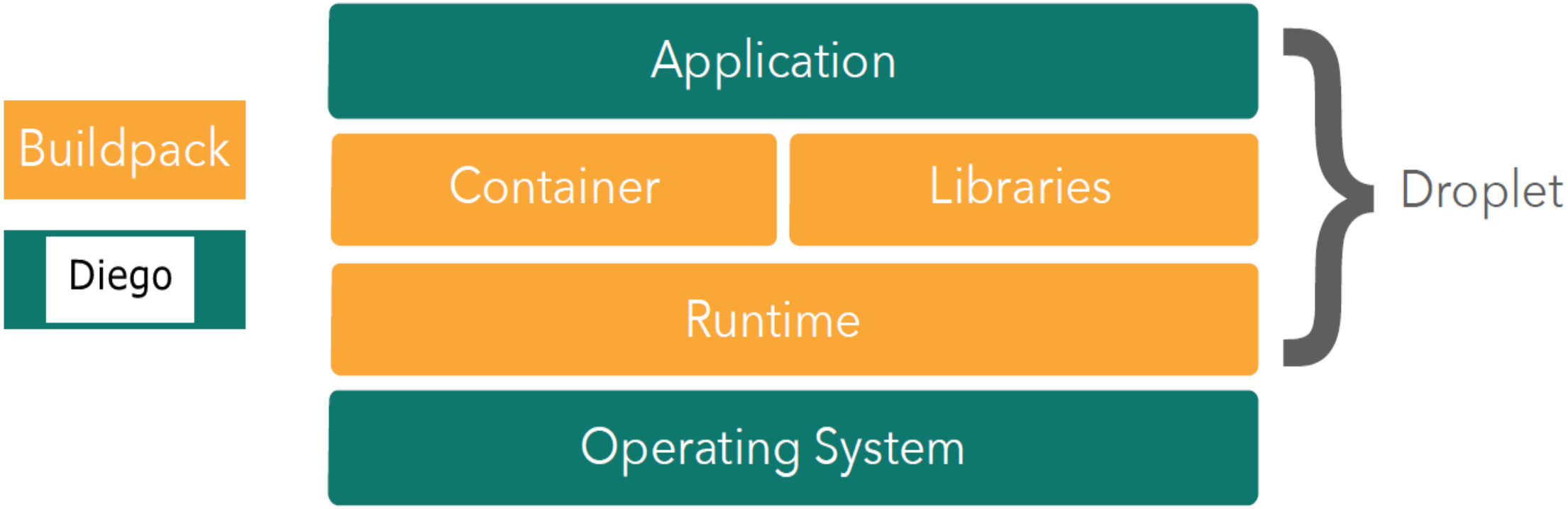
12). The Diego cells report the status of the application to the Cloud Controller.

Buildpacks

Buildpacks

- Buildpacks provide framework and runtime support for apps.
- Buildpacks typically examine your apps to determine what dependencies to download and how to configure the apps to communicate with bound services.
- When we push an app, Cloud Foundry automatically detects an appropriate buildpack for it.
- This buildpack is used to compile or prepare the app for launch.

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



System Buildpacks

Cloud Foundry includes a set of system buildpacks for common languages and frameworks.

This table lists the system buildpacks.

Name	Supported Languages, Frameworks, and Technologies	GitHub Repository
Binary	*n/a*	Binary source
Go	Go	Go source
Java	Grails, Play, Spring, or any other JVM-based language or framework	Java source
.NET Core	.NET Core	.NET Core source
Node.js	Node or JavaScript	Node.js source
PHP	Cake, Symfony, Zend, Nginx, or HTTPD	PHP source
Python	Django or Flask	Python source
Ruby	Ruby, JRuby, Rack, Rails, or Sinatra	Ruby source
Staticfile	HTML, CSS, JavaScript, or NGINX	Staticfile source
NGINX	NGINX	NGINX source

Community Buildpacks

We can find a list of unsupported, community-created buildpacks here:

https://github.com/cloudfoundry-community/cf-docs-contrib/wiki/Buildpacks#community-created

Managing Buildpacks

```
$ cf buildpacks

$ cf create-buildpack <name> <path to bits> <position>

$ cf update-buildpack <name> [-p <path>] [-i <position>]

$ cf delete-buildpack <name>
```

Buildpack commands

```
$ cf push
```

The application is tested against admin then system buildpacks.

