DigitalRebar Provision Documentation

Release v3.9

RackN team

Sep 13, 2018

Contents

simple, fast and open API-driven server provisioning.

Digital Rebar Provision (DRP) is a APLv2 simple Golang executable that provides a simple yet complete API-driven DHCP/PXE/TFTP provisioning and workflow system.

DRP is designed to be a complete data center provisioning, content scaffolding and infrastructure workflow platform with a cloud native architecture that completely replaces Cobbler, Foreman, MaaS or similar technologies. DRP offers a single golang binary (less than 30MB) with no dependencies capable of installation on a laptop, RPi or switch supporting both bare metal and virtualized infrastructure.

Key Features:

- API-driven infrastructure-as-code automation
- Multi-boot workflows using composable and reusable building blocks
- Event driven actions via Websockets API
- Extensible Plug-in Model for public, vendor and internal enhancements
- Supports ALL orchestration tools including Chef, Puppet, Ansible, SaltStack, Bosh, Terraform, etc
- RAID, IPMI, and BIOS Configuration (via commercial plugins)

CHAPTER 1

Community Resources from https://rebar.digital

- Mailing List
- Chat/messaging via the Digital Rebar `#community` channel is our preferred communication method. If you do not have a Slack invite to our channel, you can Request a Slack Invite
- Alternate chat vi Gitter Live Chat (Gitter.im) and IRC on Freenode #DigitalRebar
- Issues and Features
- Full Documentation (Github /doc sources are updatable via pull request).
- Videos on the DR Provision Playlist provide both specific and general background information.

CHAPTER 2

Install & Quick Start

Note: We HIGHLY recommend using the `latest` version of the documentation, as it contains the most up to date information. Use the version selector in the lower right corner of your browser.

Our Stable Quick Start has fast play-with-it steps. Don't worry, they are very simple and take 10 to 20 minutes.

Want the Latest Quick Start? You'll have access to the newest features in tested work for the next release. This is NOT the bleeding edge!

Regular Install for more details on the install steps. These include production options. (Previous Version Docs)

CHAPTER 3

Table of Contents

Reading on Github? Visit Generated Docs for a generated ToC.

3.1 Quick Start

Note: We HIGHLY recommend using the `latest` version of the documentation, as it contains the most up to date information. Use the version selector in the lower right corner of your browser.

This quick start guide provides a basic installation and start point for further exploration. The guide has been designed for UNIX systems: Mac OS, Linux OS, Linux VMs and Linux Packet Servers. While it is possible to install and run Digital Rebar Provision on a Windows instance, we do not cover that here. The guide employs Curl and Bash commands which are not typically considered safe, but they do provide a simple and quick process for start up.

It is possible to install on hypervisors and in virtualized environments (eg. VirtualBox, VMware Workstation/Fusion, KVM, etc.). Each of these environments requires careful setup up of your network environment and consideration with regard to competing DHCP services. The setup of these environments is outside the scope of this document.

For a full install, please see *Install*

3.1.1 Overview

- Read *Preparation* steps below
- *Install* DRP Endpoint (in "isolated" mode)
- *Start Digital Rebar Provision service* daemon
- *Install Boot Environments (bootenvs)* (OS media for installation)
- *Configure a Subnet* to answer DHCP requests
- *Create a Workflow*

- *Set The Defaults* for defaultBootEnv, defaultStage, unknownBootEnv, and defaultWorkflow
- boot your first Machine and *Install your first Machine* an OS on it

This document refers to the `drpcli` command line tool for manipulating the `dr-provision` service. We do not specify any paths in the documentation. However, in our default quickstart for *isolated* mode, the `drpcli` tool will NOT be installed in any system PATH locations. You must do this, or you may use the local setup symbolic link. For example - simply change `drpcli` to `./drpcli` in the documentation below. Or … copy the binary to a PATH location.

You can perform all of the actions outlined in this document via the hosted web UX Portal. If you choose to use the web Portal, please ensure the setup is completed by reviewing the output of the `Info & Preferences` panel named *System Wizard* are completed successfully. Basic documentation on the Web Portal UX is available.

3.1.2 Preparation

Please make sure your environment doesn't have any conflicts or issues that might cause PXE booting to fail. Some things to note:

- only one DHCP server on a local subnet
- your Machines should be set to PXE boot the correct NIC (on the correct provisioning network interface)
- if you customize Reservations - you must also add all of the correct PXE boot options (see *Creating a Reservation*)
- you need the network information for the subnet that your target Machines will be on
- Mac OSX may require additional setup (see notes below)
- we rely heavily on the `jq` tool for use with the Command Line tool (`drpcli`) - install it if you don't have it already

3.1.3 Install

To begin, execute the following commands in a shell or terminal:

```
mkdir drp ; cd drp
curl -fsSL get.rebar.digital/stable | bash -s -- --isolated install
```

Note: If you want to try the latest code, you can checkout the development tip using `curl -fsSL get.rebar.digital/tip | bash -s -- --isolated --drp-version=tip install`

The command will pull the *stable* `dr-provision` bundle and checksum from github, extract the files, verify prerequisites are installed, and create some initial directories and links.

Note: By default the installer will pull in the default Community Content packages. If you are going to add your own or different (eg RackN registered content), append the `--nocontent` flag to the end of the install command.

Note: The "install.sh" script that is executed (either via 'stable' or 'tip' in the initial 'curl' command), has it's own version number independent of the Digital Rebar Provision endpoint version that is installed (also typically called 'tip' or 'stable'). It is NOT recommend to "mix-n-match" the installer and endpoint version that's being installed.

For reference, you can download the installer (`install.sh`), and observe what the shell script is going to do (highly recommen

```
curl -fsSL get.rebar.digital/stable -o install.sh
```

Once the installer is downloaded, you can execute it with the appropriate `install` options (try `bash ./install.sh --help` for details).

It is recommended that directory is used for this process. The `mkdir drp ; cd drp` command does this as the `drp` directory. The directory will contain all installed and operating files. The `drp` directory can be anything.

Even for *production* installs (without `--isolated`), it is recommended to run the `install.sh` script in a directory to contain all the install files for easy clean-up and removal if Digital Rebar Provision needs to be removed from the system.

3.1.4 Start Digital Rebar Provision service

Our quickstart uses *isolated* mode install, and the `dr-provision` service is not installed in the system path. You need to manually start `dr-provision` each time the system is booted up. The *production* mode installation (do not specify the `--isolated` install flag) will install in to system directories, and provide helpers to setup `init`, `systemd`, etc. start up scripts for the service.

Once the install has completed, your terminal should then display something like this (please use the output from YOUR install version, the below is just an example that may be out of date with the current versions output):

```
# Run the following commands to start up dr-provision in a local isolated␣
↪way.
# The server will store information and serve files from the ./drp-data␣
↪directory.

sudo ./dr-provision --base-root=`pwd`/drp-data --local-content="" --default-
↪content="" > drp.log 2>&1 &
```

Note: On MAC DARWIN there is one additional step. You may have to add a route for broadcast addresses to work. This can be done with following command `sudo route -n add -net 255.255.255.255 192.168.100.1` In this example, the `192.168.100.1` is the IP address of the interface that you want to send messages through. The install script should make suggestions for you.

The next step is to execute the *sudo* command which will start an instance of Digital Rebar Provision service that uses the `drp-data` directory for object and file storage.

Note: Before trying to install a BootEnv, please verify that the installed BootEnvs matches the above BootEnv Names that can be installed: `drpcli bootenvs list | jq '.[].Name'`

You may also use the RackN Portal UX by pointing your web browser to:

```
https://<ip_address_of_your_endpoint>:8092/
```

Please note that your browser will be redirected to the RackN Portal, pointing at your newly installed Endpoint. Use the below username/password pair to authenticate to the DRP Endpoint. Additional capabilities and features can be unlocked by also using the RackN Portal Login (upper right “Login” blue button).

The default username & password used for administering the *dr-provision* service is:

```
username: rocketskates
password: r0cketsk8ts
```

3.1.5 Install Boot Environments (bootenvs)

With Digital Rebar Provision running; it is now time to install the specialized Digital Rebar Provision content, and the required boot environments (BootEnvs). We generally refer to this as “content”.

Note: This documentation assumes you are using the default `drp-community-content` pack.

During the install step above, the installer output a message on how to install install BootEnvs. You must install the `sledgeham`

```
drpcli bootenvs list | jq '.[].Name'
```

1. install the *sledgehammer* Boot Environment, used for discovery and provisioning workflow
2. install the CentOS Boot Environment <optional>
3. install the Ubuntu Boot Environment <optional>

These steps should be performed from the newly installed *dr-provision* endpoint (or via remote *drpcli* binary with the use of the `--endpoint` flag):

```
drpcli bootenvs uploadiso sledgehammer
drpcli bootenvs uploadiso ubuntu-16.04-install
drpcli bootenvs uploadiso centos-7-install
```

The `uploadiso` command will fetch the ISO image as specified in the BootEnv JSON spec, download it, and then “explode” it in to the `drp-data/tftpboot/` directory for installation use. You may optionally choose one or both of the CentOS and Ubuntu BootEnvs (or any other Community Content supported BootEnv) to install; depending on which Operating System and Version you wish to test or use.

3.1.6 Configure a Subnet

A Subnet defines a network boundary that the DRP Endpoint will answer DHCP queries for. In this quickstart, we assume you will use the local network interface as a subnet definition, and that your Machines are all booted from the local subnet (layer 2 boundary). A Subnet specification must include all of the necessary DHCP boot options to correctly PXE boot a Machine.

Note: DRP supports the use of external DHCP servers, DHCP Proxy, etc. However, this is considered an advanced topic, and not discussed in the QuickStart.

Starting with Stable release v3.7.0 and newer, Digital Rebar Provision supports “magic” DHCP Boot Options for *next-server* and *bootfile* (option code 67). This means that these options should work “magically” for you without needing to be set.

HOWEVER - VirtualBox has a broken iPXE implementation.

If you are creating a subnet for an older version (before v3.7.0) of Digital Rebar Provision, you must set the *next-server* to your DRP Endpoint IP Address, and set the Option 67 value to `lpxelinux.0` for Legacy BIOS mode Machines.

If you are *using VirtualBox*, you must set the *next-server* value to the DRP Endpoint IP address _and_ the DHCP Option 67 value to `lpxelinux.0`

Note: The UX will create a Subnet based on an interface of the DRP Endpoint with sane defaults - it is easier to create a subnet via the UX.

If you are using a VirtualBox environment, and if you set the Name of the *Subnet* to `vboxnet0`, the UX will automatically correct the Option 67 bootfile value to support the broken iPXE environment for VirtualBox networks.

You must still set all of the remaining network values correctly in your Subnet specification, even in the UX.

To create a basic Subnet from command line we must create a JSON blob that contains the Subnet and DHCP definitions. Below is a _sample_ you can use. *PLEASE ENSURE* you modify the network parameters accordingly. Ensure you change the network parameters according to your environment.

```
###
#  EXAMPLE - please modify the below values according to your environment  !!
↪!
###

echo '{
  "Name": "local_subnet",
  "Subnet": "10.10.16.10/24",
  "ActiveStart": "10.10.16.100",
  "ActiveEnd": "10.10.16.254",
  "ActiveLeaseTime": 60,
  "Enabled": true,
  "ReservedLeaseTime": 7200,
  "Strategy": "MAC",
  "Options": [
    { "Code": 3, "Value": "10.10.16.1", "Description": "Default Gateway" },
    { "Code": 6, "Value": "8.8.8.8", "Description": "DNS Servers" },
    { "Code": 15, "Value": "example.com", "Description": "Domain Name" }
  ]
}' > /tmp/local_subnet.json

# edit the above JSON spec to suit your environment
#
# for v3.6.0 and older:
#  add a next-server after "Name" with the IP address of your DRP Endpoint,␣
↪like:
#    NextServer": "10.10.16.10",
#
# for v3.6.0 and older:
#  add DHCP Option 67 to the Options map, like:
#    { "Code": 67, "Value": "lpxelinux.0", "Description": "Bootfile" },
#
vim /tmp/local_subnet.json

drpcli subnets create - < /tmp/local_subnet.json
```

Note: Option 67 (bootfile name) specifies the PXE boot file. The *lpxelinux.0* boot file is for Legacy BIOS machines. If you are booting a UEFI system, you will need to make more advanced changes to support UEFI boot mode. Please see the FAQ on *UEFI Boot Support - Option 67*. DRP v3.7.0 and newer has magic helpers to try and set the Legacy/UEFI bootfile for you, but custom usage or custom/unique PXE implementations may require changes.

3.1.7 Create a Workflow

Workflows define a series of *Stages* that a Machine transitions through, driven by Digital Rebar Provision. Not only do the drive basic Operating System installation, but they also allow for advanced application installation and configuration if desired. *Workflows* also allow for some basic power management functions for hardware that does not support IPMI-like functions.

For our QuickStart use case, we'll create two simple *Workflows*:

1. Discovery
2. Operating System Install

1. Create the Discovery Workflow

```
# you must have installed the 'sledgehammer' ISO (see steps above)
drpcli workflows create '{ "Name": "discovery", "Stages": [ "discover",
↪"sledgehammer-wait" ] } '

# for packet.net environment, insert:     "packet-discover",
# between discover and sledgehammer-wait stages
# requires 'packet-ipmi' plugin provider installed and plugin configured
```

2. Now we will create the Installation workflow. You can select any OS that is supported in the `drp-community-content` package. Simply change the `Name` and the initial install Stage accordingly - use the following `jq` command to filter the list of Available BootEnvs for installation:

```
drpcli stages list | jq '.[] | select(.Available==true) | .Name' | grep "\-
↪install"
```

Using one of the *Available* BootEnvs as listed above, create, the OS install Workflow:

```
# example for CentOS 7
drpcli workflows create '{ "Name": "centos7", "Stages": [ "centos-7-install",
↪ "complete" ] } '

# example for Ubuntu 18.04
drpcli workflows create '{ "Name": "ubuntu18", "Stages": [ "ubuntu-18.04-
↪install", "complete" ] } '

# example for Debian 8
drpcli workflows create '{ "Name": "debian9", "Stages": [ "debian-9-install",
↪ "complete" ] } '

# in all cases for packet.net environemnt, insert:   "packet-ssh-keys",
# between the "install" and "complete" stages
# requires 'packet-ipmi' plugin provider installed and plugin configured
```

Note: You should receive a JSON blob back with the results of the command. It is important you check the *Errors* field for any messages. For example, if you see:

`"Stage debian-9-install is not available"`

Then the BootEnv *debian-9-install* is not installed or available, and this Workflow will not install an Operating System. Verify you successfully completed the `drpcli bootenvs uploadiso ...` steps outlined above.

3.1.8 Set The Defaults

One of the basic safety mechanisms for newly installed DRP Endpoints, is to prevent accidental Installation of a Machine, if it should PXE boot against a DRP Endpoint . . . **before** you are ready for that to happen!! So we must first set the default actions for a few system wide preferences. One of those defaults will point to our Discovery Workflow (see *Create a Workflow*).

Any Machine that boots will by default be placed in to the Discovery Workflow, which will NOT install an Operating System, but will enroll the machine for management by Digital Rebar Provision

Define the default Workflow, Default Stage, Default BootEnv, and the Unknown BootEnv:

```
# make sure you use the 'Name' specified in the Discovery Workflow,
# if you changed it from the default we specified
drpcli prefs set defaultWorkflow discovery unknownBootEnv discovery␣
↪defaultBootEnv sledgehammer defaultStage discover
```

Now any "unknown" or new Machines that boot against the DRP Endpoint will be *discovered* and sit idly by waiting (*sledgehammer-wait*) for your next commands (eg. install an operating system).

3.1.9 Install your first Machine

Content configuration is the most complex topic with Digital Rebar Provision. The basic provisioning setup with the above "ISO" uploads, default preferences, and simple workflows will allow you to install an operating system on the Machine with *manual power management* (on/off/reboot etc) transitions.

More advanced workflows and plugin_providers will allow for complete automation workflows with complex stages and state transitions. To keep things "quick", the below are just bare basics, for more details and information, please see the Content documentation section.

1. PXE Boot your Machine

    - ensure your test Machine is on the same Layer 2 subnet as your DRP endpoint, or that you've configured your networks *IP Helper* to forward your DHCP requests to your DRP Endpoint
    - the Machine should be in the same subnet as defined in the Subnets section above (not strictly required, but this is a simplified quickstart environment!)
    - set your test machine or VM instance to PXE boot
    - power the Machine on (or reboot it) and verify from the console that the NIC begins the PXE boot process
    - verify that the DRP Endpoint responds with a DHCP lease to the Machine

    The Machine should boot in to the Sledgehammer BootEnv - which will bring the console to a prompt that looks like (the version signature may differ):

    ```
    Digital Rebar: Sledgehammer␣
    ↪6122f34b46b5b74b668d6779e33f5fcd0f44a8cc
    Kernel 3.10.0-693.21.1.el7.x86_64 on an x86_64

    d0c-c4-7a-e5-48-b6 login:
    ```

2. Get your Machines UUID so you can set the Workflow for it

    - once your machine has booted, and received DHCP from the DRP Endpoint, it will now be "registered" with the Endpoint for installation

- by default, DRP will NOT attempt an OS install unless you explicitly direct it to (for safety's sake!)
- obtain your Machine's ID, you'll use it to define your BootEnv (see *Filter Out gohai-inventory* for more detailed/cleaner syntax)

```
drpcli machines list | jq '.[].Uuid'
```

3. Set the Workflow to to your Operating System Workflow you defined above; replace *<UUID>* with your machines ID from the above command:

```
# example for CentOS 7 workflow
drpcli machines update <UUID> '{ "Workflow": "centos7" }'

# example for Debian 9 workflow
drpcli machines update <UUID> '{ "Workflow": "debian9" }'

# example for Ubuntu 18.04 workflow
drpcli machines update <UUID> '{ "Workflow": "ubuntu18" }'
```

4. Reboot your Machine - it should now kick off a BootEnv install as you specified above.

- watch the console, and you should see the appropriate installer running
- the machine should reboot in to the Operating System you specified once install is completed

Note: Digital Rebar Provision is capable of automated workflow management of the boot process, power control, and much more. This quickstart walks through the simplest process to get you up and running with a single test install. Please review the rest of the documentation for further configuration details and information on automation of your provisioning environment.

3.1.10 More Advanced Workflow

The above procedure uses manual reboot of Machines, and manual application of the BootEnv definition to the Machine for final installation. A simple workflow can be used to achieve the same effect, but it is a little more complex to setup. See the *Digital Rebar Provision Operations* documentation for further details.

3.1.11 Machine Power Management

Fully automated provisioning control requires use of advanced RackN features (plugins) for Power Management actions. These are done through the IPMI subsystem, with a specific IPMI plugin for a specific environments. Some existing plugins exist for environments like:

- bare metal - hardware based BMC (baseboard management controller) functions that implement the IPMI protocol
- Virtual Box
- Packet bare metal hosting provider (https://www.packet.net/)
- Advanced BMC functions are supported for some hardware vendors (eg Dell, HP, IBM, etc)

Contact RackN for additional details and information.

3.1.12 Isolated vs Production Install Mode

The quickstart guide does NOT create a production deployment and the DRP Endpoint service will NOT restart on failure or reboot. You will have to start the *dr-provision* service on each system reboot (or add appropiate startup scripts).

A production mode install will install to `/var/lib/dr-provision` directory (by default), while an isolated install mode will install to `$PWD/drp-data`.

For more detailed installation information, see: *Install*

3.1.13 Clean Up

Once you are finished exploring Digital Rebar Provision in isolated mode, the system can cleaned by removing the directory containing the isolated install. In the previous sections, we used ''drp" as the directory containing the isolated install. Removing this directory will clean up the installed files.

For production deployments, the `install.sh` script can be run with the `remove` argument instead of the `install` argument to clean up the system. This will not remove the data files stored in `/var/lib/dr-provision`, `/etc/dr-provision`, or `/usr/share/dr-provision`. The `tools/install.sh` script is in the directory where you ran the `install.sh` script the first time. The script can be also redownloaded and run through curl | bash.

```
tools/install.sh remove
```

To additionally remove the data files, run instead:

```
tools/install.sh --remove-data remove
```

3.1.14 Ports

The Digital Rebar Provision endpoint service requires specific TCP Ports be accessible on the endpoint. Please see *Ports* for more detailed information.

If you are running in a Containerized environment, please ensure you are forwarding all of the ports appropriately in to the container. If you have a Firewall or packet filtering service on the node running the DRP Endpoint - ensure the appropriate ports are open.

3.1.15 Videos

We constantly update and add videos to the DR Provision Playlist so please check to make sure you have the right version!

3.2 Install

The install script does the following steps (in a slightly different order). See *Quick Start* for details about the script.

3.2.1 Get Code

The code is delivered by zip file with a sha256sum to validate contents. These are in github under the releases tab for the Digital Rebar Provision project.

There are at least 3 releases to choose from:

- **tip** - This is the most recent code. This is the latest build of master. It is bleeding edge and while the project attempts to be very stable with master, it can have issues.
- **stable** - This is the most recent **stable** code. This is a tag that tracks the version-based tag.
- **v3.0.0** - There will be a set of Semantic Versioning (aka semver) named releases.

Previous releases will continue to be available in tag/release history. For additional information, see *Releases*.

When using the **install.sh** script, the version can be specified by the **–drp-version** flag, e.g. *–drp-version=v3.0.0*.

An example command sequence for Linux would be:

```
mkdir dr-provision-install
cd dr-provision-install
curl -fsSL https://github.com/digitalrebar/provision/releases/download/tip/
↪dr-provision.zip -o dr-provision.zip
curl -fsSL https://github.com/digitalrebar/provision/releases/download/tip/
↪dr-provision.sha256 -o dr-provision.sha256
sha256sum -c dr-provision.sha256
unzip dr-provision.zip
```

At this point, the **install.sh** script is available in the **tools** directory. It can be used to continue the process or continue following the steps in the next sections. *tools/install.sh –help* will provide help and context information.

3.2.2 Configuration Options

Using `dr-provision --help` will provide the most complete list of configuration options. The following common items are provided for reference. Please note these may change from version to version, check the current scripts options with the `--help` flag to verify current options.

```
--version                  Print Version and exit
--disable-provisioner      Disable provisioner
--disable-dhcp             Disable DHCP
--static-port=             Port the static HTTP file server should listen on␣
↪(default: 8091)
--tftp-port=               Port for the TFTP server to listen on (default: 69)
--api-port=                Port for the API server to listen on (default: 8092)
--dhcp-port=               Port for the DHCP server to listen on (default: 67)
--backend=                 Storage backend to use. Can be either 'consul' or
↪'directory' (default: directory)
--data-root=               Location we should store runtime information in␣
↪(default: /var/lib/dr-provision)
--static-ip=               IP address to advertise for the static HTTP file␣
↪server (default: 192.168.124.11)
--file-root=               Root of filesystem we should manage (default: /var/
↪lib/tftpboot)
--dhcp-ifs=                Comma-seperated list of interfaces to listen for␣
↪DHCP packets
--debug-bootenv=           Debug level for the BootEnv System - 0 = off, 1 =␣
↪info, 2 = debug (default: 0)
--debug-dhcp=              Debug level for the DHCP Server - 0 = off, 1 = info,
↪ 2 = debug (default: 0)
--debug-renderer=          Debug level for the Template Renderer - 0 = off, 1␣
↪= info, 2 = debug (default: 0)
--tls-key=                 The TLS Key File (default: server.key)
--tls-cert=                The TLS Cert File (default: server.crt)
```

3.2.3 Prerequisites

dr-provision requires two applications to operate correctly, **bsdtar** and **7z**. These are used to extract the contents of iso and tar images to be served by the file server component of **dr-provision**. The `install.sh` script will attempt to ensure these packages are installed by default. However, if you are installing via manual process or baking your own installer, you must ensure these prerequisistes are met.

For Linux, the **bsdtar** and **p7zip** packages are required.

ubuntu

sudo apt-get install -y bsdtar p7zip-full

centos/redhat

sudo yum install -y bsdtar p7zip

Darwin

The new package, **p7zip** is required, and **tar** must also be updated. The **tar** program on Darwin is already **bsdtar**

- 7z - install from homebrew: `brew install p7zip`
- libarchive - update from homebrew to get a functional tar: `brew install libarchive --force ; brew link libarchive --force`

At this point, the server can be started.

Note: In a future release, the required packages may be removed, which will help ensure cross-platform compatibility without relying on these external dependencies.

3.2.4 Running The Server

Additional support materials in *FAQ / Troubleshooting*.

The **install.sh** script provides two options for running **dr-provision**.

The default values install the server and cli in /usr/local/bin. It will also put a service control file in place. Once that finishes, the appropriate service start method will run the daemon. The **install.sh** script prints out the command to run and enable the service. The method described in the *Quick Start* can be used to deploy this way if the *–isolated* flag is removed from the command line. Look at the internals of the **install.sh** script to see what is going on.

Note: The default location for storing runtime information is `/var/lib/dr-provision` unless overridden by `--data-root`

Alternatively, the **install.sh** script can be passed the *–isolated* flag and it will setup the current directory as an isolated "test drive" environment. This will create a symbolic link from the bin directory to the local top-level directory for the appropriate OS/platform, create a set of directories for data storage and file storage, and display a command to run. This is what the *Quick Start* method describes.

The default username & password used for administering the *dr-provision* service is:

```
username: rocketskates
password: r0cketsk8ts
```

Please review *–help* for options like disabling services, logging or paths.

Note: sudo may be required to handle binding to the TFTP and DHCP ports.

Once running, the following endpoints are available:

- https://127.0.0.1:8092/swagger-ui - swagger-ui to explore the API
- https://127.0.0.1:8092/swagger.json - API Swagger JSON file
- https://127.0.0.1:8092/api/v3 - Raw api endpoint
- https://127.0.0.1:8092/ux - Redirects to Community Portal (maintained by RackN)
- http://127.0.0.1:8091 - Static files served by http from the *test-data/tftpboot* directory
- udp 69 - Static files served from the test-data/tftpboot directory through the tftp protocol
- udp 67 - DHCP Server listening socket - will only serve addresses when once configured. By default, silent.
- udp 4011 - BINL Server listening socket - will only serve bootfiles when once configured. By default, silent.

The API, File Server, DHCP, BINL, and TFTP ports can be configured, but DHCP, BINL, and TFTP may not function properly on non-standard ports.

If the SSL certificate is not valid, then follow the *Generate Certificate* steps.

Note: On MAC DARWIN there is one additional step. You may have to add a route for broadcast addresses to work. This can be done with the following comand. The 192.168.100.1 is the IP address of the interface that you want to send messages through. The install script will make suggestions for you.

```
sudo route add 255.255.255.255 192.168.100.1
```

3.2.5 Production Deployments

The following items should be considered for production deployments. Recommendations may be missing so operators should use their best judgement.

Start DRP Without Root (or sudo)

If you are using DHCPD and TFTPD services of DRP, you will need to be able to bind to port 67 and 69 (respectively). Typically Unix/Linux systems require root privileges to do this. DRP doesn't start as root, and then drop privileges with a `fork()` to another less privileged user by default.

To enable DRP endpoint to run as a non-privileged user and ensure a higher level of security, it's possible to use the Linux "*setcap*" (Capabilities) system to assign rights for the *dr-provision* binary to open low numbered (privileged) ports. The process is relatively simple, but does (clearly/obviously) require root permissions initially to enable the capabilities for the binary. Once the capabilities have been set, the *dr-provision* binary can be run as a standard user.

To enable any non-privileged user to start up the dr-provision binary and bind to privileged ports 67 and 69, do the following:

# in "isolated" mode, as the user you installed DRP as:

```
sudo setcap "cap_net_raw,cap_net_bind_service=+ep" $HOME/bin/linux/amd64/dr-
↪provision
```

or, in "production" mode:

```
sudo setcap "cap_net_raw,cap_net_bind_service=+ep" /usr/local/bin/dr-provision
```

Start the "dr-provision" binary as an ordinary user, and now it will have permission to bind to privileged ports 67 and 69.

Note: The *setcap* command must reference the actual binary itself, and can not be pointed at a symbolic link. Additional refinement of the capabilities may be possible. For extremely security conscious setups, you may want to refer to the StackOverflow discussion (eg setting capabilities on a per-user basis, etc.): https://stackoverflow.com/questions/1956732/is-it-possible-to-configure-linux-capabilities-per-user

System Logs

The Digital Rebar Provision service logs by sending output to standard error. To capture system logs, SystemD (or Docker) should be configured to direct this output to the desired log management infrastructrure.

Job Log Rotation

If you are using the jobs system, Digital Rebar Provision stores job logs based on the directory configuration of the system. This data is considered compliance related information; consequently, the system does not automatically remove these records.

Operators should set up a job log rotation mechanism to ensure that these logs to not exhaust available disk space.

Removal of Digital Rebar Provision

To remove Digital Rebar Provision, you can use the *tools/install.sh* script to remove programs for a `production` installs. The *tools/install.sh* script should be run as root or under sudo unless the `setcap` process was used.

```
tools/install.sh remove
```

To remove programs and data use.

```
tools/install.sh --remove-data remove
```

For *iolated* installs, remove the directory used to contain the isolated install. In the example above, the directory *dr-provision-install* was used to isolate the install process. A command like this would clean up the system.

```
sudo rm -rf dr-provision-install
```

3.3 Environment Setup Instructions

This document indexes resources for setting up specific environments for use or testing of Digital Rebar Provision with various tools.

- rs_setup_packet

- VirtualBox _(not completed yet)_
- Local KVM _(not completed yet)_

3.4 Key Features

Digital Rebar Provision is a new generation of data center automation designed for operators with a cloud-first approach. Data center provisioning is surprisingly complex because it's caught between cutting edge hardware and arcane protocols embedded in firmware requirements that are still very much alive.

3.4.1 Swagger REST API & CLI

Cloud-first means having a great, tested API. Years of provisioning experience went into this 3rd generation design and it shows. That includes a powerful API-driven DHCP server.

3.4.2 Security & Authenticated API

Not an afterthought, we use both HTTPS and user authentication for the API. Our mix of basic and bearer token authentication recognizes that both users and automation will use the API. This brings a new level of security and control to data center provisioning.

3.4.3 Stand-alone multi-architecture Golang binary

There are no dependencies or prerequisites, plus upgrades are drop in replacements. That allows users to experiment isolated on their laptop and then easily register it as a SystemD service.

3.4.4 Nested Template Expansion

In Digital Rebar Provision, Boot Environments are composed of reusable template snippets. These templates can incorporate global, profile or machine specific properties that enable users to set services, users, security or scripting extensions for their environment. Configuration at Global, Group/Profile and Node level. Properties for templates can be managed in a wide range of ways that allows operators to manage large groups of servers in consistent ways.

3.4.5 Multi-mode (but optional) DHCP

Network IP allocation is a key component of any provisioning infrastructure; however, DHCP needs are highly site dependent. Digital Rebar Provision works as a multi-interface DHCP listener and can also resolve addresses from DHCP forwarders. It can even be disabled if the environment already has a DHCP service that can configure a the "next boot" provider.

3.4.6 Dynamic Provisioner templates for TFTP and HTTP

For security and scale, Digital Rebar Provision builds provisioning files dynamically based on the Boot Environment Template system. This means that critical system information is not written to disk and files do not have to be synchronized. Of course, when a file needs to be served it works too.

3.4.7 Node Discovery Bootstrapping

Digital Rebar's long-standing discovery process is enabled in the Provisioner with the included discovery boot environment. That process includes an integrated secure token sequence so that new machines can self-register with the service via the API. This eliminates the need to pre-populate the Digital Rebar Provision system.

3.4.8 Multiple Seeding Operating Systems

Digital Rebar Provision comes with a long list of Boot Environments and Templates including support for many Linux flavors, Windows, ESX and even LinuxKit. Our template design makes it easy to expand and update templates even on existing deployments.

3.4.9 Two-stage TFTP/HTTP Boot

Our specialized Sledgehammer and Discovery images are designed for speed with optimized install cycles the improve boot speed by switching from PXE TFTP to IPXE HTTP in a two stage process. This ensures maximum hardware compatibility without creating excess network load.

3.5 Server Architecture

Digital Rebar Provision is provided by a single binary that contains tools and images needed to operate. These are expanded on startup and made available by the file server services.

