
Study Guide
Cloud Foundry
Certified Developer
Linux Academy
Cloud Assessments

Contents

Getting Code to Production is Too Slow!

What are Some of the Problems?

- Development cycles are monolithic
- Knowledge gaps
- Manual processes
- Human error
- Lack of environments
- QA takes too long to test code
- Dev/Prod parity
- Organizational Policies

Waterfall

The Problems with Waterfall

- Software development is not manufacturing or construction
- It's ridged and inflexible
- It's obsolete
- Big Design Up Front (BDUF)
- Designs look feasible on paper
- Feedback loop comes at the end

Code Deployment

What are Some of the Problems?

- All changes are compiled for release
- Releases are monolithic in nature
- Large deploys carry a lot of risks
- Whose job is it to push the button?
- Deploying code is a manual process
- Outcomes are unpredictable
- Rollbacks are all or nothing
- Failed/delayed deploys clog the pipeline

Operations vs. Development vs. QA

What are Some of the Problems?

- Departments are siloed
- ”Throw it over the fence" mentality
- Operations take too long
- Who’s job is it?
- The Blame Game
- “Not my job” attitude
- Lack of resources
- Shadow IT

Monolithic Applications

What are Some of the Problems?

- The application is independent and self-contained
- Tightly coupled systems
- Bugs/issues have far-reaching ramifications
- No modularity
- All or nothing deploys

Agile

What is Agile?

- A set of values and principles
- Adaptive planning
- Evolutionary development
- Early delivery
- Continuous improvement
- Rapid and flexible response to change

The Manifesto for Agile Software Development

- Individuals and Interactions more than processes and tools
- Working Software more than comprehensive documentation
- Customer Collaboration more than contract negotiation
- Responding to Change more than following a plan

CI/CD

Continuous Integration/Continuous Delivery

- Code is always ready to deploy
- Integrating early and often
- Developers write tests
- Integration tests are run automatically
- Constant feedback
- Defects are easier to resolve
- Releasing software faster and more frequently

DevOps

What is DevOps?

- Software development (Dev) and software operation (Ops)
- Software engineering practice
- It is a movement and not about tools
- Strongly advocates automation and monitoring
- Aims to shorten development cycles
- Deploy and scale services independently
- Continuous refactoring
- Continuous delivery and deployment using microservices

The Benefits of DevOps

- Breaks down silos
- Shorter Development Cycle
- Increased Release Velocity
- Improved Defect Detection
- Reduced Deployment Failures and Rollbacks
- Reduced Time to Recover upon Failure

Microservices

What are Microservices?

- Variant of the service-oriented architecture
- Applications are loosely coupled services
- Services should be fine-grained
- Protocols should be lightweight
- Modularity
- Small autonomous teams
- Continuous refactoring
- Enable continuous integration and delivery

What is Traditional Infrastructure?

- In-house infrastructure
- Lives in your datacenter
- Bare metal
- Virtual servers
- You manage everything
- Care and feeding

What is IaaS?

What is Infrastructure as a Service?

- Datacenters need not apply
- Virtual machines and other resources as a service
- Infrastructure is elastic ephemeral
- Examples of IaaS:
    - Amazon Web Services
    - VMware products
    - Azure
    - Google Cloud

What is PaaS?

What is Platform as a Service?

- Does not replace all your IT infrastructure
- Provider supplies the infrastructure
- Virtual Machines need not apply
- Instead provides it for key services
- Focuses on creating and running applications
- Geared toward software development

Pivotal Cloud Foundry Overview

Pivotal Cloud Foundry Website:

```
http://run.pivotal.io/
```

Pivotal Cloud Foundry Login:

```
https://login.run.pivotal.io/login
```

What is Cloud Foundry

Cloud Foundry is an open source platform designed to run cloud-native applications. It's a polyglot platform that lets you deploy applications written in any language. Applications in Cloud Foundry are highly available and scalable.

Cloud Foundry:

- Originally developed by VMware
- Transferred to Pivotal Software, a joint venture by EMC, VMware, and General Electric
- Written in Ruby and Go
- Release in 2011
- Governed by Cloud Foundry Foundation
- Open source cloud-native platform (PaaS)
- IaaS platform agnostic
- Build, test, deploy your apps easily
- Available as either open source, commercial product, or a through hosting provider
- The CLI is supported on Linux, Mac, and Windows
- Support any language or framework by using buildpacks
- Supports docker images

Example of supported languages:

- Java
- Python
- Node.js
- Ruby
- Go
- .NET

Examples of supported IaaS platforms:

- VMware vSphere
- AWS
- Google Cloud Platform
- Azure
- OpenStack

Certified providers:

- Atos Canopy
- CenturyLink App Fog
- GE Predix
- HPE Helion Stackato 4.0
- Huawei FusionStage
- IBM Bluemix
- Pivotal Cloud Foundry
- SAP Cloud Platform
- Swisscom Application Cloud

Diego

Diego is the container management system for Cloud Foundry. It schedules and runs Tasks and Long-Running Processes (LRP):

- Tasks run once
- LRPs are web applications

Diego Brain

- Distribute Tasks and LRPs to Diego Cells
- Corrects discrepancies between Actual and Desired counts
- Ensure fault-tolerance and long-term consistency

Auctioneer

- Holds auctions for Tasks and LRP instances
- Distributes work using the auction algorithm
- Communicates over HTTPS with Diego Cells
- Uses Consul/Locket so only one auction at a time

Routes

The router routes incoming traffic to the appropriate component either a:

- Cloud Controller component
- Hosted application running on a Diego Cell

This is not a traditional router but a reverse proxy that centrally manages and routes incoming HTTP(S) traffic to the appropriate component. It periodically queries the Diego Bulletin Board System (BBS) to determine which cells and containers each application currently runs on.

Authentication

- All traffic needs to be authenticated. The UAA service manages:
  - Cloud Foundry Developers
  - Application client/end users
  - Applications requiring application-to-application integrations

- UAA is an OAuth2 authorization service that issues tokens for client apps when they act on behalf of the CF user
- Acts as a user identity store for a user's password

App Lifecycle

Cloud Controller

The Cloud Controller (CC) directs the deployment of applications. To push an app to Cloud Foundry, you target the Cloud Controller. The Cloud Controller then directs the Diego Brain through the CC-Bridge components to coordinate individual Diego cells to stage and run applications.

The Cloud Controller also maintain records of orgs, spaces, user roles, services, and more:

- Exposed using a RESTful API
- Users can use the API to:
  - Push, stage, run, update, and retrieve applications
  - Push, stage, run one-off tasks

nsync

nsync receives a message from the Cloud Controller when the user scales an app. It writes the number of instances into a DesiredLRP structure in the Diego BBS database.

