Cloud Foundry

Jorge Simão, Ph.D.

Content

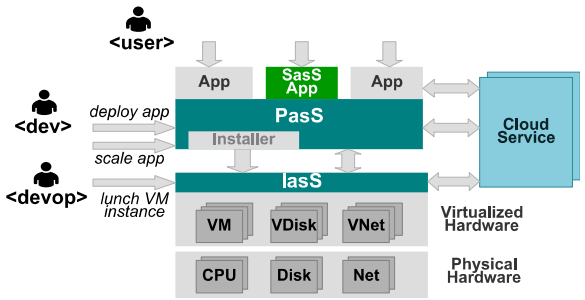
Cloud Computing: IasS vs. PasS

Cloud computing involves the use of virtualized computing resources, including Hardware - CPUs, storage, and networking, Software, and Services – to simplify the processes of application deployment. Motivating factors include: lower-prices affordable by economy of scale, reduced time-to-market and faster development-delivery life-cycle, and/or simply as a way to outsource missing expertise inside an organization according to well defined operational interfaces.

Access to a cloud infrastructure for the purposes of application deployment can be provided at two levels.

- The virtual-infrastructure or *Infrastructure-as-Service* (**IasS**) level - where tools are provided to create, configure and manage the virtual computing resources – such as VM instances or virtual storage devices.
- The could-platform or *Platform-as-Service* (**PasS**) level – where tools are provided to deploy, scale and manage individual applications.

PasS enabling software is deployed on cloud-resources provided by a IasS , with the goal of creating a platform that makes it easy for application developers, system administrators, and operations staff, to work in an integrated way and deploy and scale highly-available applications easily. Internally, a PasS platform uses the APIs and tools provided by underlying IasS to streamline and automate the actions and activities that would otherwise take long time and effort to achieve. The use of services that applications can connect with (e.g. databases, or messaging systems), often blends the distinction between a IasS and a PasS. Diagram below captures this relationship:



About Cloud Foundry

Cloud Foundry is an open-source PasS platform, that can be deployed in a wide variety of clouds. First-class support is given to deploy applications in **AWS**, **OpenStack**, **vSphere**, and others.

To actually deploy **Cloud Foundry** on a cloud environment another software tool-chain must be used. Namely, to create the VMs instances (with selected OS images) where **Cloud Foundry** components and applications will run. Usually the specially-built **BOSH** tool-chain is used for this purpose.

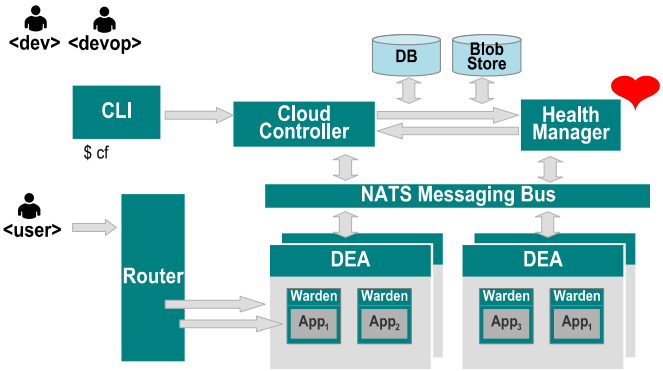
Cloud Foundry is largely agnostic about the languages, run-times, libraries, frameworks, and methodologies used to build the applications deployed to it. First-class support is currently given to deploy applications written in Java, Go, Groovy, Ruby, and PhP, and community provided extensions support other languages (as *buildpacks*). Most components in Cloud Foundry are implemented in the language Go, while BOSH (and some components in Cloud Foundry) are implemented in Ruby - but this is completely transparent and does not have any direct implication for deployed applications.