3.5.1 Services

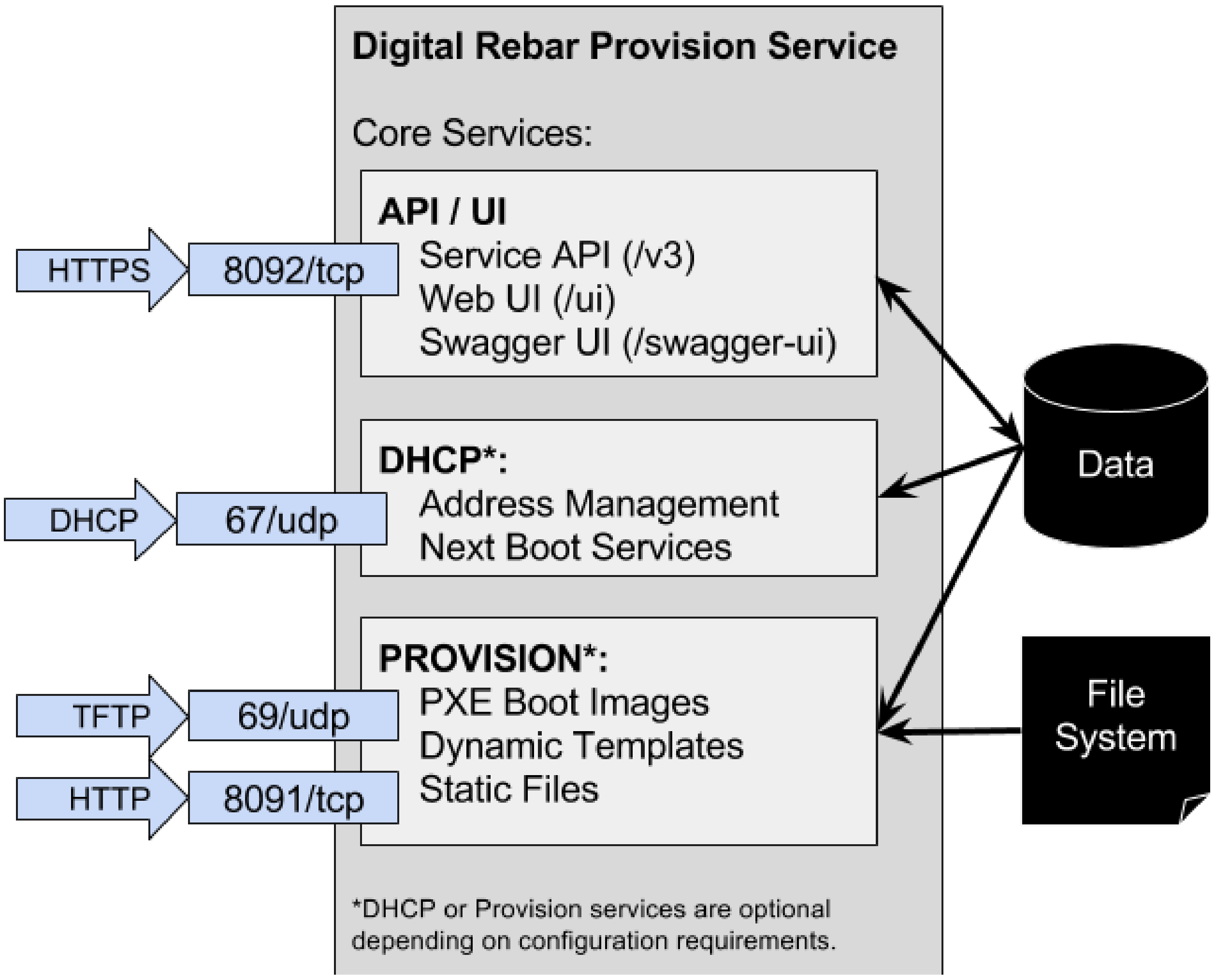
Provisioning requires handoffs between multiple services as described in the *Digital Rebar Provision Workflows* section. Since several of services are standard protocols (DHCP, TFTP, HTTP), it may be difficult to change ports without breaking workflow.

The figure below illustrates the three core Digital Rebar Provision services including protocols and default ports. The services are:

1. Web - These services provide control for the other services
   (a) API: REST endpoints with Swagger definition
   (b) UI: User interface and Swagger helpers
2. DHCP: Address management includes numerous additional option fields used to tell systems how to interact with other data center services such as provisioning, DNS, NTP and routing.
3. Provision: sends files on request during provisioning process based on a template system:
   (a) TFTP: very simple (but slow) protocol that's used by firmware boot processes because it is very low overhead.
   (b) HTTP: faster file transfer protocol used by more advanced boot processes

3.5.2 Ports

The table describes the ports that need to be available to run Digital Rebar Provision. Firewall rules may need to be altered to enable these services. The feature column indicates when the port is required. For example, the DHCP server can be turned off and that port is no longer required.


Digital Rebar Provision Service
Core Services:
API / UI
Service API (/v3)
Web UI (/ui)
Swagger UI (/swagger-ui)
DHCP*:
Address Management
Next Boot Services
PROVISION*:
PXE Boot Images
Dynamic Templates
Static Files
*DHCP or Provision services are optional depending on configuration requirements.
HTTPS
8092/tcp
DHCP
67/udp
TFTP
69/udp
HTTP
8091/tcp
Data
File System

Ports	Feature	Usage
67/udp	DHCP	DHCP Port
69/udp	PROV	TFTP Port
4011/udp	BINL	PXE/BINL port
8091/tcp	PROV	HTTP-base File Server
8092/tcp	Always	API and Swagger-UI

All default ports can be changed at start up time of the `dr-provision` service. NOTE that changing DHCP and TFTP ports has wide ranging implications and is likely not a good idea (many firmware implementations can not be changed to use alternate port numbers).

Port access requirements:

In all usage cases (67, 69, 4011, 8091, and 8092) the ports *from* the Machines being provisioned *to* the DRP Endpoint must be accessible. The DRP Endpont must be able to reach the Machines being provisioned on port 67 for In addition, the API and Swagger-UI port must be accessible to any operator/administrator workstations or systems that are controlling and managing the DRP Endpoint service. Additionally any services or integrations that interact with the DRP Endpoint (eg IPAM, DCIM, Asset Management, CMS, CMDB, etc) may need access to the API port.

3.6 Configuring the Server

Digital Rebar Provision provides both DHCP and Provisioning services but can be run with either disabled. This allows users to work in environments with existing DHCP infrastructure or to use Digital Rebar Provision as an API driven DHCP server.

3.6.1 DHCP Server (subnets)

The DHCP server is configured to enable Subnets that serve IPs and/or additional configuration information. It is possible to run the DHCP server using only predefined IP Reservations or allow the DHCP server to create IP Leases dynamically.

The DHCP server has three primary models

1. DHCP Listeners can be set on an IP for each server interface. These listeners will respond to DHCP broadcasts on the matching network(s). Operators should ensure that no other DHCP servers are set up on the configured subnets.
2. DHCP Relay allows other DHCP listeners to forward requests to the Digital Rebar Provision server. In this mode, the server is passive and can easily co-exist with other DHCP servers. This mode works with the Provisioner by setting the many optional parameters (like next boot) that are needed for PXE boot processes.
3. Proxy DHCP Mode - this allows for other DHCP servers to forward requests to the DRP Endpoint, and fillin necessary PXE related options that may be missing from the original DHCP request options. In some environments, very limited DHCP servers may be in use (which do not support the appropriate Options), or a DRP Endpoint user may not have administrative authority over the initial DHCP server in use. In this case, the rs_model_subnet should include the following configuration to enable Proxy DHCP mode:

   ```
   drpcli subnets update mysubnet '{ "Proxy": true }'
   ```

3.6.2 Provisioner (bootenvs)

The Provisioner is a combination of several services and a template expansion engine. The primary model is a boot environment (BootEnv) that contains critical metadata to describe an installation process. This metadata includes templates that are dynamically expanded when machines boot.

Digital Rebar Provision CLI has a process that combines multiple calls to install BootEnvs. The following steps will configure a system capable to *Provision O/S on Discovered Systems*.

```
drpcli bootenvs uploadiso sledgehammer
drpcli prefs set defaultStage "discover" unknownBootEnv "discovery"␣
↪defaultBootEnv "sledgehammer"
```

In addition to the basic *discovery* capability provided by the *sledgehammer* BootEnv, you will most likely want to install an Operating System related BootEnv. Basic example for this is as follows:

```
drpcli bootenvs uploadiso centos-7-install
drpcli bootenvs uploadiso ubuntu-16.04-install
```

Each content pack has various supported operating system BootEnv definitions. The default *drp-community-content* pack contains the following BootEnvs:

1. centos-7-install: CentOS 7 (most recent released version)
2. centos-7.4.1708-install: Centos 7.4.1708 (this may change as new versions are released)
3. ubuntu-16.04-install: Ubuntu 16.04
4. debian-8: Debian 8 (Jessie) version
5. debian-9: Debian 9 (Sarge) version

Additional Operating System versions are available via registered RackN content pack add-ons.

3.6.3 Default Template Identity

These settings apply to TEMPLATES only not the API.

The default password for the default o/s templates is **RocketSkates**

The default user for the default ubuntu/debian templates is **rocketskates**

3.7 Releases

3.7.1 Version Scheme

There are two named releases and many semantic versioned releases.

- **tip** - This is the currently built master code. It follows the bleeding edge of the trees.
- **stable** - This is the default release used by install. It follows to most recently released code.

The other releases follow the v<Major>.<Minor>.<Patch> format. e.g. **v3.0.0**

The *Install* process will default to **stable** but can be modified to use other releases.

Docs are also generated for these as well.

The full version string generated by the build tool looks like this:

- v3.0.0-tip-galthaus-dev-19-d012b1d6ba892c96c81c24ee93b668591663ca5c
- v3.0.0-tip-16-e1fa235dc28f7d278bc34a0d4db87c306d9d3ba8
- v3.0.0-0-821108c416dfc1486919baa52dae284975d2ad8b

The base form is base semver tag, extra pieces, gits ahead of that base, and the actual git commit checksum.

The extra pieces is either -tip to indicate that the build comes from the tip of the tree or -tip-<username>-dev where the username is who built the custom build that is ahead of tip at the time.

The *dr-provision* server reports its version on start-up or can be queried by running:

```
dr-provision --version
```

The *drpcli* client reports its version by running:

```
drpcli version
```

3.7.2 Release Process

The team will regularly move the **tip** tag to track the leading edge of the **master** branch. This will make the resulting build available in the releases tab.

Additionally, as releases cut at stable points, the **stable** tag will be set to the new most recent milestone. Release notes and comments will be added to the github release page.

3.7.3 Release Notes

Each of the release specific changes are annotated in the Release Notes documentation. You can find the release notes at:

> https://github.com/digitalrebar/provision/releases

3.8 Upgrade

While not glamorous, existing code can be overwritten by a new install and restart. That is about it. Here are a few more details.

We recommend that you backup your existing install as a safey measure.

3.8.1 Backup

It's always a good policy to backup any important data, configuration, and content information that may be related to an application before an upgrade. We strongly encourage you to backup your content prior to doing any upgrade activity.

Isolated Install

For "isolated" modes (eg. originally installed with something like `install.sh | bash -s -- install --isolated`) , perform the following tasks:

1. log in to your Provision server as the user you performed the original install as

2. copy the drp-data directory to a backup location:

```
D=`date +%Y%m%d-%H%M%S`
cp -r drp-data drp-data.backup.$D
```

Production Install

For "production" install modes (no `--isolated` flag provided to `install.sh`), perform the following tasks

1. log in to your Digital Rebar Provision server
2. for DRP version 3.0.5 or older:

```
D=`date +%Y%m%d-%H%M%S`
mkdir backups
sudo cp -r /var/lib/dr-provision backups/dr-provision.backup.$D
sudo cp -r /var/lib/tftpboot backups/dr-provision.backup.$D
```

3. for DRP version 3.1.0 or newer:

```
D=`date +%Y%m%d-%H%M%S`
mkdir backups
sudo cp -r /var/lib/dr-provision backups/dr-provision.backup.$D
```

3.8.2 Upgrade Steps

The basic steps are the same for both Isolated and Production install modes:

1. stop the existing service
2. run the installer with the "upgrade" flag
3. restart the service

Isolated Install

For isolated *Install*, update this way:

1. Stop dr-provision:

```
killall dr-provision
```

2. Return to the install directory
3. Run the install again

```
# Remember to use --drp-version to set a version other than stable if desired
# Curl/Bash from quickstart if desired, or this:
tools/install.sh upgrade --isolated
```

4. Restart *dr-provision*, as stated by the `tools/install.sh` output.

Production Install

For non-isolated (aka "production mode") *Install*, update this way:

1. Stop dr-provision, using the system method of choice

   ```
   systemctl stop dr-provision
   ```

   or

   ```
   service dr-provision stop
   ```

2. Install new code - Use the same install technique as the first install, but change `install` to `upgrade` option. *Install*

   ```
   tools/install.sh upgrade --isolated
   ```

3. Start up dr-provision

   ```
   systemctl start dr-provision
   ```

   or

   ```
   service dr-provision start
   ```

3.8.3 Version to Version Notes

In this section, notes about migrating from one release to another will be added.

Release Notes for each version can be found at: https://github.com/digitalrebar/provision/releases

v3.0.0 to v3.0.1

If parameters were added to machines or global, these will need to be manually re-added to the machine or global profile, respectively. The machine's parameter setting cli is unchanged. The global parameters will need to be changed to a profiles call.

```
drpcli parameters set fred greg
```

to

```
drpcli profiles set global fred greg
```

v3.0.1 to v3.0.2

There are changes to templates and bootenvs. Upgrade will not update these automatically, because they may be in use and working properly. it is necessary to restart by removing the bootenvs and templates directory in the data store directory (usually drp-data/digitalrebar or /var/lib/dr-provision/digitalrebar) and re-uploading the bootenvs and templates (tools/discovery-load.sh). Additionally, templates and bootenvs can be manually added and updated, with drpcli.

v3.0.2 to v3.0.3

This is a quick turn release to address the issue with updating bootenvs. This is a CLI code and docs only change.

v3.0.3 to v3.0.4

Nothing needs to be done.

v3.0.4 to v3.0.5

Nothing needs to be done.

v3.0.5 to v3.1.0

Release Notes for v3.1.0

The v3.1.0 `install.sh` script now supports an `--upgrade` flag. Depending on your installation method (eg `isolated` or `production` mode), the behavior of the flag will alter the installation process slightly. Please ensure you *Backup* your content and configurations first just in case.

For `isolated` mode:

```
install.sh --upgrade --isolated install
```

Note: You must be in the same directory path that you performed the initial install from for the upgrade to be successful.

For `production` mode:

The `production` mode update process will move around several directories and consolidate them to a single location. In previous versions (v3.0.5 and older), the following two default directories were used in `production` mode:

```
/var/lib/dr-provision - Digital Rebar Provision configurations and␣
↪information
/var/lib/tftpboot - TFTP boot root directory for serving content when TFTPD␣
↪service enabled
```

In DRP v3.1.0 and newer, the content will be moved by the `--upgrade` function as follows:

```
/var/lib/dr-provision/digitalrebar - old "dr-provision" directory
/var/lib/dr-provision/tftpboot - old "tftpboot" directory
```

Note: Digital Rebar Provision version 3.1.0 introduced a new behavior to the `subnets` definitions. `subnets` may now be `enabled` or `disableed` to selectively turn on/off provisioning for a given subnet. By default, a subnet witll be disabled. After an upgrade, you MUST enable the subnet for it to function again. See *Subnet Enabled* for additional details.

Subnet Enabled

Starting in v3.1.0, subnet objects have an enabled flag that allows for subnets to be turned off without deleting them. This value defaults to false (off). To enable existing subnets, you will need to do the following for each subnet in your system:

```
drpcli subnets update subnet1 '{ "Enabled": true }'
```

Replace *subnet1* with the name of your subnet. You may obtain a list of configured subnets with:

```
drpcli subnets list | jq -r '.[].Name'
```

v3.1.0 to v3.2.0

Release Notes for v3.2.0

There are fairly significant updates to the DRP Contents structure and layout in v3.2.0. If you are upgrading to v3.2.0 you must remove any Digital Rebar and RackN content that you have installed in your Provisioning endpoint. The following outline will help you understand the necessary steps. If you have any issues with the upgrade process, please drop by the Slack #community channel for additional help.

Please read the steps through carefully, and make note of the current contents/plugins you currently have installed. You will have to re-add these elements again. You absolutely should backup your existing install prior to this upgrade.

1. Overview

   Overiew of the update steps necessary, you should do in the following order.

   1. Update DRP to "stable" (v3.2.0)
   2. Remove Old Content
   3. Add Content back that was removed
   4. Update plugins
   5. Fix up things

2. Updating DRP Endpoint

   If you are running isolated, do this (remove `--isolated` if you are not using isolated mode):

   ```
   curl -fsSL get.rebar.digital/stable | bash -s -- upgrade --isolated
   ```

   This will force the update of the local binaries to v3.2.0 stable. Make sure you stop DRP process (`sudo killall dr-provision`, or `sudo systemctl stop dr-provision.service`).

   Verify that your `/etc/systemd/services/dr-provision` start up file is still correct for your environment, if running a production install type.

   Restart DRP (follow `--isolated` mode start steps if in isolated mode; or `sudo systemctl start dr-provision.service`)

   If in `--isolated` mode, don't forget to copy `drpcli` and/or `dr-provision` binaries to where you prefer to keep them (eg `$HOME/bin` or `/usr/local/bin`, etc… .

3. Remove old content

   With the rework of content, you need to remove the following content packages if they were previously installed.

   ```
   os-linux
   os-discovery
   drp-community-content (if you are really behind, Digital Rebar␣
   ↪Community Content).
   ipmi
   ```

(continues on next page)

(continued from previous page)

```
packet
virtualbox
```

4. Put the content back

   Install the new v3.2.0 content packs. Note that the names have changed, and the mix of "ce-" and non-Community Content names has gone away. For example; what originally was `drp-community-content` which included things like `ce-sledgehammer` is now moved to just `sledgehammer`. The RackN registered content of `os-linux` and `os-discovery` have now been folded in to the below content packs.

   ```
   drp-community-content - it is a must just get it.
   task-library - New RackN library of services for doing interesting␣
   ↪things.
   drp-community-contrib - this is old or experimental things like␣
   ↪centos6 or SL6.
   ```

5. Update the plugins

   If you have any plugins installed, update them now.

   To facilitate version tracking, plugins provide their own content as a injected content from the plugin. When the plugin is added, it will also add a content layer that will show up in the content packages section.

   Previously, a `plugin-provider` was installed separately from a Content of the same name.

6. Fix things up

   This is mainly if you were using the Community Content version of things (`drp-community-content`, and BootEnvs with a prefix of `ce-`). The BootEnvs names change, by removing the prefix of "ce-" from the name.

   Make sure all the bootenvs are up to date and available. This is a task you should always do after updating content. If the BootEnv is marked with an "X" in the UX, or `"Available": false` from the CLI/API, you'll need to reload the ISO for the BootEnv.

   Then go to *Info & Preferences* and make sure your default stage and bootenvs are still valid.

   - This is where `ce-sledgehammer` becomes `sledgehammer` and `ce-discovery` becomes `discovery`
   - The same with `ce-ubuntu-16.04-install` becomes `ubuntu-16.04-install`.
   - The same with `ce-centos-7.4.1708-install` becomes `centos-7-install`.

Example pseudo-script to make changes:

Please carefully read through this script and make sure it correlates to your installed content. It is provided only as an example, and will absolutely require (possibly just minor) modifications for your environment.

YOU MUST MODIFY THE *RACK_AUTH* variable appropriately for the download authentication to work correctly.

```
# see all contents
drpcli contents list

# list JUST the names of the contents - note what you have installed,
```

(continues on next page)

(continued from previous page)

```
# you may need to re-install it below
drpcli contents list | jq -r '.[].meta.Name' | egrep -v
↪"BackingStore|BasicStore"

# list which plugins you have installed - note it, you may need to␣
↪install
# it below
drpcli plugin_providers list | jq '.[].Name'

# go to RackN UX - log in, go to Hamburger menu (upper left, 3␣
↪horizontal lines)
# go to Organization - User Profile - copy your UUID for Unique User␣
↪Identity
export RACKN_AUTH="?username=<UUID_Unique_User_Identity>"
export CATALOG="https://qww9e4paf1.execute-api.us-west-2.amazonaws.
↪com/main/catalog"

# get raw output of just the content packs
for CONTENT in `drpcli contents list | jq -r '.[].meta.Name' | egrep␣
↪-v "BackingStore|BasicStore"`
do
  echo "remove content:   $CONTENT"
  drpcli contents destroy $CONTENT
done

# install content
for CONTENT in drp-community-content task-library drp-community-
↪contrib
do
  echo "install content:  $CONTENT"
  curl -s $CATALOG/content/${CONTENT}${RACKN_AUTH} -o $CONTENT.json
  drpcli contents create -< $CONTENT.json
done

# change "plug1", "plug2", etc... to the plugin provider names you␣
↪need
# examples:  "slack", "packet-ipmi", "ipmi"
for PLUGIN in plug1 plug2 plug3
do
  echo "install plugin:  $PLUGIN"
  curl -s $CATALOG/plugin/${PLUGIN}${RACKN_AUTH} -o $PLUGIN.json
  drpcli contents create -< $PLUGIN.json
done

# Ensure the Stage, Default, and Unknown BootEnv are set to valid␣
↪values
# adjust these as appropriate
drpcli prefs set defaultStage discover defaultBootEnv sledgehammer␣
↪unknownBootEnv discovery
```

Again - make sure you modify things appropriately in the above scriptlet. YOU MUST MODIFY THE *RACK_AUTH* variable appropriately for the download authentication to work correctly.

v3.2.0 to v3.3.0

Release Notes for v3.3.0

No aditional steps required.

v3.3.0 to v3.4.0

Release Notes for v3.4.0

Content Changes

Prior to restart Digital Rebar Provision endpoint - you may need to fix the Machines JSON entries for the `Meta` field. It used to

```
dr-provision2018/01/07 15:14:01.275082 Extracting Default Assets
panic: assignment to entry in nil map
```

To correct the problem, you will need to edit your JSON configuration files for your Machines. You can find your Machines spec files in `/var/lib/dr-provision/digitalrebar/machines` if you are running in *production* mode install. If you are running in *isolated* mode, you will need to locate your `drp-data` directory which is in the base directory where you performed the install at; the machines directory will be `drp-data/digitalrebar/machines`.

There may be two `Meta` tags. You do NOT need to modify the `Meta` tag that is located in the *Params* section.

Change the first `Meta` tag as follows:

```
# from:
"Meta":null,

# to something like:
"Meta":{"feature-flags":"change-stage-v2"},
```

It is entirely possible that the `Meta` field is completely missing. If so - inject the full `Meta` field as specified above.

`drpcli` changes

Please see the Release Notes for information related to the `drpcli` command line changes. The most notable changes that may impact your use (eg in existing scripts) of the tool:

1. Plugin upload method changed:

```
# prior to v3.4.0
drpcli plugin_providers upload $PLUGIN as $PLUG_NAME

# v3.4.0 and newer version method:
drpcli plugin_providers upload $PLUG_NAME from $PLUGIN
```

2. Many commands now have new *helper* capabilities. See each command outputs relevant help statement.

v3.4.0 to v3.5.0

Release Notes for v3.5.0

No additional changes necessary.

v3.5.0 to v3.6.0

Release Notes for v3.6.0

No additional changes necessary.

v3.6.0 to v3.7.0

Release Notes for v3.7.0

The plugin system has been updated to a new version. All plugins have been updated to use the new version. After updating to *v3.7.0*, all plugins must be updated to function. The system will start after update, but the plugin-providers will not load until they are udpated. Use the RackN UX to get the updates for the plugins.

The Task subsystem has been updated to default to *sane-exit-codes*. This is a change from the default of *original-exit-codes*. This was done to address the need of task authors to match some basic assumptions about exit codes. *1* should be a fail and not reboot your box.

Additionally, the default UX redirect has changed to the stable portal. This will result in more stable UX experience.

v3.7.0 to v3.8.0

Release Notes for v3.8.0

No additional changes necessary.

v3.8.0 to v3.9.0

Release Notes for v3.9.0

No additional changes necessary.

3.9 Local UI Removed

The old UI has been removed and a redirect to the RackN Portal UI is present instead. The UI loads into the browswer and then uses the API to access the Endpoint. The DRP endpoint does not talk to the internet. The browser acts as a bridge for content transfers. The only requirement is that the browser has access to the Endpoint and HTTPS-based access to the internet. The HTTPS-based access can be through a web proxy.

3.10 Install Script Changed

There are minor changes to the install script for isolated mode. Production mode installs are still done and updated the same way. For isolated, there are some new flags and options. Please see the commands output for more details or check the updated *Quick Start*.

For current `install.sh` script usage information, please run:

```
install.sh --help
```

For complete details.

3.11 Digital Rebar Provision Workflows

PXE Provisioning requires hand-offs between multiple services and the nodes. Generally, these flows start with network discovery (DHCP) providing a "next boot" instruction to the server's pre-execution environment (PXE) that can download a very limited operating system using a very simple file transfer protocol (TFTP). That next operating system can then take more complex actions including using HTTP to download further files.

Note: Digital Rebar Provision does not provide out-of-band management or orchestration in the Open Source version. See the RackN based advanced plugins for power management capabilities.

In the flow graphics, green steps indicate user actions.

3.11.1 Basic Discovery

The basic discovery flow uses the specialized Discovery and Sledgehammer Boot Environments (BootEnvs). These BootEnvs work with the default next boot to self-register machines with Digital Rebar Provision using the Machines API.

This recommended process allows operators to start with minimal information about the environment so they can *Provision O/S on Discovered Systems*. Without discovery, operators must create machines in advance following the *Provision O/S on Known Systems* process.

3.11.2 Provision O/S on Discovered Systems

Once systems have been discovered using *Basic Discovery*, their machine BootEnv can be set to an O/S installation image. The recommended BootEnvs for these images includes an automatic machines API call that sets the machine to boot to local disk (BootEnv = local).

3.11.3 Provision O/S on Known Systems

note This workflow requires users to know their machine information in advance.

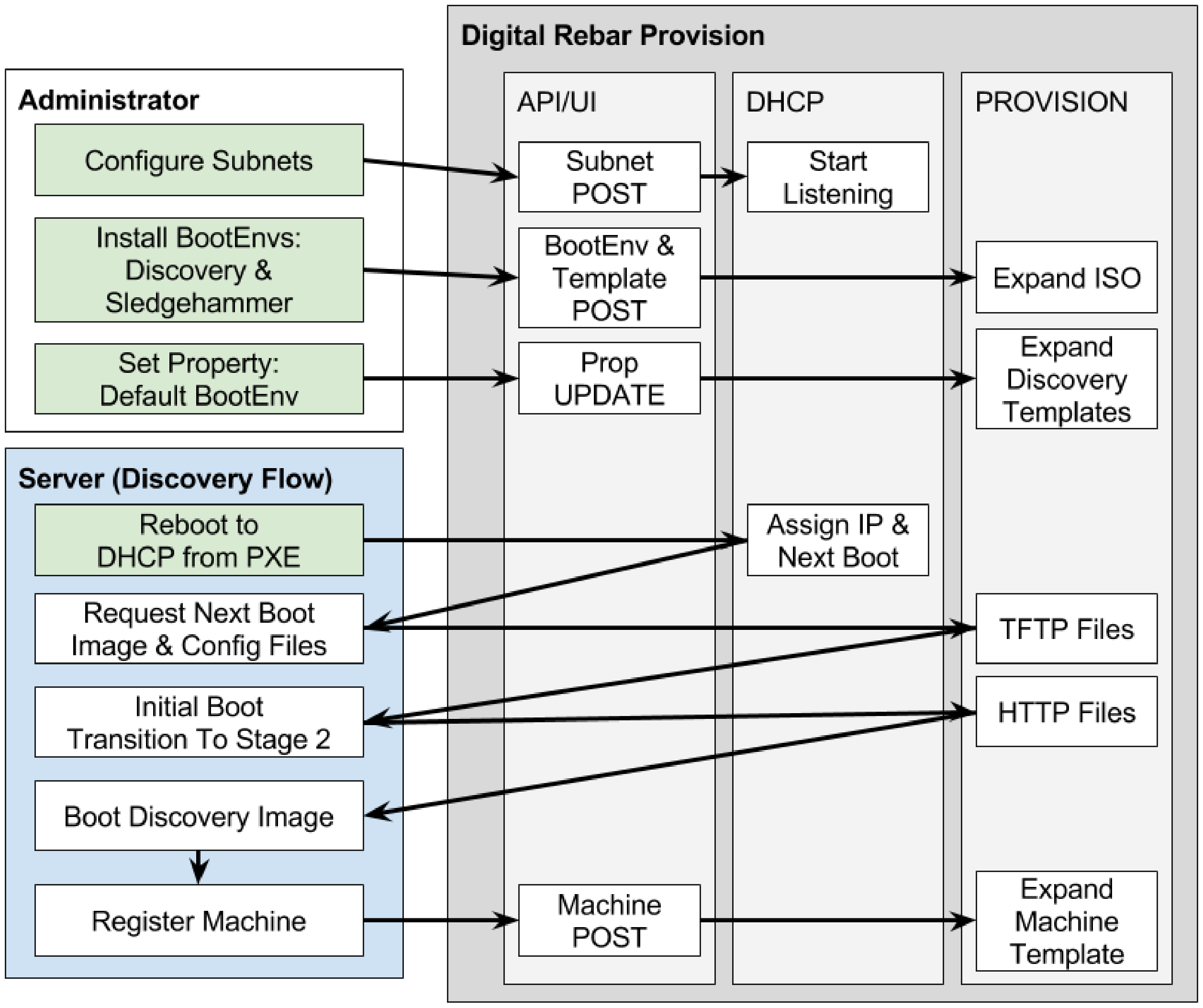
This is the typical workflow for Cobbler users in which machine definitions are determined in advance using spread sheets, stone tablets, Ouija boards or other 1990s data center inventory management techniques.

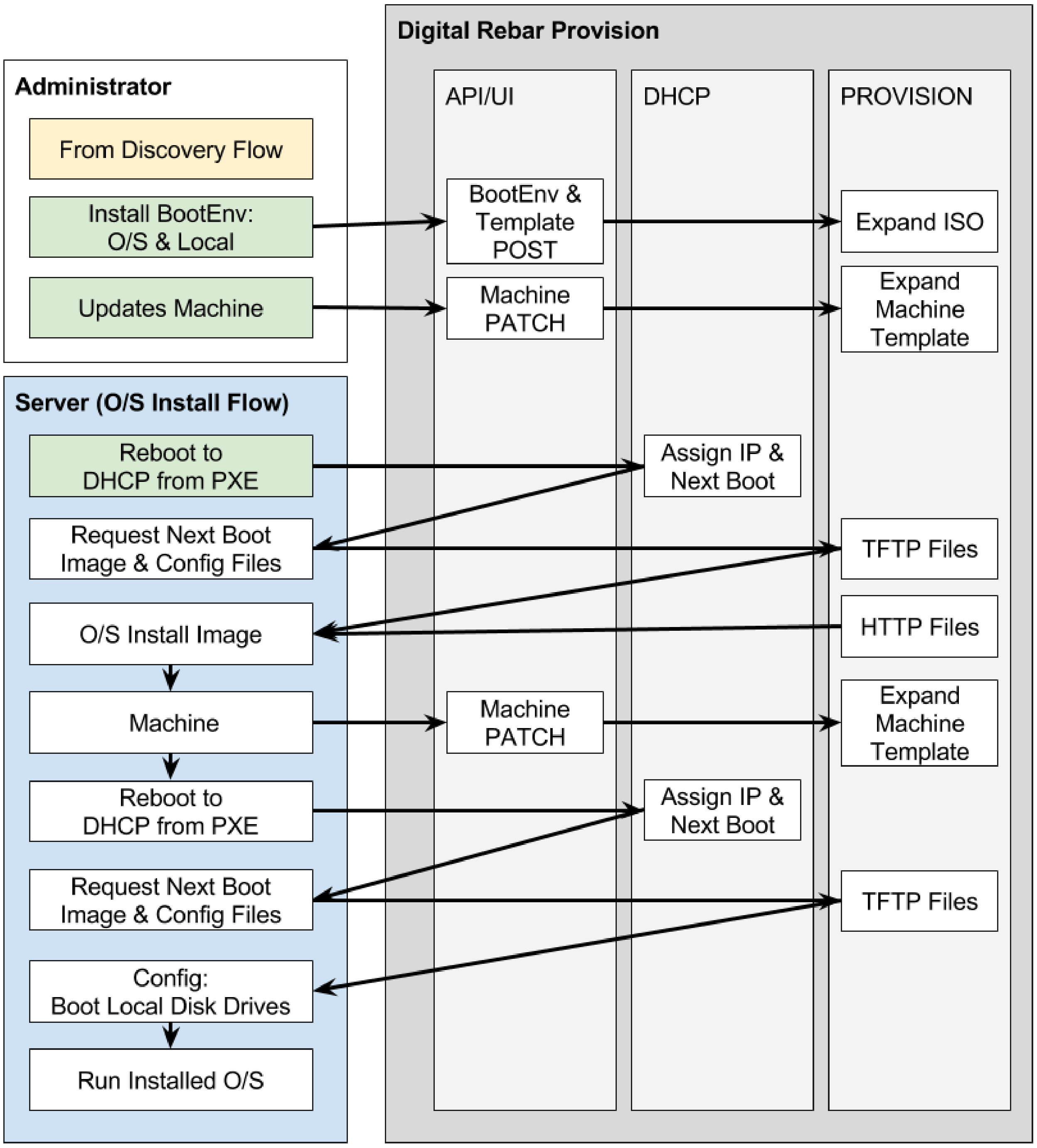
Two advantages to this approach is that it skips the discovery boot process; consequently, it is faster and can be run without DHCP.

