Pivotal Platform (formerly known as Pivotal Cloud Foundry) is now part of [VMware Tanzu](https://cloud.vmware.com/tanzu) following VMware’s acquisition of Pivotal in late 2019.

[Pivotal Platform](https://pivotal.io/platform) is a multi-cloud platform for the deployment, management, and continuous delivery of applications, containers, and functions.

There are two [runtime environments](https://docs.pivotal.io/platform/customizing/runtimes.html) available for Pivotal Platform

1. Pivotal Application Service (PAS) - a distribution of the [open-source Cloud Foundry Application Runtime](https://www.cloudfoundry.org/application-runtime/)
2. [Enterprise Pivotal Container Service (PKS)](https://docs.pivotal.io/pks/1-6/index.html) - a Kubernetes-based container service offered by Pivotal, now part of VMware. It aims to simplify the deployment and management of Kubernetes clusters in enterprise environments.

Pivotal Platform is aimed at enterprise users and offers additional features and services from Pivotal and from other third parties for installing and operating **Cloud Foundry** as well as to expand its capabilities and make it easier to use.

Major cloud platforms such as **Amazon Web Services** and **Google Cloud** also provide templates and quickstarts that automate large portions of the **Pivotal Platform deployment process**.

Pivotal Platform abstracts away the process of setting up and managing an application runtime environment so that developers can focus solely on their applications and associated data. Running a single command cf push will create a scalable environment for your application in seconds, which might otherwise take hours to spin up manually.

Pivotal Platform allows developers to deploy and deliver software quickly, without needing to manage the underlying infrastructure.

In this post, we’ll explore each of the technologies that make up a typical Pivotal Platform cluster using PAS and how they work together.

[Getting Abstract](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#getting-abstract)

Pivotal Platform creates two kinds of virtual machines that handle **different aspects of an elastic runtime environment**:

* **Component VMs** - It creates the underlying infrastructure environment for a deployment.
* **Host VMs** - It provides a generic runtime environment for applications.

**Component VMs** allow Pivotal Platform to be cloud-agnostic by providing a standardized infrastructure environment for staging and running applications on any **supported cloud hosting services** including **Azure, AWS, Google, OpenStack, and vSphere**.

The generic runtime environment provided by **Host VMs** enables **single Pivotal Platform deployment** to host multiple applications regardless of languages or dependencies.

Once an application is pushed via cf push, Pivotal Platform **stages and packages it into a binary droplet** that can be distributed and run on a host VM.

In addition to abstraction, Pivotal Platform provides significant scaling capabilities. Global-scale companies like Boeing, Wells Fargo, and Citi have adopted and used Pivotal Platform to migrate and build applications in an elastic, cloud-based environment.

**Single Pivotal Platform deployment** can run any number of applications if there are sufficient underlying infrastructure resources.

Hundreds of thousands of application instances can be spread across over a thousand host VMs.

To increase **availability and redundancy**, Pivotal Platform provides a single command to scale up the number of application instances and, if your **Pivotal Platform** is deployed across **multiple availability zones**, then it will automatically spread instances across zones.

The same way, **Component VMs** can be scaled up and assigned to multiple availability zones to **ensure high availability**.

## [Monitoring Pivotal Platform](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#monitoring-pivotal-platform)

**Pivotal Platform Operators** - People managing and monitoring a Pivotal Platform deployment.

**Developers** - The people who deploy the applications.

**End Users** - The people who access applications running on the deployment.

For operators, monitoring your application deployment’s performance and capacity indicators is vital to ensuring that applications are running optimally. It’s also necessary to check that the deployment can scale to match demand, whether scaling horizontally in terms of the number of application instances or vertically in terms of the resources available within the VMs.

