Cloud Foundry

BY JEREMY VOORHIS, REVISED AND UPDATED BY BILLY TAT

ABOUT CLOUD FOUNDRY

Cloud Foundry is an open-source, multi-cloud, multi-language application platform that supports continuous delivery.

Some of the features provided by Cloud Foundry include:

- User management
- Middleware and OS management
- Logging and metrics
- Services
- Health management
- Scaling for high availability of platforms and applications

The Cloud Foundry project is available as a collection of open-source repositories, available at github.com/cloudfoundry. You can 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

STACKS

A stack is the underlying filesystem image utilizing a given operating system used to run an application. Stacks are used in conjunction with buildpacks to provide the runtime environment for an application. When deploying Docker applications, a stack is not required.

Currently, Cloud Foundry has two stacks available: the cflinuxfs2 stack and the windows2012R2 stack. The cflinuxfs2 stack is based upon Ubuntu Trusty 14.04, while the windows2012R2 stack provides support for .NET applications. In addition to the provided stacks, Cloud Foundry also provides support for building and using your own custom stacks.

BUILDPACKS – FRAMEWORKS AND RUNTIMES

Buildpacks provide a framework from which you can use your favorite programming tools and environments when building applications. Cloud Foundry uses buildpacks as a way to encapsulate the rules for how to build, stage, and deploy your application to the cloud. For example, a Ruby on Rails application will have a file named database.yml, which describes how to connect to a database and may have a collection of assets, such as JavaScript, CSS, and images.

Runtimes encapsulate programming language environments, and because not all applications are compatible with every release of a programming language, multiple versions are supported. Buildpacks examine the developer-provided artifacts to decide the appropriate framework, runtime, and dependencies to download and install. These artifacts are also used to determine how to configure an application to interact with bound service dependencies (like PostgreSQL, Redis, RabbitMQ, etc.)

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

SYSTEM BUILDPACKS

Most common programming languages are supported for use.

NAME	SUPPORTED LANGUAGES AND FRAMEWORKS	GITHUB REPO
Binary	N/A	Binary source
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
Staticfile	HTML, CSS, or JavaScript	Staticfile source

If your application uses a language or framework that Cloud Foundry buildpacks do not support directly, you still have a few more options:

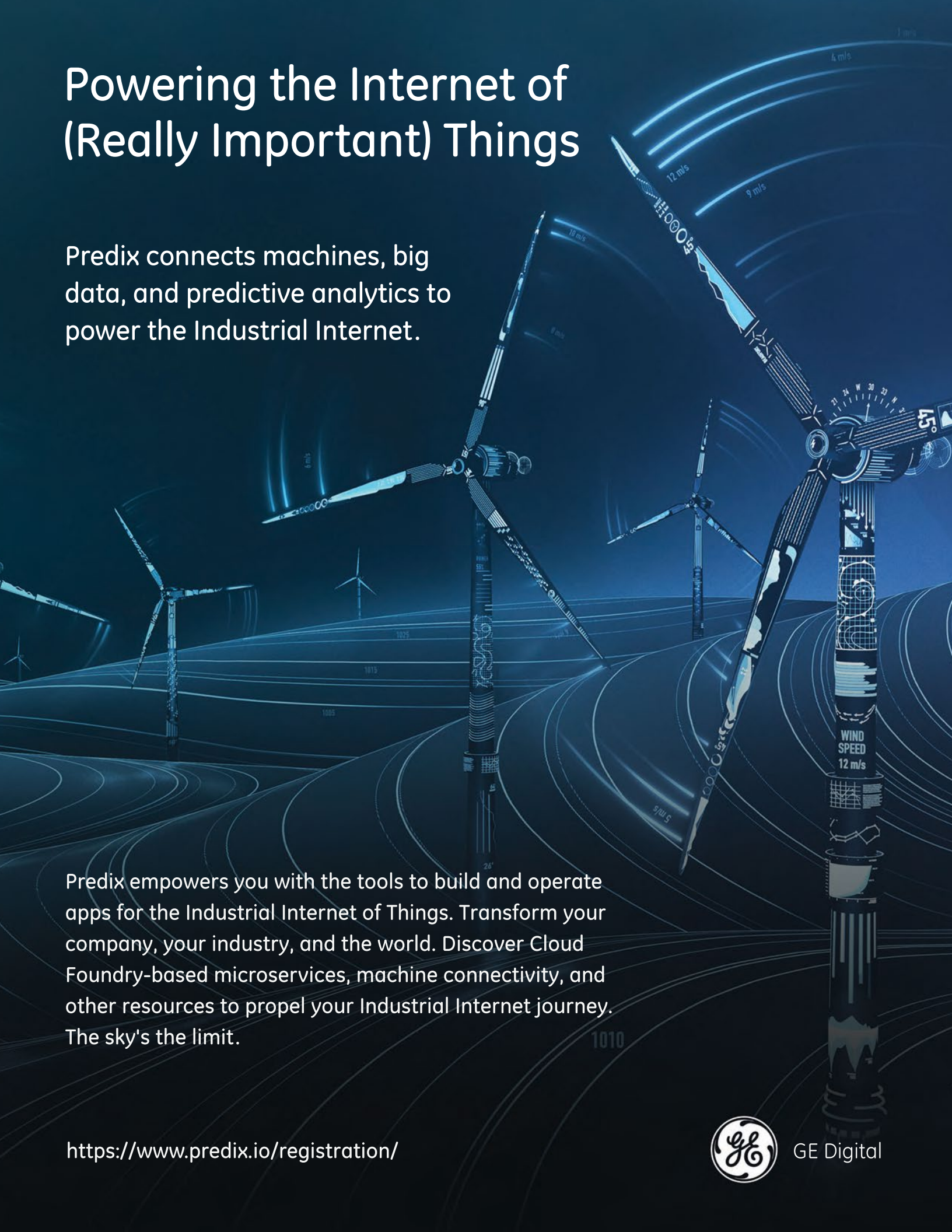
- Customize an existing buildpack
- Write your own buildpack
- Use a Cloud Foundry Community Buildpack
- Use a Heroku Third-Party Buildpack