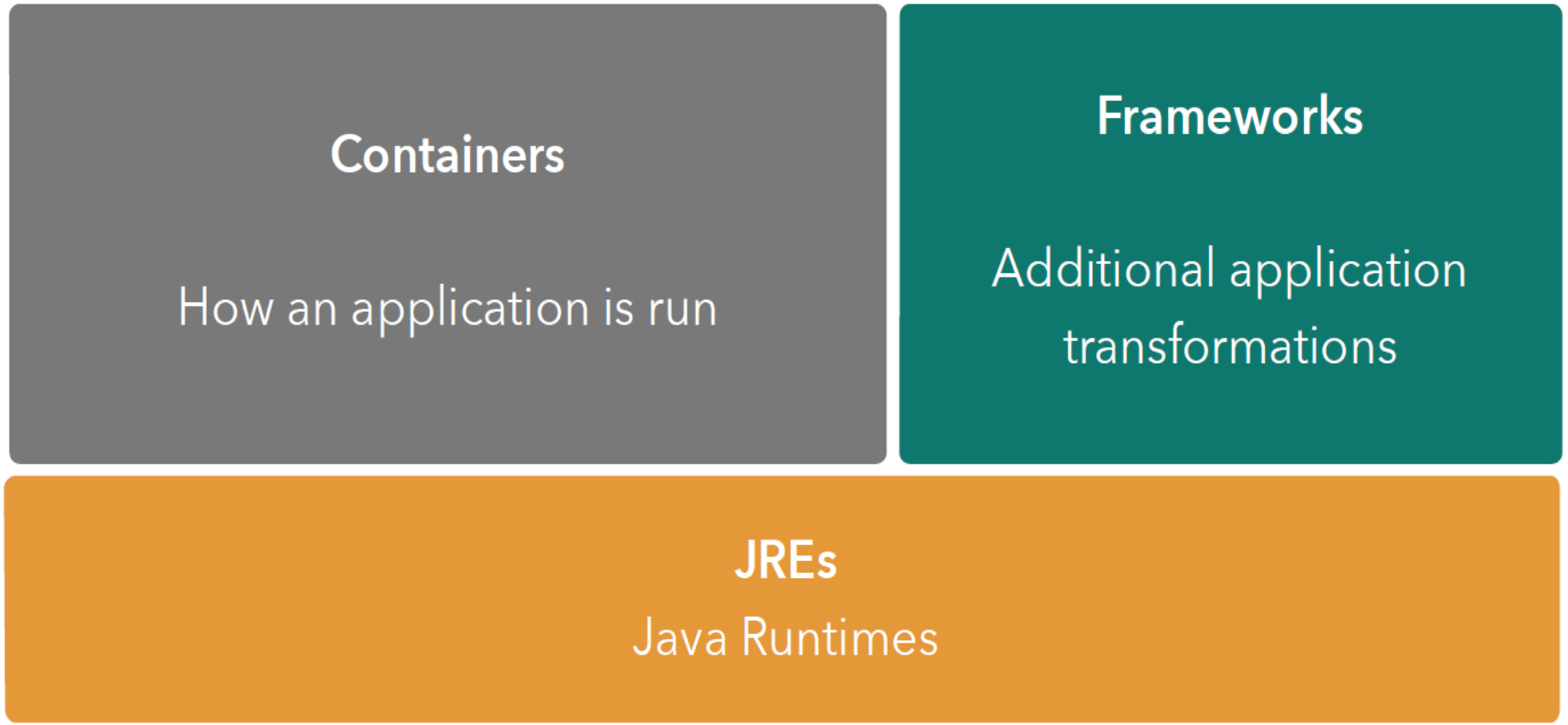
```
$ cf push -b
<url>
```

The buildpack is referenced by a Git URL

Java Buildpack

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

Java Buildpack Concepts



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

Scaling an Application Using cf scale(10 mts)

Scaling Horizontally

Horizontally scaling an application creates or destroys instances of your application.

Incoming requests to your application are automatically load balanced across all instances of your application, and each instance handles tasks in parallel with every other instance.

Adding more instances allows your application to handle increased traffic and demand.

```
$ cf scale myApp -i 5
```

Scaling Vertically

Vertically scaling an application changes the disk space limit or memory limit that Cloud Foundry applies to all instances of the application.

```
$ cf scale myApp -k 512M

$ cf scale myApp -m 1G
```

-k DISK
-m MEMORY

App Autoscaler Overview

- App Autoscaler is a marketplace service that ensures app performance and helps control the cost of running apps.
- To balance app performance and cost, Space Developers and Space Managers can use App Autoscaler to do the following:
- Configure rules that adjust instance counts based on metrics thresholds such as CPU Usage
- Modify the maximum and minimum number of instances for an app, either manually or following a schedule
- $ cf create-service **App Autoscaler**

