Cloud Foundry and OpenStack

Ramiro Salas, @ramirosalas
Advisory Architect | Pivotal

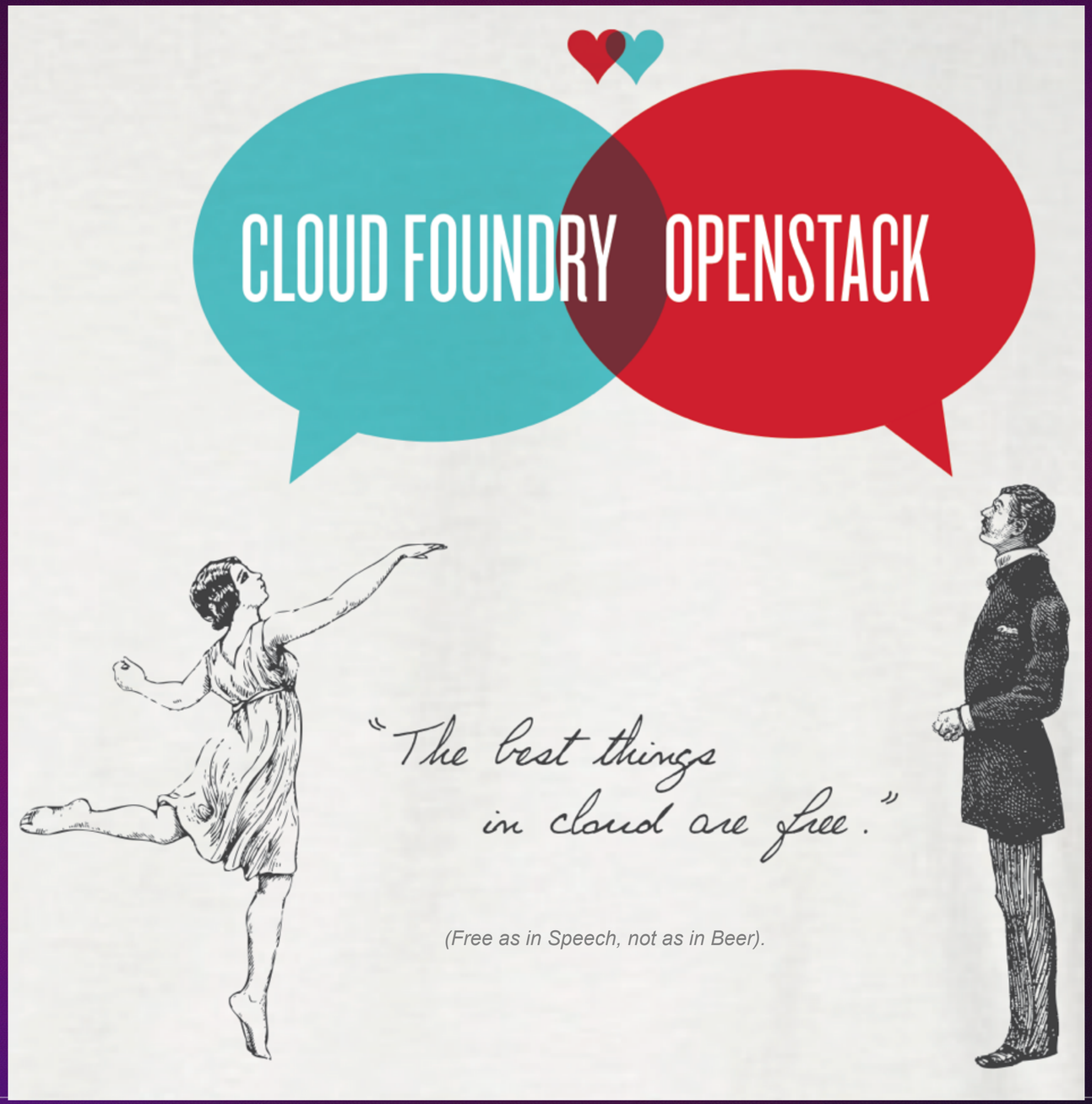
Open Source Communities

OpenStack - IaaS

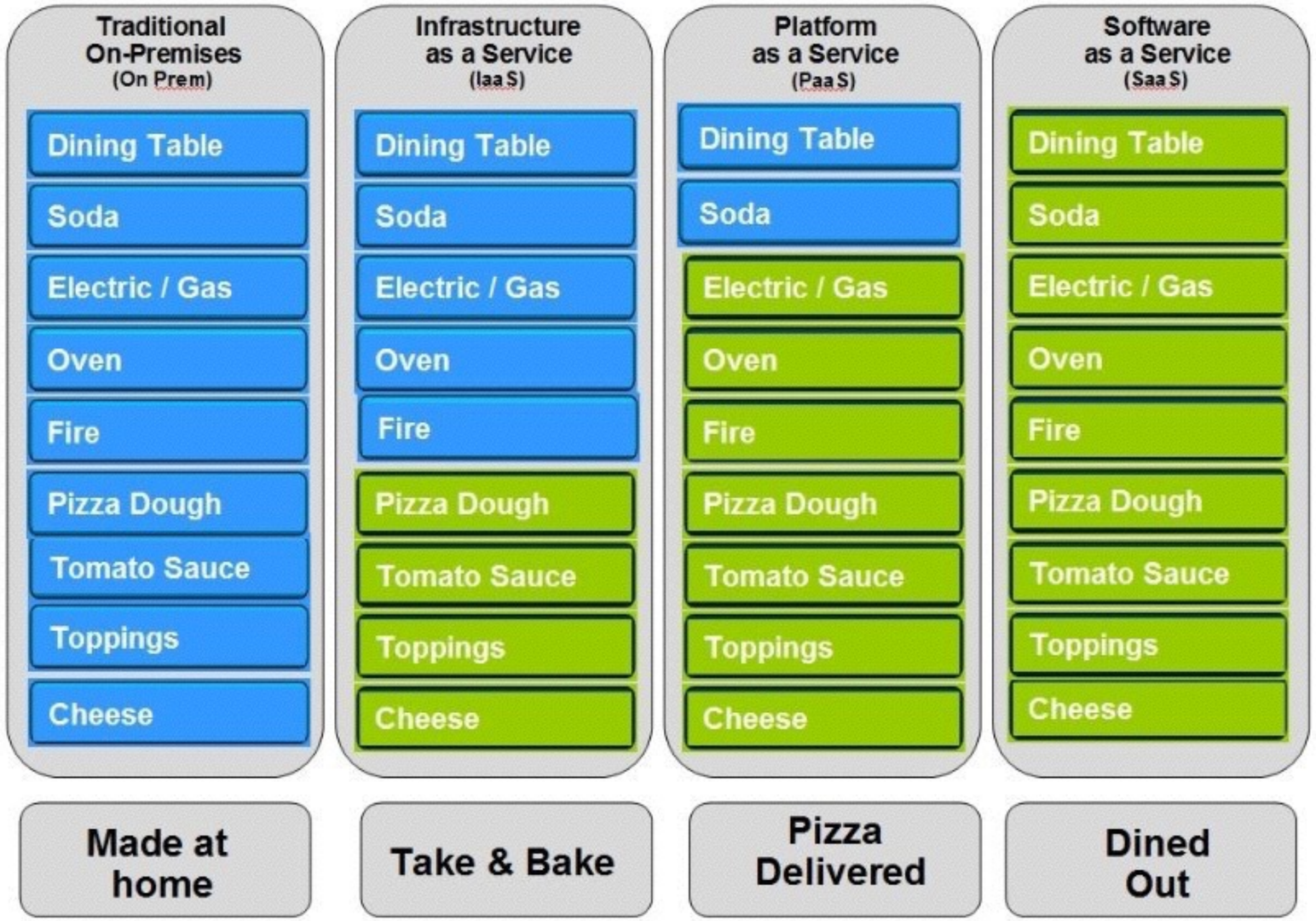
est. 2010

Cloud Foundry - PaaS

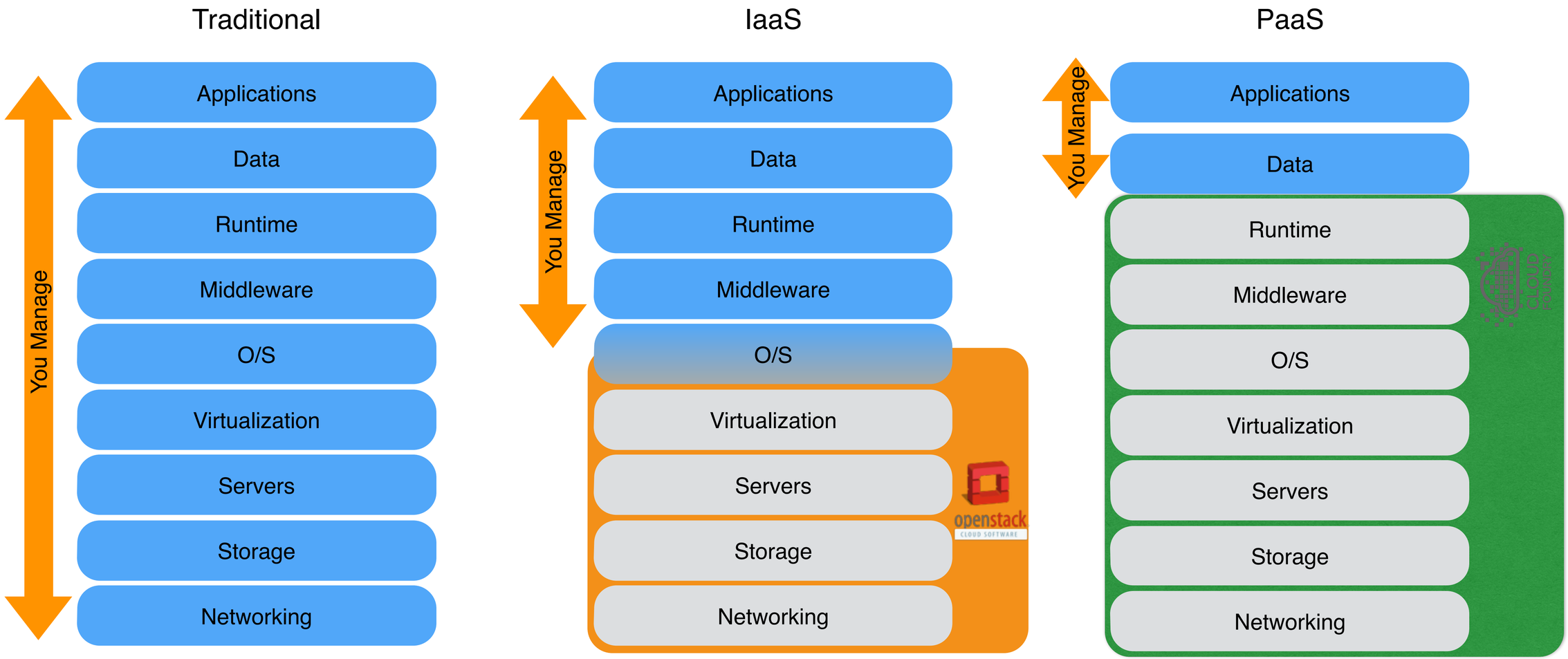
est. 2011



Pizza as a Service



What is a PaaS?



