Cloud Foundry Technical Overview

1 Introduction

Cloud Foundry BOSH is an open source tool chain for release engineering, deployment and life cycle management of cloud-scale distributed services. In this manual we describe the architecture, topology, configuration, and use of BOSH, as well as the structure and conventions used in packaging and deployment.

2 Managing Distributed Services

BOSH was originally developed in the context of the Cloud Foundry Application Platform as a Service, but even if this has been the primary consumer, the framework is general purpose and can be used to deploy other distributed services on top of a Cloud Provider Interface (CPI) provided by VMware vSphere, Amazon Web Services, or OpenStack.

3 BOSH Components

3.1 Fig 1. Interaction of BOSH Components

3.2 Infrastructure as a Service (IaaS)

The core BOSH engine is abstracted away from any particular Infrastructure as a Service (IaaS), such as VMware vSphere, AWS or OpenStack. IaaS interfaces are implemented as plugins to BOSH. Currently, BOSH supports both VMware vSphere and Amazon Web Services. Virtual Machines (VMs) are created and destroyed through Workers, which are sent instructions from the Director. Those VMs are created based on a **Stemcell** that is uploaded to BOSH's Blobstore through the Command Line Interface (CLI).

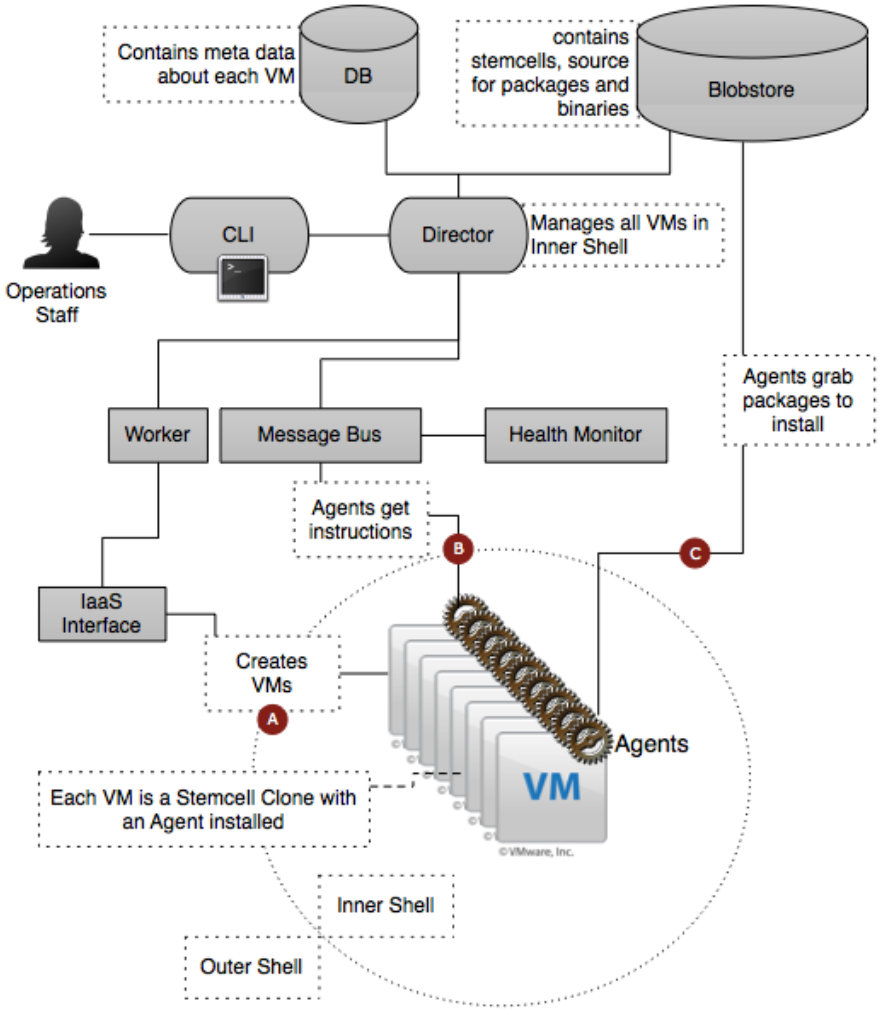

Figure 1: Figure 1

3.3 Cloud Provider Interface (CPI)

As a user of BOSH you're not directly exposed to the the BOSH Cloud Provider Interface, but it can be helpful to understand its primitives when learning how BOSH works. The current examples of these interfaces are in: `bosh/vsphere_cpi/lib/cloud/vsphere/cloud.rb` for vSphere, and `bosh/aws_cpi/lib/cloud/aws/cloud.rb` for Amazon Web Services. Within those subdirectories are Ruby classes with methods to do the following:

