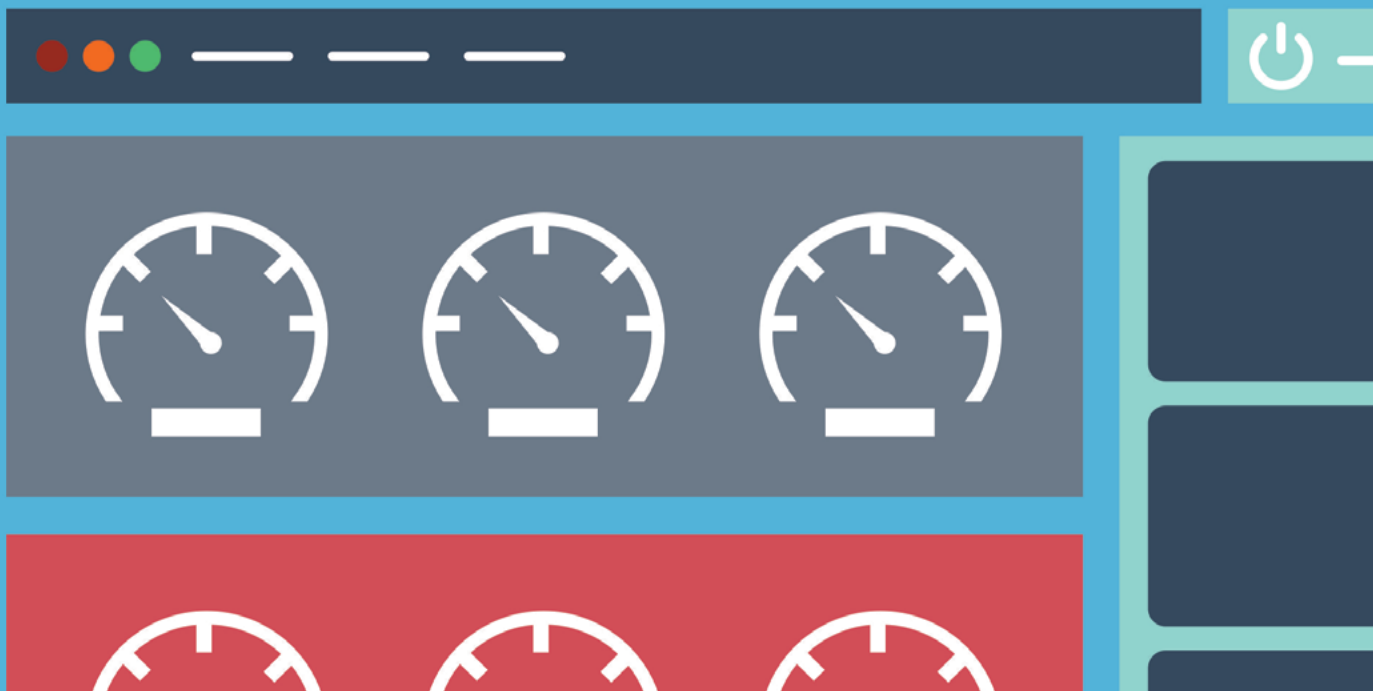


The Easiest Way to Get Started with Cloud Foundry

Deploy apps faster on any cloud with Stackato, the leading enterprise PaaS based on Cloud Foundry, Docker and other open source technologies. Free to use in production up to 20GB.

[Get Started »](#)

Learn more: stackato.com/refcard



CONTENTS

- » Supported Technologies
- » Your First Cloud Foundry App
- » Advanced Usage
- » Deploying Cloud Foundry
- » Getting Help
- » cf CLI Cheat Sheet...and more!

Cloud Foundry

Open-Source PaaS for Streamlined Development and Deployment

By Jeremy Voorhis, Revised and Updated by Billy Tat

ABOUT CLOUD FOUNDRY

Cloud Foundry is an open source Platform as a Service (PaaS) governed by the Cloud Foundry Foundation. Cloud Foundry enables developers and teams to deploy and manage applications and services (databases, message queues, 3rd party APIs) through its REST API, and handles details like load balancing and restarting crashed applications automatically.

The open source Cloud Foundry code is available as a collection of open source repositories, available at <https://github.com/cloudfoundry>. You can also sign up for a hosted service provider, or download the software from an on-premises vendor or systems integrator.

In this Refcard, you will learn about:

- Technologies supported by Cloud Foundry
- Finding a hosting provider, or deploying Cloud Foundry yourself
- Writing apps for and deploying to Cloud Foundry
- Scaling and managing apps in the cloud
- Finding help within the community
- Using the cf Command Line Interface (CLI)

Whether you are a cloud veteran, or new to PaaS, this Refcard will help you build, deploy, and manage applications on Cloud Foundry with ease.

SUPPORTED TECHNOLOGIES

BASE OS

Cloud Foundry is officially supported on Canonical's **Ubuntu 10.04 LTS** running on **X86-64** cores. Applications that are compatible with this OS and CPU architecture should require little to no modification.