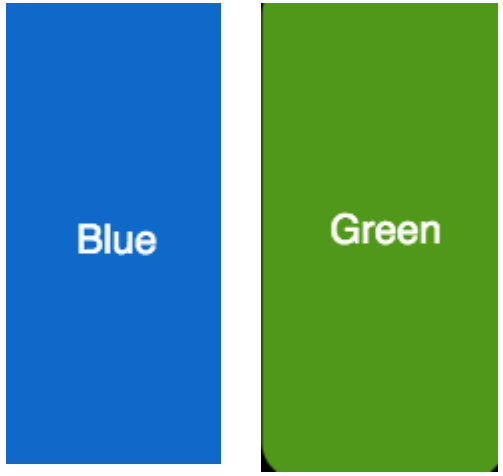
Reduce Downtime and Risk(15 mts)

"Blue Green Deployment"


ONLINE
APPLICATION


UPDATE

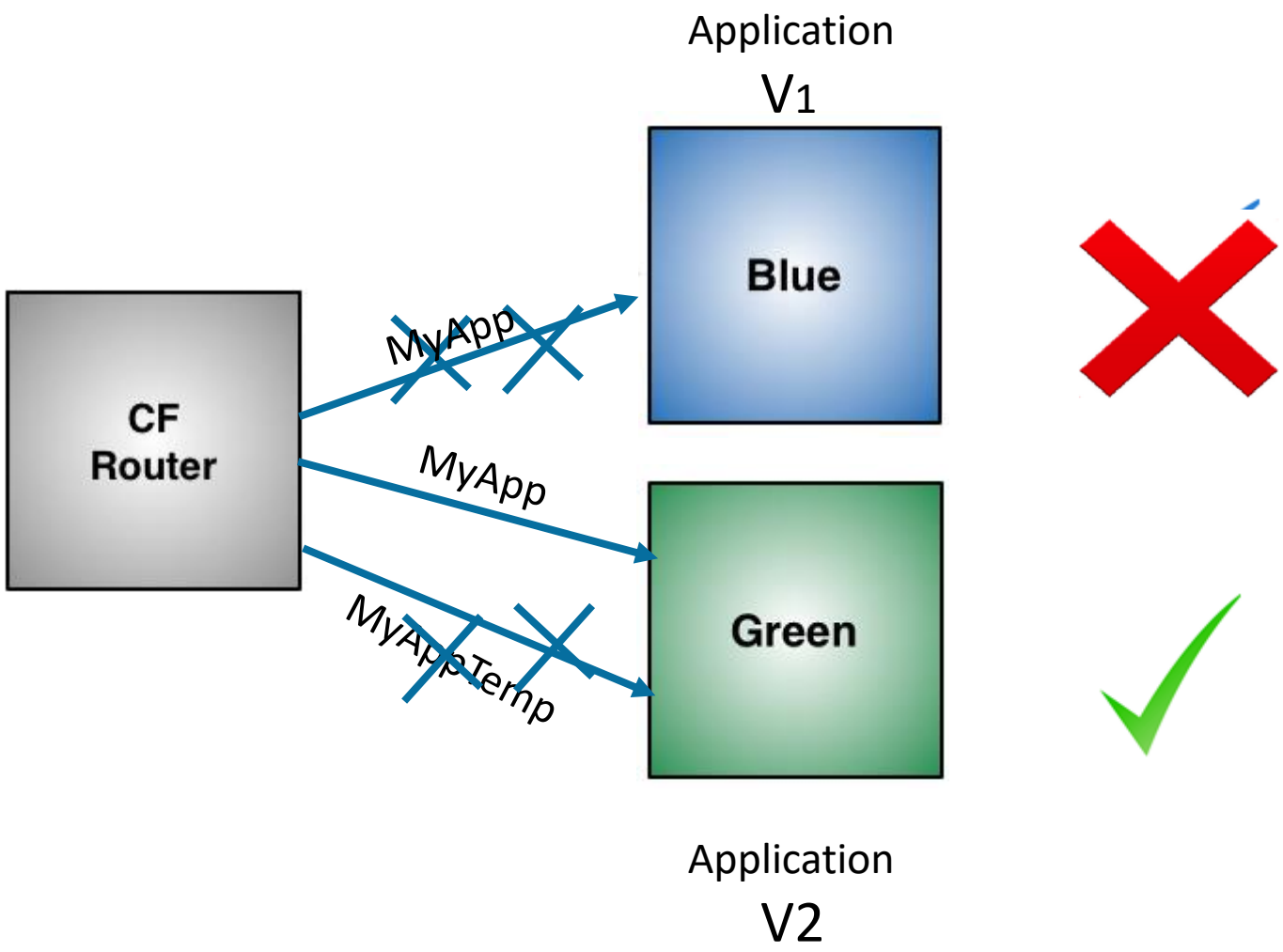

HOW
TO
LIVE
Downtime

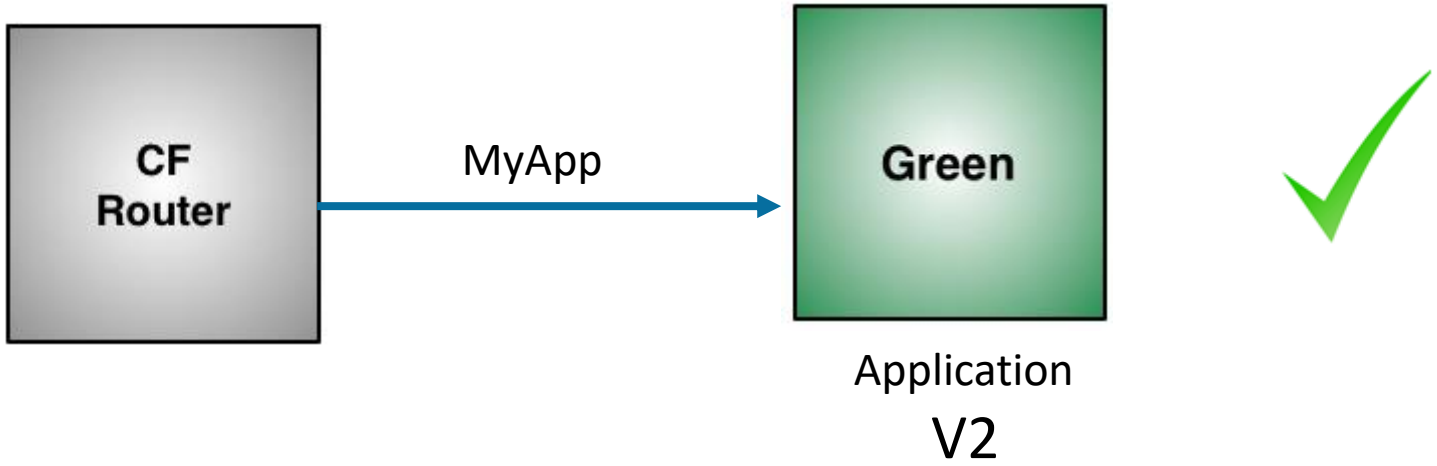


DEPLOYMENT

- Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.

- At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.


Application
V1
Blue
CF
Router
MyApp
MyApp
MyAppTemp
Green
Application
V2


CF
Router
MyApp
Green
Application
V2

Automate Implementations

There are plugins to automate the blue-green deployment process. These include:

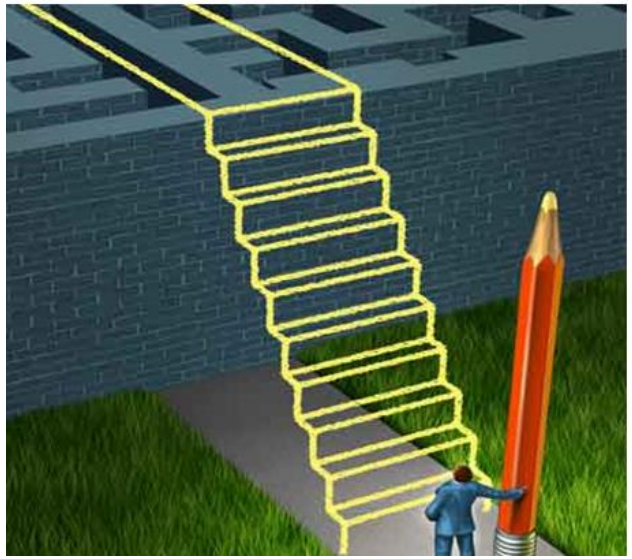
Autopilot: Autopilot is a Cloud Foundry Go plugin that provides a subcommand, zero-downtime-push, for hands-off, zero-downtime application deploys.

BlueGreenDeploy: cf-blue-green-deploy is a plugin, written in Go, for the Cloud Foundry Command Line Interface (cf CLI) that automates a few steps involved in zero-downtime deploys.

PCF Dev/ BOSH Lite

I got some knowledge in this tech talk!!

Time to practice. But, I think….



I need high-speed internet!!

PCF Account!!

