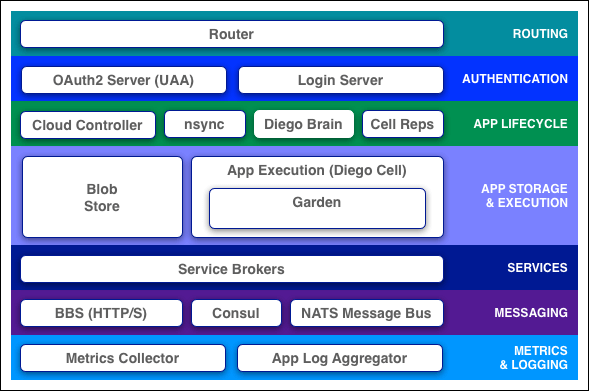
Cloud Foundry Components

**Page last updated: October 26, 2019**

**Warning:** Pivotal Cloud Foundry (PCF) v2.4 is no longer supported because it has reached the End of General Support (EOGS) phase as defined by the [Support Lifecycle Policy](https://pivotal.io/support/lifecycle_policy). To stay up to date with the latest software and security updates, upgrade to a supported version.

Cloud Foundry components include a self-service application execution engine, an automation engine for application deployment and lifecycle management, and a scriptable command line interface (CLI), as well as integration with development tools to ease deployment processes. Cloud Foundry has an open architecture that includes a buildpack mechanism for adding frameworks, an application services interface, and a cloud provider interface.

See the descriptions below for more information about Cloud Foundry components. Some descriptions include links to more detailed documentation.



Routing

Router

The [router](https://docs.pivotal.io/pivotalcf/2-4/concepts/architecture/router.html) routes incoming traffic to the appropriate component, either a Cloud Controller component or a hosted application running on a Diego Cell.

The router periodically queries the Diego Bulletin Board System (BBS) to determine which cells and containers each app currently runs on. Using this information, the router recomputes new routing tables based on the IP addresses of each cell virtual machine (VM) and the host-side port numbers for the cell’s containers.

Authentication

OAuth2 Server (UAA) and Login Server

The OAuth2 server (the [UAA](https://docs.pivotal.io/pivotalcf/2-4/concepts/architecture/uaa.html)) and Login Server work together to provide identity management.

App Lifecycle

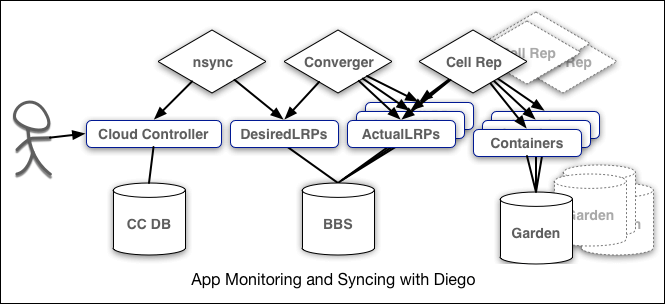
Cloud Controller and Diego Brain

The [Cloud Controller](https://docs.pivotal.io/pivotalcf/2-4/concepts/architecture/cloud-controller.html) (CC) directs the deployment of apps. To push an app to Cloud Foundry, you target the Cloud Controller. The Cloud Controller then directs the Diego Brain through the CC-Bridge components to coordinate individual [Diego cells](https://docs.pivotal.io/pivotalcf/2-4/concepts/architecture/index.html#diego-cell) to stage and run apps.

The Cloud Controller also maintain records of [orgs, spaces, user roles](https://docs.pivotal.io/pivotalcf/2-4/concepts/roles.html), services, and more.

nsync, BBS, and Cell Reps

To keep apps available, cloud deployments must constantly monitor their states and reconcile them with their expected states, starting and stopping processes as required.



The nsync, BBS, and Cell Rep components work together along a chain to keep apps running. At one end is the user. At the other end are the instances of apps running on widely-distributed VMs, which may crash or become unavailable.