DOCKER

In addition to buildpacks, Cloud Foundry provides support for running Docker images as applications. Docker allows developers

Powering the Internet of (Really Important) Things

Predix connects machines, big data, and predictive analytics to power the Industrial Internet.



Predix empowers you with the tools to build and operate apps for the Industrial Internet of Things. Transform your company, your industry, and the world. Discover Cloud Foundry-based microservices, machine connectivity, and other resources to propel your Industrial Internet journey. The sky's the limit.

https://www.predix.io/registration/

GE Digital

to bundle applications along with any dependencies into a lightweight and portable container. To push an application, use:

```
$ cf push app-name -o docker-registry-user/docker-image
```

For more information on using Docker in Cloud Foundry go to: docs.cloudfoundry.org/concepts/docker.html.

SERVICES

Cloud Foundry provides a marketplace—a collection of building blocks—from which developers can choose what they need and provision for use in their applications. This concept is called binding and includes services your app may use, such as MySQL, PostgreSQL, MongoDB, and RabbitMQ. While most apps may be simple front-end apps connected to a database service, a complex deployment may consist of many microservices covering front- and back-end requirements like messaging, logging, etc.

Various distributions offer additional supported services, and users can even add their own via the 'user-provided services' API or adding their own custom services.

COLLABORATION

To better support enterprises with a finite set of cloud resources, Cloud Foundry allows administrators to assign roles in a hierarchy of organization and spaces so that teams can keep applications within their capacity thresholds. In addition to managing user roles, administrators also manage organization and space quotas and the assignment of resources such as memory, service, and instance usage.

Developers are restricted to operating within the organizations and spaces their administrators have assigned them to. Within these assigned organizations and spaces, a developer shares resources specified in the quota with other developers in the same organization and space.

Read more about organizations and spaces here: docs.cloudfoundry.org/concepts/roles.html

CLOUD FOUNDRY PROVIDERS

Cloud Foundry is governed by the Cloud Foundry Foundation, which currently consists of over 60 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.

The Cloud Foundry Certification process requires certain components of the platform to be used as released by the foundation's (AP style says lowercase, but I'm not sure what our internal guide says) project teams, thus ensuring a level of standardization across distributions. Each year the foundation announces a set of requirements necessary to become Cloud Foundry Certified.

CERTIFIED PROVIDERS

- Cloud Foundry Foundation – cloudfoundry.org
- Atos Cloud Foundry – canopy-cloud.com/application-platforms/atos-cloud-foundry
- CenturyLink AppFog – ctl.io/appfog
- GE Predix – predix.io
- HPE Helion Stackato – stackato.com
- Huawei FusionStage – e.huawei.com/us/solutions/technical/cloud-computing
- IBM Bluemix – ibm.com/cloud-computing/bluemix
- Pivotal Cloud Foundry – pivotal.io/platform
- SAP HANA Cloud Platform – hcp.sap.com
- Swisscom Application Cloud – swisscom.ch/en/business/enterprise/offer/cloud-data-center-services/paas/application-cloud.html

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, you will need to sign up for an account at run.pivotal.io. However, these instructions are applicable to any other certified provider.

