



Cloud Foundry Architecture

What's Inside the Box?

Overview of Architectural Components
DIEGO (or DEA)

Overview

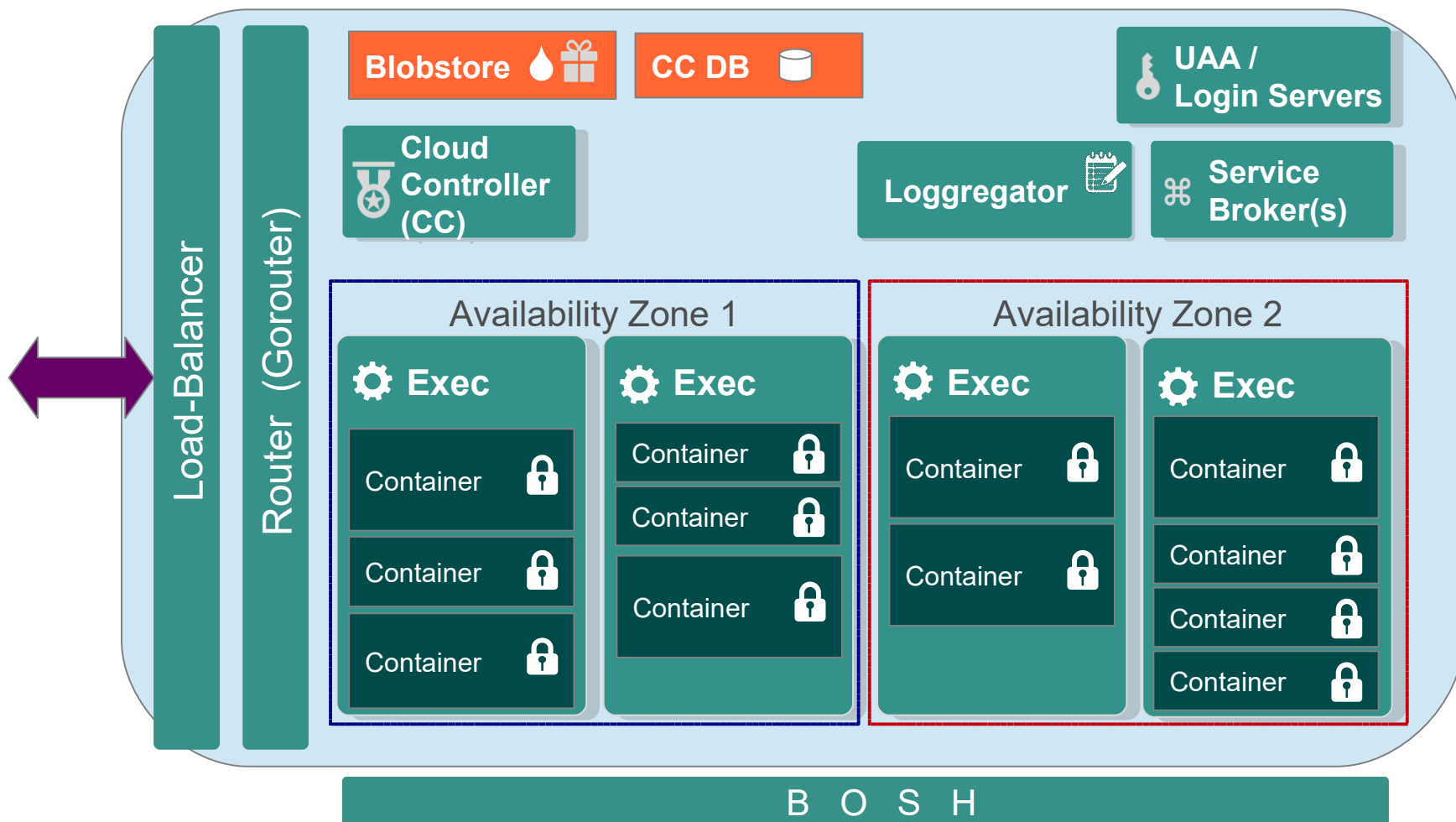
- After completing this lesson, you should understand:
 - The components that work together inside Cloud Foundry
 - The different levels of application containment
 - And what they do
 - The differences between the Diego and former DEA and architectures