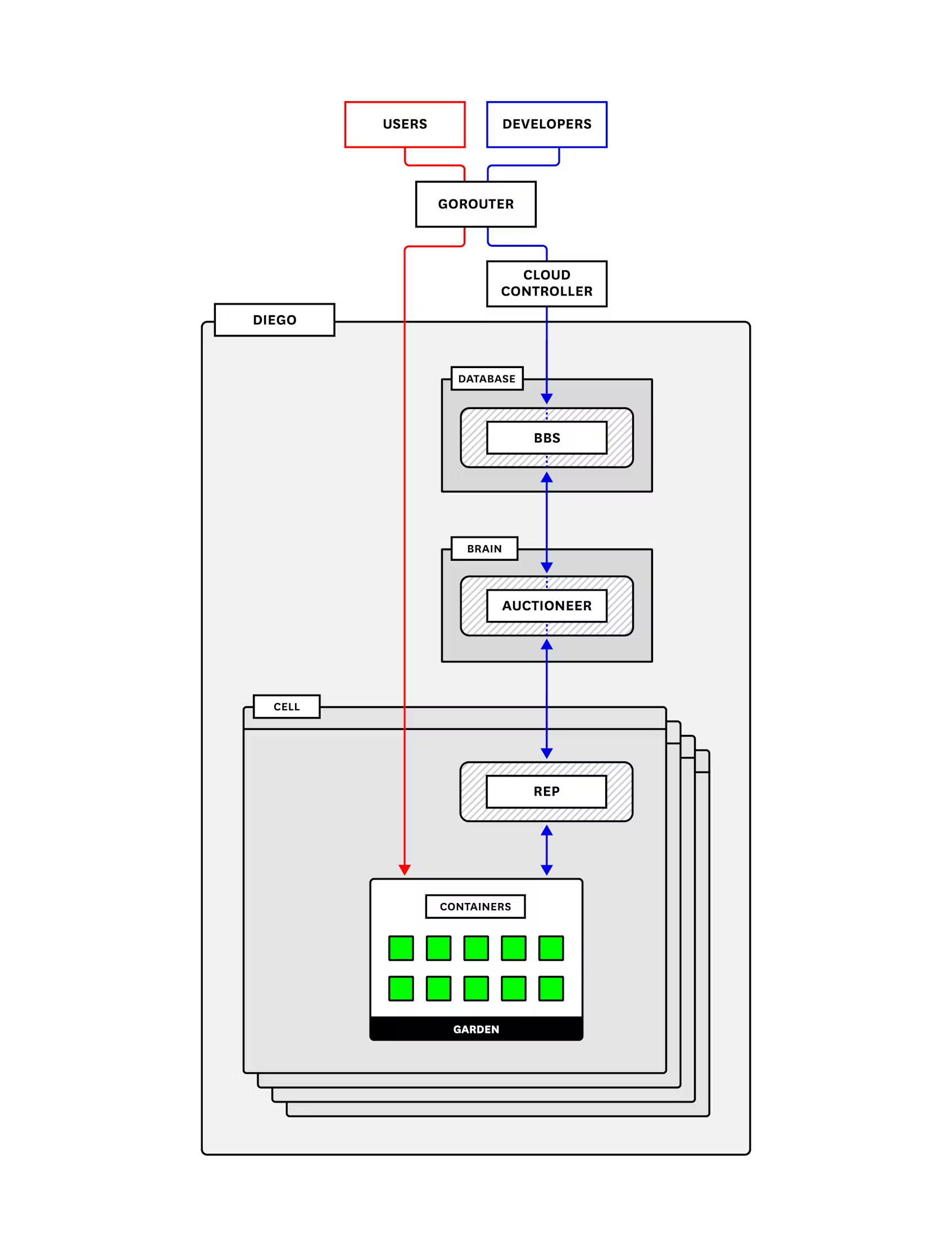
We’ll cover [the key metrics](http://www.datadoghq.com/blog/pivotal-cloud-foundry-metrics) which operators would want to collect and monitor in Pivotal Platform. Additionally, Pivotal Platform offers several ways to monitor application performance.

## [Key components of Pivotal Platform Architecture](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#key-components-of-pivotal-platform-architecture)

Pivotal Platform is a distributed system comprising many components that run, manage, and monitor the health of the deployment and its application runtime environment, which might be hosting dozens or hundreds of applications.

The primary components we will cover in this guide, along with their subsystems, are:

* [BOSH/the Ops Manager](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#bosh-and-the-ops-manager) (deployment automation)
* [The User Account and Authentication server](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#user-account-and-authentication) (identity management)
* [The Gorouter](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#gorouter) (application and system routing)
* [The Cloud Controller](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#cloud-controller) (application staging and running)
* [Diego](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego) (application execution and runtime)
* [Loggregator](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#loggregator) (logs and metric aggregation)

****

### [**BOSH and the Ops Manager**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#bosh-and-the-ops-manager)

#### **BOSH**

[**BOSH**](https://bosh.io/docs/) is a deployment manager that can automatically provision and deploy widely distributed, cloud-based software. Originally developed specifically for Cloud Foundry, BOSH can also be used outside of Cloud Foundry environments, for example to deploy a ZooKeeper or Kubernetes cluster.