Launched NOVA – Apache-Licensed Cloud Computing, in Python

It's live, it's buggy, it's beta.
Check it out:

http://novacc.org

From the website:

> *Nova is a cloud computing fabric controller (the main part of an IaaS system) built to match the popular AWS EC2 and S3 APIs. It is written in Python, using the Tornadoand Twisted frameworks, and relies on the standard AMQP messaging protocol, and the Redis distributed KVS.*
>
> *Nova is intended to be easy to extend, and adapt.*

cloud computing

This entry was posted on 28May10, 12:40 am and is filed under entrepreneurs. You can follow any responses to this entry through RSS 2.0. You can leave a response, or trackback from your own site.

Joshua McKenty

Launched NOVA – Apache-Licensed Cloud Computing, in Python

It's live, it's buggy, it's beta.
Check it out:

http://novacc.org

From the website:

> *Nova is a cloud computing fabric controller (the main part of [illegible]) match the popular AWS EC2 and S3 APIs. It is written in Python, [illegible] the Tornadoand Twisted frameworks, and relies on the standard [illegible] protocol, and the Redis distributed KVS.*
>
> *Nova is intended to be easy to extend, and adapt.*

cloud computing

This entry was posted on 28May10, 12:40 am and is filed under entrepreneurs. You can follow any responses to this entry through RSS 2.0. You can leave a response, or trackback from your own site.

How are things the same?

	OpenStack & Cloud Foundry
License	***Apache v2.0***
Community	Vendors, Users, and Developers
Architecture	***API-based services and message-passing***

Cloud Foundry Foundation

Cloud Foundry Foundation & OpenStack Members

How are things different?

	OpenStack	Cloud Foundry
Language	***Python***	***Go and Ruby***
Release Cycle	6 months, integrated	2 weeks, parallel
Governance	***Dedicated Foundation***	***Linux Foundation project***
Installation tools	Various	BOSH
Communication Hub	***IRC (#openstack-dev)***	***Mailing lists***
Source code & review	Gerritt & Private Git	GitHub & pull requests
Adoption	***Mostly OSS trials and dev/test environments***	***Mostly commercial production deployments***

Units of Value

IaaS - OpenStack

- VMs
- Networks
- Volumes
- Images
- Security Groups, etc.

Users Don't Care About:

- Hypervisors
- Real Network Topologies
- How is the Storage Managed
- Where are the Images Stored
- What Hardware is Being Used

Units of Value

IaaS - OpenStack

- VMs
- Networks
- Volumes
- Images
- Security Groups, etc.

But they still have to care about:

- IP addresses
- Disk sizes
- VM orchestration
- OS Lifecycle
- HA/DR

OpenStack Constructs

Give me a VM

```
instance = nova.servers.create(name="test", image=image,
flavor=flavor, key_name="mykey")
```

Give me a Volume

```
volume = create(8192, snapshot_id=None, source_volid=None,
name=VolName, description="My Volume", volume_type=None,
user_id=None, project_id=None, availability_zone=az1,
metadata=None, imageRef=None)
```

Similar for Networks, Images, etc.

So a new layer is born: PaaS

- Focuses exclusively on applications
- Abstracts resources even further
  - No IPs - *Message queues instead*
  - No middleware configuration - *Buildpacks*
  - Scale automatically
  - All your logs in the same place
  - Designed for Cloud Native Apps

Units of Value

PaaS - Cloud Foundry

- Applications
- Services

Apps run on **Containers**
Services run on **VMs**

- Containers are transparent
- Lifecycle is fully managed
- System changes are declarative (manifest.yml)
- Front-ends, middleware, VMs, etc. all abstracted

Structured vs. Unstructured PaaS

Unstructured

- DevOps controls every aspect of the deliverable app
  - Filesystem
  - Ports exposed
  - Layers
  - Repositories
  - Orchestration
  - Dependencies…

Example:

Custom-built systems with different pieces like:

- Docker
- Kubernetes
- Mesos…

But Often, Containers Alone Aren’t Enough…

Structured vs. Unstructured PaaS

Structured

- Developers only specifies app instances, services to bind, and memory.
- PaaS takes care of:
  - Routing
  - Security
  - Filesystem
  - Ports
  - Scheduling
  - High Availability, etc…

They Don't have to Care about the HOW

CLOUD FOUNDRY™

Cloud Foundry is…

The world's leading open source platform-as-a-service.

- Supported by dozens of major organizations
- Language and framework agnostic
- Manages both VMs and containers
- Orchestrates both applications and data services

Founded and commercialized by Pivotal Software, Inc.

Code donated to Cloud Foundry Foundation in 2015

An (Overly) Simple View of the World

Applications

- Stateless
- Run in Containers
- Horizontally Scalable
- Disposable
- No permanent storage

Data Services

- Stateful
- Run in Virtual Machines
- Multi-tenant
- Diagonally scalable
- Durable storage

12Factor.net

Methodology for building software that:

- **Use declarative formats for setup automation, to minimize time and cost for new developers joining the project;**
- **Have a clean contract with the underlying operating system, offering maximum portability between execution environments;**
- **Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;**
- **Minimize divergence between development and production, enabling continuous deployment for maximum agility;**
- **And can scale up without significant changes to tooling, architecture, or development practices.**

The Twelve Factors

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

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

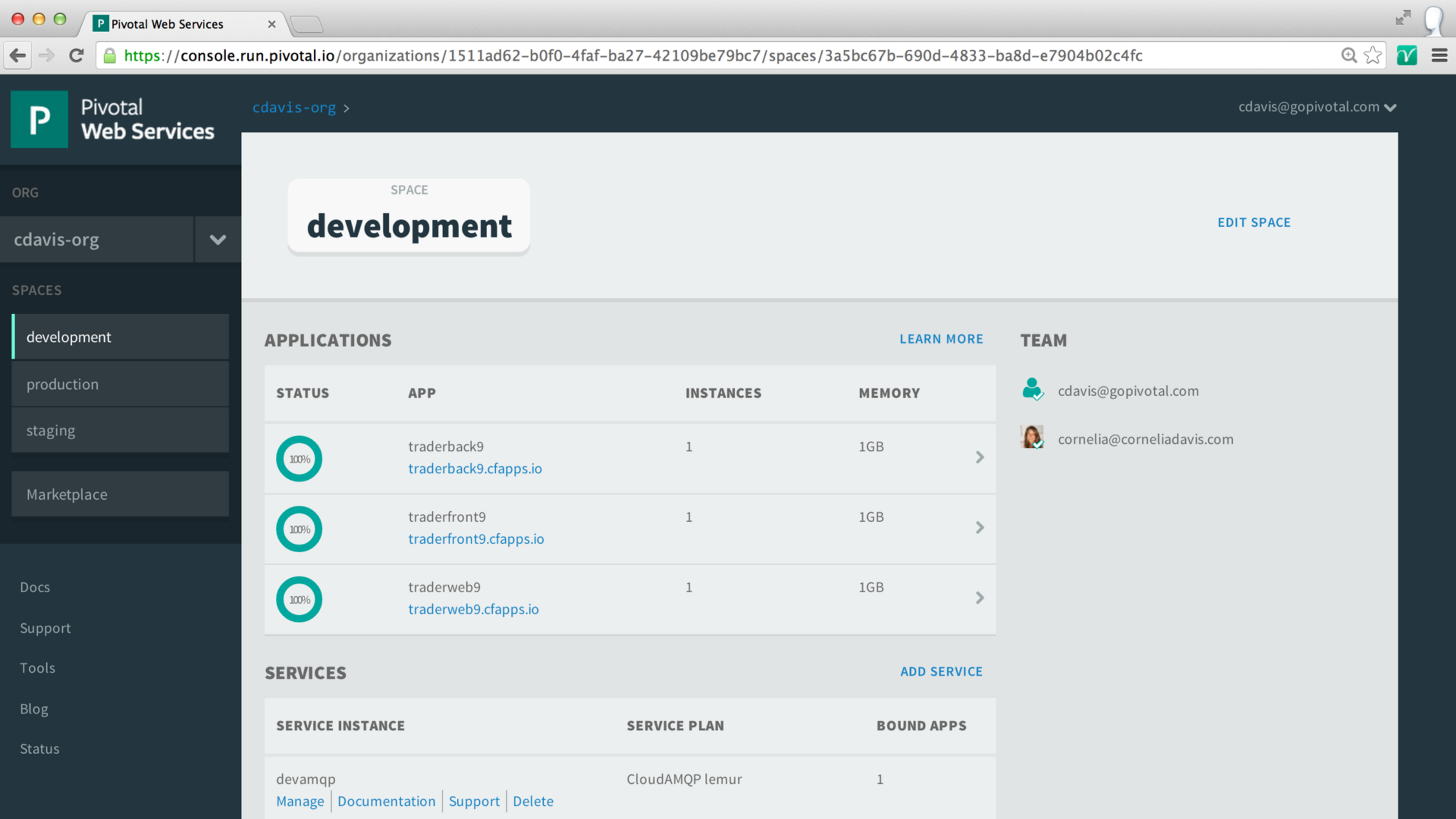
VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