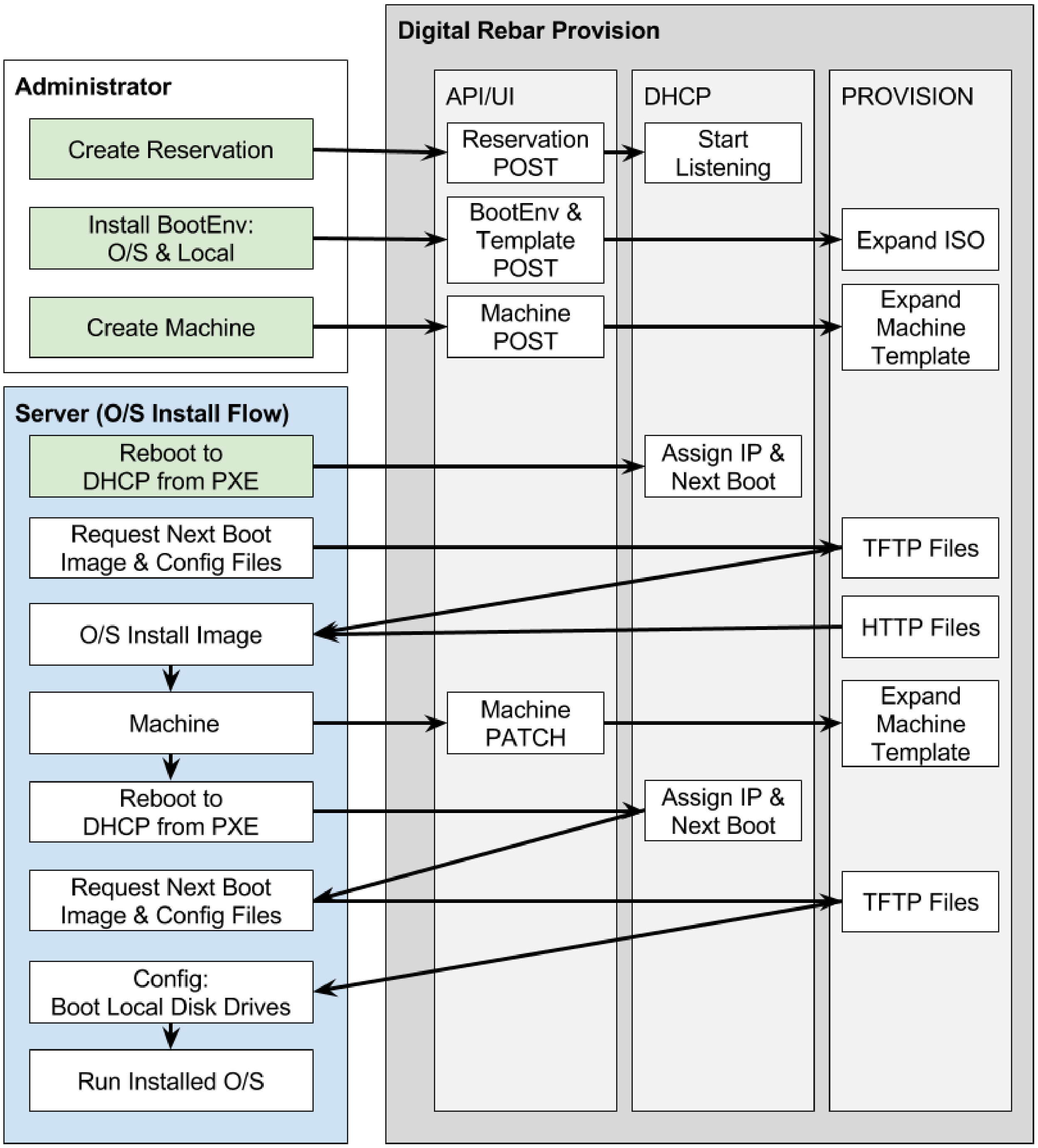
3.12 Deployment Options

Digital Rebar Provision is intended to be deployed as both a DHCP server and a Provisioner. There are cases where one or the other are desired. Each feature can be disabled by command line flags.

- *–disable-dhcp* - Turns off the DHCP server


Digital Rebar Provision
Administrator
Configure Subnets
Install BootEnvs: Discovery & Sledgehammer
Set Property: Default BootEnv
Server (Discovery Flow)
Reboot to DHCP from PXE
Request Next Boot Image & Config Files
Initial Boot Transition To Stage 2
Boot Discovery Image
Register Machine
API/UI
Subnet POST
BootEnv & Template POST
Prop UPDATE
Machine POST
DHCP
Start Listening
Assign IP & Next Boot
PROVISION
Expand ISO
Expand Discovery Templates
TFTP Files
HTTP Files
Expand Machine Template


Digital Rebar Provision
Administrator
From Discovery Flow
Install BootEnv: O/S & Local
Updates Machine
Server (O/S Install Flow)
Reboot to DHCP from PXE
Request Next Boot Image & Config Files
O/S Install Image
Machine
Reboot to DHCP from PXE
Request Next Boot Image & Config Files
Config: Boot Local Disk Drives
Run Installed O/S
API/UI
BootEnv & Template POST
Machine PATCH
Machine PATCH
DHCP
Assign IP & Next Boot
Assign IP & Next Boot
PROVISION
Expand ISO
Expand Machine Template
TFTP Files
HTTP Files
Expand Machine Template
TFTP Files


Digital Rebar Provision
Administrator
Create Reservation
Install BootEnv: O/S & Local
Create Machine
Server (O/S Install Flow)
Reboot to DHCP from PXE
Request Next Boot Image & Config Files
O/S Install Image
Machine
Reboot to DHCP from PXE
Request Next Boot Image & Config Files
Config: Boot Local Disk Drives
Run Installed O/S
API/UI
Reservation POST
BootEnv & Template POST
Machine POST
Machine PATCH
DHCP
Start Listening
Assign IP & Next Boot
Assign IP & Next Boot
PROVISION
Expand ISO
Expand Machine Template
TFTP Files
HTTP Files
Expand Machine Template
TFTP Files

- *–disable-provisioner* - Turns off the Provisioner servers (TFTP and HTTP)

The *Digital Rebar Provision API* doesn't change based upon these flags, only the services being provided.

3.12.1 DHCP Disabled

If a DHCP environment already exists or a more declarative mode is more desirable, there are a couple of things in each case to be aware of. For either case, the underlying assumption is that something will direct the node to use the provisioner as its *NextBoot* server.

Declarative Mode

Each machine must be declared through the *Digital Rebar Provision Command Line Interface (CLI)* or the *Digital Rebar Provision API*. The IP address in the rs_model_machine will be used to populate the rs_model_bootenv information. The provisioner will provide these files through TFTP and HTTP. It is left to the admin to figure out how to get the node to reference them.

External DHCP

With DHCP disabled, the admin can provide a DHCP server for distributing addresses. The DHCP will need to do the following:

- Set NextServer to an IP that routes to Digital Rebar Provision
- Set Option 67 (bootfile) to *lpxelinux.0* for legacy bios boots. Other options are available for other. See: rs_model_subnet
- Set Option 15 (dns domain) - This is needed for discovery boots to construct a meaningful FQDN for the node.
- Set Option 6 (dns server) - This is optional, but often useful in conjunction with Option 15.
- Set Option 3 (gateway) - This is optional, but may be required depending on the network routing.

3.12.2 Provisioner Disabled

In this mode, Digital Rebar Provision acts as a DHCP server only. The rs_dhcp_models describe how to use the server. Set the DHCP options that will direct to the next boot servers and other needs.

3.13 Digital Rebar Provision Operations

This section will attempt to describe common operations and actions that can be done. We will assume that `drpcli` is already somewhere in the path and have setup the environment variables to access Digital Rebar Provision. See, *Digital Rebar Provision Command Line Interface (CLI)*.

Some of these operations are in the *User Interface (UI)*, but not all. This will focus on CLI usage for now. See the *User Interface (UI)* section for UI usage.

Note: **drpcli** normally spits out JSON formatted objects or an array of objects. To help with some of the command line functions, we use the **jq** tool. This is available for both Linux and Darwin. You can specify output format to be YAML, with the `--format=yaml` flag.

3.13.1 Using the `drpcli` tool

The `drpcli` tool is designed to provide a single, easy to use compiled stand alone binary. It very closely mimics the API call parameters and usage, and allows for easy "transition" between the use of the CLI binary, and the use of the API calls.

We assume that you have the binary in your `PATH` somewhere (`/usr/local/bin/drpcli` in "production" mode by default). If you are using an "isolated" install mode, you will need to link the correct binary in to an apprpriate directory in your PATH. For example:

```
ln -s $HOME/bin/linux/amd64/dr-provision $HOME/bin/
ln -s $HOME/bin/linux/amd64/drpcli $HOME/bin/
```

`drpcli` is a self documenting binary file. At any point you can can simply hit `<Enter>` to see the current contextual help output. Some examples:

`drpcli <Enter>` Prints usage for the top-level resources that may be manipulated.

`drpcli templates <Enter>` Prints the resources that may be executed for the `templates` level.

`drpcli templates create <Enter>` Will display the commands associated with just `templates/create/`

Note that the CLI syntax closely follows the API calls. For example:

```
drpcli templates create -< some_file.json
```

Would be equivalent to the HTTP REST API resource as follows:

```
https://127.0.0.1:8092/api/v3/templates/ereate
```

Note: See the *Turn on autocomplete for the CLI* for BASH shell auto completion.

3.13.2 Preference Setting

By default Digital Rebar Provision (DRP) attempts to "do no harm". This means that by default, any system that receives a DHCP lease from the DRP Endpoint, will by default, be set to `local` mode, which means boot from local disks. You must explicitly change this behavior to enable provisioning activities of Machines.

It is necessary to get or set the preferences for the system, to enable OS Installs (BootEnvs).

```
# Show the current preference settings
drpcli prefs list

# Or get a specific one
drpcli prefs get unknownBootEnv
```

Set preference(s):

```
# Set a preference
drpcli prefs set unknownBootEnv discovery

# or chain multiples as pairs
drpcli prefs set unknownBootEnv discovery defaultBootEnv sledgehammer␣
↪defaultStage discover
```

The system does validate values to make sure they are sane, so watch for errors.

3.13.3 BootEnv Operations

A rs_model_bootenv is the primary component for an Operating System installation definition. A rs_model_bootenv is comprised of two primary pieces:

1. A rs_model_bootenv JSON/YAML specification
2. (usually) an ISO Image that installs that rs_model_bootenv

The JSON/YAML specification will contain a set of definitions for the ISO image. The default distributed rs_model_bootenv specs use the public mirror repos for the ISO images. You can create a customer rs_model_bootenv with a pointer to your own hosted ISO images. An example looks something like:

```
root@demo:~$ drpcli bootenvs show ubuntu-16.04-install
{
"Available": true,
"Name": "ubuntu-16.04-install",
"OS": {
  "Family": "ubuntu",
  "IsoFile": "ubuntu-16.04.3-server-amd64.iso",
  "IsoSha256":
↪"a06cd926f5855d4f21fb4bc9978a35312f815fbda0d0ef7fdc846861f4fc4600",
  "IsoUrl": "http://mirrors.kernel.org/ubuntu-releases/16.04/ubuntu-16.04.3-
↪server-amd64.iso",
  "Name": "ubuntu-16.04",
<...snip...>
```

This stanza shows the Ubuntu 16.04 rs_model_bootenv along with the associated Mirror HTTP location the ISO will be installed from.

Installing a "Canned" BootEnv

Manipulating rs_model_bootenv and rs_model_template are handled by their own commands. There are some additional helpers especially when following the layout of the initial *Install*.

To install a provided rs_model_bootenv, do the following from the install location.

```
drpcli bootenvs uploadiso ubuntu-16.04-install
```

This is a CLI helper that is not in the API that will read the provided YAML rs_model_bootenv file, upload the included or referenced rs_model_template files (from the *templates* peer directory), upload the rs_model_bootenv, and check for an existing ISO in the ISO repository. If an ISO is not present in the already uploaded list, it will check a local isos directory for the file. If that is not present and the rs_model_bootenv contains a URL for the ISO, the ISO will attempt to be downloaded to the local isos directory and then uploaded into Digital Rebar Provision. Once upload, the ISO is "exploded" for access by machines in the file server file system space.

Listing Installed BootEnvs

A list of all existing rs_model_bootenv installed on the DRP Endpoint can be obtained with the *list* command. However, you usually do not wish to see all of the JSON values, and a simple `jq` filter can help output just the keys you are interested in, as follows:

```
drpcli bootenvs list | jq -r '.[].Name'

Outputs:
centos-7-install
```

(continues on next page)

(continued from previous page)

```
centos-7.4.1708-install
debian-8-install
debian-9-install
discovery
ignore
local
sledgehammer
ubuntu-16.04-install
```

Cloning a BootEnv

Sometimes there is a rs_model_bootenv but it is necessary to make changes. These can be handled by rs_model_template inclusion, but for now let's just focus on basic "cut and paste" style editing.

```
drpcli bootenvs show ubuntu-16.04-install --format yaml > new-file.yaml
# Edit the file
#  change the Name field to something new. *MUST DO THIS*
#  change the OS->Name field to something new to avoid sharing an iso␣
↪directory.
#  Edit other parameters as needed
drpcli bootenvs create - < new-file.yaml
```

This is a shallow clone. It will reuse the templates unless they are explicitly modified. It is possible to use the *install* command, but any new templates would need to be added to a *templates* directory in the current directory.

Creating a BootEnv

It might be necessary to create an empty rs_model_bootenv by doing the following:

```
drpcli bootenvs create emtpy_bootenv
```

This rs_model_bootenv will not be *Available*, but will allow for additional editing.

Editing a BootEnv

It might be necessary to edit a rs_model_bootenv. To do this, get the latest copy with the *show* command. Edit the file as needed. Then using the *update* command, put the value back. The *–format=yaml* is optional, but I find YAML easier to edit.

```
drpcli bootenvs show discovery --format=yaml > discovery.yaml
# Edit the discovery.yaml as needed
drpcli bootenvs update discovery - < discovery.yaml
```

3.13.4 Subnet Operations

Subnet definitions provide the necessary information for DHCP IP Address lease assignments, and allows Machines to be enrolled/discovered by a DRP Endpoint. For any Layer 2 subnet/network that you wish to install Machines from, you must also specify a Subnet definition for. In some environments, a Subnet definition may not be needed to allow Machines to be discovered.

Cloning a Subnet

It might be necessary to create a new subnet from an existing one. To do this, do the following:

```
drpcli subnets show eth0 | jq -r > new_subnet.json
# edit the new_subnet.json file with the new information
drpcli subnets create -< new_subnet.json
```

Creating a new Subnet

A new subnet can be created from a JSON specification. It is necessary to use all of the following JSON keys to successfully create a new Subnet that can be immediately used to manage machines – the rest of the keys will be autofilled with reasonable defaults.

```
echo '
{
  "Name": "local_subnet",
  "Subnet": "10.10.16.10/24",
  "ActiveStart": "10.10.16.100",
  "ActiveEnd": "10.10.16.254",
  "ActiveLeaseTime": 60,
  "Enabled": true,
  "ReservedLeaseTime": 7200,
  "Strategy": "MAC",
  "Options": [
    { "Code": 3, "Value": "10.10.16.1", "Description": "Default Gateway" },
    { "Code": 6, "Value": "8.8.8.8", "Description": "DNS Servers" },
    { "Code": 15, "Value": "example.com", "Description": "Domain Name" }
  ]
} ' > /tmp/local_subnet.json

drpcli subnets create -< /tmp/local_subnet.json
```

Note that the "Description" is purely cosmetic and not used - however, it can be safely specified as it'll be ignored (it's added here for the readers reference). You must provide the minimum DHCP Options as specified above. You can find a complete set of DHCP Options at:

https://www.iana.org/assignments/bootp-dhcp-parameters/bootp-dhcp-parameters.xhtml

For complete documentation and information you can find the DHCP Options officially documented in RFC2132

Updating a Subnet

From time to time, you may need to modify an existing Subnet definition. Depending on your changes, you have a couple of options.

Set the NTP Server pool via DHCP Option 42 for subnet "local_subnet":

```
drpcli subnets set local_subnet option 42 to "0.pool.ntp.org"
```

Set the DHCP IP assignment from the following pick list for subnet "local_subnet". See rs_model_pickers for a detailed descrip

```
drpcli subnets pickers local_subnet hint,nextFree,mostExpired
```

Set the nextserver for PXE operation for subnet "local_subnet":

```
drpcli subnets  nextserver  local_subnet 10.16.167.10
```

Set the subnet DHCP range of IP addresses for subnet "local_subnet":

```
drpcli subnets range local_subnet 192.168.45.100 192.168.45.255
```

Set Active lease to 60 mins, and reserved lease to 7200 mins for subnet "local_subnet":

```
drpcli subnets leasetimes local_subnet 60 7200
```

Update a subnet to set it to disabled (do not discover, and do not provision on this subnet, for subnet "local_subnet":

```
drpcli subnets update local_subnet '{ "Enabled": false }'
```

Update a subnet with the contents of the specified JSON file, for subnet "local_subnet":

```
drpcli subnets update local_subnet -< update-local_subnet.json
```

Deleting a Subnet

To remove a Subnet and subsequently cease PXE provisioning operations for that Subnet, perform the following:

```
drpcli subnets destroy local_subnet
```

List and Show Subnets

Viewing configuration for all subnets can be done with the `list` command as follows:

```
drpcli subnets list
```

To `show` an individual subnet, you will need the subnet name. To show just the subnet names, you can use `jq` to filter the outpu

```
drpcli subnets list | jq '.[].Name'
```

Once you have determined which subnet you'd like to show specific information for, you can do so with the following command:

```
# show the YAML formatted output for 'local_subnet' subnet
drpcli subnets show local_subnet --format=yaml
```

3.13.5 Template Operations

Templates are reusable blocks of code, that are dynamically expanded when used. This allows for very sophisticated and complex operations. It also allows for carefully crafted Templates to be re-usable across a broad set of use cases.

Cloning a Template

It might be necessary to create a new template from an existing one. To do this, do the following:

```
drpcli templates show net_seed.tmpl | jq -r .Contents > new.tmpl
# Edit the new.tmpl to be what is required
drpcli templates upload new.tmpl as new_template
```

In this case, we are using `jq` to help us out. `jq` is a JSON processing command line filter. JSON can be used to retrieve the required data. In this case, we are wanting the Contents of the template. We save that to file, edit it, and upload it as a new template, *new_template*.

It is possible to use the **create** subcommand of template, but often times **upload** is easier.

Note: Remember to add the new template to a rs_model_bootenv or another rs_model_template as an embedded template.

Updating a Template

It might be necessary to edit an existing template. To do this, do the following:

```
drpcli templates show net_seed.tmpl | jq -r .Contents > edit.tmpl
# Edit the edit.tmpl to be what is desired
drpcli templates upload edit.tmpl as net_seed.tmpl
```

We use `jq` to get a copy of the current template, edit it, and use the upload command to replace the template. If there already is a template present, then it can be replaced with the upload command.

3.13.6 Param Operations

rs_model_param are simply key/value pairs. However, DRP provides a strong typing model to enforce a specific type to a given Param. This ensures that Param values are valid elements as designed by the operator.

Creating a Param

It might be necessary to create a new rs_model_param, an empty Param may be created by doing the following:

```
drpcli params create '{ "Name": "fluffy" }'

or

drpcli params create fluffy
```

The system will attempt to use any sent string as the Name of the Param. To be complete, it is required to also speciy the Type that param must be:

```
drpcli params create '{ "Description": "DNS domainname", "Name": "domainname
↪", "Schema": { "type": "string" } }'
```

In this example, the type `string` was defined for the param.

Deleting a Param

It might be necessary to delete a rs_model_param.

```
drpcli params destroy fluffy
```

Note: The destroy operation will fail if the param is in use.

Editing a Param

It might be necessary to update a Param. An example to add a `type` of `string` to our `fluffy` param above would be:

```
drpcli params update fluffy '{ "Schema": { "type": "string" } }'
```

3.13.7 Profile Operations

rs_model_profile are simply collections of rs_model_param - they conveniently group multiple rs_model_param for easy consumption by other elements of the provisioning service.

Creating a Profile

It might be necessary to create a rs_model_profile. An empty profile can be created by doing the following:

```
drpcli profiles create '{ "Name": "myprofile" }'

or

drpcli profiles create myprofile
```

The system will attempt to use any sent string as the Name of the profile.

Additionally, JSON can be provided to fill in some default values.

```
drpcli profiles create '{ "Name": "myprofile", "Params": { "string_param1":
↪"string", "map_parm1": { "key1": "value", "key2": "value2" } } }'
```

Alternatively, you can create profiles from an existing file containing JSON, as follows:

```
echo '{ "Name": "myprofile", "Params": { "string_param1": "string", "map_
↪parm1": { "key1": "value", "key2": "value2" } } }' > my_profile.json
drpcli profiles create -< my_profile.json
```

Deleting a Profile

It might be necessary to delete a rs_model_profile. It is possible to use the destroy command in the profile CLI, but the rs_model_profile must not be in use. Use the following:

```
drpcli profiles destroy myprofile
```

Altering an Existing Profile (including the `global` profile)

It might be necessary to update an existing rs_model_profile, including **global**. parameter values can be *set* by doing the following:

```
drpcli profiles set myprofile param crazycat to true
# These last two will show the value or the whole profile.
drpcli profiles get myprofile param crazycat
drpcli profiles show myprofile
```

Note: Setting a parameter's value to **null** will clear it from the structure.

Alternatively, the update command can be used to send raw JSON similar to create.

```
drpcli profiles update myprofile '{ "Params": { "string_param1": "string",
↪"map_parm1": { "key1": "value", "key2": "value2" }, "crazycat": null } }'
```

Update is an additive operation by default. So, to remove items, **null** must be passed as the value of the key to be removed.

3.13.8 Machine Operations

A rs_model_machine is typically a physical bare metal server, as DRP is intended to operate on bare metal infrastructure. However, it can represent a Virtual Machine instance and provision it equally. DRP does not provide *control plane* activities for virtualized environments (eg *VM Create*, etc. operations).

Creating a Machine

It might be necessary to create a rs_model_machine. Given the IP that the machine will boot as all that is required is to create the machine and assign a rs_model_bootenv. To do this, do the following:

```
drpcli machine create '{ "Name": "greg.rackn.com", "Address": "1.1.1.1" }'
```

This would create the rs_model_machine named *greg.rackn.com* with an expected IP Address of *1.1.1.1*. *dr-provision* will create the machine, create a UUID for the node, and assign the rs_model_bootenv based upon the *defaultBootEnv* rs_model_prefs.

```
drpcli machine create '{ "Name": "greg.rackn.com", "Address": "1.1.1.1",
↪"BootEnv": "ubuntu-16.04-install" }'
```

This would do the same thing as above, but would create the rs_model_machine with the *ubuntu-16.04-install* rs_model_bootenv.

Note: The rs_model_bootenv MUST exist or the create will fail.

To create an empty rs_model_machine, do the following:

```
drpcli machine create jill.rackn.com
```

This will create an empty rs_model_machine named *jill.rackn.com*.

Note: The *defaultBootEnv* rs_model_bootenv MUST exist or the create will fail.

Adding or Removing a Profile to a Machine

It might be necessary to add or remove a rs_model_profile to or from a rs_model_machine. To add a profile, do the following:

```
drpcli machines addprofile "dff3a693-76a7-49ce-baaa-773cbb6d5092" myprofile
```

To remove a profile, do the following:

```
drpcli machines removeprofile "dff3a693-76a7-49ce-baaa-773cbb6d5092"␣
↪myprofile
```

The rs_model_machine update command can also be used to modify the list of rs_model_profile.

Changing BootEnv on a Machine

It might be necessary to change the rs_model_bootenv associated with a rs_model_machine. To do this, do the following:

```
drpcli machines bootenv drpcli "dff3a693-76a7-49ce-baaa-773cbb6d5092"␣
↪mybootenv
```

Note: The rs_model_bootenv *MUST* exists or the command will fail.

Rename a Machine

By default, machines are given names based on the machines primary network MAC address. Most infrastructure environments need to rename machines to fall in line with a naming scheme in use with the company. To do that safely, we will use the existing Machins object information as a baseline to apply a Patch operation to the JSON. This is a two step process that is completed in the following example:

First lets define the Machine (UUID) that we're going to operate on, and lets get the current name of the machine for reference

```
# get our machine to operate on
export UUID="f6ca7bb6-d74f-4bc1-8544-f3df500fb15e"

# our reference starting point for 'Name'
drpcli machines show $UUID | jq '.Name'
"fred"
```

We now need to obtain the Machine object JSON tha we are going to apply the patch against.

```
# get current machine object that we want to reference the change against
drpcli machines show $UUID  > /tmp/machine.json
```

Now that we have our reference Machine object, we'll use the `update` option to the `machines` manipulation.

```
# set the name using the reference JSON object
drpcli machines update $UUID '{ "Name": "wilma" }' --ref /tmp/machine.json

# outputs
{
  "Address": "147.75.66.137",
  <...snip...>
  "Name": "wilma",
  <...snip...>
```

Here is a single command example (using our `$UUID` variable above) that does not require temporary files.

```
drpcli machines update $UUID '{ "Name": "barney" }' --ref "$(drpcli machines show
↪$UUID)"
```

You can update "unsafely", but if multiple updates occur, you can't guarantee that you're changing what you expected to (eg. so

```
# this is a BAD way to do it - as it does not guarantee atomicity
drpcli machines update $UUID '{ "Name": "betty" }'

# outputs
{
  "Address": "147.75.66.137",
  <...snip...>
  "Name": "betty",
  <...snip...>
```

3.13.9 DHCP Operations

Creating a Reservation

It might be necessary to create a rs_model_reservation. This would be to make sure that a specific MAC Address received a specific IP Address. Here is an example command.

```
drpcli reservations create '{ "Addr": "1.1.1.1", "Token": "08:00:27:33:77:de
↪", "Strategy": "MAC" }'
```

Additionally, it is possible to add DHCP options or the Next Boot server.

```
drpcli reservations create '{ "Addr": "1.1.1.5", "Token": "08:01:27:33:77:de
↪", "Strategy": "MAC", "NextServer": "1.1.1.2", "Options": [ { "Code": 44,
↪"Value": "1.1.1.1" } ] }'
```

Remember to add an option 1 (netmask) if a subnet is not being used to fill the default options.

3.13.10 Advanced Workflow

Placeholder for Advanced Workflow overview.

Stages

Placeholder for Stages information.

Stage Maps

Placeholder for Stage Map information.

Tasks

Placeholder for Tasks information.

Jobs

Placeholder for Jobs information.

3.13.11 User Operations

Creating a User

It might be necessary to create a rs_model_user. By default, the user will be created without a valid password. The user will only be able to access the system through granted tokens.

To create a user, do the following:

```
drpcli users create fred
```

Note: This rs_model_user will *NOT* be able to access the system without additional admin action.

Granting a User Token

Sometimes as an administrator, it may be necessary to grant a limited use and scope access token to a user. To grant a token, do the following:

```
drpcli users token fred
```

This will create a token that is valid for 1 hour and can do anything. Additionally, the CLI can take additional parameters that alter the token's scope (model), actions, and key.

```
drpcli users token fred ttl 600 scope users action password specific fred
```

This will create a token that is valid for 10 minutes and can only execute the password API call on the rs_model_user object named *fred*.

To use the token in with the CLI, use the -T option.

```
drpcli -T <token> bootenvs list
```

Deleting a User

It might be necessary to remove a reset from the system. To remove a user, do the following:

```
drpcli users destroy fred
```

Revoking a User's Password

To clear the password from a rs_model_user, do the following:

```
drpcli users update fred '{ "PasswordHash": "12" }'
```

This basically creates an invalid hash which matches no passwords. Issued tokens will still continue to function until their times expire.

Secure User Creation Pattern

A secure pattern would be the following:

- Admin creates a new account

```
drpcli users create fred
```

- Admin creates a token for that account that only can set the password and sends that token to new user.

```
drpcli users token fred scope users action password ttl 3600
```

- New user uses token to set their password

```
drpcli -T <token> users password fred mypassword
```

3.14 OS Support

Digital Rebar Provision, through its rs_model_bootenv, provides the ability to use and install many types of operating systems.

The default installation includes an assets directory that contain pre-existing bootenvs and templates. Example Ubuntu, CentOS, and other operating systems are available to start from.

This section of the docs provides some hints and gotchas for the various platforms.

3.14.1 LinuxKit

1. *linuxkit*

3.15 LinuxKit

LinuxKit is one of Docker's latest creations to provide an immutable simple OS that can run containers and other container platforms. The goal is to provide a simple to update, secure, and maintain OS that can facilitate container deployments.

It turns out that Digital Rebar Provision can easily deploy LinuxKit images. The example assets provided with Digital Rebar Provision contains some examples of LinuxKit deployments.

There are three currently available. Though they are really simple and easily cloned.

- lk-sshd - the example sshd
- lk-k8s-master - the k8s project master image

- lk-k8s-node - the k8s project node image

Here are the following steps that work on an Ubuntu 16.04 desktop with KVM already setup. The basic overview is:

- Get and start dr-provision
- Configure dr-provision to handle KVM network
- Get and build LinuxKit
- Make some ISOs
- Install BootEnvs
- Run Nodes

Let's hit each of the steps. For a simple play with it trial, create a directory for both Digital Rebar Provision and LinuxKit.

Also, there is a video of these steps.

3.15.1 Get, Start, and Configure Digital Rebar Provision

There are already many pages for this, *Quick Start* and *Install*. The main thing is to use the **tip** version at the moment. It is best to include the *discovery* and *sledgehammer* images. The KVM system used (the Digital Rebar test tool, kvm-slave) always PXE boots and machines can be easily added just by starting them.

For a simple trail, use the install process with the *–isolated* flag.

Something like:

```
mkdir -p lk-dr-trial/dr-provision
cd lk-dr-trial/dr-provision
curl -fsSL https://raw.githubusercontent.com/digitalrebar/provision/master/
↪tools/install.sh | bash -s -- --isolated --drp-version=tip install
# Follow the instructions at the end of the script
```

3.15.2 Get and Build LinuxKit

First, create a directory for everything, clone LinuxKit, and build it. This assumes that *go install* was executed.

```
cd lk-dr-trial
git clone https://github.com/linuxkit/linuxkit.git
cd linuxkit
make
```

After a few minutes, there should be a bin directory with **moby** ready to go.

3.15.3 Make some ISOs

The provided rs_model_bootenv deploying the sshd example and k8s project. To build these, we need to do a couple of things.

- **Edit the *examples/sshd.yml***
  - Replace the "#your ssh key here" line with the contents of the SSH public key. e.g. ~/.ssh/id_rsa.pub
- Run the moby build command

```
# edit files
bin/moby build -output iso-bios examples/sshd.yml
```

This will generate an ISO, *sshd.iso*. Copy this file into the assets/isos directory (creating it if it doesn't exist) in the dr-provision install directory.

Additionally, we can build the Kubernetes images. We still need to edit a couple of files.

- cd projects/kubernetes
- **Edit the *k8s-master.yml***
  - Replace the "#your ssh key here" line with the contents of the SSH public key. e.g. ~/.ssh/id_rsa.pub
- **Edit the *k8s-node.yml***
  - Replace the "#your ssh key here" line with the contents of the SSH public key. e.g. ~/.ssh/id_rsa.pub
- **Edit the *Makefile***
  - Add **-output iso-bios** to the moby build lines. This will make sure to make ISOs.
  - E.g. ../../bin/moby build -output iso-bios -name kube-master kube-master.yml
  - E.g. ../../bin/moby build -output iso-bios -name kube-node kube-node.yml
- Run the make command

```
cd projects/kubernetes
# edit files
make
```

This will generate two ISO images, *kube-master.iso* and *kube-node.iso*. Copy these files into the assets/iso directory in the dr-provision install directory.

3.15.4 Install BootEnvs

At this point, we can add the rs_model_bootenv to Digital Rebar Provision.

- Change to the Digital Rebar Provision directory and then to the assets directory.
- Run the following

```
cd lk-dr-trial/dr-provision/assets
export RS_KEY=rocketskates:r0cketsk8ts # or whatever it is it set to.
../drpcli bootenvs install bootenvs/lk-sshd.yml
../drpcli bootenvs install bootenvs/lk-k8s-master.yml
../drpcli bootenvs install bootenvs/lk-k8s-node.yml
```

This will make all three rs_model_bootenv available for new nodes.

3.15.5 Run Nodes

At this point, it is possible to boot some nodes and run them. They can have pre-existing nodes or discovered nodes. This will use discovered nodes.

First, start some nodes. The Digital Rebar team prefers kvm-slave tool that starts KVM on my Digital Rebar Provision network. .e.g. tools/kvm-slave Anything that PXEs and can three will work.

Once they are discovered, something like this from **drpcli machines list** should appear:

```
[
  {
    "Address": "192.168.124.21",
    "BootEnv": "sledgehammer",
    "Errors": null,
    "Name": "d52-54-54-07-00-00.example.com",
    "Uuid": "4cc8678e-cdc0-48ee-b898-799103840d7f"
  },
  {
    "Address": "192.168.124.23",
    "BootEnv": "sledgehammer",
    "Errors": null,
    "Name": "d52-54-55-00-00-00.example.com",
    "Uuid": "c22a3db3-dba8-4138-8375-7a546c8097e8"
  },
  {
    "Address": "192.168.124.22",
    "BootEnv": "sledgehammer",
    "Errors": null,
    "Name": "d52-54-54-7d-00-00.example.com",
    "Uuid": "d8d5b78a-976b-41c6-a968-31c73ba2b8a4"
  }
]
```

At this point, the BootEnv field should be changed to the environment of choice.

```
cd lk-dr-trial/dr-provision
./drpcli machines bootenv "4cc8678e-cdc0-48ee-b898-799103840d7f" lk-sshd
./drpcli machines bootenv "d8d5b78a-976b-41c6-a968-31c73ba2b8a4" lk-k8s-
↪master
./drpcli machines bootenv "c22a3db3-dba8-4138-8375-7a546c8097e8" lk-k8s-node
```

Now, reboot those kvm instances (close the KVM console window or kill the qemu process). Once the systems boot up, it should be possible to ssh into them from the account the ssh key is from (as root).

And that is all for the sshd image.