Essentially, BOSH allows Pivotal Platform to be deployed in any cloud **by providing an interface to build required infrastructure components on top of a given IaaS platform**.

BOSH handles the deployment of the underlying Pivotal Platform’s infrastructure environment by launching and managing all required component VMs via the **BOSH Director**.

Through common plugins such as the **Health Monitor**, BOSH can track the health of its VMs (Both Component VMs and Health VMs) and also [self-heal](https://bosh.io/docs/resurrector/) if it detects that **a VM has crashed or has otherwise become inaccessible**. It will attempt to recreate the faulty VM automatically to avoid downtime.

The **BOSH Director** reads YAML based deployment config manifest to determine **what VMs, persistent disks, and other resources are required, as well as how many availability zones to use.**

For a given cloud provider, the BOSH Director relies **on an IaaS-specific manifest, or**[**cloud config**](https://bosh.io/docs/cloud-config/)**,** that is applied on top of a baseline deployment config manifest for Pivotal Platform.

**The cloud config maps Pivotal Platform resources to specific resources of a given cloud provider**, for instance mapping Pivotal Platform availability zone z1 to AWS availability zone us-east-1a or Google Cloud availability zone us-central1-f.

The Director launches VMs built on **[stemcells](https://bosh.io/docs/stemcell" \t "_blank)**, which include **a base operating system, a BOSH agent for monitoring, and any required utilities and configuration files.**

[**Releases**](https://bosh.io/releases), which are layered on top of the stemcell, contain **a versioned set of configurations, source codes, binaries, scripts, and anything else that might be required to run a specific software package on the VM.**

#### [**Ops Manager**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#ops-manager)

Pivotal Platform wraps the BOSH API with the **Ops Manager**, which provides a web-based GUI to automate tasks and help administer the deployment.

**The Ops Manager generates manifest files from the options an operator has selected and sends them to the BOSH Director.**

**Within a Pivotal Platform deployment, operators can also use the Ops Manager to install their desired application runtime environments (Linux or Windows) on host VMs.**

**BOSH releases are available as buildpacks, which can be installed to provide runtime support for applications and services.**

### [**User Account and Authentication**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#user-account-and-authentication)

The User Account and Authentication (UAA) server is PAS’s **central identity and access management (IAM) component**. Operators can use the UAA APIs to create and manage user accounts. UAA also acts as **an OAuth2 server** and can generate authentication tokens for client applications based on the access scope they’ve been granted.

### [**Gorouter**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#gorouter)

The **Grouter** is Pivotal Application Service’s router **(written in Go).** It handles and routes the incoming requests. These requests come either from **operators or developers** sending commands - such as **cf push** that are then routed to the [Cloud Controller API](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#cloud-controller), or from **end users** accessing application instances running on the deployment.

The Gorouter communicates with the **Diego BBS** to keep track of which applications exist, how many instances there are, and where they are running to maintain **a routing table** and load balance user traffic appropriately. It provides basic **round-robin load balancing** for sending user traffic to available application instances.

The Gorouter **only supports requests over HTTP/HTTPS**. If you use TLS encryption and want it to terminate as close to the instance as possible (as opposed to terminating at a load balancer) **you may optionally enable and configure TCP routing**. **In this case, a separate TCP router handles routing for TCP traffic.**

### [**Cloud Controller**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#cloud-controller)

The **Cloud Controller (CC)** provides API endpoints for operators and developers to interact with **PAS and the application instances running on a specific domain**. Commands include staging applications, starting, or stopping applications, collecting health information, and querying that all desired applications are running.

The CC is also responsible for communicating with the **User Account and Authentication server** to authenticate the user and ensure that they have the **proper permissions** to perform the requested operation/task.