Cloud Foundry Architecture

A **Cloud Foundry** installation runs on a multi-VM cloud distributed environment. The overall architecture is built out of several components (processes) deployed in different VMs, that interact internally trough a messaging system (**NATS**). A user-interface (**CLI** or **Web-Console**) is used by developers and administrators to deploy and manage applications. Applications artefacts are bundled with its run-time dependencies according to the instructions specified in a language&framework specific script-collection (a *buildpack*), to complete a runnable image - a *droplet*. Droplets are executed in VMs managed by a **Droplet Execution Engine** (**DEA**). Application droplets run inside a virtualization container (**Warden**) that isolates applications running in the same VM, making sure that each one takes only their fare-share of resources (e.g. memory). A **Cloud-Controller**

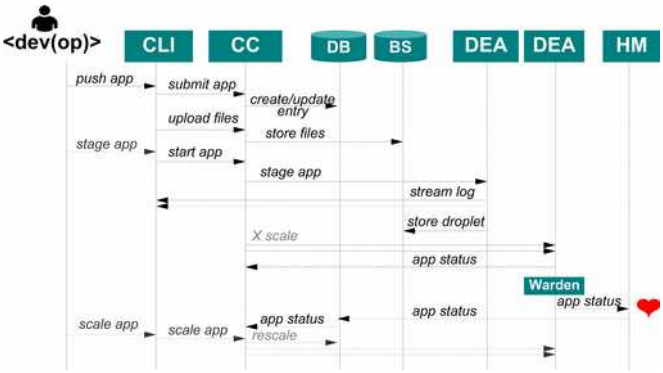
orchestrates the different action required to stage and control applications, and provides a REST API that clients (CLI or Web-Console) can connect with. A database is used to keep all the meta-data about staged applications and running droplets, and a blob-store is used to store binary objects such as applications artefacts and droplets. A **Router** is used to dynamically map application URLs to one of the VMs where application instances are deployed. A **Health Manager** is responsible to collect status information about running apps, and make this information available, so that corrective action can take place (e.g. replace a missing application instance).

Diagram below illustrates the relationship between Cloud Foundry components running in a cloud-based distributed installation.



Cloud Foundry Workflow

Whenever an application is pushed to a Cloud Foundry installation using the CLI, a carefully orchestrated sequence of steps is followed to make sure the application is staged, started, scaled, and made accessible. Figure below shows a time-diagram with these sequence of steps.



Command Line Interface (CLI)

Interaction with a Cloud Foundry installation can be done using a CLI tool (Command Line Interface), named **cf**. The CLI tool connects to a Cloud Foundry application controller using a REST API, and issues commands to push, scale, and configure applications and services. The CLI can be installed and run in any of the common OSs (Linux, Mac, Windows), including/normally from a developer or operator workstation or laptop.

Running **cf** without any parameters will prompt it to display the list of all supported command-line commands organized by categories. Each command is identified by name, and the most commonly used ones, also by a single or multi-letter alias. Optional parameters to commands are specified and identified by name prefixed by -, while mandatory parameters are identified positionally just after the command name. The general syntax of the CLI commands is described below:

» TIP: CLI Command Syntax

```
$ cf
[env vars] cf [global options] command [args...] [options]
```

» TIP: Asking for Command Help

```
$ cf login -h
...
USAGE:
   cf login [-a API_URL] [-u USERNAME] [-p PASSWORD] [-o ORG] [-s SPACE]
```

The first step to follow to start using the Cloud Foundry CLI is to specify the URI of the application controller API, and the user credentials that authenticate the CLI requests. This can be conveniently done in a single step with command **login** (or **l**). Optional parameters **-u** and **-p** are used to specify user credentials. If omitted, the username/email and password are asked interactively. Parameter **-a** is use to specify the URI of the Cloud Controller REST API endpoint to connect with. Alternatively, the command **api** can be use to set (or show) the API endpoint. In the examples below, we use mostly the Pivotal API hosted in AWS: **api.run.pivotal.io**.

