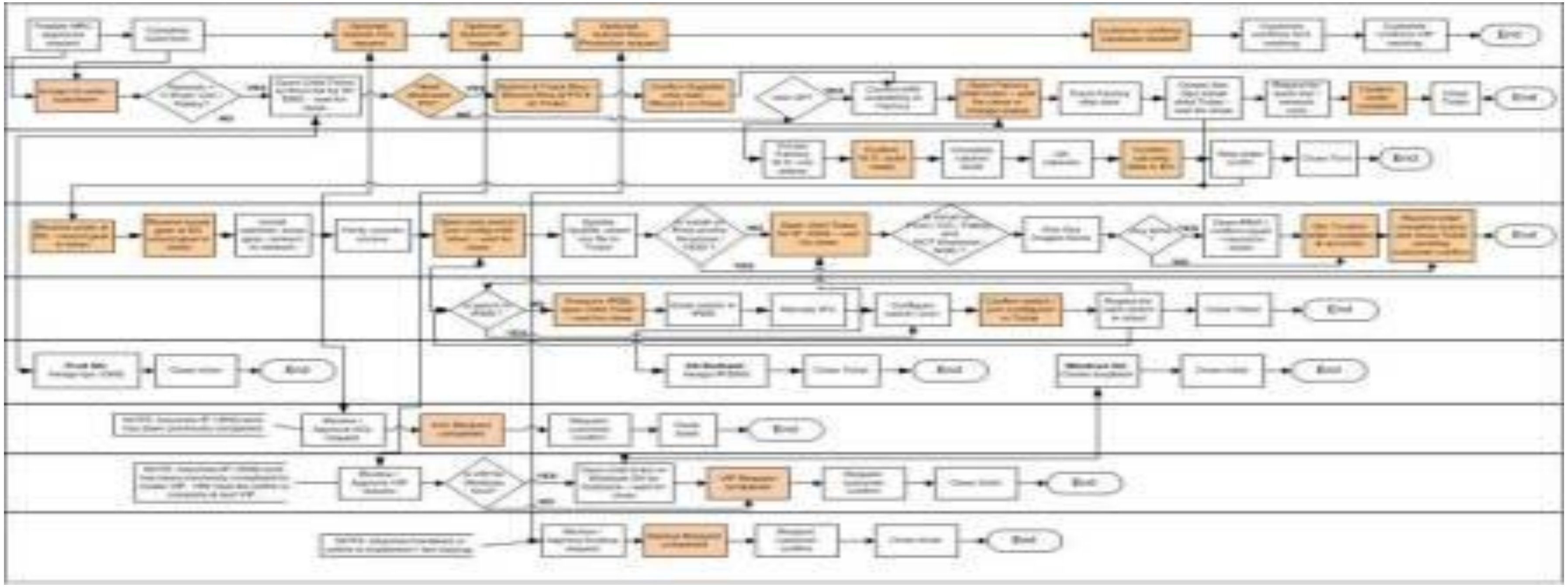
CLOUD FOUNDRY™

Cloud Foundry Technical Overview

By Pivotal

Simplify Application Deployment, from: this…



* An actual application provisioning/update flow in a large enterprise. Image is blurred for privacy reasons

To: Pushing apps to the cloud with a few easy verbs

Operator

```
cf-iaas.yml
provision <my cloud>
add_capacity <my cloud>
```

Cloud Deployment: 2-4 hours

Developer

```
target <my cloud>
push <my app>
create <my services>
bind <my services>
scale <my app> +100
```

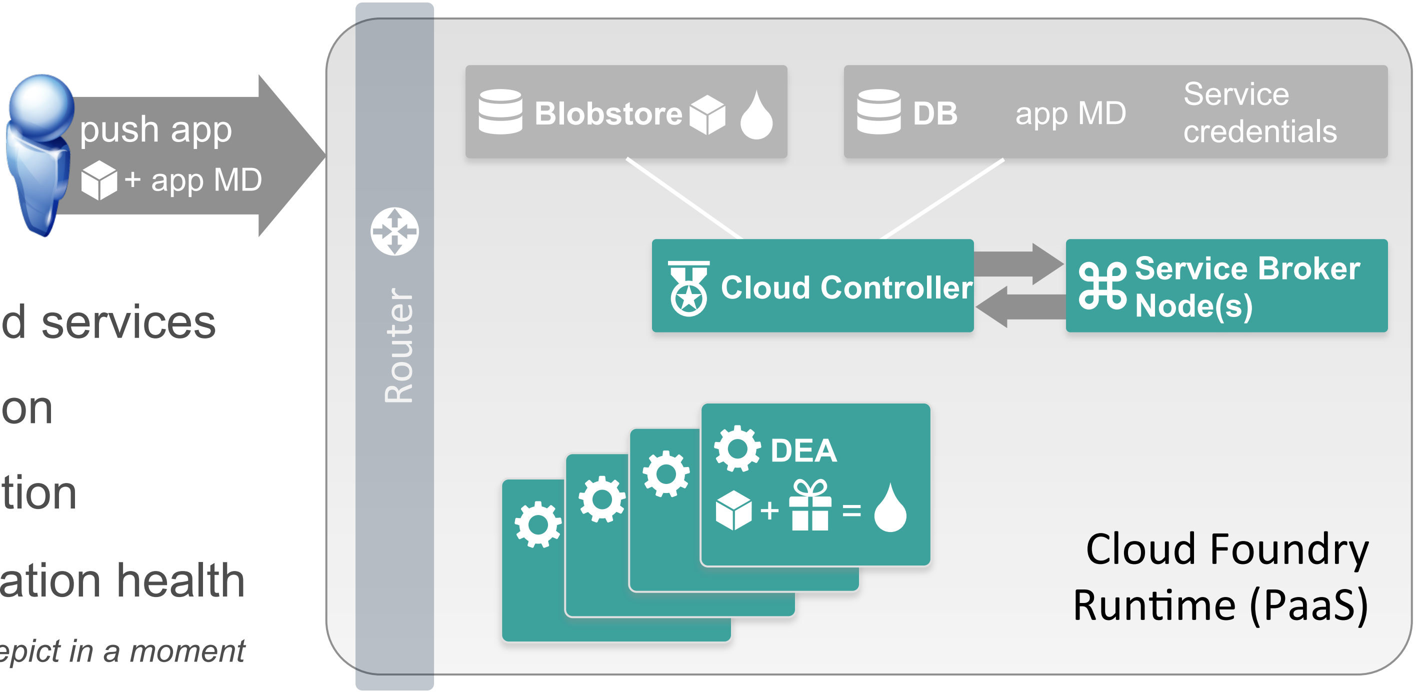
App Deployment: 30-90 seconds

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

Developer

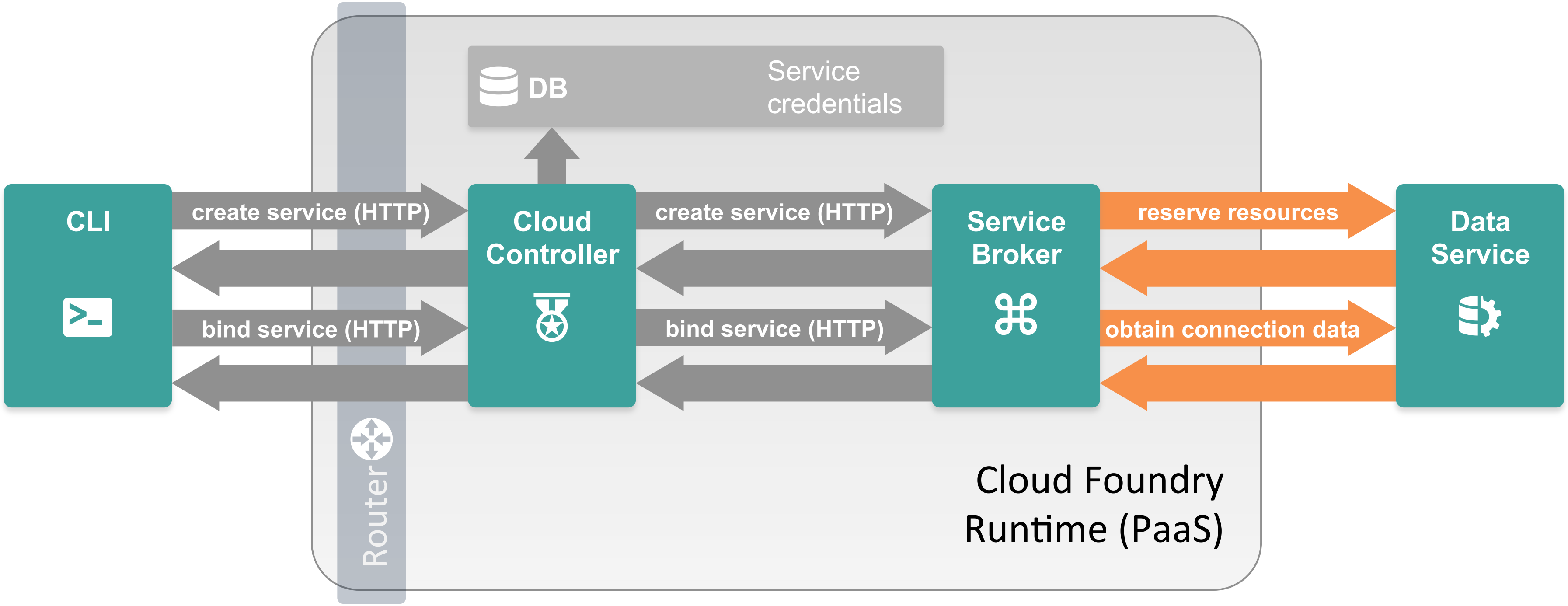
1. ① Upload app bits and metadata
2. ② Create and bind services
3. ③ Stage application
4. ④ Deploy application
5. ⑤ Manage application health