Developers can stage applications that are of 2 types as follows,

1.**Buildpack based application**

2.**Docker image-based application**

**Buildpack based application.**

* Buildpack-based applications require one or more [buildpacks](https://docs.pivotal.io/pivotalcf/buildpacks/index.html" \t "_blank) to provide dependencies. **For example, the Python Buildpack is needed to stage a Django application.**
* When a developer pushes an application for staging, the CC sends needed files and instructions to **PAS’s container orchestration and management component, called**[**Diego**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego).
* For staging and running applications, the CC requires a **MySQL database and a blobstore**.
* The CC creates a record and stores application metadata in the database including the application name, applicable [orgs, spaces](https://docs.pivotal.io/pivotalcf/console/manage-spaces.html), services, and user roles, the number of instances to spin up; memory and disk quotas; etc. It also will create and bind a route to the new application.
* The CC packages and stores required **binary files** generated at various points in the staging process in the **blobstore**. These binaries consist of five types: **application packages** (source code, resource files, etc.), **buildpacks** (e.g., application dependencies), **a resource cache** (larger files from the application package), **a buildpack cache** (larger files created during application staging), and **droplets** (the fully staged and ready-to-run application).
* The **CC sends instructions to Diego** to distribute and execute the staging tasks, which take everything the application needs and package it into a droplet that can be run on **a Cloud Foundry container**. **When staging task is completed, Diego sends the droplet to the CC to store in its blobstore and notifies the CC that the application is ready.** Finally, after staging, the Cloud Controller signals Diego to start the application and continues to communicate with Diego for updates on the applications status.
* **Once an application is running,** **the CC makes it possible to**[**bind services**](https://docs.pivotal.io/pcf-dev/dev-services.html)**to the application.** Services provision reserves resources for an application on demand. They can provide a wide range of types of resources. A few examples might be a web application account, a set of environment variables, or a dedicated Redis cluster.

[**Docker image-based application**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#docker-image-applications)

* Docker image-based applications don’t require any buildpacks because the image contains all the dependencies required to run the application.
* The CC sends the image to Diego for staging, receives and stores required metadata about the image from Diego, and then instructs Diego to schedule processes to run the application.

### [**Diego**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego)

Diego is the container orchestration system for Pivotal Platform deployments, having replaced the previous DEA (Droplet Execution Agent). Diego handles the creation and management of the containers that stage and run applications. **Operators can use the Ops Manager to choose which runtime backend they want to use - the**[**Guardian**](https://github.com/cloudfoundry/guardian/)**backend for Linux or**[**Garden Windows**](https://github.com/cloudfoundry/garden-windows)**for Windows (or both).**

These backends are managed through the [**Garden API**](https://github.com/cloudfoundry/garden/). When a developer deploys their application with cf push, **Diego uses Garden to create a generic, abstracted, containerized environment for any kind of instance, whether it be a buildpack-based droplet, a Docker image, or a Windows Server container.**

Diego includes a health monitor and is self-healing: it will attempt to restart instances that have crashed to ensure that the number of running instances matches what the deployment configuration requires.

To accomplish its orchestration tasks, Diego relies on three main components:

* the [Diego Brain](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego-brain)
* the [database VM](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#database-vm)
* one or more [Diego cells](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego-cells)

Before discussing these Diego components, it’s important to understand **tasks** and **Long-Running Processes (LRPs)**, as these concepts are fundamental to know how Pivotal Platform runs applications.

#### [**Tasks and LRPs**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#tasks-and-lrps)

Pivotal Application Service translates incoming application-specific requests and processes into **generic, abstracted tasks or Long-Running Processes (LRPs)**. This abstraction lets **PAS to forward all requests and processes to Diego so that it can schedule and assign them to host VMs known as**[**cells**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego-cells)**for execution.**

**Tasks are one-time processes** - that is, tasks are inherently terminating and provide a success/failure response. A task could be a database migration, or an initial service setup (for example, staging tasks for getting an application up and running). **Containers running tasks are destroyed once they are completed.**

**LRPs are one or more application instances or other processes that are generally continuous, scalable, and always available.** PAS attempts to ensure high availability and resilience by maintaining multiple running instances of the same LRP across availability zones.

**Diego and the Cloud Controller work together to ensure that the number of running LRPs (ActualLRPs) matches the number that the system should be running (DesiredLRPs).** That is, they make sure that Diego is always running as many application instances as users are expecting and will automatically terminate or spin up new instances if there are any discrepancies.