» Example: Login to PWS with CF-CLI

```
$ cf login -u myuser@myorg.org -p “s3cret!” -a api.run.pivotal.io
API endpoint: https://api.run.pivotal.io
Authenticating...
OK
Targeted org myorg
```

```
Targeted space development
API endpoint:   https://api.run.pivotal.io (API version: 2.22.0)
User:           myuser@einnovator.org
Org:            myorg
Space:          development
```

» Example: Showing and Changing API

```
$ cf api
API endpoint: https://api.run.pivotal.io (API version: 2.22.0)
$ cf api cf.aws.myorg.org
Setting api endpoint to cf.aws.myorg.org...
OK
API endpoint:   https://cf.aws.myorg.org (API version: 2.22.0)
Not logged in. Use 'cf login' to log in.
```

» Example: Getting API Info

```
$ curl api.run.pivotal.io/info
{"name":"vcap","build":"2222","support":"http://...", ... }
```

Table below summarizes the commands supported by the CLI for basic access to a Cloud Foundry installation:

Command	Description
login, l	Login to Cloud Foundry API
api	Show or Set API
logout, lo	Logout from Cloud Foundry API
auth	Authenticate user
target, t	Set target space or organization

Pushing Applications

The main purpose of Cloud Foundry is to allow applications to be deployed and run in a cloud environment. The CLI supports this with command **push** (or **p**). Each application is identified by a unique name. The location of Application artefacts (files) is specified with parameter **-p**. This can be a single file (e.g. a Java WAR file) or a directory (e.g. the root of a Ruby source directory tree). If omitted the current directory is assumed. For some languages like Java, a single file must be specified (a WAR or runnable JAR). When an application is push a *buildpack* is automatically selected. The options **-b name|URI** overrides this. Application are pushed to the current space (set with command **target -s**), unless option **-s space** overrides this.

As the application is being pushed the log details of the creation of the droplet are streamed to the CLI. The application is started automatically after the droplet is created, unless the option **–no-start** is specified.

A web route is also automatically established (unless option **–no-route** is specified), with URI having *hostname* the name of the app and with a default domain determined by API connect to. An alternative hostname the route can be specified with parameter **–n hostname**, and an alternative DNS domain specified with **-d domain**.

» Example: Pushing Application

```
$ cf push mypowerapp -p powerapp.war
Creating app mypowerapp in org myorg / space development as
myuser@myorg.org...
OK
Creating route mypowerapp.cfapps.io...
OK
Binding mypowerapp.cfapps.io to mypowerapp...
OK
Uploading mypowerapp...
Uploading app files from: mypowerapp.war
Uploading 3.1K, 10 files
Done uploading
OK
Starting app mypowerapp in org myorg / space development as
myuser@myorg.org...
...
App started
...
```

The command **apps** (or **a**) displays the list of pushed application with its details such as status and scale parameters, such as number of instances, and allocated memory. The command **app** display the status of an individual app. (The details are showed when the app is pushed.)

» Example: Listing Applications

```
$ cf apps
Getting apps in org myorg / space development as ...
OK
name   requested state   instances   memory   disk   urls
mypowerapp   started        1/1          1G      1G
mypowerapp.cfapps.io
```

» Example: Getting Application Status and Details

```
$ cf app mypowerapp
Showing health and status for app mypowerapp in org myorg ...
OK
...
requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: mypowerapp.cfapps.io
last uploaded: Wed Feb 11 10:24:13 UTC 2015
     state     since                  cpu    memory          disk
#0   running   2015-02-11 10:24:51 AM   0.0%   129.8M of 1G   107M
```

of 1G

Once an application is pushed it can be rescaled with command **scale** – including, the number of instances – parameter **-i nInstances**, the memory allocated to each instance – parameter **-m memSize**, or the temporary disk quota allocated to each instance – parameter **-k diskSize**. Option **-f** forces application restart.