BUILDPACKS – FRAMEWORKS AND RUNTIMES

Buildpacks manage the framework and runtime support of an application. In Cloud Foundry's domain, a framework encapsulates the knowledge of how to build, stage, and run an application. For example, a Ruby on Rails application will have a file named `database.yml`, which describes how to connect to the database, and may have a collection of assets, such as JavaScript, CSS, and images to be minified.

Runtimes encapsulate programming language environments. Because not all applications are compatible with every release of a programming language, multiple versions are supported. Buildpacks examine user-provided artifacts to decide the appropriate framework, runtime, and dependencies to download and install. These user-provided artifacts are also

used to determine how to configure an application to interact with bound services.

When you push an application, Cloud Foundry automatically detects which buildpack is required and installs it for the application to run.

Cloud Foundry includes a set of system buildpacks for use:

SYSTEM BUILDPACK

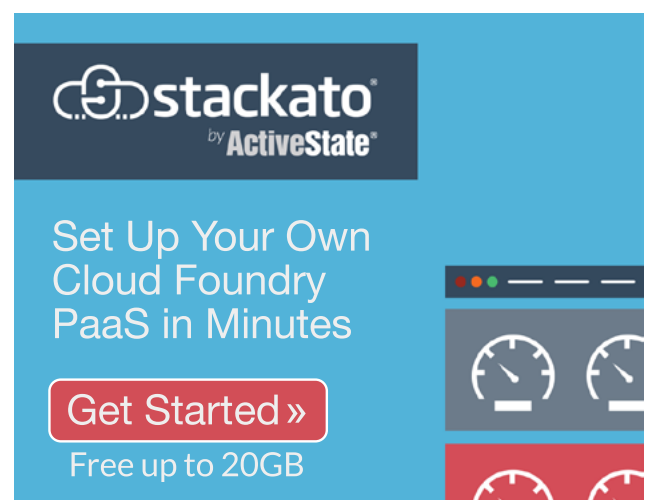
NAME	SUPPORTED LANGUAGES AND FRAMEWORKS	GITHUB REPO
Go	N/A	Go source
Java	Grails, Play, Spring, or any other JVM-based language or framework	Java source
Node.js	Node or JavaScript	Node.js source
PHP	N/A	PHP source
Python	N/A	Python source
Ruby	Ruby, Rack, Rails or Sinatra	Ruby source

If your application uses a language or framework that Cloud Foundry buildpacks do not support, you can try the following:

- Customize an existing buildpack
- Write [your own buildpack](#)
- Use a [Cloud Foundry Community Buildpack](#)
- Use a [Heroku Third-Party Buildpack](#)

SERVICES

Services are hosted software that Cloud Foundry users can provision and bind to their applications. Most of these services are database systems such as MySQL, PostgreSQL, and MongoDB; but RabbitMQ, an AMQP message broker,



stackato[®]
by ActiveState[®]

Set Up Your Own
Cloud Foundry
PaaS in Minutes

Get Started »

Free up to 20GB

is another example of a service. Services can be provisioned by Cloud Foundry users through the REST API, and can be bound to apps in an N:M topology. While most services will be databases bound to a single application, a complex deployment may consist of two applications—a front-end and back-end connected via a RabbitMQ service.

Various distributions offer additional supported services and users can even add their own via the ‘user-provided services’ API.

COLLABORATION

Cloud Foundry provides support for teams of people via its organizations feature. Cloud Foundry users have the ability to create an organization and assign resources (e.g. memory, service instances) to spaces within that organization. This allows organizations to give developers access to build and test their applications, and to assign another set of resources and users for operations. Read more about organizations here: <http://docs.cloudfoundry.org/concepts/roles.html>.

CLOUD FOUNDRY PROVIDERS

Cloud Foundry is governed by the Cloud Foundry Foundation, which currently consists of over 40 member companies. Many of the companies that have emerged as Cloud Foundry providers further extended the open source code with interesting capabilities, including additional programming languages, services, third-party add-on marketplaces, and enterprise support.

PROVIDERS

- Cloud Foundry Foundation – <http://cloudfoundry.org/>
 - Governing foundation for open source Cloud Foundry
- ActiveState Stackato – <http://www.activestate.com/stackato>
 - Commercial distribution as downloadable software
 - Free for production use up to 20GB RAM
- AppFog – <http://appfog.com/>
 - Commercial hosted service
 - 30-day free trial (up to 2GB RAM)
- HP Helion Development Platform – <http://www8.hp.com/us/en/cloud/helion-developer.html>
 - Commercial distribution bundled with HP Helion OpenStack
 - Downloadable software or free trial on HP Helion Public Cloud
- IBM Bluemix – <http://www.ibm.com/software/bluemix/>
 - Commercial hosted service
 - 30-day free trial (up to 2GB RAM)
- Pivotal Cloud Foundry – <http://pivotal.io/platform-as-a-service/pivotal-cloud-foundry>
 - Commercial distribution as downloadable software
 - 90 day evaluation term, non-production use, up to 50 app instances
- Pivotal Web Services – <https://run.pivotal.io/>
 - Commercial hosted service
 - 60-day free trial (up to 2GB RAM)

YOUR FIRST CLOUD FOUNDRY APP

SETTING UP

Before you can start using Cloud Foundry, a target API is required, along with an API client. For this tutorial, we will use <http://run.pivotal.io/> as an example. You can sign up for a free account there. However, these instructions are applicable to any other provider. Download the cf CLI at <https://github.com/cloudfoundry/cli/releases> and follow the instructions for your operating system at <http://docs.cloudfoundry.org/devguide/installcf/install-go-cli.html>.