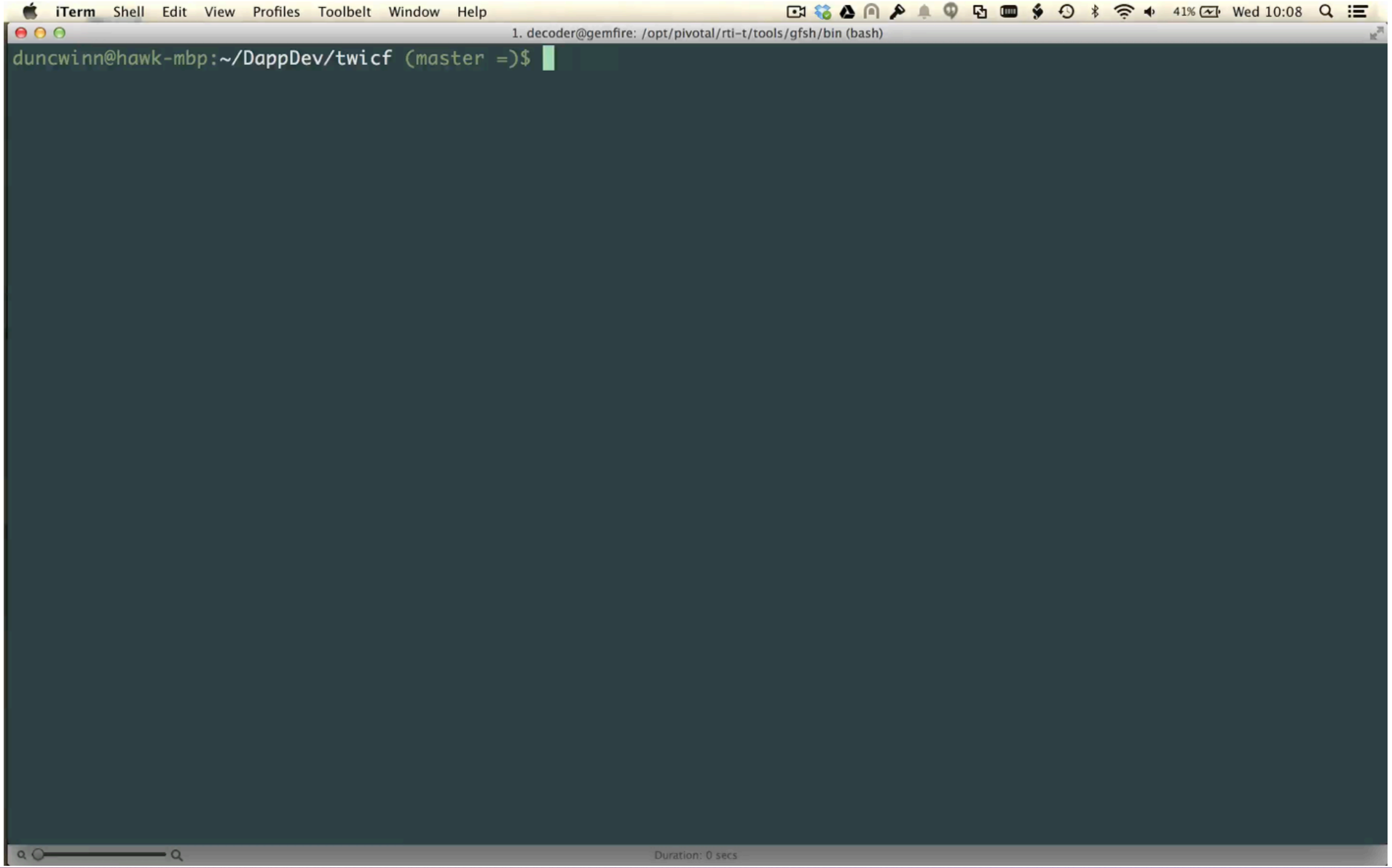
X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

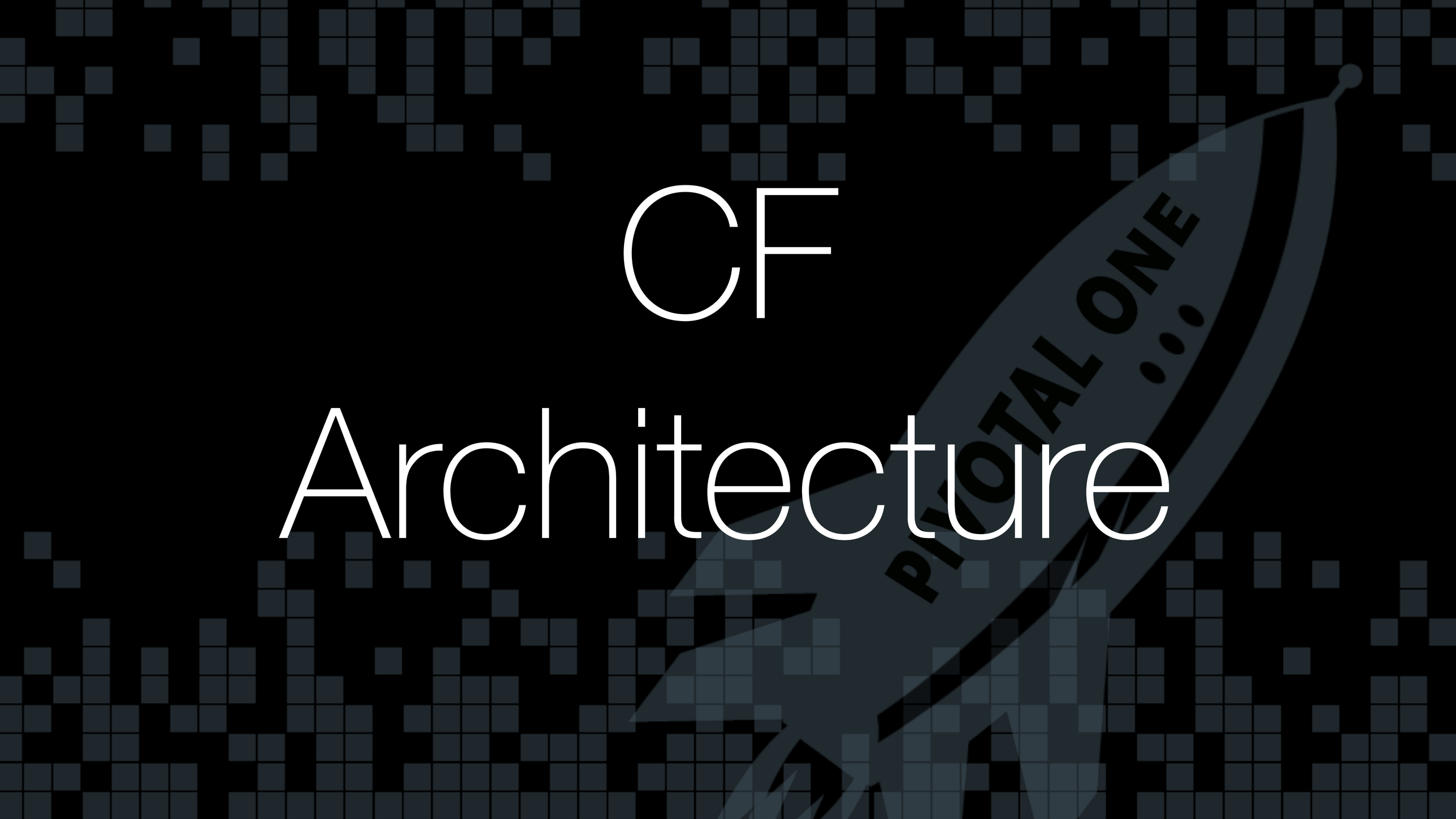
XII. Admin processes
Run admin/management tasks as one-off processes


Pivotal Web Services
https://console.run.pivotal.io/organizations/1511ad62-b0f0-4faf-ba27-42109be79bc7/spaces/3a5bc67b-690d-4833-ba8d-e7904b02c4fc
Pivotal Web Services
cdavis-org >
cdavis@gopivotal.com
ORG
cdavis-org
SPACES
development
production
staging
Marketplace
Docs
Support
Tools
Blog
Status
SPACE
development
EDIT SPACE
APPLICATIONS
LEARN MORE
STATUS
APP
INSTANCES
MEMORY
100%
traderback9
traderback9.cfapps.io
1
1GB
100%
traderfront9
traderfront9.cfapps.io
1
1GB
100%
traderweb9
traderweb9.cfapps.io
1
1GB
TEAM
cdavis@gopivotal.com
cornelia@corneliadavis.com
SERVICES
ADD SERVICE
SERVICE INSTANCE
SERVICE PLAN
BOUND APPS
devamqp
Manage | Documentation | Support | Delete
CloudAMQP lemur
1

PCF Demo: cf push



CF Architecture



Cloud Foundry: Applications and Services

Services (virtual machines): managed by "***BOSH***"

Applications (containers): managed by "***Runtime***"