…which we will depict in a moment



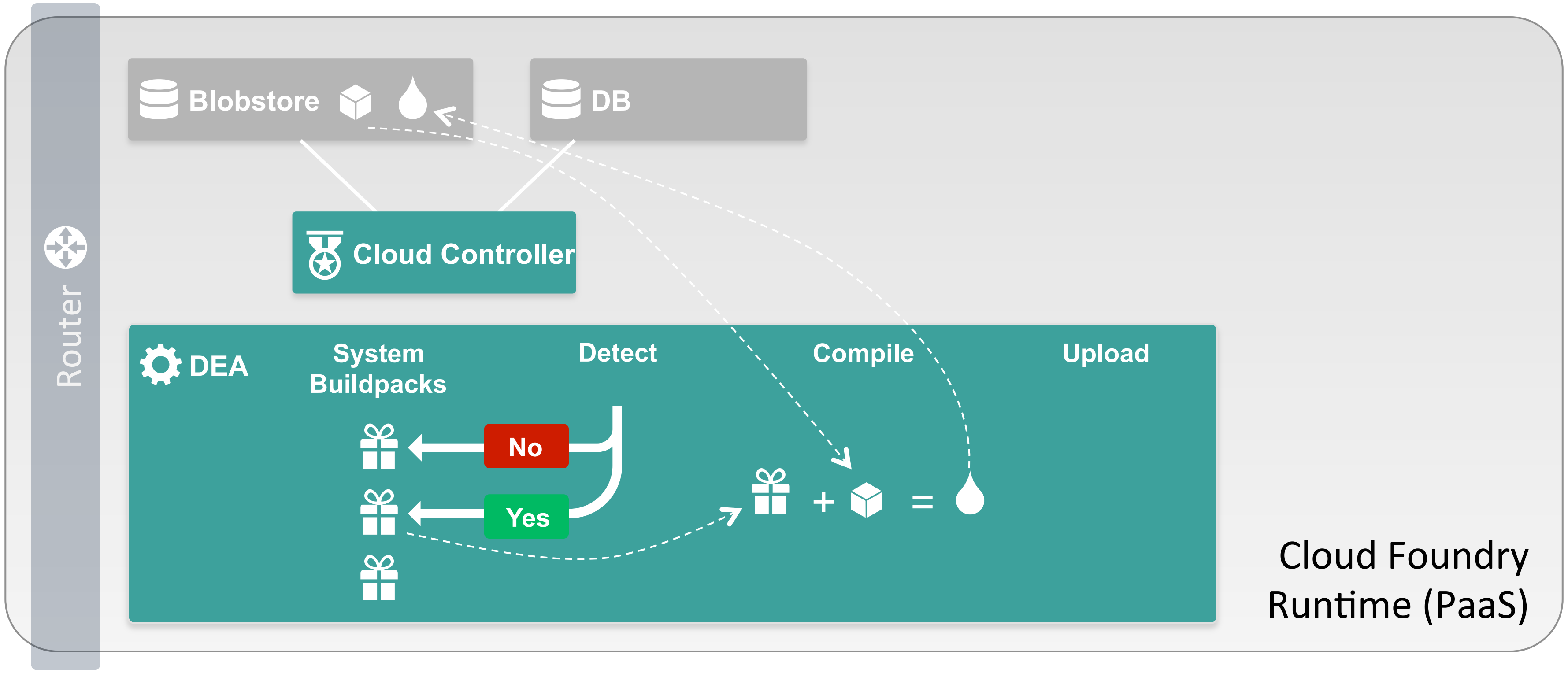
Creating and Binding a *Service*

Developer



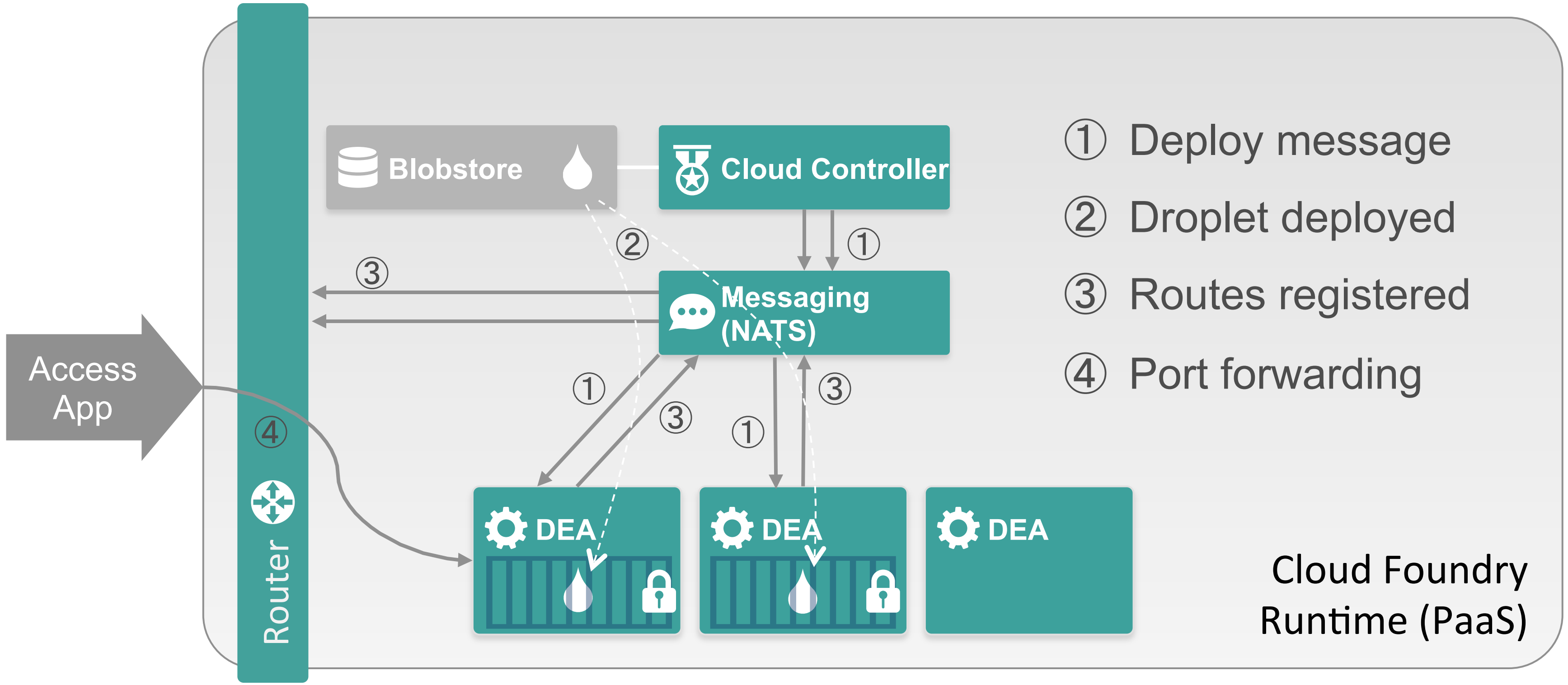
Stage an *Application*

Developer



Deploying an *Application*

Developer



① Deploy message

② Droplet deployed

③ Routes registered

④ Port forwarding

Monitoring and Replacing an *Application*

Developer

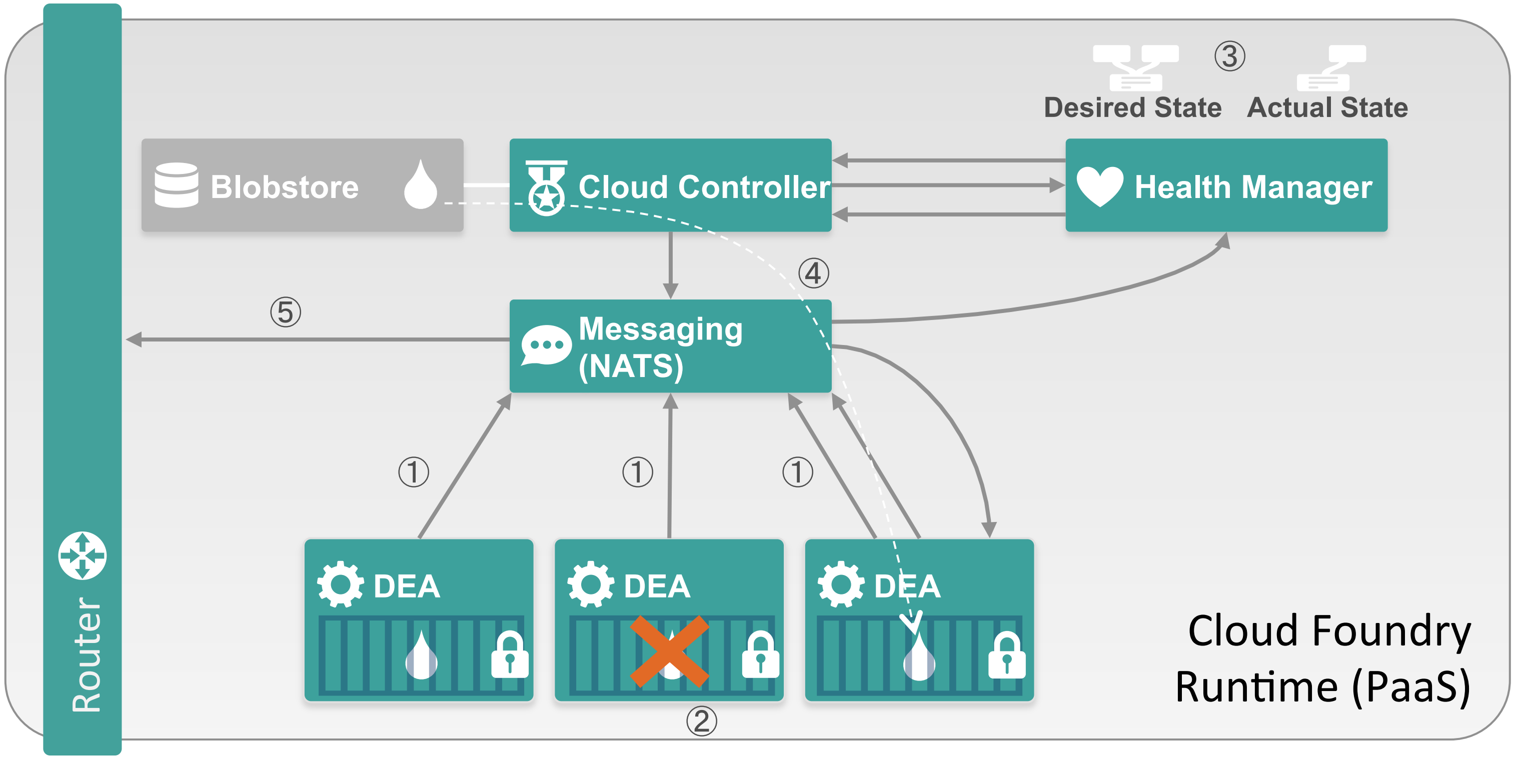
① App status messaging

② App instance fails

③ Health manager detects and advises

④ New app instance deployed

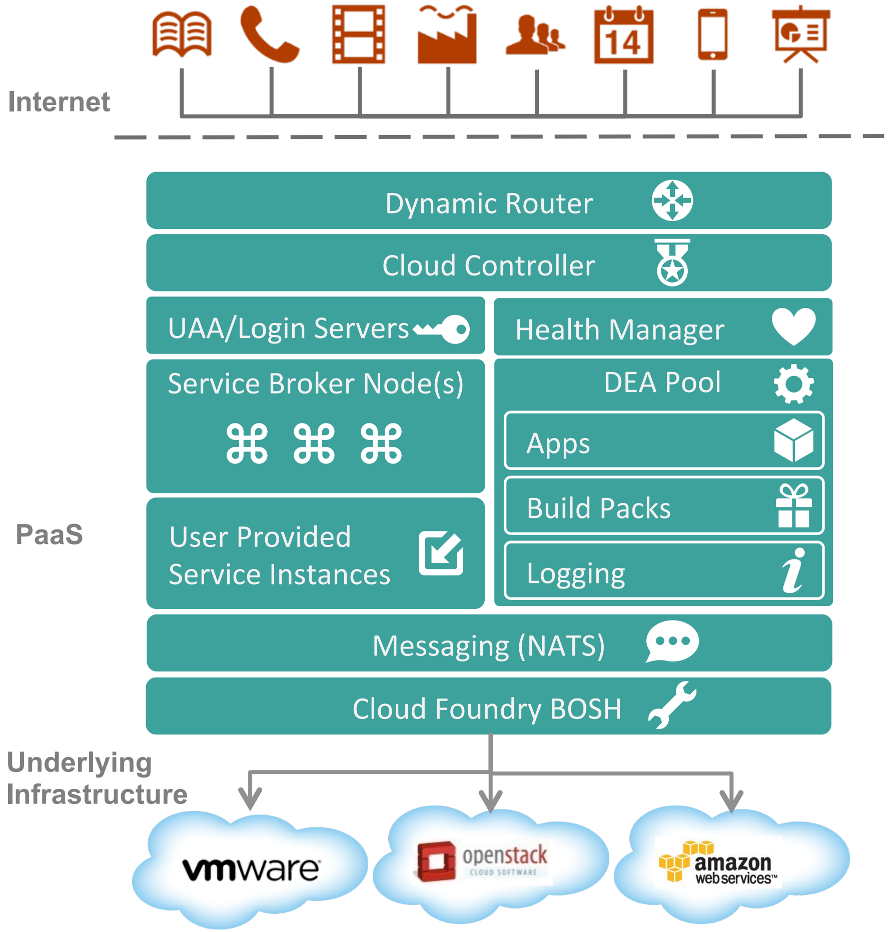
⑤ Routing table updated



Cloud Foundry Architecture

The **Cloud Foundry** platform is abstracted as a set of large-scale distributed services. It uses **Cloud Foundry Bosh** to operate the underlying infrastructure from IaaS providers (e.g., VMware, Amazon AWS, OpenStack).

Components are dynamically discoverable and loosely coupled, exposing health through HTTP endpoints so agents can collect state information (app state & system state) and act on it.



Router

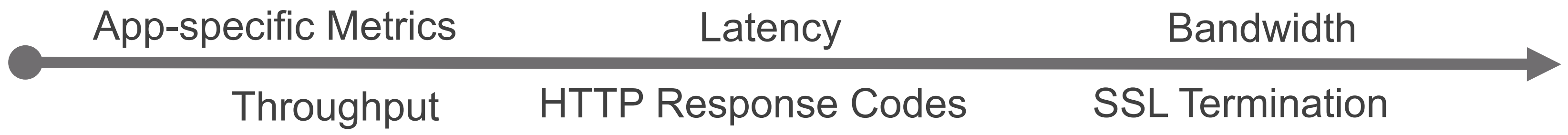
How It Works:

The router shapes and routes all external system traffic (HTTP/API) and application traffic from the internet/intranet. It maintains a dynamic routing table for each load-balanced app instance with IP addresses and ports.

Responsible For:

- Load balancing
- Maintaining an active routing table
- Access logs
- Supports web-sockets

Roadmap:



Cloud Controller

How It Works:

The Cloud Controller maintains command and control systems, including interface with clients (CLI, Web UI, Spring STS), account and provisioning control. It also provides RESTful interface to domain objects (apps, services, organizations, spaces, service instances, user roles, and more).

Responsible For:

- Expected App state, state transitions, and desired convergence
- Permissions/Auth
- Orgs/Spaces/Users
- Services management
- App placement
- Auditing/Journaling and billing events
- Blob storage

Roadmap:

Availability Zone Aware Placement

Richer Auditing with Queries and Filters

Oauth Scope and Role Mapping

OpenStack Swift Blob Configuration

UAA and Login Servers

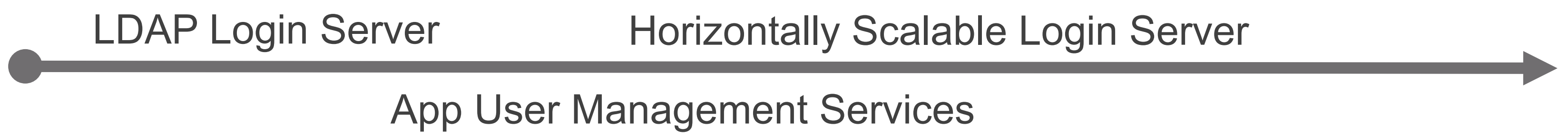
How It Works:

"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. Composed of: UAA Server, Command Line Interface, Library.

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

Roadmap:


LDAP Login Server
Horizontally Scalable Login Server
App User Management Services

Health Manager

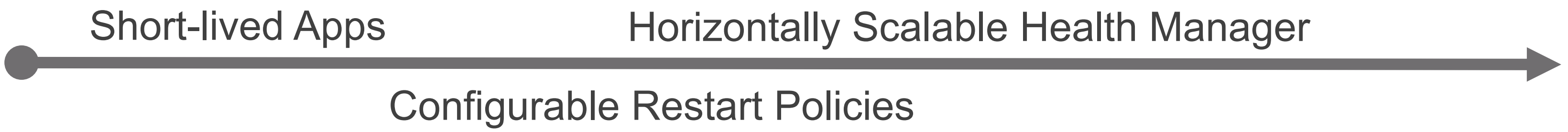
How It Works:

Health Manager monitors application uptime by listening to the NATS message bus for mismatched application states (expected vs. actual). The Cloud Controller publishes expected state and the DEAs publish actual state. State mismatches are reported to the Cloud Controller.

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 CC can do that!)

Roadmap:


Short-lived Apps
Horizontally Scalable Health Manager
Configurable Restart Policies

DEA

How It Works:

“Droplet Execution Agents” are secure and fully isolated containers. DEAs are responsible for an Apps lifecycle: building, starting and stopping Apps as instructed. They periodically broadcast messages about their state via the NATS message bus.

Responsible For:

- Managing Linux containers (Warden)
- Monitoring resource pools
  - Process
  - File system
  - Network
  - Memory
- Managing app lifecycle
- App log and file streaming
- DEA heartbeats (NATS to CC, HM)

Roadmap:

Placement Pools for Advanced Resource Allocation and Isolation

Evaluation of Windows .NET DEAs from Iron Foundry

Aggregated Logs Including All App Instances and App-related System Logs

App Log Draining with Syslog

Buildpacks

How It Works:

Buildpacks are Ruby scripts that detect application runtimes/frameworks/plugins, compile the source code into executable binaries, and release the app to an assigned DEA. Runtime components can be cached for faster execution of subsequent app pushes.