For Kubernetes, a few more steps are required. In this example, 192.168.124.22 is the master. We need to SSH into its kubelet container and start kubeadm. Something like this:

```
ssh root@192.168.124.22
nsenter --mount --target 1 runc exec --tty kubelet sh
kubeadm-init.sh
```

This will run for a while and start up the master. It will output a line that looks like this:

```
kubeadm join --token bb38c6.117e66eabbbce07d 192.168.65.22:6443
```

This will need to run on each k8s-node. It is necessary to SSH into the kubelet on the k8s node. Something like this:

```
ssh root@192.168.124.23
nsenter --mount --target 1 runc exec --tty kubelet sh
kubeadm join --token bb38c6.117e66eabbbce07d 192.168.65.22:6443
```

Now, wait for a while and if the KVM instances have Internet access, then kubernetes will be up. The default access for this cluster is through the kubelet container though others are probably configurable.

```
ssh root@192.168.124.22
nsenter --mount --target 1 runc exec --tty kubelet sh
kubectl get nodes
```

There are ssh helper scripts in the *linuxkit/projects/kubernetes* directory, but they do not always work with the latest k8s containers.

3.16 User Interface (UI)

The Digital Rebar Provision UI is intended to help users with basic operational needs. Advanced users should review the API documentation or consider using Digital Rebar for integration and workflow capabilities.

3.16.1 Subnets

Configuring Subnets is a critical first step in Digital Rebar Provision operation. The basic UI will show all configured subnets and provide an easy way to add broadcast subnets based on the known interfaces.

To edit or delete a subnet, click on the name of the subnet to populate the editing area below the list. To create a relay subnet, click on the add subnet link. To create a broadcast subnet, click on the link provided after the name of the unassigned interfaces.

There are two primary types of subnets: broadcast and relay:

- Broadcast subnets are associated with the addresses provided by the Digital Rebar Provision host system. Digital Rebar Provision can have multiple DHCP ranges; however, only one subnet can be assigned per interface _and_ the subnet CIDR must include the IP of the interface (the range should NOT). By convention, subnets associated with an interface will be named as the interface.
- Relay subnets answer requests forwarded from DHCP relays such as those provided by switches. These can be any suitable IP range. Since the Relay subnets are not broadcast, they do not conflict with existing DHCP servers in the environment.

Digital Rebar Provision can operate in a permissive reservation mode or require users to whitelist systems before they are serviced. The *OnlyReservations* flag will operate as a reservations required (whitelist) mode when true; otherwise, Digital Rebar Provision permissive reservation mode will give out addresses to any valid DHCP request.

In additional to serving IPs, DHCP servers provide critical configuration (aka DHCP Options) information to the clients. Setting Option 67, Next Boot, is essential for Digital Rebar Provision to operate as a Provisioner. This information includes next boot (67), gateway (3), domain name (15), DNS (6) and other important information. It is encoded in the responses according to IETF RFC 2132

Consult the Godocs for more details about the specific fields.

3.16.2 Boot Environments (bootenvs)

Configuring at least one Boot Environment is a critical first step in Digital Rebar Provision operation. The Digital Rebar CentOS based in-memory discovery image, Sledgehammer, will be installed on first use by default.

The UI will show a complete list of potential Boot Environments;

3.16.3 Machines

Machines are central to the provisioning process, as they connect Boot Environments to incoming IP addresses. Desired BootEnvs can be assigned to machines from this page.

If a machine is not given a BootEnv, it will use the BootEnv listed as *defaultBootEnv* on the BootEnvs Preferences page.

Profiles can also be bound to machines from this view. Machines will relay the parameters of a profile to the templates provided by the selected BootEnv.

3.16.4 Profiles

Profiles provide a convenient way to apply sets of parameters to a machine. Multiple profiles can be assigned to one machine, and will be referenced in the order they are listed.

Parameters can be linked to specific profiles through the profiles page, which can then be attached to machines through the machines UI.

3.16.5 Templates

Templates contain important instructions for the provisioning process, and are comprised of golang text/template strings. Once templates are rendered along with any assigned parameters, they are used by the BootEnv to boot the target machine.

Templates may contain other templates, known as sub-templates.

3.16.6 Parameters (params)

Parameters are passed to a template from a machine, and help to drive the template's functions. They consist of key/value pairs that provide configuration to the renderer.

Profiles allow params to be applied in bulk, or they can be attached to templates individually.

3.16.7 Reservations

Reservations link tokens to specific IP addresses. This view shows a list of existing reservations along with tokens and strategies associated with each. Currently, MAC is the only available strategy.

Reservations may contain options to be applied to connected servers, which are also visible through the UI.

3.16.8 Leases

Leases show individual links between tokens and addresses, created through reservation or subnet strategies. Leases remain valid for short periods of time, and cannot be edited.

The expiration time of each lease is visible through the UI. Once a lease has expired, it may be removed.

3.16.9 Tasks

During the boot process, tasks provide additional configuration to machines in the form of templates. BootEnvs will use these sets of templates to construct specific jobs for a machine.

Within a task, templates are processed in the order they are assigned, so it's important to check that templates are attached correctly to a task.

3.16.10 Jobs

A job defines a machine's current step in its boot process. After completing a job, the machine creates a new job from the next instruction in the machine's task list.

Machines will only process one job at a time, and jobs aren't created until the instant they are required.

3.17 RackN Portal

The RackN Portal provides a complete graphical solution to manage Digital Rebar Provision.

3.17.1 Home Page

The RackN Portal home page has an option in the upper left corner to select the three lines providing three groupings of information.

Registered Endpoints

The Registered Endpoints contain the IP addresses of previously installed Digital Rebar Provision for a specific user.

Organization

This section provides information about your account with RackN.

User Profile

This section contains the following information:

- Support - Complete the Message Subject and Message Body to send a RackN administrator a support question.
- **Security - Manage your account security**
    - Account ID of the User
    - Email Address
    - Phone Number (Verify Phone option)
    - Log out of the System
    - Change your Password
    - Delete the Account

Members

This section provides information about the membership organization (Group) the user is connected to.

The following information is provided:

- Email Address
- Assigned User GUID (Group User ID)
- **Group Rights:**
    - Edit Members (Y/N)
    - Edit the Catalog (Y/N)
    - Edit Endpoints

Subscriptions

This section provides a list of all community (free) and RackN paid services actively used by the User.

The following three types of features are listed:

- Content - YAML or JSON processes for Digital Rebar Provision to execute during the provisioning process
- Plugin - Binary packages extending the basic functionality of Digital Rebar Provision. These are installed via the UX and then configured by the user.
- Interface Features - Advanced UX technology available for purchase from RackN.

Endpoints

This section provides a list of Endpoints available to the user from the membership organization (Group) the user is connect to. There is an option to change the Icons used in the RackN Portal.

For each Endpoint listed the following data is shown:

- Endpoint ID
- IP Address
- Name of the Endpoint (Default is IP Address:8092)
- Owner of the Endpoint (Email Address)

Catalog

This section contains a list of all available community and RackN features, plugins and interface features. Users can select services required for their Digital Rebar Provision solution.

Content

This is a list of content packages currently available to your Digital Rebar Provision service. This packages contain the instructions for DRP to perform an operation such as kubespray which installs Kubernetes via Kubespray playbooks.

Plugins

This is a list of Plugins currently available to your Digital Rebar Provision service. These packages enable DRP to provide additional provisioning offerings such as immutable image deployment or Windows provisioning. Some of these Plugins are freely available to all community members and some are only available from RackN for a fee.

Interface Features

This is a list of UX technology available for purchase from RackN.

3.17.2 System

This part of the RackN Portal contains a quick overview of key activities being managed by Digital Rebar Provision including all machines, machine stages, machine activity, bulk actions on those machines, plugins, and the Information and Preferences screen. The Information and Preferences screen is the default home page at login.

Video: Why RackN built the Portal https://youtu.be/levzBw5t7gc.

Overview

The Overview page shows a live view of all the active machine stages running via the RackN Portal including their State and BootEnv settings. The page also shows the running machines activity.

The top of the page offers the following services via a single blue button for each action:

- Refresh - Refresh the content on the active page
- Add Machine - Add a new machine via the RackN Portal
- Add Stage - Add a new stage into the Workflow process
- Add Profile - Add a new profile to the RackN Portal

Machines

The Machines page highlights the following information for all machines/nodes the Digital Rebar Provision endpoint is currently aware of. Information shown for each machine:

- Machine Status (Active/Standby)
- Name of the Machine
- UUID
- IP Address
- Workflow Stage
- BootEnv

Selecting a machine name will provide additional information in a new screen:

- Name of the Machine
- UUID
- Description
- IP Address

- Boot Environment (local/)
- Workflow Stage
- Tasks
- Profiles
- Additional Parameters

The top of the page offers the following services via a single button for each action:

- Refresh - Refesh the content on the page to show all machines managed by the RackN Portal
- Filter - Allows the user to show a specific set of machines based on a variety of filters including IP Address, BootEnv, Key, Name, Stage, UUID, Workflow,etc
- Add - Add a new machine via the RackN Portal
- Clone - Clone a machine by selecting a machine listed
- Delete - Delete a machine by selecting a machine listed
- Show - This will show additional features for each machine including Plugin, RAM, NICs, CPUs, Make, and Family

Bulk Actions

The Bulk Actions page allows the operator to perform a series of changes to a selected machine or group of machines. Available options for machine manipulation include:

- Runnable - Select a machine or a set of machines to make them able to run or stop an active process
- Profiles - Select a machine or a set of machines to assign a specific profile for processing
- Workflows - Select a machine or a set of machines to assign a specific workflow for processing
- Stages - Select a machine or a set of machines to assign a specific stage for processing
- Actions - Select a machine or a set of machines to perform an action such as poweron, poweroff, powercycle, createVM, startVM, stop VM, destroyVM, nextbootpxe or nextbootdisk

The top of the page offers the following services via a single button for each action:

- Refresh - Refresh the content on the page
- Filter - Allows the user to show a specific set of machines based on a variety of filters including IP Address, BootEnv, Key, Name, Stage, UUID, Workflow,etc
- Delete - Delete a machine by selecting a machine listed
- Bulk Actions
- Force

Plugins

The Plugin page lists all currently available plugins in the RackN Portal. For each plugin listed the following is included:

- Lock - The plugin is available or disabled/locked for your use
- Name - The name of the plugin
- Description - Details about the plugin

- Provider - The name of the Profile that allows a machine to use the plugin

The top of the page offers the following services via a single button for each action:

- Refresh - Refresh the content on the page
- Filter - Allows the user to show a specific set of plugins based on a variety of filters including Available, Key, Name, Provider, Read Only and Valid
- Add - Select a new plugin to add from the Select Plugin Provider page
- Clone - Make a duplicate of a specific plugin or set of plugins
- Delete - Remove a plugin or set of plugins from the RackN Portal

Info & Preferences

The System Info and Preferences page gives a complete high level overview of all activities in the RackN Portal. It is divided into four sections:

- System Wizard
- Version Inspection
- System Preferences
- System Diagnostics

System Wizard This section contains status information on the availability of the RackN Portal to function. All sections should have a green checkmark unless you are working with a Packet.net or other external environment where a network is not required in which case Subnets will have a red X.

These items are given status for availability:

- Content
- Boot Environment
- Subnets
- ISOs
- Preferences
- Workflows
- Machines

Version Inspection This section contains a list of plugins currently available to the RackN Portal for usage. A Refresh button is available to have the system recheck this information.

System Preferences This section allows the operator to set a variety of global properties for the RackN Portal. A Save button is available to ensure all changes are saved to the system.

These are the properties available for updates:

- Default Workflow
- Default Stage
- Default BootEnv
- Known Token Timeout
- Unknown Token Timeout
- Unknown BootEnv

- BootEvn Logs
- DHCP Logs
- Renderer Logs
- Debug Frontend
- Debug Plugins
- Default Log Level

System Diagnostics This section contains information the system itself including:

- Version tip - Build # of the Digital Rebar Provision being operated by the RackN Portal
- Feature Flags - The list of features supported in the RackN Portal for the connected Digital Rebar Provision instance.
- Endpoint MAC Address and API Port - Machine information on the DRP Endpoint the RackN Portal is connected to
- OS and Architecture - The Operating System and Processor of the Endpoint machine

The top of the page offers the following services via a single button for each action:

- Refresh - Refresh the content on the page

3.17.3 Networking

This section contains information to the DRP endpoint about all networking settings within the available infrastructure.

Switches

This section lists all LLDP discovered switches.

The blue Refresh button will update the list ensuring you have the latest switch information on the network.

Subnets

This section lists all Subnets discovered in the network.

Each Subnet contains the following information:

- Locked or Unlocked
- Name
- Subnet Address
- Start IP Address Range
- End IP Address Range
- Active Time
- Reserve Time

The top of the page contains a series of blue buttons for additional actions:

- Refresh - Refresh the list of Subnets should a new Subnet be in the system

- Filter - Select a specific set of Subnets based on the following options: Active Address, Address Available, Enabled, Key, Name, NextServer, Proxy, ReadOnly, Strategy, Subnet CIDR Address, Valid Subnet
- Add - Add a new Subnet to the system
- Clone - Clone a selected Subnet to add to the system
- Delete - Remove all the selected Subnets

Configure Subnets

Configuring Subnets is a critical first step in Digital Rebar Provision operation. The basic UI will show all configured subnets and provide an easy way to add broadcast subnets based on the known interfaces.

To edit or delete a subnet, click on the name of the subnet to populate the editing area below the list. To create a relay subnet, click on the add subnet link. To create a broadcast subnet, click on the link provided after the name of the unassigned interfaces.

There are two primary types of subnets: broadcast and relay:

- Broadcast subnets are associated with the addresses provided by the Digital Rebar Provision host system. Digital Rebar Provision can have multiple DHCP ranges; however, only one subnet can be assigned per interface _and_ the subnet CIDR must include the IP of the interface (the range should NOT). By convention, subnets associated with an interface will be named as the interface.
- Relay subnets answer requests forwarded from DHCP relays such as those provided by switches. These can be any suitable IP range. Since the Relay subnets are not broadcast, they do not conflict with existing DHCP servers in the environment.

Digital Rebar Provision can operate in a permissive reservation mode or require users to whitelist systems before they are serviced. The *OnlyReservations* flag will operate as a reservations required (whitelist) mode when true; otherwise, Digital Rebar Provision permissive reservation mode will give out addresses to any valid DHCP request.

In additional to serving IPs, DHCP servers provide critical configuration (aka DHCP Options) information to the clients. Setting Option 67, Next Boot, is essential for Digital Rebar Provision to operate as a Provisioner. This information includes next boot (67), gateway (3), domain name (15), DNS (6) and other important information. It is encoded in the responses according to IETF RFC 2132

Consult the Godocs for more details about the specific fields.

Leases

Leases show individual links between tokens and addresses, created through reservation or subnet strategies. Leases remain valid for short periods of time, and cannot be edited. The expiration time of each lease is visible through the UI. Once a lease has expired, it may be removed.

This page shows a list of Leases available in the system.

Reservations

Reservations link tokens to specific IP addresses. This view shows a list of existing reservations along with tokens and strategies associated with each. Currently, MAC is the only available strategy. Reservations may contain options to be applied to connected servers, which are also visible through the UI.

This page shows a list of Reservations available in the system.

3.17.4 Provision

This section contains the setup information to provision a machine. The workflow process uses this information when configuring and deploying a new machine.

Boot Environments

Configuring at least one Boot Environment is a critical first step in Digital Rebar Provision operation. The Digital Rebar CentOS based in-memory discovery image, Sledgehammer, will be installed on first use by default.

The UI will show a complete list of potential Boot Environments with the following information:

- Locked or Unlocked
- Name - The name of the Boot Environment.
- D/L
- ISO - ISO or TAR Image involved with the named Boot Environment
- Description - More information about the specific Boot Environment

The top of the page includes a set of additional actions:

- Refresh - Refresh the list of available Boot Environments for the Endpoint to use
- Filter - Refine the list of Boot Environments based on these options: Available, Key, Name, OnlyUnknown, OSName, ReadOnly, Valid
- Add - Add a new Boot Environment
- Clone - Clone a selected Boot Environment
- Delete - Remove a selected Boot Environment

Templates

Templates contain important instructions for the provisioning process, and are comprised of golang text/template strings. Once templates are rendered along with any assigned parameters, they are used by the BootEnv to boot the target machine. Templates may contain other templates, known as sub-templates.

The UI will show a complete list of Templates with the following information:

- Locked or Unlocked
- ID - Template name
- Preview - Description of what the Template does

The top of the page includes a set of additional actions:

- Refresh - Refresh the list of available Templates for the Endpoint to use
- Filter - Refine the list of Templates based on these options: Available, ID, Key, ReadOnly, Valid
- Add - Add a new Template
- Clone - Clone a selected Template
- Delete - Remove a selected Template

Params

Parameters are passed to a template from a machine, and help to drive the template's functions. They consist of key/value pairs that provide configuration to the renderer. Profiles allow params to be applied in bulk, or they can be attached to templates individually.

The UI will show a complete list of Params with the following information:

- Locked or Unlocked
- Name - Name of the Param
- Type - Value of the Param; e.g. Object, String, Array, etc
- Description - Additional information about the Param

The top of the page includes a set of additional actions:

- Refresh - Refresh the list of available Params for the Endpoint to use
- Filter - Refine the list of Params based on these options: Available, Key, Name, ReadOnly, Valid
- Add - Add a new Param
- Clone - Clone a selected Param
- Delete - Remove a selected Param

Profiles

Profiles provide a convenient way to apply sets of parameters to a machine. Multiple profiles can be assigned to one machine, and will be referenced in the order they are listed. Parameters can be linked to specific profiles through the profiles page, which can then be attached to machines through the machines UI.

The UI will show a complete list of Profiles with the following information:

- Locked or Unlocked
- Name - Name of the Profiles
- Description - Additional information about the Profiles

The top of the page includes a set of additional actions:

- Refresh - Refresh the list of available Profiles for the Endpoint to use
- Filter - Refine the list of Params based on these options: Available, Key, Name, ReadOnly, Valid
- Add - Add a new Profile
- Clone - Clone a selected Profile
- Delete - Remove a selected Profile
- Ansible - Present information about the Ansible Inventory Grid

3.17.5 Control

This section of the UX contains critical information for managing the provisioning of nodes within Digital Rebar. Workflows are the high level instruction path necessary for DRP to perform a provision. Workflows are composed of individual stages which accomplish a specific task.

Workflow

This section lists all available Workflows currently available to execute. Each Workflow has the following details:

- Lock/Unlock
- Name
- Description
- Stages

To the right is a list of ALL available stages that can be used to build a new Workflow.

At the top of the screen are a series of blue boxes offering additional functionality:

- Switch to Change/Stage Map - This screen shows the Workflow processes available and how the process order from one state to the next. Steps can be added to Workflows and a Workflow Wizard is available to create a new Workflow.
- Refresh - Update the Workflow list with the latest available Workflows
- Filter - Select which Workflow to list by Available, Key, Name, ReadOnly, and Valid
- Add - Add a new Workflow
- Clone - Clone a Workflow
- Delete - Delete a Workflow

For more details *Digital Rebar Provision Workflows*

Stages

This section lists all available Stages within the DRP system. Each Stage has the following details:

- Lock/Unlock
- Name
- Boot Environment
- Description

At the top of the screen are a series of blue boxes offering additional functionality:

- Refresh - Update the Stages list with the latest available Stages
- Filter - Select which Stage to list by Available, BootEnv, Key, Name, ReadOnly, Reboot and Valid
- Add - Add a new Stage
- Clone - Clone a Stage
- Delete - Delete a Stage

For more details *Stage*

Tasks

This section lists all available Tasks within the DRP system. Each Task has the following details:

- Lock/Unlock
- Name

- Description

At the top of the screen are a series of blue boxes offering additional functionality:

- Refresh - Update the Task list with the latest available Stages
- Filter - Select which Task to list by Available, Key, Name, ReadOnly, and Valid
- Add - Add a new Task
- Clone - Clone a Task
- Delete - Delete a Task

Task Details

During the boot process, tasks provide additional configuration to machines in the form of templates. BootEnvs will use these sets of templates to construct specific jobs for a machine.

Within a task, templates are processed in the order they are assigned, so it's important to check that templates are attached correctly to a task.

For more details *Task*

Jobs

This section lists all available Jobs within the DRP system. Each Job has the following details:

- State
- Start Time
- Run Time
- Job UUID
- Machine
- Stage
- Task

At the top of the screen are a series of blue boxes offering additional functionality:

- Refresh - Update the Jobs list with the latest available Jobs
- Filter - Select which Task to list by Archived, Available, BootEnv, Current, EndTime, Key, Machine, Previous, ReadOnly, Stage, StartTime, State, Task, Uuid, Workflow, and Valid
- Delete - Delete a Job

Job Details

A job defines a machine's current step in its boot process. After completing a job, the machine creates a new job from the next instruction in the machine's task list.

Machines will only process one job at a time, and jobs are not created until the instant they are required.

For more details *Job*

3.17.6 Synchronize and Upload

This section contains information on the available content packages, ISOs, and Plugins available in the current DRP as well as tools not yet installed on this specific endpoint.

Content Packages

Two separate tables are shown containing the following information:

- Endpoint Content - Manage all the installed content packages on the DRP endpoint : An Upgrade button indicates if the content packages have been updated and a new version needs to be downloaded
- Community & User Name Content - Additional content packages available to the user but not on the DRP endpoint

Content Package Information

- Preview - Provide a complete look into the Content Packag including its Boot Envs, Params, Profiles, Stages, Tasks, and Templates. An option to see the raw JSON is also available
- Diff - Show the latest changes between the current and most recent version of the Content Package
- Upgrade - Upgrade the Content Package to its latest version
- Remove - Remove the Content Package from the current DRP endpoint

The top of the page has a set of blue boxes for additional information:

- Catalog - Gives a complete list of all possible content packages for the DRP endpoint
- Refresh - Update the current list of information on the page
- Show All - Show all options installed and not installed for the DRP endpoint

Boot ISOs

This page shows all available Boot ISOs and Images for the DRP endpoint to use.

The top of the page has a set of blue boxes for additional information:

- Refresh - Update the current list of ISOs
- Upload - Add a new ISO image to the DRP endpoint
- Delete - Remove an ISO image from the DRP endpoint

Plugin Providers

This section contains two separate tables showing Plugins available for the DRP endpoint.

- Endpoint Plugin Providers - Manage all the installed Plugins on the DRP endpoint.
- Organization Plugin Providers - Additional Plugins my organization can run however are not yet installed on my DRP endpoint

Plugin Information

- Plugin Name
- Upgrade - Get the latest code for a specific Plugin on the DRP endpoint

- Remove - Remove the Plugin from the DRP endpoint which will have it appear on the Organization Plugin Providers list
- Transfer - Move the Plugin to the DRP endpoint from the Organization Plugin Providers list

The top of the page has a set of blue boxes for additional information:

- Catalog - Gives a complete list of all possible Plugins for the DRP endpoint
- Upload - Add a new Plugin to the DRP endpoint
- Refresh - Update the current list of information on the page

Support Files

These files are located on the DRP webserver and are available via TFTP to start the PXE boot on new machines.

3.17.7 Endpoint Admin

This section provides information on all the endpoint activity including users and logs as well as a generic Log Out button to exit the RackN Portal.

Users

This section provides a list of all Users currently on the DRP endpoint. Each User has the following information:

- Lock/Unlock
- Name

Additional functionality is available from a series of blue boxes on the top of the page:

- Refresh - Update the list of Users showing on the page
- Filter - List the Users based on the following options: Available, Key, Name, ReadOnly, and Valid
- Add - Add a new User to the DRP endpoint
- Clone - Clone an existing User on the DRP endpoint
- Delete - Delete an existing User on the DRP endpoint

Logs

This section provides a list of all activity on the DRP endpoint via a Log file.

Additional functionality is available from a series of blue boxes on the top of the page:

- Refresh - Update the log data being shown to include the latest new logs

Logout

Press this button to logout of the DRP endpoint.

3.18 Swagger UI

The Digital Rebar Provision UI includes Swagger to allow exploration and testing of the *Digital Rebar Provision API*.

To access the Swagger web interface, point your web browser at your DRP Endpoint IP addres, at the following URL:

```
https://<IP_ADDRESS>:8092/swagger-ui
```

Ensure that the input form contains the same *IP_ADDRESS* reference as your DRP Endpoint. Click the *Authorize* button to obtain an API Token from the Username/Password authentication.

See *Default Template Identity* for default credentials.

Note: Due to the limitations of the Swagger specification, there are important API optimizations that are not exposed in the *swagger.json*. Please review the *Digital Rebar Provision API* for the complete capabilities.

3.19 Digital Rebar Provision Command Line Interface (CLI)

The Digital Rebar Provision Command Line Interface (drpcli) provides a simplified way to interact with the *Digital Rebar Provision API*. The command line tool (`drpcli`) is auto-generated from source code via reference of the API. This means the CLI should implement 100% coverage of the API.

3.19.1 Overview

The CLI provides help for commands and follows a pattern of chained parameters with a few flags for additional modifications.

Some examples are:

```
drpcli bootenvs list
drpcli subnets get mysubnet
drpcli preferences set defaultBootEnv discovery
```

The *drpcli* has help at each layer of command and is the easiest way to figure out what can and can not be done.

```
drpcli help
drpcli bootenvs help
```

Each object in the rs_data_architecture has a CLI subcommand.

Note: VERY IMPORTANT - the **update** commands use the **PATCH** operation for the objects in the *Digital Rebar Provision API*. This has the implication that for map like components (Params sections of rs_model_machine and rs_model_profile) the contents are merged with the existing object. For the Params sections specifically, use the subaction *params* to replace contents.

By default, the CLI will attempt to access the *dr-provision* API endpoint on the localhost at port 8092 with the username and password of *rocketskates* and *r0cketsk8ts*, respectively. All three of these values can be provided by environment variable or command line flag.

Option	Environment Variable	Flag	Format
Username	RS_KEY	-P or –password	String, but when part of RS_KEY it is: username:password
Password	RS_KEY	-U or –username	String, but when part of RS_KEY it is: username:password
Token	RS_TOKEN	N/A	Base64 encoded string from a generate token API call.
Endpoint	RS_ENDPOINT	-E or –endpoint	URL for access, https://IP:PORT. e.g. https://127.0.0.1:8092

Note: It is necessary to specify either a username and password or a token.

Another useful flag is *–format*. this will change the tool output to YAML instead of JSON. This can be helpful when editing files by hand. e.g. *–format yaml*

For Bash users, the drpcli can generate its own bash completion file. Once generated, it is necessary to restart the terminal/shell or reload the completions.

linux

```
sudo drpcli autocomplete /etc/bash_completion.d/drpcli
. /etc/bash_completion
```

Darwin

Assuming that Brew is in use to update and manage bash and bash autocompletion.

```
sudo drpcli autocomplete /usr/local/etc/bash_completion.d/drpcli
. /usr/local/etc/bash_completion
```

3.19.2 Command Line Reference

rs_cli_command

3.20 Digital Rebar Provision API

In general, the Digital Rebar Provision API is documented via the Swagger spec and introspectable for machines via */swagger.json* and for humans via */swagger-ui*.

All API calls are available under */api/v3* based on the Digital Rebar API convention.

3.20.1 API Filters

The API includes index driven filters for large deployments that can be used to pre-filter requests from the API.

The list of available indexes is provided by the `/api/v3/indexes` and `/api/v3/indexes/[model]` calls. These hashs provide a list of all keys that can be used for filters with some additional metadata.

To use the index, simply include one or more indexs and values on the request URI. For example:

```
/api/v3/machines?Runnable=true&Available=true
```

The filter specification allows more complex filters using functions:

- Key=Eq(Value) (that is the same as Key=Value)
- Lt(Value)
- Lte(Value)
- Gt(Value)
- Gte(Value)
- Ne(Value)
- Ranges: * Between(Value1,Value2) (edited) * Except(Value1,Value2)

The query string applies ALL parameters are to be applied (as implied by the & separator). All must match to be returned.

3.20.2 Filtering by Param Value

The API includes specialized filter behavior for Params that allows deep searching models for Param values.

To filter Machines or Profiles by Param values, pass the Param name and value using the normal Field filter specification. When the Field is not found, the backend will search model's Params keys and evalute the filter against the Param value.

3.20.3 Payload Reduction (slim)

Since Params and Meta may contain a lot of data, the API supports the `?slim=[Params,Meta]` option to allow requests to leave these fields unpopulated. If this data is needed, operators will have to request the full object or object's Params or Meta in secondary calls.

```
/api/v3/machines?slim=Params,Meta
```

Only endpoints that offer the `slim-objects` feature flag (v3.9+) will accept this flag.

3.20.4 API Exception & Deprecation Notes

There are times when the API and models have exceptions or changes that do not follow the normal pattern. This section is designed to provide a reference for those exceptions.

This section is intended to provide general information about and functional of the API. We maintain this section for legacy operators, when possible avoid using deprecated features!

Machines.Profile (deprecated by flag: profileless-machine) What would otherwise be Machine.Params is actually embedded under Machines.Profile.Params. This deprecated composition simplifies that precedence calculation for Params by making Machines the top of the Profiles stack. All the other fields in the Machines.Profile are ignored.

Methods

System

GET /system/actions

List system actions System

- **Description:** List System actions

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoSystemResponse*
401	*NoSystemResponse*

POST /system/actions/{cmd}

Call an action on the system.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoSystemResponse*
401	*NoSystemResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /system/actions/{cmd}

List specific action for System

- **Description:** List specific {cmd} action for System

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoSystemResponse*
401	*NoSystemResponse*
400	*ErrorResponse*

PluginProviders

POST /plugin_providers/{name}

Upload a plugin provider to a specific {name}.

- **Produces:** [u'application/json']
- **Consumes:** [u'application/octet-stream']

Parameters

Name	Position	Description	Type
name*	path		string
Body	body		Raw Octet Stream

Responses

Code	Type
201	*PluginProviderInfoResponse*
415	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
507	*ErrorResponse*
409	*ErrorResponse*

DELETE /plugin_providers/{name}

Delete a plugin provider

- **Description:** The plugin provider will be removed from the system.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
422	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
204	*NoContentResponse*
401	*NoContentResponse*

HEAD /plugin_providers/{name}

See if a Plugin Provider exists

- **Description:** Return 200 if the Plugin Provider specified by {name} exists, or return NotFound.

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /plugin_providers/{name}

Get a specific plugin with {name}

- **Produces:** [u'application/json']
- **Description:** Get a specific plugin specified by {name}.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*PluginProviderResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

HEAD /plugin_providers

- **Produces:** [u'application/json']
- **Description:** Stats of the list of plugin_provider on the system to create plugins

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*PluginProvidersResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /plugin_providers

- **Produces:** [u'application/json']
- **Description:** Lists possible plugin_provider on the system to create plugins

Responses

Code	Type
200	*PluginProvidersResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Indexes

GET /indexes

- **Produces:** [u'application/json']
- **Description:** List all static indexes for objects

Responses

Code	Type
200	*IndexesResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

GET /indexes/{prefix}

- **Produces:** [u'application/json']
- **Description:** Get static indexes for a specific object type

Parameters

Name	Position	Description	Type
prefix*	path		string

Responses

Code	Type
200	*IndexResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

GET /indexes/{prefix}/{param}

Get information on a specific index for a specific object type.

- **Produces:** [u'application/json']
- **Description:** Unlike the other routes, you can probe for parameter-defined indexes using this route.

Parameters

Name	Position	Description	Type
prefix*	path		string
param*	path		string

Responses

Code	Type
200	*IndexResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Params

POST /params

Create a Param

- **Description:** Create a Param from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Param*

Responses

Code	Type
201	*ParamResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /params

Stats of the List Params filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /params

Lists Params filtered by some parameters.

- **Description:** This will show all Params by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*ParamsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

PUT /params/{name}

Put a Param

- **Description:** Update a Param specified by {name} using a JSON Param

Parameters

Name	Position	Description	Type
Body*	body		*Param*
name*	path		string

Responses

Code	Type
200	*ParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /params/{name}

Patch a Param

- **Description:** Update a Param specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*ParamResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /params/{name}

See if a Param exists

- **Description:** Return 200 if the Param specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /params/{name}

Delete a Param

- **Description:** Delete a Param specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*ParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /params/{name}

Get a Param

- **Description:** Get the Param specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*ParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Prefs

POST /prefs

Create a Pref

- **Description:** Create a Pref from the provided object

Parameters

Name	Position	Description	Type
Body	body		map of strings to string

Responses

Code	Type
201	*PrefsResponse*
422	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

GET /prefs

Lists Prefs

- **Description:** This will show all Prefs by default

Responses

Code	Type
200	*PrefsResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Events

POST /events

Create an Event

- **Description:** Create an Event from the provided object

Parameters

Name	Position	Description	Type
Body	body		*Event*

Responses

Code	Type
422	*ErrorResponse*
403	*NoContentResponse*
204	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

Templates

POST /templates/{id}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
id*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoTemplateResponse*
401	*NoTemplateResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /templates/{id}/actions/{cmd}

List specific action for a template Template

- **Description:** List specific {cmd} action for a Template specified by {id}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
id*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoTemplateResponse*
401	*NoTemplateResponse*
400	*ErrorResponse*

PUT /templates/{id}

Put a Template

- **Description:** Update a Template specified by {id} using a JSON Template

Parameters

Name	Position	Description	Type
Body*	body		*Template*
id*	path		string

Responses

Code	Type
200	*TemplateResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /templates/{id}

Patch a Template

- **Description:** Update a Template specified by {id} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
id*	path		string

Responses

Code	Type
200	*TemplateResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /templates/{id}

See if a Template exists

- **Description:** Return 200 if the Template specifiec by {id} exists, or return NotFound.

Parameters

Name	Position	Description	Type
id*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /templates/{id}

Delete a Template

- **Description:** Delete a Template specified by {id}

Parameters

Name	Position	Description	Type
id*	path		string

Responses

Code	Type
200	*TemplateResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /templates/{id}

Get a Template

- **Description:** Get the Template specified by {id} or return NotFound.