» Example: Scaling an Application

```
$ cf scale mypowerapp -i 2 -m 512M -k 128M -f
Scaling app mypowerapp in org myorg  / space development as ...
OK
Stopping app mypowerapp in org myorg  / space development as ...
OK
Starting app mypowerapp in org myorg / space development as …
0 of 2 instances running, 2 starting
2 of 2 instances running
App started
...
requested state: started
instances: 2/2
usage: 512M x 2 instances
urls: mypowerapp.cfapps.io
last uploaded: Wed Feb 11 10:24:13 UTC 2015

     state     since                  cpu    memory         disk
#0   running   2015-02-11 11:54:51 AM   0.0%   128.6M of 512M
107M of 128M
#1   running   2015-02-11 11:54:44 AM   1.5%   135.2M of 512M
107M of 128M
```

Application and instance status can be changed with commands: **stop** (or **sp**) **–** to stop an application; **start** (or **st**) – to start an application; restart (or **rs**)- to restart; **restart-app-instance** – to restart the app instance with specified index. Command **restage** (or **rg**) recreates the droplet of an application, without pushing a new release (required after changing environment variables that affect the the buildpack).

Table below summarizes the commands supported by Cloud Foundry CLI used to push and stage applications:

Command	Description
push, p	Push an application
apps, a	List pushed applications
app	Show app status and details
scale	(re)Scale application
restage, rg	Restage application
stop,sp	Stop application
start,st	Start application
restart,rs	Restart application
restart-app-instance	Restart specific application instance

Logging

Application instances aggregated **logs** can be inspect with command logs – the tail of the logs. Option **–recent** show last entries in the logs. Command **events** show recent life-cycle events for an app.

» Example: Show App Recent Aggregated Logs

```
$ cf logs –recent mypowerapp
```

» Example: Show Recent App Life-Cycle Events

```
$ cf events mypowerapp
```

Service Bindings

Cloud Foundry allows applications to connect and use external services to support their business functionality – such as, a data-base to read/write data, or a messaging system to communicate asynchronously with other applications. Pre-configured or externally managed services are available through a service directories designated marketplaces. The CLI command **marketplace** (or **m**) lists all available services in a marketplace, and corresponding plans (pricing vs. feature model). With option **-s service** show the descriptions of the plans.

» Example: Show List of Services in Marketplace

```
$ cf marketplace
Getting services from marketplace in org myorg / space ... OK
service        plans                          description
...
cleardb        spark, boost*, amp*, shock*    Highly available MySQL...
cloudamqp      lemur, tiger*, bunny*, ...     Managed HA RabbitMQ...
```

» Example: Show Details of a Service

```
$ cf m -s cloudamqp
Getting service plan information for service cloudamqp as … OK
service plan   description                                  free or paid
lemur          Shared cluster with low limits for free      free
tiger          Shared cluster for production apps           paid
...
```

The command **create-service** (or **cs**) requests for a service (instance) to be provisioned to be used by applications – i.e.

the Cloud Controller asks to the service provider to allocated and setup whatever resources and configuration required for application to use a service. Service instance creation is bound to a *space* (in an *organization*). The command **services** (or **s**) display the list of all service instances provisioned in the current target space, and command **service** show the details for a single provisioned service. The plan for a service instance can be updated with command **update-service**. The name can be changed with command **rename-service**. A service instance can be delete with command **delete-service** (or **ds**).

» Example: Help on create-service

```
$ cf create-service -h
...
USAGE:
   cf create-service SERVICE PLAN SERVICE_INSTANCE
```

» Example: Create/Provision a Service (Instance)

```
$ cf create-service cleardb spark mysql
Creating service mysql in org myorg / space development … OK
$ cf cs cloudamqp lemur amqp
Creating service amqp in org myorg / space development ...
```

» Example: List Service Instances (in current target Space)

```
$ cf services
Getting services in org myorg / space development as ... OK
name          service       plan      bound apps
amqp          cloudamqp     lemur
mysql         cleardb       spark
```

» Example: Show Details of a Service Instance

```
$ cf service mysql
Service instance: mysql
Service: cleardb
Plan: spark
Description: Highly available MySQL for your Apps.
Documentation url:
Dashboard: https://cloudfoundry.appdirect.com/...?serviceUuid=22...
```

A service instance can not be used by an application until it is explicitly bound to the application. This is done with command **bind-service** (or **bs**). The effect of this is to make the details about the service instance, including access URL and credentials, available in the environment of an application (in variable VCAP_SERVICES). The app still has the responsibility to lookup this informations from the environment and use it to configure an appropriate driver to connect to the service. Any number of apps can bind to the same service instance. The command **unbind-service** (or **us**) remove the binding of an app to a service instance.

