**Diego Architecture**

1. [Diego Architecture](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#architecture)
2. [Diego Core Components](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#core)
   * [Diego Brain](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#brain-components)
   * [Diego Cell Components](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#cell-components)
   * [Database VMs](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#database-vms)
   * [Access VMs](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#access-vms)
   * [Consul](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#consul)
3. [Cloud Controller Bridge Components](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bridge-components)
   * [Stager](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#stager)
   * [CC-Uploader](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#cc-uploader)
   * [Nsync](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#nsync)
   * [TPS](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#tps)
4. [Platform-specific Components](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#platform-specific)
   * [Garden Backends](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#backends)
   * [App Lifecycle Binaries](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#app-lifecycles)
5. [Other Components](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#other-components)
   * [Route-Emitter](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#route-emitter)

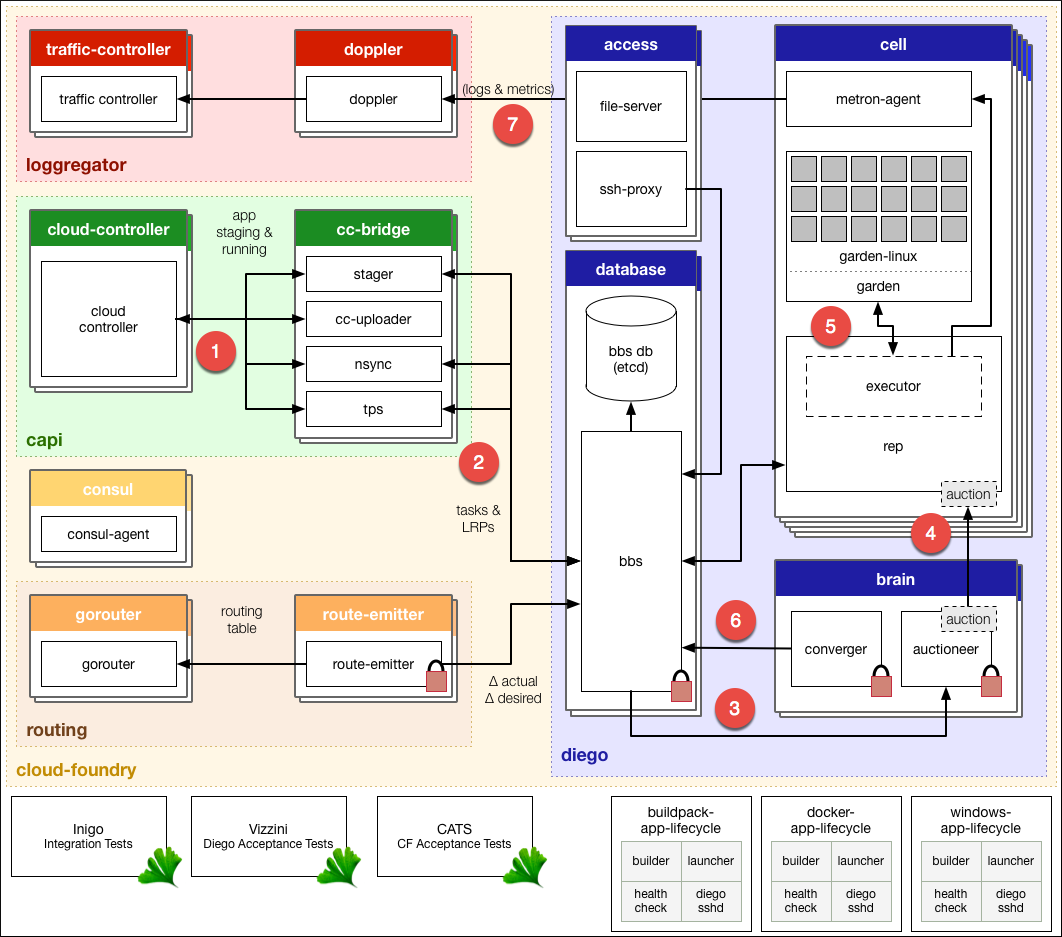
**Page last updated: January 20, 2017**

This topic provides an overview of the structure and components of Diego, the container management system for Pivotal Cloud Foundry versions 1.6 and newer.

**Diego Architecture**

Cloud Foundry has used two architectures for managing application containers: [Droplet Execution Agents](http://docs.pivotal.io/pivotalcf/1-7/concepts/architecture/execution-agent.html) (DEA) and Diego. With the DEA architecture, the [Cloud Controller](http://docs.pivotal.io/pivotalcf/1-7/concepts/architecture/cloud-controller.html) schedules and manages applications on the DEA nodes. In the newer Diego architecture, Diego components replace the DEAs and the [Health Manager (HM9000)](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/dea-vs-diego.html#hm9k), and assume application scheduling and management responsibility from the Cloud Controller.

Refer to the following diagram and descriptions for information about the way Diego handles application requests.