Here is how the components work together:

* **nsync** receives a message from the Cloud Controller when the user scales an app. It writes the number of instances into a DesiredLRP structure in the Diego BBS database.
* **BBS** uses its convergence process to monitor the DesiredLRP and ActualLRP values. It launches or kills app instances as appropriate to ensure the ActualLRP count matches the DesiredLRP count.
* **Cell Rep** monitors the containers and provides the ActualLRP value.

App Storage and Execution

Blobstore

The blobstore is a repository for large binary files, which GitHub cannot easily manage because GitHub is designed for code. The blobstore contains the following:

* Application code packages
* Buildpacks
* Droplets

You can configure the blobstore as either an internal server or an external S3 or S3-compatible endpoint. For more information about the blobstore, see [How to use Elastic Runtime BLOB Storage Data](https://community.pivotal.io/s/article/How-to-use-Elastic-Runtime-BLOB-Storage-Data) in the Pivotal Support Knowledge Base.

Diego Cell

App instances, app tasks, and staging tasks all run as [Garden](https://docs.pivotal.io/pivotalcf/2-4/concepts/architecture/garden.html) containers on the Diego Cell VMs. The Diego cell rep component manages the lifecycle of those containers and the processes running in them, reports their status to the Diego BBS, and emits their logs and metrics to [Loggregator](https://docs.pivotal.io/pivotalcf/2-4/concepts/architecture/index.html#metrics-logging).

Services

Service Brokers

Apps typically depend on [services](https://docs.pivotal.io/pivotalcf/services/) such as databases or third-party SaaS providers. When a developer provisions and binds a service to an app, the service broker for that service is responsible for providing the service instance.

Messaging

Internal HTTPS and BBS

Cloud Foundry component VMs communicate with each other internally through HTTP and HTTPS protocols, sharing temporary messages and data stored in Diego’s [Bulletin Board System (BBS)](https://docs.pivotal.io/pivotalcf/2-4/concepts/diego/diego-architecture.html#bbs).

* BOSH Director colocates a [BOSH DNS](https://bosh.io/docs/dns/) server on every deployed VM. All VMs keep up-to-date DNS records for all the other VMs in the same foundation, enabling service discovery between VMs. BOSH DNS also provides client-side load-balancing by randomly selecting a healthy VM when multiple VMs are available.
* Diego’s [Bulletin Board System](https://docs.pivotal.io/pivotalcf/2-4/concepts/diego/diego-architecture.html#bbs) (BBS) stores more frequently updated and disposable data such as cell and app status, unallocated work, and heartbeat messages, as well as longer-lived distributed locks. The BBS stores data in MySQL, using the [Go MySQL Driver](https://github.com/go-sql-driver/mysql).

The route-emitter component uses the NATS protocol to broadcast the latest routing tables to the routers.

Metrics and Logging

Loggregator

The Loggregator (log aggregator) system streams application logs to developers.

# Diego Components and Architecture

**Page last updated: June 20, 2019**

**Warning:** Pivotal Cloud Foundry (PCF) v2.4 is no longer supported because it has reached the End of General Support (EOGS) phase as defined by the [Support Lifecycle Policy](https://pivotal.io/support/lifecycle_policy). To stay up to date with the latest software and security updates, upgrade to a supported version.

This topic describes the components that form and interact with the Diego system.

## Overview

Cloud Foundry uses the Diego system to manage app containers. Diego components assume app scheduling and management responsibility from the Cloud Controller.

Diego is a self-healing container management system that attempts to keep the correct number of instances running in Diego cells to avoid network failures and crashes. Diego schedules and runs Tasks and Long-Running Processes (LRP). For more about Tasks and LRPs, see [How the Diego Auction Allocates Jobs](https://docs.pivotal.io/pivotalcf/2-4/concepts/diego/diego-auction.html).