Why BOSH

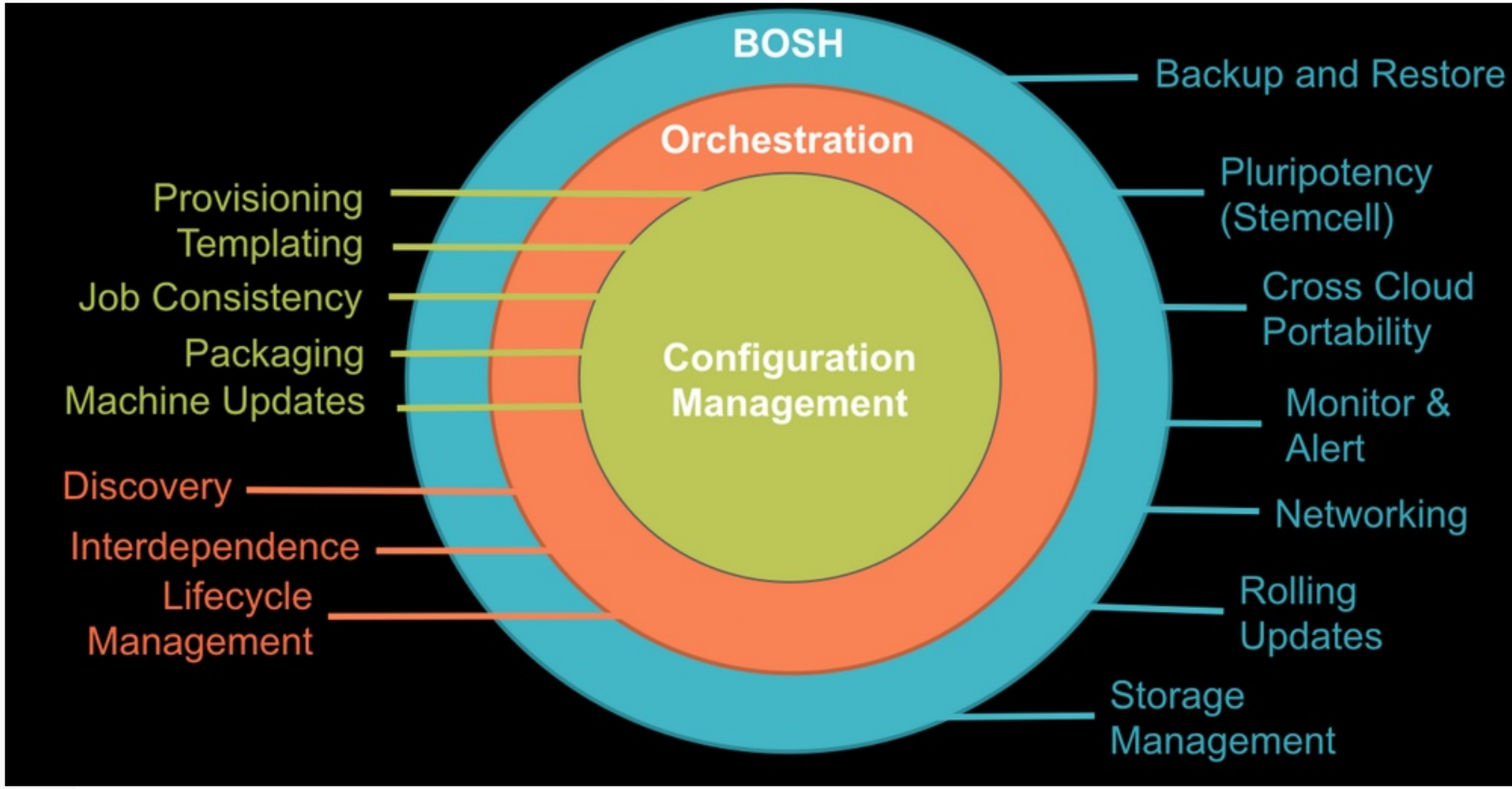
Provision services, not machines

Enables continuous delivery

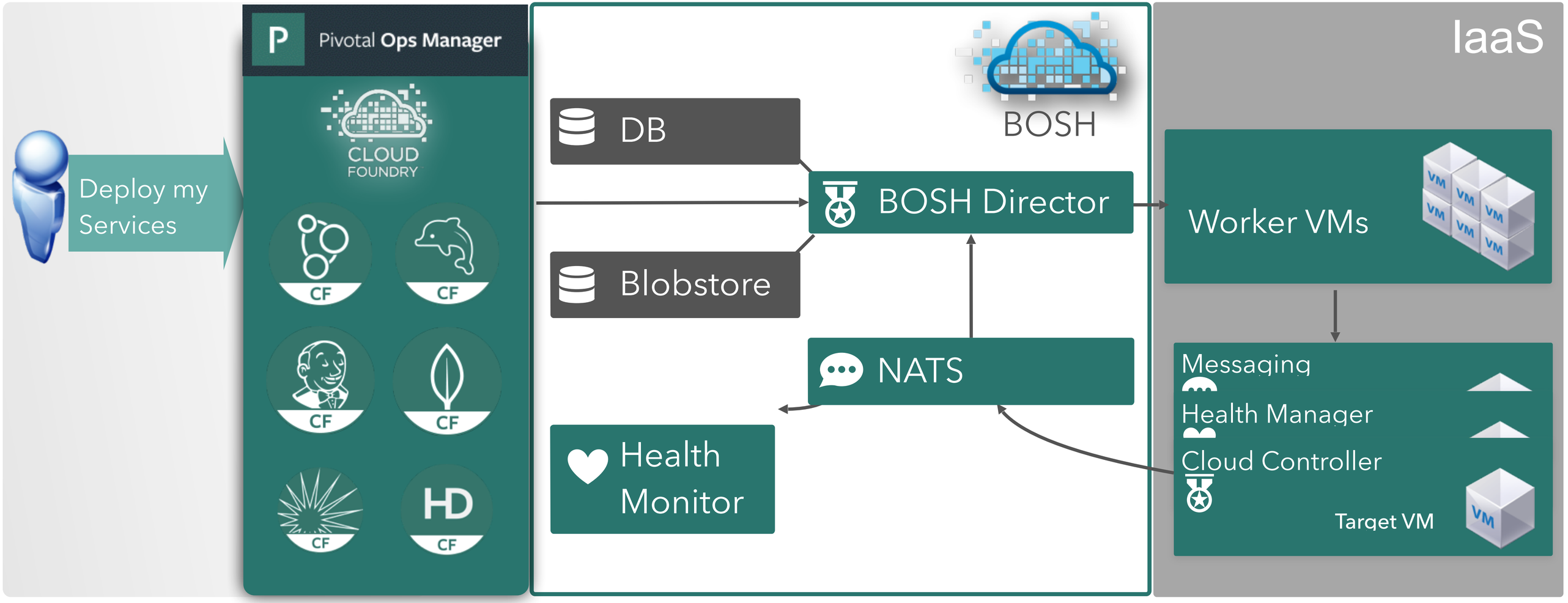
Cloud-agnostic view of Platform Ops

Holistic Toolchain for “rule them all"

Eliminate bespoke automation on top of config management

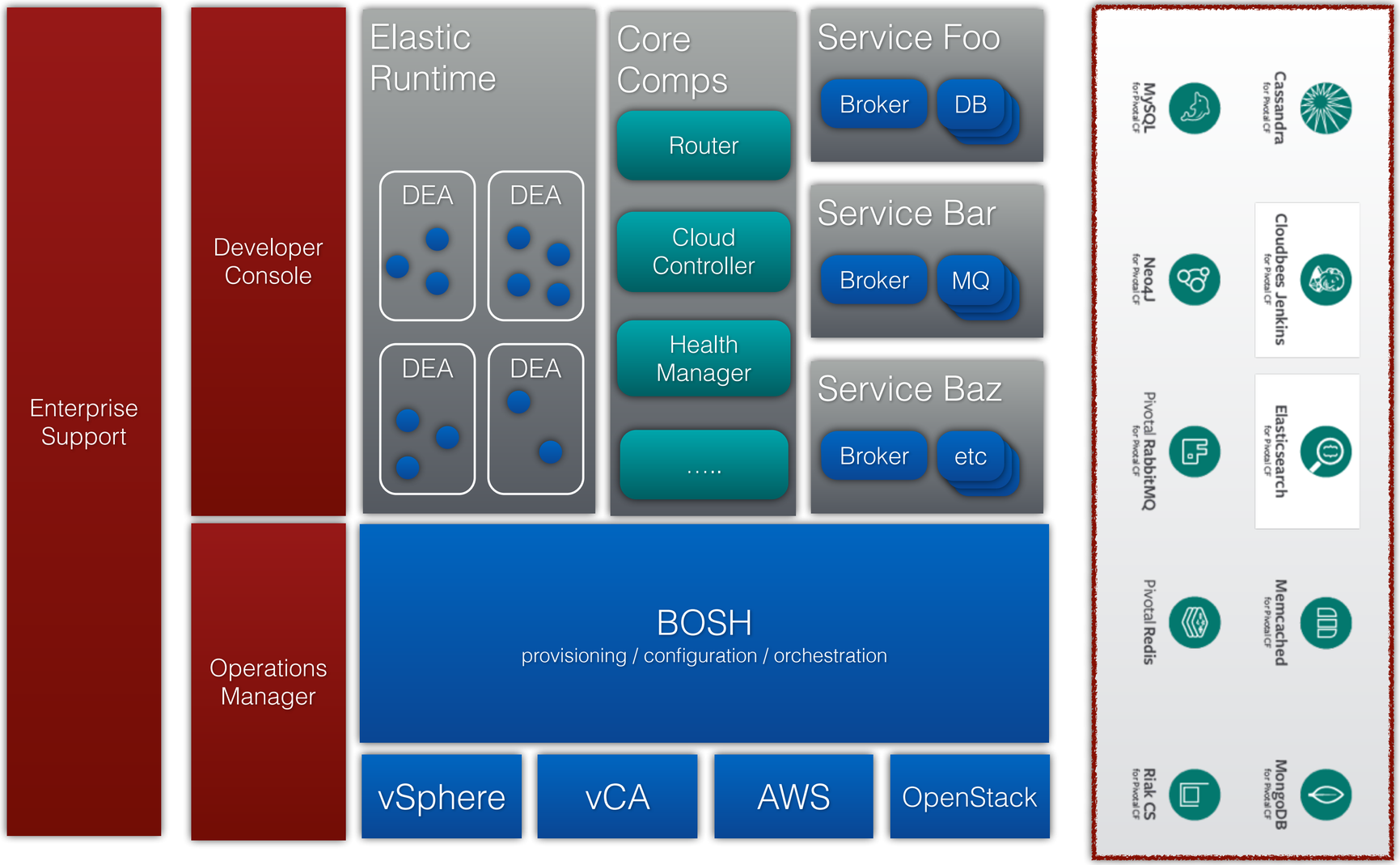


Ops Manager + BOSH



Pivotal Cloud Foundry Architecture

Enterprise Cloud Foundry

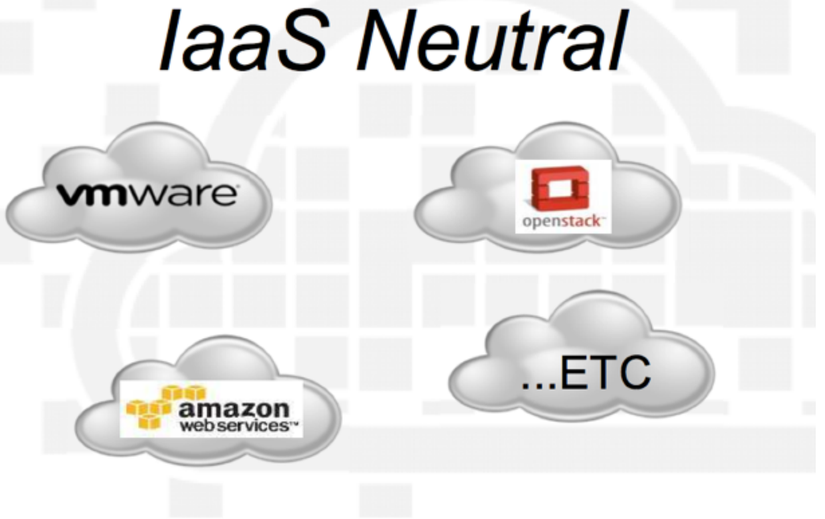


BOSH: Cloud Provider Interface

Stemcell
  create_stemcell(image, cloud_properties)
  delete_stemcell(stemcell_id)

VM
  create_vm(agent_id, stemcell_id, resource_pool,
            networks, disk_locality, env)
  delete_vm(vm_id)
  reboot_vm(vm_id)
  configure_networks(vm_id, networks)