Test your new installation by running the cf CLI. You should see a listing of available commands and options.

DEPLOYING A NODE.JS APP

Now you’re ready to deploy your first app. Let’s create a Node.js app. Add the following code to a file named server.js:

```
var http = require('http');

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello from Cloud Foundry!\n");
}).listen(process.env.VCAP_APP_PORT);
```

This is nearly identical to the canonical “Hello, World!” application for Node.js, with one exception. The web server is configured to listen on a port defined by an environment variable named `VCAP_APP_PORT`. VCAP is the code name for the collection of components that implement the core of Cloud Foundry. In order to manage a massive number of apps, VCAP will assign your app a TCP port to listen to, and its load balancer will automatically direct traffic to each of your app’s processes.

Next, create a file named package.json with the following code:

```
{
  "name": "hello-refcardz",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  }
}
```

The package.json file is required in order for the Node.js buildpack to detect your app as a Node.js app and prepare an appropriate environment.

Now that you have your Cloud Foundry account and the cf CLI, you are ready to deploy. To login and deploy, run the following commands.

```
$ cf api api.run.pivotal.io
$ cf login -u your-email@example.com
$ cf push hello-refcardz
```

After successfully pushing your app, you should be able to visit its URL and see the text “Hello from Cloud Foundry!”

Here is an abbreviated example of what you should see:

```
$ cf api api.run.pivotal.io
Setting api endpoint to api.run.pivotal.io... OK
$ cf login -u your-email@example.com
API endpoint: https://api.run.pivotal.io
Uploading hello-refcardz... OK
[...snip...]
Starting hello-refcardz... OK
Checking hello-refcardz... OK
$ curl hello-refcardz.cfapps.io
Hello from Cloud Foundry!
```

ADDING A SERVICE

Now that you can deploy an app, let's add a service to make it more useful. To see the list of available services and their associated plans, run the following command:

```
$ cf marketplace
```

Let's create a Redis service named *hello-redis* using the free 25mb plan.

```
$ cf create-service rediscloud 25mb hello-redis
```

In order for your app to connect to the Redis service, you must bind the service to your app with the following command:

```
$ cf bind-service hello-refcardz hello-redis
```

Cloud Foundry will now generate Redis credentials for your app and expose them using the environment variable named `VCAP_SERVICES` at runtime. Now, let's write some code that makes use of Redis. Open `server.js` in your text editor and change it to the following:

```
var http = require('http');
var vcap_services = process.env.VCAP_SERVICES;
var rediscloud_service = JSON.parse(vcap_services)
  ["rediscloud"][0];
var creds = rediscloud_service.credentials;
var redis = require('redis');
var client = redis.createClient(creds.port, creds.hostname,
  {no_ready_check: true});

client.auth(creds.password);

http.createServer(function(request, response) {
  client.incr("counter");
  client.get("counter", function(err, count) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.end("This page has been viewed " + count + "
  times\n");
  });
}).listen(process.env.VCAP_APP_PORT);
```

Before we can push our new code, we need to tell Cloud Foundry that we want to use the `node_redis` library to connect. Update the `package.json` file to include the `redis` library then push the code.

```
{
  "name": "hello-refcardz",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "redis": "^0.12.1"
  },
  "scripts": {
    "start": "node server.js"
  }
}

$ cf push hello-refcardz
```

Now visit your app's URL and enjoy your shiny new hit counter.

CONNECTING TO YOUR SERVICE LOCALLY

For development and administration, developers may want to connect to their services within Cloud Foundry from their workstations. You can do this using the service's credentials and the appropriate client for the server. The service's credentials are available in the service dashboard. The dashboard's address can be found by running the `cf service` command:

```
$ cf service hello-redis
```

Assuming you created the Redis service in the previous section, and the `redis-cli` command is available, you can connect to your service like so:

```
$ redis-cli -h <host> -p <port> -a <password> -n <db name>

$ GET counter

"88"
```

SCALING UP AND OUT

If your app requires more than the default amount of memory, you can add more using the `cf scale` command. (Use the `cf app` command to see how much memory your app is currently consuming.)

Cloud Foundry does not support auto-scaling. Whether you are deploying to a hosting provider or hosting Cloud Foundry yourself, you can implement auto-scaling by monitoring memory and CPU usage through the REST API, with the [cfoundry](#) client library, or by scripting the `cf CLI`. This usage data can be used to make decisions to add and remove application instances automatically.