- create_stemcell
- delete_stemcell
- create_vm
- delete_vm
- reboot_vm
- configure_networks
- create_disk
- delete_disk
- attach_disk
- detach_disk

In addition to these methods are others specific to each cloud interface. For example, the Amazon Web Services interface includes methods for Elastic Block Store, which are unnecessary on vSphere. Please refer to the API documentation in the files listed above for a detailed explanation of the CPI primitives.

The CPI is used primarily to do low level creation and management of resources in an IaaS. Once a resource is up and running, command and control is handed over to the higher level BOSH Director-Agent interaction.

3.4 BOSH Director

The Director is the core orchestrating component in BOSH which controls creation of VMs, deployment, and other life cycle events of software and services.

3.5 BOSH CLI

The BOSH Command Line Interface is the mechanism for users to interact with BOSH using a terminal session. BOSH commands follow the format shown below:

```
$bosh [--verbose] [--config|-c <FILE>] [--cache-dir <DIR>]
        [--force] [--no-color] [--skip-director-checks] [--quiet]
        [--non-interactive]
```

A full overview of BOSH commands and installation appears in the BOSH CLI (chapter 7) and BOSH installation (section 4.1) sections.

3.6 Stemcells

A BOSH Stemcell is a VM template with an embedded BOSH Agent. The Stemcell used for Cloud Foundry is a standard Ubuntu distribution. Stemcells are uploaded using the BOSH CLI and used by the Director when creating VMs through the CPI. When the Director creates a VM through the CPI, it will pass along configurations for networking and storage, as well as the location and credentials for the BOSH Message Bus and the BOSH Blobstore.

3.7 Agent

When a Stemcell is created, it is comprised of the barebones operating system (Ubuntu in the case of Cloud Foundry) and an Agent. The Agent listens for instructions from the Director and performs operations on the VM according to these instructions. A typical instruction would be to download and install new packages from the Blobstore.

3.8 Releases

A Release in BOSH is a packaged bundle of service descriptors known as Jobs. Jobs are collections of software bits and configurations. Any given Release contains all the static bits (source or binary) required to have BOSH manage an application or a distributed service.

A Release is typically not restricted to any particular environment. As such, it can be re-used across clusters handling different stages in a service life cycle, such as Development, QA, Staging, or Production. The BOSH CLI manages both the creation of Releases and their deployments into specific environments.

3.9 Deployments

While BOSH Stemcells and Releases are static components, we say that they are bound together into a Deployment by a Deployment Manifest. In the Deployment Manifest, you declare pools of VMs, which networks they live on, and which Jobs (service components) from the Release you want to activate. Job configurations specify life cycle parameters, the number of instances of a Job, and network and

storage requirements. Furthermore, the Deployment Manifest allows you to specify properties used to parameterize configuration templates contained in the Release.

Using the BOSH CLI, you specify a Deployment Manifest and perform a Deploy operation (`bosh deploy`), which creates or updates resources on your cluster according to your specifications.

3.10 Blobstore

The BOSH Blobstore is used to store the content of Releases (BOSH Jobs (section 9.2) and Packages (section 9.3) in their source form as well as the compiled image of BOSH Packages). Releases (chapter 9) are uploaded by the BOSH CLI and inserted into the Blobstore by the BOSH Director. When you deploy a Release, BOSH will orchestrate the compilation of packages and store the result in the Blobstore. When BOSH deploys a BOSH Job to a VM, the BOSH Agent will pull the specified Job and associated BOSH Packages from the Blobstore.

BOSH also uses the Blobstore as an intermediate store for large payloads, such as log files (see BOSH logs) and output from the BOSH Agent that exceeds the max size for messages over the message bus.

There are currently three Blobstores supported in BOSH:

1. Atmos[1]
2. S3[2]
3. simple blobstore server[3]

The default BOSH configuration uses the simple blobstore server, as the load is very light and low latency is preferred.

3.11 Health Monitor

The BOSH (Health) Monitor receives health status and life cycle events from the BOSH Agents and can through notification plugins (such as email), notify if components are in an unexpected state. It has a simple awareness of events in the system, so as to not alert if a component is updated.