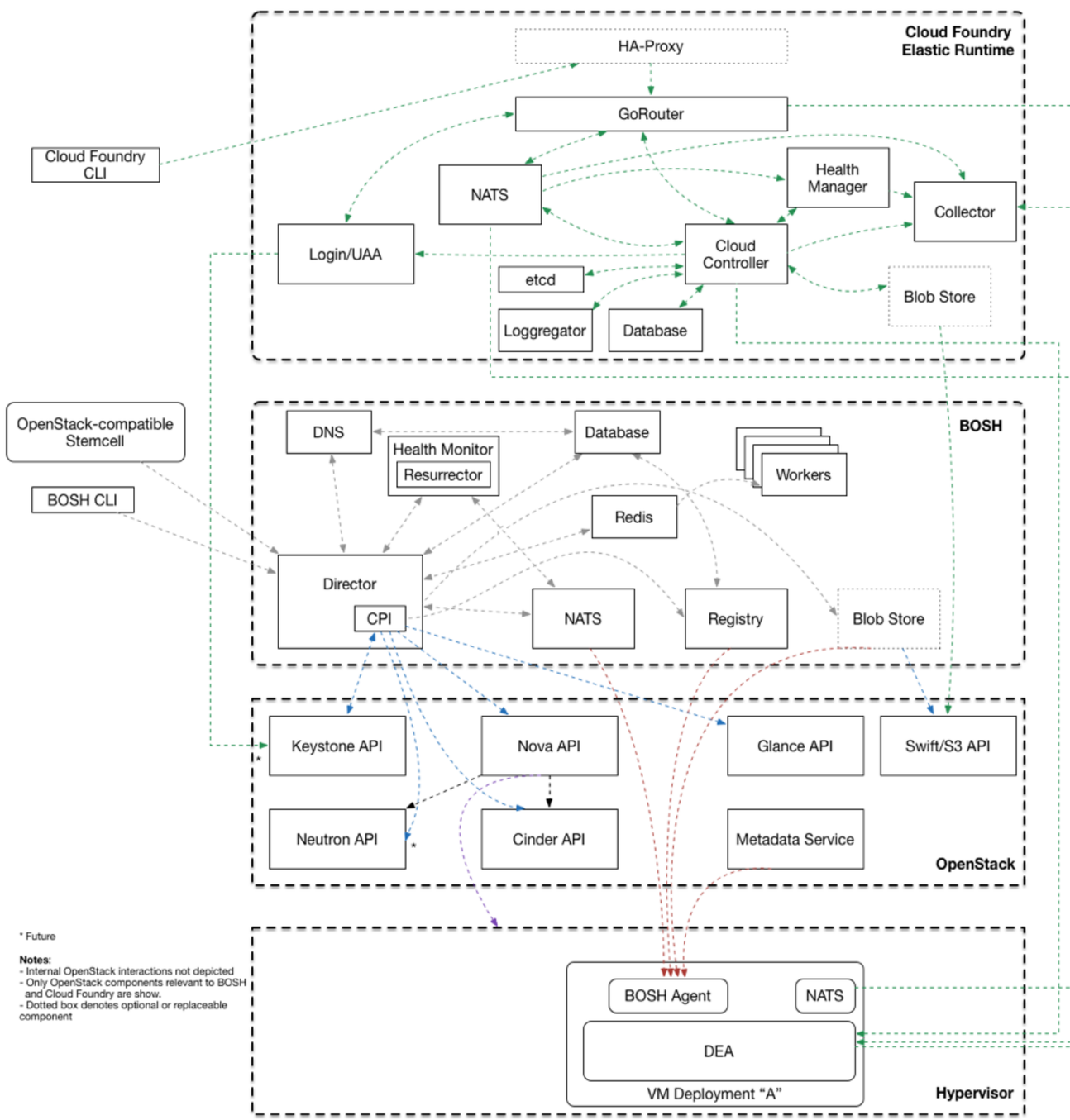
Disk
  create_disk(size, vm_locality)
  delete_disk(disk_id)
  attach_disk(vm_id, disk_id)
  detach_disk(vm_id, disk_id)



OpenStack Integration

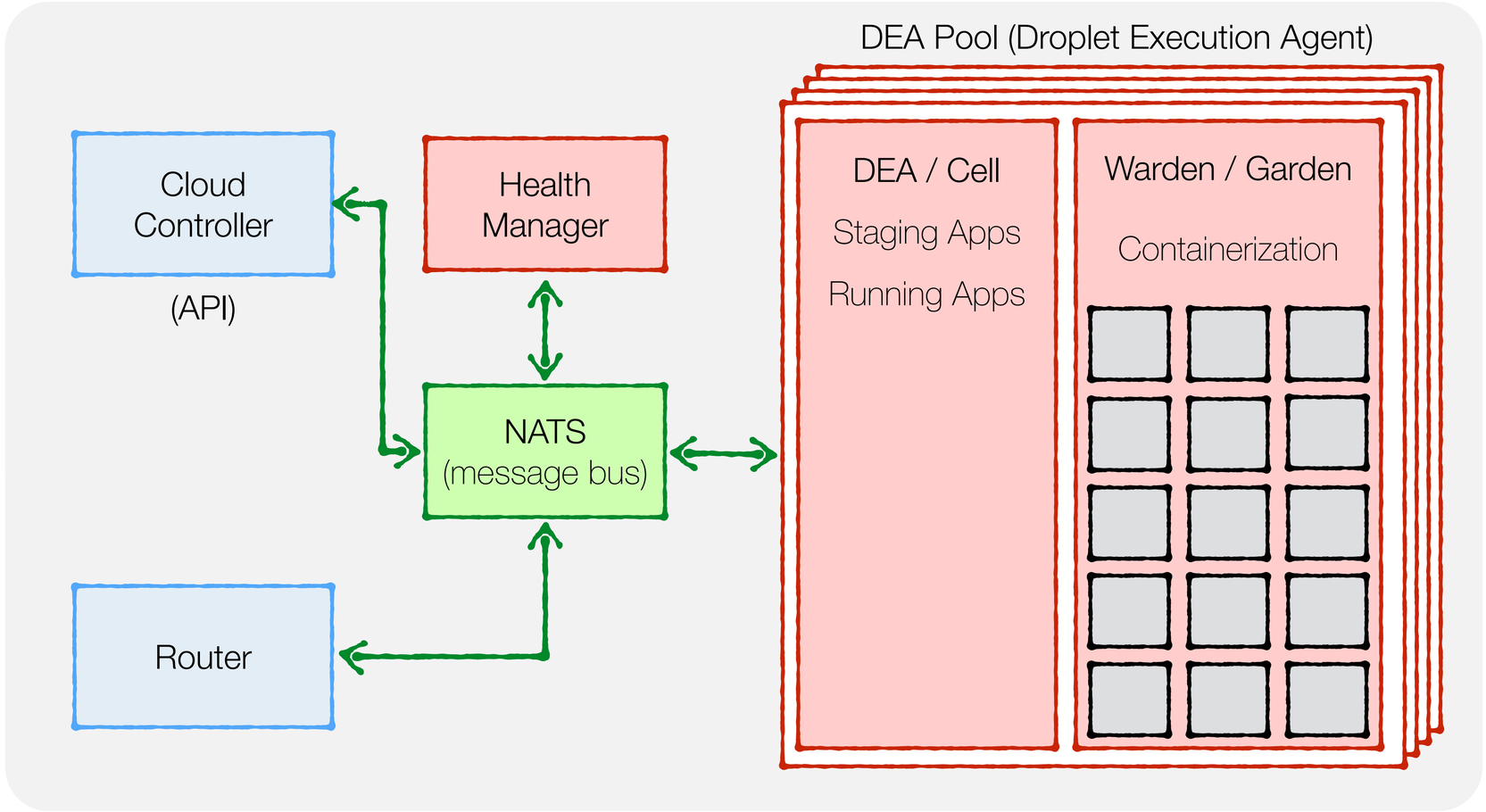
BOSH CPI

- Can use S3 interfaces for blobstore (Swift/Ceph)
- Uses Glance API to upload stemcells
- Interfaces directly with Nova (Cinder and Neutron are called via Nova)
- Credentials obtained via Keystone