You can submit, update, and retrieve the desired number of Tasks and LRPs using the [Bulletin Board System (BBS) API](https://github.com/cloudfoundry/bbs).

## How Diego Runs an App

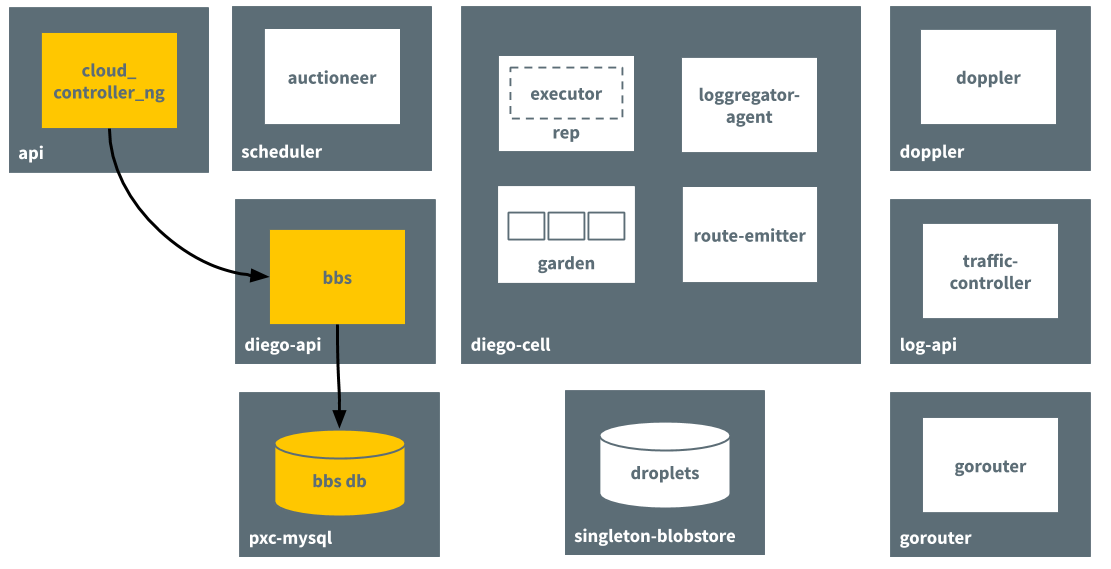
The following sections describe how Diego handles a request to run an app. This is only one of the processes that happen in Diego. For example, running an app assumes the app has already been staged. For more information about the staging process, see [How Applications are Staged](https://docs.pivotal.io/pivotalcf/2-4/concepts/how-applications-are-staged.html).

The diagrams below do not include all of the components Diego. For information about each Diego component, see [Diego Components](https://docs.pivotal.io/pivotalcf/2-4/concepts/diego/diego-architecture.html#components).

**Note**: The images below are based on the VM names in an open-source deployment of Cloud Foundry Application Runtime. In Pivotal Application Service (PAS), the processes interact in the same way, but are on different VMs. Correct VM names for each process are in the components sections of this topic.