[1] http://www.emc.com/storage/atmos/atmos.htm
[2] http://aws.amazon.com/s3/
[3] https://github.com/cloudfoundry/bosh/tree/master/simple_blobstore_server

3.12 Message bus

BOSH uses the NATS[4] message bus for command and control.

4 Using BOSH

Before we can use BOSH we need to install the BOSH CLI. You will need a running development environment with an uploaded Stemcell. You can learn about those steps in the BOSH Installation (chapter 5) section.

4.1 Installing BOSH Command Line Interface

The following steps install BOSH CLI on Ubuntu 10.04 LTS. You can install on either a physical or Virtual Machine.

Install Ruby via rbenv

1. Bosh is written in Ruby. Let's install Ruby's dependencies

   ```
   sudo apt-get install git-core build-essential libsqlite3-dev curl libmysqlclient-dev li
   ```

2. Get the latest version of rbenv

   ```
   cd
   git clone git://github.com/sstephenson/rbenv.git .rbenv
   ```

3. Add ∼/.rbenv/bin to your $PATH for access to the rbenv command-line utility

   ```
   echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
   ```

4. Add rbenv init to your shell to enable shims and autocompletion

   ```
   echo 'eval "$(rbenv init -)"' >> ~/.bash_profile
   ```

5. Download Ruby 1.9.2 *Note: You can also build ruby using ruby-build plugin for rbenv. See https://github.com/sstephenson/ruby-build*

   ```
   wget http://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.2-p290.tar.gz
   ```

6. Unpack and install Ruby

[4] http://github.com/dcollison/nats

```
tar xvfz ruby-1.9.2-p290.tar.gz
cd ruby-1.9.2-p290
./configure --prefix=$HOME/.rbenv/versions/1.9.2-p290
make
make install
```

7. Restart your shell so the path changes take effect

```
source ~/.bash_profile
```

8. Set your default Ruby to be version 1.9.2

```
rbenv global 1.9.2-p290
```

9. Update rubygems and install bundler. *Note: After installing gems (`gem install` or `bundle install`) run `rbenv rehash` to add new shims*

```
gem update --system
gem install bundler
rbenv rehash
```

Install Local BOSH and BOSH Releases

1. Sign up for the Cloud Foundry Gerrit server at http://reviews.cloudfoundry.org[5]
2. Set up your ssh public key (accept all defaults)

```
ssh-keygen -t rsa
```

3. Copy your key from ∼/.ssh/id_rsa.pub into your Gerrit account
4. Create ∼/.gitconfig as follows (Make sure that the email specified is registered with gerrit):

```
[user]
name = YOUR_NAME
email = YOUR_EMAIL
[alias]
gerrit-clone = !bash -c 'gerrit-clone $@' -
```

5. Clone gerrit tools using git

```
git clone git@github.com:vmware-ac/tools.git
```

[5] http://reviews.cloudfoundry.org

NOTE: PUBLIC TOOLS REPO IN FINAL DRAFT

1. Add gerrit-clone to your path

```
echo 'export PATH="$HOME/tools/gerrit/:$PATH"' >> ~/.bash_profile
```

2. Restart your shell so the path changes take effect

```
source ~/.bash_profile
```

3. Clone BOSH repositories from Gerrit

```
git gerrit-clone ssh://reviews.cloudfoundry.org:29418/release.git
git gerrit-clone ssh://reviews.cloudfoundry.org:29418/bosh.git
```

4. Run some rake tasks to install the BOSH CLI

```
cd ~/bosh
rake bundle_install (Note: if this fails run 'gem pristine rake' and retry)
cd cli
bundle exec rake build
gem install pkg/bosh_cli-x.x.x.gem
rbenv rehash
```

Deploy to your BOSH Environment

With a fully configured environment, we can begin deploying a Cloud Foundry Release to our environment. As listed in the prerequisites, you should already have an environment running, as well as the IP address of the BOSH Director. To set this up, skip to the BOSH Installation (chapter 5) section.

Point BOSH at a Target and Clean your Environment

1. Target your Director (this IP is an example.) **NOTE: EXAMPLE WORKS FOR INTERNAL USE (u: admin /p: admin)**

```
bosh target 11.23.128.219:25555
```

2. Check the state of your BOSH settings.

```
bosh status
```

3. The result of your status will be akin to:

```
Target         dev48 (http://11.23.128.219:25555) Ver: 0.3.12 (01169817)
UUID           4a8a029c-f0ae-49a2-b016-c8f47aa1ac85
User           admin
Deployment     not set
```

4. List any previous Deployments (we will remove them in a moment). If this is your first Deployment, there will be none listed.

