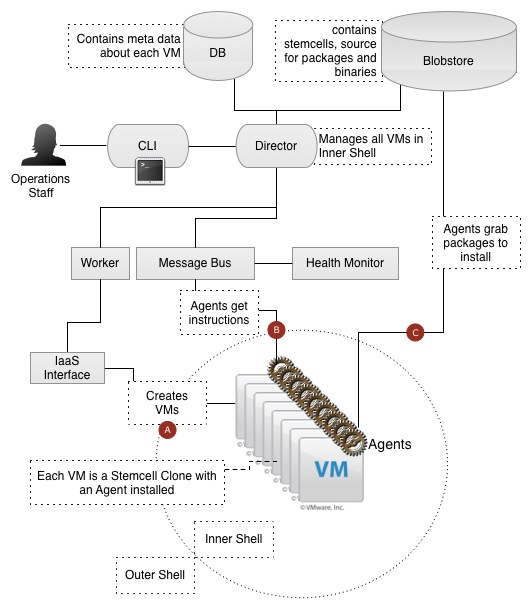
<http://cloudfoundry.github.io/docs/running/bosh/components/>

<http://cloudfoundry.github.io/docs/using/managing-apps/cf/>

https://github.com/cloudfoundry/vcap-services-sample-release

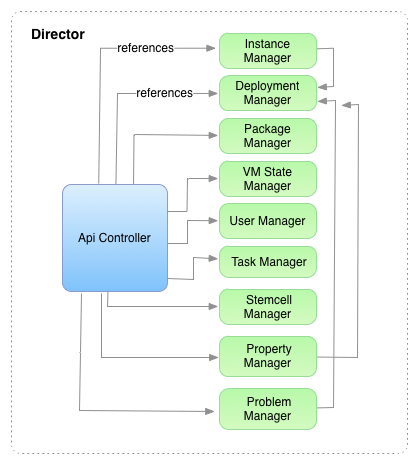
**BOSH Components**



## 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. Command and control is handed over to the the Director-Agent interaction after the CPI has created resources.

There are specific sub components to manage each of the tasks mentioned above. All these are instances of the following classes referenced from the ApiController.



### Deployment Manager

Responsible for creating, updating and deleting the deployments which are specified in the deployment file.

Endpoints and Http Method type exposed by the director which are used to access the deployment manager are described below.

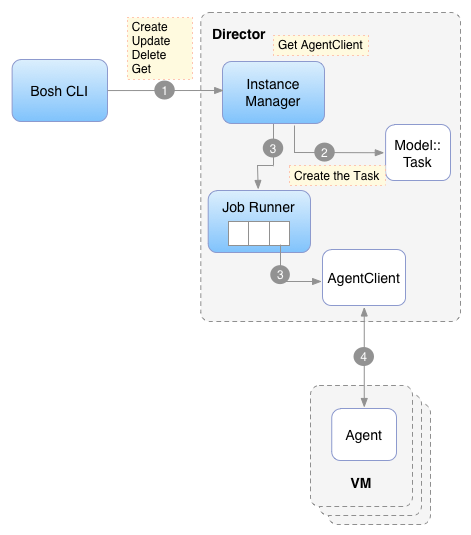
| URL | HTTP METHOD TYPE | DESCRIPTION |
| --- | --- | --- |
| /deployments | POST |  |
| /deployments/:deployment/jobs/:job | PUT | Change the state of a job in a deployment based on the parameter |
| /deployments/:deployment/jobs/:job/:index/logs | GET | Get logs of a particular job in a deployment |
| /deployments/:name | DELETE | Delete a deployment |

### Instance Manager

Instance Manager helps in managing VM Instances created using Bosh deployments.

Some of the functions it performs are 1. Helps in connecting to the VM instance using ssh through an Agent Client 2. Finding an instance 3. Fetching log from a particular instance

Figure below describes the flow when a user tries to SSH into a VM using Bosh CLI



### Problem Manager

This component helps scan a deployment for problems and helps apply resolutions. It uses a model deployment\_problem to keep info about the problem and has 1: many relationship with Deployment Model.

### Property Manager

Properties are attributes specified for jobs in the deployment file. Allows you to find properties associated with a deployment, update a particular property for a deployment. References the deployment Manager.