Download the Cloud Foundry CLI at github.com/cloudfoundry/cli/releases and follow the instructions for your operating system at docs.cloudfoundry.org/cf-cli/install-go-cli.html.

Test your new installation by opening a command line tool and running `cf -help`. You should see output containing information about the cf CLI and a listing of available commands along with their associated 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:

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 Cloud Foundry 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.)

Application autoscaling is currently in the incubating stages. Whether you are deploying to a host provider or hosting Cloud Foundry yourself, you can implement autoscaling by monitoring memory and CPU usage through the REST API or by scripting the Cloud Foundry 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 push` command, and send it enough requests to see both the successful case and the error. You can stream the application's logs and display only the STDERR entries with the `cf logs hello-refcardz -recent | grep ERR`. For a more detailed breakdown of logs and other log filtering options, refer to: docs.cloudfoundry.org/devguide/deploy-apps/streaming-logs.html

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 Cloud Foundry 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 might 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 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 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 deployment** 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.

You can script these steps and even run them via your continuous integration server. Some Cloud Foundry providers may also have functionality to simplify these steps.

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 as you would 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.

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:

```
// 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”
    }
}
```

```
// 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 the configuration with a single `cf 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 github.com/Stackato-Apps/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 github.com/cloudfoundry/cf-release.

BOSH is an open-source framework created by VMware’s Site Reliability Engineering Team to deploy distributed systems. It is an infrastructure-agnostic framework allowing it the ability to build on top of most IaaS providers, including AWS, OpenStack, Google Cloud Platform, Azure, etc. 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 github.com/cloudfoundry/bosh.

BOSH LITE

If you don’t want or need a full-fledged BOSH-managed deployment, i.e. 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 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: docs.cloudfoundry.org
- Mailing lists: lists.cloudfoundry.org/archives
- BOSH developer documentation: bosh.io/docs
- Unofficial Cloud Foundry wiki: github.com/cloudfoundry-community/cf-docs-contrib/wiki
- IRC: #cloudfoundry on irc.freenode.net
- Slack: cloudfoundry.slack.com

CF CLI CHEAT SHEET

GETTING STARTED

CF LOGIN	LOG USER IN.
`cf logout`	Log user out.
`cf passwd`	Change user password.
`cf target`	Set or view the targeted org or space.
`cf api [URL]`	Set or view target API URL.
`cf auth USERNAME PASSWORD`	Authenticate user non-interactively.

APPLICATIONS

CF APPS	LIST ALL APPS IN THE TARGET SPACE.
`cf app APP_NAME`	Display health and status for app.
`cf push APP_NAME`	Push a new app or sync changes to an existing app.
`Cf push APP_NAME -o user/docker_image_name`	Push a Docker image
`cf scale APP_NAME`	Change or view the instance count, disk space limit, and memory limit for an app.
`cf delete APP_NAME`	Delete an app.
`cf rename APP_NAME NEW_APP_NAME`	Rename an app.

CF APPS	LIST ALL APPS IN THE TARGET SPACE.
cf start APP_NAME	
cf stop APP_NAME	Stop an app.
cf restart APP_NAME	Restart an app.
cf restage APP_NAME	Restage an app.
cf restart-app-instance APP_NAME INDEX	Terminate the running application Instance at the given index and instantiate a new instance of the application with the same index.
cf events APP_NAME	Show recent app events.
cf files APP_NAME [PATH]	Print out a list of files in a directory or the contents of a specific file.
cf logs APP_NAME	Tail or show recent logs for an app.
cf env APP_NAME	Show all env variables for an app.