   ```
   bosh deployments
   ```

5. The result of `bosh deployments` should be akin to:

   ```
   +-------+
   | Name  |
   +-------+
   | dev48 |
   +-------+
   ```

6. Delete the existing Deployments (ex: dev48.)

   ```
   bosh delete deployment dev48
   ```

7. Answer `yes` to the prompt and wait for the deletion to complete.

8. List previous Releases (we will remove them in a moment). If this is your first Deployment, there will be none listed.

   ```
   bosh releases
   ```

9. The result of `bosh releases` should be akin to:

   ```
   +---------------+---------------+
   | Name          | Versions      |
   +---------------+---------------+
   | cloudfoundry  | 47, 55, 58    |
   +---------------+---------------+
   ```

10. Delete the existing Releases (ex: cloudfoundry)

    ```
    bosh delete release cloudfoundry
    ```

11. Answer `yes` to the prompt and wait for the deletion to complete.

Create a Release

1. Change directories into the release directory.

NOTE: This is not correct yet - Get correct locations and names from Oleg

```
cd ~/release
```

```
This directory contains the Cloud Foundry deployment and release files.
```

1. Reset your environment

```
bosh reset release
```

2. Answer `yes` to the prompt and wait for the environment to be reset
3. Create a Release

```
bosh create release --force --with-tarball
```

4. Answer `cloudfoundry` to the `release name` prompt
5. Your terminal will display information about the release including the Release Manifest, Packages, Jobs, and tarball location.
6. Open `bosh-sample-release/cloudfoundry.yml` in your favorite text editor and confirm that `name` is `cloudfoundry` and `version` matches the version that was displayed in your terminal (if this is your first release, this will be version 1.)

Deploy the Release

1. Upload the cloudfoundry Release to your Environment.

```
bosh upload release dev_releases/cloudfoundry-1.tgz
```

2. Your terminal will display information about the upload, and an upload progress bar will reach 100% after a few minutes.
3. Open `releases/cloudfoundry.yml` and make sure that your network settings match the environment that you were given.
4. Deploy the Release.

```
bosh deploy
```

5. Your deployment will take a few minutes.

6. You may now target the Cloud Foundry deployment using VMC, as described in the Cloud Foundry documentation.

5 BOSH Installation

Deploying BOSH is a two step process. First, The BOSH Deployer is used to deploy a micro BOSH, which will live in a single virtual machine. The second step is to use the micro BOSH as a means to deploy the final, distributed production BOSH on multiple VMs. The graphic below illustrates this two step process.

NOTE: Matt will create a graphic for this

5.1 Prerequisites

1. It is recommend that you install into an empty gemset (or similar.)

2. Install some core packages on Ubuntu.

```
% sudo apt-get -y install libsqlite3-dev genisoimage
```

3. Build the BOSH Deployer.

```
% cd bosh/deployer
% bundle install
% rake install
```

Once you have installed micro BOSH, you will see some extra commands appear after typing `bosh` on your command line.

The `bosh micro` commands must also be run within the deployments directory

```
% bosh help
...
Micro
    micro deployment [<name>] Choose micro deployment to work with
    micro status              Display micro BOSH deployment status
    micro deployments         Show the list of deployments
    micro deploy <stemcell>   Deploy a micro BOSH instance to the currently
                    selected deployment
                    --update   update existing instance
    micro delete              Delete micro BOSH instance (including
```

```
                    persistent disk)
micro agent <args>        Send agent messages
micro apply <spec>        Apply spec
```

5.2 Configuration

For a minimal configuration example, see: `deployer/spec/assets/test-bootstrap-config.yml`. Note that `disk_path` is `BOSH_Deployer` rather than `BOSH_Disks`. A datastore folder other than `BOSH_Disks` is required if your vCenter hosts other Directors. The `disk_path` folder needs to be created manually. Also, your configuration must live inside a `deployments` directory and follow the convention of having a `$name` subdir containing `micro_bosh.yml`, where `$name` is your Deployment name.

For example:

```
% find deployments -name micro_bosh.yml
deployments/vcs01/micro_bosh.yml
deployments/dev32/micro_bosh.yml
deployments/dev33/micro_bosh.yml
```

Deployment state is persisted to deployments/bosh-deployments.yml.