Roadmap

- **Architectural Components**
 - **Common Components**
 - Introduction to Elastic Runtime (Diego)
 - Elastic Runtime Internals
 - Appendix: Previous Elastic Runtime Architecture (DEAs)

Pivotal CF Components

HA Zone 1
HA Zone 2



Exec – Execution Agent

UAA – User Authorization & Authentication



Pivotal™



Load Balancer (HA Proxy)

Description

- HAProxy is separate component from the Router
 - Sits in front of the router
 - Only one instance allowed
 - Single-point of failure
- Cloud Foundry can only have one IP address visible to outside world
 - This is the Load Balancer
 - It passes requests to the Router(s) to send to the right CF component

Responsible For

- Load-balancing across multiple routers (if used)
- SSL Termination
- **Note:** Typically need to configure your own HAProxy or other load-balancer (such as F5 or NSX) for improved performance, SSL handling and high availability,



Router

Description

- Routes all incoming HTTP traffic
 - System traffic (cf commands)
 - Application traffic
- Maintains dynamic routing table for each load-balanced app instance
 - Knows IP addresses and ports.
- Multiple routers possible
 - Configured by admin in Ops Manager
- From CF 1.4 rewritten in Go – the *GoRouter*

Responsible For

- Load balancing across application instances
- Maintaining an active routing table
- Access logs
- Supports web-sockets
- In CF 1.6 will *also* do SSL termination



Cloud Controller

Description

- Command and Control
 - Responds to clients (CLI, Web UI, Spring STS)
 - Account and provisioning control.
- Provides RESTful interface to domain objects
 - Apps, services, organizations, spaces, service instances, user roles, and more ...
- Multiple Cloud Controllers possible
 - Configurable by Admin

Responsible For

- Expected Application state, state transitions, and desired convergence
- Permissions/Authorization
- Organizations/Spaces/Users
- Services management
- Application placement
- Auditing/Journaling and billing events
- Blob storage



BlobStore

Description

- Storage for binary large objects
- Eliminates need for upload / re-staging when scaling applications
- Currently NFS mounted storage
 - Or Amazon S3 store

Responsible For

- Stores uploaded application packages (cf push)
- Stores Droplets

Cloud Controller Database (CCDB)



Description

- Storage for application metadata
- Used exclusively by the Cloud Controller

Responsible For


- Stores information on
 - Application name
 - # of instances requested
 - Memory limit
 - Routes
 - Bound services
- A Postgres DB instance



Availability Zones

Description

- Separate shared points of failure such as a power, network, cooling, roof, floor.
- Allows Cloud Controller to locate instances on separate zones to boost redundancy
- Marked on diagram as:

 HA Zone 1
 HA Zone 2

Responsible For

- Enhancing application redundancy
- Provides one of the layers of High Availability



Service Broker

Description

- Provide an interface for native and external 3rd party services
 - Service processes run on Service Nodes
 - Or with external as-a-service providers
- *Examples*
 - Mail server
 - Database,
 - Messaging
 - Many more ...
- Typically one service broker for each marketplace service.

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



Loggregator

Description

- Master logging process
 - Accepts logs from application instances
 - Accepts logs from other CF components
- Make log data available to external *sinks*
- Uses Diego's *Doppler* server since CF 1.4

Responsibilities

- Non-Persistent, temporary log-storage
- Accumulates logs from multiple sources and aggregates by application
- Provides logs to external sources such as **cf logs** or App Manager console
- Supports log “drains” to syslog servers



UAA and Login Services

Description

- UAA = “User Authorization and Authentication”
- Provides identity, security and authorization services.
- Manages third party OAuth 2.0 access credentials
- Can provide application access & identity-as-a-service for CF apps
- Composed of
 - UAA Server
 - Command Line Interface
 - Library.
- Multiple UAA/Login Servers possible