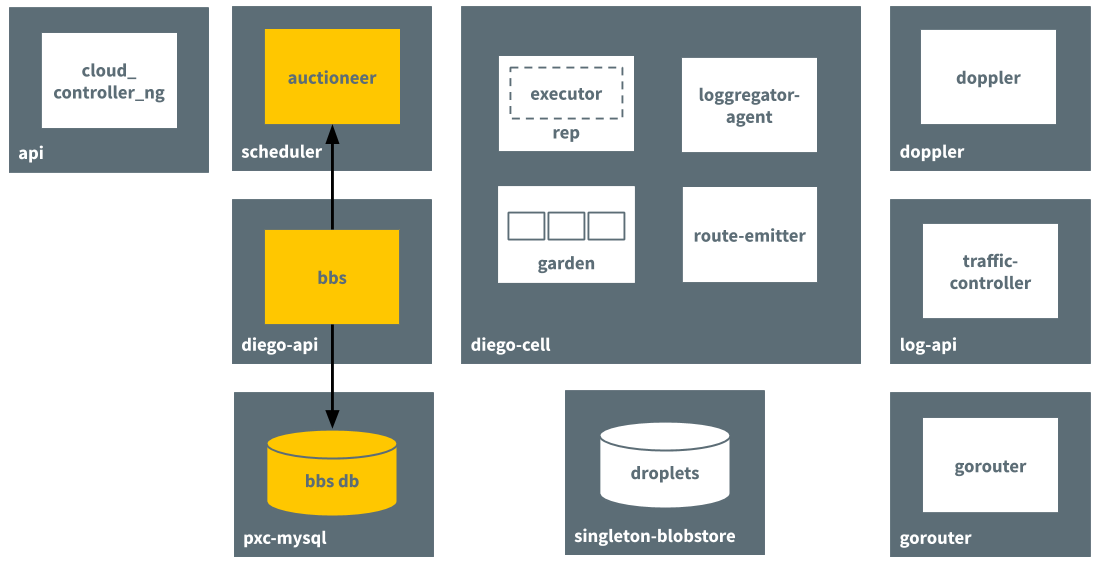
### Receives the Request to Run an App

The Cloud Controller passes requests to run apps to the Diego BBS, which stores information about the request in its database.



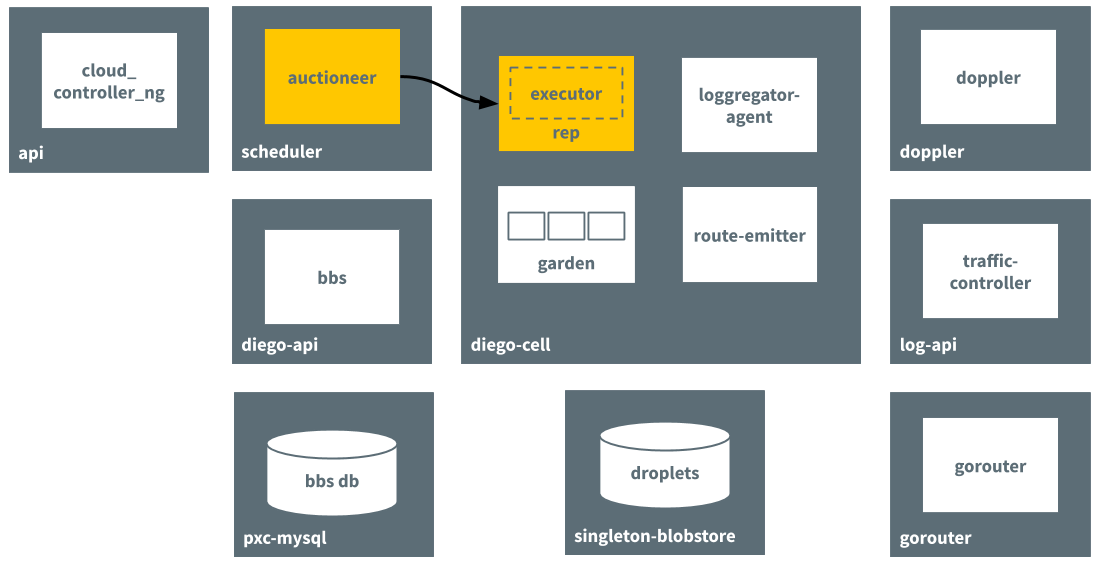
### Passes Request to the Auctioneer Process

The BBS contacts the Auctioneer to create an auction based on the desired resources for the app. It references the information stored in its database.