5.3 Deployment

1. Download a micro BOSH stemcell:

```
% mkdir -p ~/stemcells
% cd stemcells
% bosh public stemcells
+------------------------------+-----------------------------------------------------+
| Name                         | Url                                                 |
+------------------------------+-----------------------------------------------------+
| bosh-stemcell-0.4.7.tgz      | https://blob.cfblob.com/rest/objects/4e4e7...h120= |
| micro-bosh-stemcell-0.1.0.tgz | https://blob.cfblob.com/rest/objects/4e4e7...5Mms= |
| bosh-stemcell-0.3.0.tgz      | https://blob.cfblob.com/rest/objects/4e4e7...mw1w= |
| bosh-stemcell-0.4.4.tgz      | https://blob.cfblob.com/rest/objects/4e4e7...r144= |
+------------------------------+-----------------------------------------------------+
To download use 'bosh download public stemcell <stemcell_name>'.
% bosh download public stemcell micro-bosh-stemcell-0.1.0.tgz
```

2. Set the micro BOSH Deployment using:

```
% cd /var/vcap/deployments
```

```
% bosh micro deployment dev33
Deployment set to '/var/vcap/deployments/dev33/micro_bosh.yml'
```

3. Deploy a new micro BOSH instance and create a new persistent disk.

```
% bosh micro deploy ~/stemcells/micro-bosh-stemcell-0.1.0.tgz
```

4. Update an existing micro BOSH instance. The existing persistent disk will be attached to the new VM.

```
% bosh micro deploy ~/stemcells/micro-bosh-stemcell-0.1.1.tgz --update
```

5.4 Deleting a micro BOSH deployment

The `delete` command will delete the VM, Stemcell and persistent disk.
Example:

```
% bosh micro delete
```

5.5 Checking Status of a micro BOSH deploy

The status command will show the persisted state for a given micro BOSH instance.

```
% bosh micro status
Stemcell CID   sc-f2430bf9-666d-4034-9028-abf9040f0edf
Stemcell name  micro-bosh-stemcell-0.1.0
VM CID         vm-9cc859a4-2d51-43ca-8dd5-220425518fd8
Disk CID       1
Deployment     /var/vcap/deployments/dev33/micro_bosh.yml
Target         micro (http://11.23.194.100:25555) Ver: 0.3.12 (00000000)
```

5.6 Listing Deployments

The `deployments` command prints a table view of deployments/bosh-deployments.yml.

```
% bosh micro deployments
```

5.7 Applying a specification

The micro-bosh-stemcell includes an embedded `apply_spec.yml`. This command can be used to apply a different spec to an existing instance. The `apply_spec.yml` properties are merged with your Deployment's network.ip and cloud.properties.vcenters properties.

```
% bosh micro apply apply_spec.yml
```

5.8 Sending messages to the micro BOSH agent

The CLI can send messages over HTTP to the agent using the `agent` command.
Example:

```
% bosh micro agent ping
"pong"
```

5.9 Deploying Production BOSH through Micro BOSH

TODO: The steps below are only an outline. Need to expand on them.

1. Once your micro BOSH instance is deployed, you can target its Director:

   ```
   $ bosh micro status
   ...
   Target         micro (http://11.23.194.100:25555) Ver: 0.3.12 (00000000)

   $ bosh target http://11.23.194.100:25555
   Target set to 'micro (http://11.23.194.100:25555) Ver: 0.3.12 (00000000)'

   $ bosh status
   Updating director data... done

   Target         micro (http://11.23.194.100:25555) Ver: 0.3.12 (00000000)
   UUID           b599c640-7351-4717-b23c-532bb35593f0
   User           admin
   Deployment     not set
   ```

2. Upload a Stemcell. **NOTE** Do not use the micro BOSH Stemcell. Use a BOSH Stemcell.

3. Upload the BOSH release.

4. Ensure `bosh deployment` is set to the BOSH deployment.

5. Run `bosh deploy`.
6. Wait for successful deployment.
7. Target the newly deployed BOSH Director.
8. Your newly deployed Production BOSH is ready to use.
9. *Optional*: Delete micro BOSH deployment.

5.10 vCenter Configuration

Someone write this eh?

6 Configure BOSH Director

[NOTE] The current `chef-solo` based installer is being re-written as a mini-bosh instance.

To install BOSH into an infrastructure we currently assume that the target VMs have been created.