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



Cloud Foundry BOSH

Description

- Tool chain for managing large scale distributed systems
 - Release engineering
 - Deployment and lifecycle management
- Continuous and predictive updates with minimal downtime
- Control primitives (CPI) written for each underlying infrastructure provider
 - The 'glue' between CF and underlying IaaS

Responsible For

- IaaS installer
 - Currently vSphere
- VM creation and management
- Continuous and predictive updates with minimal downtime
- High Availability
 - Restarts failed CF *internal* processes
 - Restarts Execution Agent VMs
- CPI (Cloud Provider Interface) to control underlying infrastructure (IaaS) primitives

Roadmap

- **Architectural Components**
 - Common components
 - **Introduction to Elastic Runtime (Diego)**
 - Elastic Runtime Internals
 - Appendix: Previous Elastic Runtime Architecture (DEAs)

What Is DIEGO?

- Internal project name for Elastic Runtime enhancements
- Written to support
 - Multiple operating systems on the Execution Agent VMs
 - In particular MS Windows and .NET
 - More scalable architecture
 - Avoid Cloud Controller and Health Management becoming bottlenecks
 - Support containers other than Warden
 - In particular Docker

Be careful, many online references still discuss the older DEA architecture

DIEGO vs DEA

- **DEA** = Original Cloud Foundry internal architecture
 - Apps in Warden containers in *Droplet Execution Agents* (DEAs)
 - *Health Manager* handled restart of failed app instances
 - Code written in Ruby, running on Linux
 - See Appendix (at end of this chapter) for details
- **DIEGO** = DEA in Go
 - Misleading name – it became a lot more than that!
 - Rewrite of DEA *and other components* in Google GO
 - Major refactoring of the Cloud Controller and internals
 - Available since 1.5 ,default from 1.6
- *But from the outside, CF works the same way*