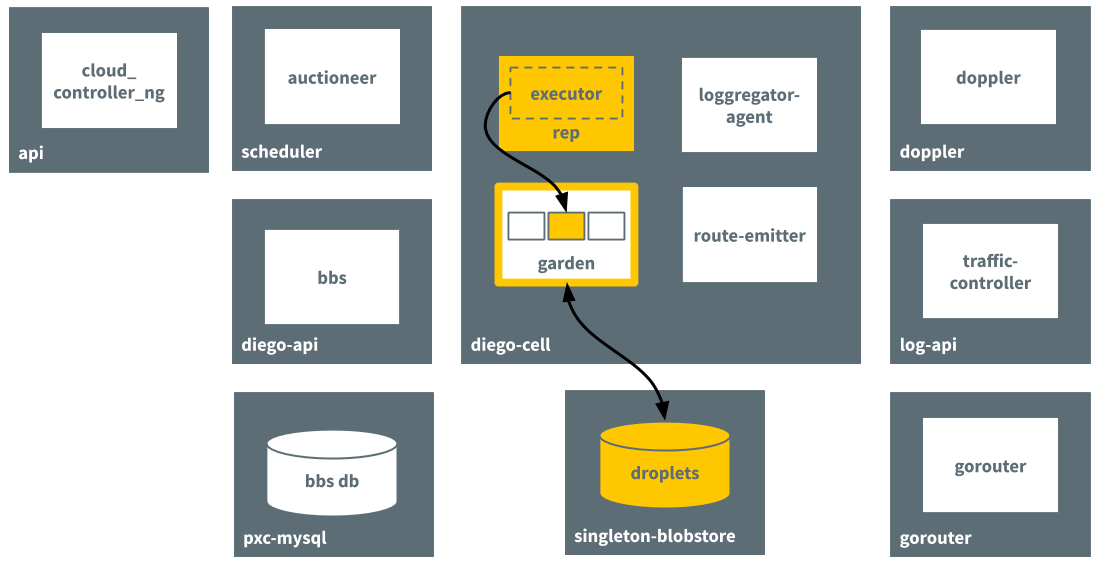
### Performs Auction

Through an auction, the Auctioneer finds a Diego cell to run the app on. The Rep job on the Diego cell accepts the auction request.



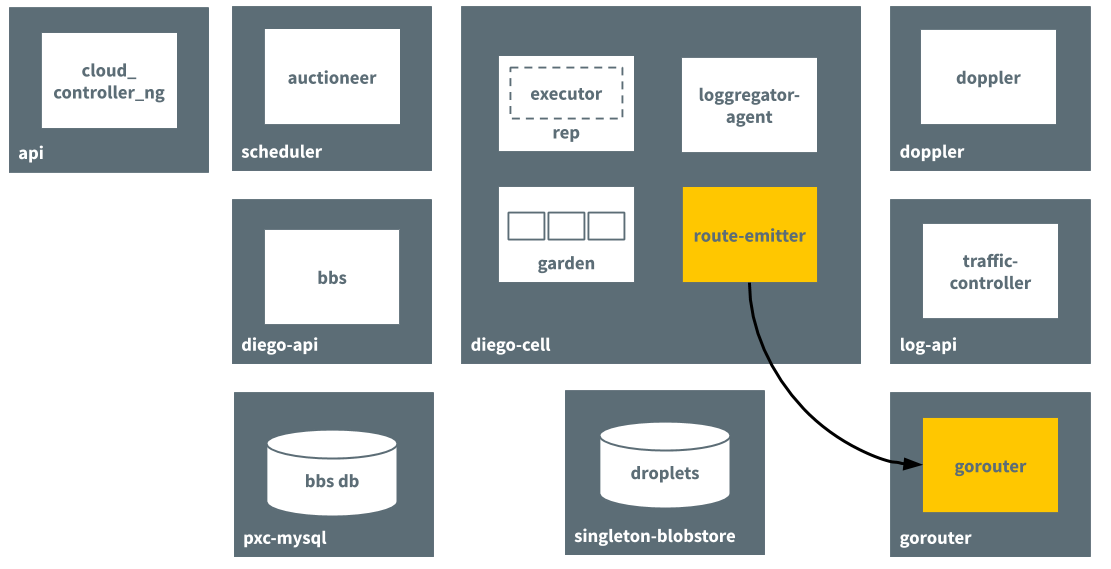
### Creates Container and Runs App

The in-process Executor creates a Garden container in the Diego cell. Garden downloads the droplet that resulted from the staging process and runs the app in the container.