Parameters

Name	Position	Description	Type
id*	path		string

Responses

Code	Type
200	*TemplateResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /templates/{id}/actions

List template actions Template

- **Description:** List Template actions for a Template specified by {id}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
id*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoTemplateResponse*
401	*NoTemplateResponse*

POST /templates

Create a Template

- **Description:** Create a Template from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Template*

Responses

Code	Type
201	*TemplateResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /templates

Stats of the List Templates filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: ID = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: ID=fred - returns items named fred ID=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
ID	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /templates

Lists Templates filtered by some parameters.

- **Description:** This will show all Templates by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: ID = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: ID=fred - returns items named fred ID=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
ID	query		string

Responses

Code	Type
200	*TemplatesResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

Subnets

PUT /subnets/{name}

Put a Subnet

- **Description:** Update a Subnet specified by {name} using a JSON Subnet

Parameters

Name	Position	Description	Type
Body*	body		*Subnet*
name*	path		string

Responses

Code	Type
200	*SubnetResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /subnets/{name}

Patch a Subnet

- **Description:** Update a Subnet specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*SubnetResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /subnets/{name}

See if a Subnet exists

- **Description:** Return 200 if the Subnet specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /subnets/{name}

Delete a Subnet

- **Description:** Delete a Subnet specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*SubnetResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /subnets/{name}

Get a Subnet

- **Description:** Get the Subnet specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*SubnetResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /subnets/{name}/actions

List subnet actions Subnet

- **Description:** List Subnet actions for a Subnet specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoSubnetResponse*
401	*NoSubnetResponse*

POST /subnets/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoSubnetResponse*
401	*NoSubnetResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /subnets/{name}/actions/{cmd}

List specific action for a subnet Subnet

- **Description:** List specific {cmd} action for a Subnet specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoSubnetResponse*
401	*NoSubnetResponse*
400	*ErrorResponse*

POST /subnets

Create a Subnet

- **Description:** Create a Subnet from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Subnet*

Responses

Code	Type
201	*SubnetResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /subnets

Stats of the List Subnets filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string NextServer = IP Address Subnet = CIDR Address Strategy = string Available = boolean Valid = boolean ReadOnly = boolean Enabled = boolean Proxy = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value

Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Strategy	query		string
NextServer	query		string
Subnet	query		string
Name	query		string
Enabled	query		string
Proxy	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /subnets

Lists Subnets filtered by some parameters.

- **Description:** This will show all Subnets by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string NextServer = IP Address Subnet = CIDR Address Strategy = string Available = boolean Valid = boolean ReadOnly = boolean Enabled = boolean Proxy = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Strategy	query		string
NextServer	query		string
Subnet	query		string
Name	query		string
Enabled	query		string
Proxy	query		string

Responses

Code	Type
200	*SubnetsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

Jobs

POST /jobs/{uuid}/plugin_actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoJobResponse*
401	*NoJobResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /jobs/{uuid}/plugin_actions/{cmd}

List specific action for a job Job

- **Description:** List specific {cmd} action for a Job specified by {uuid}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoJobResponse*
401	*NoJobResponse*
400	*ErrorResponse*

GET /jobs/{uuid}/plugin_actions

List job plugin_actions Job

- **Description:** List Job plugin_actions for a Job specified by {uuid}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
plugin	query		string
os	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoJobResponse*
401	*NoJobResponse*

POST /jobs

Create a Job

- **Description:** Create a Job from the provided object, Only Machine and UUID are used.

Parameters

Name	Position	Description	Type
Body*	body		*Job*

Responses

Code	Type
201	*JobResponse*
202	*JobResponse*
204	*NoContentResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /jobs

Stats of the List Jobs filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Uuid = string Stage = string Task = string State = string Machine = string Archived = boolean StartTime = datetime EndTime = datetime Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Uuid=fred - returns items named fred Uuid=Lt(fred) - returns items that alphabetically less than fred. Uuid=Lt(fred)&Archived=true - returns items with Uuid less than fred and Archived is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Uuid	query		string
Stage	query		string
Task	query		string
State	query		string
Machine	query		string
Archived	query		string
StartTime	query		string
EndTime	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /jobs

Lists Jobs filtered by some parameters.

- **Description:** This will show all Jobs by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Uuid = string Stage = string Task = string State = string Machine = string Archived = boolean StartTime = datetime EndTime = datetime Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Uuid=fred - returns items named fred Uuid=Lt(fred) - returns items that alphabetically less than fred. Uuid=Lt(fred)&Archived=true - returns items with Uuid less than fred and Archived is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Uuid	query		string
Stage	query		string
Task	query		string
State	query		string
Machine	query		string
Archived	query		string
StartTime	query		string
EndTime	query		string

Responses

Code	Type
200	*JobsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

PUT /jobs/{uuid}

Put a Job

- **Description:** Update a Job specified by {uuid} using a JSON Job

Parameters

Name	Position	Description	Type
Body*	body		*Job*
uuid*	path		string

Responses

Code	Type
200	*JobResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*

PATCH /jobs/{uuid}

Patch a Job

- **Description:** Update a Job specified by {uuid} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
uuid*	path		string

Responses

Code	Type
200	*JobResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*

HEAD /jobs/{uuid}

See if a Job exists

- **Description:** Return 200 if the Job specifiec by {uuid} exists, or return NotFound.

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /jobs/{uuid}

Delete a Job

- **Description:** Delete a Job specified by {uuid}

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*JobResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /jobs/{uuid}

Get a Job

- **Description:** Get the Job specified by {uuid} or return NotFound.

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*JobResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /jobs/{uuid}/actions

List job plugin_actions Job

- **Description:** List Job plugin_actions for a Job specified by {uuid}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
plugin	query		string
os	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoJobResponse*
401	*NoJobResponse*

PUT /jobs/{uuid}/log

Append the string to the end of the job's log.

- **Produces:** [u'application/json']
- **Consumes:** [u'application/octet-stream']

Parameters

Name	Position	Description	Type
Body*	body		Raw Octet Stream
uuid*	path		string

Responses

Code	Type
204	*NoContentResponse*
415	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
500	*ErrorResponse*

GET /jobs/{uuid}/log

Get the log for this job

- **Produces:** [u'application/octet-stream', u'application/json']
- **Description:** Get log for the Job specified by {uuid} or return NotFound.

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*JobLogResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Users

PUT /users/{name}

Put a User

- **Description:** Update a User specified by {name} using a JSON User

Parameters

Name	Position	Description	Type
Body*	body		*User*
name*	path		string

Responses

Code	Type
200	*UserResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /users/{name}

Patch a User

- **Description:** Update a User specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*UserResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /users/{name}

See if a User exists

- **Description:** Return 200 if the User specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /users/{name}

Delete a User

- **Description:** Delete a User specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*UserResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /users/{name}

Get a User

- **Description:** Get the User specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*UserResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /users/{name}/token

Get a User Token

- **Description:** Get a token for the User specified by {name} or return error

Parameters

Name	Position	Description	Type
name*	path		string
ttl	query		integer
roles	query		string

Responses

Code	Type
200	*UserTokenResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

GET /users/{name}/actions

List user actions User

- **Description:** List User actions for a User specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoUserResponse*
401	*NoUserResponse*

POST /users/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoUserResponse*
401	*NoUserResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /users/{name}/actions/{cmd}

List specific action for a user User

- **Description:** List specific {cmd} action for a User specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoUserResponse*
401	*NoUserResponse*
400	*ErrorResponse*

PUT /users/{name}/password

Set the password for a user.

- **Description:** Update a User specified by {name} using a JSON User

Parameters

Name	Position	Description	Type
Body*	body		*UserPassword*
name*	path		string

Responses

Code	Type
200	*UserResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

POST /users

Create a User

- **Description:** Create a User from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*User*

Responses

Code	Type
201	*UserResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /users

Stats of the List Users filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean

Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /users

Lists Users filtered by some parameters.

- **Description:** This will show all Users by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*UsersResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

Interfaces

GET /interfaces

- **Produces:** [u'application/json']
- **Description:** Lists possible interfaces on the system to serve DHCP

Responses

Code	Type
200	*InterfacesResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

GET /interfaces/{name}

Get a specific interface with {name}

- **Produces:** [u'application/json']
- **Description:** Get a specific interface specified by {name}.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*InterfaceResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Isos

GET /isos

Lists isos in isos directory

- **Description:** Lists the isos in a directory under /isos.

Responses

Code	Type
200	*IsosResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /isos/{path}

Upload an iso to a specific {path} in the tree under isos.

- **Produces:** [u'application/json']
- **Description:** The iso will be uploaded to the {path} in /isos. The {path} will be created.
- **Consumes:** [u'application/octet-stream']

Parameters

Name	Position	Description	Type
path*	path		string
Body	body		Raw Octet Stream

Responses

Code	Type
201	*IsoInfoResponse*
415	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
507	*ErrorResponse*
409	*ErrorResponse*

DELETE /isos/{path}

Delete an iso to a specific {path} in the tree under isos.

- **Description:** The iso will be removed from the {path} in /isos.

Parameters

Name	Position	Description	Type
path*	path		string

Responses

Code	Type
422	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
204	*NoContentResponse*
401	*NoContentResponse*

GET /isos/{path}

Get a specific Iso with {path}

- **Produces:** [u'application/octet-stream', u'application/json']
- **Description:** Get a specific iso specified by {path} under isos.

Parameters

Name	Position	Description	Type
path*	path		string

Responses

Code	Type
200	*IsoResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Workflows

POST /workflows

Create a Workflow

- **Description:** Create a Workflow from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Workflow*

Responses

Code	Type
201	*WorkflowResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /workflows

Stats of the List Workflows filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string
Reboot	query		string
BootEnv	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /workflows

Lists Workflows filtered by some parameters.

- **Description:** This will show all Workflows by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string
Reboot	query		string
BootEnv	query		string

Responses

Code	Type
200	*WorkflowsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /workflows/{name}/actions

List workflow actions Workflow

- **Description:** List Workflow actions for a Workflow specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoWorkflowResponse*
401	*NoWorkflowResponse*

POST /workflows/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoWorkflowResponse*
401	*NoWorkflowResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /workflows/{name}/actions/{cmd}

List specific action for a workflow Workflow

- **Description:** List specific {cmd} action for a Workflow specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoWorkflowResponse*
401	*NoWorkflowResponse*
400	*ErrorResponse*

PUT /workflows/{name}

Put a Workflow

- **Description:** Update a Workflow specified by {name} using a JSON Workflow

Parameters

Name	Position	Description	Type
Body*	body		*Workflow*
name*	path		string

Responses

Code	Type
200	*WorkflowResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /workflows/{name}

Patch a Workflow

- **Description:** Update a Workflow specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*WorkflowResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /workflows/{name}

See if a Workflow exists

- **Description:** Return 200 if the Workflow specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /workflows/{name}

Delete a Workflow

- **Description:** Delete a Workflow specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*WorkflowResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /workflows/{name}

Get a Workflow

- **Description:** Get the Workflow specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*WorkflowResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Leases

POST /leases/{address}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
address*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoLeaseResponse*
401	*NoLeaseResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /leases/{address}/actions/{cmd}

List specific action for a lease Lease

- **Description:** List specific {cmd} action for a Lease specified by {address}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
address*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoLeaseResponse*
401	*NoLeaseResponse*
400	*ErrorResponse*

GET /leases/{address}/actions

List lease actions Lease

- **Description:** List Lease actions for a Lease specified by {address}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
address*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoLeaseResponse*
401	*NoLeaseResponse*

HEAD /leases

Stats of the List Leases filtered by some parameters.

- **Description:** This return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

Functional Indexs: Addr = IP Address Token = string Strategy = string ExpireTime = Date/Time Available = boolean Valid = boolean ReadOnly = boolean

Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Addr	query		string
Token	query		string
Strategy	query		string
ExpireTime	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /leases

Lists Leases filtered by some parameters.

- **Description:** This will show all Leases by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Addr = IP Address Token = string Strategy = string ExpireTime = Date/Time Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Addr	query		string
Token	query		string
Strategy	query		string
ExpireTime	query		string

Responses

Code	Type
200	*LeasesResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

HEAD /leases/{address}

See if a Lease exists

- **Description:** Return 200 if the Lease specifiec by {address} exists, or return NotFound.

Parameters

Name	Position	Description	Type
address*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /leases/{address}

Delete a Lease

- **Description:** Delete a Lease specified by {address}

Parameters

Name	Position	Description	Type
address*	path		string

Responses

Code	Type
200	*LeaseResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /leases/{address}

Get a Lease

- **Description:** Get the Lease specified by {address} or return NotFound.

Parameters

Name	Position	Description	Type
address*	path		string

Responses

Code	Type
200	*LeaseResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

Info

GET /info

Return current system info.

- **Produces:** [u'application/json']

Responses

Code	Type
200	*InfoResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Files

GET /files

Lists files in files directory or subdirectory per query parameter

- **Description:** Lists the files in a directory under /files. path=<path to return> Path defaults to /

Parameters

Name	Position	Description	Type
path	query		string

Responses

Code	Type
200	*FilesResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /files/{path}

Upload a file to a specific {path} in the tree under files.

- **Produces:** [u'application/json']
- **Description:** The file will be uploaded to the {path} in /files. The {path} will be created.
- **Consumes:** [u'application/octet-stream']

Parameters

Name	Position	Description	Type
path*	path		string
Body	body		Raw Octet Stream

Responses

Code	Type
201	*FileInfoResponse*
415	*ErrorResponse*
404	*ErrorResponse*
403	*ErrorResponse*
401	*NoContentResponse*
400	*ErrorResponse*
507	*ErrorResponse*
409	*ErrorResponse*

DELETE /files/{path}

Delete a file to a specific {path} in the tree under files.

- **Description:** The file will be removed from the {path} in /files.

Parameters

Name	Position	Description	Type
path*	path		string

Responses

Code	Type
422	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
204	*NoContentResponse*
401	*NoContentResponse*

GET /files/{path}

Get a specific File with {path}

- **Produces:** [u'application/octet-stream', u'application/json']
- **Description:** Get a specific file specified by {path} under files.

Parameters

Name	Position	Description	Type
path*	path		string

Responses

Code	Type
200	*FileResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Roles

POST /roles/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoRoleResponse*
401	*NoRoleResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /roles/{name}/actions/{cmd}

List specific action for a role Role

- **Description:** List specific {cmd} action for a Role specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoRoleResponse*
401	*NoRoleResponse*
400	*ErrorResponse*

POST /roles

Create a Role

- **Description:** Create a Role from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Role*

Responses

Code	Type
201	*RoleResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /roles

Stats of the List Roles filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /roles

Lists Roles filtered by some parameters.

- **Description:** This will show all Roles by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*RolesResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

PUT /roles/{name}

Put a Role

- **Description:** Update a Role specified by {name} using a JSON Role

Parameters

Name	Position	Description	Type
Body*	body		*Role*
name*	path		string

Responses

Code	Type
200	*RoleResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /roles/{name}

Patch a Role

- **Description:** Update a Role specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*RoleResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /roles/{name}

See if a Role exists

- **Description:** Return 200 if the Role specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /roles/{name}

Delete a Role

- **Description:** Delete a Role specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*RoleResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /roles/{name}

Get a Role

- **Description:** Get the Role specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*RoleResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /roles/{name}/actions

List role actions Role

- **Description:** List Role actions for a Role specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoRoleResponse*
401	*NoRoleResponse*

Profiles

PUT /profiles/{name}

Put a Profile

- **Description:** Update a Profile specified by {name} using a JSON Profile

Parameters

Name	Position	Description	Type
Body*	body		*Profile*
decode	query		string
name*	path		string

Responses

Code	Type
200	*ProfileResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*

PATCH /profiles/{name}

Patch a Profile

- **Description:** Update a Profile specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
decode	query		string
name*	path		string

Responses

Code	Type
200	*ProfileResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /profiles/{name}

See if a Profile exists

- **Description:** Return 200 if the Profile specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /profiles/{name}

Delete a Profile

- **Description:** Delete a Profile specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*ProfileResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /profiles/{name}

Get a Profile

- **Description:** Get the Profile specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*ProfileResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /profiles/{name}/actions

List profile actions Profile

- **Description:** List Profile actions for a Profile specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoProfileResponse*
401	*NoProfileResponse*

POST /profiles/{name}/params/{key}

- **Description:** Set as single Parameter {key} for a profile specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
key*	path		string
Body*	body		Raw Octet Stream

Responses

Code	Type
200	*ProfileParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

GET /profiles/{name}/params/{key}

Delete a single profile parameter

- **Description:** Delete a single parameter {key} for a Profile specified by {uuid}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
key*	path		string

Responses

Code	Type
200	*ProfileParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /profiles

Create a Profile

- **Description:** Create a Profile from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Profile*

Responses

Code	Type
201	*ProfileResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /profiles

Stats of the List Profiles filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /profiles

Lists Profiles filtered by some parameters.

- **Description:** This will show all Profiles by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred.

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*ProfilesResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

POST /profiles/{name}/params

- **Description:** Sets parameters for a profile specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ProfileParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

PATCH /profiles/{name}/params

- **Description:** Update params for Profile {name} with the passed-in patch

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
decode	query		string
name*	path		string

Responses

Code	Type
200	*ProfileParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

GET /profiles/{name}/params

List profile params Profile

- **Description:** List Profile parms for a Profile specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*ProfileParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /profiles/{name}/pubkey

Get the public key for secure params on a profile

- **Description:** Get the public key for a Profile specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*PubKeyResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

POST /profiles/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoProfileResponse*
401	*NoProfileResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /profiles/{name}/actions/{cmd}

List specific action for a profile Profile

- **Description:** List specific {cmd} action for a Profile specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoProfileResponse*
401	*NoProfileResponse*
400	*ErrorResponse*

DELETE /profiles/{uuid}/params/{key}

Delete a single profile parameter

- **Description:** Delete a single parameter {key} for a Profile specified by {uuid}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
key*	path		string

Responses

Code	Type
200	*ProfileParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Stages

POST /stages/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoStageResponse*
401	*NoStageResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /stages/{name}/actions/{cmd}

List specific action for a stage Stage

- **Description:** List specific {cmd} action for a Stage specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoStageResponse*
401	*NoStageResponse*
400	*ErrorResponse*

POST /stages

Create a Stage

- **Description:** Create a Stage from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Stage*

Responses

Code	Type
201	*StageResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /stages

Stats of the List Stages filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string
Reboot	query		string
BootEnv	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /stages

Lists Stages filtered by some parameters.

- **Description:** This will show all Stages by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string
Reboot	query		string
BootEnv	query		string

Responses

Code	Type
200	*StagesResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

PUT /stages/{name}

Put a Stage

- **Description:** Update a Stage specified by {name} using a JSON Stage

Parameters

Name	Position	Description	Type
Body*	body		*Stage*
name*	path		string

Responses

Code	Type
200	*StageResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /stages/{name}

Patch a Stage

- **Description:** Update a Stage specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*StageResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /stages/{name}

See if a Stage exists

- **Description:** Return 200 if the Stage specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /stages/{name}

Delete a Stage

- **Description:** Delete a Stage specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*StageResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /stages/{name}

Get a Stage

- **Description:** Get the Stage specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*StageResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /stages/{name}/actions

List stage actions Stage

- **Description:** List Stage actions for a Stage specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoStageResponse*
401	*NoStageResponse*

Tenants

POST /tenants/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoTenantResponse*
401	*NoTenantResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /tenants/{name}/actions/{cmd}

List specific action for a tenant Tenant

- **Description:** List specific {cmd} action for a Tenant specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoTenantResponse*
401	*NoTenantResponse*
400	*ErrorResponse*

POST /tenants

Create a Tenant

- **Description:** Create a Tenant from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Tenant*

Responses

Code	Type
201	*TenantResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /tenants

Stats of the List Tenants filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /tenants

Lists Tenants filtered by some parameters.

- **Description:** This will show all Tenants by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Reboot = boolean BootEnv = string Available = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*TenantsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

PUT /tenants/{name}

Put a Tenant

- **Description:** Update a Tenant specified by {name} using a JSON Tenant

Parameters

Name	Position	Description	Type
Body*	body		*Tenant*
name*	path		string

Responses

Code	Type
200	*TenantResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /tenants/{name}

Patch a Tenant

- **Description:** Update a Tenant specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*TenantResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /tenants/{name}

See if a Tenant exists

- **Description:** Return 200 if the Tenant specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /tenants/{name}

Delete a Tenant

- **Description:** Delete a Tenant specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*TenantResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /tenants/{name}

Get a Tenant

- **Description:** Get the Tenant specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*TenantResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /tenants/{name}/actions

List tenant actions Tenant

- **Description:** List Tenant actions for a Tenant specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoTenantResponse*
401	*NoTenantResponse*

Tasks

PUT /tasks/{name}

Put a Task

- **Description:** Update a Task specified by {name} using a JSON Task

Parameters

Name	Position	Description	Type
Body*	body		*Task*
name*	path		string

Responses

Code	Type
200	*TaskResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /tasks/{name}

Patch a Task

- **Description:** Update a Task specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*TaskResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /tasks/{name}

See if a Task exists

- **Description:** Return 200 if the Task specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /tasks/{name}

Delete a Task

- **Description:** Delete a Task specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*TaskResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /tasks/{name}

Get a Task

- **Description:** Get the Task specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*TaskResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /tasks/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoTaskResponse*
401	*NoTaskResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /tasks/{name}/actions/{cmd}

List specific action for a task Task

- **Description:** List specific {cmd} action for a Task specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoTaskResponse*
401	*NoTaskResponse*
400	*ErrorResponse*

POST /tasks

Create a Task

- **Description:** Create a Task from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Task*

Responses

Code	Type
201	*TaskResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /tasks

Stats of the List Tasks filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Provider = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /tasks

Lists Tasks filtered by some parameters.

- **Description:** This will show all Tasks by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Provider = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string

Responses

Code	Type
200	*TasksResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /tasks/{name}/actions

List task actions Task

- **Description:** List Task actions for a Task specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoTaskResponse*
401	*NoTaskResponse*

Logs

GET /logs

- **Produces:** [u'application/json']
- **Description:** Return current contents of the log buffer

Responses

Code	Type
200	*LogResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Reservations

POST /reservations/{address}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
address*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoReservationResponse*
401	*NoReservationResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /reservations/{address}/actions/{cmd}

List specific action for a reservation Reservation

- **Description:** List specific {cmd} action for a Reservation specified by {address}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
address*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoReservationResponse*
401	*NoReservationResponse*
400	*ErrorResponse*

POST /reservations

Create a Reservation

- **Description:** Create a Reservation from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Reservation*

Responses

Code	Type
201	*ReservationResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /reservations

Stats of the List Reservations filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Addr = IP Address Token = string Strategy = string NextServer = IP Address Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Addr	query		string
Token	query		string
Strategy	query		string
NextServer	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /reservations

Lists Reservations filtered by some parameters.

- **Description:** This will show all Reservations by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Addr = IP Address Token = string Strategy = string NextServer = IP Address Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Addr	query		string
Token	query		string
Strategy	query		string
NextServer	query		string

Responses

Code	Type
200	*ReservationsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /reservations/{address}/actions

List reservation actions Reservation

- **Description:** List Reservation actions for a Reservation specified by {address}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
address*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoReservationResponse*
401	*NoReservationResponse*

PUT /reservations/{address}

Put a Reservation

- **Description:** Update a Reservation specified by {address} using a JSON Reservation

Parameters

Name	Position	Description	Type
Body*	body		*Reservation*
address*	path		string

Responses

Code	Type
200	*ReservationResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /reservations/{address}

Patch a Reservation

- **Description:** Update a Reservation specified by {address} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
address*	path		string

Responses

Code	Type
200	*ReservationResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /reservations/{address}

See if a Reservation exists

- **Description:** Return 200 if the Reservation specific by {address} exists, or return NotFound.

Parameters

Name	Position	Description	Type
address*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /reservations/{address}

Delete a Reservation

- **Description:** Delete a Reservation specified by {address}

Parameters

Name	Position	Description	Type
address*	path		string

Responses

Code	Type
200	*ReservationResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*

GET /reservations/{address}

Get a Reservation

- **Description:** Get the Reservation specified by {address} or return NotFound.

Parameters

Name	Position	Description	Type
address*	path		string

Responses

Code	Type
200	*ReservationResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

BootEnvs

PUT /bootenvs/{name}

Put a BootEnv

- **Description:** Update a BootEnv specified by {name} using a JSON BootEnv

Parameters

Name	Position	Description	Type
Body*	body		*BootEnv*
name*	path		string

Responses

Code	Type
200	*BootEnvResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /bootenvs/{name}

Patch a BootEnv

- **Description:** Update a BootEnv specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
name*	path		string

Responses

Code	Type
200	*BootEnvResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /bootenvs/{name}

See if a BootEnv exists

- **Description:** Return 200 if the BootEnv specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /bootenvs/{name}

Delete a BootEnv

- **Description:** Delete a BootEnv specified by {name}

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*BootEnvResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /bootenvs/{name}

Get a BootEnv

- **Description:** Get the BootEnv specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*BootEnvResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /bootenvs

Create a BootEnv

- **Description:** Create a BootEnv from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*BootEnv*

Responses

Code	Type
201	*BootEnvResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /bootenvs

Stats of the List BootEnvs filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean OnlyUnknown = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
OnlyUnknown	query		string
Name	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /bootenvs

Lists BootEnvs filtered by some parameters.

- **Description:** This will show all BootEnvs by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Available = boolean Valid = boolean ReadOnly = boolean OnlyUnknown = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
OnlyUnknown	query		string
Name	query		string

Responses

Code	Type
200	*BootEnvsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

POST /bootenvs/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /bootenvs/{name}/actions/{cmd}

List specific action for a bootenv BootEnv

- **Description:** List specific {cmd} action for a BootEnv specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

GET /bootenvs/{name}/actions

List bootenv actions BootEnv

- **Description:** List BootEnv actions for a BootEnv specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Meta

PATCH /meta/{type}/{id}

Patch metadata on an Object of {type} with an ID of {id}

- **Description:** Update metadata on a specific Object using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
type*	path		string
id*	path		string

Responses

Code	Type
200	*MetasResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /meta/{type}/{id}

Get Metadata for an Object of {type} idendified by {id}

- **Description:** Get the appropriate Metadata or return NotFound.

Parameters

Name	Position	Description	Type
type*	path		string
id*	path		string

Responses

Code	Type
200	*MetaResponse*
403	*NoContentRespons*
401	*NoContentResponse*

Machines

PUT /machines/{uuid}

Put a Machine

- **Description:** Update a Machine specified by {uuid} using a JSON Machine

Parameters

Name	Position	Description	Type
force	query		string
Body*	body		*Machine*
uuid*	path		string

Responses

Code	Type
200	*MachineResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /machines/{uuid}

Patch a Machine

- **Description:** Update a Machine specified by {uuid} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
force	query		string
Body*	body		*Patch*
uuid*	path		string

Responses

Code	Type
200	*MachineResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /machines/{uuid}

See if a Machine exists

- **Description:** Return 200 if the Machine specifiec by {uuid} exists, or return NotFound.

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /machines/{uuid}

Delete a Machine

- **Description:** Delete a Machine specified by {uuid}

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*MachineResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /machines/{uuid}

Get a Machine

- **Description:** Get the Machine specified by {uuid} or return NotFound.

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*MachineResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /machines/{uuid}/actions

List machine actions Machine

- **Description:** List Machine actions for a Machine specified by {uuid}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /machines/{uuid}/pubkey

Get the public key for secure params on a machine

- **Description:** Get the public key for a Machine specified by {uuid}

Parameters

Name	Position	Description	Type
uuid*	path		string

Responses

Code	Type
200	*PubKeyResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

POST /machines

Create a Machine

- **Description:** Create a Machine from the provided object

Parameters

Name	Position	Description	Type
force	query		string
Body*	body		*Machine*

Responses

Code	Type
201	*MachineResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /machines

Stats of the List Machines filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  X-DRP-LIST-COUNT - number of objects in the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Uuid = UUID string Name = string BootEnv = string Address = IP Address Runnable = true/false Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Uuid	query		string
Name	query		string
BootEnv	query		string
Address	query		string
Runnable	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /machines

Lists Machines filtered by some parameters.

- **Description:** This will show all Machines by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Uuid = UUID string Name = string BootEnv = string Address = IP Address Runnable = true/false Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Uuid	query		string
Name	query		string
BootEnv	query		string
Address	query		string
Runnable	query		string

Responses

Code	Type
200	*MachinesResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

POST /machines/{uuid}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /machines/{uuid}/actions/{cmd}

List specific action for a machine Machine

- **Description:** List specific {cmd} action for a Machine specified by {uuid}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
uuid*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*

POST /machines/{uuid}/params

- **Description:** Sets parameters for a machine specified by {uuid}

Parameters

Name	Position	Description	Type
Body*	body		map of strings to object
uuid*	path		string

Responses

Code	Type
200	*MachineParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

PATCH /machines/{uuid}/params

- **Description:** Update params for Machine {uuid} with the passed-in patch

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
uuid*	path		string

Responses

Code	Type
200	*MachineParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

GET /machines/{uuid}/params

List machine params Machine

- **Description:** List Machine parms for a Machine specified by {uuid}

Parameters

Name	Position	Description	Type
aggregate	query		string
decode	query		string
uuid*	path		string

Responses

Code	Type
200	*MachineParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /machines/{uuid}/params/{key}

- **Description:** Set as single Parameter {key} for a machine specified by {uuid}

Parameters

Name	Position	Description	Type
Body*	body		Raw Octet Stream
uuid*	path		string
key*	path		string

Responses

Code	Type
200	*MachineParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

DELETE /machines/{uuid}/params/{key}

Delete a single machine parameter

- **Description:** Delete a single parameter {key} for a Machine specified by {uuid}

Parameters

Name	Position	Description	Type
aggregate	query		string
decode	query		string
uuid*	path		string
key*	path		string

Responses

Code	Type
200	*MachineParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /machines/{uuid}/params/{key}

Delete a single machine parameter

- **Description:** Delete a single parameter {key} for a Machine specified by {uuid}

Parameters

Name	Position	Description	Type
aggregate	query		string
decode	query		string
uuid*	path		string
key*	path		string

Responses

Code	Type
200	*MachineParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

Plugins

POST /plugins

Create a Plugin

- **Description:** Create a Plugin from the provided object

Parameters

Name	Position	Description	Type
Body*	body		*Plugin*

Responses

Code	Type
201	*PluginResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /plugins

Stats of the List Plugins filtered by some parameters.

- **Description:** This will return headers with the stats of the list.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Provider = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value

Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string
Provider	query		string

Responses

Code	Type
200	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

GET /plugins

Lists Plugins filtered by some parameters.

- **Description:** This will show all Plugins by default.

  You may specify: Offset = integer, 0-based inclusive starting point in filter data. Limit = integer, number of items to return

  Functional Indexs: Name = string Provider = string Available = boolean Valid = boolean ReadOnly = boolean

  Functions: Eq(value) = Return items that are equal to value Lt(value) = Return items that are less than value Lte(value) = Return items that less than or equal to value Gt(value) = Return items that are greater than value Gte(value) = Return items that greater than or equal to value Between(lower,upper) = Return items that are inclusively between lower and upper Except(lower,upper) = Return items that are not inclusively between lower and upper

  Example: Name=fred - returns items named fred Name=Lt(fred) - returns items that alphabetically less than fred. Name=Lt(fred)&Available=true - returns items with Name less than fred and Available is true

Parameters

Name	Position	Description	Type
offset	query		integer
limit	query		integer
Available	query		string
Valid	query		string
ReadOnly	query		string
Name	query		string
Provider	query		string

Responses

Code	Type
200	*PluginsResponse*
403	*NoContentResponse*
401	*NoContentResponse*
406	*ErrorResponse*

POST /plugins/{name}/params/{key}

- **Description:** Set as single Parameter {key} for a plugin specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
key*	path		string
Body*	body		Raw Octet Stream

Responses

Code	Type
200	*PluginParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

GET /plugins/{name}/params/{key}

Delete a single plugin parameter

- **Description:** Delete a single parameter {key} for a Plugin specified by {uuid}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
key*	path		string

Responses

Code	Type
200	*PluginParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

POST /plugins/{name}/params

- **Description:** Sets parameters for a plugin specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
Body*	body		map of strings to object

Responses

Code	Type
200	*PluginParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

PATCH /plugins/{name}/params

- **Description:** Update params for Plugin {name} with the passed-in patch

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*PluginParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
409	*ErrorResponse*
401	*NoContentResponse*

GET /plugins/{name}/params

List plugin params Plugin

- **Description:** List Plugin parms for a Plugin specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*PluginParamsResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /plugins/{name}/pubkey

Get the public key for secure params on a plugin

- **Description:** Get the public key for a Plugin specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*PubKeyResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

DELETE /plugins/{uuid}/params/{key}

Delete a single plugin parameter

- **Description:** Delete a single parameter {key} for a Plugin specified by {uuid}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string
key*	path		string

Responses

Code	Type
200	*PluginParamResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

PUT /plugins/{name}

Put a Plugin

- **Description:** Update a Plugin specified by {name} using a JSON Plugin

Parameters

Name	Position	Description	Type
Body*	body		*Plugin*
decode	query		string
name*	path		string

Responses

Code	Type
200	*PluginResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

PATCH /plugins/{name}

Patch a Plugin

- **Description:** Update a Plugin specified by {name} using a RFC6902 Patch structure

Parameters

Name	Position	Description	Type
Body*	body		*Patch*
decode	query		string
name*	path		string

Responses

Code	Type
200	*PluginResponse*
406	*ErrorResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
400	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

HEAD /plugins/{name}

See if a Plugin exists

- **Description:** Return 200 if the Plugin specifiec by {name} exists, or return NotFound.