» Example: Bind Service Instances to App

```
$ cf bind-service mypowerapp mysql
Binding service mysql to app mypowerapp in org myorg / space ...
$ cf bind-service mypowerapp amqp
```

» Example: Confirm Service Instances are Bound to App

```
$ cf services
Getting services in org myorg / space development as ... OK
name          service       plan      bound apps
amqp          cloudamqp     lemur     mypowerapp
mysql         cleardb       spark     mypowerapp
```

It is also possible to bind application to service (instances) that are not available in the marketplace, with command **create-user-provided-service** (or **cups**). All the information necessary to connect to the service, such as URL and credentials, is specified with parameter **-p** – either using a JSON object (*{ name:value, ..}*) as syntax, or by specifying a comma separated list of attribute names whose values are asked interactively. This service information will be available to bound applications in the environment as is the case with marketplace services.

» Example: Create User Provided Service

```
$ cf cups mydb -p '{"url":"jdbc:mysql:db.myorg.org",
"user":"myuser","password":"s3cret"}'
Creating user provided service mydb in org myorg / space .. .
```

» Example: Create User Provided Service (Windows)

```
$ cf cups mydb -p
"{""url"":""jdbc:mysql:db.myorg.org"",""user"":""myuser""
,""password"":""s3cret""}"
```

Table below summarizes the CLI commands related to services used by developers.

Command	Description
marketplace, m	List services in marketplace
create-service, cs	Create/Provision service instance
update-service	Change plan for service instance
rename-service	Change name of service instance
delete-service	Delete service instance
services, s	List provisioned services
service	Show service instance details

bind-service,bs	Bind service to application
create-user-provided-service, cups	create user provided (non marketplace) service
update-user-provided-service, uups	update user provided (non marketplace) service

Manifest Files

All the settings specified when an app is pushed can be conveniently specified in a manifest (configuration) file, written in YAML syntax. By default, a file named *manifest.yml* in the current work directory is used as manifest, unless an alternative is specified with **push** option **-f**. A manifest file can configure a single or multiple apps. The use of a manifest file makes the *app-name* parameter optional in command **push –** the name found in the manifest is used. With multiple app manifest files, omitting the app name in the **push** command pushes all apps. If a name is specified, only the app with that name is pushed. Manifest settings overwrite previous command settings, but settings in the current **push** command overwrite manifest settings.

Application settings are done under field named *applications*, whose value is a list of value objects – one per application – whose field *name* designates the app. Settings common to all apps are specified as fields outside the field *applications*. [YAML supports both inline and block syntax-style to define objects. Symbol – is used to define items in list values (block), or list syntax [.., ..] (inline). Fields are defined as *name : value* (inline), or with the *value* in a newline (block).]

» Example: Manifest File for a Single App (manifest.yml)

```
applications:
- name: mypowerapp
  memory: 512M
  instances: 2
  host: mypowerapp
  hosts:
  - mightypowerapp
  - superpowerapp
  path: ./mypowerapp.war
  services:
  - amqp
  - mysql
  env:
    defaultLocale: en-US
    spring.profiles.active: dev,jpa
```

» Example: Push App with Single App Manifest

```
$ cf push
```

» Example: Use Alternative Manifest File

```
$ cf push -f manifest2.yml
```

» Example: Manifest File for Multiple Apps

```
domain: myorg.org
services:
 - amqp
 - mysql
env:
 spring.profiles.active: dev,jpa
applications:
- name: mypowerapp
  instances: 2
  memory: 512M
  path: ./mypowerapp.war
  env:
    defaultLocale: en-US
- name: otherpowerapp
  instances: 1
  memory: 512M
  path: ./otherpowerapp.war
```

Environment Variables

As it also the case in non-cloud scenarios, application deployed trough Cloud Foundry have available to them a set of externally defined settings that can be used to influence its bootstrap configuration or running behavior – the **Environment**. Some environment variables are automatically set by the Cloud Foundry, including:

- VCAP_APPLICATION – JSON object with application detail (e.g. name, scaling limits, URIs, etc.)
- VCAP_SERVICES – JSON object with list and details of services instance an application is bound to.