Responsible For:

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

(*) Cloud Foundry Buildpacks are compatible with Heroku

Roadmap:

vFabric Import Tool

Enhanced Caching

Buildpack Management Including Updates and Versioning

Messaging (NATS)

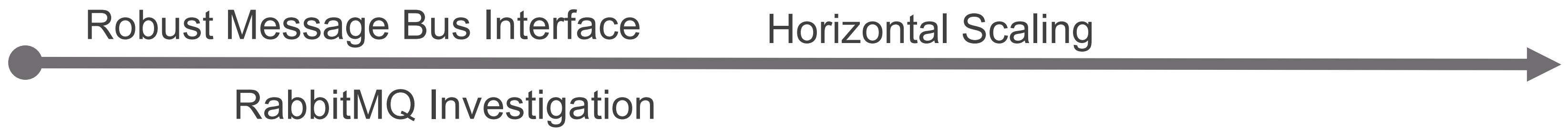
How It Works:

NATS is a fast internal messaging bus to manage system wide communication via a publish-and-subscribe mechanism.

Responsible For:

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

Roadmap:



⌘ Service Broker

How It Works:

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.).

Responsible For:

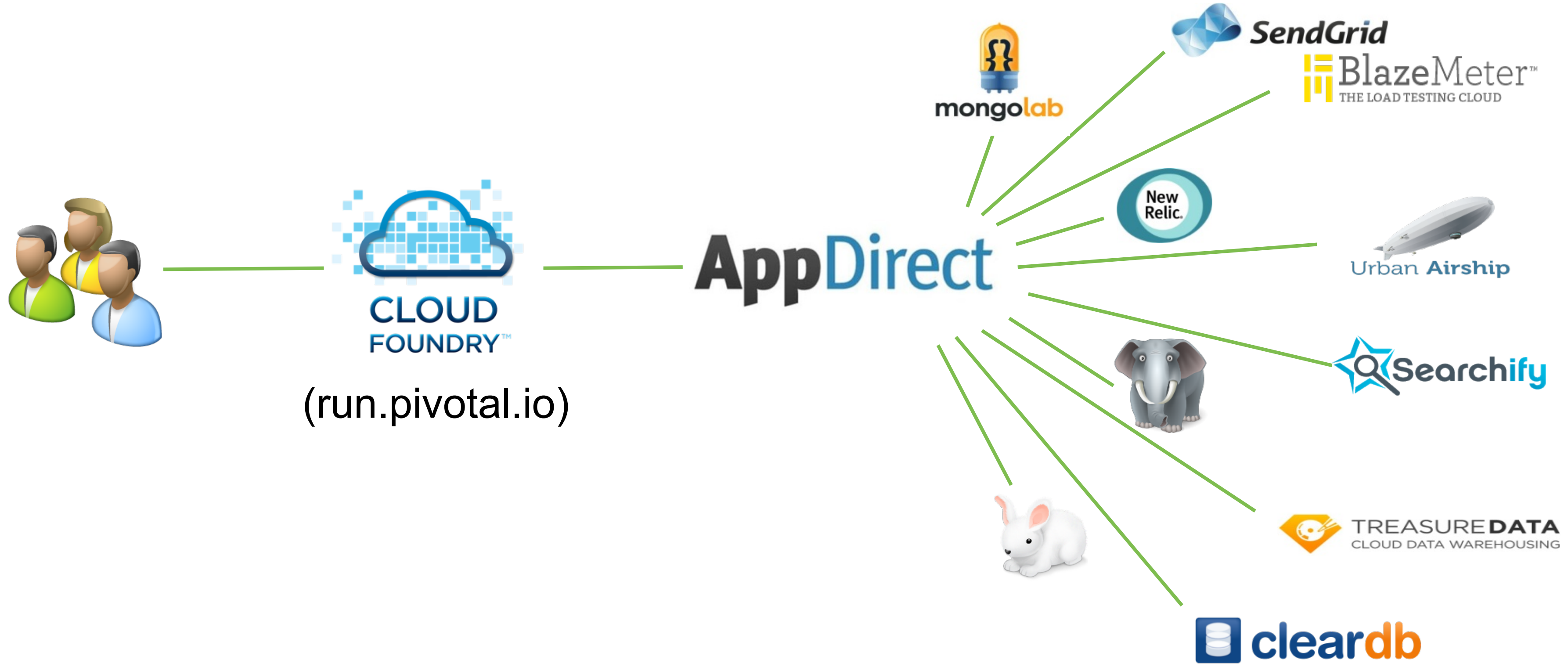
- Advertising service catalog
- Makes create/delete/bind/unbind calls to service nodes
- Requests inventory of existing instances and bindings from cloud controller for caching, orphan management
- SaaS marketplace gateway
- Implemented as HTTP enpoint, written in any language.

Roadmap:

Multi-Node Support

Asynchronous protocol

Service Broker Example: run.pivotal.io + AppDirect



User Provided Service Instances

How It Works:

UPSI (formerly “Service Connectors”) store meta-data in the Service Broker to enable Cloud Foundry to connect to local services that are NOT managed by Cloud Foundry (e.g., OracleDB, DB2, SQLServer, etc.)

Responsible For:

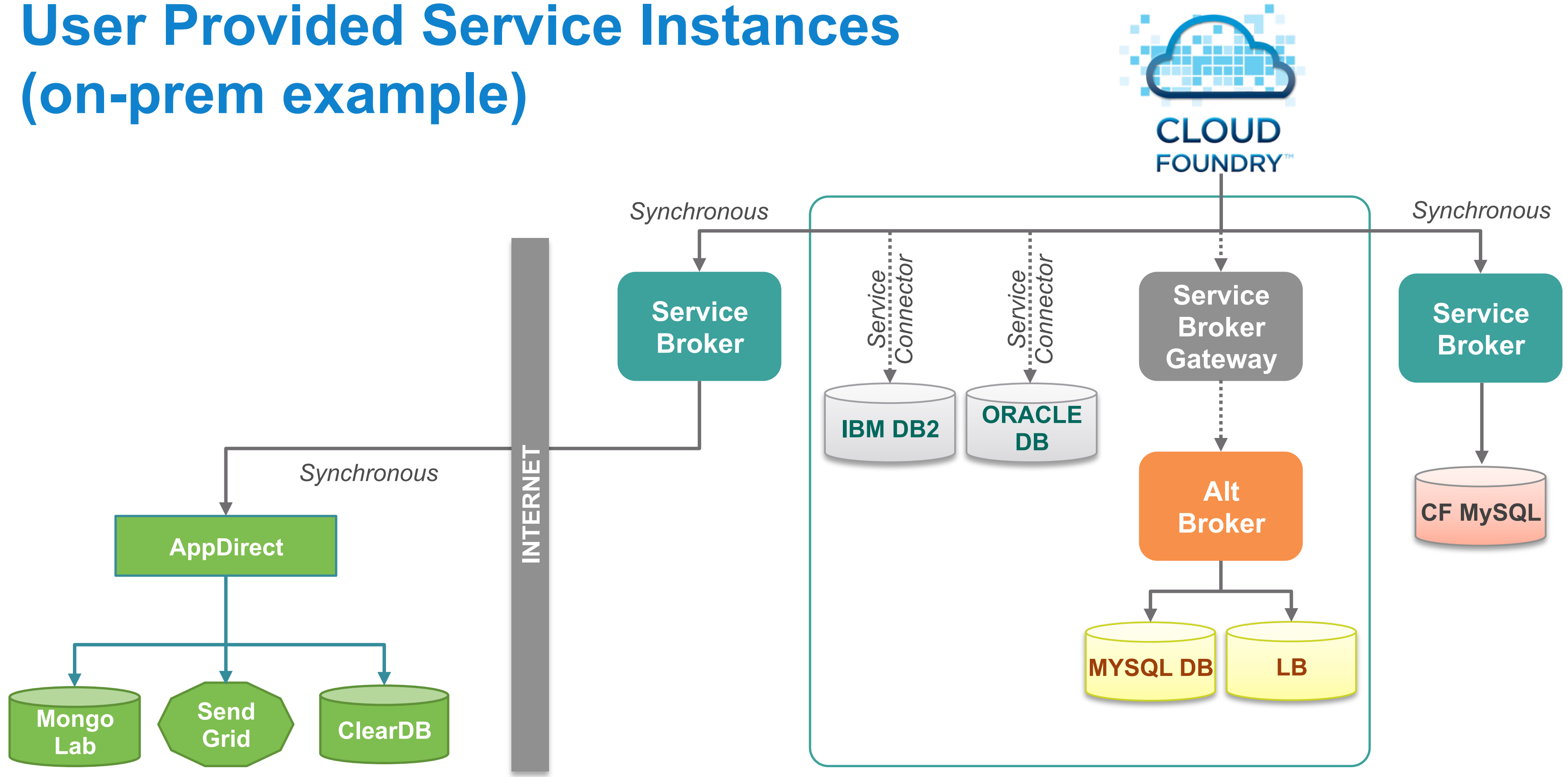
- Metadata management

Roadmap:

Service Type Templates (OracleDB, DB2, SQLServer, MQSeries)

Investigate Sharing Service Instances Across Spaces

User Provided Service Instances (on-prem example)