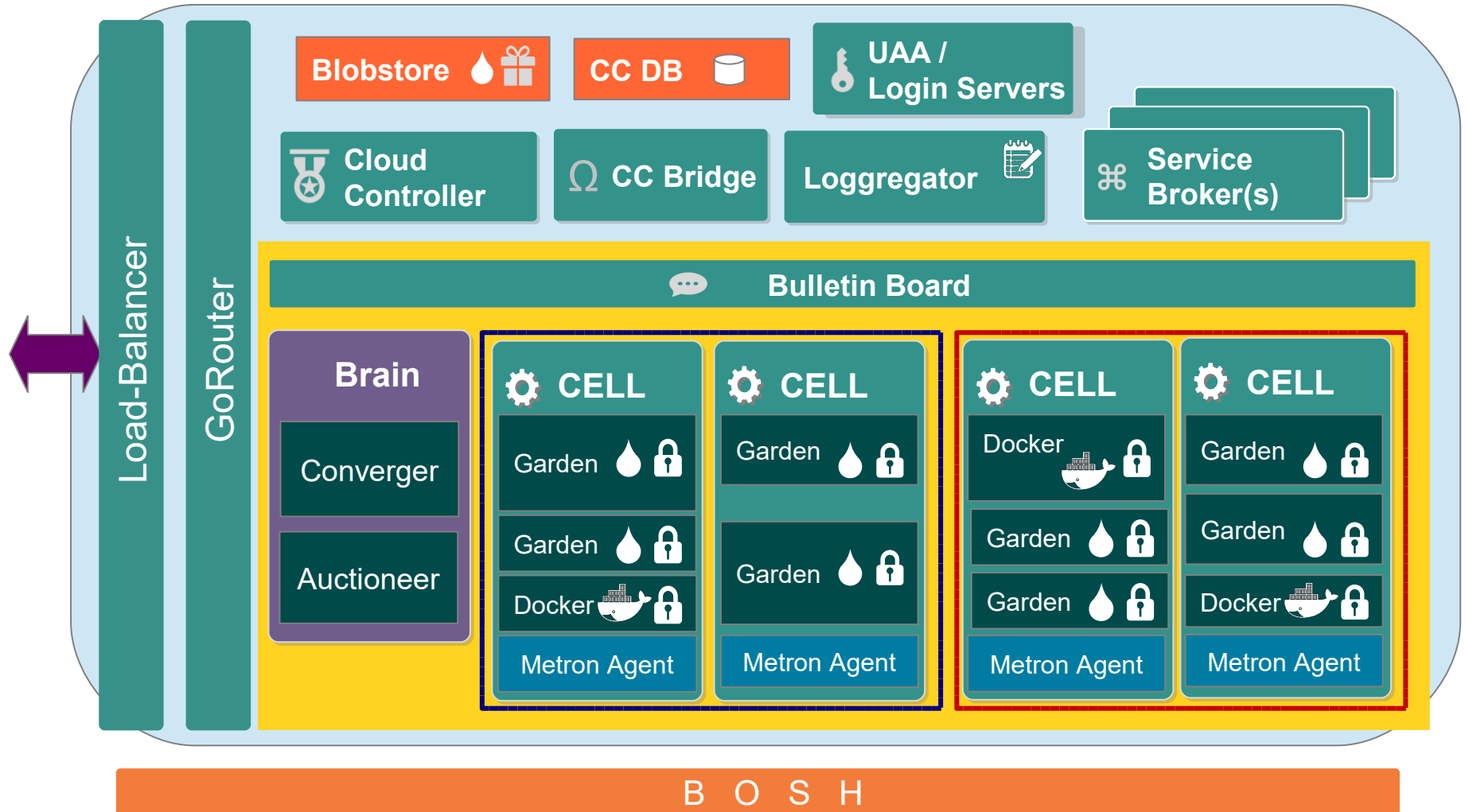
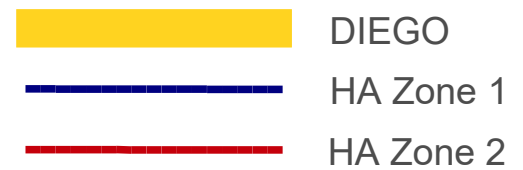
Roadmap

- **Architectural Components**
 - Common components
 - Introduction to Elastic Runtime (Diego)
 - **Elastic Runtime Internals**
 - Appendix: Previous Elastic Runtime Architecture (DEAs)

DIEGO Summary

- Elastic Runtime internal architecture
 - Default since V1.6 (optional in V1.5)
 - Refactors Cloud Controller
 - Delegates to new components in the “Brain”
 - Auctioneer, Converger
 - Cells replaces DEAs
 - Warden containers can have multiple implementations
 - Not *just* for running Droplets
 - Not *just* Linux
- From a CF user perspective – *nothing changes*

Cloud Foundry – Diego





CC Bridge

Description

- Translate app-specific requests into the generic language of LRP and Task
 - For example requests to stage, start, stop, scale applications
- Allows existing CF components to interact with Diego

Responsibilities

- Places requests (actions) onto the Bulletin Board to be actioned by a Cell's "Executor"
- Ensures that these actions take place (eventually)
- Tracks running processes
 - Provides info on running tasks and LRPs *to* Cloud Controller
 - Accepts desired state of tasks and LRPs *from* Cloud Controller

LRP = *Long Running Process* – typically an application instance



Bulletin Board System (BBS)

Description

- Holds CF “actions”
 - Typical CF actions are to start/stop application instances (LPS) or one-off task processes
- Essentially an action queue, currently backed by *etcd*
 - Other implementations likely
- Decouples control components (like the CC) from Diego
 - CC manages applications
 - Diego talks tasks and LRPs

Responsibilities

- Central co-ordination for activities (actions) within CF

etcd



Description

- A fast distributed open-source key-value store
- Written in Go and uses the Raft protocol*

Responsibilities

- Cluster coordination and state management
- The *consistent* store at the heart of Diego