TODO: check if we can provide vm_builder instructions for creating and // uploading these to IaaS.

```
~/projects/deployments/mycloud/cloud
  assets/
    director/
      director.yml.erb          <1>
      chef.rb                   <2>
      config.yml                <3>

cd ~/projects/bosh/chef_deployer
rake install

cd ~/projects/bosh/release
chef_deployer deploy ~/projects/deployments/mycloud/cloud
```

7 BOSH CLI

The BOSH command line interface is used to interact with the BOSH director to perform actions on the cloud. For the most recent documentation on its functions, install bosh (section 4.1) and simply type `bosh`. Usage:

```
bosh [--verbose] [--config|-c <FILE>] [--cache-dir <DIR]
     [--force] [--no-color] [--skip-director-checks] [--quiet]
     [--non-interactive]
     command [<args>]
```

Currently available bosh commands are:

```
Deployment
  deployment [<name>]       Choose deployment to work with (it also updates
                            current target)
  delete deployment <name>  Delete deployment
                            --force    ignore all errors while deleting
                                       parts of the deployment
  deployments               Show the list of available deployments
  deploy                    Deploy according to the currently selected
                            deployment manifest
                            --recreate recreate all VMs in deployment
  diff [<template_file>]    Diffs your current BOSH deployment
                            configuration against the specified BOSH
                            deployment configuration template so that you
                            can keep your deployment configuration file up to
                            date. A dev template can be found in deployments
                            repos.

Release management
  create release            Create release (assumes current directory to be a
                            release repository)
                            --force    bypass git dirty state check
                            --final    create production-ready release
                                       (stores artefacts in blobstore,
                                       bumps final version)
                            --with-tarball
                                       create full release tarball(by
                                       default only manifest is created)
                            --dry-run  stop before writing release manifest
                                       (for diagnostics)
  delete release <name> [<version>]
                            Delete release (or a particular release version)
                            --force    ignore errors during deletion
  verify release <path>     Verify release
  upload release [<path>]   Upload release (<path> can point to tarball or
                            manifest, defaults to the most recently created
                            release)
  releases                  Show the list of available releases
  reset release             Reset release development environment (deletes
                            all dev artifacts)
```

```
  init release [<path>]     Initialize release directory
  generate package <name>   Generate package template
  generate job <name>       Generate job template

Stemcells
  upload stemcell <path>    Upload the stemcell
  verify stemcell <path>    Verify stemcell
  stemcells                 Show the list of available stemcells
  delete stemcell <name> <version>
                            Delete the stemcell
  public stemcells          Show the list of publicly available stemcells for
                            download.
  download public stemcell <stemcell_name>
                            Downloads a stemcell from the public blobstore.

User management
  create user [<name>] [<password>]
                            Create user

Job management
  start <job> [<index>]     Start job/instance
  stop <job> [<index>]      Stop job/instance
                            --soft     stop process only
                            --hard     power off VM
  restart <job> [<index>]   Restart job/instance (soft stop + start)
  recreate <job> [<index>]  Recreate job/instance (hard stop + start)

Log management
  logs <job> <index>        Fetch job (default) or agent (if option provided)
                            logs
                            --agent    fetch agent logs
                            --only <filter1>[...]
                                       only fetch logs that satisfy given
                                       filters (defined in job spec)
                            --all      fetch all files in the job or agent log
                                       directory

Task management
  tasks                     Show the list of running tasks
  tasks recent [<number>]   Show <number> recent tasks
  task [<task_id>|last]     Show task status and start tracking its output
                            --no-cache don't cache output locally
                            --event|--soap|--debug
                                       different log types to track
                            --raw      don't beautify log
```

```
  cancel task <id>          Cancel task once it reaches the next cancel
                            checkpoint

Property management
  set property <name> <value>
                            Set deployment property
  get property <name>       Get deployment property
  unset property <name>     Unset deployment property
  properties                List current deployment properties
                            --terse    easy to parse output

Maintenance
  cleanup                   Remove all but several recent stemcells and
                            releases from current director (stemcells and
                            releases currently in use are NOT deleted)
  cloudcheck                Cloud consistency check and interactive repair
                            --auto     resolve problems automatically (not
                                       recommended for production)
                            --report   generate report only, don't attempt
                                       to resolve problems

Misc
  status                    Show current status (current target, user,
                            deployment info etc.)
  vms [<deployment>]        List all VMs that supposed to be in a deployment
  target [<name>] [<alias>] Choose director to talk to (optionally creating
                            an alias). If no arguments given, show currently
                            targeted director
  login [<name>] [<password>]
                            Provide credentials for the subsequent
                            interactions with targeted director
  logout                    Forget saved credentials for targeted director
  purge                     Purge local manifest cache

Remote access
  ssh <job> [index] [<options>] [command]
                            Given a job, execute the given command or start an
                            interactive session
                            --public_key <file>
                            --gateway_host <host>
                            --gateway_user <user>
                            --default_password
                                       Use default ssh password. Not
                                       recommended.
  scp <job> <--upload | --download> [options] /path/to/source /path/to/destination
                            upload/download the source file to the given job.
```

```
                              Note: for dowload /path/to/destination is a
                              directory
                              --index <job_index>
                              --public_key <file>
                              --gateway_host <host>
                              --gateway_user <user>
  ssh_cleanup <job> [index] Cleanup SSH artifacts

Blob
  upload blob <blobs>       Upload given blob to the blobstore
                              --force    bypass duplicate checking
  sync blobs                Sync blob with the blobstore
                              --force    overwrite all local copies with the
                                         remote blob
  blobs                     Print blob status
```