**Dieogo Brain**

**The main purpose of the Diego Brain is to schedule and assign incoming requests to the cells for execution.** The primary component responsible for this is the **Auctioneer**. The Auctioneer receives work from the Cloud Controller via the BBS (the Database VM).

Then **Auctioneer** communicates with a cell via the cell’s **Rep** (the point of contact between the cell and the rest of the deployment) to auction off this work after the **Stager** translates it into Tasks and LRPs.

The Auctioneer assigns tasks and LRPs in batches. **This process lets PAS to balance load and maintain high availability as much as possible.**

The Diego Brain’s other components carry out additional functions, including maintaining correct LRP counts, storing and handling resources required for application staging, or providing SSH access to application containers.

#### [**Database VM**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#database-vm)

Diego’s database VM is essentially responsible for monitoring, storing, and updating the state of the deployment and of the work that is assigned to the cells. There are two primary components within the database VM:

* the bulletin board system (BBS)
* Locket

The **BBS** is the intermediary between the CC and the Auctioneer and provides an API to communicate with and send requests to the Diego cells, which in turn create the containers that run the requested work. **As such, it is the gateway through which information about DesiredLRPs and ActualLRPs flows and is vital to maintaining an accurate picture of the Diego cluster.**

The BBS requires its own relational database (MySQL or Postgres) to maintain a record of cell status, unallocated work, and other information. It uses this database to keep an up-to-date image of all the work the Diego cluster is handling and sends that to the Auctioneer when it is assigning a new batch so that work can be distributed appropriately. This also helps avoid duplicating LRPs or Tasks.

The BBS also runs regular convergence assessments that compare the running state on the Diego cells against the desired state provided by the Cloud Controller to ensure they are the same.

**Locket** uses a key-value store and provides an API for service discovery and registering locks. Certain components must register locks for processes to ensure that, for example, there are no conflicts resulting from multiple cells accepting the same work.

#### [**Diego cells**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#diego-cells)

Diego cells are host VMs that run the containers doing the actual work. **Multiple Diego cells can be running in the deployment using different stem cells and releases, or even Garden backends.** The Rep on each cell performs several functions: for starters, it registers and maintains the cell’s presence with the BBS, and it communicates with the Auctioneer to bid for auctioned jobs. When a task or LRP is accepted, the Rep’s internal process, the **Executor**, instructs Garden to create a container to run it. The Rep also forwards information about completed tasks to the BBS.

The Rep constantly monitors the containers running within its cell to make sure they are healthy and compares their number and state to what the BBS expects to confirm that there are no discrepancies.

**The Rep also monitors resource allocation in comparison to the cell’s capacity. Finally, it forwards metrics and logs from the containers to the central Loggregator system**.

Another important component in the Diego cells is the **Route-Emitter**. This registers and records changes to LRP states and emits updated routing tables to the Gorouter to make sure that application instances are accessible.

### [**Loggregator**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#loggregator)

**Loggregator** is a system that aggregates, streams logs and metrics from PAS infrastructure components as well as instrumented applications running on the deployment.

Loggregator collects **platform metrics** from PAS infrastructure components, **system-level resource** metrics from the VMs, and **application logs**.

Loggregator will also collect available metrics from **installed services**. That is, **some Pivotal add-on products (for example, Redis) publish their own metrics, which are then forwarded to Loggregator and included in a data stream called the**[**Firehose**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#firehose)**.**

**A diagram of a diagram

Description automatically generated**

Application logs include any logs that an application writes to stderr or stdout. **They also include log messages emitted by PAS components as they process requests related to staging, running, and interacting with an application.**

Note, however, that Loggregator does not include PAS component system logs, or logs produced by internal processes and written to files on the VMs. Instead, these are streamed through **RSYSLOG** and can be accessed by connecting to a third-party **SYSLOG** drain.

#### [**Firehose**](https://www.datadoghq.com/blog/pivotal-cloud-foundry-architecture/#firehose)

**The Firehose is a WebSocket endpoint that clients can connect to access the full Loggregator stream.**

The **data streaming from the Firehose** can be accessed using **nozzles**, components that can consume information from the Loggregator stream and decode the messages from the protocol buffers.

Access to the Firehose requires a user with special permissions, given the possibly sensitive nature of information included in the logs.

**The Firehose is where operators can get the bulk information available about their PAS deployment and its health.**