Developer

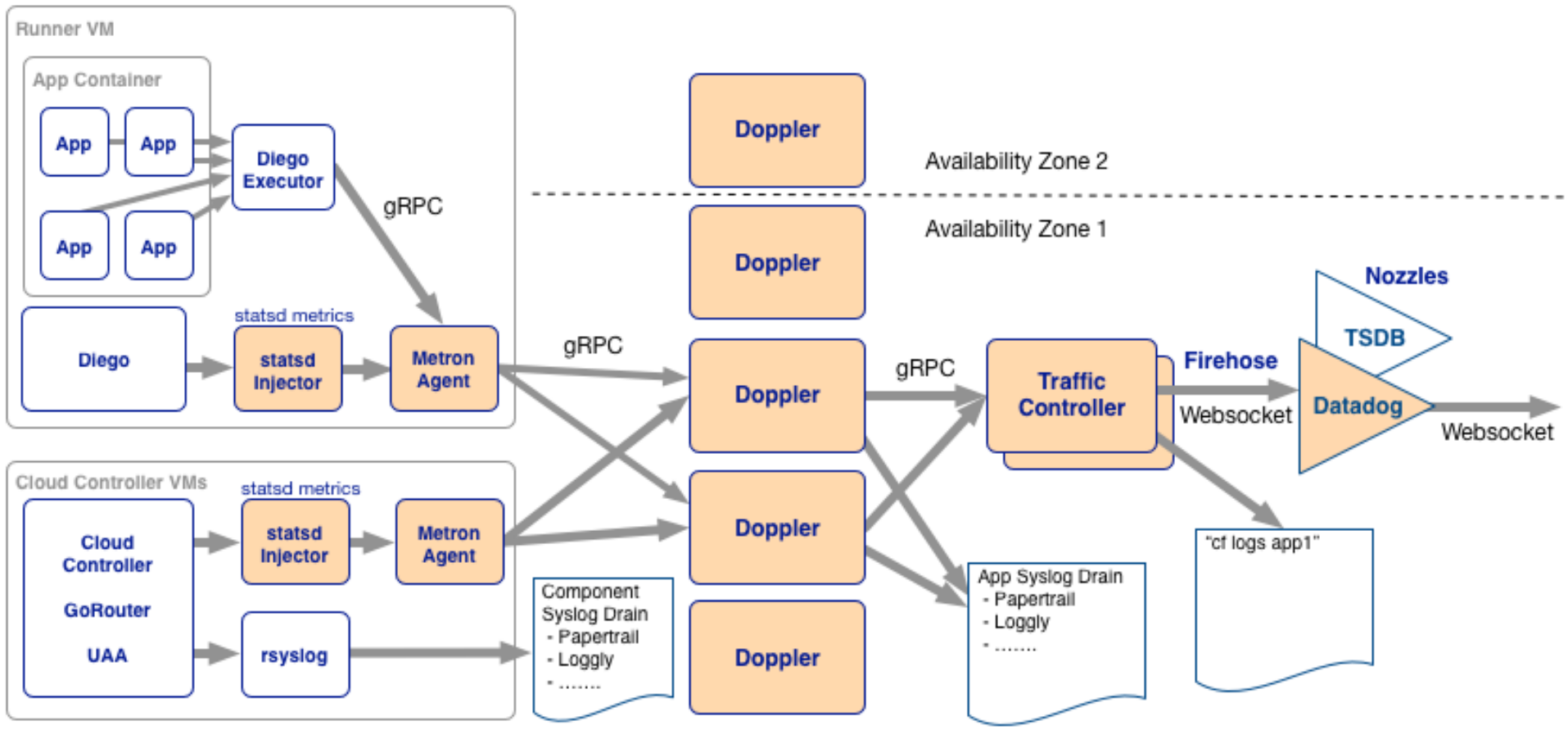
PCF Dev is a small footprint distribution of Pivotal Cloud Foundry (PCF) intended to be run locally on a developer machine.