Customer environment variables are set with command **set-env** (or **se**), and removed with command **unset-env**. The command **env** (or **e**) show the list of all environment variables.

» Example: Set App Custom Environment Variables

```
$ cf set-env mypowerapp defaultLocale en-UK
Setting env variable 'spring.profiles.active' to 'dev' for app ...
$ cf set-env mypowerapp spring.profiles.active dev
```

» Example: Display App Environment Variables

```
$ cf env mypowerapp
Getting env variables for app mypowerapp in org myorg / space ...OK
System-Provided:
{ "VCAP_SERVICES": {
  "cleardb": [ … ],
  "cloudamqp": [ … ],
  "user-provided": [ {  "credentials": { .. } , … }, … ]}
{ "VCAP_APPLICATION": {
  "application_name": "mypowerapp",
  … }
}
```


```
User-Provided:
defaultLocale: en-US
spring.profiles.active: dev
```

Table below summarizes environment related CLI commands.

Command	Description
env, e	List all environment variables
set-env, se	Set custom environment variable
unset-env	Unset custom environment variable

Spring Cloud

Spring Cloud is a collection of projects that simplifies the way Java application operate in a Cloud environment. **Spring Cloud Core** and **Spring Cloud Connectors** projects, in particular, allows application to easily access and parse environment variables such as VCAP_APPLICATION and VCAP_SERVICES, and create drivers for bound service instances. The Java class **Cloud** provides an API that encapsulates access to app instance information and the service instance. A pluggable mechanisms (based on Java services SPI) allows the Cloud Foundry specifics to be automatically enabled.

» Example: Getting Cloud App Info with Spring Cloud

```
CloudFactory cloudFactory = new CloudFactory();
Cloud cloud = cloudFactory.getCloud();
ApplicationInstanceInfo info = cloud.getApplicationInstanceInfo();
writer.format("AppId: %s\n", info.getAppId());
writer.format("InstanceId: %s\n", info.getInstanceId());
writer.format("Properties: %s\n", info.getProperties());
```

» Example: Getting Info on Bound Services Instances

```
List<ServiceInfo> services = cloud.getServiceInfos();
for (ServiceInfo service: services) {
    writer.format("Service: %s<br/>\n", service.getId());
}
```

Resources

- Cloud Foundry Documentation - http://docs.cloudfoundry.org/
- Pivotal Cloud Foundry Page – http://www.pivotal.io/platform-as-a-service/pivotal-cloud-foundry
- Spring Cloud Project - http://projects.spring.io/spring-cloud
- Git repository for Cloud Foundry - https://github.com/cloudfoundry

About the Author

Jorge Simão is a software engineer and IT Trainer, with two decades long experience on education delivery both in academia and industry. Expert in a wide range of computing topics, he his an author, trainer, and director (Education & Consulting) at EInnovator. He holds a B.Sc., M.Sc., and Ph.D. in Computer Science and Engineering.

Cloud Foundry Training & Consulting

Take a three-day instructor-led course in Developing Applications with Cloud Foundry and PCF. The course provides hands-on experience on deploying and managing applications in Cloud Foundry, using the CLI tool and Web-Console, and guides you trough the internal components and architecture of Cloud Foundry, staging workflow, admin concepts, service bindings, build-packs, logging, monitoring, and Spring Cloud. Consulting sessions on the follow-up of training also available. Book now an on-site class: **www.einnovator.org/course/cloud-foundry-developer**

Contacts

Training – Bookings & Inquiries
training@einnovator.org

Consultancy – Partnerships & Inquiries
consulting@einnovator.org

General Info
info@einnovator.org