BBS

BBS uses its convergence process to monitor the DesiredLRP and ActualLRP values. It launches or kills application instances as appropriate to ensure the ActualLRP count matches the DesiredLRP count.

Cell Rep

Cell Rep monitors the containers and provides the ActualLRP value.

App Storage and Execution

Blobstore

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

- Application code packages:
    - Buildpacks
    - Droplets

You can configure the blobstore as either an internal server or an external S3 or S3-compatible endpoint.

Diego Cell

Application instances, application tasks, and staging tasks all run as Garden containers on the Diego Cell VMs. The Diego cell rep component manages the lifecycle of those containers and the processes running in them, reports their status to the Diego BBS, and emits their logs and metrics to Loggregator.

Garden and runC

- Go API for creating and managing containers
- runC implements Open Container Initiative (OCI)

Services

Service Brokers

Applications typically depend on services such as databases or third-party SaaS providers. When a developer provisions and binds a service to an application, the service broker for that service is responsible for providing the service instance.

Messaging

Cloud Foundry component VMs communicate with each other internally through HTTP and HTTPS protocols, sharing temporary messages and data stored in two locations:

- Consul
- Bulletin Board System (BBS)

Consul

A Consul server stores longer-lived control data, such as component IP addresses and distributed locks that prevent components from duplicating actions.

Bulletin Board System (BBS)

Bulletin Board System (BBS) stores more frequently updated and disposable data such as cell and application status, unallocated work, and heartbeat messages. The BBS stores data in MySQL, using the Go MySQL Driver.

Metrics and Logging

Loggregator

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

Metrics Collector

The metrics collector gathers metrics and statistics from the components. Operators can use this information to monitor a Cloud Foundry deployment.

Diego

Diego is the container management system for Cloud Foundry. It schedules and runs Tasks and Long-Running Processes (LRP):

- Task run once
- LRPs are web applications

Long-Running Processes (LRP)

- Web applications
- Can have many instances
- App instances run in containers
- Containers are immutable
- Isolates CPU, memory, and disk
- Containers run on Cells

Cell

Application instances, application tasks, and staging tasks all run as Garden containers on the Diego Cell VMs. The Diego cell rep component manages the lifecycle of those containers and the processes running in them, reports their status to the Diego BBS, and emits their logs and metrics to Loggregator.

Cells:

- Are VMs
- Run application instances
- Run application tasks
- Use Garden for creating and managing containers

Garden

- Go API for creating and managing containers
- Has a pluggable back to support different platforms:
    - Supports Linux and Windows-based apps
- runC implements Open Container Initiative (OCI)

Cell Rep

- Cell Rep monitors the containers and provides the ActualLRP value
- Manages containers lifecycle
- Participates in auctions to accept new Tasks and LRP instances
- The processes running in them
- Reports statuses to Diego BBS
- Executor is a sub-process:
    - Manages allocations of containers to the cell
    - Forwards logs and metrics to Metron

Diego Brain

- Distribute Tasks and LRPs to Diego Cells
- Corrects discrepancies between Actual and Desired counts
- Ensure fault-tolerance and long-term consistency
- Auctioneer:
    - Holds auctions for Tasks and LRP instances
    - Distributes work using the auction algorithm
    - Communicates over HTTPS with Diego Cells
    - Uses Consul/Locket so only one auction at a time

Metron

Forwards app logs, errors, and metrics to the Loggregator.

Deigo Database

- Bulletin Board System (BBS):
  - Maintains a real-time representation of the state
  - Maintains a lock in Consul/Locket to ensure that only one BBS is active
  - Enforce the desired state of LRPs
  - Resends auction requests for Tasks
- MySQL
- Provides a consistent key-value data store to Diego

Loggerator

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

Doppler

- Gathers logs from the Metron Agents
- Stores them in temporary buffers
- Forwards them to:
  - Traffic Controller
  - Third-party syslog drains

Traffic Controller

- Handles client requests for logs
- Gathers and collates messages from all Doppler servers
- Provides external API
- Exposes the Firehose

Firehose

WebSocket endpoint that streams all the event data Data stream includes logs:

- Logs
- HTTP events
- Container metrics
- Cloud Foundry system components

Nozzles

- Nozzles are programs which consume data from the Loggregator Firehose
- Configured to select, buffer, and transform data
- Forward data to other applications and services
- Examples:
    - JMX Bridge
    - Datadog nozzle
    - Syslog nozzle

Cloud Controller

- Directs the deployment of applications
- Coordinate individual Diego cells to stage and run applications
- Exposed using a RESTful API
- Users can use the API to:
    - Push, stage, run, update, and retrieve applications
    - Push, stage, run one-off tasks

Cloud Controller Database

- Maintains records for the logical hierarchical structure
- Store orgs, spaces, services, user roles, and App data

Blobstore

- Is a repository for large binary files
- The blobstore contains the following:
  - Application packages
  - Buildpacks
  - Resource Cache
  - Buildpack Cache
  - Droplets

CC-Bridge

- Translate app-specific requests from the Cloud Controller to the BBS
- Components:
  - Stager
  - CC-Uploader
  - Nsync Bulker
  - Nsync Listener
  - TPS Listener
  - TPS Watcher

Routing

- Routes incoming traffic to a Cloud Controller component or a hosted application
- Is a reverse proxy that manages incoming HTTP/HTTPS traffic
- Queries the Diego Bulletin Board System where apps are currently running on

Gorouter

- Implemented in Go
- Gives the router full control over every connection
- Which simplifies support for WebSockets and other types of traffic
- A Single process contains all routing logic
- Removing unnecessary latency

Setting up your Pivotal Account

Creating a new user account at pivotal.io

1. Goto http://run.pivoal.io/ and click “SIGN UP FOR FREE”.
2. Fill out all of the required fields and “Click Sign Up”.
3. Click on the link sent you after registration.
4. Click Pivotal Web services
5. You only need to check the Terms of Service then click “Next: Claim Your Trial”.
6. Enter your mobile number and click “Send me my code”.
7. Enter your code and click “Submit”.
8. Enter the org and click “Start Free Trial”.
9. Your registration is complete.