To: Pushing apps to the cloud with a few easy verbs

Operator

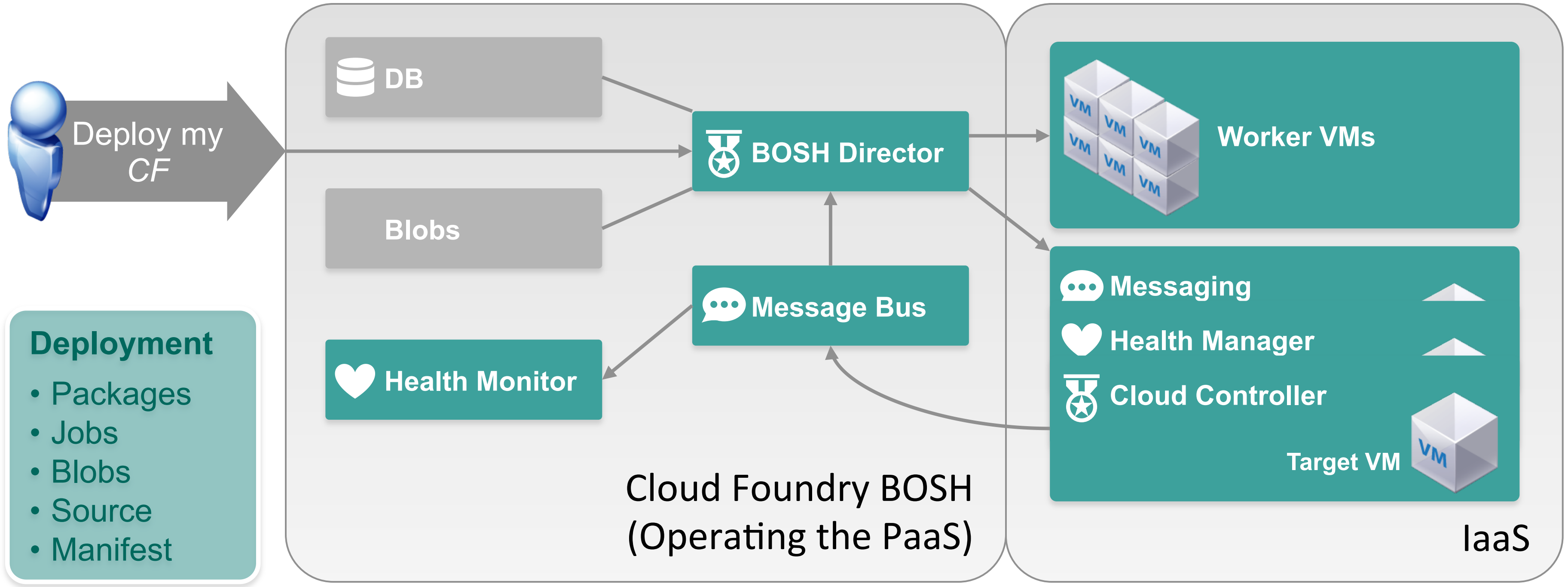
```
cf-iaas.yml
provision <my cloud>
add_capacity <my cloud>
```

Developer

```
target <my cloud>
push <my app>
create <my services>
bind <my services>
scale <my app> +100
```

Deploying the CF Runtime with Cloud Foundry *BOSH*

Operator



Deployment

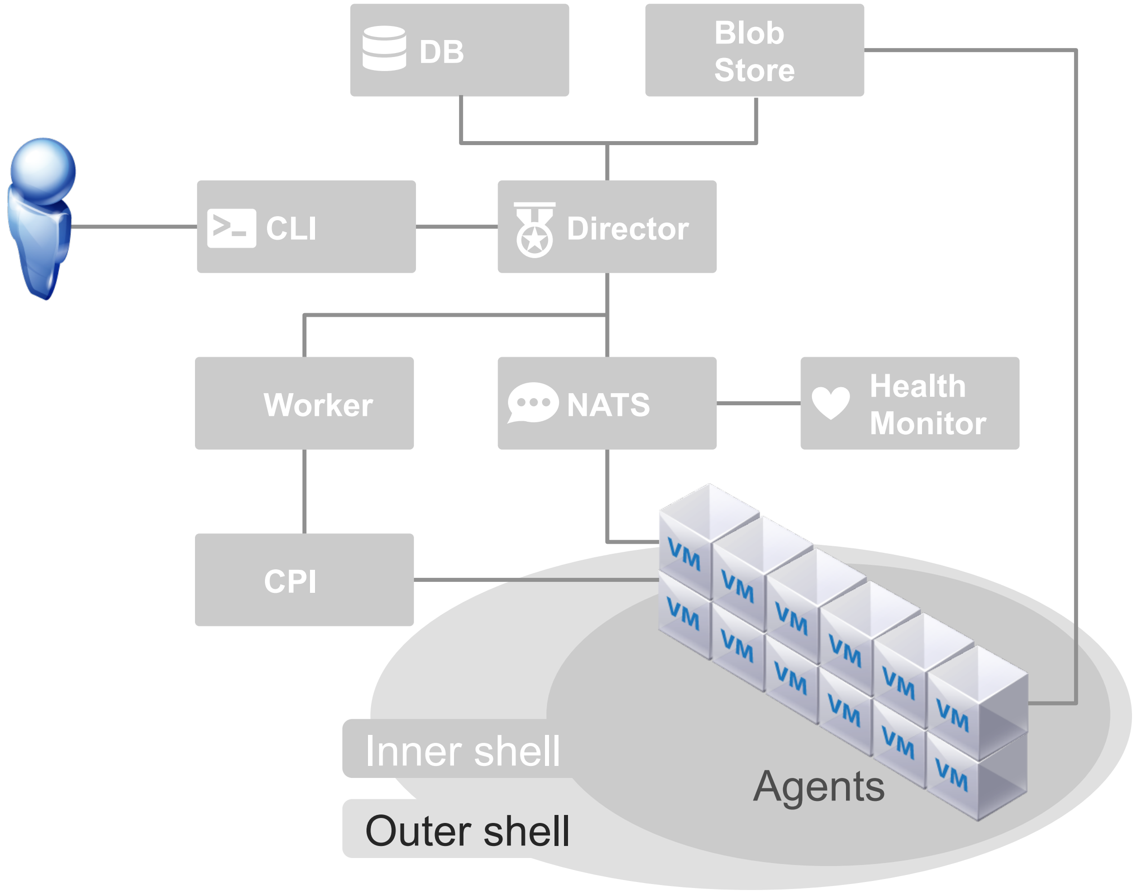
- Packages
- Jobs
- Blobs
- Source
- Manifest

BOSH (Outer Shell) Logical View

Deploys and manages large scale distributed systems. **BOSH** provides the means to go from deployment (i.e., Chef/Puppet) to VM creation and management (i.e., cloud CPI). It includes interfaces for vSphere, vCloud, AWS and OpenStack. Additional CPI can be written for alternative IaaS providers.

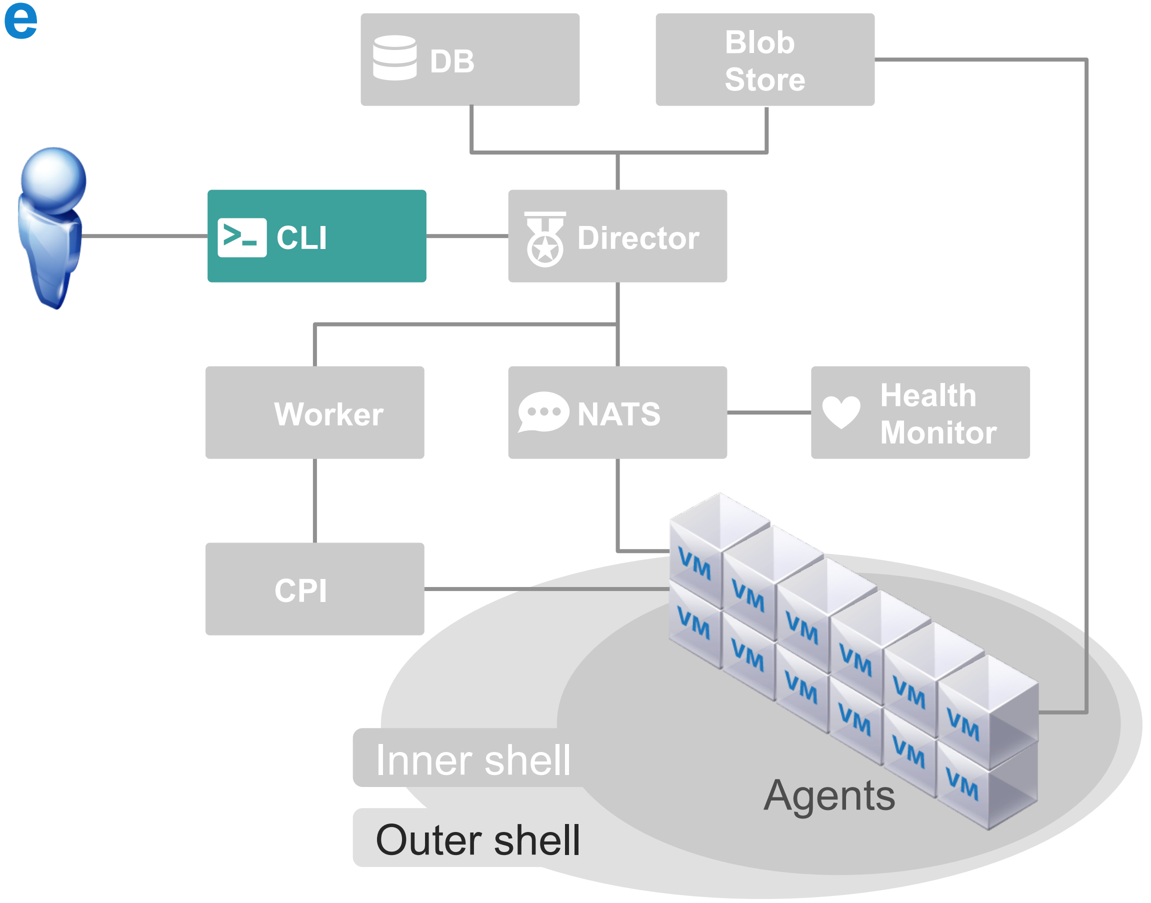
Key Elements:

- CLI
- Director
- Blobstore
- Workers
- Message Bus
- Health Monitor
- IaaS CPI
- Agents



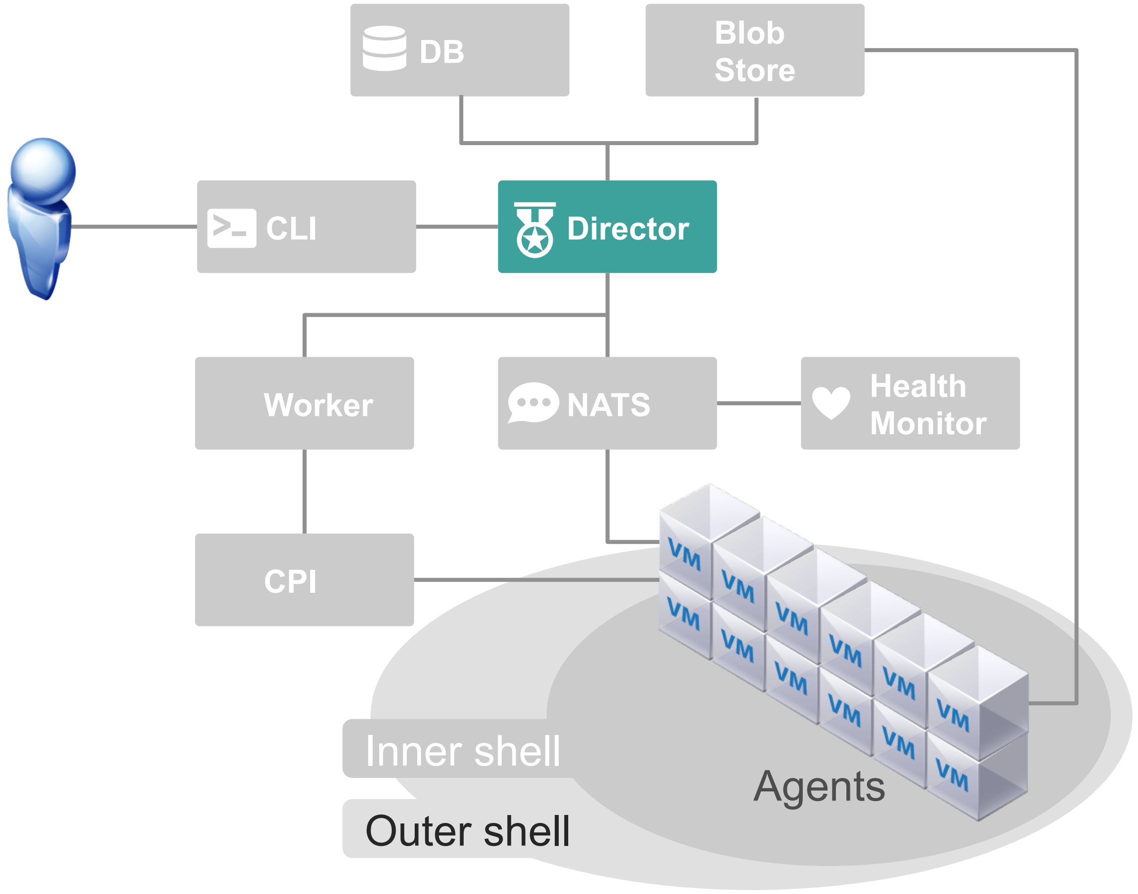
BOSH: Command Line Interface

The Command Line Interface is how users interact with BOSH using a terminal session to do a deployment, create and upload releases, and upload ‘stemcells’ (i.e. a VM template with an embedded Agent).



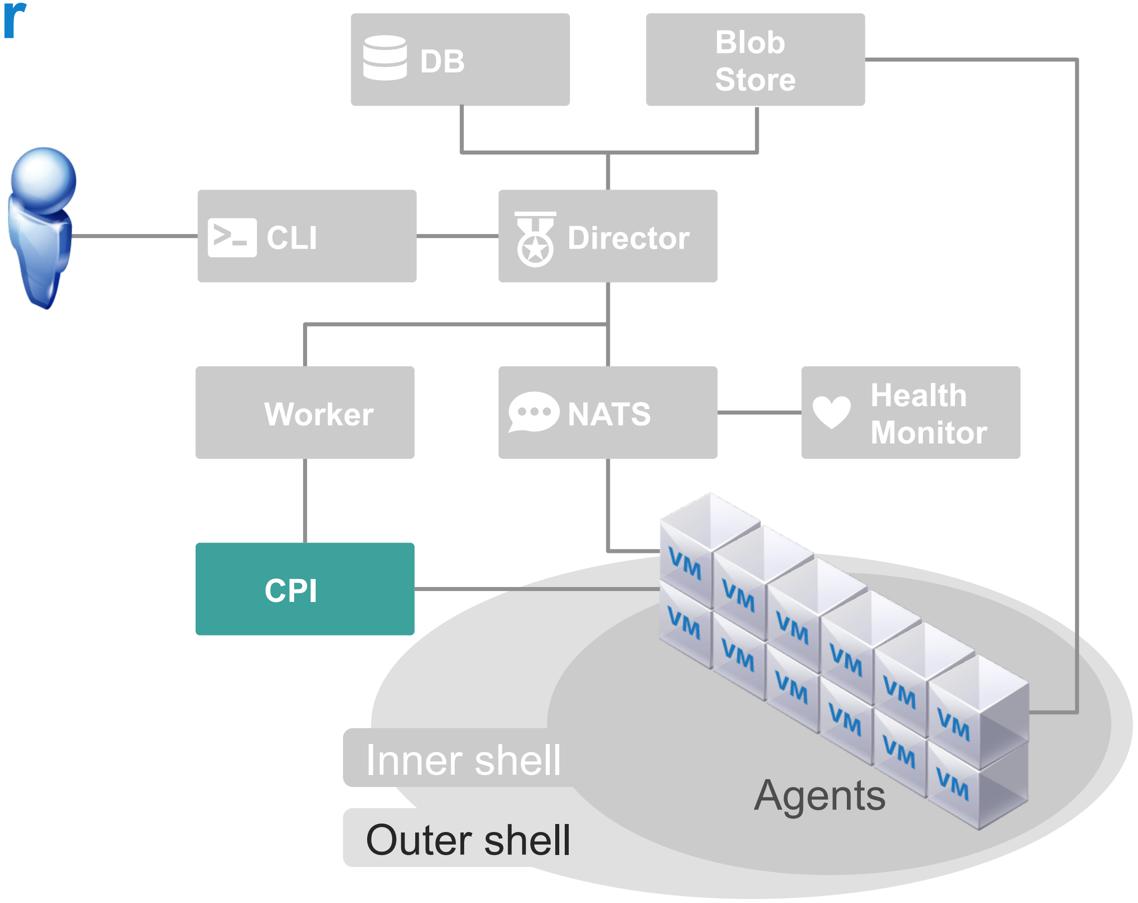
BOSH: Director

The **core orchestrating** component in BOSH which controls creation of VMs, deployment, and other life cycle events of software and services. Command and control is handed over to the the Director-Agent interaction after the CPI has created resources.



BOSH: Cloud Provider Interface (CPI)

The core BOSH engine **is abstracted from any particular IaaS**. IaaS interfaces are implemented as plugins to BOSH. Currently, BOSH supports both VMware vSphere and Amazon Web Services. These CPIs allow for automated VM and storage disk provisioning, and network management.



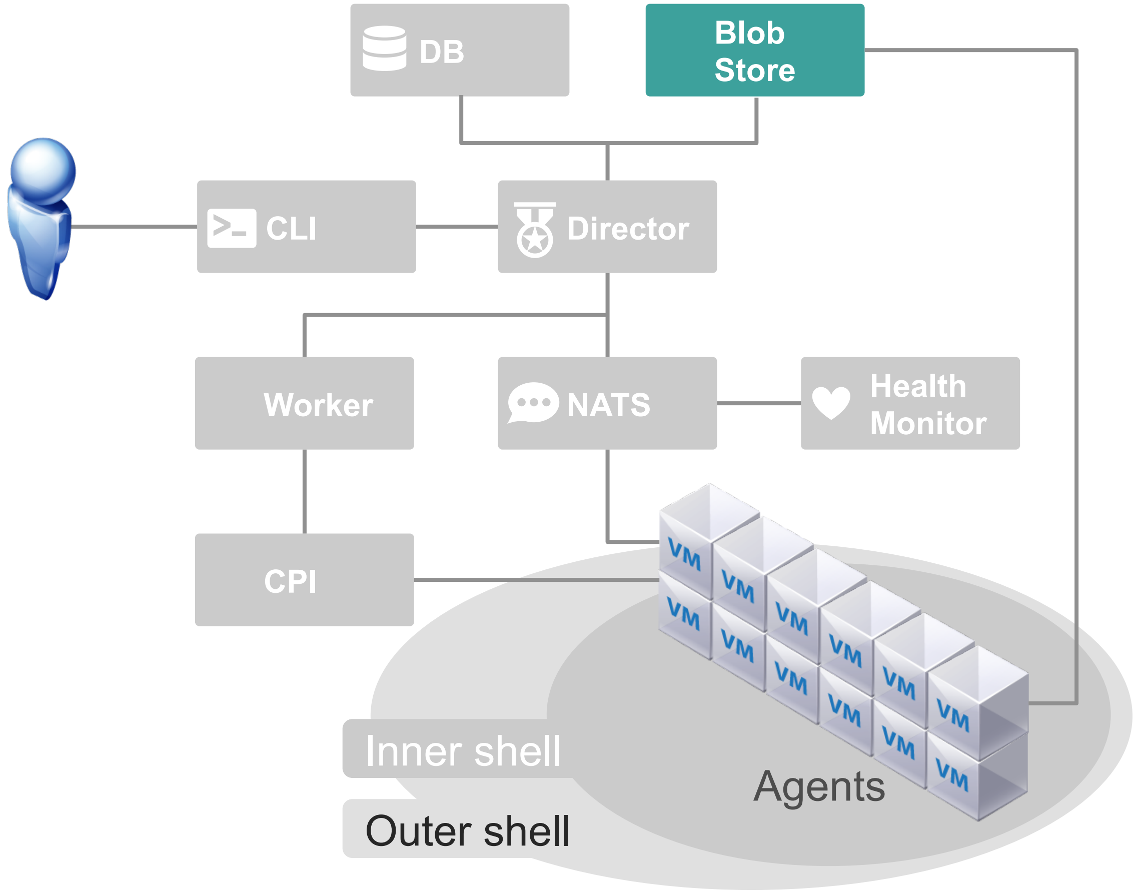
BOSH: Cloud Provider Interface

```
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)
```