ENVIRONMENT VARIABLES

If you need to configure your app and do not want to use the file system, you can use environment variables. This is especially convenient when you have development and production deployments of your app and do not want to maintain separate config files for things like API tokens, Google Analytics codes, and more. Use the `env`, `set-env`, and `unset-env` commands to view, set, and remove environment variables.

DEBUGGING AND MONITORING

Let's make our app crash occasionally by changing our `server.js` file to the following code:

```
var express = require('express');
var expressWinston = require('express-winston');
var winston = require('winston');
var errorHandler = require('errorhandler');
var app = express();

app.use(expressWinston.logger({
  transports: [
    new winston.transports.Console({
      json: true,
      colorize: true,
    })
  ]
}));

app.use(errorHandler());

app.get('/', function(req, res, next) {
  if (Math.random() < 0.3) {
    return next(new Error("Your luck has run out!"));
  }
  res.end("It's your lucky day!");
});

app.listen(process.env.VCAP_APP_PORT);
```

Notice that we've also added the [express](#), [winston](#) and [winston-express](#), and [errorhandler](#) libraries. Let's add them to our `package.json` file so it looks like this:

```

{
  "name": "hello-refcardz",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "errorhandler": "^1.3.4",
    "express": "^4.11.2",
    "express-winston": "^0.2.12",
    "winston": "^0.9.0"
  },
  "scripts": {
    "start": "node server.js"
  }
}

```

Now push your app with the `cf` command, and send it enough requests to see both the successful case and the error. Pull the logs with the `cf logs` command, and you will see requests logged to `logs/stdout.log`, and errors with full stack traces logged to `logs/stderr.log`.

While this tutorial only demonstrates error handling for Node.js apps, the operating principles are the same for all frameworks. `STDOUT` is logged to `logs/stdout.log` and `STDERR` is logged to `logs/stderr.log`. Additional log files can be pulled with `cf logs`.

For production use, you might also want to integrate with a third-party logging service such as [Loggly](#) or [Papertrail](#) for log retention and search, [New Relic](#) for performance monitoring, [Pingdom](#) for uptime monitoring, and [PagerDuty](#) to wake you if things go wrong.

ADVANCED USAGE

PREDICTABLE DEPLOYMENTS WITH MANIFESTS

When you push your application with a manifest present in your current working directory, the `cf` CLI will read the attributes in the manifest, such as the name, path to files on disk, and desired RAM—instead of requiring you to pass them as arguments.

A manifest is a [YAML](#) document that describes how your application will be deployed. This file records the buildpack, memory, number of instances, URL, and services that are required to run your application. This file serves as [configuration management](#) for your app. While some details may differ from deployment to deployment, it is useful to check this file into source control, and create modified copies of it for staging or personal development environments within a hosted account, or even within your local Cloud Foundry instance.

If your application does not contain a manifest, you can generate one after it has been pushed by running `cf create-app-manifest`. Here's a sample manifest, taken from our Node.js app in the previous section:

```

applications:
- name: hello-refcardz
  memory: 64M
  instances: 1
  host: hello-refcardz
  domain: cfapps.io
  services:
  - hello-redis

```

An application manifest will also record the path to your application files relative to the manifest file itself. Manifests also support multiple applications, allowing you to reproduce a [service-oriented architecture](#) with a single `cf` push.

You can learn more about Cloud Foundry manifests at <http://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html>.

ZERO-DOWNTIME DEPLOYMENTS

While the `cf` command does provide the ability to restart an app, it may not be obvious that this is implemented by stopping and then starting the app regardless of any requests the app may be serving. How can you update your Cloud Foundry app without being penalized with downtime?

In Chapter 10 of their book titled *Continuous Delivery*, Jez Humble and David Farley describe a technique called **Blue-Green Deployments** that allows you to accomplish this. A blue-green deployment takes advantage of two identical production environments side-by-side, with the ability to toggle between the two. This also adds the ability to roll changes back quickly.

To implement blue-green deployments with Cloud Foundry, create two applications and use the router as the toggling mechanism.

1. Deploy your app with the suffix `-green`. Give it its own unique, non-production URL.
2. Deploy your app with the suffix `-blue`. Give it its own unique, non-production URL.
3. Ensure their environment variables and bindings are identical.
4. Map your production URL to your green deployment.
5. Deploy your changes (code, environment variables) to your blue deployment.
6. Test your blue deployment using its unique URL.
7. If the tests succeed, map your production URL to the blue deployment and unmap it from green.

Note that during step 7, the production URL will be mapped to both deployments for a brief window. This implies that both versions of your app must be compatible with any services they are bound to.

These steps can be scripted, and even run by your continuous integration server. Some Cloud Foundry providers, such as Stackato and HP Helion, have functionality which handles these steps automatically.

CONNECTING TO THIRD-PARTY APIS

Cloud Foundry makes it easy to manage the configuration of your apps through **environment variables**. When you need to connect to a third-party API you should create environment variables for its URL and credentials (username, password, API token, etc.). This allows you to use the same application code to connect to the production API in your production deployment, and the API's sandbox account for development and testing.