cf set-env APP_NAME ENV_VAR_NAME ENV_VAR_VALUE	Set an env variable for an app.
cf unset-env APP_NAME ENV_VAR_NAME	Remove an env variable.
cf stacks	List all stacks (a stack is a pre-built file system, including an operating system, that can run apps).
cf copy-source SOURCE-APP TARGET-APP	Make a copy of the app source code from one application to another. Unless overridden, the copy-source command will restart the application.
cf create-app-manifest APP	Create an app manifest for an app that has been pushed successfully.

SERVICES

CF MARKETPLACE	LIST AVAILABLE OFFERINGS IN THE MARKETPLACE.
cf services	List all service instances in the target space.
cf service SERVICE_INSTANCE	Show service instance info.
cf create-service SERVICE PLAN SERVICE_INSTANCE	Create a service instance.
cf update-service SERVICE_INSTANCE	Update a service instance.
cf delete-service SERVICE_INSTANCE	Delete a service instance.
cf rename-service SERVICE_INSTANCE NEW_SERVICE_INSTANCE	Rename a service instance.
cf bind-service APP_NAME SERVICE_INSTANCE	Bind a service instance to an app.
cf unbind-service APP_NAME SERVICE_INSTANCE	Unbind a service instance from an app.
cf create-user-provided-service SERVICE_INSTANCE	Make a user-provided service instance available to cf apps.
cf update-user-provided-service SERVICE_INSTANCE	Update user-provided service instance name value pairs.

ORGANIZATIONS (V2)

CF ORGS	LIST ALL ORGS.
cf org ORG	Show org info.
cf create-org ORG	Create an org.
cf delete-org ORG	Delete an org.
cf rename-org ORG NEW_ORG	Rename an org.

SPACES (V2)

CF SPACES	LIST ALL SPACES IN AN ORG.
cf space SPACE	Show space info.
cf create-space SPACE	Create a space.
cf delete-space SPACE	Delete a space.
cf rename-space SPACE NEW_SPACE	Rename a space.

DOMAINS (V2)

CF DOMAINS	LIST DOMAINS IN THE TARGET ORG.
cf create-domain ORG DOMAIN	Create a domain in an org for later use.
cf delete-domain DOMAIN	Delete a domain.
cf create-shared-domain DOMAIN	Create a domain that can be used by all orgs (admin-only).
cf delete-shared-domain DOMAIN	Delete a shared domain.

ROUTES (V2)

CF ROUTES	LIST ALL ROUTES IN THE CURRENT SPACE.
cf create-route SPACE DOMAIN	Create a URL route in a space for later use.
cf check-route HOST DOMAIN	Perform a simple check to determine whether a route currently exists or not.
cf map-route APP_NAME DOMAIN	Add a URL route to an app.
cf unmap-route APP_NAME DOMAIN	Remove a URL route from an app.
cf delete-route DOMAIN	Delete a route.
cf delete-orphaned-routes	Delete all orphaned routes (e.g.: those that are not mapped to an app).

BUILDPACKS

CF BUILDPACKS	LIST ALL BUILDPACKS.
cf create-buildpack BUILDPACK PATH POSITION	Create a buildpack.
cf update-buildpack BUILDPACK	Update a buildpack.
cf rename-buildpack BUILDPACK_NAME NEW_BUILDPACK_NAME	Rename a buildpack.
cf delete-buildpack BUILDPACK	Delete a buildpack.

USER ADMIN

CF CREATE-USER USERNAME PASSWORD	CREATE A NEW USER.
cf delete-user USERNAME	Delete a user.
cf org-users ORG	Show org users by role.
cf set-org-role USERNAME ORG ROLE	Assign an org role to a user.
cf unset-org-role USERNAME ORG ROLE	Remove an org role from a user.
cf space-users ORG SPACE	Show space users by role.
cf set-space-role USERNAME ORG SPACE ROLE	Assign a space role to a user.
cf unset-space-role USERNAME ORG SPACE ROLE	Remove a space role from a user.

ORG ADMIN

CF QUOTAS	LIST AVAILABLE USAGE QUOTAS.
cf quota QUOTA	Show quota info.
cf set-quota ORG QUOTA	Assign a quota to an org.
cf create-quota QUOTA	Define a new resource quota.
cf delete-quota QUOTA	Delete a quota.

SPACE ADMIN

CF SPACE-QUOTAS	LIST AVAILABLE SPACE RESOURCE QUOTAS.
cf space-quota SPACE_QUOTA_NAME	Show space quota info.
cf create-space-quota QUOTA	Define a new space resource quota.
cf update-space-quota SPACE-QUOTA-NAME	Update an existing space quota.
cf delete-space-quota SPACE-QUOTA-NAME	Delete a space quota definition and unassign the space quota from all spaces.
cf set-space-quota SPACE-NAME SPACE-QUOTA-NAME	Assign a space quota definition to a space.
cf unset-space-quota SPACE QUOTA	Unassign a quota from a space.

SERVICE ADMIN

CF SERVICE-AUTH-TOKENS	LIST SERVICE AUTH TOKENS.
cf create-service-auth-token LABEL PROVIDER TOKEN	Create a service auth token.
cf update-service-auth-token LABEL PROVIDER TOKEN	Update a service auth token.
cf delete-service-auth-token LABEL PROVIDER	Delete a service auth token.
cf service-brokers	List service brokers.
cf create-service-broker SERVICE_BROKER USERNAME PASSWORD URL	Create a service broker.