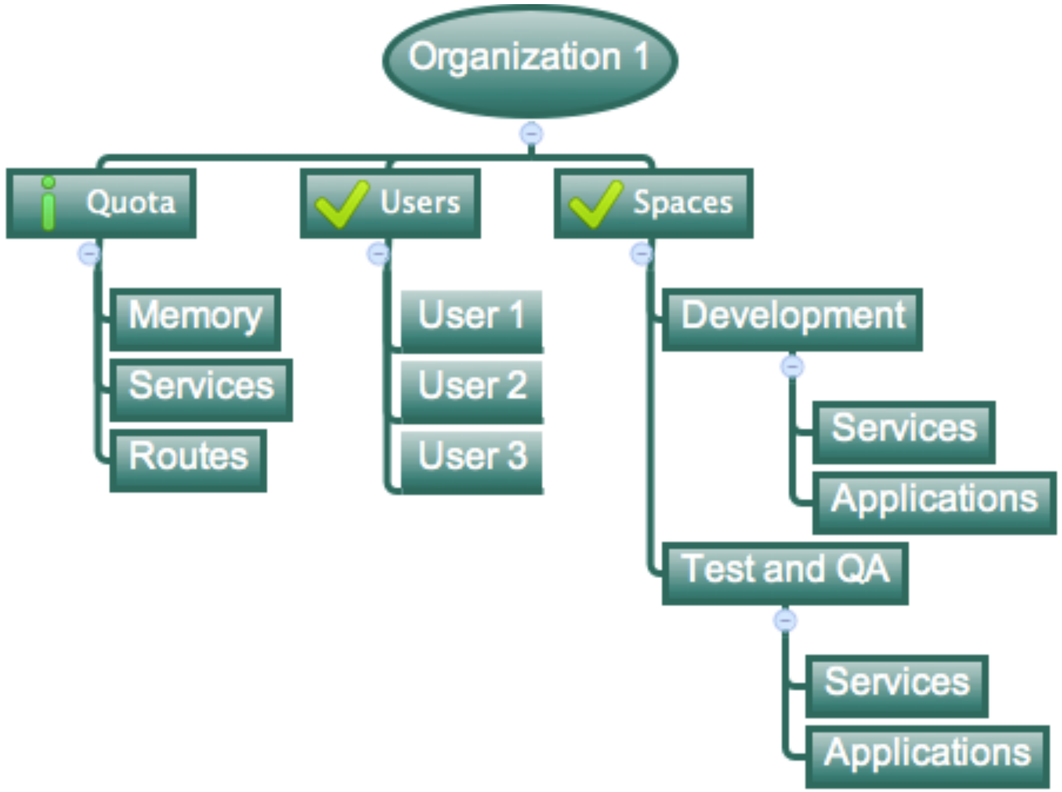


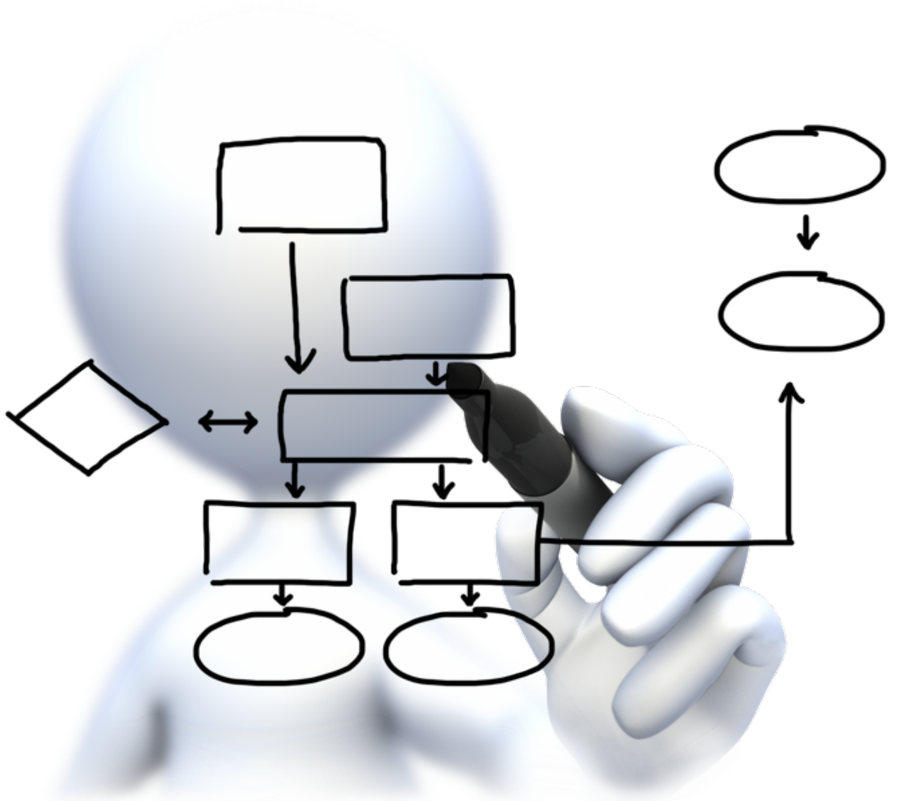
Process flow

> cf Push App



Orgs, Spaces, Users and Quotas





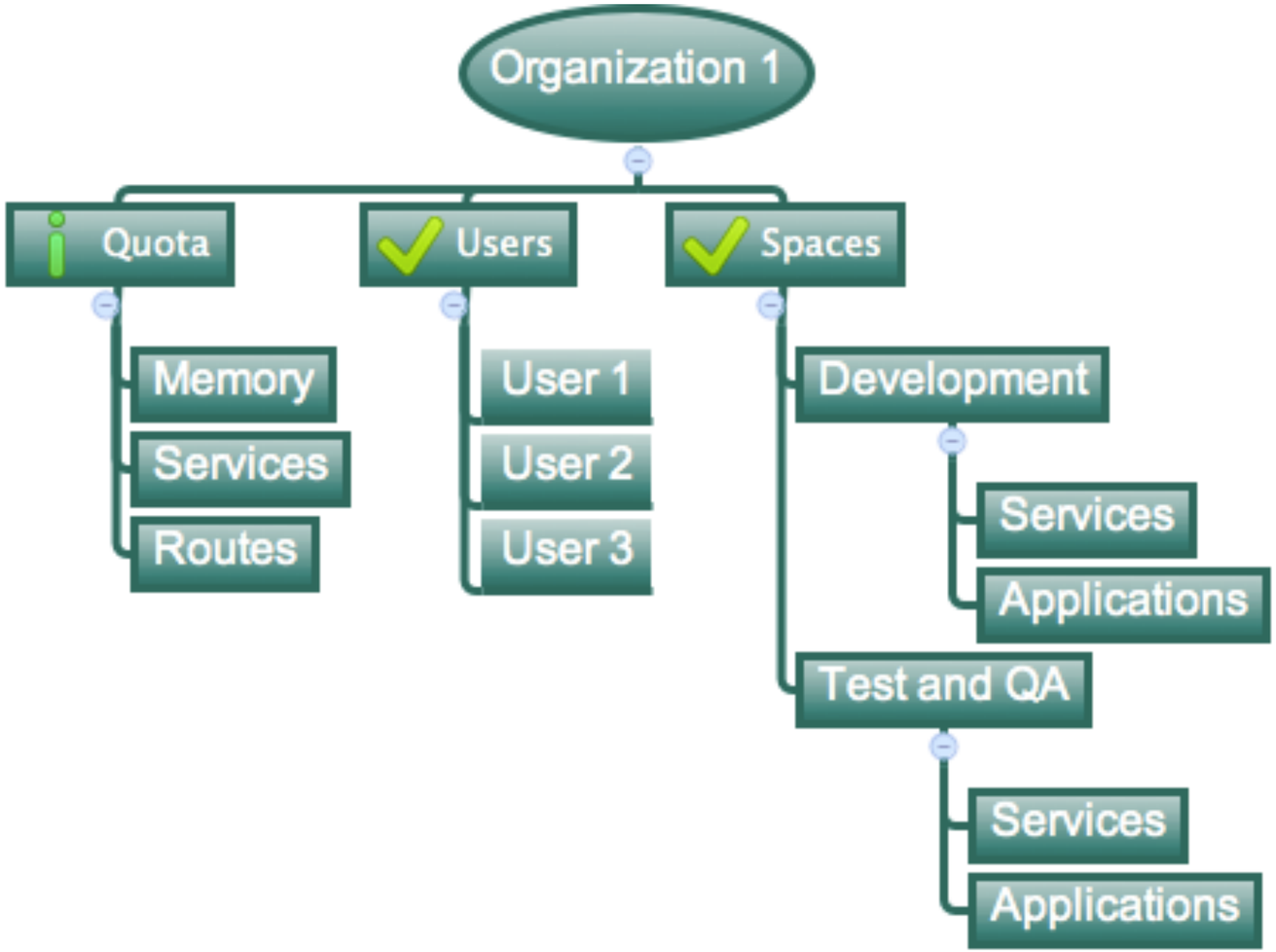
Organizations

Logical division within a Pivotal CF install / Foundation.

Each organization has its own users and assigned quota

User permissions / roles are specified per space within an organization

Sub-divided into Spaces



Quotas and Plans

Different quota limits (e.g. “small”, “enterprise”, “default”, “runaway”) can be assigned per Organization

Quota defines

- Total Memory
- Total # of Services
- Total # of Routes

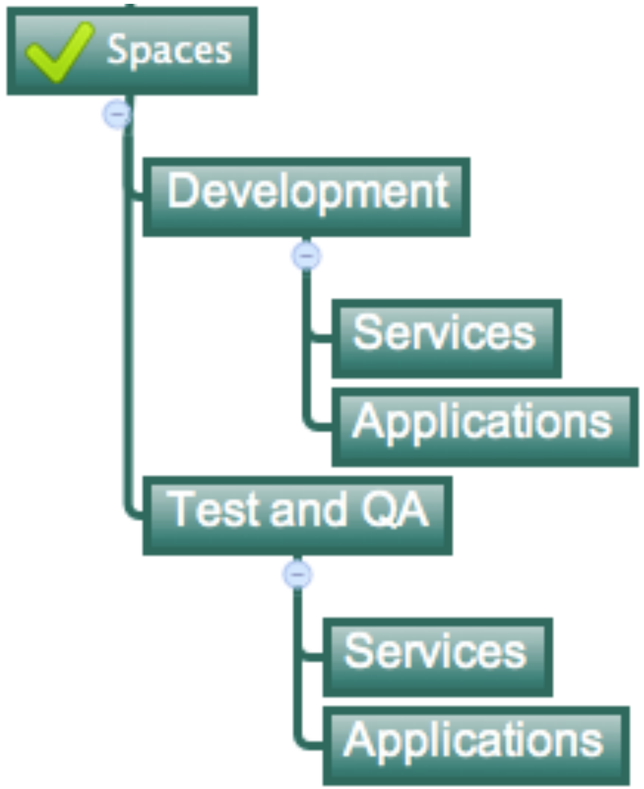
Spaces

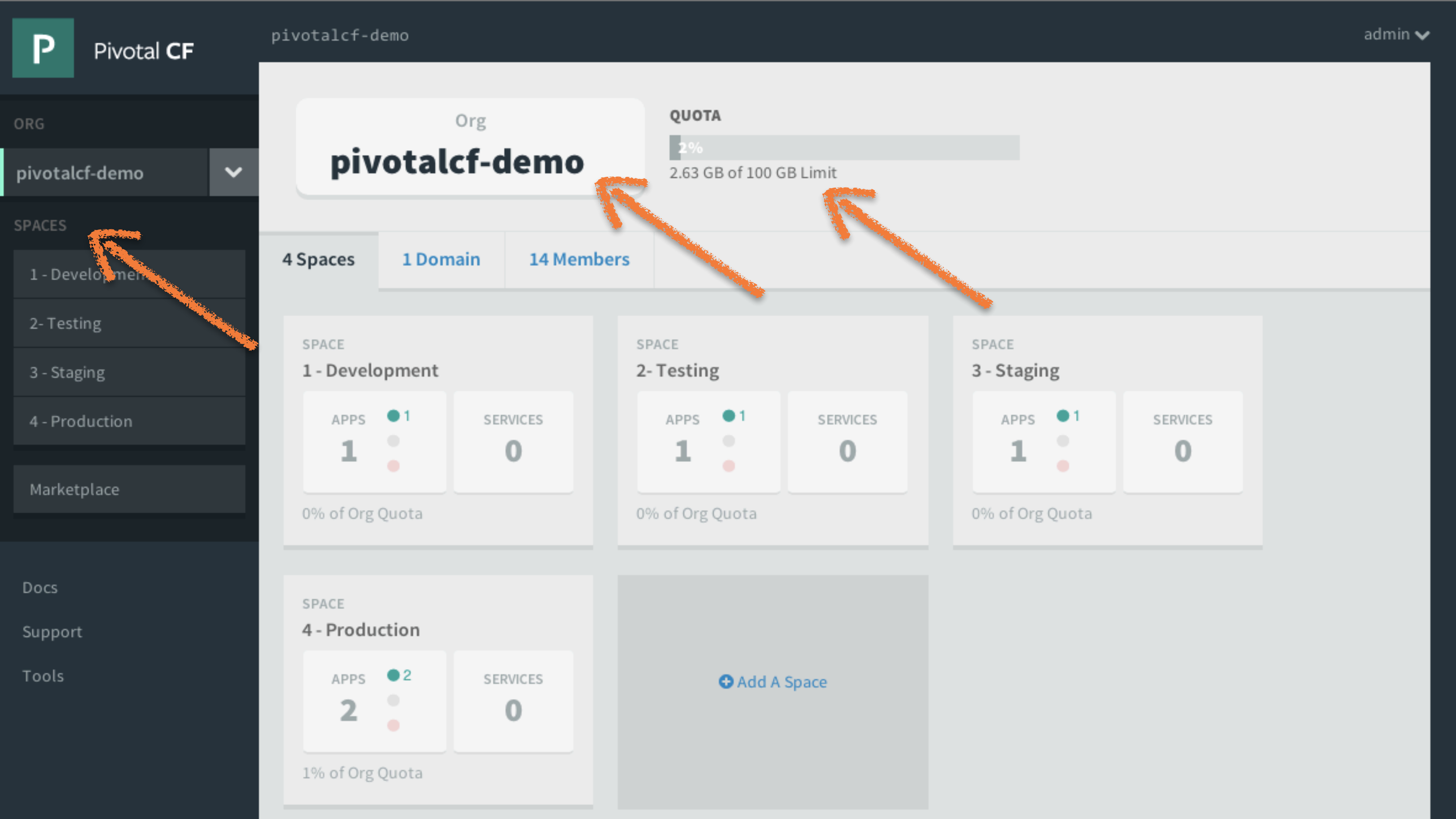
Logical sub-division within an organization