CONNECTING APPS WITHIN CLOUD FOUNDRY

There are two common methods to connect your apps within Cloud Foundry: APIs and services.

When connecting apps via APIs, the app that provides the API should be implemented like any public-facing API:

- Ensure all requests are authenticated.
- Ensure the API will only serve responses over HTTPS.
- Apps consuming the API should be configured as if they were connecting to a third-party API, with configuration stored in environment variables.

Using APIs internally has the unfortunate effect that app instances cannot talk directly to each other, and requests will need to pass through the SSL terminator and load balancer.

It is also possible to connect two apps by using a service. By connecting two apps to a message queue such as [RabbitMQ](#) or [Redis](#), Cloud Foundry's service bindings handle the details of authentication for you, making your apps' configuration more secure and less fragile.

Both RabbitMQ and Redis offer pub/sub messaging, which allows producers to post messages to the service, and consumers to subscribe to messages that interest them. Using pub/sub messaging, it is possible to implement the most common messaging patterns. Let's look at a native request/reply example that measures round trip latency via Redis's pub/sub API.

First, create a Redis service and name it `redis-rtt`. Then create a standalone (non-web) app with the following code and a corresponding `package.json` file. Bind it to your newly created Redis service. A URL does not need to be mapped as this is a standalone app.

```
// echo.js
var vcap_services = process.env.VCAP_SERVICES;
var rediscloud_service = JSON.parse(vcap_services)
["rediscloud"][0];
var creds = rediscloud_service.credentials;
var redis = require('redis');
var subscriber = redis.createClient(creds.port, creds.
hostname, {no_ready_check: true});
var publisher = redis.createClient(creds.port, creds.
hostname, {no_ready_check: true});

subscriber.auth(creds.password);
publisher.auth(creds.password);

subscriber.on("pmessage", function(pattern, channel,
message) {
  replyChannel = "client:" + channel.split(":")[1];
  publisher.publish(replyChannel, message);
});

subscriber.psubscribe("server:*");

// package.json

{
  "name": "echo-svc",
  "version": "1.0.0",
  "description": "",
  "main": "echo.js",
  "dependencies": {
    "redis": "^0.12.1"
  },
  "scripts": {
    "start": "node echo.js"
  }
}
```

Next, let's create a web app that publishes a timestamp message to our echo service. Be sure to include the `redis` and `express` dependencies in the `package.json` file. Bind it to our `redis-rtt` service, and:

```
// index.js

var vcap_services = process.env.VCAP_SERVICES;
var rediscloud_service = JSON.parse(vcap_services)
["rediscloud"][0];
var creds = rediscloud_service.credentials;
var redis = require('redis');
var express = require('express');
var app = express();
```

```
app.get('/', function(req, rep) {
  var subscriber = redis.createClient(creds.port, creds.
hostname, {no_ready_check: true});
  var publisher = redis.createClient(creds.port, creds.
hostname, {no_ready_check: true});
  subscriber.auth(creds.password);
  publisher.auth(creds.password);

  publisher.incr("rtt:counter", function(err, counter) {
    subscriber.on("message", function(channel, message) {
      var lat = Date.now() - parseInt(message);
      rep.end(" Latency is " + lat + "ms");
      subscriber.unsubscribe(channel);
      subscriber.end();
    });
    subscriber.subscribe("client:" + counter);
    publisher.publish("server:" + counter, Date.now());
    publisher.end();
  });
});

app.listen(process.env.VCAP_APP_PORT);

// package.json

{
  "name": "rtt-svc",
  "version": "1.0.0",
  "main": "index.js",
  "dependencies": {
    "redis": "^0.12.1",
    "express": "^4.12.0"
  },
  "scripts": {
    "start": "node index.js"
  }
}
```

Now when you visit your app, you will see the approximate latency for a round trip to your echo service via Redis pub/sub messaging. Because of the bindings that you created between Redis and your echo service, and Redis and your web app, you have not needed to share the location of either app with each other, or manage a single password.

Finally, let's tie it all together with a manifest. This time, we'll use a single manifest for both of our apps and our Redis service. This will allow us to re-create this configuration with a single `push`.

```
applications:
- name: echo-svc
  memory: 64M
  no-route: true
  instances: 1
  path: ./echo-svc
  services:
  - redis-rtt
- name: rtt-svc
  memory: 64M
  instances: 1
  path: ./rtt-svc
  services:
  - redis-rtt
```

You can clone the example code from <https://github.com/ActiveState/cf-redis-examples>.

DEPLOYING CLOUD FOUNDRY

BOSH

Cloud Foundry is a distributed system with dozens of interconnected components, so it is non-trivial to deploy. If you want to deploy Cloud Foundry yourself, VMware recommends [BOSH](#). The top-level construct of BOSH is the release, and the

release that implements Cloud Foundry is located at <https://github.com/cloudfoundry/cf-release>.