CF SERVICE-AUTH-TOKENS	LIST SERVICE AUTH TOKENS.
cf update-service-broker SERVICE_BROKER USERNAME PASSWORD URL	Update a service broker.
cf delete-service-broker SERVICE_BROKER	Delete a service broker.
cf rename-service-broker SERVICE_BROKER NEW_SERVICE_BROKER	Rename a service broker.
cf migrate-service-instances v1_SERVICE v1_PROVIDER v1_PLAN v2_SERVICE v2_PLAN	Migrate service instances from one service plan to another.
cf purge-service-offering SERVICE	Recursively remove a service and child objects from Cloud Foundry database without making requests to a service broker.
cf service-access	List service access settings.
cf enable-service-access SERVICE	Enable access to a service or service plan for one or all orgs.
cf disable-service-access SERVICE	Disable access to a service or service plan for one or all orgs.

SECURITY GROUP

CF SECURITY-GROUP SECURITY_GROUP	SHOW A SINGLE SECURITY GROUP.
cf security-groups	List all security groups.
cf create-security-group SECURITY_GROUP PATH_TO_JSON_RULES_FILE	Create a security group.
cf update-security-group SECURITY_GROUP PATH_TO_JSON_RULES_FILE	Update a security group.
cf delete-security-group SECURITY_GROUP	Delete a security group.
cf bind-security-group SECURITY_GROUP ORG SPACE	Bind a security group to a space.
cf unbind-security-group SECURITY_GROUP ORG SPACE	Unbind a security group from a space.
cf bind-staging-security-group SECURITY_GROUP	Bind a security group to the list of security groups to be used for staging applications.
cf staging-security-groups	List security groups in the staging set for applications.
cf unbind-staging-security-group SECURITY_GROUP	Unbind a security group from the set of security groups for staging applications.
cf bind-running-security-group SECURITY_GROUP	Bind a security group to the list of security groups to be used for running applications.
cf running-security-groups	List security groups in the set of security groups for running applications.
cf unbind-running-security-group SECURITY_GROUP	Unbind a security group from the set of security groups for running applications.

ENVIRONMENT VARIABLE GROUPS

CF RUNNING-ENVIRONMENT-VARIABLE-GROUP	RETRIEVE THE CONTENTS OF THE RUNNING ENVIRONMENT VARIABLE GROUP.
cf staging-environment-variable-group	Retrieve the contents of the staging environment variable group.
cf set-staging-environment-variable-group '{"name":"value","name":"value"}'	Pass parameters as JSON to create a staging environment variable group.
cf set-running-environment-variable-group '{"name":"value","name":"value"}'	Pass parameters as JSON to create a running environment variable group.

FEATURE FLAGS

CF FEATURE-FLAGS	RETRIEVE LIST OF FEATURE FLAGS WITH STATUS OF EACH FLAG-ABLE FEATURE.
cf feature-flag FEATURE_NAME	Retrieve an individual feature flag with status.
cf enable-feature-flag FEATURE_NAME	Enable the use of a feature so that users have access to and can use the feature.
cf disable-feature-flag FEATURE_NAME	Disable the use of a feature so that users have access to and can use the feature.

ADVANCED

CF CURL PATH	EXECUTES A RAW REQUEST, CONTENT-TYPE SET TO APPLICATION/JSON BY DEFAULT.
cf config [OPTIONS]	Write default values to the config.
cf oauth-token	Retrieve and display the OAuth token for the current session.

ADD/REMOVE PLUGIN REPOSITORY

CF ADD-PLUGIN-REPO [REPO_NAME] [URL]	ADD A NEW PLUGIN REPOSITORY.
cf remove-plugin-repo [REPO_NAME] [URL]	Remove a plugin repository.
cf list-plugin-repos	List all the added plugin repository.
cf repo-plugins	List all available plugins in all added repositories.

ADD/REMOVE PLUGIN

CF PLUGINS	LIST ALL AVAILABLE PLUGIN COMMANDS.
cf install-plugin URL or LOCAL-PATH/TO/PLUGIN	Install the plugin defined in the command argument.
cf uninstall-plugin PLUGIN-NAME	Uninstall the plugin defined in the command argument.

ABOUT THE AUTHOR

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 a Product Manager for Hewlett Packard Enterprise, where he researches open source technologies and collaborates with development, support and QA for HPE Helion Stackato, an enterprise distribution of Cloud Foundry. Previously, Billy worked at a startup developing an iOS app for movie ticketing and recommendations. In his spare time, Billy likes to hit the courts for a game of basketball.

REVIEWED BY: Jayson DeLancey, GE Digital

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
50795

BROUGHT TO YOU IN PARTNERSHIP WITH
GE Digital