IaaS Neutral

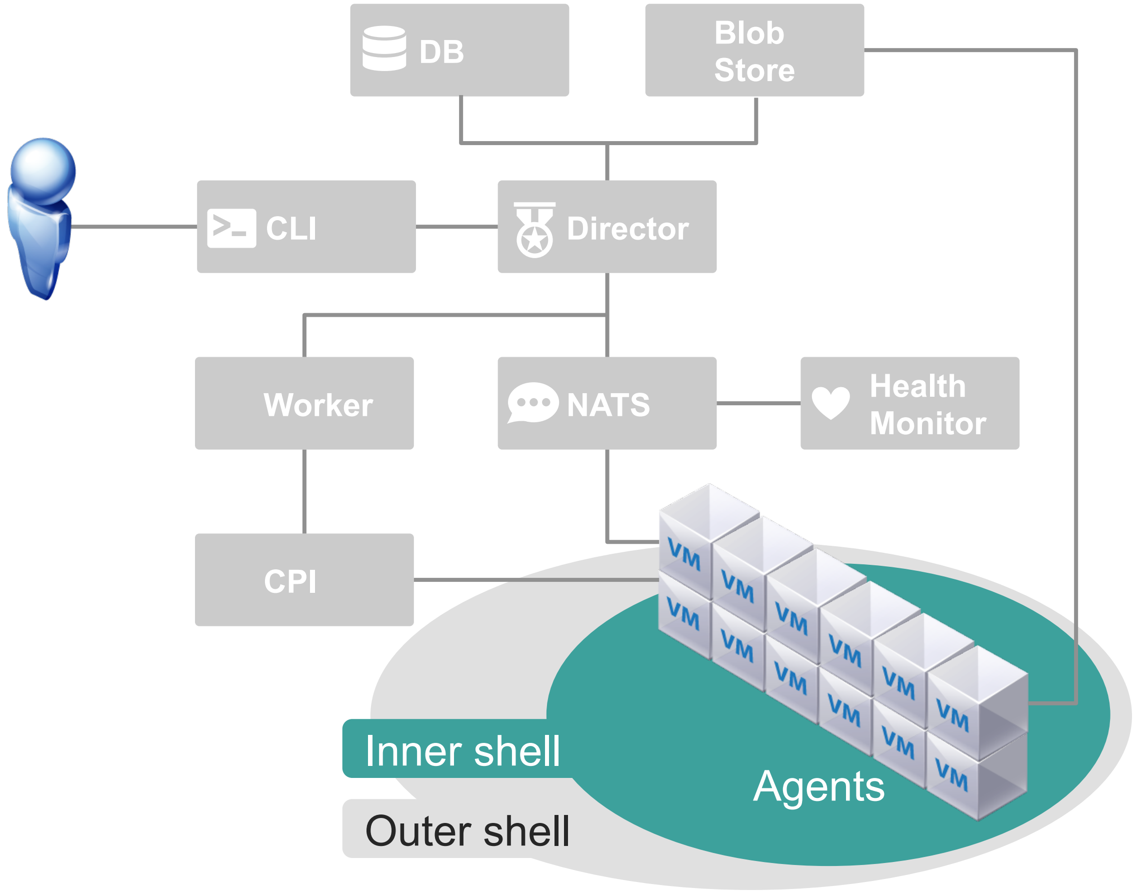
BOSH: Blobstore

Used to **store the content of Releases, Jobs and Packages** in their source form as well as the compiled image. When you deploy a Release, BOSH will orchestrate the compilation of packages and store the result in Blobstore. When BOSH deploys a Job to a VM, the Agent will pull the specified Job and associated Packages from the Blobstore.



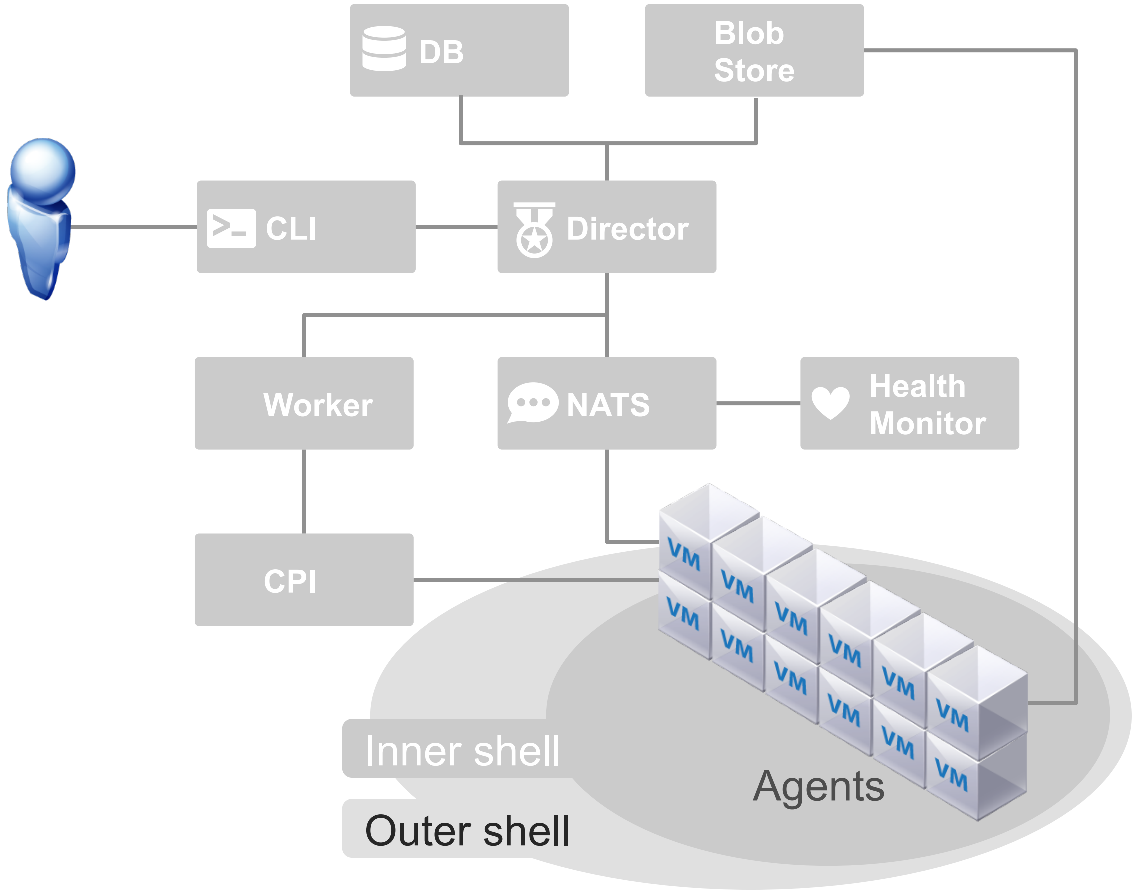
BOSH: Agents

Every VM contains an Agent. Through the Director-Agent interaction, **VMs are given Jobs,** or roles, within Cloud Foundry. If the VM's job is to run MySQL, for example, the Director will send instructions to the Agent about which packages must be installed and what the configurations for those packages are.