### Resource Manager

Used to get access to the resources stored in the BlobStore. Some of the actions performed through a resource manager are

1. Get a Resource using an Id

2. Delete a resource by giving an resource Id

3. Get the resource path from an Id

### Release Manager

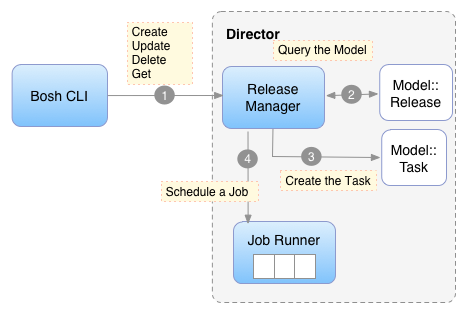
Manages the creation and deletion of releases. Each release references a Release Manager and contains a Deployment Plan object as well as an array of templates.

Director routes the request coming at the following endpoints to the release manager for managing the release lifecycle

| URL | HTTP METHOD TYPE | RESPONSE BODY | DESCRIPTION |
| --- | --- | --- | --- |
| /releases | GET | {“name” => release.name,“versions” => versions, “inuse” => versionsin\_use} | Get the list of all releases uploaded |
| /releases | POST |  | Create a release for the user specified. |

#### Lifecycle of a Release

Figure below shows the interaction between various components of a Director when a release is created/ updated or deleted.



### Stemcell Manager

Stemcell Manager manages the Stem cells. It is responsible for creating, deleting or finding a stemcell.

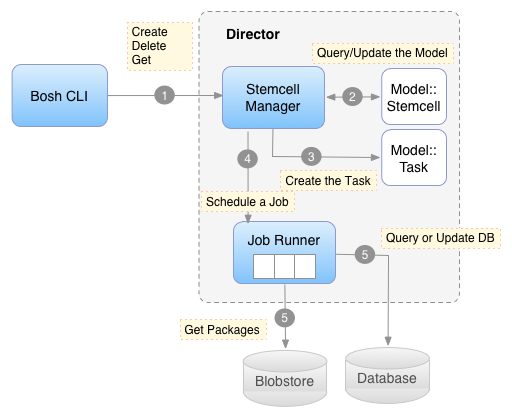
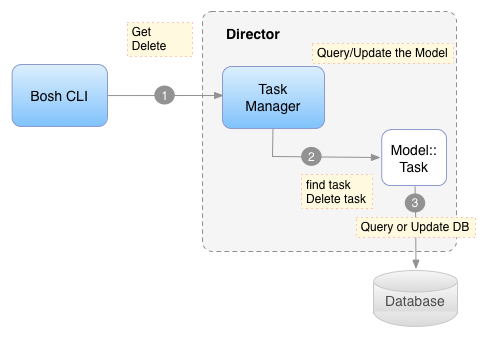


Table below shows the endpoints exposed by the director for managing the Stemcells lifecycle

| URL | HTTP METHOD TYPE | RESPONSE BODY | DESCRIPTION |
| --- | --- | --- | --- |
| /stemcells | GET | { “name” => stemcell.name, “version” => stemcell.version, “cid” => stemcell.cid} | Json specifying the stemcell name, version and cid of the stem cell. |
| /stemcells | POST |  | Stemcell binary file |
| /stemcells | DELETE |  | Delete the specified stemcell |

### Task Manager

Task Manager is responsible for managing the tasks which are created and are being run the Job Runner



Following Http Endpoints are exposed by the Director to get information about a task

| URL | HTTP METHOD TYPE | RESPONSE BODY | DESCRIPTION |
| --- | --- | --- | --- |
| /tasks | GET |  | Get all the tasks being executed of type"updatedeployment", “deletedeployment”, “updaterelease”,“deleterelease”, “updatestemcell”, “deletestemcell” |
| /tasks/:id | GET |  | Send back output for a task with the given id |
| /tasks/:id/output | GET |  | Sends back output of given task id and params[:type] |
| /task/:id | DELETE |  | Delete the task specified by a particular Id |

### User Manager

Manages the users stored in the Director’s database. Main functions performed by the User Manager are

1. Create a User

2. Delete a User

3. Authenticate a User

4. Get a User

5. Update a User

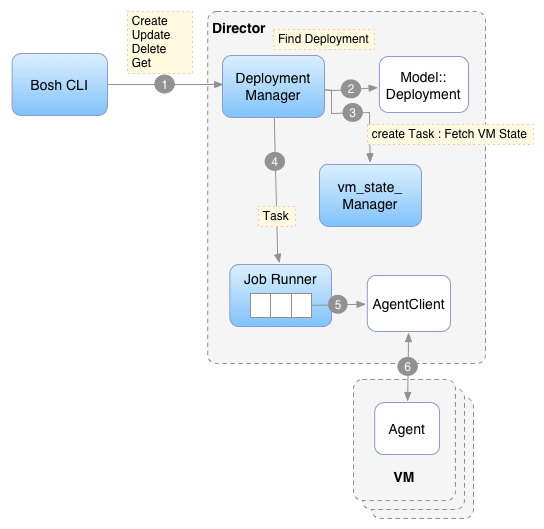
User Management is delegated by the director to the User Manager with the following URLs

| URL | HTTP METHOD TYPE | HTTP REQUEST BODY | DESCRIPTION |
| --- | --- | --- | --- |
| /users | POST |  | Create a User |
| /users/:username | PUT |  | Update a User |
| /users/:username | DELETE |  | Delete a User |