* https://www.youtube.com/embed/06cTPhi-3_8

Converger



Description

- Makes sure that the system is *eventually* consistent
- Refactoring of instance recovery functionality that used to be in the the old DEA Health Manager and the Cloud Controller

Responsibilities

- Compare desired system state with actual system state
 - Take remedial action if they are different so desired and actual *converge*
- Place start (scale up), stop (scale down) and restart (instance recovery) requests on the Bulletin Board
- Spots requests on BBS too long
 - Forces BBS to try them again

Auctioneer



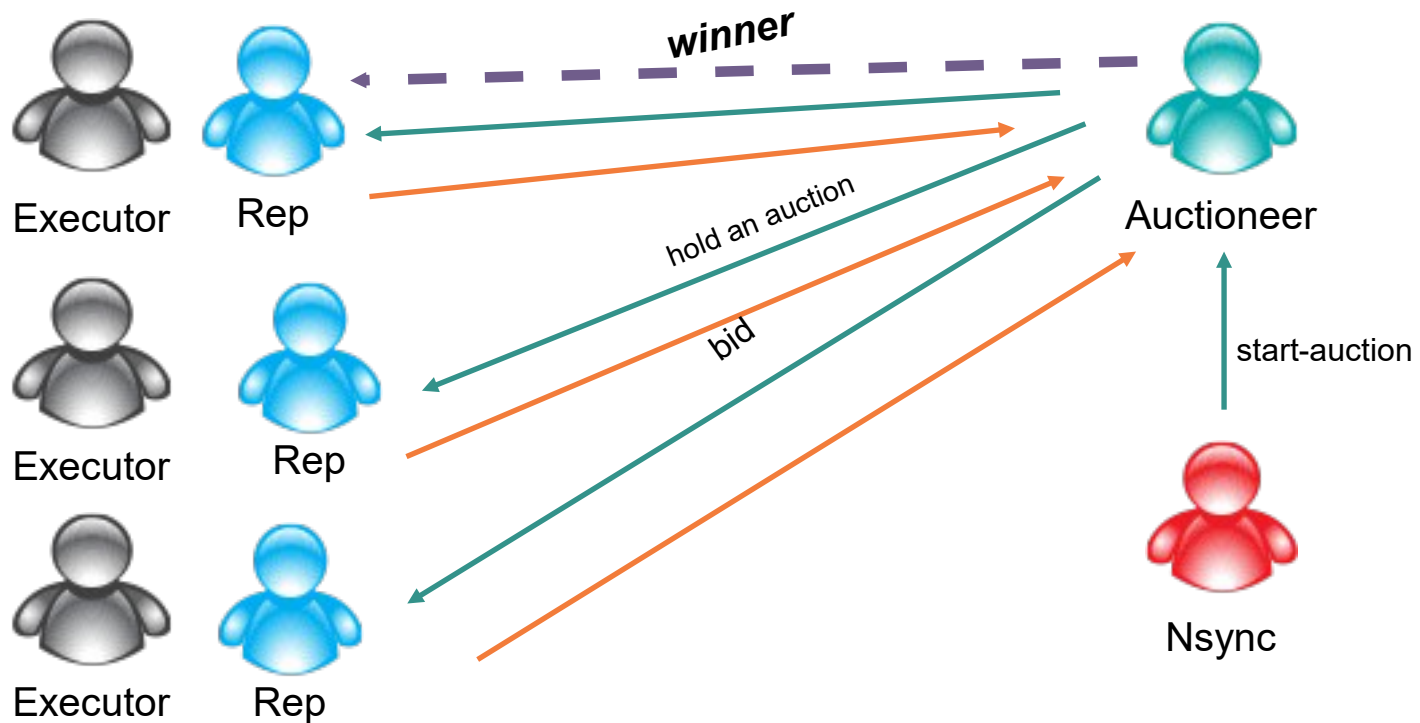
Description

- Determining which Cell to run an application instance
 - Decision uses an Auctioning placement algorithm
 - Complex because distributed and asynchronous
- Also determines which instance to stop when scaling down
- One-off tasks not managed by auction yet
 - *But will in the future*