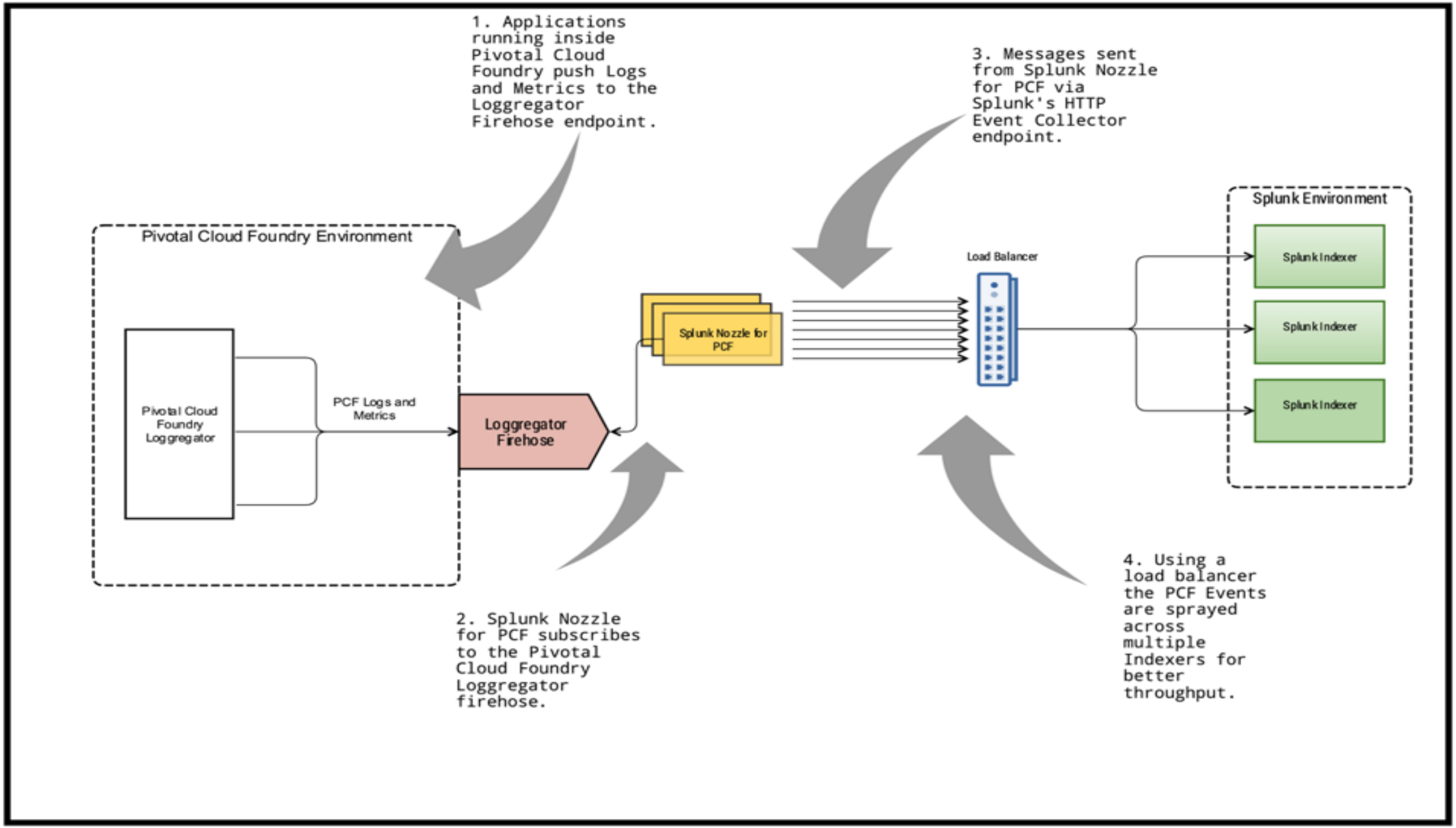
It delivers the essential elements of the Pivotal Cloud Foundry experience quickly through a condensed set of components.

Operator

BOSH Lite v2 is a Director VM running in VirtualBox (locally)

Splunk Firehose Nozzle for PCF


Runner VM
App Container
App
App
App
App
Diego Executor
gRPC
Diego
statsd metrics
statsd Injector
Metron Agent
Cloud Controller VMs
statsd metrics
Cloud Controller
GoRouter
UAA
statsd Injector
Metron Agent
rsyslog
Component Syslog Drain
- Papertrail
- Loggly
- .....
Doppler
Doppler
Doppler
Doppler
Doppler
Availability Zone 2
Availability Zone 1
gRPC
gRPC
Traffic Controller
Firehose
Websocket
Nozzles
TSDB
Datadog
Websocket
App Syslog Drain
- Papertrail
- Loggly
- .....
"cf logs app1"


1. Applications running inside Pivotal Cloud Foundry push Logs and Metrics to the Loggregator Firehose endpoint.
3. Messages sent from Splunk Nozzle for PCF via Splunk's HTTP Event Collector endpoint.
Pivotal Cloud Foundry Environment
Pivotal Cloud Foundry Loggregator
PCF Logs and Metrics
Loggregator Firehose
Splunk Nozzle for PCF
Load Balancer
Splunk Environment
Splunk Indexer
Splunk Indexer
Splunk Indexer
2. Splunk Nozzle for PCF subscribes to the Pivotal Cloud Foundry Loggregator firehose.
4. Using a load balancer the PCF Events are sprayed across multiple Indexers for better throughput.

Splunk Firehose Nozzle for Pivotal Cloud Foundry?

Pivotal Cloud Foundry consolidates application logs and platform components' metrics to a PCF component known as Loggregator. To get events out of PCF and into your Splunk environment, we need a Nozzle that attaches to the Loggregator Firehose.

This is where the Splunk Firehose Nozzle for Pivotal Cloud Foundry comes in!

Splunk Firehose Nozzle connects to the Loggregator Firehose Endpoint and streams all available events into your Splunk environment via the HTTP Event Collector (HEC).