Installing Cloud Foundry CLI

Installing the Cloud Foundry CLI on CentOS

```
sudo wget -O /etc/yum.repos.d/cloudfoundry-cli.repo https://
packages.cloudfoundry.org/fedora/cloudfoundry-cli.repo
sudo yum install cf-cli -y
```

Installing the Cloud Foundry CLI on Ubuntu

```
wget -q -O - https://packages.cloudfoundry.org/debian/cli.
cloudfoundry.org.key | sudo apt-key add -
echo "deb http://packages.cloudfoundry.org/debian stable main" |
sudo tee /etc/apt/sources.list.d/cloudfoundry-cli.list
apt-get update
apt-get install cf-cli
```

Logging into Cloud Foundry via the CLI

```
cf login -a https://api.run.pivotal.io -u <USERNAME> -p <PASSWORD>
```

Pushing Your First App

Pushing your App

```
cf push <APP_NAME> --random-route
```

Lists your Apps

```
cf apps
```

Getting more details on your App

```
cf app <APP_NAME>
```

```
cf --help
```

Orgs

An org is a development account that an individual or multiple collaborators can own and use. All collaborators access an org with user accounts. Collaborators in an org share a resource quota plan, applications, services availability, and custom domains.

By default, an org has the status of active. An admin can set the status of an org to suspended for various reasons such as failure to provide payment or misuse. When an org is suspended, users cannot perform certain activities within the org, such as push apps, modify spaces, or bind services.

Commands

Listing Orgs

NAME: orgs - List all orgs USAGE: cf orgs ALIAS: o

Example:

```
cf orgs
```

Show Users in an Org

NAME: org-users - Show org users by role USAGE: cf org-users ORG OPTIONS: -a List all users in the org

Example:

```
cf org-users tthomen-la-org
```

User Accounts

A user account represents an individual person within the context of a CF installation. A user can have different roles in different spaces within an org, governing what level, and type of access they have within that space.

An org role needs to be assigned to a user before you can assign a space role.

Commands

Creating a User Account

NAME: create-user - Create a new user USAGE: cf create-user USERNAME PASSWORD cf create-user USERNAME –origin ORIGIN OPTIONS: –origin Origin for mapping a user account to a user in an external identity provider.

Example:

```
cf create-user travis@linuxacademy.com password123456
```

Spaces

Every application and service is scoped to a space. Each org contains at least one space. A space provides users with access to a shared location for application development, deployment, and maintenance. Each space role applies only to a particular space.

Commands

List Spaces

NAME: spaces - List all spaces in an org USAGE: cf spaces.

Example:

```
cf spaces
```

Creating a New Space

NAME: create-space - Create a space USAGE: cf create-space SPACE [-o ORG] [-q SPACE_QUOTA] OPTIONS: -o Organization -q Quota to assign to the newly created space.

Example:

```
cf create-space dev -o tthomen-la-org -q
```

Deleting a Space

NAME: delete-space - Delete a space USAGE: cf delete-space SPACE [-o ORG] [-f] OPTIONS: -f Force deletion without confirmation -o Delete space within specified org.

Example:

```
cf delete-space dev -o tthomen-la-org -f
```

List Users in a Given Space

NAME: space-users - Show space users by role USAGE: cf space-users ORG SPACE.

Example:

```
cf space-users dev tthomen-la-org
```

Changing Orgs and Spaces

NAME: target - Set or view the targeted org or space USAGE: cf target [-o ORG] [-s SPACE] ALIAS: t OPTIONS: -o Organization -s Space.

Example:

```
cf target -s dev
```

Roles and Permissions

A user can have one or more roles. The combination of these roles defines the user's overall permissions in the org and within specific spaces in that org.

Org Roles:

- Org Managers are managers or other users who need to administer the org:
  - Create and manage permissions at the Org and Space level
  - Add private domains
  - Create and manage Spaces
- Org Billing Managers create and manage billing account and payment information:
  - View users and their roles in an Org
  - View an Org's quotas

    - View applications instances, application statuses, services instances and application/service bindings

- Org Auditors view but cannot edit user information and org quota usage information:
    - View users and their roles in an Org
    - View an Org's quotas

Space Roles:

- Space Managers are managers or other users who administer a space within an org:
    - Invite and manage users at a Space level
    - View Org quotas plans
    - View and edit space
    - View applications instances, application statuses, services instances and application/service bindings
- Space Developers are application developers or other users who manage applications and services in a space:
    - View all the Space users and their roles
    - View Org quotas plans
    - View all Spaces under an Org
    - View applications instances, application statuses, services instances and application/service bindings
    - Deploy, run, and manage applications
    - Create and bind services to an application
    - Assign routes, scale applications instances, memory allocation, add disk limits
    - Rename applications
- Space Auditors view but cannot edit the space:
    - View users and roles

- View Org quotas plans
- View all of the spaces in an Org
- View the status, number of instances, services bindings, and resources of an application

Commands

Adding an Org Role to a User

NAME: set-org-role - Assign an org role to a user USAGE: cf set-org-role USERNAME ORG ROLE ROLES: ‘OrgManager’ - Invite and manage users, select and change plans, and set spending limits ‘BillingManager’ - Create and manage the billing account and payment info ‘OrgAuditor’ - Read-only access to org info and reports.

Example:

```
cf set-org-role travis@linuxacademy.com linux-academy OrgManager
```

Removing a User from an Org Role

NAME: unset-org-role - Remove an org role from a user USAGE: cf unset-org-role USERNAME ORG ROLE ROLES: ‘OrgManager’ - Invite and manage users, select and change plans, and set spending limits ‘BillingManager’ - Create and manage the billing account and payment info ‘OrgAuditor’ - Read-only access to org info and reports.

Example:

```
cf unset-org-role travis@linuxacademy.com linux-academy OrgManager
```

Assign a Space Role to a User

NAME: set-space-role - Assign a space role to a user USAGE: cf set-space-role USERNAME ORG SPACE ROLE ROLES: ‘SpaceManager’ - Invite and manage users, and enable features for a given space ‘SpaceDeveloper’ - Create and manage apps and services, and see logs and reports ‘SpaceAuditor’ - View logs, reports, and settings in this space.

Example:

```
cf set-space-role travis@linuxacademy.com linux-academy dev
SpaceDeveloper
```

Remove a Space Role from a User

NAME: unset-space-role - Remove a space role from a user USAGE: cf unset-space-role USERNAME ORG SPACE ROLE ROLES: 'SpaceManager' - Invite and manage users, and enable features for a given space 'SpaceDeveloper' - Create and manage apps and services, and see logs and reports 'SpaceAuditor' - View logs, reports, and settings in this space.