Responsibilities

- Gather state from the appropriate sources of truth at decision-time
 - Run algorithm to decide
- Tell the cell that wins the auction to either start or stop the requested application instance
 - By placing the action on the BBS

Auctioneering Algorithm



Cells



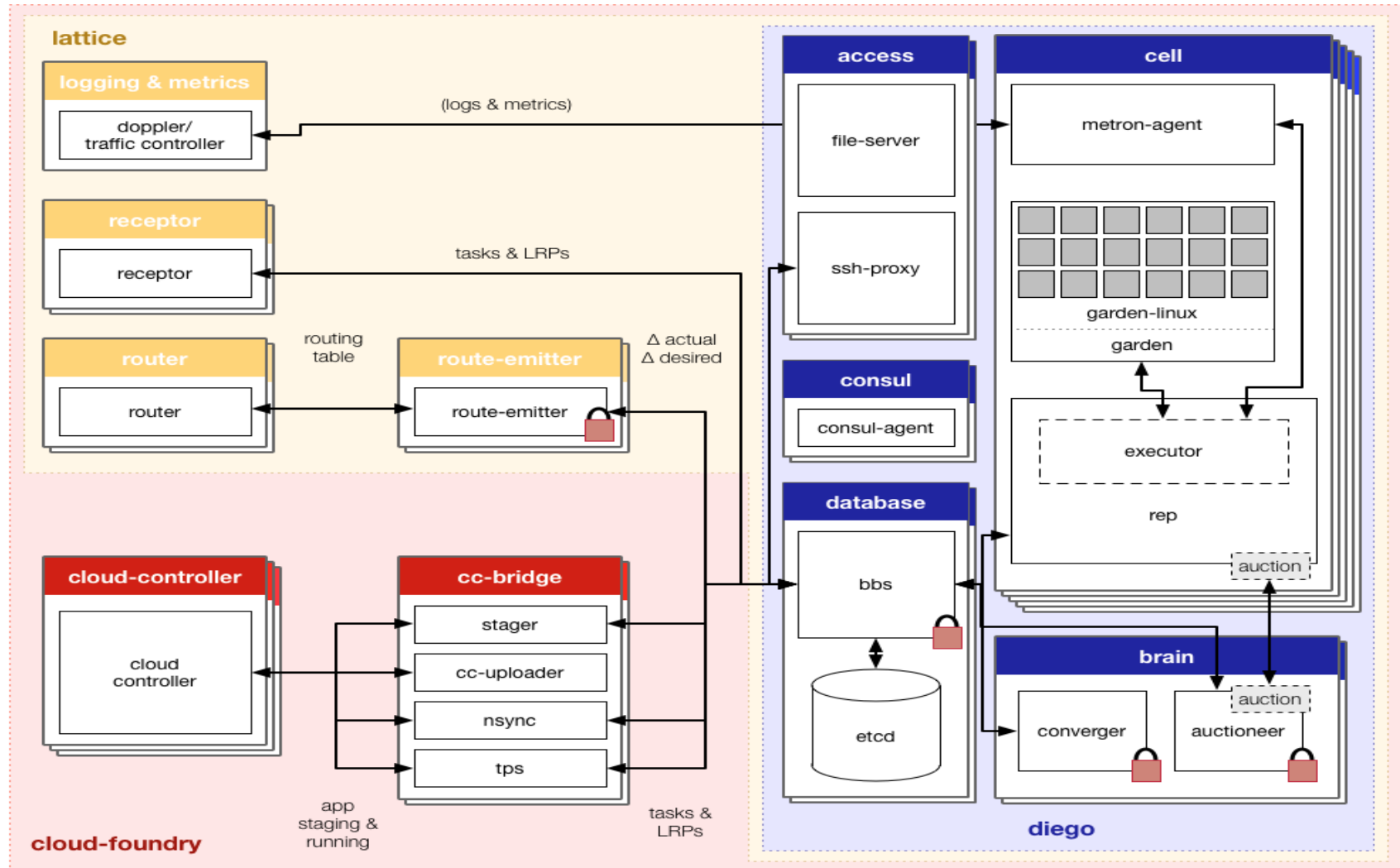
Description

- Secure and fully isolated container
 - Typically a Linux VM
 - But *could* be a Windows VM
 - Replace DEAs
- Periodically broadcast messages about their state
- Typically *many* Cells in a Cloud Foundry installation
- Runs an *Executor* internally to manage containers