8 Stemcells + releases /Director interaction

upload

9 Releases

9.1 Release Repository

A BOSH Release is built from a directory tree following a structure described in this section:

9.2 Jobs

TODO: job templates TODO: prepare script TODO: use of properties TODO: "the job of a vm" TODO: monitrc (gonit) TODO: DNS support

9.3 Packages

TODO: ishisness!

Package Compilation

Packages are compiled on demand during the deployment. The director (section 3.4) first checks to see if there already is a compiled version of the package for

the stemcell version it is being deployed to, and if it doesn't already exist a compiled version, the director will instantiate a compile VM (using the same stemcell version it is going to be deployed to) which will get the package source from the blobstore, compile it, and then package the resulting binaries and store it in the blobstore.

To turn source code into binaries each package has a `packaging` script that is responsible for the compilation, and is run on the compile VM. The script gets two environment variables set from the BOSH agent which tells it where to install the files the package generates `BOSH_INSTALL_TARGET`, and the other is `BOSH_COMPILE_TARGET` which is the directory containing the source (which is the current directory when the `packaging` script is invoked). The `BOSH_INSTALL_TARGET` is set to `/var/vcap/data/packages/<package name>/<package version>`. When the package is installed a symlink is created from `/var/vcap/packages/<package name>` which points to the latest version of the package. This link should be used when refering to another package in the `pckaging` script.

There is an optional `pre_packaging` script, which is run when the source of the package is assembled during the `bosh create release`. It can for instance be used to limit which parts of the source that get packages up and stored in the blobstore. It gets the environment variable `BUILD_DIR` set by the BOSH cli (chapter 7), which is the directory containing the source to be packaged.

Package specs

Dependencies

The package `spec` file contains a section which lists other packages the current package depends on. These dependencies are compile time dependencies, as opposed to the job dependencies which are runtime dependencies.

When the director plans the compilation of a package during a deployment, it first makes sure all dependencies are compiled before it proceeds to compile the current package, and prior to commencing the compilation all dependent packages are installed on the compilation VM.

9.4 Sources

final release

9.5 Blobs

To create final releases you need to configure your release repository with a blobstore. This is where BOSH will upload the final releases to, so that the release can later be retreived from another computer.

To prevent the release repository from becoming bloated with large binary files (source tar-balls), large files can be placed in the `blobs` directory, and then uploaded to the blobstore.

For production releases you should use either the Atmos or S3 blobstore and configure them as described below.

Atmos

To use Atmos, edit `config/final.tml` and add the following (replacing the `url`, `uid` and `secret` with your account information):

```
blobstore:
  provider: atmos
  options:
    tag: BOSH
    url: https://blob.cfblob.com
    uid: 1876876dba98981ccd091981731deab2/user1
    secret: ahye7dAS93kjWOIpqla9as8GBu1=
```

S3

To use S3, edit `config/final.tml` and add the following (replacing the `access_key_id`, `bucket_name`, `encryption_key` and `secret_access_key` with your account information):

```
blobstore:
  provider: s3
  options:
    access_key_id: KIAK876234KJASDIUH32
    bucket_name: 87623bdc
    encryption_key: sp$abcd123$foobar1234
    secret_access_key: kjhasdUIHIkjas765/kjahsIUH54asd/kjasdUSf
```

Local

If you are just trying out BOSH and don't have an Atmos or S3 account, you can use the local blobstore provider (which stored the files on disk instead of a remote server).

```
blobstore:
  provider: local
  options:
    blobstore_path: /path/to/blobstore/directory
```

Note that local should **only** be used for testing purposes as it can't be shared with others (unless they run on the same system).