Example:

```
cf unset-space-role travis@linuxacademy.com linux-academy dev
SpaceDeveloper
```

Manifest Basics

```
---
applications:
- name: my-app
  memory: 512M
  instances: 2
```

Specifying a manifest file:

```
cf push my-app -f path_to_manifests
```

buildpack: You can use the buildpack attribute to specify it:

- By name: MY-BUILDPACK
- By GitHub URL: https://github.com/cloudfoundry/java-buildpack.git
- By GitHub URL with a branch or tag: https://github.com/cloudfoundry/java-buildpack.git#v3.3.0 for the v3.3.0 tag

```
---
  ...
  buildpack: buildpack_URL
```

command: Specify the custom start command in your application manifest.

```
---
  ...
  command: bundle exec rake VERBOSE=true
```

disk_quota: Disk_quota attribute to allocate the disk space for your app instance This attribute requires a unit of measurement: M, MB, G, or GB, in upper case or lower case.

```
---
  ...
  disk_quota: 1024M
```

docker: If you are using a Docker image, then you may use the docker attribute to specify it and an optional Docker username.

```
---
  ...
  docker:
    image: docker-image-repository/docker-image-name
    username: docker-user-name
```

memory: Memory attribute to specify the memory limit for all instances of an app. Unit of measurement: M, MB, G, or GB, in upper case or lower case

```
---
  ...
  memory: 1024M
```

no-route: You can use the no-route attribute with a value of true.

```
---
  ...
  no-route: true
```

path: Path attribute to tell Cloud Foundry the directory location where it can find your application.

```
---
  ...
  path: /path/to/application/bits
```

random-route: You can use random-route to randomize routes declared with the route attribute in the app manifest.

```
---
  ...
  random-route: true
```

routes: Routes attribute to provide multiple HTTP and TCP routes. Each route for this app is created if it does not already exist.

```
---
   ...
   routes:
   - route: example.com
   - route: www.example.com/foo
   - route: tcp-example.com:1234
```

timeout:

```
---
   ...
   timeout: 80
```

Services: Services block consists of a heading, then one or more service instance names.

```
---
   ...
   services:
    - instance_ABC
    - instance_XYZ
```

Using Environment Variables In A Manifest

The env block consists of a heading, then one or more environment variable/value pairs.

```
---
   ...
   env:
     RAILS_ENV: production
     RACK_ENV: production
```

Application Management on Cloud Foundry

Application Container Lifecycle

Deployment

- Application deployment process:
  - Uploading
  - Staging
  - Starting
- The default time limits for the phases are as follows:
  - Upload: 15 minutes
  - Stage: 15 minutes
  - Start: 60 seconds
- The time limit for starting apps through an application manifest

Crash Events

- CF automatically restarts it by rescheduling the instance on another container three times
- CF waits thirty seconds before attempting another restart
- The wait time doubles each restart until the ninth restart
- This duration until the 200th restart
- CF stops trying to restart the app instance after the 200th restart

Evacuation

- CF automatically relocates the instances on VMs that are shutting down
- CF recreates the app instances on another VM
- Once healthy, the old instances are then shut down
- App instances may be in a duplicated state for a brief period

Shutdown

- CF sends the app process in the container a SIGTERM
- The process has ten seconds to shut down gracefully
- CF sends a SIGKILL after 10 seconds

Starting, Restarting and Restaging

How Applications are Started

Applications can be started by:

- The -c flag at the command line
- The command attribute in a manifest
- The buildpack

A null can be used to force the use of the buildpack start command.

Pushing an app:

```
cf push <YOUR-APP>
```

Pushing an app with a start command:

```
cf push <YOUR-APP> -c "node <YOUR-APP>.js"
```

Pushing an app and forcing the use of a buildpack start command:

```
cf push <YOUR-APP> -c "null"
```

Restarting an Application

- Command: cf restart YOUR-APP
- Stops your application
- Restarts the application using the current droplet
- The Diego cell unpacks, compiles, and runs a droplet on a container
- A restart refreshes the application's environment
- Changes to environment variables consumed by the buildpack need to be restaged

Restarting:

```
cf restart <YOUR-APP>
```

Restage an Application

- Command: cf restage YOUR-APP
- Stops your application and restages it
- Compiles a new droplet and starts it
- Environment variable used by the buildpack
- Application source is not updated

Restaging:

```
cf restage <YOUR-APP>
```

Running Tasks

In contrast to a long-running process (LRP), tasks run for a finite amount of time, then stop. Tasks run in their own containers and are designed to use minimal resources. After a task runs, Cloud Foundry destroys the container running the task.

As a single-use object, a task can be checked for its state and for a success or failure message.

Use Cases for Tasks

- Migrating a database
- Sending an email
- Running a batch job
- Running a data processing script
- Processing images
- Optimizing a search index
- Uploading data
- Backing-up data
- Downloading content

How Tasks are Run

- Tasks are always executed asynchronously
- They run independently
- Task life-cycle:
    - A user initiates a task in Cloud Foundry:
        - cf run-task APPNAME “TASK”
        - Cloud Controller v3 API call
        - Cloud Foundry Java Client

- Cloud Foundry creates a container specifically for the task
- Cloud Foundry runs the task on the container using the value passed
- Cloud Foundry destroys the container
- The container also inherits environment variables, service bindings, and security groups

Running a task from the command line:

```
cf run-task <APPNAME> "TASK"
```

Task Logging

- Any data or messages the task outputs to STDOUT or STDERR
- The task execution history is retained for one month

Run a Task on an Application

- Push your application
- Run your task on the deployed application

Push your application and run a task:

```
cf push <APP-NAME>
cf run-task <APP-NAME> "TASK" --name <TASK-NAME>
```

Accessing Apps

SSH Access Control Hierarchy

Operators, space managers, and space developers can configure SSH access for Cloud Foundry, spaces, and apps:

- Operator: Entire deployment, configure the deployment to allow or prohibit SSH access
- Space Manager: Space, cf CLI allow-space-ssh, and disallow-space-ssh commands
- Space Developer: Application, cf CLI enable-ssh, and disable-ssh commands

Configuring SSH Access at the Application Level

cf enable-ssh enables SSH access to all instances of an app:

```
cf enable-ssh <MY-AWESOME-APP>
```

cf disable-ssh disables SSH access to all instances of an app:

```
cf disable-ssh <MY-AWESOME-APP>
```

Configuring SSH Access at the Space Level

cf allow-space-ssh allows SSH access into all apps in a space:

```
cf allow-space-ssh <SPACE-NAME>
```

cf disallow-space-ssh disallows SSH access into all apps in a space:

```
cf disallow-space-ssh <SPACE-NAME>
```

Checking SSH Permissions

cf ssh-enabled checks whether an app is accessible with SSH:

```
cf ssh-enabled <MY-AWESOME-APP>
```

```
cf ssh <MY-AWESOME-APP>
```

What Are Buildpacks?