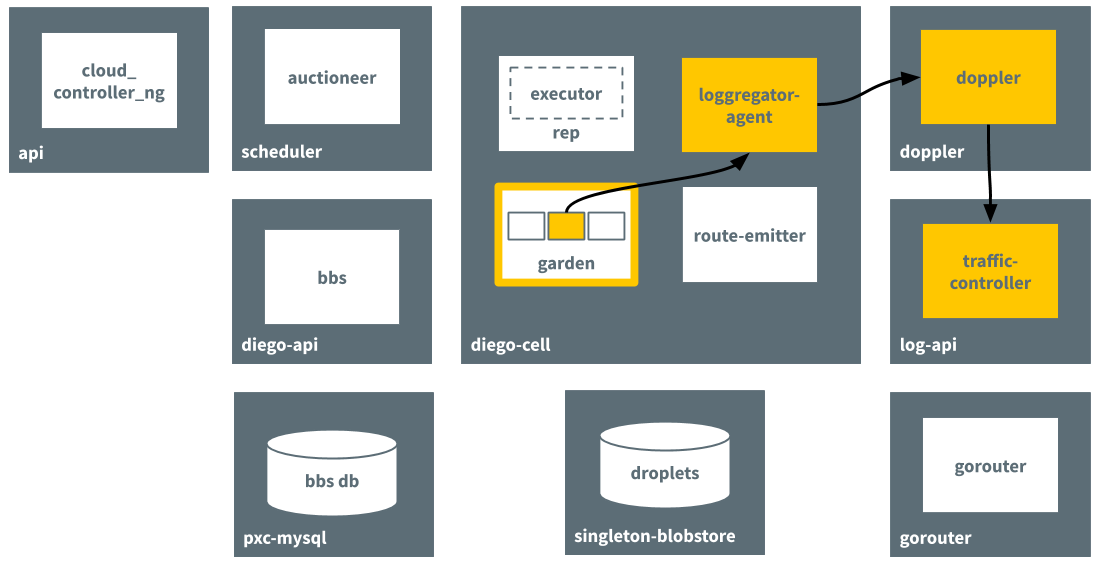
### Emits Route for App

The route-emitter process emits a route registration message to Gorouter for the new app running on the the Diego cell.



### Sends Logs to Loggregator

The Loggregator agent forwards app logs, errors, and metrics to the Cloud Foundry Loggregator. For more information, see the [Application Logging in Cloud Foundry](https://docs.pivotal.io/pivotalcf/2-4/devguide/deploy-apps/streaming-logs.html) topic.



## Diego Components

The following table describes the jobs that are part of the Cloud Foundry Diego BOSH release.