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*NoContentResponse*
404	*NoContentResponse*
403	*NoContentResponse*
401	*NoContentResponse*

DELETE /plugins/{name}

Delete a Plugin

- **Description:** Delete a Plugin specified by {name}

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*PluginResponse*
404	*ErrorResponse*
403	*NoContentResponse*
422	*ErrorResponse*
401	*NoContentResponse*

GET /plugins/{name}

Get a Plugin

- **Description:** Get the Plugin specified by {name} or return NotFound.

Parameters

Name	Position	Description	Type
decode	query		string
name*	path		string

Responses

Code	Type
200	*PluginResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*

GET /plugins/{name}/actions

List plugin actions Plugin

- **Description:** List Plugin actions for a Plugin specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionsResponse*
404	*ErrorResponse*
403	*NoPluginResponse*
401	*NoPluginResponse*

POST /plugins/{name}/actions/{cmd}

Call an action on the node.

- **Description:** Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string
Body*	body		map of strings to object

Responses

Code	Type
200	*ActionPostResponse*
404	*ErrorResponse*
403	*NoPluginResponse*
401	*NoPluginResponse*
400	*ErrorResponse*
409	*ErrorResponse*

GET /plugins/{name}/actions/{cmd}

List specific action for a plugin Plugin

- **Description:** List specific {cmd} action for a Plugin specified by {name}

  Optionally, a query parameter can be used to limit the scope to a specific plugin. e.g. ?plugin=fred

Parameters

Name	Position	Description	Type
name*	path		string
cmd*	path		string
plugin	query		string

Responses

Code	Type
200	*ActionResponse*
404	*ErrorResponse*
403	*NoPluginResponse*
401	*NoPluginResponse*
400	*ErrorResponse*

Contents

POST /contents

- **Description:** Create content into Digital Rebar Provision

Parameters

Name	Position	Description	Type
Body	body		*Content*

Responses

Code	Type
201	*ContentSummaryResponse*
415	*ErrorResponse*
404	*ErrorResponse*
403	*ErrorResponse*
500	*ErrorResponse*
401	*NoContentResponse*
400	*ErrorResponse*
507	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /contents

- **Produces:** [u'application/json']
- **Description:** Lists possible contents on the system to serve DHCP

Responses

Code	Type
200	*ContentsResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

PUT /contents/{name}

- **Description:** Replace content in Digital Rebar Provision

Parameters

Name	Position	Description	Type
Body	body		*Content*
name*	path		string

Responses

Code	Type
200	*ContentSummaryResponse*
415	*ErrorResponse*
404	*ErrorResponse*
403	*ErrorResponse*
500	*ErrorResponse*
401	*NoContentResponse*
400	*ErrorResponse*
507	*ErrorResponse*
422	*ErrorResponse*
409	*ErrorResponse*

DELETE /contents/{name}

Delete a content set.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
204	*NoContentResponse*
404	*ErrorResponse*
403	*NoContentResponse*
401	*NoContentResponse*
422	*ErrorResponse*
409	*ErrorResponse*

GET /contents/{name}

Get a specific content with {name}

- **Produces:** [u'application/json']
- **Description:** Get a specific content specified by {name}.

Parameters

Name	Position	Description	Type
name*	path		string

Responses

Code	Type
200	*ContentResponse*
404	*ErrorResponse*
403	*NoContentResponse*
500	*ErrorResponse*
401	*NoContentResponse*

Responses

ReservationsResponse

ReservationsResponse

- **Description:** ReservationsResponse returned on a successful GET of all the reservations

Parameters

Name	Position	Description	Type
Payload	Body		*array of Reservation*

IsoResponse

IsoResponse

- **Description:** IsoResponse returned on a successful GET of an iso

Parameters

Name	Position	Description	Type
Payload	Body		Raw Octet Stream

PluginParamsResponse

PluginParamsResponse

- **Description:** PluginParamsResponse return on a successful GET of all Plugin's Params

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

FileInfoResponse

FileInfoResponse

- **Description:** FileInfoResponse returned on a successful upload of a file

Parameters

Name	Position	Description	Type
Payload	Body		*BlobInfo*

IsosResponse

IsosResponse

- **Description:** IsosResponse returned on a successful GET of isos

Parameters

Name	Position	Description	Type
Payload	Body		*IsoPaths*

ActionResponse

ActionResponse

- **Description:** ActionResponse return on a successful GET of a single Action

Parameters

Name	Position	Description	Type
Payload	Body		*AvailableAction*

RoleResponse

RoleResponse

- **Description:** RoleResponse returned on a successful GET, PUT, PATCH, or POST of a single role

Parameters

Name	Position	Description	Type
Payload	Body		*Role*

PrefsResponse

PrefsResponse

- **Description:** PrefsResponse returned on a successful GET of all preferences

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to string

IndexesResponse

IndexesResponse

- **Description:** IndexesResponse lists all the static indexes for all the object types

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

ParamResponse

ParamResponse

- **Description:** ParamResponse returned on a successful GET, PUT, PATCH, or POST of a single param

Parameters

Name	Position	Description	Type
Payload	Body		*Param*

IndexResponse

IndexResponse

- **Description:** IndexResponse lists all of the static indexes for a specific type of object

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to Unknown

PluginProviderInfoResponse

PluginProviderInfoResponse

- **Description:** PluginProviderInfoResponse returned on a successful upload of an iso

Parameters

Name	Position	Description	Type
Payload	Body		*PluginProviderUploadInfo*

LeaseResponse

LeaseResponse

- **Description:** LeaseResponse returned on a successful GET, PUT, PATCH, or POST of a single lease

Parameters

Name	Position	Description	Type
Payload	Body		*Lease*

ActionsResponse

ActionsResponse

- **Description:** ActionsResponse return on a successful GET of all Actions

Parameters

Name	Position	Description	Type
Payload	Body		*array of AvailableAction*

UsersResponse

UsersResponse

- **Description:** UsersResponse returned on a successful GET of all the users

Parameters

Name	Position	Description	Type
Payload	Body		*array of User*

JobsResponse

JobsResponse

- **Description:** JobsResponse return on a successful GET of all Jobs

Parameters

Name	Position	Description	Type
Payload	Body		*array of Job*

TasksResponse

TasksResponse

- **Description:** TasksResponse return on a successful GET of all Tasks

Parameters

Name	Position	Description	Type
Payload	Body		*array of Task*

RolesResponse

RolesResponse

- **Description:** RolesResponse returned on a successful GET of all the roles

Parameters

Name	Position	Description	Type
Payload	Body		*array of Role*

ErrorResponse

ErrorResponse

- **Description:** ErrorResponse is returned whenever an error occurs

Parameters

Name	Position	Description	Type
Payload	Body		*Error*

MachineResponse

MachineResponse

- **Description:** MachineResponse return on a successful GET, PUT, PATCH or POST of a single Machine

Parameters

Name	Position	Description	Type
Payload	Body		*Machine*

ReservationResponse

ReservationResponse

- **Description:** ReservationResponse returned on a successful GET, PUT, PATCH, or POST of a single reservation

Parameters

Name	Position	Description	Type
Payload	Body		*Reservation*

StagesResponse

StagesResponse

- **Description:** StagesResponse returned on a successful GET of all the stages

Parameters

Name	Position	Description	Type
Payload	Body		*array of Stage*

PluginParamResponse

PluginParamResponse

- **Description:** PluginParamResponse return on a successful GET of one Plugin's Param

Parameters

Name	Position	Description	Type
Payload	Body		Raw Octet Stream

ProfilesResponse

ProfilesResponse

- **Description:** ProfilesResponse returned on a successful GET of all the profiles

Parameters

Name	Position	Description	Type
Payload	Body		*array of Profile*

BootEnvsResponse

BootEnvsResponse

- **Description:** BootEnvsResponse returned on a successful GET of all the bootenvs

Parameters

Name	Position	Description	Type
Payload	Body		*array of BootEnv*

PluginsResponse

PluginsResponse

- **Description:** PluginsResponse return on a successful GET of all Plugins

Parameters

Name	Position	Description	Type
Payload	Body		*array of Plugin*

InterfacesResponse

InterfacesResponse

- **Description:** InterfacesResponse returned on a successful GET of all interfaces

Parameters

Name	Position	Description	Type
Payload	Body		*array of Interface*

ParamParamsResponse

ParamParamsResponse

- **Description:** ParamParamsResponse return on a successful GET of all Param's Params

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

WorkflowResponse

WorkflowResponse

- **Description:** WorkflowResponse returned on a successful GET, PUT, PATCH, or POST of a single workflow

Parameters

Name	Position	Description	Type
Payload	Body		*Workflow*

SubnetResponse

SubnetResponse

- **Description:** SubnetResponse returned on a successful GET, PUT, PATCH, or POST of a single subnet

Parameters

Name	Position	Description	Type
Payload	Body		*Subnet*

PubKeyResponse

PubKeyResponse

- **Description:** PubKeyResponse is returned on a successful GET of a Paramer public key for secure params.

Parameters

Name	Position	Description	Type
Payload	Body		No Data Returned

JobParamsResponse

JobParamsResponse

- **Description:** JobParamsResponse return on a successful GET of all Job's Params

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

ProfileResponse

ProfileResponse

- **Description:** ProfileResponse returned on a successful GET, PUT, PATCH, or POST of a single profile

Parameters

Name	Position	Description	Type
Payload	Body		*Profile*

PluginResponse

PluginResponse

- **Description:** PluginResponse return on a successful GET, PUT, PATCH or POST of a single Plugin

Parameters

Name	Position	Description	Type
Payload	Body		*Plugin*

ProfileParamsResponse

ProfileParamsResponse

- **Description:** ProfileParamsResponse return on a successful GET of all Profile's Params

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

JobActionsResponse

JobActionsResponse

- **Description:** JobActionsResponse return on a successful GET of a Job's actions

Parameters

Name	Position	Description	Type
Payload	Body		*JobActions*

LeasesResponse

LeasesResponse

- **Description:** LeasesResponse returned on a successful GET of all the leases

Parameters

Name	Position	Description	Type
Payload	Body		*array of Lease*

MachineParamResponse

MachineParamResponse

- **Description:** MachineParamResponse return on a successful GET of a single Machine param

Parameters

Name	Position	Description	Type
Payload	Body		Raw Octet Stream

ContentResponse

ContentResponse

- **Description:** ContentsResponse returned on a successful GET of a contents

Parameters

Name	Position	Description	Type
Payload	Body		*Content*

ContentSummaryResponse

ContentSummaryResponse

- **Description:** ContentSummaryResponse returned on a successful Post of a content

Parameters

Name	Position	Description	Type
Payload	Body		*ContentSummary*

TemplatesResponse

TemplatesResponse

- **Description:** TemplatesResponse return on a successful GET of all templates

Parameters

Name	Position	Description	Type
Payload	Body		*array of Template*

MachinesResponse

MachinesResponse

- **Description:** MachinesResponse return on a successful GET of all Machines

Parameters

Name	Position	Description	Type
Payload	Body		*array of Machine*

TenantsResponse

TenantsResponse

- **Description:** TenantsResponse returned on a successful GET of all the tenants

Parameters

Name	Position	Description	Type
Payload	Body		*array of Tenant*

ProfileParamResponse

ProfileParamResponse

- **Description:** ProfileParamResponse return on a successful GET of a single Param for a Profile

Parameters

Name	Position	Description	Type
Payload	Body		Raw Octet Stream

UserTokenResponse

UserTokenResponse

- **Description:** UserTokenResponse returned on a successful GET of user token

Parameters

Name	Position	Description	Type
Payload	Body		*UserToken*

InfoResponse

InfoResponse

- **Description:** InfosResponse returned on a successful GET of an info

Parameters

Name	Position	Description	Type
Payload	Body		*Info*

SubnetsResponse

SubnetsResponse

- **Description:** SubnetsResponse returned on a successful GET of all the subnets

Parameters

Name	Position	Description	Type
Payload	Body		*array of Subnet*

FilesResponse

FilesResponse

- **Description:** FilesResponse returned on a successful GET of files

Parameters

Name	Position	Description	Type
Payload	Body		*FilePaths*

ParamsResponse

ParamsResponse

- **Description:** ParamsResponse returned on a successful GET of all the params

Parameters

Name	Position	Description	Type
Payload	Body		*array of Param*

TemplateResponse

TemplateResponse

- **Description:** TemplateResponse return on a successful GET, PUT, PATCH or POST of a single Template

Parameters

Name	Position	Description	Type
Payload	Body		*Template*

BootEnvResponse

BootEnvResponse

- **Description:** BootEnvResponse returned on a successful GET, PUT, PATCH, or POST of a single bootenv

Parameters

Name	Position	Description	Type
Payload	Body		*BootEnv*

MachineParamsResponse

MachineParamsResponse

- **Description:** MachineParamsResponse return on a successful GET of all Machine's Params

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

PluginProvidersResponse

PluginProvidersResponse

- **Description:** PluginProvidersResponse returned on a successful GET of all plugin_provider

Parameters

Name	Position	Description	Type
Payload	Body		*array of PluginProvider*

NoContentResponse

NoContentResponse

- **Description:** NoContentResponse is returned for deletes and auth errors

Parameters

Name	Position	Description	Type
Payload	Body		No Data Returned

JobResponse

JobResponse

- **Description:** JobResponse return on a successful GET, PUT, PATCH or POST of a single Job

Parameters

Name	Position	Description	Type
Payload	Body		*Job*

UserResponse

UserResponse

- **Description:** UserResponse returned on a successful GET, PUT, PATCH, or POST of a single user

Parameters

Name	Position	Description	Type
Payload	Body		*User*

TaskParamsResponse

TaskParamsResponse

- **Description:** TaskParamsResponse return on a successful GET of all Task's Params

Parameters

Name	Position	Description	Type
Payload	Body		map of strings to object

JobLogResponse

JobLogResponse

- **Description:** This is a HACK - I can't figure out how to get swagger to render this a binary. So we lie. We also override this object from the server directory to have a binary format which turns it into a stream.

  JobLogResponse returned on a successful GET of a log

Parameters

Name	Position	Description	Type
Payload	Body		string

WorkflowsResponse

WorkflowsResponse

- **Description:** WorkflowsResponse returned on a successful GET of all the workflows

Parameters

Name	Position	Description	Type
Payload	Body		*array of Workflow*

TenantResponse

TenantResponse

- **Description:** TenantResponse returned on a successful GET, PUT, PATCH, or POST of a single tenant

Parameters

Name	Position	Description	Type
Payload	Body		*Tenant*

InterfaceResponse

InterfaceResponse

- **Description:** InterfacesResponse returned on a successful GET of an interfaces

Parameters

Name	Position	Description	Type
Payload	Body		*Interface*

IsoInfoResponse

IsoInfoResponse

- **Description:** IsoInfoResponse returned on a successful upload of an iso

Parameters

Name	Position	Description	Type
Payload	Body		*BlobInfo*

StageResponse

StageResponse

- **Description:** StageResponse returned on a successful GET, PUT, PATCH, or POST of a single stage

Parameters

Name	Position	Description	Type
Payload	Body		*Stage*

ContentsResponse

ContentsResponse

- **Description:** ContentsResponse returned on a successful GET of all contents

Parameters

Name	Position	Description	Type
Payload	Body		*array of ContentSummary*

PluginProviderResponse

PluginProviderResponse

- **Description:** PluginProvidersResponse returned on a successful GET of an plugin_provider

Parameters

Name	Position	Description	Type
Payload	Body		*PluginProvider*

ActionPostResponse

ActionPostResponse

- **Description:** ActionPostResponse return on a successful POST of action

Parameters

Name	Position	Description	Type
Payload	Body		Raw Octet Stream

LogResponse

LogResponse

- **Description:** LogResponse is returned in response to a log dump request.

Parameters

Name	Position	Description	Type
Payload	Body		*array of Line*

TaskResponse

TaskResponse

- **Description:** TaskResponse return on a successful GET, PUT, PATCH or POST of a single Task

Parameters

Name	Position	Description	Type
Payload	Body		*Task*

SingleIndexResponse

SingleIndexResponse

- **Description:** SingleIndexResponse tests to see if a single specific index exists. Unlike the other index API endpoints, you can probe for dynamic indexes this way.

Parameters

Name	Position	Description	Type
Payload	Body		*Index*

FileResponse

FileResponse

- **Description:** This is a HACK - I can't figure out how to get swagger to render this a binary. So we lie. We also override this object from the server directory to have a binary format which turns it into a stream.

  FileResponse returned on a successful GET of a file

Parameters

Name	Position	Description	Type
Payload	Body		string

Definitions

Index

Index

- **Description:** Index holds details on the index

Fields

Name	Description	Type
Unique	Unique tells you whether there can be mutiple entries in the index for the same key that refer to different items.	boolean
Type	Type gives you a rough idea of how the string used to query this index should be formatted.	string

---

JobAction

JobAction

- **Description:** If path is specified, then the runner will place the contents into that location. If path is not specified, then the runner will attempt to bash exec the contents.

Fields

Name	Description	Type
Content*		string
Path*		string
Meta*		map of strings to string
Name*		string

Subnet

Subnet

- **Description:** Subnet represents a DHCP Subnet

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Subnet*	Subnet is the network address in CIDR form that all leases acquired in its range will use for options, lease times, and NextServer settings by default	string
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this Subnet. This should tell what it is for, any special considerations that should be taken into account when using it, etc.	string
Pickers*	Pickers is list of methods that will allocate IP addresses. Each string must refer to a valid address picking strategy. The current ones are: "none", which will refuse to hand out an address and refuse to try any remaining strategies. "hint", which will try to reuse the address that the DHCP packet is requesting, if it has one. If the request does not have a requested address, "hint" will fall through to the next strategy. Otherwise, it will refuse to try any remaining strategies whether or not it can satisfy the request. This should force the client to fall back to DHCPDISCOVER with no requsted IP address. "hint" will reuse expired leases and unexpired leases that match on the requested address, strategy, and token. "nextFree", which will try to create a Lease with the next free address in the subnet active range. It will fall through to the next strategy if it cannot find a free IP. "nextFree" only considers addresses that do not have a lease, whether or not the lease is expired. "mostExpired" will try to recycle the most expired lease in the subnet's active range. All of the address allocation strategies do not consider any addresses that are reserved, as lease creation will be handled by the reservation instead. We will consider adding more address allocation strategies in the future.	array of string
OnlyReservations*	OnlyReservations indicates that we will only allow leases for which there is a preexisting reservation.	boolean
Validated	Validated tracks whether or not the model has been validated.	boolean
Documentation	Documentation of this subnet. This should tell what the subnet is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
Enabled*	Enabled indicates if the subnet should hand out leases or continue operating leases if already running.	boolean
Options		*array of DhcpOption*
ReservedLeaseTime*	ReservedLeasTime is the default lease time we will hand out to leases created from a reservation in our subnet.	integer
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
ActiveLeaseTime*	ActiveLeaseTime is the default lease duration in seconds we will hand out to leases that do not have a reservation.	integer
Proxy	Proxy indicates if the subnet should act as a proxy DHCP server. If true, the subnet will not manage ip addresses but will send offers to requests. It is an error for Proxy and Unmanaged to be true.	boolean
Unman-	Unmanaged indicates that dr-provision will never send boot-related options to machines that get leases from this subnet. If false, dr-provision will send whatever boot-related options it would nor-	boolean

Section

Section

map of strings to object

Param

Param

- **Description:** Specifically, it contains a description of what the information is for, detailed documentation about the param, and a JSON schema that the param must match to be considered valid.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	Description is a one-line description of the parameter.	string
Documentation	Documentation details what the parameter does, what values it can take, what it is used for, etc.	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Secure*	Secure implies that any API interactions with this Param will deal with SecureData values.	boolean
Schema*	Schema must be a valid JSONSchema as of draft v4.	Raw Octet Stream
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	Name is the name of the param. Params must be uniquely named.	string

Content

Content

- **Description:** Files??

Fields

Name	Description	Type
meta*		*ContentMetaData*
sections		*Sections*

Role

Role

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	Description of role	string
Documentation	Documentation of this role. This should tell what the role is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Claims	Claims that the role support.	*array of Claim*
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	Name is the name of the user	string

BlobInfo

BlobInfo

Fields

Name	Description	Type
Path		string
Size		integer

Template

Template

- **Description:** Template represents a template that will be associated with a boot environment.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this template	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Validated	Validated tracks whether or not the model has been validated.	boolean
ID*	ID is a unique identifier for this template. It cannot change once it is set.	string
Contents*	Contents is the raw template. It must be a valid template according to text/template.	string

JobActions

JobActions

array of JobAction

BootEnv

BootEnv

- **Description:** BootEnv encapsulates the machine-agnostic information needed by the provisioner to set up a boot environment.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
BootParams*	A template that will be expanded to create the full list of boot parameters for the environment.	string
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this boot environment. This should tell what the boot environment is for, any special considerations that should be taken into account when using it, etc.	string
Kernel*	The partial path to the kernel for the boot environment. This should be path that the kernel is located at in the OS ISO or install archive.	string
Initrds*	Partial paths to the initrds that should be loaded for the boot environment. These should be paths that the initrds are located at in the OS ISO or install archive.	array of string
Documentation	Documentation of this boot environment. This should tell what the boot environment is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
Templates*	The templates that should be expanded into files for the boot environment.	*array of TemplateInfo*
OptionalParams	The list of extra optional parameters for this bootstate. They can be present as Machine.Params when the bootenv is applied to the machine. These are more other consumers of the bootenv to know what parameters could additionally be applied to the bootenv by the renderer based upon the Machine.Params	array of string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
RequiredParams*	The list of extra required parameters for this bootstate. They should be present as Machine.Params when the bootenv is applied to the machine.	array of string
OnlyUnknown*	OnlyUnknown indicates whether this bootenv can be used without a machine. Only bootenvs with this flag set to 'true' be used for the unknownBootEnv preference.	boolean
OS		*OsInfo*
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	The name of the boot environment. Boot environments that install an operating system must end in '-install'.	string

Sections

Sections

map of strings to Unknown

Lease

Lease

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Addr*	Addr is the IP address that the lease handed out.	string
SkipBoot	SkipBoot indicates that the DHCP system is allowed to offer boot options for whatever boot protocol the machine wants to use.	boolean
State*	State is the current state of the lease. This field is for informational purposes only.	string
Strategy*	Strategy is the leasing strategy that will be used determine what to use from the DHCP packet to handle lease management.	string
ExpireTime*	ExpireTime is the time at which the lease expires and is no longer valid The DHCP renewal time will be half this, and the DHCP rebind time will be three quarters of this.	string
NextServer	NextServer is the IP address that we should have the machine talk to next. In most cases, this will be our address.	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Via	Via is the IP address used to select which subnet the lease belongs to. It is either an address present on a local interface that dr-provision is listening on, or the GIADDR field of the DHCP request.	string
Token*	Token is the unique token for this lease based on the Strategy this lease used.	string
Duration	Duration is the time in seconds for which a lease can be valid. ExpireTime is calculated from Duration.	integer
Validated	Validated tracks whether or not the model has been validated.	boolean
Options	Options are the DHCP options that the Lease is running with.	*array of DhcpOption*

Profile

Profile

- **Description:** There is one special profile named 'global' that acts as a global set of parameters for the system.

  These can be assigned to a machine's profile list.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this profile. This can contain any reference information for humans you want associated with the profile.	string
Documentation	Documentation of this profile. This should tell what the profile is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Params	Any additional parameters that may be needed to expand templates for BootEnv, as documented by that boot environment's RequiredParams and OptionalParams.	map of strings to object
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	The name of the profile. This must be unique across all profiles.	string

Stat

Stat

Fields

Name	Description	Type
count*		integer
name*		string

OsInfo

OsInfo

- **Description:** OsInfo holds information about the operating system this BootEnv maps to. Most of this information is optional for now.

Fields

Name	Description	Type
Name*	The name of the OS this BootEnv has. It should be formatted as family-version.	string
IsoFile	The name of the ISO that the OS should install from.	string
IsoUrl	The URL that the ISO can be downloaded from, if any.	string
Version	The version of the OS, if any.	string
Family	The family of operating system (linux distro lineage, etc)	string
IsoSha256	The SHA256 of the ISO file. Used to check for corrupt downloads.	string
Codename	The codename of the OS, if any.	string

Workflow

Workflow

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description		string
Documentation		string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Stages		array of string
Validated	Validated tracks whether or not the model has been validated.	boolean
Name		string

PluginProvider

PluginProvider

- **Description:** Plugin Provider describes the available functions that could be instantiated by a plugin.

Fields

Name	Description	Type
OptionalParams		array of string
HasPublish		boolean
Name		string
Documentation	Documentation of this plugin provider. This should tell what the plugin provider is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
Content	Content Bundle Yaml string - can be optional or empty	string
PluginVersion	This is used to indicate what version the plugin is built for	integer
Version		string
AvailableActions		*array of AvailableAction*
RequiredParams		array of string

Plugin

Plugin

- **Description:** This contains the configuration need to start this plugin instance.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this plugin. This can contain any reference information for humans you want associated with the plugin.	string
Documentation	Documentation of this plugin. This should tell what the plugin is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Params	Any additional parameters that may be needed to configure the plugin.	map of strings to object
Provider*	The plugin provider for this plugin	string
PluginErrors	Error unrelated to the object validity, but the execution of the plugin.	array of string
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	The name of the plugin instance. THis must be unique across all plugins.	string

---

DhcpOption

DhcpOption

Fields

Name	Description	Type
Code*	Code is a DHCP Option Code.	integer
Value*	Value is a text/template that will be expanded and then converted into the proper format for the option code	string

---

ContentSummary

ContentSummary

Fields

Name	Description	Type
Counts		map of strings to integer
meta		*ContentMetaData*
Warnings		array of string

Job

Job

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Machine*	The machine the job was created for. This field must be the UUID of the machine.	string
Archived*		boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Uuid*	The UUID of the job. The primary key.	string
Workflow	The workflow that the task was created in.	string
Validated	Validated tracks whether or not the model has been validated.	boolean
State*	The state the job is in. Must be one of "created", "running", "failed", "finished", "incomplete"	string
ExitState	The final disposition of the job. Can be one of "reboot","poweroff","stop", or "complete" Other substates may be added as time goes on	string
Current*	Whether the job is the "current one" for the machine or if it has been superceded.	boolean
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Task	The task the job was created for. This will be the name of the task.	string
NextIndex*	The next task index that should be run when this job finishes. It is used in conjunction with the machine CurrentTask to implement the server side of the machine agent state machine.	integer
CurrentIndex*	The current index is the machine CurrentTask that created this job.	integer
StartTime	The time the job entered running.	No Data Returned
BootEnv	The bootenv that the task was created in.	string
EndTime	The time the job entered failed or finished.	No Data Returned
Stage	The stage that the task was created in.	string
Previous	The UUID of the previous job to run on this machine.	string

User

User

- **Description:** User is an API user of DigitalRebar Provision

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	Description of user	string
Roles	Roles is a list of Roles this User has.	array of string
PasswordHash	PasswordHash is the scrypt-hashed version of the user's Password.	array of integer
Secret	Token secret - this is used when generating user token's to allow for revocation by the grantor or the grantee. Changing this will invalidate all existing tokens that have this user as a user or a grantor.	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	Name is the name of the user	string

Interface

Interface

Fields

Name	Description	Type
Index	Index of the interface	integer
Addresses*	A List of Addresses on the interface (CIDR)	array of string
DnsDomain	Possible DNS for domain for this interface	string
DnsServers	Possible DNS for this interface	array of string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
ActiveAddress	The interface to use for this interface when advertising or claiming access (CIDR)	string
Gateway	Possible gateway for this interface	string
Name*	Name of the interface	string

Reservation

Reservation

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Subnet	Subnet is the name of the Subnet that this Reservation is associated with. This property is read-only.	string
Errors	If there are any errors in the validation process, they will be available here.	array of string
Addr*	Addr is the IP address permanently assigned to the strategy/token combination.	string
Documentation	Documentation of this reservation. This should tell what the reservation is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
Token*	Token is the unique identifier that the strategy for this Reservation should use.	string
Strategy*	Strategy is the leasing strategy that will be used determine what to use from the DHCP packet to handle lease management.	string
NextServer	NextServer is the address the server should contact next. You should only set this if you want to talk to a DHCP or TFTP server other than the one provided by dr-provision.	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Scoped*	Scoped indicates that this reservation is tied to a particular Subnet, as determined by the reservation's Addr.	boolean
Duration	Duration is the time in seconds for which a lease can be valid. ExpireTime is calculated from Duration.	integer
Validated	Validated tracks whether or not the model has been validated.	boolean
Options	Options is the list of DHCP options that apply to this Reservation	*array of DhcpOption*
Description	A description of this Reservation. This should tell what it is for, any special considerations that should be taken into account when using it, etc.	string

---

Line

Line

Fields

Name	Description	Type
Group	Group is an abstract number used to group Lines together	integer
Service	Service is the name of the log.	string
Seq	Seq is the sequence number that the Line was emitted in. Sequence numbers are globally unique.	integer
Level		*Level*
Ignore-Publish	Should the line be published or not as an event.	boolean
File	File is the source file that generated the line	string
Time	Time is when the Line was created.	No Data Returned
Line	Line is the line number of the line that generated the line.	integer
Data	Data is any auxillary data that was captured.	array of Raw Octet Stream
Message	Message is the message that was logged.	string
Principal	Principal is the user or system that caused the log line to be emitted	string

AvailableAction

AvailableAction

- **Description:** Plugins can provide actions for machines Assumes that there are parameters on the call in addition to the machine.

Fields

Name	Description	Type
OptionalParams		array of string
Model		string
RequiredParams		array of string
Command		string
Provider		string

TemplateInfo

TemplateInfo

- **Description:** TemplateInfo holds information on the templates in the boot environment that will be expanded into files.

Fields

Name	Description	Type
Path*	A text/template that specifies how to create the final path the template should be written to.	string
Meta	Metadata for the TemplateInfo. This can be used by the job running system and the bootenvs to handle OS, arch, and firmware differences.	map of strings to string
ID	The ID of the template that should be expanded. Either this or Contents should be set	string
Contents	The contents that should be used when this template needs to be expanded. Either this or ID should be set.	string
Name*	Name of the template	string

Tenant

Tenant

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description		string
Documentation	Documentation of this tenant. This should tell what the tenant is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Members		map of strings to array
Users		array of string
Validated	Validated tracks whether or not the model has been validated.	boolean
Name		string

Stage

Stage

- **Description:** Stage encapsulates a set of tasks and profiles to apply to a Machine in a BootEnv.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Templates*	The templates that should be expanded into files for the stage.	*array of TemplateInfo*
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this stage. This should tell what it is for, any special considerations that should be taken into account when using it, etc.	string
Documentation	Documentation of this stage. This should tell what the stage is for, any special considerations that should be taken into account when using it, etc. in rich structured text (rst).	string
Reboot	Flag to indicate if a node should be PXE booted on this transition into this Stage. The nextbootpxe and reboot machine actions will be called if present and Reboot is true	boolean
Profiles	The list of profiles a machine should use while in this stage. These are used after machine profiles, but before global.	array of string
OptionalParams	The list of extra optional parameters for this stage. They can be present as Machine.Params when the stage is applied to the machine. These are more other consumers of the stage to know what parameters could additionally be applied to the stage by the renderer based upon the Machine.Params	array of string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Tasks	The list of initial machine tasks that the stage should run	array of string
BootEnv*	The BootEnv the machine should be in to run this stage. If the machine is not in this bootenv, the bootenv of the machine will be changed.	string
RequiredParams*	The list of extra required parameters for this stage. They should be present as Machine.Params when the stage is applied to the machine.	array of string
RunnerWait	This flag is deprecated and will always be TRUE.	boolean
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	The name of the stage.	string

Info

Info

Fields

Name	Description	Type
binl_port*		integer
scopes		map of strings to object
stats*		*array of Stat*
features		array of string
License		*LicenseBundle*
tftp_port*		integer
os*		string
address*		*IP*
dhcp_enabled*		boolean
tftp_enabled*		boolean
api_port*		integer
version*		string
file_port*		integer
binl_enabled*		boolean
dhcp_port*		integer
arch*		string
id*		string
prov_enabled*		boolean

Machine

Machine

- **Description:** Machine represents a single bare-metal system that the provisioner should manage the boot environment for.