List all buildpacks:

```
cf buildpacks
```

Push a new app or sync changes to an existing app:

```
cf push <APP_NAME> -b <BUILDPACK_NAME>
cf push <APP_NAME> -b <BUILDPACK_URL>
```

Create a buildpack:

```
cf create-buildpack <BUILDPACK>
```

Community Buildpacks

Cloud Foundry Community Buildpacks: https://github.com/cloudfoundry-community/cf-docs-contrib/wiki/Buildpacks

Managing Routes

List all routes in the current space or the current organization:

```
cf routes
```

Create a url route in a space for later use:

```
cf create-route <SPACE> <DOMAIN> --hostname <HOSTNAME>
```

Delete a route:

```
cf delete-route <DOMAIN> --<hostname HOSTNAME>
```

Add a url route to an app:

```
cf map-route <APP_NAME> <DOMAIN> --hostname <HOSTNAME>
```

Remove a url route from an app:

```
cf unmap-route <APP_NAME> <DOMAIN> --hostname <HOSTNAME>
```

Private Domains

List domains in the target org:

```
cf domains
```

Create a domain in an org for later use:

```
cf create-domain <ORG> <DOMAIN>
```

Delete a domain:

```
cf delete-domain <DOMAIN>
```

Using SSL With Your App

Create a SSL certificate:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem
-days 365 -nodes -subj "/C=US/ST=<STATE>/L=<CITY>/O=<COMPANY>/
OU=<DEPARTMENT>/CN=*.cfapps.io"
```

Create a manifest that contains the SSL certificates:

```
properties:
  router:
    tls_pem:
    - cert_chain: |
        -----BEGIN CERTIFICATE-----
        SSL_CERTIFICATE_SIGNED_BY_PRIVATE_KEY
        -----END CERTIFICATE-----
      private_key: |
        -----BEGIN EXAMPLE RSA PRIVATE KEY-----
        RSA_PRIVATE_KEY
        -----END EXAMPLE RSA PRIVATE KEY-----
    enable_ssl: true

applications:
- name: <APP_NAME>
  buildpack: staticfile_buildpack
  disk_quota: 512M
  instances: 1
  memory: 512M
```

HTTP Routing

HTTP Headers

HTTP traffic passed from Gorouter to an app includes the following HTTP headers:

- X-Forwarded-Proto
- X-Forwarded-For

If your load balancer terminates TLS on the client side of Gorouter, it must append these headers to requests forwarded to Gorouter.

X-Forwarded-Proto

Gives the scheme of the HTTP request from the client. The scheme is HTTP if the client made an insecure request or HTTPS if the client made a secure request. Developers can configure their apps to reject insecure requests by inspecting the HTTP headers of incoming traffic and rejecting traffic that includes X-Forwarded-Proto with the scheme of HTTP.

X-Forwarded-For

Gives the IP address of the client originating the request.

HTTP Headers for App Instance Routing

Developers who want to obtain debug data for a specific instance of an app can use the HTTP header X-CF-APP-INSTANCE to request an app instance.

Perform the following steps to make an HTTP request to a specific app instance:

```
cf app YOUR-APP --guid
cf app YOUR-APP
  curl app.example.com -H "X-CF-APP-INSTANCE":"YOUR-APP-GUID:YOUR-
INSTANCE-INDEX"
```

What is Blue Green Deployment?

Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.

At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live, and Green is idle.

As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

This technique can eliminate downtime due to application deployment. In addition, blue-green deployment reduces risk: if something unexpected happens with your new version on Green, you can immediately roll back to the last version by switching back to Blue.

Application Design for the Cloud

What are Cloud-Native Applications?

Cloud-native is an approach to building and running applications that fully exploit the advantages of the cloud computing delivery model. Cloud-native is about how applications are created and deployed, not where.

A cloud-native organization has the following characteristics:

- Microservices
- Continuous Delivery
- DevOps Culture

The Twelve Factors

I. Codebase One codebase tracked in revision control, many deploys.

II. Dependencies Explicitly declare and isolate dependencies.

III. Config Store config in the environment.

IV. Backing services Treat backing services as attached resources.

V. Build, release, run Strictly separate build and run stages.

VI. Processes Execute the app as one or more stateless processes.

VII. Port binding Export services via port binding.

VIII. Concurrency Scale-out via the process model.

IX. Disposability Maximize robustness with fast startup and graceful shutdown.

X. Dev/prod parity Keep development, staging, and production as similar as possible.

XI. Logs Treat logs as event streams.

XII. Admin processes Run admin/management tasks as one-off processes.

Building Node.JS Apps

Package.json and Application Bundling

- Package.json is required
- Cloud Foundry runs npm install for you

Example package.json file:

```
{
  "name": "first",
  "version": "0.0.1",
  "author": "Demo",
  "dependencies":  {
    "express": "3.4.8",
    "consolidate": "0.10.0",
    "swig": "1.3.2" },
  "engines": {
    "node": "0.12.7",
    "npm": "2.7.4"
  }
}
```

Specify the version of Node.JS in the Engine Node

```
{
  "engines": {
    "node": "0.12.7",
    "npm": "2.7.4"
  }
}
```

Application Port

- You must use the PORT environment variable
- User port 3000 for your local environment

Use process.env.PORT to Specify the port from Cloud Foundry:

```
app.listen(process.env.PORT || 3000);
```

Low Memory

- If instances are occasionally restarted due to memory constraints
- Node does not know how much memory it is allowed to use
- This sets max_old_space_size based on the available memory in the instance

```
cf set-env  OPTIMIZE_MEMORY true
```

Starting A Node.JS Application

- Node.JS apps require a start command
- Manifest: command: node my-app.js
- Cf push: cf push my-app -c “node my-app.js”
- Package.json: scripts node

```
 { “scripts”: { “postinstall”: “bower install”,
“start”: “node ./bin/www” } }
```

Environment Variables

- Access env vars by using process.env
- VCAP Servies: process.env.VCAP_SERVICES
- Node Buildpack Environment Variables
  - BUILD_DIR
  - CACHE_DIR
  - PATH

Bind Services

- VCAP Servies: process.env.VCAP_SERVICES
- The cfenv module provides default values when running locally
  - https://www.npmjs.com/package/cfenv
- Connecting to a service:
  - RabbitMQ: amqp module
  - Mongo: mongodb or mongoose module
  - MySQL: mysql module
  - Postgres: pg module
  - Redis: redis module
- Module decencies need to be in package.json

Building Ruby Apps

Application Bundling

- Gemfile and Gemfile.lock is needed
- Specify ruby version in Gemfile: ruby '2.4.3'
- Rack and Sinatra, you must have a config.ru

Asset Precompilation

- Cloud Foundry supports the Rails asset pipeline
- You do not precompile assets before deploying your app
- Precompiling before deploying reduces the time it takes to stage an app
- Command: rake assets:precompile
- To prevent reinitialization during precompilation add:

- application.rb