### VM State Manager

Helps fetch the VM State by creating a task which runs the Hob : VmState

The vm state is fetched by creating a GET request on the /deployments/:name/vms endpoint in the Director. name is the name of the deployment.



## Messaging (NATS)

BOSH uses the [NATS](https://github.com/derekcollison/nats) message bus for command and control.

## Workers

A description of Workers and what they do.

## Health Monitor

The BOSH Health Monitor receives health status and life cycle events from the [BOSH Agent](http://cloudfoundry.github.io/docs/running/bosh/components/agent.html) and can send alerts through notification plugins (such as email). The Health Monitor has a simple awareness of events in the system, so as not to alert if a component is updated.

## Stemcell

A Stemcell is a VM template with an embedded [BOSH Agent](http://cloudfoundry.github.io/docs/running/bosh/components/agent.html) The Stemcell used for Cloud Foundry is a standard Ubuntu distribution. Stemcells are uploaded using the [BOSH CLI](http://cloudfoundry.github.io/docs/running/bosh/components/stemcell.html#bosh-cli) and used by the [BOSH Director](http://cloudfoundry.github.io/docs/running/bosh/components/director.html) when creating VMs through the Cloud Provider Interface. 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 [Message Bus](http://cloudfoundry.github.io/docs/running/bosh/components/messaging.html) and the [Blobstore](http://cloudfoundry.github.io/docs/running/bosh/components/blobstore.html).

## Agent

BOSH Agents listen for instructions from the [BOSH Director](http://cloudfoundry.github.io/docs/running/bosh/components/director.html). Every VM contains an Agent. Through the Director-Agent interaction, VMs are given Jobs, or roles, within Cloud Foundry. If the VM's job is to run MySQL, for example, the Director will send instructions to the Agent about which packages must be installed and what the configurations for those packages are.

## Blobstore

The BOSH Blobstore is used to store the content of Releases (BOSH [Jobs](http://cloudfoundry.github.io/docs/running/bosh/reference/jobs.html) and [Packages](http://cloudfoundry.github.io/docs/running/bosh/reference/packages.html) in their source form as well as the compiled image of BOSH Packages. [Releases](http://cloudfoundry.github.io/docs/running/bosh/reference/releases.html) are uploaded by the [BOSH CLI](http://cloudfoundry.github.io/docs/running/bosh/reference/bosh-cli.html) and inserted into the Blobstore by the [BOSH Director](http://cloudfoundry.github.io/docs/running/bosh/components/director.html). 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](http://www.emc.com/storage/atmos/atmos.htm)
2. [S3](http://aws.amazon.com/s3/)
3. [simple blobstore server](https://github.com/cloudfoundry/bosh/tree/master/simple_blobstore_server)

For example configurations of each Blobstore, see the Blobs section below . The default BOSH configuration uses the simple blobstore server, as the load is very light and low latency is preferred.

### 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

Atmos is a shared storage solution from EMC. To use Atmos, edit config/final.tml and config/private.yml, and add the following (replacing the url, uid and secret with your account information):

File config/final.yml

**---**

blobstore:

provider: atmos

options:

tag: BOSH

url: https:**//**blob.cfblob.com

uid: 1876876dba98981ccd091981731deab2**/**user1

File config/private.yml

**---**

blobstore\_secret: ahye7dAS93kjWOIpqla9as8GBu1=

#### S3

To use S3, a shared storage solution from Amazon, edit config/final.tml and config/private.yml, and add the following (replacing the accesskeyid, bucketname, encryptionkey and secretaccesskey with your account information):

File config/final.yml

**---**

blobstore:

provider: s3

options:

access\_key\_id: KIAK876234KJASDIUH32

bucket\_name: 87623bdc

encryption\_key: sp$abcd123$foobar1234

File config/private.yml

**---**

blobstore\_secret: kjhasdUIHIkjas765**/**kjahsIUH54asd**/**kjasdUSf

#### Local

If you are 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).

File config/final.yml

**---**

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).

## Micro BOSH

A description of Micro BOSH here