Responsibilities

- Participate in app auctions
- Manage apps lifecycle (Executor)
 - Building, starting and stopping Apps as instructed
- Manage app containers
 - Pivotal's secure containers
 - Or any other container with same interface
 - For example to run Docker
- Monitor resource pools
 - Process, File System, Network, Memory

CF and Diego – Full Internal Architecture



Other Internal Components

- Rep
 - Represents Cell
 - Mediates BBS comms
 - Bids on tasks
 - Schedules them on the Executor
- Receptor
 - Respond to request for tasks and desired LRPs
 - Fetch information about currently running tasks and LRP instances
- Executor
 - Responsible for executing work (actions) given to it
 - Spins up a Garden container and executes the work encoded in Task/LRP
- Metron Agent
 - Monitors all containers on Cell
 - Provides logging *and* metrics to Loggregator for all apps/tasks running on Cell

Summary

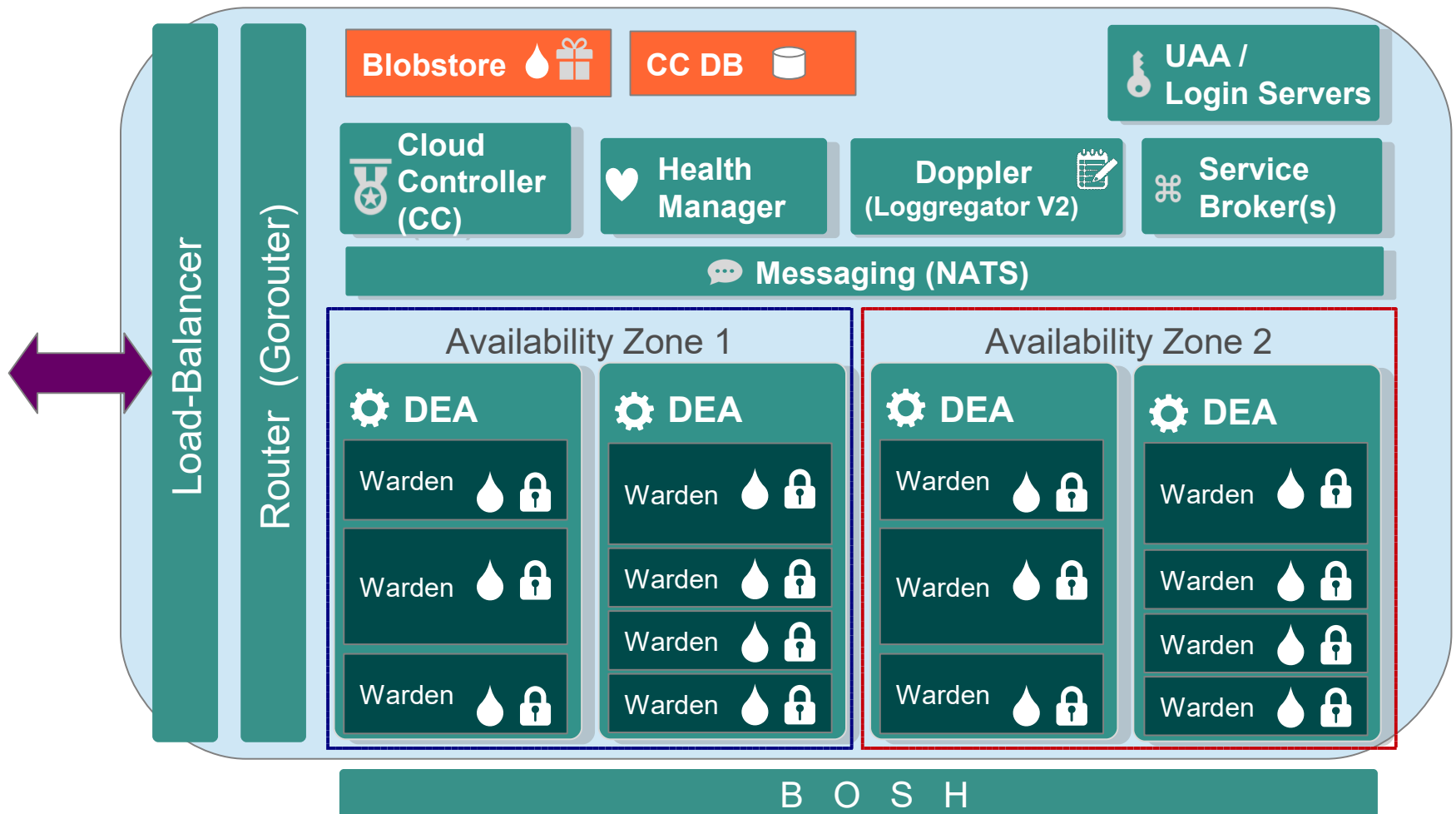
- After completing this lesson, you should have learned about:
 - Cloud Foundry's internal architecture
 - How application instances are managed within various containers
 - The differences between Elastic Runtime architectures
- More information
 - <https://github.com/cloudfoundry-incubator/diego-design-notes>
 - Section appendix: DEA Architecture