View a larger version of this image at the [Diego Design Notes repo](http://htmlpreview.github.io/?https://raw.githubusercontent.com/cloudfoundry-incubator/diego-design-notes/master/clickable-diego-overview/clickable-diego-overview.html).

1. The Cloud Controller passes requests to stage and run applications to the [Cloud Controller Bridge](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bridge-components) (CC-Bridge).
2. The CC-Bridge translates staging and running requests into [Tasks and Long Running Processes](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-auction.html#processes)(LRPs), then submits these to the [Bulletin Board System](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bbs) (BBS) through an API over HTTP.
3. The BBS submits the Tasks and LRPs to the [Auctioneer](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#auctioneer), part of the [Diego Brain](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#brain-components).
4. The Auctioneer distributes these Tasks and LRPs to [Cells](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#cell-components) through an [Auction](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-auction.html).
5. Once the Auctioneer assigns a Task or LRP to a Cell, an in-process [Executor](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#executor) creates a [Garden](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html" \l "garden)container in the Cell. The Task or LRP runs in the container.
6. The [BBS](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bbs) tracks desired LRPs, running LRP instances, and in-flight Tasks. It also periodically analyzes this information and corrects discrepancies to ensure consistency between ActualLRP and DesiredLRP counts.
7. The [Metron Agent](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html" \l "metron-agent), part of the Cell, forwards application logs, errors, and metrics to the Cloud Foundry Loggregator. For more information, see the [Application Logging in Cloud Foundry](http://docs.pivotal.io/pivotalcf/1-7/devguide/deploy-apps/streaming-logs.html)topic.

**Diego Core Components**

Components in the Diego core run and monitor Tasks and LRPs. The core consists of the following major areas:

* [Brain](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#brain-components)
* [Cells](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#cell-components)
* [Database VMs](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#database-vms)
* [Access VMs](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#access-vms)
* [Consul](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#consul)

**Diego Brain**

Diego Brain components distribute Tasks and LRPs to Diego Cells, and correct discrepancies between ActualLRP and DesiredLRP counts to ensure fault-tolerance and long-term consistency. The Diego Brain consists of the Auctioneer.

**Auctioneer**

* Uses the [auction package](https://github.com/cloudfoundry-incubator/auction) to run Diego Auctions for Tasks and LRPs
* Communicates with Cell [Reps](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#rep) over HTTP
* Maintains a lock in the BBS that restricts auctions to one Auctioneer at a time

Refer to the [Auctioneer repo](https://github.com/cloudfoundry-incubator/auctioneer) on GitHub for more information.

**Diego Cell Components**

Diego Cell components manage and maintain Tasks and LRPs.

**Rep**

* Represents a Cell in Diego Auctions for Tasks and LRPs
* Mediates all communication between the Cell and the BBS
* Ensures synchronization between the set of Tasks and LRPs in the BBS with the containers present on the Cell
* Maintains the presence of the Cell in the BBS
* Runs Tasks and LRPs by asking the in-process Executor to create a container and RunAction recipes

Refer to the [Rep repo](https://github.com/cloudfoundry-incubator/rep) on GitHub for more information.

**Executor**

* Runs as a logical process inside the Rep
* Implements the generic Executor actions detailed in the [API documentation](https://github.com/cloudfoundry-incubator/receptor/blob/master/doc/actions.md)
* Streams STDOUT and STDERR to the Metron agent running on the Cell

Refer to the [Executor repo](https://github.com/cloudfoundry-incubator/executor) on GitHub for more information.

**Garden**

* Provides a platform-independent server and clients to manage Garden containers
* Defines the [garden-linux](https://github.com/cloudfoundry-incubator/garden-linux) interface for container implementation

Refer to the [Garden repo](https://github.com/cloudfoundry-incubator/garden) on GitHub for more information.

**Metron Agent**

Forwards application logs, errors, and application and Diego metrics to the [Loggregator](https://github.com/cloudfoundry/loggregator)Doppler component

Refer to the [Metron repo](https://github.com/cloudfoundry/loggregator/tree/develop/src/metron) on GitHub for more information.

**Database VMs**

**Diego Bulletin Board System**

* Maintains a real-time representation of the state of the Diego cluster, including all desired LRPs, running LRP instances, and in-flight Tasks
* Provides an RPC-style API over HTTP to [Diego Core](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#core) components and external clients, including the [SSH Proxy](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#ssh-proxy), [CC-Bridge](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bridge-components), and [Route Emitter](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#route-emitter).
* Ensure consistency and fault tolerance for Tasks and LRPs by comparing desired state, stored in the database, with actual state, from running instances
* Acts to keep DesiredLRP count and ActualLRP count synchronized in the following ways:
  + If the DesiredLRP count exceeds the ActualLRP count, requests a start auction from the Auctioneer
  + If the ActualLRP count exceeds the DesiredLRP count, sends a stop message to the Rep on the Cell hosting an instance
* Limits the number of containers that can be concurrently in the starting state to 200. If your deployment requires more than 200 containers, containers after the first 200 will not be started until some in the first 200 leave the starting state.
* Monitors for potentially missed messages, resending them if necessary

For more information, see the [Bulletin Board System repo](https://github.com/cloudfoundry-incubator/bbs) on GitHub.