```
config.assets.initialize_on_precompile = false
```

- You can use live compilation mode by adding: application.rb

```
Rails.application.config.assets.compile = true
```

Environment Variables

- Access env vars by using ENV[]
- Ruby Buildpack Environment Variables:
  - BUNDLE_BIN_PATH
  - BUNDLE_GEMFILE
  - BUNDLE_WITHOUT
  - DATABASE_URL
  - GEM_HOME
  - GEM_PATH
  - RACK_ENV
  - RAILS_ENV
  - RUBYOPT
- VCAP Servies: ENV['VCAP_SERVICES']

Bind Services

- The cf-app-utils gem
- VCAP_SERVICES defines DATABASE_URL environment variable
- Non-Rails Apps use can use DATABASE_URL
- vcap_services = JSON.parse(ENV['VCAP_SERVICES'])

Building Java Apps

- Supported artifact types:
  - Grails
  - Groovy
  - Java
  - Play Framework
  - Spring Boot
  - Servlet
- Java apps should run without changes to the buildpack
- Fork the buildpack

Grails

Grails packages applications into WAR files for deployment into a Servlet container. To build the WAR file and deploy it, run the following:

```
grails prod war
cf push my-application -p target/my-application-version.war
```

Groovy

Groovy applications based on both Ratpack and a simple collection of files are supported.

Ratpack

Ratpack packages applications into two different styles; Cloud Foundry supports the distZip style. To build the ZIP and deploy it, run the following:

```
gradle distZip
cf push my-application -p build/distributions/my-application.zip
```

Raw Groovy

Groovy applications that are made up of a single entry point plus any supporting files can be run without any other work.

To deploy them, run the following:

```
cf push my-application
```

Java Main

Java applications with a main() method can be run provided that they are packaged as self-executable JARs.

Maven

A Maven build can create a self-executable JAR. To build and deploy the JAR, run the following:

```
mvn package
cf push my-application -p target/my-application-version.jar
```

Gradle

A Gradle build can create a self-executable JAR. To build and deploy the JAR, run the following:

```
gradle build
cf push my-application -p build/libs/my-application-version.jar
```

Play Framework

The Play Framework packages applications into two different styles. Cloud Foundry supports both the staged and dist styles.

To build the dist style and deploy it, run the following:

```
play dist
cf push my-application -p target/universal/my-application-version.
zip
```

Spring Boot CLI

Spring Boot can run applications comprised entirely of POGOs. To deploy then, run the following:

```
spring grab *.groovy
cf push my-application
```

Servlet

Java applications can be packaged as Servlet applications.

Maven

A Maven build can create a Servlet WAR. To build and deploy the WAR, run the following:

```
mvn package
cf push my-application -p target/my-application-version.war
```

Gradle

A Gradle build can create a Servlet WAR. To build and deploy the JAR, run the following:

```
gradle build
cf push my-application -p build/libs/my-application-version.war
```

Java and Grails Best Practices

Provide JDBC driver

The Java buildpack does not bundle a JDBC driver with your application. If your application will access a SQL RDBMS, include the appropriate driver in your application.

Allocate Sufficient Memory

If you do not allocate sufficient memory to a Java application when you deploy it, it may fail to start, or Cloud Foundry may terminate it. You must allocate enough memory to allow for the following:

- Java heap
- Metaspace, if using Java 8
- PermGen, if using Java 7 or earlier
- Stack size per Thread
- JVM overhead

Extension

The Java Buildpack is also designed to be easily extended. It creates abstractions for three types of components (containers, frameworks, and JREs) in order to allow users to easily add functionality.

Environment Variables

You can access environments variable programmatically.

For example, you can obtain VCAP_SERVICES as follows:

```
System.getenv("VCAP_SERVICES");
```

Deploying Out a Node.JS Microservice

Clone the User API

```
git clone https://github.com/linuxacademy/content-user-api.git
```

Clone the User GUI

```
git clone https://github.com/linuxacademy/content-user-gui.git
```

Container Management within Cloud Foundry

Scaling Your Apps

Factors such as user load, or the number and nature of tasks performed by an application, can change the disk space and memory the application uses.

Scaling Horizontally

Horizontally scaling an application creates or destroys instances of your application.

Incoming requests to your application are automatically load balanced across all instances of your application, and each instance handles tasks in parallel with every other instance. Adding more instances allows your application to handle increased traffic and demand.

```
cf scale APP_NAME -i INSTANCES
```

Scaling Vertically

Vertically scaling an application changes the disk space limit or memory limit that Cloud Foundry applies to all instances of the application.

```
cf scale APP_NAME -k DISK
cf scale APP_NAME -m MEMORY
```

Working with Services in Cloud Foundry

Services Overview

Services

- The resources provisioned out are service instances
- Services are provisioned from the marketplace
- Examples of services include:
  - Databases
  - Shared or dedicated server
  - Accounts on a SaaS application
- Service as a factory that delivers service instances

User-Provided Service Instances

- User-provided service instances are created by using your applications
- Once created, they behave like service instances created through the marketplace
- They are managed like service instances:
  - Listing
  - Renaming
  - Deleting
  - Binding
  - Unbinding

Service Commands

cf marketplace: List available offerings in the marketplace:

```
cf marketplace
```

cf services: List all service instances in the target space:

```
cf services
```

cf service: Show service instance info:

```
cf service <SERVICE_INSTANCE>
```

cf create-service: Create a service instance:

```
cf create-service
```

cf bind-service: Bind a service instance to an app:

```
cf bind-service <APP_NAME> <SERVICE_INSTANCE>
```

cf unbind-service: Unbind a service instance from an app:

```
cf unbind-service <APP_NAME> <SERVICE_INSTANCE>
```

cf delete-service: Delete a service instance:

```
cf delete-service <SERVICE_INSTANCE>
```

Services

Clone Spring Music:

```
git clone https://github.com/linuxacademy/content-spring-music.git
cd content-spring-music/
```

Build Spring Music with MySQL:

```
./gradlew clean assemble
java -jar -Dspring.profiles.active=mysql build/libs/spring-music.
jar
cf push
```

Create a MySQL service:

```
cf create-service cleardb spark spring-music-db-service-<USERNAME>


cf bind-service spring-music spring-music-db-service-<USERNAME>
```

Setting up a Mongo Service

Create the service:

```
cf create-service mlab sandbox user-mongodb-<USERNAME>
cf services
```

Clone the API:

```
git clone https://github.com/linuxacademy/content-user-api.git
cd content-user-api/
git checkout mongodb
```

Edit manifest.yml and change application name from user-api-xxx to user-api-. Next change user-mongodb-xxx to the service you created. For example: user-mongodb-.

Push application:

```
cf push
```

Add a record:

```
curl -H "Content-Type: application/json" -X POST -d '{ "first_
name": "Bob", "last_name": "Dobs", "username": "bdobs", "email":
"bdobs@example.com" }' http://user-api-xxx.cfapps.io/users
```

Clone the GUI:

```
git clone https://github.com/linuxacademy/content-user-gui.git
cd content-user-gui
```

Edit manifest.yml and change application name from user-gui-xxx to user-gui-. Next change API_URL from user-api-xxx.cfapps.io your API application name. For example: user-api-.cfapps.io.

Push application:

```
cf push
```

User-Provided Service

Create service out of user-api:

```
cf app user-api-<USERNAME>
cf create-user-provided-service user-api-service -p uri
http://<URI>
```

Bind service to app:

```
cf bind-service user-gui-<USERNAME> user-api-service
```

Route Services Overview

Cloud Foundry application developers may wish to apply transformation or processing to requests before they reach an application. Common examples of use cases include authentication, rate limiting, and caching services. Route Services are a kind of Marketplace Service that developers can use to apply various transformations to application requests by binding an application's route to a service instance. Through integrations with service brokers and, optionally, with the Cloud Foundry routing tier, providers can offer these services to developers with a familiar, automated, self-service, and on-demand user experience.

How It Works

Binding a service instance to a route associates the route_service_url with the route in the CF router. All requests for the route are proxied to the URL specified by route_service_url.

Once a route service completes its function it may choose to forward the request to the originally requested URL or to another location, or it may choose to reject the request; rejected requests will be returned to the originating requestor. The CF router includes a header that provides the originally requested URL, as well as two headers that are used by the router itself to validate the request sent by the route service.

Headers

The following HTTP headers are added by the Gorouter to requests forwarded to route services.

X-CF-Forwarded-Url

The X-CF-Forwarded-Url header contains the originally requested URL. The route service may choose to forward the request to this URL or to another.

X-CF-Proxy-Signature

The X-CF-Proxy-Signature header contains an encrypted value which only the Gorouter can decrypt.

The header contains the originally requested URL and a timestamp. When this header is present, the Gorouter will reject the request if the requested URL does not match that in the header, or if a timeout has expired.

X-CF-Proxy-Metadata

The X-CF-Proxy-Metadata header aids in the encryption and description of X-CF-Proxy-Signature.

Setting Up A Logging Route Service

Clone Logger

```
git clone https://github.com/linuxacademy/content-logging-route-service
```

Push the Logging Route Service as an App

```
cf push logger-<USERNAME> -m 512M -k 512M
```

Create a User-provided Service Instance

```
cf create-user-provided-service logger -r https://<LOGGER_APP_
NAME>.cfapps.io
```

Bind the User-provided Service Instance to the Route of the User API App

```
cf bind-route-service <DOMAIN> logger --hostname user-api-
<USERNAME>
```

Tail the Logs

```
cf logs logger-<USERNAME>
```

Unbind the User-provided

```
cf unbind-route-service <DOMAIN> logger --hostname user-api-
<USERNAME>
```

Delete the Service

```
cf delete-service logger
```

Delete the Logger App

```
cf delete logger-<USERNAME>
```

What Are Service Brokers?

Overview

Services are integrated with Cloud Foundry by implementing a documented API for which the cloud controller is the client; we call this the Service Broker API.

Service Broker is the term we use to refer to a component of the service which implements the

service broker API.

The Service Broker API defines an HTTP interface between the services marketplace of a platform and service brokers.

The service broker is the component of the service that implements the Service Broker API, for which a platform's marketplace is a client. Service brokers are responsible for advertising a catalog of service offerings and service plans to the marketplace, and acting on requests from the marketplace for provisioning, binding, unbinding, and de-provisioning.

URL properties

This specification defines the following properties that might appear within URLs:

- service_id
- plan_id
- instance_id
- binding_id
- operation

Service Broker API

- Catalog Management: describes the plans that the service will be offering:
  - GET /v2/catalog
- Provisioning: Creates an instance of the:
  - PUT /v2/service_instances/:instance_id
- Deprovisioning: Destroys an instance of the service:
  - DELETE /v2/service_instances/:instance_id
- Bind: Binds a service instance to an app:
  - PUT /v2/service_instances/:instance_id/service_bindings/:binding_id
- Unbind: Unbinds a service instance from the app:
  - DELETE /v2/service_instances/:instance_id/service_bindings/:binding_id

Register a Broker

```
cf create-service-broker <BROKER_NAME> <USERNAME> <PASSWORD>
<BROKER_URL> --space-scoped
```

List Service Brokers

```
cf service-brokers
```

Bind the Service Broker Instance to an App

```
cf bs <APP_NAME> <BROKER_NAME>
```

Unbind the Service Broker Instance from the App

```
cf us <APP_NAME> <BROKER_NAME>
```

Update a Broker

```
cf update-service-broker <BROKER_NAME> <USERNAME> <PASSWORD>
<BROKER_URL>
```

Remove a Broker

```
 cf delete-service-broker <BROKER_NAME>
```

Service Broker Documentation

```
https://github.com/openservicebrokerapi/servicebroker/blob/v2.13/
spec.md
```

GitHub Service Broker

```
https://github.com/linuxacademy/content-github-service-broker-ruby
```

Setting Up A Github Service Brokers

Clone Logger

```
git clone https://github.com/linuxacademy/content-github-service-broker-ruby.git
```

Create a Token

```
curl -u <your-github-username> -d '{"scopes": ["repo", "delete_repo"], "note": "CF Service Broker"}' https://api.github.com/authorizations
```

Edit service_broker/config/settings.yml and add in your Github user name and token that was generated. Change the id, name, and plan_id to make unique.

Push the Github Service Broker as an app

```
cf push github-<USERNAME> -m 512m -k 512m
```

Create a Service Broker

```
cf create-service-broker github-repo-<USERNAME> admin password http://github-<USERNAME>.cfapps.io/ --space-scoped
```

Create Service Instances

```
cf create-service github-repo-<USERNAME> public github-repo
```

Delete the Service Broker Instance

```
cf delete-service <SERVICE_NAME>
```

Delete the Service Broker

```
cf delete-service-broker <SERVICE_BROKER>
```

Setting Up A Github Service Brokers

Clone logger

```
git clone https://github.com/linuxacademy/content-github-service-broker-ruby.git
```

Push the Logging Route Service as an app