Appendix: DEA Architecture

- Previous internal architecture prior to DIEGO rewrite
- This section covers
 - DEAs and Warden containers
 - Where applications run
 - Health Manager
 - Ensures instance recovery
 - *More on this later*

Pivotal CF Components – Original

— HA Zone 1
— HA Zone 2



DEA – Droplet Execution Agent

UAA – User Authorization & Authentication



Pivotal™



Droplet Execution Agents (DEA)

Description

- Secure and fully isolated containers
 - Actually a Linux VM
- Responsible for an Apps lifecycle
 - Building, starting and stopping Apps as instructed
- Periodically broadcast messages about their state
 - Via the NATS message bus.
- Typically many DEAs in a Cloud Foundry installation

Responsible For

- Managing Linux containers
 - Pivotal's Warden containers
- Monitoring resource pools
 - Process
 - File system
 - Network
 - Memory
- Managing app lifecycle
- App log and file streaming
- DEA heartbeats
 - via NATS to Cloud Controller & Health Manager

Warden Container

Description

- Isolated Process
 - Safe, lightweight alternative to full VM
 - Runs a Droplet
- Pivotal's secure implementation of LXC (Secure Linux Container)
- Isolates applications from each other
- Allows multiple applications running on each VM

Responsible For

- Isolates applications running on the same VM
 - Individual failures does not affect other applications on the VM
 - Uses kernel namespaces to isolate network, disk, memory and CPU
 - Uses Linux *cgroups* to do resource management
- Secures applications from environment
- Runs Droplets



Health Manager

Description

- Monitors application uptime
- Listens to NATS message bus for mismatched application states (expected vs. actual)
 - Cloud Controller publishes expected state
 - DEAs publish actual state
- State mismatches are reported to the Cloud Controller.
- Multiple Health Managers possible

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 Cloud Controller can do that!



Messaging (NATS)

Description

- Fast internal messaging bus
- Manages system-wide communication
- Uses a publish-and-subscribe mechanism
- *Not-Another Transport System*
- Single NATS per Cloud Foundry installation

Responsible For

- Non-Persistent messaging
- Pub/Sub
- Queues (app events)
- Directed messages (INBOX)
- **Note:** Diego uses direct communication via HTTP (REST) instead