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Profile		*Profile*
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	A description of this machine. This can contain any reference information for humans you want associated with the machine.	string
Tasks	The tasks this machine has to run.	array of string
Workflow*	Workflow is the workflow that is currently responsible for processing machine tasks.	string
Secret	Secret for machine token revocation. Changing the secret will invalidate all existing tokens for this machine	string
CurrentJob	The UUID of the job that is currently running on the machine.	string
CurrentTask*		integer
Profiles	An array of profiles to apply to this machine in order when looking for a parameter during rendering.	array of string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
Runnable*	Indicates if the machine can run jobs or not. Failed jobs mark the machine not runnable.	boolean
Params	Replaces the Profile.	map of strings to object
HardwareAddrs	HardwareAddrs is a list of MAC addresses we expect that the system might boot from.	array of string
Address	The IPv4 address of the machine that should be used for PXE purposes. Note that this field does not directly tie into DHCP leases or reservations – the provisioner relies solely on this address when determining what to render for a specific machine.	string
Uuid*	The UUID of the machine. This is auto-created at Create time, and cannot change afterwards.	string
BootEnv	The boot environment that the machine should boot into. This must be the name of a boot environment present in the backend. If this field is not present or blank, the global default bootenv will be used instead.	string
Stage	An optional value to indicate tasks and profiles to apply.	string
OS	OS is the operating system that the node is running in	string
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	The name of the machine. THis must be unique across all machines, and by convention it is the FQDN of the machine, although nothing enforces that.	string

---

Task

Task

Fields

Name	Description	Type
Available	Available tracks whether or not the model passed validation.	boolean
Templates*	Templates lists the templates that need to be rendered for the Task.	*array of TemplateInfo*
Errors	If there are any errors in the validation process, they will be available here.	array of string
Description	Description is a one-line description of this Task.	string
Documentation	Documentation should describe in detail what this task should do on a machine.	string
OptionalParams*	OptionalParams are extra optional parameters that a template rendered for the Task may use.	array of string
ReadOnly	ReadOnly tracks if the store for this object is read-only	boolean
RequiredParams*	RequiredParams is the list of parameters that are required to be present on Machine.Params or in a profile attached to the machine.	array of string
Validated	Validated tracks whether or not the model has been validated.	boolean
Name*	Name is the name of this Task. Task names must be globally unique	string

License

License

Fields

Name	Description	Type
StartDate		No Data Returned
Version		string
Name		string
PurchaseDate		No Data Returned
SoftExpireDate		No Data Returned
LongLicense		string
ShortLicense		string
HardExpireDate		No Data Returned
Active		boolean
Data		Raw Octet Stream

Level

Level

integer

IP

IP

- **Description:** Note that in this documentation, referring to an IP address as an IPv4 address or an IPv6 address is a semantic property of the address, not just the length of the byte slice: a 16-byte slice can still be an IPv4 address.

array of integer

UserPassword

UserPassword

Fields

Name	Description	Type
Password		string

ContentMetaData

ContentMetaData

- **Description:** All fields must be strings

Fields

Name	Description	Type
Description		string
Documentation	Optional fields	string
Writable	Informational Fields	boolean
Source		string
RequiredFeatures		string
Version		string
Type		string
Overwritable		boolean
Name*		string

Error

Error

Fields

Name	Description	Type
Type		string
Model		string
Code	code is the HTTP status code that should be used for this Error	integer
Messages	Messages are any additional messages related to this Error	array of string
Key		string

UserToken

UserToken

Fields

Name	Description	Type
Info		*Info*
Token		string

Patch

Patch

array of Operation

IsoPaths

IsoPaths

array of string

Claim

Claim

- **Description:** User is an API user of DigitalRebar Provision

Fields

Name	Description	Type
action		string
scope		string
specific		string

UUID

UUID

- **Description:** A UUID is a 128 bit (16 byte) Universal Unique IDentifier as defined in RFC 4122.

array of integer

LicenseBundle

LicenseBundle

Fields

Name	Description	Type
OwnerEmail		string
Grantor		string
GrantorEmail		string
ContactEmail		string
Owner		string
ContactId		string
Contact		string
Licenses		*array of License*
OwnerId		string
GenerationVersion		string

FilePaths

FilePaths

array of string

PluginProviderUploadInfo

PluginProviderUploadInfo

Fields

Name	Description	Type
path		string
size		integer

Operation

Operation

- **Description:** operation represents a valid JSON Patch operation as defined by RFC 6902

Fields

Name	Description	Type
path	Path is a JSON Pointer as defined in RFC 6901 All Operations must have a Path	string
from	From is a JSON pointer indicating where a value should be copied/moved from. From is only used by copy and move operations.	string
value	Value is the Value to be used for add, replace, and test operations.	Raw Octet Stream
op	Op can be one of: "add" "remove" "replace" "move" "copy" "test" All Operations must have an Op.	string

Event

Event

- **Description:** In general, the event generates for a subject of the form: type.action.key

Fields

Name	Description	Type
Object	Object - the data of the object.	Raw Octet Stream
Key	Key - the id of the object	string
Time	Time of the event.	string
Action	Action - what happened	string
Type	Type - object type	string
Principal	Principal - the user or subsystem that caused the event to be emitted	string

3.21 Developer Environment

This page is intended for people who are building Digital Rebar Provision from sources or contributing to the code base. We maintain inline documentation and test environment and contributors are expected to participate in maintenance of those efforts.

Note: Prerequisites: go version 1.8 or better. These documents expect ability to both install and update Golang.

3.21.1 Developer Quick Start

To get started quickly, all the installation steps are rolled into a script. The script can be run directly from Github by copying the following lines:

```
curl -fsSL https://raw.githubusercontent.com/digitalrebar/provision/master/
↪tools/build.sh | bash
```

The script will use the current **GOPATH** variable for placing the code. If **GOPATH** isn't set, it will be set to *$HOME/go*.

Once the script is complete, it is possible to change the directory to the source area and continue development.

```
export GOPATH=${GOPATH:-$HOME/go}
cd "$GOPATH/src/github.com/digitalrebar/provision"
```

If more details on how to run the result are needed, consult the *Install* section. The **install.sh** script can be used to install from the source directory after a build.

3.21.2 Building The Server

After the code and assets have been obtained once, the process can be repeated from the project root with the following command:

```
tools/build.sh
```

Another reason to use the *tools/build.sh* script is that it will inject version information into the built binaries to make it easier to track what version is deployed. Developer builds and production builds will be identifiable through the *version* command on both the server and the cli.

3.21.3 Serving UI from File System

When working on the Digital Rebar Provision UI, it is possible to skip the generate steps by using the `--dev-ui` flag. Generally, this is started using `--dev-ui ./embedded/assets/ui/public`.

Brunch (`npm install brunch -g`) is required to build new versions of the ui. use `brunch build --production` to use minify javascript.

3.21.4 Running the Tests

Digital Rebar Provision uses the Golang test libraries and the development team works hard to maintain test coverage.

The `tools/test.sh` in the provision root directory is the main way to test the entire code base.

To test individual modules from their subdirectories run: `go test`

3.21.5 How to get Swagger-Ui

DigiatlRebar Provision uses Swagger to generate interactive help for the API. This is in the tree by default. If an update is needed, do the following:

- git clone https://github.com/swagger-api/swagger-ui
- cp -r swagger-ui/dist/* embedded/assets/swagger-ui
- change in embedded/assets/swagger-ui/index.html:

```
@@ -38,7 +38,7 @@
       if (url && url.length > 1) {
         url = decodeURIComponent(url[1]);
       } else {
-        url = "http://petstore.swagger.io/v2/swagger.json";
+        url = "https://127.0.0.1:8092/swagger.json";
       }

       hljs.configure({
```

- Rebuild the world (**tools/build.sh**)

3.21.6 Packaging the Code

Once the code is built, the code can be package for storage in Github or for use by the **install.sh** script.

Running the **tools/package.sh** script will generate a **dr-provision.zip** and **dr-provision.sha256** file. These files can be used with the *Install* process.

3.22 Developing the Command Line Interface (CLI)

Using the **build.sh** process for the server will generate the files needed to build the cli.

It generates the following directories:

- client
- models

The cli uses those client files to access the server. The editable cli code lives in:

- cli

The hope is that the CLI will use a generated client library based upon the generated swagger.json file. This will help ensure that we are building a valid and viable swagger.json file. The build.sh tool generates all the components need for the cli and also builds multiple instances of it.

3.22.1 Building Client

While a single *go build* command will generate the cli, it is safer to use the *build.sh* script to ensure that all the parts are accurately generated.

- tools/build.sh

The results are stored in the bin directory based upon OS and platform. We currently build windows, linux, and darwin for amd64.

3.22.2 Running Client

After building the code, use the tools/install.sh script to get a cli and dr-provision binary in the top-level directory for the platform.

- tools/install.sh –isolated install

Once that has been done a single time, symbolic links are created so that running commands from the top-level directory should work.

- ./drpcli

For more information, see *Install*.

3.23 Developing Documentation

As an open source project, we encourage community feedback and involvement. Docs can be updated by pull requests against the github repository either from a private tree or directly against the tree.

A couple of notes about consistency.

1. Digital Rebar is the name of the parent project and can be abbreviated DR.
2. Digital Rebar Provision or DR Provision or DRP can be used to reference this part of the project.
3. API docs generated from the go files as part of swagger annotations of the godoc comments. Update there, please.
4. CLI docs generated from the cli files as part of cobra structures. The tools generate those. Update there, please.

Otherwise, try and find a good place for what needs to be added. And Thanks!

3.23.1 Documentation Tooling

There are a lot of ways to work with ReStructuredText… Below is only one possible way of setting up a working environment to ensure you are writing clean RST based documentation. This method is designed to be as lightlweight as possible, while still being as accurate as possible with the final rendered doc changes. There are a lot of editors tha will render RST formatted markup, but very few of them render it the same way, or similar enough to the final rendered documentation styles to be correct.

This process uses the following elements:

1. Install Sphinx to correctly render the RST markup text to HTML
2. Edit the doc in any text editor of your preference (eg *vim*, *atom*, *emacs*, etc.)
3. Use a terminal window with a `watch` function to rebuild the HTML document tree
4. Use a web browser with an auto-refresh extension to view the rendered HTML

Install and Setup Sphinx and Swagger

These setup steps were tested and verified on a Mac platform. However, they should be the same for Linux.

1. Pre-requirement is to have `pip` installed:

```
sudo easy_install pip
```

2. Install Sphinx:

```
pip install Sphinx
```

Note: for more detailed information, please review the Sphinx website on how to install it, at: http://www.sphinx-doc.org/en/stable/tutorial.html#install-sphinx

3. Install SwaggerDoc Library

```
sudo -H pip install -r requirements.txt
```

Note:

If you receive an error message on your first HTML tree build (when you run `make html`) similar to:

```
sphinx-build -b html -d _build/doctrees   . _build/html
Running Sphinx v1.6.5
Extension error:
Could not import extension sphinxcontrib.swaggerdoc (exception: No module named␣
↪swaggerdoc)
make: *** [html] Error 1
```

You will need to build and install the *swaggerdoc* components manually. To do so, do the following:

```
git clone https://github.com/galthaus/sphinx-swaggerdoc
cd sphinx-swaggerdoc/
sudo python setup.py  install
cd ..
rm -rf spinx-swaggerdoc/
```

Edit the Docs

- Checkout/clone the Digital Rebar Provision repo from Github
- Modify the doc(s) as appropriate
- Verify the modifications are rendered correctly
- Create a branch
- Submit a pull request for your changes

Rebuild the HTML Rendered Docs

To assist with editing/reviewing your changes, you will want to rebuild the RST format documentation to rendered HTMl. The Sphinx package is used to do this. Generally, you simply change to the *base git cloned Provision* directory, and do:

```
make html
```

If you are making a lot of changes, you will want to use a file watching utility to see watch for file writes, and automatically kick the `make html` process off for you.

On Mac OS X - you can install the `fswatch` package:

```
brew install fswatch
```

An example use would be:

```
# cd to the base Provision git directory
fswatch -o -0 -r doc | xargs -0 -n 1 -I {} make html
```

Now, any time any changes are made to the files in the `doc/` directory, the `make html` process will be automatically kicked off for you. You should run this in a separate terminal window.

View Your Doc Edits Locally

Once you've run `make html` as above, the _RSTr_ format files will be built into _HTML_ generated versions. You can view the effects of your changes by opening your web browser up, and navigating to the _HTML_ generated docs on your local disk. The below path example references my home directory, and github path location. You'll have to modify this to your local User and location where you've stored your github clone.

Point your browser to the on-disk rendered location (which is the `_build/` directory in the base git repo on disk). For example:

```
file:///Users/shane/github/digitalrebar/provision/_build/html/doc/
dev/dev-docs.html
```

Auto-Refresh Browser

The last piece of the puzzle, you will want to set your web browser to auto-refresh a given tab or window. This way, the HTML rendered documentation will be refreshed in the browser. There are several add-ons/extensions that will do this for you. Here at RackN we have used the following extensions:

Chrome *Auto Refresh Plus* extension: https://chrome.google.com/webstore/detail/auto-refresh-plus/hgeljhfekpckiiplhkigfehkdpldcggm

Firefox *Tab Reloader* add-on (works on Chrome, Firefox, and Opera; but limited to 10 second reloads as minimum reload
https://add0n.com/tab-reloader.html

Simply set your browser tab to refresh every 5 or so seconds.

3.24 FAQ / Troubleshooting

The following section is designed to answer frequently asked questions and help troubleshoot Digital Rebar Provision installs.

Want ligher reading? Checkout our rs_fun.

3.24.1 Bind Error

Digital Rebar Provision will fail if it cannot attach to one of the required ports.

- Typical Error Message: "listen udp4 :67: bind: address already in use"
- Additional Information: The conflicted port will be included in the error between colons (e.g.: *:67:*)
- Workaround: If the conflicting service is not required, simply disable that service
- Resolution: Stop the offending service on the system. Typical corrective actions are:
  - 67 - dhcp. Correct with *sudo pkill dnsmasq*

See the port mapping list on start-up for a complete list.

3.24.2 TFTP Error

In the dr-provision logfiles you may occassionally see error messages relating to `TFTP Aborted`. These errors are (typically) benign and expected behavior. The TFTP protocol does not specify a mechanism to obtain the size of a file to transfer for calculating completed transfer; without first requesting the file. Digital Rebar Provision initiates the transfer request an then immediately aborts it. This obtains the file size for the next transfer to validate the file was served correctly.

Simply ignore these errors. If you receive these errors and you believe you should be provisioning correctly, check that you have correctly specified the default/unknown BootEnv, default Stage, and default Workflow are set correctly.

error messages may appear similarly to:

```
May 24 13:48:22 ubuntu dr-provision[7092]: dr-provision2018/05/24 20:48:22.
↪006224 [280:13]static [error]: /home/travis/gopath/src/github.com/
↪digitalrebar/provision/midlayer/tftp.go:82
May 24 13:48:22 ubuntu dr-provision[7092]: [280:13]TFTP: lpxelinux.0:␣
↪transfer error: sending block 0: code=0, error: TFTP Aborted
```

3.24.3 Generate Certificate

Sometimes the cert/key pair in the github tree is corrupt or not sufficient for the environment. The following command can be used to rebuild a local cert/key pair.

```
sudo openssl req -new -x509 -keyout server.key -out server.crt -days 365 -
↪nodes
```

It may be necessary to install the openssl tools.

3.24.4 Add SSH Keys to Authorized Keys

VIDEO TUTORIAL: https://www.youtube.com/watch?v=StQql8Xn08c

To have provisioned operating systems (including discovery/sledgehammer) add SSH keys, you should set the `access-keys` parameter with a hash of the desired keys. This Param should be applied to the Machines you wish to update, either directly via adding the Param to the Machines, or by adding the Param to a Profile that is subsequently added to the Machines. NOTE that the `global` Profile applies to all Machines, and you can add it to `global` should you desire to add the set of keys to ALL Machines being provisioned.

The below example adds *User1* and *User2* SSH keys to the profile *my-profile*. Change appropriately for your enviornment.

```
cat << END_KEYS > my-keys.json
{
  "Params": {
    "access-keys": {
      "user1": "ssh-rsa user_1_key user1@krib",
      "user2": "ssh-rsa user_2_key user2@krib"
    }
  }
}
END_KEYS

drpcli profiles update my-profile -< keys.json
```

3.24.5 Example Docker Volume Usage

Digital Rebar Provision writes content in the Docker Container to the `/provision/drp-data` directory by default. Most DRP Endpoint provisioning systems will want to have persistent data across the container runtimes. For this you need to add a Docker Volume. The below example shows you how to use the locak Docker host as the backing store for the volume. You can also use any of the container based networked storage solutions to back your volume in.

1. Create a volume for the container

```
export VOL="drp-data"

# create a Docker volume
docker volume create $VOL
```

2. Lets verify that the volume is currently empty

```
docker volume inspect $VOL | jq '.[].Mountpoint'
# outputs:
# "/docker/volumes/drp-data/_data"

# show the contents of the current (empty) volume
ls -la $(docker volume inspect $VOL | jq -r '.[].Mountpoint')
# total 0
# drwxr-xr-x. 2 root root  40 Aug 21 00:41 .
# dr-xr-x---. 1 root root 180 Aug 21 00:41 ..
```

3. Lanch DRP, using our newly created volume:

```
# now run DRP with our volume mapped to /provision/drp-data:
docker run --volume $VOL:/provision/drp-data --name drp -itd --net host␣
↪digitalrebar/provision:stable
```

4. Verify that DRP extracted the assets on the host in the mounted volume location:

```
# when DRP starts up, it extracts and builds the default assets
# in the writable backing store (directory structure):
ls $(docker volume inspect drp-data | jq -r '.[].Mountpoint')
# outputs:
# digitalrebar  job-logs  plugins  replace  saas-content  secrets  tftpboot ␣
↪ux
```

3.24.6 Set SSH Root Mode

The Param `access-ssh-root-mode` defines the login policy for the *root* user. The default vaule is `without-password` which means the remote SSH *root* user must access must be performed with SSH keys (see *Add SSH Keys to Authorized Keys*). Possible values are:

value	definition
`without-password`	require SSH public keys for root login, no forced commands
`yes`	allow SSH *root* user login with password
`no`	do not allow SSH *root* user login at all
`forced-commands-only`	only allow forced commands to run via remote login

3.24.7 What are the default passwords?

When using the community BootEnvs for installation, the password is set to a variant of `RocketSkates`. See *Default Template Identity* for complete details.

We *strongly* recommend changing this default or, better, using SSH `without-password` options as per *Set SSH Root Mode* above.

3.24.8 Turn on autocomplete for the CLI

The DRP CLI has built in support to generate autocomplete (tab completion) capabilities for the BASH shell. To enable, you must generate the autocomplete script file, and add it to your system. This can also be added to your global shell `rc` files to enable autocompletion every time you log in. NOTE that most Linux distros do this slightly differently. Select the method that works for your distro.

You must specify a filename as an argument to the DRP CLI autocomplete command. The filename will be created with the autocomplete script. If you are writing to system areas, you need `root` access (eg via *sudo*).

For Debian/Ubuntu and RHEL/CentOS distros:

```
sudo drpcli autocomplete /etc/bash_completion.d/drpcli
```

For Mac OSX (Darwin):

```
sudo drpcli autocomplete /usr/local/etc/bash_completion.d/drpcli
```

Once the autocomplete file has been created, either log out and log back in, or `source` the created file to enable autocomplete i

```
source /etc/bash_completion.d/drpcli
```

Note:

If you receive an error message when using autocomplete similar to:

```
bash: _get_comp_words_by_ref: command not found
```

Then you will need to install the `bash-completion` package (eg. `sudo yum -y install bash-completion` or `sudo apt -y install bash-completion`).

You will also need to log out and then back in to your shell account to correct the bash_completion issue.

3.24.9 Turn Up the Debug

To get additional debug from dr-provision, set debug preferences to increase the logging. See rs_model_prefs.

3.24.10 Missing VBoxNet Network

Virtual Box does not add host only networks until a VM is attempting to use them. If you are using the interfaces API (or UX wizard) to find available networks and `vboxnet0` does not appear then start your VM and recreate the address.

Virtual Box may also fail to allocate an IP to the host network due to incomplete configuration. In this case, `ip addr` will show the network but no IPv4 address has been allocated; consequently, Digital Rebar will not report this as a working interface.

3.24.11 Debug Sledgehammer

If the sledgehammer discovery image should fail to launch Runner jobs successfully, or other issues arise with the start up sequ

```
journalctl -u sledgehammer
```

3.24.12 Convert Isolated Install to Production Mode

There currently is no officually supported *migration* tool to move from an `Isolated` to `Production` install mode. However, any existing customizations, Machines, Leases, Reservations, Contents, etc. can be moved over from the Isolated install directory structure to a Production install directory, and you should be able to retain your Isolated mode environment.

All customized content is stored in subdirectories as follows:

Isolated: in `drp-data/` in the Current Working Directory the installation was performed in Production: in `/var/lib/dr-provision`

The contents and structure of these locations is the same. Follow the below procedure to safely move from Isolated to Production mode.

1. backup your current `drp-data` directory (eg `tar -czvf /root/drp-isolated-backup.tgz drp-data/`)
2. `pkill dr-provision` service
3. perform fresh install on same host, without the `--isolated` flag
4. follow the start up scripts setup - BUT do NOT start the `dr-provision` service at this point
5. copy the `drp-data/*` directories recursively to `/var/lib/dr-provision` (eg: `unalias cp; cp -ra drp-data/* /var/lib/dr-provision/`)
6. make sure your startup scripts are in place for your production mode (eg: `/etc/systemd/system/dr-provision.service`)
7. start the new production version with `systemctl start dr-provision.service`
8. verify everything is running fine
9. delete the `drp-data` directory (suggest retaining the backup copy for later just in case)

Note: WARNING: If you install a new version of the Digital Rebar Provision service, you must verify that there are no Contents differences between the two versions. Should the `dr-provision` service fail to start up; it's entirely likely that there may be some content changes that need to be addressed in the JSON/YAML files prior to the new version being started. See the *Upgrade* notes for any version-to-version specific documentation.

3.24.13 Custom Kickstart and Preseeds

Starting with `drp-community-content` version 1.5.0 and newer, you can now define a custom Kickstart or Preseed (aka *kic*

```
export UUID="f6ca7bb6-d74f-4bc1-8544-f3df500fb15e"
drpcli machines set $UUID param select-kickseed to "my_kickstart.cfg"
```

Of course, you can apply a Param to a Profile, and apply that Profile to a group of Machines if desired.

Note: The Digital Rebar default kickstart and preseeds have Digital Rebar specific interactions that may be necessary to replicate. Please review the default kickstart and preseeds for patterns and examples you may need to re-use. We HIGHLY recommend you start with a *clone* operation of an existing Kickstart/Preseed file; and making appropriate modifications from that as a baseline.

3.24.14 Download RackN Content via Command Line

If you need to download RackN content requiring authentication, you can do this via the command line by adding Auth Token to the download URL. Your Auth Token is your RackN UserID (UUID) found after logging in to the RackN Portal User Management panel.

Here is an example download using our Auth Token, and using the Catalog to locate the correct download URL based on our DI

```
# Set our RACKN_AUTH token to our UUID
export RACKN_AUTH="?username=<rackn_username_uuid>"

# set our DRP OS and ARCH type
export DRP_ARCH="amd64"
export DRP_OS="linux"

# set our catalog location
PACKET_URL="https://qww9e4paf1.execute-api.us-west-2.amazonaws.com/main/catalog/
↪plugins/packet-ipmi${RACKN_AUTH}"

# obtain our parts for the final plugin download
PART=`curl -sfSL $PACKET_URL | jq -r ".$DRP_ARCH.$DRP_OS"`
BASE=`curl -sfSL $PACKET_URL | jq -r '.base'`

# download the plugin - AWS cares about extra slashes ... blech
curl -s ${BASE}${PART}${RACKN_AUTH} -o drp-plugin-packet-ipmi
```

3.24.15 Update Community Content via Command Line

Here's a brief example of how to upgrade the Community Content installed in a DRP Endpoint using the command line. Please note that some RackN specific content requires authentication to download, while community content does not. See *Download RackN Content via Command Line* for additional steps with RackN content.

Perform the following steps to obtain new content.

View our currently installed Content version:

```
$ drpcli contents show drp-community-content | jq .meta.Version
  "v1.4.0-0-ec1a3fa94e41a2d6a83fe8e6c9c0e99c5a039f79"
```

Get our new version (in this example, explicitly set version to `v1.5.0`. However, you may also specify `stable`, or `tip`, and do

```
export VER="v1.5.0"
curl -sfL -o drp-cc.yaml https://github.com/digitalrebar/provision-content/
↪releases/download/${VER}/drp-community-content.yaml
```

It is suggested that you view this file and ensure it contains the content/changes you are expecting.

Now update the content.

Note: Content that is marked *writable* (field `"ReadOnly": false`) may need to be destroyed, and recreated if it's currently in use on other objects. For *read only* content you can safely update the content.

```
$ drpcli contents update drp-community-content -< drp-cc.yaml
  {
    "Counts": {
      "bootenvs": 7,
      "params": 18,
      "profiles": 1,
      "stages": 13,
      "tasks": 7,
      "templates": 15
  <...snip...>
```

Now verify that our installed content matches the new vesion we expected . . .

```
$ drpcli contents show drp-community-content | jq .meta.Version
  "v1.5.0-0-13f1aff688b53d5dfdab9a1a0c1098bd3c6dc76c"
```

3.24.16 Nested Templates (or "Sub-templates")

The Golang templating language does not provide a call-out to include another template. However, at RackN, we've added the a

```
{{template "nested.tmpl" .}}

# or alternatively:

{{$templateName := (printf "part-seed-%s.tmpl" (.Param "part-scheme")) -}}
{{.CallTemplate $templateName .}}
```

The `template` construct is a text string that refers to a given template name which exists already.

The `CallTemplate` construct can be a variable or expression that evaluates to a string.

3.24.17 How Can I manipulate values during Golang Template rendering?

The Digital Rebar Provision integrates most of the Sprig Function Library in the Golang Template rendering operations. That means that you may include their string, math and flow functions into your pipelines.

For example: *{{.Param "noCamelCase/hashiCorp" | snakecase }}* or *{{.Param "cool/tech" | regexMatch "([DRP]*)"}}*

Please consult the Sprig website for a full list of functions.

Note: Digital Rebar Provision blocks functions that could be used to operate on the endpoint outside of DRP template rendering for security reasons.

3.24.18 Change a Machines Name

If you wish to update/change a Machine Name, you can do:

```
export UUID="abcd-efgh-ijkl-mnop-qrst"
drpcli machines update $UUID '{ "Name": "foobar" }'
```

Note: Note that you can NOT use the `drpcli machines set ...` construct as it only sets Param values. The Machines name is a Field, not a Parameter. This will NOT work: `drpcli machines set $UUID param Name to foobar`.

3.24.19 Set *hostname* in a DHCP Reservation

If you create a DHCP Reservation for a system (or convert an active Lease to Reservation), you can also set the Hostname for the Machine. If you are pre-creating Reservations, this will allow you to have a pre-set hostname when the Machine first comes up. Additionally, if you create/destroy your machine objects, but would like a hostname to persist with the Machine Reservation when the machine returns, you can do this.

Note: The UX version (at least as of v1.2.1 and older) does not support setting DHCP options to the Reservation. You will have to perform these actions using either the CLI or API. The CLI method is outlined below.

This procedure assumes you have a Reservation created already, and we are going to update the existing Reservation. You can combine this procedure with creating a new Reservation, but only if you perform the operation via the CLI or API.

```
# show the current Reservation:
drpcli reservations show 192.168.8.100

# create a Hostname specification in the DHCP Options section of the␣
↪reservation:
drpcli reservations update 192.168.8.100 '{ "Options": [ { "Code": 12, "Value
↪": "pxe-client-8-100" } ] }'
```

In the above exmaple, we are assuming our DHCP Reservation is for a Reservation identified by the IP Address `192.168.8.100`, and that we are setting the hostname (DHCP Option 12) to `pxe-client-8-100`.

3.24.20 UEFI Boot Support - Option 67

Starting with v3.7.1 and newer, a DHCP Subnet specification will try to automatically determine the correct values for the `next-server` and *DHCP Option 67* values. In most cases, you shouldn't need to change this. Older versions of DRP may need the `next-boot` and/or the *DHCP Option 67* values set to work correctly. This is especially true of Virtualbox environments prior to v3.7.1. You will need to force the *DHCP Option 67* to `lpxelinux.0`.

The DHCP service in Digital Rebar Provision can support fairly complex boot file service. You can use advanced logic to ensure you send the right PXE boot file to a client, based on Legacy BIOS boot mode, or UEFI boot mode. Note that UEFI boot mode can vary dramatically in implementations, and some (sadly; extensive) testing may be necessary to get it to work for your system. We have several reports of field deployments with various UEFI implementations working with the new v3.7.0 and newer "magic" Option 67 values.

Here is an example of an advanced Option 67 parameter for a DHCP Subnet specification:

```
{{if (eq (index . 77) "iPXE") }}default.ipxe{{else if (eq (index . 93) "0")}}
↪lpxelinux.0{{else}}bootx64.efi{{end}}
```

If you run in to issues with UEFI boot support - please do NOT hesitate to contact us on the Slack Channel as we may have updated info to help you with UEFI boot support.

An example of adding this to your Subnet specification might look something like:

```
# assumes your subnet name is "eth1" - change it to match your Subnet name:
# you may need to delete the existing value if there is one, first, by doing:
# drpcli subnets set eth1 option 67 to null # The setting to null is not needed␣
↪with v3.7.1 and beyond.
drpcli subnets set eth1 option 67 to '{{if (eq (index . 77) "iPXE") }}default.ipxe
↪{{else if (eq (index . 93) "0")}}lpxelinux.0{{else}}bootx64.efi{{end}}'
```

3.24.21 lpxelinux.0 error: no such file or directory

After TFTPing lpxelinux.0, logs (or network packet traces) may show an error similar to:

```
477    0.378296662    10.10.20.76    10.10.31.96    TFTP    159    Error Code,␣
↪Code:
File not found, Message: open /var/lib/dr-provision/tftpboot/pxelinux.cfg/
↪16089a59-9abd-48c2-850a-2ac3bc134935: no such file or directory``
```

This is expected behavior that is standard PXE *waterfall* searching for a valid filename to boot from. For full reference, please see the syslinux reference documentation, at:

http://www.syslinux.org/wiki/index.php?title=PXELINUX#Configuration

The expected behavior is for a client to attempt to download files in the following order:

1. client id (DRP does not use this option, which is what generates the error)
2. mac address (in the form of `01-88-99-aa-bb-cc-dd`)
3. ip address in uppercase Hexadecimal format, stepping through IP, subnet, and classful boundaries
4. fall back to the default defined file

Due to this behavior, filenames will be specified that do not exist, and the error message related to that probe request is a normal message. This is NOT an indicator that provisioning is broken in your environment.

3.24.22 Render a Kickstart or Preseed

Kickstart and Preseed files only created by request and are not stored on a filesystem that is viewable. They are dynamically generated on the fly, and served from the virtual Filesystem space of the Digital Rebar HTTP server (on port 8091 by default). However, it is possible to render a kickstart or preseed to evaluate how it is going to operate, or troubleshoot issues with your config files.

When a machine is in provisioning status, you can view the dynamically generated preseed or kickstart from the TFTP server (or via the HTTP gateway). Provisioning status means the Machine has been plaed in to an installable BootEnv via a Stage. If (for exaxmple) placed in to `centos-7-install` Stage, the `compute.ks` can be rendered for the machine. Or, if placed in to `ubuntu-16.04-install` Stage, the `seed` can be rendered for the machine.

Get the Machine ID, then use the following constructed URL:

```
MID="7f65279a-7e5c-4e69-af40-dd01af4c5667"
DRP="10.10.10.10"
TYPE="seed"   # seed for ubuntu, or compute.ks for centos

http://${DRP}:8091/machines/${MID}/${TYPE}
```

Example URLs:

ubuntu/debian: http://10.10.10.10:8091/machines/7f65279a-7e5c-4e69-af40-dd01af4c5667/seed

centos/redhat: http://10.10.10.10:8091/machines/7f65279a-7e5c-4e69-af40-dd01af4c5667/compute.ks

Note: A simple trick . . . you can create a non-existent Machine, and place that machine in different BootEnvs to render provisioning files for testing purposes. For example, put the non-existent Machine in the `centos-7-install` Stage, then render the `compute.ks` kickstart URL above.

3.24.23 Booting Ubunutu Without External Access

Default Ubuntu ISOs will attempt to check internet repositories, this can cause problems during provisioning if your environment does not have outbound access.

To workaround this problem, you need to supply a DNS and gateway for your subnet. There are several ways to do this:

1. Internal to Digital Rebar: Define Options 3 (Gateway) and 6 (DNS) for your machines' Subnet.
2. External to Digital Rebar: Adding `default_route=true` to the boot parameters and include a DNS server on the local subnet in DHCP.

3.24.24 Kubernetes Dashboard

For rs_krib, the `admin.conf` files is saved into the `krib/cluster-admin-conf` profile parameter and can be downloaded after installation is complete. Using this file `kubectl --kubeconfig=admin.conf` allows autheticated access to the cluster. Please see the KRIB documentation for more details.

For other deployments such as Ansible Kubespray or the Kubeadm deployments of Kubernetes are all maintained by the respective Kubernetes communities. Digital Rebar simply implements a basic version of those configurations. Access to the Kubernetes Dashboard is often changing, and being updated by the community. Please check with the respective communities about how to correctly access the Dashboard.

Some things to note in general:

- Access is restricted; as well it should
- You must configure/enable access to the Dashboard
- Our implmentations usually have a mechanism configured, but this changes over time

Some things that have worked in the past:

- `kubectl proxy` - enabled Proxy access to the Kubernetes Master to get to the Dashboard
- **try stopping the Proxy container, and running `kubectl proxy --address 0.0.0.0 --accept-hosts '.*'`**
    - carefully consider this implication - you are enable access from all hosts !!!
- any other solutions, please let us know. . . we'll add them here

3.24.25 Expand Templates from Failed Job

If you have a task/template that has failed, once it's been run by the Job system, you can collect the rendered template. The rendered template will be in JSON format, so it may be hard to parse.

```
# set Endpoint and User/Pass appropriately for your environment
export RS_ENDPOINT="https://127.0.0.1:8092"
export RS_KEY="rocketskates:r0cketsk8ts"

# get your Job ID from the failed job, and set accordingly:
JOBID="abcdefghijklmnopqrstuvwxyz"
curl -k -u $RS_KEY $RS_ENDPOINT/api/v3/jobs/$JOBID/actions > $JOBID.json

# optional - if you have the remarshal tools installed:
json2yaml $JOBID.json > $JOBID.yaml
```

3.24.26 RBAC - Limit Users to Just Poweron and Poweroff IPMI Controls

The Role Base Access and Controls subsystem allows an operator to construct user account permissions to limit the scope that a user can impact the Digital Rebar Provision system. Below is an example of how to create a *Claim* that assigns the `Role` named `prod-role` that limits t to only allow IPMI `poweron` and ‘‘poweroff‘ actions. These permissions are applied to the _specific_ set of _scope_ *Machines*:

```
drpcli roles update prod-role '"Claims": [{"action": "action:poweron,␣
↪action:poweroff", "scope": "machines", "specific": "*"}]'
```

Now simply assign this Role to the given users you wish to limit their permissions on.