```
cf push logger-<USERNAME> -m 512M -k 512M
```

Create a User-provided Service Instance

```
cf create-user-provided-service logger -r https://<LOGGER_APP_NAME>.cfapps.io
```

Bind the User-provided Service Instance to the Route of the User API App

```
cf bind-route-service <DOMAIN> logger --hostname user-api-<USERNAME>
```

Tail the Logs

```
cf logs logger-<USERNAME>
```

Unbind the User-provided

```
cf unbind-route-service <DOMAIN> logger --hostname user-api-<USERNAME>
```

Delete the Service

```
cf delete-service logger
```

Delete the Logger App

```
cf delete logger-<USERNAME>
```

Cloud-Native Application Security

Container Security

Container Mechanics

Each instance of an app deployed to CF runs within its own self-contained environment, a Garden container. This container isolates processes, memory, and the filesystem using operating system features and the characteristics of the virtual and physical infrastructure where CF is deployed.

CF achieves container isolation by namespacing kernel resources that would otherwise be shared. The intended level of isolation is set to prevent multiple containers that are present on the same host from detecting each other. Every container includes a private root filesystem, which includes a Process ID (PID), namespace, network namespace, and mount namespace.

CF creates container filesystems using the Garden Roots (GrootFS) tool. It stacks the following using OverlayFS:

- A read-only base filesystem: This filesystem has the minimal set of operating system packages and Garden-specific modifications common to all containers. Containers can share the same read-only base filesystem because all writes are applied to the read-write layer

- A container-specific read-write layer: This layer is unique to each container, and its size is limited by XFS project quotas. The quotas prevent the read-write layer from overflowing into unallocated space

Resource control is managed using Linux control groups. Associating each container with its own cgroup or job object limits the amount of memory that the container may use. Linux cgroups also require the container to use a fair share of CPU compared to the relative CPU share of other containers.

Inbound and Outbound Traffic

A host VM has a single IP address.

Inbound requests flow from the load balancer through the router to the host cell, then into the application container. The router determines which application instance receives each request.

Outbound traffic flows from the application container to the cell, then to the gateway on the cell's virtual network interface. Depending on your IaaS, this gateway may be a NAT to external networks.

Network Traffic Rules

Administrators configure rules to govern container network traffic. This is how containers send traffic outside of CF and receive traffic from outside the Internet. These rules can prevent system access from external networks and between internal components and determine if apps can establish connections over the virtual network interface.

Administrators configure these rules at two levels:

- Application Security Groups (ASGs) apply network traffic rules at the container level
- Container-to-Container networking policies determine app-to-app communication. Within CF, apps can communicate directly with each other, but the containers are isolated from outside CF

Container Security

CF secures containers through the following measures:

- Running application instances in unprivileged containers by default
- Hardening containers by limiting functionality and access rights
- Only allowing outbound connections to public addresses from application containers

Types

Garden has two container types: unprivileged and privileged. Currently, CF runs all application instances and staging tasks in unprivileged containers by default. This measure increases security by eliminating the threat of root escalation inside the container.

Application Security Groups

About Application Security Groups

Application Security Groups (ASGs) are collections of egress rules that specify the protocols, ports, and IP address ranges where app or task instances send traffic.

ASGs define allowed rules, and their order of evaluation is unimportant when multiple ASGs apply to the same space or deployment. The platform sets up rules to filter and log outbound network traffic from app and task instances. ASGs apply to both buildpack-based and Docker-based apps and tasks.

Staging and Running ASGs

Staging ASG for app and task staging, and a running ASG for app and task runtime.

Default ASGs

Cloud Foundry pre-configures two ASGs: public_networks and dns.

Unless you modify these before your initial deployment, these ASGs are applied by default to all containers in your deployment.

Creating ASGs

Create a rules file: a JSON-formatted single array containing objects that describe the rules. Run cf create-security-group SECURITY-GROUP PATH-TO-RULES-FILE.

Bind ASGs

- To bind an ASG to the platform-wide staging ASG set, run cf bind-staging-security-group
- To bind an ASG to a space-scoped running ASG set, run cf bind-security-group

Unbind ASGs

- To unbind an ASG from the platform-wide staging ASG set, run cf unbind-staging-security-group
- To unbind an ASG from the platform-wide running ASG set, run cf unbind-running-security-group

Delete ASGs

To delete an ASG, run cf delete-security-group.

Example ASG

```
[
  {
    "protocol": "icmp",
    "destination": "0.0.0.0/0",
    "type": 0,
    "code": 0
  },
  {
    "protocol": "tcp",
    "destination": "10.0.11.0/24",
    "ports": "80,443",
    "log": true,
    "description": "Allow http and https traffic from ZoneA"
  }
]
```

Troubleshooting Applications on Cloud Foundry

Logging Overview

A twelve-factor app never concerns itself with routing or storage of its output stream. It should not attempt to write to or manage logfiles. Instead, each running process writes its event stream, unbuffered, to stdout. During local development, the developer will view this stream in the foreground of their terminal to observe the app's behavior.

Log Draining

Create a new system in Papertrail:

```
https://papertrailapp.com/systems/new
```

Create syslog drain service

```
cf cups user-api-log-drain -l syslog://<SYSLOG_DRAIN_URL>:<SYSLOG_
DRAIN_PORT>
```

Bind syslog drain service:

```
cf bs <APP_NAME> <SYSLOG_DRAIN_SERVICE>
cf restart <APP_NAME>
```

Monitoring

Get license key from New Relic by going to Account Settings.

Crate New Relic Service

```
cf create-user-provided-service newrelic -p uri
```

Bind the New Relic Service to you Apps

```
cf bs <APP_NAME> newrelic
```

Restage Your Apps so the Droplet can be Rebuilt with New Relic

```
cf restage <APP_NAME>
```

Conclusion

Preparing For The Exam

https://www.cloudfoundry.org/certification/

About The Exam

https://www.cloudfoundry.org/certification/

Taking The Exam

- What Does the Exam Cost? $300
- How long do I have to take the exam? Up to 4 hours
- The passing score for the exam is 67%
- ID Requirements:
  - Passport
  - Government-issued driver's license/permit
  - National identity card
  - State- or province-issued identity card
- One free retake per Exam purchase within 12 months
- The Certification is valid for 24 months
- Exam Proctoring:
  - Online
  - Use your own computer
  - Chrome or Chromium browser
  - Microphone and Camera

- Reliable internet access
- Work area must be clean
- No talking during the exam
- No note taking during the exam

Compatibility Check

```
https://www.examslocal.com/ScheduleExam/Home/CompatibilityCheck
```

Questions?

```
certification-support@cloudfoundry.org
```