BOSH is an infrastructure-agnostic open source framework created by VMware’s Site Reliability Engineering Team to deploy distributed systems. BOSH conquers the problem of deploying distributed systems by dividing them into *deployments*, *jobs*, *packages*, and *releases*.

A deployment is a collection of cloud resources—compute, storage, network, and related entities—that are created and scaled by BOSH in order to deliver a release.

A BOSH release is a collection of jobs that cooperate to support the service described by a release. Each VM created by a BOSH deployment will run a single job, which is supervised by *Monit*.

A BOSH job references a collection of packages. A BOSH package describes how to download, build, and archive a software package such as Nginx, the JVM, or NFS, so that it can be delivered onto a BOSH deployment as part of a distributed system.

You can learn more about BOSH at <https://github.com/cloudfoundry/bosh>.

BOSH LITE

If you don’t want or need a full-fledged BOSH-managed deployment, e.g. for development, evaluation, or small deployments, you can use BOSH Lite. BOSH Lite utilizes Vagrant and a Warden BOSH CPI to create an instance of Cloud Foundry on a single virtual machine. Vagrant creates the initial VM and BOSH deploys each Cloud Foundry component into their own Warden container.

For more information about BOSH Lite visit <https://github.com/cloudfoundry/bosh-lite>.

GETTING HELP

If you have a problem and you haven’t found the answer in this Refcard, you can find help within the Cloud Foundry community.

RESOURCES

- Official Docs: <http://docs.cloudfoundry.org/>
- Official Support: <http://support.cloudfoundry.com/home>
- Directories of mailing lists, source code, and more: <http://cloudfoundry.org/>
- Cloud Foundry and BOSH developer documentation: <http://cloudfoundry.github.com/> and <http://bosh.io/docs/>
- Unofficial Cloud Foundry wiki: <https://github.com/cloudfoundry-community/cf-docs-contrib/wiki>
- IRC: #cloudfoundry on irc.freenode.net

CF CLI CHEAT SHEET

GETTING STARTED

<code>cf login</code>	Log user in.
<code>cf logout</code>	Log user out.
<code>cf passwd</code>	Change user password.

<code>cf target</code>	Set or view the targeted org or space.
<code>cf api [URL]</code>	Set or view target API URL.
<code>cf auth USERNAME PASSWORD</code>	Authenticate user non-interactively.

APPLICATIONS

<code>cf apps</code>	List all apps in the target space.
<code>cf app APP_NAME</code>	Display health and status for app.
<code>cf push APP_NAME</code>	Push a new app or sync changes to an existing app.
<code>cf scale APP_NAME</code>	Change or view the instance count, disk space limit, and memory limit for an app.
<code>cf delete APP_NAME</code>	Delete an app.
<code>cf rename APP_NAME NEW_APP_NAME</code>	Rename an app.
<code>cf start APP_NAME</code>	Start an app.
<code>cf stop APP_NAME</code>	Stop an app.
<code>cf restart APP_NAME</code>	Restart an app.
<code>cf restage APP_NAME</code>	Restage an app.
<code>cf restart-app-instance APP_NAME INDEX</code>	Terminate the running application instance at the given index and instantiate a new instance of the application with the same index.
<code>cf events APP_NAME</code>	Show recent app events.
<code>cf files APP_NAME [PATH]</code>	Print out a list of files in a directory or the contents of a specific file.
<code>cf logs APP_NAME</code>	Tail or show recent logs for an app.
<code>cf env APP_NAME</code>	Show all env variables for an app.
<code>cf set-env APP_NAME ENV_VAR_NAME ENV_VAR_VALUE</code>	Set an env variable for an app.
<code>cf unset-env APP_NAME ENV_VAR_NAME</code>	Remove an env variable.
<code>cf stacks</code>	List all stacks (a stack is a pre-built file system, including an operating system, that can run apps).
<code>cf copy-source SOURCE_APP TARGET_APP</code>	Make a copy of app source code from one application to another. Unless overridden, the copy-source command will restart the application.
<code>cf create-app-manifest APP</code>	Create an app manifest for an app that has been pushed successfully.

SERVICES

<code>cf marketplace</code>	List available offerings in the marketplace.
<code>cf services</code>	List all service instances in the target space.

<code>cf service SERVICE_INSTANCE</code>	Show service instance info.
<code>cf create-service SERVICE_PLAN SERVICE_INSTANCE</code>	Create a service instance.
<code>cf update-service SERVICE_INSTANCE</code>	Update a service instance.
<code>cf delete-service SERVICE_INSTANCE</code>	Delete a service instance.
<code>cf rename-service SERVICE_INSTANCE NEW_SERVICE_INSTANCE</code>	Rename a service instance.
<code>cf bind-service APP_NAME SERVICE_INSTANCE</code>	Bind a service instance to an app.
<code>cf unbind-service APP_NAME SERVICE_INSTANCE</code>	Unbind a service instance from an app.
<code>cf create-user-provided-service SERVICE_INSTANCE</code>	Make a user-provided service instance available to cf apps.
<code>cf update-user-provided-service SERVICE_INSTANCE</code>	Update user-provided service instance name value pairs.