Users authorized at an organization level can have different roles per space

Services and Applications are created / specified per Space

Same Service can have different meanings per space




Pivotal CF
pivotalcf-demo
admin
ORG
pivotalcf-demo
SPACES
1 - Development
2- Testing
3 - Staging
4 - Production
Marketplace
Docs
Support
Tools
Org
pivotalcf-demo
QUOTA
2%
2.63 GB of 100 GB Limit
4 Spaces
1 Domain
14 Members
SPACE
1 - Development
APPS
1
1
SERVICES
0
0% of Org Quota
SPACE
2- Testing
APPS
1
1
SERVICES
0
0% of Org Quota
SPACE
3 - Staging
APPS
1
1
SERVICES
0
0% of Org Quota
SPACE
4 - Production
APPS
2
2
SERVICES
0
1% of Org Quota
Add A Space

Overview: Deploying *App* to Cloud Foundry *Runtime*

① Upload app bits and metadata

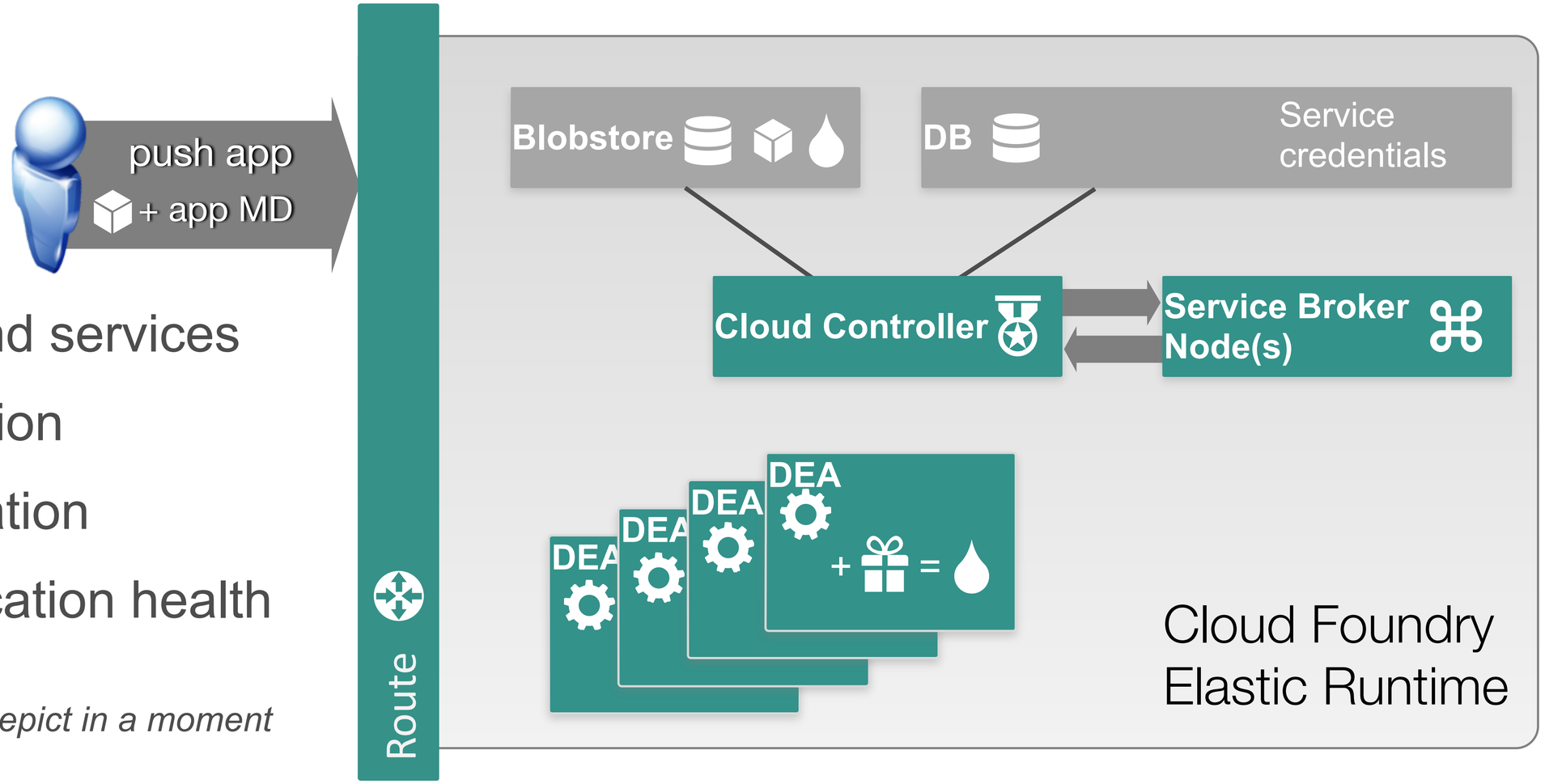
② Create and bind services

③ Stage application

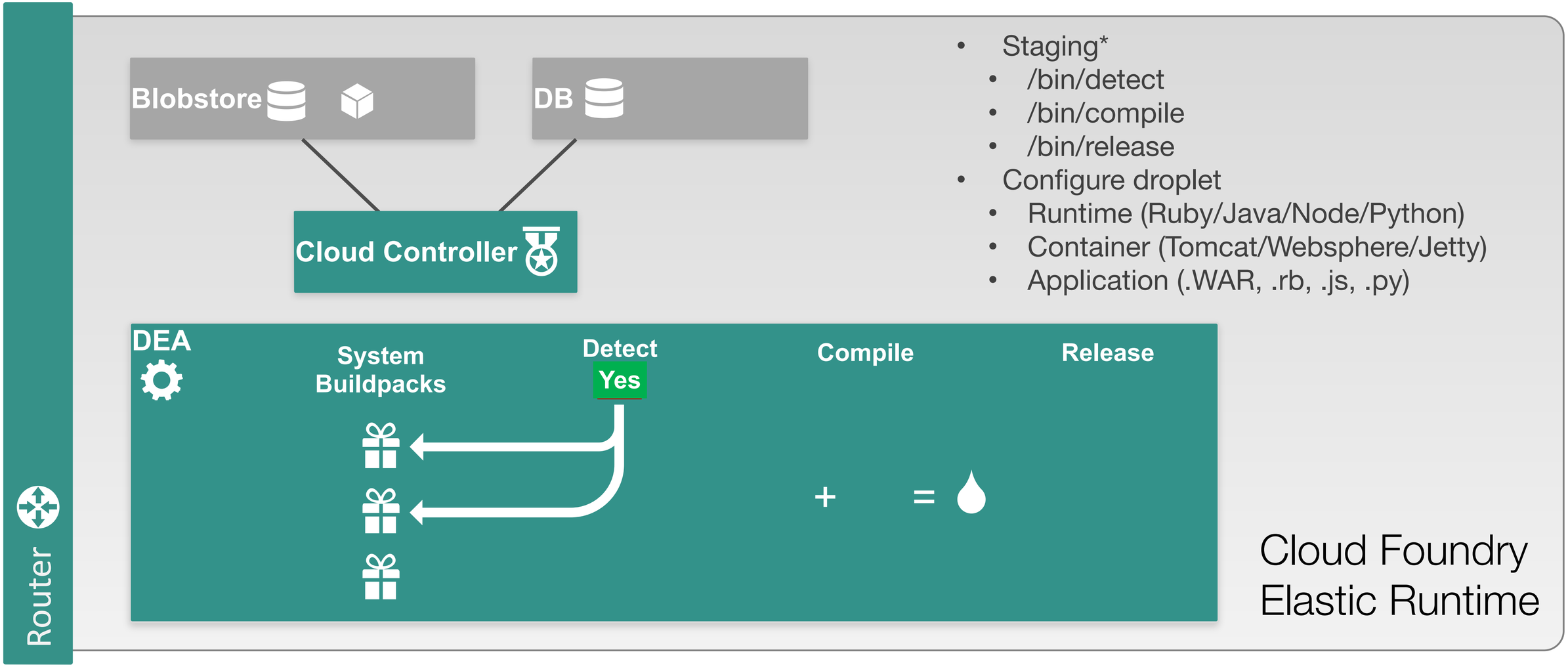
④ Deploy application

⑤ Manage application health

…which we will depict in a moment

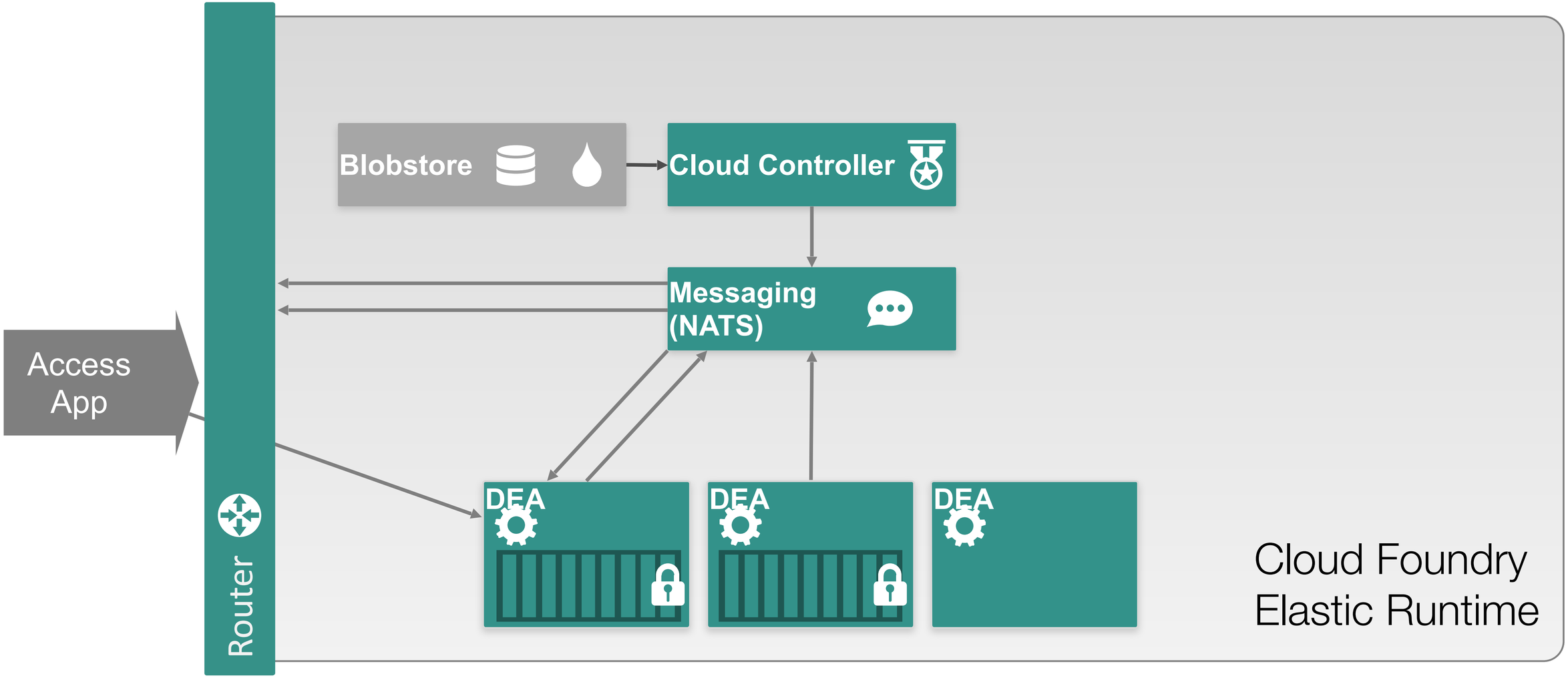


Stage an *Application*



- Staging*
  - /bin/detect
  - /bin/compile
  - /bin/release
- Configure droplet
  - Runtime (Ruby/Java/Node/Python)
  - Container (Tomcat/Websphere/Jetty)
  - Application (.WAR, .rb, .js, .py)