Configuration Steps:

The Splunk Firehose Nozzle for PCF collects events from the PCF Loggregator endpoint and streams them to Splunk via HTTP event collector. Nozzle has in-memory queue buffers to increase reliability, and has parallel client to scale out multiple ingestion channels to HEC.

https://github.com/cloudfoundry-community/splunk-firehose-nozzle/

Pivotal Web Services

Pivotal Web Services

Search by App Name press /

adityahrudayam777@gmail.com

ORG

MST

SPACES

development

Marketplace

Docs

Support

Tools

Blog

Status

SPACE

development

RUNNING 0 STOPPED 8 CRASHED 0

Apps (8) Services (3) Routes (15) Members (2) Settings

Apps

Status	Name	Instances	Memory	Last Push	Route
Stopped	agency	1	1 GB	4 months ago	https://agency777.cfapps.io
Stopped	company	1	1 GB	4 months ago	https://company777.cfapps.io
Stopped	cook	1	1 GB	4 months ago	https://cookex111.cfapps.io
Stopped	day4.springboot	1	1 GB	2 months ago	https://day4ex.cfapps.io
Stopped	greeter	1	1 GB	4 months ago	https://greeter777.cfapps.io
Stopped	message-generation	1	1 GB	4 months ago	https://message-generation777.cfapps.io
Stopped	simpleExample	1	1 GB	2 months ago	https://simpleExample777.cfapps.io

Orgs

An org is a development account that an individual or multiple collaborators can own and use. All collaborators access an org with user accounts. Collaborators in an org share a resource quota plan, applications, services availability, and custom domains.

Spaces

Every application and service is scoped to a space. Each org contains at least one space. A space provides users with access to a shared location for application development, deployment, and maintenance. Each space role applies only to a particular space.

Roles and Permissions

A user can have one or more roles. The combination of these roles defines the user’s overall permissions in the org and within specific spaces in that org.

Admin
Admin Read-Only
Global Auditor (env variables)
Org Managers
Org Auditors
Org Billing Managers

To add or remove users :

ORG/SPACES -> Members

Routes
The Cloud Foundry Gorouter routes requests to applications by associating an app with an address, known as a route. We call this association a mapping.

Domains
Domains indicate to a developer that requests for any route created from the domain will be routed to Cloud Foundry.

This requires DNS to be configured out-of-band to resolve the domain name to the IP address of a load balancer configured to forward requests to the CF routers.

Default domains:

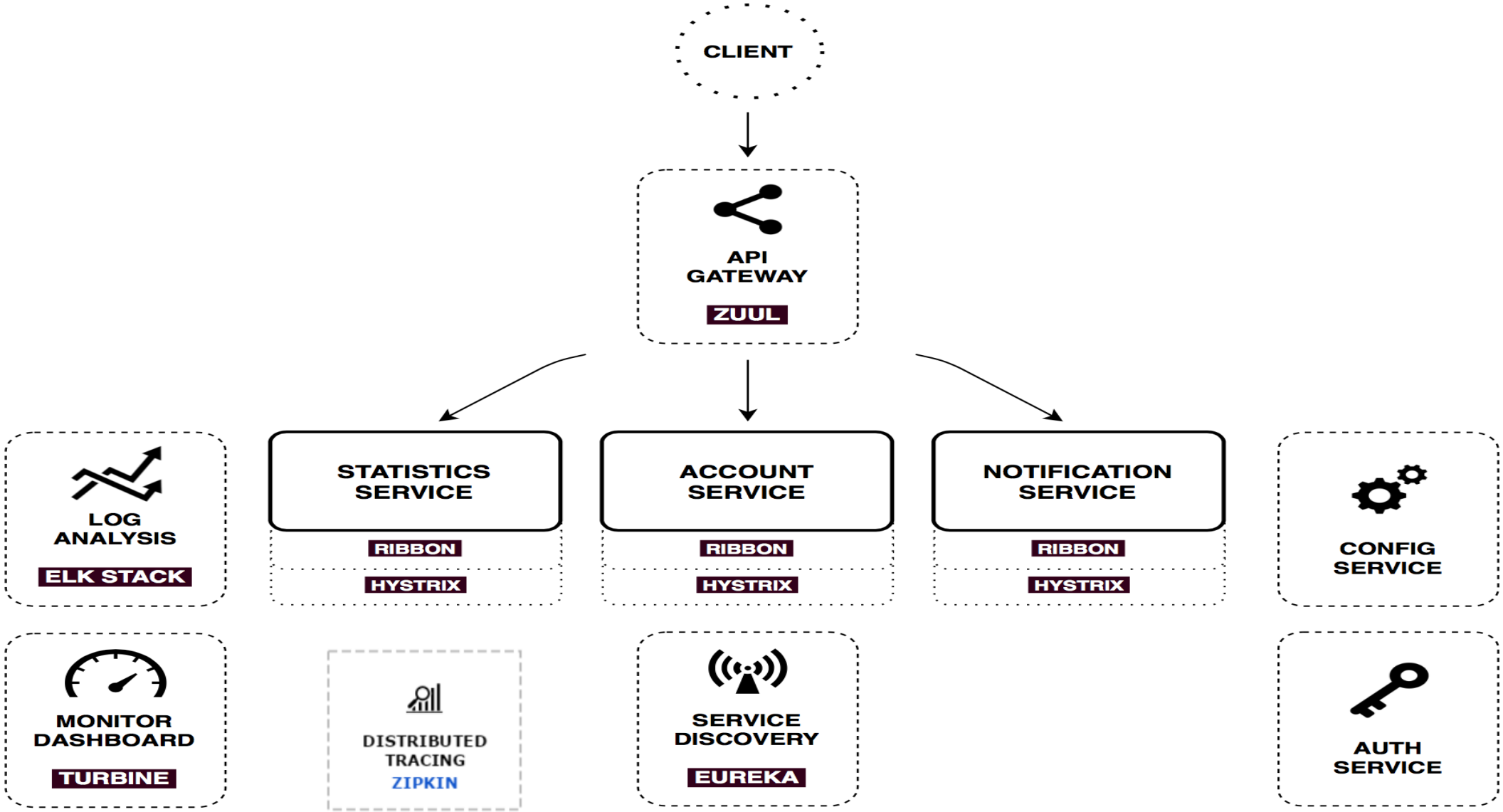
```
cf domains
name            status   type
cfapps.io       shared
cf-tcpapps.io   shared   tcp
```