ORGANIZATIONS (V2)

<code>cf orgs</code>	List all orgs.
<code>cf org ORG</code>	Show org info.
<code>cf create-org ORG</code>	Create an org.
<code>cf delete-org ORG</code>	Delete an org.
<code>cf rename-org ORG NEW_ORG</code>	Rename an org.

SPACES (V2)

<code>cf spaces</code>	List all spaces in an org.
<code>cf space SPACE</code>	Show space info.
<code>cf create-space SPACE</code>	Create a space.
<code>cf delete-space SPACE</code>	Delete a space.
<code>cf rename-space SPACE NEW_SPACE</code>	Rename a space.

DOMAINS (V2)

<code>cf domains</code>	List domains in the target org.
<code>cf create-domain ORG DOMAIN</code>	Create a domain in an org for later use.
<code>cf delete-domain DOMAIN</code>	Delete a domain.
<code>cf create-shared-domain DOMAIN</code>	Create a domain that can be used by all orgs (admin-only).
<code>cf delete-shared-domain DOMAIN</code>	Delete a shared domain.

ROUTES (V2)

<code>cf routes</code>	List all routes in the current space.
<code>cf create-route SPACE DOMAIN</code>	Create a URL route in a space for later use.
<code>cf check-route HOST DOMAIN</code>	Perform a simple check to determine whether a route currently exists or not.
<code>cf map-route APP_NAME DOMAIN</code>	Add a URL route to an app.
<code>cf unmap-route APP_NAME DOMAIN</code>	Remove a URL route from an app.
<code>cf delete-route DOMAIN</code>	Delete a route.
<code>cf delete-orphaned-routes</code>	Delete all orphaned routes (e.g.: those that are not mapped to an app).

BUILDPACKS

<code>cf buildpacks</code>	List all buildpacks.
<code>cf create-buildpack</code>	Create a buildpack.
<code>cf update-buildpack BUILDPACK</code>	Update a buildpack.
<code>cf rename-buildpack BUILDPACK_NAME NEW_BUILDPACK_NAME</code>	Rename a buildpack.
<code>cf delete-buildpack BUILDPACK</code>	Delete a buildpack.

USER ADMIN

<code>cf create-user USERNAME PASSWORD</code>	Create a new user.
<code>cf delete-user USERNAME</code>	Delete a user.
<code>cf org-users ORG</code>	Show org users by role.
<code>cf set-org-role USERNAME ORG ROLE</code>	Assign an org role to a user.
<code>cf unset-org-role USERNAME ORG ROLE</code>	Remove an org role from a user.
<code>cf space-users ORG SPACE</code>	Show space users by role.
<code>cf set-space-role USERNAME ORG SPACE ROLE</code>	Assign a space role to a user.
<code>cf unset-space-role USERNAME ORG SPACE ROLE</code>	Remove a space role from a user.

ORG ADMIN

<code>cf quotas</code>	List available usage quotas.
<code>cf quota QUOTA</code>	Show quota info
<code>cf set-quota ORG QUOTA</code>	Assign a quota to an org.
<code>cf create-quota QUOTA</code>	Define a new resource quota.
<code>cf delete-quota QUOTA</code>	Delete a quota.

SPACE ADMIN

<code>cf space-quotas</code>	List available space resource quotas.
<code>cf space-quota SPACE_QUOTA_NAME</code>	Show space quota info.
<code>cf create-space-quota QUOTA</code>	Define a new space resource quota.
<code>cf update-space-quota SPACE_QUOTA_NAME</code>	Update an existing space quota.
<code>cf delete-space-quota SPACE_QUOTA_NAME</code>	Delete a space quota definition and unassign the space quota from all spaces.
<code>cf set-space-quota SPACE_NAME SPACE_QUOTA_NAME</code>	Assign a space quota definition to a space.
<code>cf unset-space-quota SPACE_QUOTA</code>	Unassign a quota from a space.

SERVICE ADMIN

<code>cf service-auth-tokens</code>	List service auth tokens.
<code>cf create-service-auth-token LABEL PROVIDER TOKEN</code>	Create a service auth token.
<code>cf update-service-auth-token LABEL PROVIDER TOKEN</code>	Update a service auth token.
<code>cf delete-service-auth-token LABEL PROVIDER</code>	Delete a service auth token.
<code>cf service-brokers</code>	List service brokers.
<code>cf create-service-broker SERVICE_BROKER USERNAME PASSWORD URL</code>	Create a service broker.
<code>cf update-service-broker SERVICE_BROKER USERNAME PASSWORD URL</code>	Update a service broker.
<code>cf delete-service-broker SERVICE_BROKER</code>	Delete a service broker.
<code>cf rename-service-broker SERVICE_BROKER NEW_SERVICE_BROKER</code>	Rename a service broker.
<code>cf migrate-service-instances v1_SERVICE v1_PROVIDER v1_PLAN v2_SERVICE v2_PLAN</code>	Migrate service instances from one service plan to another.
<code>cf purge-service-offering SERVICE</code>	Recursively remove a service and child objects from Cloud Foundry database without making requests to a service broker.
<code>cf service-access</code>	List service access settings.
<code>cf enable-service-access SERVICE</code>	Enable access to a service or service plan for one or all orgs.
<code>cf disable-service-access SERVICE</code>	Disable access to a service or service plan for one or all orgs.