Deploying an *Application*



Under the Hood

Buildpacks

Containers

Droplet Execution Agents

Buildpacks

Defines the rules to create a fully-contained execution environment

App + Buildpack = Droplet

A Droplet is a fully self-sufficient, referentially correct package that can be executed in an isolated environment

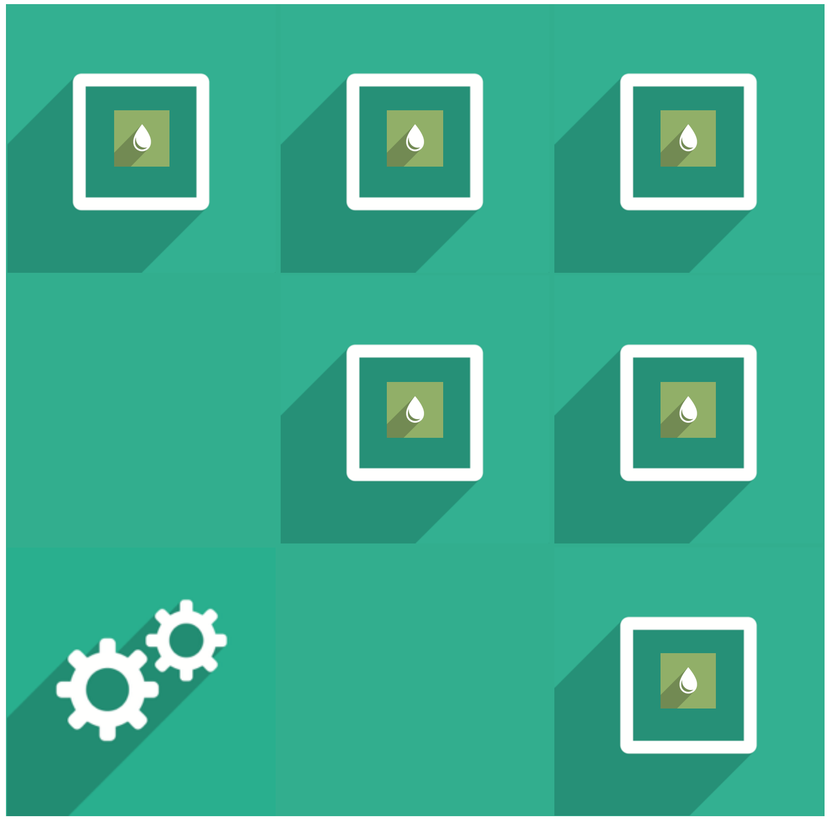
Containers

Isolated environments within an OS VM that run Droplets according to defined rules

There can be many Containers per OS VM thus increasing VM utilization and density

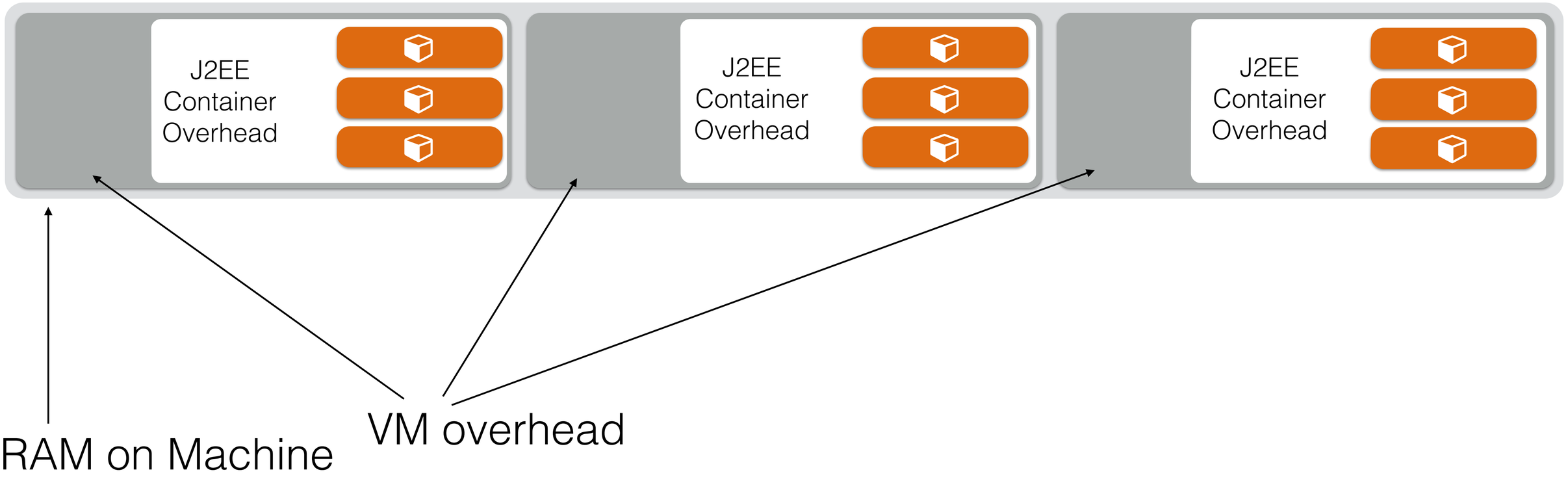
Droplet Execution Agents

VMs that host Containers and can create/destroy them as needed or ordered



Why Containers?

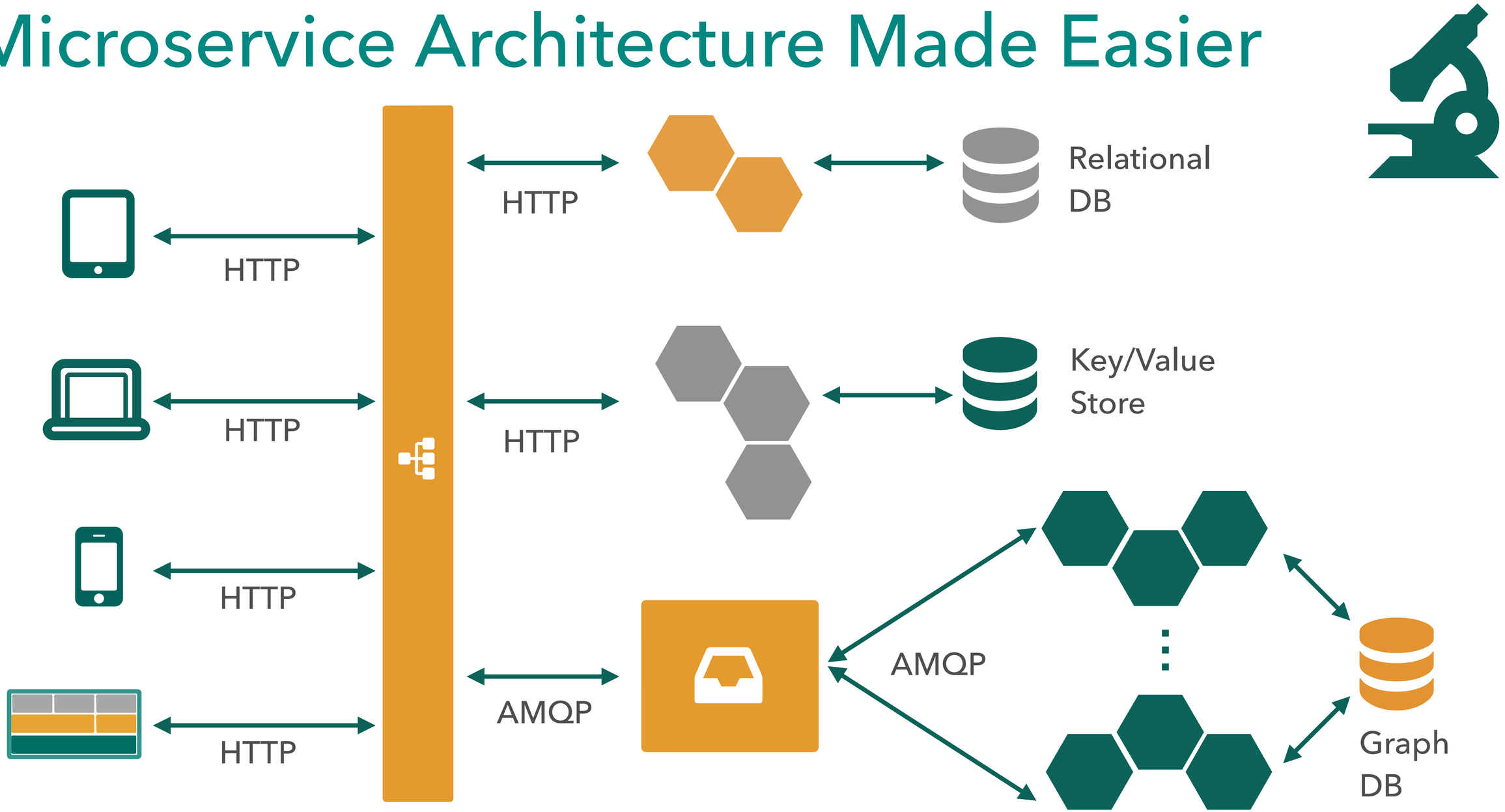
VMs are an inefficient level of isolation



Why Containers?

Containers + microservices allow denser packing and looser coupling of components

Microservice Architecture Made Easier



Resources

Documentation: http://docs.cloudfoundry.org

Meetups: http://cloudfoundry.meetup.com/