|  |  |
| --- | --- |
| **Component** | **Function** |
| **Job:** auctioneer **VM:** diego\_brain | * Distributes work through auction to Cell Reps over SSL/TLS. See [How the Diego Auction Allocates Jobs](https://docs.pivotal.io/pivotalcf/2-4/concepts/diego/diego-auction.html). * Maintains a lock in Locket to ensure only one auctioneer handles auctions at a time. |
| **Job:** bbs **VM:** diego\_database | * Maintains a real-time representation of the state of the Diego cluster, including desired LRPs, running LRPs, and in-flight Tasks. * Provides an RPC-style API over HTTP to Diego Core components for external clients as well as internal clients, including the SSH Proxy and Route Emitter. * Ensures consistency and fault tolerance for Tasks and LRPs by comparing desired state with actual state. * Keeps DesiredLRP and ActualLRP counts synchronized. If the DesiredLRPcount 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 |
| **Job:** file\_server **VM:** diego\_brain | * Serves static assets that can include general-purpose App Lifecycle binaries |
| **Job:** locket **VM:** diego\_database | * Provides a consistent key-value store for maintenance of distributed locks and component presence |
| **Job:** rep **VM:** diego\_cell | * Represents a Cell in Diego Auctions for Tasks and LRPs * Runs Tasks and LRPs by creating a container and then running actions in it * Periodically ensures its set of Tasks and ActualLRPs in the BBS is in sync with the containers actually present on the Cell * Manages container allocations against resource constraints on the Cell, such as memory and disk space * Streams stdout and stderr from container processes to the metron-agent running on the Cell, which in turn forwards to the Loggregator system * Periodically collects container metrics and emits them to Loggregator * Mediates all communication between the Cell and the BBS * Maintains a presence record for the Cell in Locket |
| **Job:** route\_emitter **VM:** diego\_cell | * Monitors DesiredLRP and ActualLRP states, emitting route registration and unregistration messages to Gorouter when it detects changes. * Periodically emits the entire routing table to the Cloud Foundry Gorouter. |
| **Job:** ssh\_proxy **VM:** diego\_brain | * Brokers connections between SSH clients and SSH servers * Runs inside instance containers and authorizes access to app instances based on Cloud Controller roles |

### Additional Information

The following resources provide more information about Diego components:

* The [Diego Release repository](https://github.com/cloudfoundry/diego-release) on GitHub.
* The [Auctioneer repository](https://github.com/cloudfoundry/auctioneer) on GitHub.
* The [Bulletin Board System repository](https://github.com/cloudfoundry/bbs) on GitHub.
* The [File Server repository](https://github.com/cloudfoundry/file-server) on GitHub.
* The [Rep repository](https://github.com/cloudfoundry/rep) on GitHub.
* The [Executor repository](https://github.com/cloudfoundry/executor) on GitHub.
* The [Route-Emitter repository](https://github.com/cloudfoundry/route-emitter) on GitHub.
* [Application SSH](https://docs.pivotal.io/pivotalcf/2-4/concepts/diego/ssh-conceptual.html), [Application SSH Overview](https://docs.pivotal.io/pivotalcf/2-4/devguide/deploy-apps/app-ssh-overview.html), and the [Diego SSH repository](https://github.com/cloudfoundry-incubator/diego-ssh) on GitHub.

## Components from Other Releases

The following table describes jobs that interact closely with Diego but are not part of the Diego Cloud Foundry BOSH release.

|  |  |
| --- | --- |
| **Component** | **Function** |
| **Job:** bosh-dns-aliases **VM:** all | * Provides service discovery through colocated DNS servers on all BOSH-deployed VMs * Provides client-side load-balancing by randomly selecting a healthy VM when multiple VMs are available |
| **Job:** cc\_uploader **VM:** diego\_brain | * 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 |
| **Job:** database **VM:** mysql | * Provides a consistent key-value data store to Diego |
| **Job:** loggregator-agent **VM:** all | * Forwards app logs, errors, and app and Diego metrics to the Loggregator Doppler component |
| **Job:** cloud\_controller\_clock **VM:** clock\_global | * Runs a Diego sync process to ensure desired app data in Diego is in sync with the Cloud Controller. |

### App Lifecycle Binaries

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

* The **Builder**, which stages a CF app. The Builder runs as a Task on every staging request. It performs static analysis on the app code and does any necessary pre-processing before the app is first run.
* The **Launcher**, which runs a CF app. The Launcher is set as the Action on the DesiredLRP for the app. It 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 app from inside the container. The Healthcheck is set as the Monitor action on the DesiredLRP for the app.

#### **Current Implementations**

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

### Additional Information