SECURITY GROUP

<code>cf security-groups</code>	List all security groups.
<code>cf security-group SECURITY_GROUP</code>	Show a single security group.
<code>cf create-security-group SECURITY_GROUP PATH_TO_JSON_RULES_FILE</code>	Create a security group.
<code>cf update-security-group SECURITY_GROUP PATH_TO_JSON_RULES_FILE</code>	Update a security group.
<code>cf delete-security-group SECURITY_GROUP</code>	Delete a security group.
<code>cf bind-security-group SECURITY_GROUP ORG SPACE</code>	Bind a security group to a space.
<code>cf unbind-security-group SECURITY_GROUP ORG SPACE</code>	Unbind a security group from a space.
<code>cf bind-staging-security-group SECURITY_GROUP</code>	Bind a security group to the list of security groups to be used for staging applications.
<code>cf staging-security-groups</code>	List security groups in the staging set for applications.
<code>cf unbind-staging-security-group SECURITY_GROUP</code>	Unbind a security group from the set of security groups for staging applications.
<code>cf bind-running-security-group SECURITY_GROUP</code>	Bind a security group to the list of security groups to be used for running applications.
<code>cf running-security-groups</code>	List security groups in the set of security groups for running applications.
<code>cf unbind-running-security-group SECURITY_GROUP</code>	Unbind a security group from the set of security groups for running applications.

ENVIRONMENT VARIABLE GROUPS

<code>cf running-environment-variable-group</code>	Retrieve the contents of the running environment variable group.
<code>cf staging-environment-variable-group</code>	Retrieve the contents of the staging environment variable group.
<code>cf set-staging-environment-variable-group '{"name": "value", "name": "value"}'</code>	Pass parameters as JSON to create a staging environment variable group.
<code>cf set-running-environment-variable-group '{"name": "value", "name": "value"}'</code>	Pass parameters as JSON to create a running environment variable group.

FEATURE FLAGS

<code>cf feature-flags</code>	Retrieve list of feature flags with status of each flag-able feature.
<code>cf feature-flag FEATURE_NAME</code>	Retrieve an individual feature flag with status.
<code>cf enable-feature-flag FEATURE_NAME</code>	Enable the use of a feature so that users have access to and can use the feature.
<code>cf disable-feature-flag FEATURE_NAME</code>	Disable the use of a feature so that users do not have access to and cannot use the feature.

ADVANCED

<code>cf curl PATH</code>	Executes a raw request, content-type set to application/json by default.
<code>cf config [OPTIONS]</code>	Write default values to the config.
<code>cf oauth-token</code>	Retrieve and display the OAuth token for the current session.

ADD/REMOVE PLUGIN REPOSITORY

<code>cf add-plugin-repo [REPO_NAME] [URL]</code>	Add a new plugin repository.
<code>cf remove-plugin-repo [REPO_NAME] [URL]</code>	Remove a plugin repository.
<code>cf list-plugin-repos</code>	List all the added plugin repositories.
<code>cf repo-plugins</code>	List all available plugins in all added repositories.

ADD/REMOVE PLUGIN

<code>cf plugins</code>	List all available plugin commands.
<code>cf install-plugin URL or LOCAL_PATH/TO/PLUGIN</code>	Install the plugin defined in command argument.
<code>cf uninstall-plugin PLUGIN-NAME</code>	Uninstall the plugin defined in command argument.

ABOUT THE AUTHORS



Jeremy Voorhis is a product manager, software engineer, sysadmin, OSS contributor, speaker and technology startup adviser with 10 years of experience.



Billy Tat is a web app developer with expertise in various languages and frameworks including Ruby on Rails, Python and Objective-C. He is currently Junior Technical Product Manager for ActiveState, where he researches open source technologies and collaborates with development, support and QA. Previously, Billy worked at a startup developing an iOS app for movie ticketing, showtimes and recommendations. In his spare time, Billy is an avid NBA fan and PC gamer.

RECOMMENDED BOOK



Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS)
Hardcover - Jan 28 2014
by Michael J. Kavis (Author)

BUY NOW



BROWSE OUR COLLECTION OF 250+ FREE RESOURCES, INCLUDING:

RESEARCH GUIDES: Unbiased insight from leading tech experts

REFCARDZ: Library of 200+ reference cards covering the latest tech topics

COMMUNITIES: Share links, author articles, and engage with other tech

JOIN NOW



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZONE IS A DEVELOPER'S DREAM," SAYS PC MAGAZINE.

Copyright © 2015 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513
888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1



9 781936 502776

VERSION 1.0 \$7.95