**etcd**

* Provides a consistent key-value data store to Diego

**Access VMs**

**File Server**

* This “blobstore” serves static assets that can include general-purpose [App Lifecycle binaries](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#app-lifecycles)and application-specific droplets and build artifacts.

Refer to the [File Server repo](https://github.com/cloudfoundry-incubator/file-server) on GitHub for more information.

**SSH Proxy**

* Brokers connections between SSH clients and SSH servers running inside instance containers

Refer to [Understanding Application SSH](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/ssh-conceptual.html), [Application SSH Overview](http://docs.pivotal.io/pivotalcf/1-7/devguide/deploy-apps/app-ssh-overview.html), or the [Diego SSH Github repo](https://github.com/cloudfoundry-incubator/diego-ssh) for more information.

**Consul**

* Provides dynamic service registration and load balancing through DNS resolution
* Provides a consistent key-value store for maintenance of distributed locks and component presence

Refer to the [Consul repo](https://github.com/hashicorp/consul) on GitHub for more information.

**Consuladapter**

Consuladapter provides a driver for interfacing with etcd.

Refer to the [Consuladapter repo](https://github.com/cloudfoundry-incubator/consuladapter) on GitHub for more information.

**Cloud Controller Bridge Components**

The Cloud Controller Bridge (CC-Bridge) components translate app-specific requests from the Cloud Controller to the BBS. These components include the following:

**Stager**

* Translates staging requests from the Cloud Controller into generic Tasks and LRPs
* Sends a response to the Cloud Controller when a Task completes

Refer to the [Stager repo](https://github.com/cloudfoundry-incubator/stager) on GitHub for more information.

**CC-Uploader**

* Mediates uploads from the Executor to the Cloud Controller
* Translates simple HTTP POST requests from the Executor into complex multipart-form uploads for the Cloud Controller

Refer to the [CC-Uploader repo](https://github.com/cloudfoundry-incubator/cc-uploader) on GitHub for more information.

**Nsync**

* Listens for app requests to update the DesiredLRPs count and updates DesiredLRPs through the BBS
* Periodically polls the Cloud Controller for each app to ensure that Diego maintains accurate DesiredLRPs counts

Refer to the [Nsync repo](https://github.com/cloudfoundry-incubator/nsync) on GitHub for more information.

**TPS**

* Provides the Cloud Controller with information about currently running LRPs to respond to cf apps and cf app APP\_NAME requests
* Monitors ActualLRP activity for crashes and reports them the Cloud Controller

Refer to the [TPS repo](https://github.com/cloudfoundry-incubator/tps) on GitHub for more information.

**Platform-specific Components**

**Garden Backends**

Garden contains a set of interfaces that each platform-specific backend must implement. These interfaces contain methods to perform the following actions:

* Create and delete containers
* Apply resource limits to containers
* Open and attach network ports to containers
* Copy files into and out of containers
* Run processes within containers
* Stream STDOUT and STDERR data out of containers
* Annotate containers with arbitrary metadata
* Snapshot containers for redeploys without downtime

Refer to the [Garden repo](https://github.com/cloudfoundry-incubator/garden) on GitHub for more information.

**Current Implementations**

* [Garden-Linux](https://github.com/cloudfoundry-incubator/garden-linux) provides a Linux-specific implementation of a Garden interface.

**App Lifecycle Binaries**

The following three platform-specific binaries deploy applications and govern their lifecycle:

* The **Builder**, which stages a CF application. The [CC-Bridge](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bridge-components) runs the Builder as a Task on every staging request. The Builder performs static analysis on the application code and does any necessary pre-processing before the application is first run.
* The **Launcher**, which runs a CF application. The CC-Bridge sets the Launcher as the Action on the DesiredLRP for the application. The Launcher executes the start command with the correct system context, including working directory and environment variables.
* The **Healthcheck**, which performs a status check on running CF application from inside the container. The [CC-Bridge](http://docs.pivotal.io/pivotalcf/1-7/concepts/diego/diego-architecture.html#bridge-components) sets the Healthcheck as the Monitor action on the DesiredLRP for the application.

**Current Implementations**

* [Buildpack App Lifecycle](https://github.com/cloudfoundry-incubator/buildpack-app-lifecycle) implements the Cloud Foundry buildpack-based deployment strategy.
* [Docker App Lifecycle](https://github.com/cloudfoundry-incubator/docker-app-lifecycle) implements a Docker deployment strategy.

**Other Components**

**Route-Emitter**

* Monitors DesiredLRP and ActualLRP states, emitting route registration and unregistration messages to the Cloud Foundry [router](https://github.com/cloudfoundry/gorouter) when it detects changes
* Periodically emits the entire routing table to the Cloud Foundry router

Refer to the [Route-Emitter repo](https://github.com/cloudfoundry-incubator/route-emitter) on GitHub for more information.