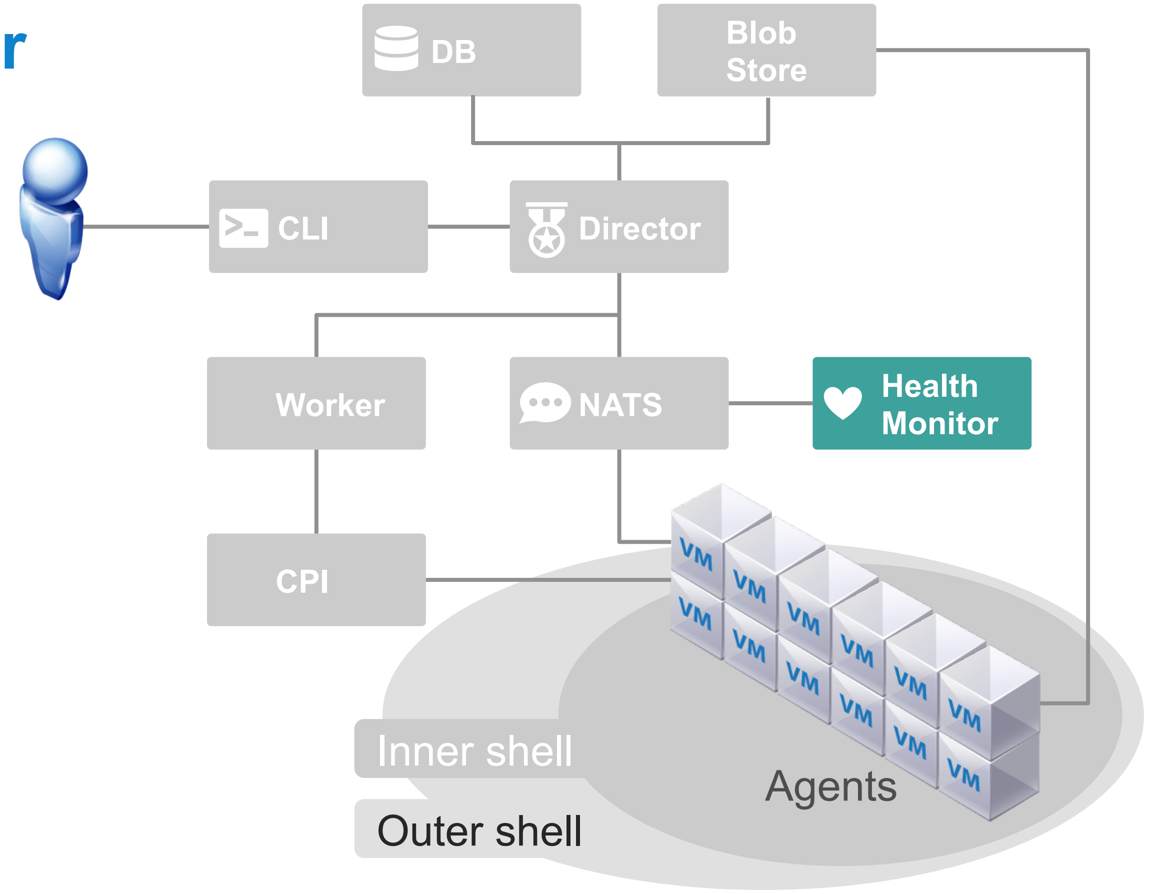
BOSH: Stemcells

A Stemcell is a VM template with an embedded Agent. Stemcells are uploaded using the CLI and used by the Director when creating VMs through the CPI. When the Director creates a VM through the CPI, it will pass along configurations for networking and storage, as well as the location and credentials for the Message Bus (NATS) and the Blobstore.



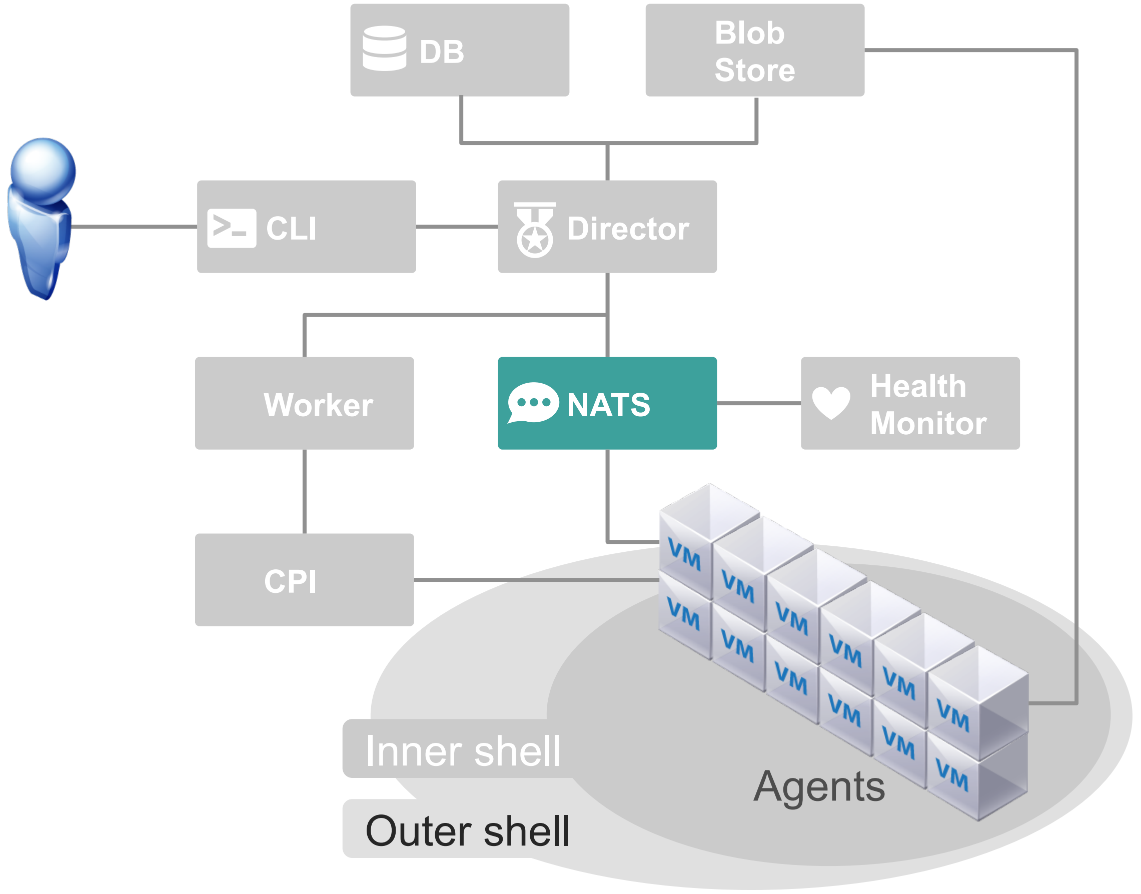
BOSH: Health Monitor

Receives health status and life cycle events from Agents and can send alerts through notification plugins (such as email) to operations staff.



BOSH: NATS

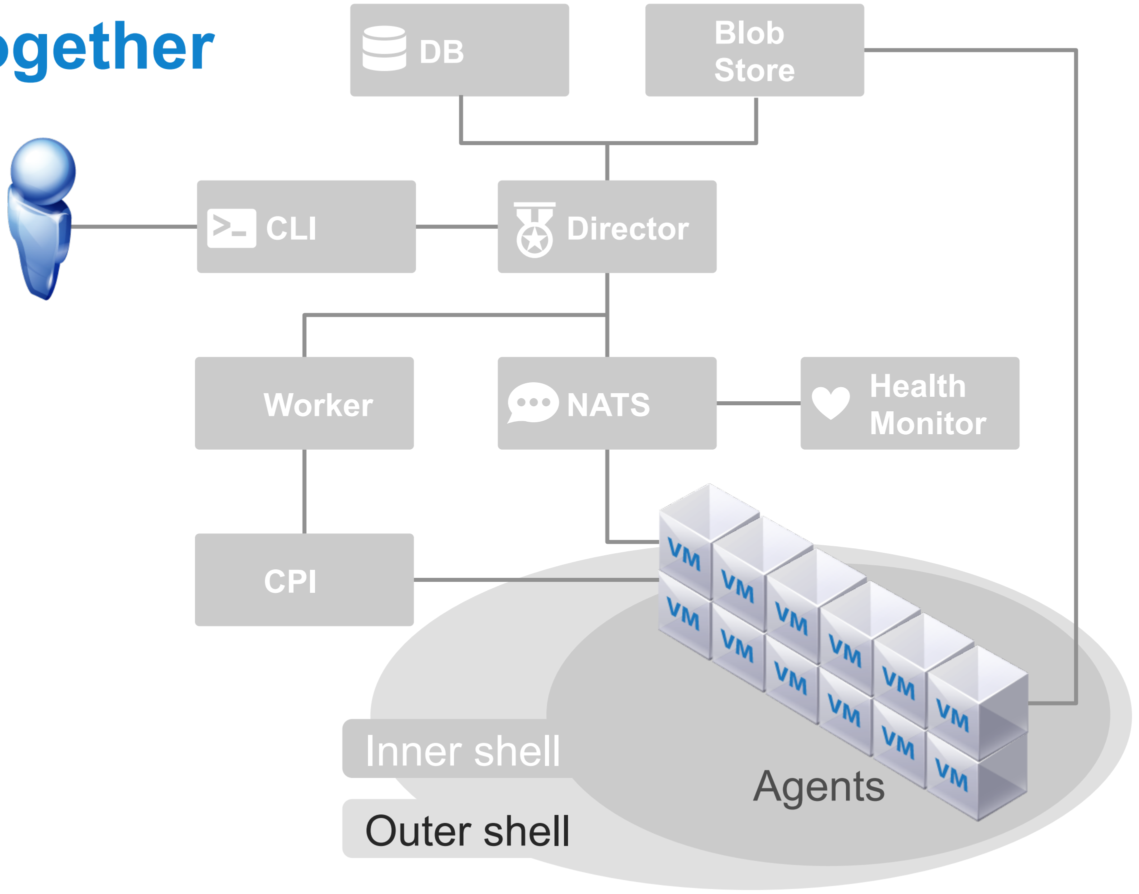
BOSH components use NATS, a lightweight pub sub messaging system, for command and control.



BOSH: Putting it all together

When you deploy Cloud Foundry the following sequence of steps occur:

1. Target a BOSH director using CLI
2. Upload a Stemcell
3. Get a Release from a repo
4. Create a deployment manifest
5. BOSH Deploy Cloud Foundry:
   - Prepare deployment
   - Compile packages
   - Create and bind VMs
   - Pull in job configurations
   - Create needed job instances – this is where things get pushed live


DB
Blob Store
CLI
Director
Worker
NATS
Health Monitor
CPI
VM
Inner shell
Outer shell
Agents