9.6 Versioning schemes

9.7 Configuring Releases

9.8 Building Releases

9.9 Final Releases

10 BOSH Deployments

10.1 Steps of a Deployment

Here are the steps that take place in a BOSH deployment.

1. Preparing deployment
    a) binding deployment - Creates an entry in the Director's database for the deployment if it doesn't exist.
    b) binding release - Makes sure the release specified in deployment configuration exists then locks it from being deleted.
    c) binding existing deployment - Takes existing VMs and sets them up to be used for the deployment.
    d) binding resource pools - Gives idle VMs network reservations.
    e) binding stemcells - Makes sure the stemcell specified has been uploaded and then locks it from being deleted.
    f) binding templates - Sets up internal data objects to track packages and their pre-reqs for installation.
    g) binding unallocated VMs - For each job instance required it determines whether a VM running the instance already exists and assigns one if not.
    h) binding instance networks - Reserves networks for each VM that doesn't have one.
2. Compiling packages - Calculates all packages and their dependencies that need to be compiled. It then begins compiling the packages and storing their output in the blobstore. The number of `workers` specified in the deployment configuration determines how many VMs can be created at once for compiling.
3. Preparing DNS - Creates DNS entry if it doesn't exist.

4. Creating bound missing VMs - Creates new VMs, deletes extra/oudated/idle VMs.
5. Binding instance VMs - Any unbound VMs are setup for the deployment.
6. Preparing configuration - Pulls in the configurations for each job to be run.
7. Updating/deleting jobs - Deletes unneeded instances, creates needed instances, updates existing instances if they are not already updated. This is the step where things get pushed live.
8. Refilling resource pools - Creates missing VMs across resource pools after all instance updaters are finished to create additional VMs in order to balance resource pools.

10.2 BOSH Property Store

10.3 BOSH Deployment Manifest

TODO: options global/job propertes TODO: cloud_properties for the cli

11 BOSH Troubleshooting

11.1 BOSH ssh

To ssh to a running Job, first find the name and index of it. Use `bosh vms` to display a list of the VMs that are running and what Job is on each. To ssh to it, run `bosh ssh <job_name> <index>`. The password is whatever is set in the Stemcell. For default Stemcells it is ca$hc0w.

11.2 BOSH Logs

When troubleshooting BOSH or BOSH deployments, it's important to read log files so that problems can be narrowed down. There are three types of logs.

1. BOSH Director logs, via `bosh task <task_number>`

   This contains the output from the BOSH Director whenever a BOSH command is run on it. If there is an issue when running a BOSH command, these logs are where you should start. For instance, if you run `bosh deploy` and it fails, then the BOSH Director will have a log of where things went wrong. To access these logs, find the task number of the failed command by running `bosh tasks recent`. Then, run `bosh task <task_number>`. The Director's logger writes to the logs.

2. Agent logs, in `/var/vcap/bosh/log` or via `bosh logs`

   These logs contain the output from the agents. When an issue with VM setup is suspected, these logs are useful. They will show the actions of the agent, such as setting up network, disks, and running the Job. If a `bosh deploy` fails because one of the VMs is having a problem, you will want to use the BOSH Director logs to find which machine it was. Then, either ssh and access `/var/vcap/bosh/log` or use `bosh logs <job_name> <index> -agent`.

3. Service logs

   These are the logs produced by the actual jobs running on VMs. These may be logs produced by Redis, or a webserver, etc.... These logs will vary because it is up to the Deployment to configure where they are output to. Typically, the output path is defined in a config file in `release/jobs/<job_name>/templates/<config_file>`. For Cloud Foundry, our Jobs are typically configured to log to `/var/vcap/sys/log/<job_name>/<job_name>.log`. These logs can also be accessed via `bosh logs <job_name> <index>`.

11.3 BOSH Cloud Check

BOSH cloud check is a BOSH command line utility that automatically checks for problems in VMs and Jobs that have been deployed. It checks for things such as unresponsive/out-of-sync VMs, unbound disks, etc. To use it, run `bosh cck` and it will prompt you for actions to take if there are any problems found.