Spring Cloud Services for PCF

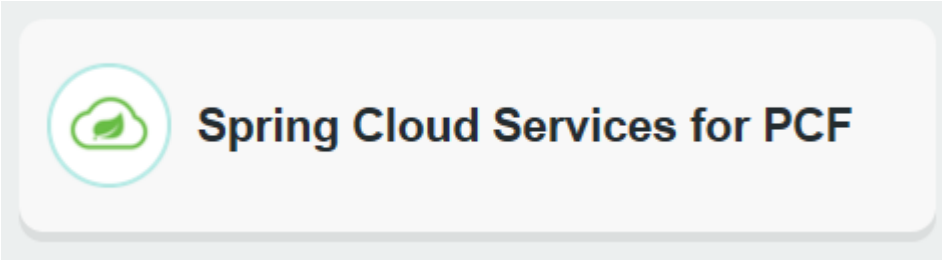
Microservice architecture

is a method of developing software applications as a suite of: independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal.

The microservices style is usually organized around ***business capabilities and priorities.***


CLIENT
API GATEWAY
ZUUL
LOG ANALYSIS
ELK STACK
STATISTICS SERVICE
RIBBON
HYSTRIX
ACCOUNT SERVICE
RIBBON
HYSTRIX
NOTIFICATION SERVICE
RIBBON
HYSTRIX
CONFIG SERVICE
MONITOR DASHBOARD
TURBINE
DISTRIBUTED TRACING
ZIPKIN
SERVICE DISCOVERY
EUREKA
AUTH SERVICE

Spring Cloud Services for PCF



- Spring Cloud (http://projects.spring.io/spring-cloud/) provides tools for Spring developers to quickly apply some of the common patterns found in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus).

- Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud, developers can quickly stand up services and applications that implement those patterns.

- The Spring Cloud Services suite adds several of the central coordination services found in Spring Cloud to the Pivotal Cloud Foundry Marketplace.

Features

- Config Server (based on Spring Cloud Config Server)
- Service Registry (based on Eureka via Spring Cloud Netflix)
- Circuit Breaker Dashboard (based on Hystrix and Turbine via Spring Cloud Netflix)

User-Provided Service Instances (CUPS)

User-provided service instances enable developers to use services that are not available in the marketplace with their applications running on Cloud Foundry.

Example:

- Create user provided service instance in PCF for the "oracle service" hosted in Amazon AWS Cloud (IaaS))

  ```
  cf cups oracle-aws -p
  '{"uri":"oracle://admin:xxx@myoracleinstance.c9anhdk2mqpc.us-
  west-1.rds.amazonaws.com:1521/ORCL"}'
  ```

- Bind the oracle-aws service with SimpleServiceApp either through manifest file or manually by using

  ```
  cf bind-service SimpleServiceApp oracle-aws.
  ```

Reference:

1. Cloud Foundry Documentation

2. Pivotal Cloud Platform Deep Dive - Roadshow slides

3. Cloud Foundry Summit

K.V. RAMANA RAO
JBOSSRAMANA@GMAIL.COM
9848834750