3.24.27 JQ Usage Examples

JQ Raw Mode

Raw JSON output is usefull when passing the results of one `jq` command in to another for scripted interaction. Be sure to speci

```
<some command> | jq -r ...
```

Filter Out gohai-inventory

The `gohai-inventory` module is extremely useful for providing Machine classification information for use by other stages or tasks. However, it is very long and causes a lot of content to be output to the console when listing Machine information. Using a simple `jq` filter, you can delete the `gohai-inventory` content from the output display.

Note that since the Param name is `gohai-inventory`, we have to provide some quoting of the Param name, since the dash (−

```
drpcli machines list | jq 'del(.[].Params."gohai-inventory")'
```

Subsequently, if you are listing an individual Machine, then you can also filter it's `gohai-inventory` output as well, with:

```
drpcli machines show <UUID> | jq 'del(.Params."gohai-inventory")'
```

List BootEnv Names

Get list of bootenvs available in the installed content, by name:

```
drpcli bootenvs list | jq '.[].Name'
```

Reformat Output With Specific Keys

Get list of machines, output "Name:Uuid" pairs from the the JSON output:

```
drpcli machines list | jq -r '.[] | "\(.Name):\(.Uuid)"'
```

Output is printed as follows:

```
machine1:05abe5dc-637a-4952-a1be-5ec85ba00686
machine2:0d8b7684-9d0e-4c3e-9f89-eded02357521
```

You can modify the output separator (colon in this example) to suit your needs.

Extract Specific Key From Output

`jq` can also pull out only specific Keys from the JSON input. Here is an example to get ISO File name for a bootenv:

```
drpcli contents show os-discovery | jq '.sections.bootenvs.discovery.OS.IsoFile'
```

Display Job Logs for Specific Machine

The Job Logs provide a lot of information about the provisioning process of your DRP Endpoint. However, you often only want

```
export UUID=`abcd-efgh-ijkl-mnop-qrps"
drpcli jobs list | jq ".[] | select(.Machine==\"$UUID\")"
```

List Machines with a Given Profile Added to Them

Starting sometime after v3.9.0 the API will allow you to filter Machines that have a given `Profile` applied to them. If you don'

```
# set the PROFILE variable to the name you want to match
export PROFILE=foobar
drpcli machines list | jq -r ".[] | select(.Profiles[] == \"$PROFILE\") | \"\(.
↪Name)\""
```

In this case, we simply list the output of the Machines `Name`. You can change the final `\(.Name)` to any valid JSON key(s) on the Machine Object.

3.25 Architecture Reference

This section is intended to act as a reference to the internals of dr-provision for developers and power users.

3.25.1 Architecture Overview

Architecturally, *dr-provision* is split into several different packages:

backend is responsible for making sure that all the data is valid and gets written to the persistent store whenever things get updated, along with storing any non-persistent runtime data we need to keep track of.

midlayer is where the TFTP, static HTTP, and DHCP services live, along with the plugin management code.

frontend is responsible for providing the REST API.

models define the data models that the other packages use, along with some common functionality that can be shared between the client and server side.

api defines a client-side Go API for interacting with *dr-provision*.

cli provides our default CLI for interacting with *dr-provision*.

plugin implements the core client code that all plugins should use to act as a *dr-provision* plugin.

backend

In Memory Database

All operating data lives in memory all the time. The only time *dr-provision* reads information from persistent storage is when it is starting up, otherwise we treat the persistent store as write-only. The only exception to this design principle is streaming log data from job execution. We may revisit this design principle if memory pressure becomes a real constraint.

The backend provides a DataTracker that is responsible for holding all of the persistible data. DataTracker also implements indexing mechanism that the other layers use to ensure that the other layers can quickly find what they need, along with a locking scheme to ensure that the data stays consistent. The backend also provides a RequestTracker to ensure that logging and locking on a per-request basis is consistent.

Pluggable Storage

The mechanism for storing persistent data should be pluggable, and only rely on basic key/value store semantics in the absence of transactions. The backend relies on an external Go package (https://github.com/digitalrebar/store) to abstract some basic behaviour on top of various key-value type stores. Adding support for new storage types will be a matter of adding them to that package, not to dr-provison itself.

Single Point of Validation

To the extent that it is feasible, all object validation happenS in the backend. Since the backend is responsible for writing data to the persistent store, it is also the best place to implement all data validation.

Static FS with Dynamic Overlay

A large part of what *dr-provision* does boils down to rendering templates and making them available in the right place at the right time. Whenever a machine, bootenv, or stage changes, there are generally templates that have to be rerendered and made available via static HTTP and TFTP to ensure that machines boot to the corrent environment over the network, have the right OS installation templates, load the proper credentials, etc. We already serve static HTTP and TFTP content from a user-configurable location to provide basic files needed to PXE boot a system and provide all the packages and files needed to install an OS.

For dynamic content, however, we don't always want to write files to a filesystem where anyone with a web browser can discover them, and where we have to worry about cleaning up dynamic content whenever something changes. To that end, we provide a static FS implementation that can register for callbacks to be involed whenever someone accesses a file via TFTP or HTTP. That allows us to defer template rendering until the files are actually requested, and it allows us to transparently proxy requests to remote repositories when we don't actually have a file tree present locally.

Dynamic Remote IP to Local IP Caching

dr-provision is designed to work seamlessly on a multi-homed system and to deal with complex local network configurations. To that end, every other subsystem that listens for packets is instrumented to capture the IP where the request originated and the local IP address that the request came in on. Template rendering and DHCP request handling use this information to make sure that we supply the best IP address that a client should use to contact *dr-provision* at for any future communication.

midlayer

The *midlayer* package handles some basic services that DRP provides as well as the content package management system.

TFTP Service

The TFTP service ties together the pin TFTP handling package and the static FS that the backend provides to handle TFTP requests. We only allow clients to get files, uploading them is not allowed. Remote and local IP addresses for each connection are cached

Static HTTP Service

The Static HTTP service implements a simple high-performance HTTP server that serves files using the static FS that the backend provides. Remove and local IP addresses for each connection are cached.

DHCP Service

The DHCP service built in to *dr-provision* is designed to be fully API driven and to provide all the features needed to manage system IP address assignments through the complete provisioning lifecycle. As such, it has a few interesting features that other DHCP servers may not have:

- The ability to have different ways of determining what unique attribute in a DHCP packet to use to allocate an IP address. When you see references to Strategy and Token in the DHCP models, Strategy refers to the unique attribute the DHCP server should use, and Token refers to the value that the Stategy picked.

For now, the only implemented Strategy is MAC, which has the DHCP server use the MAC address of the network adaptor of the network interface as the unique value of the Token.

- The DHCP server is fully API driven. You can add, remove, and modify Reservations and Subnets on the fly, and changes take effect immediately.
- Built-in ProxyDHCP support, on a subnet by subnet basis. *dr-provision* can coexist with other DHCP servers to only provide PXE support for specific address ranges, leaving address management to your preexisting DHCP infrastructure.

Plugin Management

dr-provision can add extended functionality via external plugins. The midlayer implements all of the functionality needed to accept plugin uploads, interrogate them to discover what functionality they implement, import any content built in to the plugin, and hand off requests and events to the plugin for further processing.

Content Package Management

The *Content Package Management* system builds a stack of content layers that are provided to the *backend* to provide objects to the rest of the system. The data stack has the following layers used in this order:

Table 1: Definitions

Heading	Definition
Layer Type	Type of layer in the data stack as reported in the content layer meta data
Overwritable	Can layers above overwrite content packages at this layer.
Can Override	Can a content package at this layer override lower layers.
Writable	Can the system receive written objects
Many	Can multiple content packages be added to this layer
Use	Who provideds and its use

Table 2: Content Package Management

Layer	Overwritable	Can Override	Writable	Many	Use
writable	yes	no	yes	no	Persistent layer
local	yes	yes	no	no	Layer providing content from local filesystem, /etc/dr-provision directory
dynamic	no	yes	no	yes	Layer providing dynamic content packages provided by the API
default	yes	yes	no	no	Layer providing default content that is always present, but replaceable.
plugin	no	yes	no	yes	Layer providing plugin provided content packages.
basic	yes	yes	no	no	Layer providing mandatory DRP model objects.

When an object is looked up, the look up code will start walking down the stack until the object is found and it will be returned. When an object is to be updated or created, the *Writable* aspect of the layer will be checked to see if the object can be updated or created. If the object can be stored in a layer, it will be used. The content layer stack places the wriable store at the top of the stack.

The simplified view of the stack from the API can be boiled down to:

- Create - Created object's key must not exist in the stack.
- Read - Object will be searched from the top down until it is found.
- Update - Updated object must exist only in the writable layer.
- Delete - Deleted Object must exist only in the writable layer.

frontend

The DRP frontend implements a REST + JSON API for others to interact with and manage *dr-provision*. The *dr-provision* API is available via HTTPS, and we will upgrade to HTTP v2 opportunistically.

Threaded Logging

Each individual request to the API is logged using a unique ID, and that ID is threaded through to all the code paths that the request affects. Detailed logging along with an arbitrary token can also be enabled on a per-request basis to aid in debugging and audit purposes.

Basic and JWT Token Authentication

You can authenticate to the *dr-provision* API via basic auth and via time-limited JWT tokens. We also provide means to invalidate tokens globally and on a per-user basis.

Websocket-based Event Delivery

Authenticated users can open a websocket and arrange for a variety of different events to be watched for. This eliminates the need to poll in a loop for a wide variety of different situations.

models

Every valid *dr-provision* object has a Model that is implemented in this package. These models are authoritative, and their JSON serialization in Go is the canonical wire format.

api

The API package implements the reference Go client API for *dr-provision*. You should consult the go docs for the API at https://godoc.org/github.com/digitalrebar/provision/api for in-depth discussion on how to use the client API.

cli

The CLI package implements the reference Go client CLI for *dr-provision*. The main program for *drpcli* includes this set of functions.

plugin

The plugin package implements the Go core functions needed to create a *dr-provision* plugin.

3.25.2 Data Models

Digital Rebar Provision uses several different data models to manage the task of discovering and provisioning machines in a data center.

Common Model Fields

Model Metadata

Virtually every model contains some embedded metadata, available under the **Meta** field. This data, which consists of a map of string -> string pairs, is ignored by the dr-provision server. The most common metadata fields you will see are:

icon The icon that the UX will use to display instances of this model. Users can choose icons from http://fontawesome.io/icons/.

color The color the icon will be displayed as

title The full name that the UX will use.

feature-flags A comma-seperated list of strings that indicate which features are available for a model. Provision uses feature flags to help the various components and content layers to converge on a supported set of avaiable features.

Model Validation

Models also contain common fields that track the validity and availability of individual objects. These fields are:

- Validated: a boolean value that indicates whether a given object is semantically valid or not. Semantically invalid objects will never be saved, and if one is returned via the API the Errors field will be populated with a list of messages indicating what is invalid.
- Available: a boolean value that indicates whether the object is available to be used, not whether it is semantically valid – an object that is invaild can never be available, while an object that is not available can be semantically valid.
- Errors: a list of strings that contain any error messages that occurred in the process of checking whether a given object is valid and available. Error messages are designed to be human readable.

Objects are checked for validity and availability on initial startup of dr-provision (when they are all loaded into memory), and thereafter every time they are updated. You must check each object returned from an API interaction to ensure that it is valid and available before using it.

Other Common Model Fields

Models can contain other common fields that may be present for user edification and API tracking purposes, but that do not affect how dr-provision will use or interpret changes to the objects. These extra fields are:

- ReadOnly: a boolean value that indicates whether the object can be modified via the API. This field is set to True if the object was loaded from a read-only content layer.
- Description: A brief description of what the object is for, how it should be used, etc. Descriptions should be one line long.
- Documentation: A longer description of what the object is for and how it should be used, generally a few lines to a few paragraphs long. For now, only Params and Tasks have a Documentation field, but other models may add them as situations demand.

3.25.3 Authentication Models

These models work together to manage authentication, authorization, and other access control mechanisms in *dr-provision*.

User

Users keep track of who is allowed to talk to drp-provision, and what actions they are allowed to take in the system. User objects contain the following fields:

- **Name**: A unique name for the user. It cannot be changed after the user is created.
- **PasswordHash**: The scrypt hashed version of the user's Password. This field is always empty when accessed via the API. Changing the Password will also rotate the Secret field.
- **Secret**: A random string used to generate and validate access tokens. Changing this field will invalidate any existing tokens, and replace Secret with a new random value.
- **Roles**: A list of Role names that the User has been assigned.

In addition to the roles asigned to the User, all Users also get a claim that allows them to get themself, change their passwords, and get a Token for themselves.

Claim

Claims grant the ability to perform specific actions against specific objects. Claims have the following fields:

- **Scope**: The API top-level path or object type the Claim pertains to. Object IDs referenced in the **Specific** field must be unique in this Scope. This field may be one of the following values:
  - *, which refers to all top-level Scopes.
  - A single top-level object type or API path component.
  - A comma-seperated list of top-level object types or API components.
- **Specific**: The specific object instances referred to by the Scope. This field may have one of the following values:
  - *, which refers to all Objects in Scope.
  - A single unique ID for the Scope. This ID must refer to the field by which the object is natively referred to in the API.
  - A comma-seperated list of unique IDs.
- **Action**: The action being performed. Common actions include "get", "list","update", and "delete", and different object types can have other actions. This field may have one of the following values:
  - *, which refers to all Actions valid in the Scope under consideration,
  - A single Action.
  - A comma-seperated list of Actions

  Two actions have specialized semantics:

  - *action* By itself, *action* gives permission for all valid plugin-provided actions. *action:actionName* gives permission for the specific actionName. If you want to give access to more than one plugin-provided action, you can specify multiple *action:actionName* instances in the comma-seperated list of Actions.

  - *update* By itself, *update* gives permission to update any field (including Params, if the final Object has them). You may also give permission to update a specific field in an Object by specifying an *update:/Field* specifier. The */Field* part after the colon must be a valid RFC6901 JSON Pointer to the field you want to allow to be updated. This will allow updated to that field and any subfields it might have. This level of access control works on the JSON representation of the Object.

Claims are partially ordered by the access they grant, with the superuser claim `` `{Scope:"*" Action:"*" Specific:"*"}` `` granting access to everything and the empty claim `` `{Scope:"" Action:"" Specific:""}` `` granting access to nothing. If you have two claims *a* and *b*, claim *a* is said to contain claim *b* if *a* is capable of satisfying every authentication request *b* is.

Role

Roles are licensed features – to perform any interaction with a Role besides listing them and getting them, you must have a license with the **rbac** feature enabled.

Roles are named lists of Claims that can be assigned to a User. Roles have the following fields:

- **Name**: The unique name of the Role.
- **Claims**: A list of Claims that the Role provides.

Roles are also partially ordered by the access they grant based on their Claims. Role *a* is said to contain role *b* if every claim *b* contains can be satisfied by a claim on *a*

dr-provision provides a default *superuser* role that contains just the superuser claim. By default, the rocketskates user will be assigned this role.

Collectively, Roles and Claims control what a caller can do with the API.

Tenants

Tenants are licensed features – to perform any interaction with a Tenant besides listing them and getting them, you must have a license with the **rbac** feature enabled.

Tenants control what objects a user can see via the dr-provision API. Tenants have the following fields:

- **Name**: The unique name of the Tenant.
- **Users**: The list of Users that are in this Tenant. Users can be in at most one Tenant at a time.
- **Members**: The objects that are in the Tenant. This field is structured as a JSON object whose keys specify the Scope of the objects, and whose values are lists of object indentifiers. Access is only restricted if the Scope of the object is present in the Members field of the tenant – objects whose Scope is not present do not have restricted visibility.

Object visibility restrictions based on a Tenant are processed before Roles are processsed, which means that a Role granting access to an object that is not allowed by the Tenant will be ignored.

By default, Users are not members of a Tenant, and can therefore potentially see everything via the API (subject to Role based restrictions, of course.)

3.25.4 How Authentication Works

User Tokens

User tokens are created by accessing *GET /api/v3/users/:username/token*. By default, a token created using this method can act on behalf of user. This includes rights for the user to get information about themself, change their password, and fetch a token for themselves, and any access granted by additional roles the user has. Users can restrict access that a token generated in this method by passing an optional comma-seperated list of Roles as a parameter during the API request, and any requested Roles that would increase the scope of the allowed Claims will be silently dropped.

Machine Tokens

Certain common machine usage patterns (discovery, running tasks, etc) also need to interact with the API, and hence need a Token that authorizes them to perform those actions. These tokens have a fixed set of permissions:

- Machine Discovery: This token has the ability to create and get Machines, and nothing else. It is needed to allow Sledgehammer to create a machine for itself during initial system discovery.
- Machine Operations: This token gives a Machine the ability to modify itself, get stages and tasks, create events, create a reservation and modify a reservation for the machine's address, and create and manage Jobs for itself.

These machine tokens are generated as part of template expansion via the .GenerateToken command (which generates tokens that expire according to the unknownTokenTimeout and knownTokenTimeout preferences), and the .GenerateInfiniteToken command, which generates a Machine Operations token that expires in 3 years and is intended to grant long-term access for the task runner. These tokens cannot be generated by any other means.

How Tokens Are Checked

1. A request is made to the API. If the request contains *Authorization: Bearer*, that token is used. If the request contains *Authorization: Basic*, the contained username/password is checked and used to create a one-use Token.
2. Claims are created based on the API path requested and the HTTP method. For example, a *GET /api/v3/users* request creates a Claim of `` `{Scope: "users",Action:"list",Specific: ""}` ``, a *GET /api/v3/users/bob* creates a Claim of `` `{Scope: "users", Action: "get" ,Specific: "bob"}` ``, a *PATCH /api/v3/bootenvs/fred* that wants to patch OS.Name and OS.IsoName generates `` `{Scope: "bootenvs", Action: "update:/OS/Name", Specific: "fred"}` `` and `` `{Scope: "bootenvs", Action: "update:/OS/IsoName", Specific: "fred"}` ``, and so on.
3. The token is checked to make sure it is still valid based on the system Secret, the user Secret, and the grantor Secret. If any of these have changed, or the token has expired, the API will return a 403.
4. The list of created Claims is tested to see if it is contained by any one of the Roles contained in the Token, or by any direct Claims contained in the Token. If all of the created Claims are satisfied, the request is considered to be authorized, otherwise the API will return a 403.
5. The API carries out the request and returns an appropriate response.

3.25.5 Provisioning Models

These models work together to manage any and all lifecycle needs for managing Machines in *dr-provision*. This includes:

- Keeping track of what machines are being managed by *dr-provision*.
- Controlling what OS environment any given Machine will boot to over the network.
- Managing the order in which Tasks will be run on Machines.
- Making sure that any files that are needed to complete the provisioning process are available and valid.

Template

Template expansion underlies just about everything that DigitalRebar Provision does. All template expansion in *dr-provision* happens accroding to the rules defined by the golang text/template package, which (along with this document) is required reading if you want to build content for *dr-provision*. Template objects define common template content that different parts of *dr-provision* can reuse as they see fit. Template objects contain the following fields:

- **ID**: A unique identifier for the Template.
- **Contents**: The contents of the Template. It must be parseable as a go text/template.

TemplateInfo

Closely related to the *Template* is the TemplateInfo object, which is included as part of the *BootEnv*, *Stage*, and *Task* objects. TemplateInfo objects have the following fields:

- **Name**: The name of this TemplateInfo.
- **Path**: A string that will be expanded as if it were a *Template* to generate a path for the template.
- **Contents**: If present, a string that will be expanded as if it were a *Template* to generate the file that will be made available at the location indicated by the Path field. Contents must be empty or not present if ID is set.
- **ID**: If present, the ID of the *Template* that will be used to generated the file that will be made available at the location indicated by the Path field. ID must be empty or not present if Contents is set.

Rendering Templates

Whenever *dr-provision* needs to render something as a template (whether or not it is a Template object, a TemplateInfo object, or just a string that might contain a template), it always does so in the context of a RenderData object, which provides a slew of useful helper functions along with references to the applicable objects. RenderData is what *dr-provision* uses for *dot* or *{{ . }}* when executing a template. RenderData has the following fields:

- **Machine**: the Machine that we are rendering templates for. Except for rendering the unknownBootEnv, all template rendering that *dr-provision* does happens against a machine and one of a BootEnv, a Task, or a Stage. If Machine is present, the following helpers are present on RenderData:
  - **.Machine.Path** returns a machine-specific Path fragment (based on the Machine UUID) that can be used to store or refer to machine specific information via the static file server or via TFTP. It is particularly useful for ensuring that templates are expanded into a unique file space for each machine by using it in a TemplateInfo Path field.
  - **.Machine.Address** returns the IP address of the Machine as recorded in the Lease or Reservation.
  - **.Machine.HexAddress** returns the IP address of the Machine in hex format, suitable for use by anything expecting a hex encoded IP address.
  - **.Machine.Url** returns a machine specific http URL that can be used to access machine specific information via http.
  - **.ParamExists <key>** returns true if the specified key is a valid parameter available for this rendering.
  - **.Param <key>** returns the value for the specified key for this rendering. .Param and .ParamExists always look parameters up in the following order:
    1. Params set directly on a Machine.
    2. Params set on the Profiles that have been added to a Machine, in the order of that Machine's Profiles list.

3. Params set on the Profiles added to the Stage that the Machine is currently in, in the order of that Stage's Profile list.
4. The current default Profile.
5. The default value defined as part of the JSON schema for the Param.

Param returns values as simple strings! For complex output, look at .ParamAsJSON and .ParamAsYAML below.

– **.ParamAsJSON <key>** returns the value for the specified key for this rendering preserved in JSON formatting. This is important for templates that rely on `jq` or other commands that need consistent formatting

  Note: .ParamAsJSON will use the .Param lookup order above.

– **.ParamAsYAML <key>** returns the value for the specified key for this rendering preserved in YAML formatting. This is important for configuration files and templates that need consistent formatting

  Note: .ParamAsYAML will use the .Param lookup order above.

– **.Repos <tag>, <tag>,. . .** returns Repos (as defined by the package-repositories param currently in scope) with the matching tags.

– **.MachineRepos** will return a list of OS package repositories that can be used to install packages on the Machine. The repos returned will be for .Machine.OS

– **.InstallRepos** will return at most one OS package repository that can be used to install an OS from, and at most one OS package repository that contains security updates to apply during OS install.

– **[Sprig functions]** are string, math, file and flow functions for golang templates from the Sprig Function Library. They can be added to pipeline evaluation to perform useful template rendering operations.

- **Env**: The BootEnv that we are rendering templates for, if applicable. Unless the BootEnv has the OnlyUnknown flag set, RenderData will also include a Machine. If Env is present, the following helpers will also be present on RenderData:

  – **.Env.PathFor <proto> <partial>** is a helper that makes it easier to build paths that the client side shuld expect. proto should be either **http** or **tftp**, and partial is a partial path relative to the root of a package repository.

  – **.Env.JoinInitrds <proto>** joins together a list of initrds in a way that is applicable for the passed in proto.

  – **.BootParams** returns a rendered version of .Env.BootParams. It will be rendered against the current RenderData.

  – **.Env.OS.FamilyName**: The contents of .Env.OS.Family if present, otherwise the result of splitting .Env.OS.Name by hyphens and taking the first part.

  – **.Env.OS.FamilyVersion**: The contents of .Env.OS.Version if present, otherwise the result of splitting .Env.OS.Name by hyphens and taking the second part.

  – **.Env.OS.FamilyType**: The type of .Env.OS.FamilyName. rhel for distros based on RHEL, debian for distros based on Debian, otherwise the same as .Env.OS.FamilyName. More return types will be added upon request.

  – **.Env.OS.VersionEq <testVersion>**: Splits testVersion and .Env.OS.FamilyVersion into pieces seperated by a period. Returns true if .Env.OS.FamilyVersion has at least as many pieces as testVersion and all the pieces they have in common are numerically equal.

- **Task**: the Task we are rendering templates for, if applicable. RenderData will include a Machine.
- **Stage**: the Stage we are rendering templates for, if applicable. RenderData will include a Machine.

RenderData includes the following helper methods:

- **.ProvisionerAddress** returns an IP address that is on the provisioner that is the most direct access to the machine.
- **.ProvisionerURL** returns an HTTP URL to access the base file server root
- **.ApiURL** returns an HTTPS URL to access the Digital Rebar Provision API
- **.GenerateToken** generates either a **known token** or an **unknown token** for use by the template to update objects in Digital Rebar Provision. The tokens are valid for a limited time as defined by the **knownTokenTimeout** and **unknownTokenTimeout** rs_model_prefs respectively. The tokens are also restricted to the function the can perform. The *known token* is limited to only reading and updating the specific machine the template is being rendered for. If a machine is not present during the render, an *unknown token* is generated that has the ability to query and create machines. These are used by the install process to indicate that the install is finished and that the *local* BootEnv should be used for the next boot and during the discovery process to create the newly discovered machine.
- **.GenerateInfiniteToken** works like **.GenerateToken**, but creates a token with a 3 year timeout.
- **.ParseURL <segment> <url>** parses the specified URL and return the segment requested.
- **template <string> .** includes the template specified by the string. String can NOT be a variable and note that template does NOT have a dot (.) in front.
- **.CallTemplate <string> .** works like **template** but allows for template expansion inside the string to allow for dynamic template references. Note that CallTemplate does have dot (.) in frount.

Param

Params are how *dr-provision* provides validation and a last-ditch default value for data that we use during template expansion. Strictly speaking, you do not have to define a Param in order to use it during template expansion, but *dr-provision* will not be able to enforce that param data is syntactically valid. A Param object has the following fields:

- **Name**: The unique name of the Param. Any time you update a Profile or add, remove, or change a parameter value on another object, *dr-provision* will check to see if a Param exists for the corresponding parameter key.
- **Schema**: A JSON object that contains a valid JSONSchema (draft v4 or higher) that describes what a valid value for the Param looks like. You may also provide a default value for the Param using the *default* stanza in the JSON schema.
- **Secure**: Data managed in this param must be handled in a secure fashion. It will never be passed in cleartext over the API without proper Role based authorization, will be stored in an encrypted wrapper, and will only be made available in an unencrypted form for schema validation on the server, performing plugin actions, and running Tasks on a machine.

Secure Params

Secure param management is a licensed feature. You must have a license with the **secure-params** feature enabled to be able to create and retrieve secure param values. SecureData uses a simple encryption mechanism based on the NACL Box API (as implemented by libsodium, golang.org/x/crypto/nacl/box, tweetnacl-js, PyNaCl, and many others), using curve25519 and xsalsa20 for crypto, and poly1305 for message verification.

Secure params are handled by the API and stored on the backend using a SecureData struct, which has the following fields:

- **Payload**: The encrypted payload. When marshalled to JSON, this should be converted to a base64 encoded string.
- **Nonce**: 24 cryptographically random bytes. When marshalled to JSON, this should be converted into a base64 encoded string.

- **Key**: a 32 byte curve25519 ephemeral public key. When marshalled to JSON, this should be converted to a base64 encoded string.

When a Param has the Secure flag, the following additional steps must be taken to set and get values for this param on objects that hold params.

Setting Secure Param Values

1. Get the peer public key for the object you want to set a secure param on from its *pubkey* endpoint. These endpoints are at */api/v3/<objectType>/<objectID>/pubkey* – as an example, the pubkey endpoint for the global profile is */api/v3/profiles/global/pubkey*. Access to these API endpoints requires an appropriate Claim with the **updateSecure** action. These API endpoints return a JSON string containing the base64 encoding of an array containing 32 bytes.
2. Generate local ephemeral curve25519 public and private keys using a cryptographically secure random number source.
3. Generate a 24 byte nonce using a cryptographically secure random number source.
4. Encrypt the JSON-marshalled param using the nonce, the peer public key, and the ephemeral private key.
5. Generate a SecureData struct with **Key** set to the ephemeral public key, **Nonce** set to the generated nonce, and **Payload** set to the encrypted data.
6. Use the SecureData struct in place of the raw param value when making API calls to add, set, or update params.

Retrieving Decrypted Secure Data Values

In order to retrieve decrypted secure data values, you must have an appropriate Claim with the **getSecure** action. That will allow you to make GET requests to the params API endpoints for param-carrying objects with the *decode=true* query parameter. That will cause the frontend to decrypt any encryped parameter values before returning from the API call.

Task

Tasks in *dr-provision* represent the smallest discrete unit work that the machine agent can use to perform work on a specific machine. The machine agent creates and executes a Job for each Task on the machine. Tasks have the following fields:

- **Name**: The unique name of the task.
- **RequiredParams**: A list of parameters that are required to be present (directly or indirectly) on a Machine to use this Task. It is used to verify that a Machine has all the parameters it needs to be able to execute this Task.
- **OptionalParams**: A list of parameters that the Task may use if present (directly or indirectly) on a Machine.
- **Templates**: A list of TemplateInfos that will be rendered into Job Actions when the machine agent starts exeuting this Task as a Job.

Rendering a Task for a Machine

The Templates for a Task are rendered for a specific Machine whenever the Actions for the Job for that particular task/machine combo are requested.

All referenced templates can refer to each other by their ID (if referring to a Template object directly), or by the TemplateInfo Name (if the TemplateInfo object), in addition to all the Template objects by ID.

Profile

Profiles are named collections of parameters that can be used to provide common sets of parameters across multiple Machines. Profile objects have the following fields:

- **Name**: The unique name of the Profile.
- **Params**: a map of param name -> param value pairs for this Profile.

Stage

Stages are used to define a set of Tasks that must be run in a specific order, potentially in a specific BootEnv. Stages contain the following fields:

- **Name**: The unique name of the Stage.
- **Templates**: A list of TemplateInfos that will be template-expanded for a Machine whenever it transitions to a new Stage.
- **RequiredParams**: A list of parameters that are required to be present (directly or indirectly) on a Machine to use this Stage. It is used to verify that a Machine has all the parameters it needs to be able to boot using this Stage.
- **OptionalParams**: A list of parameters that the Stage may use if present (directly or indirectly) on a Machine.
- **BootEnv**: The boot environment that the Stage must run in. If this field is empty or blank, the assumption is that the Stage will function no matter what environment the machine was booted in. Changing the Stage of a Machine will always change the boot environment of the machine to the one that the stage needs, if any.
- **Profiles**: This is a list of Profile names that will be used for param resolution at template expansion time. These profiles have a higher priority than the default profile,and a lower priority than profiles attached to a Machine directly.
- **Tasks**: This is a list of Task names that will replace the Tasks list on a Machine whenever the Machine switches to using this Stage.
- **Reboot**: DEPRECATED. This flag indicates whether or not the Machine must be rebooted if a Machine switches to this Stage. Generally, if this flag is set the Stage will also have a specific BootEnv defined as well. While this flag is still honored, the runner will automatically reboot the machine as needed to satisfy the BootEnv of the Stage.
- **RunnerWait**: DEPRECATED. This flag used to indicate that the machine agent should wait for more Tasks to be added to the Machine once it finishes runnning the Tasks for this Stage. The runner will currently always wait unless it is explicitly told to exit by an entry in the change-stage/map (also deprecated), or by the exit status of a Task.

Rendering a Stage for a Machine

The Stage for a Machine is rendered *dr-provision* starts up, whenever a Machine changes to a different Stage, or whenever a Stage referred to by a machine changes.

All of the templates referred to by the Templates section of the Stage will be rendered as static files available over the http and tftp services of the provisioner at the paths indicated by each entry in the Templates section. All referenced templates can refer to each other by their ID (if referring to a Template object directly), or by the TemplateInfo Name (if the TemplateInfo object), in addition to all the Template objects by ID.

BootEnv

Boot Environments (or BootEnv for short) are what DigitalRebar Provision uses to model a network boot environment. Each BootEnv contains the following fields:

- **Name**: The name of the boot environment. Each bootenv must have a unique name, and bootenvs that are responsible for booting into an environment that will install an OS on a machine must end in *-install*.
- **OnlyUnknown**: a boolean value indicating that this boot environment is tailored for use by unknown machines. Most boot environments will not have this flag.
- **OS**: an embedded structure that contains some basic information on the OS that this BootEnv will boot into, if applicable. OS contains the following fields:
  - **Name**: the name of the OS this BootEnv will boot into or install. It must be in the format of *distro-version*. centos-7, debian-8, windows-2012r2, ubuntu-16.04 are all examples of what an OS name should look like.
  - **Family**: The family of the OS, if any.
  - **Codename**: The codename of the OS, if any. Generally only really used by Debian, Ubuntu, and realted Linux distributions.
  - **Version**: The version of the OS, if any.
  - **IsoFile**: As an install convienence, DigitalRebar Provision contains built-in ISO expansion functionality that can be used to provide a local mirror for installing operating systems. This field indicates the name of an install archive (usually a .iso file) that should be expanded to provide a local install repo for an operating system.
  - **IsoSha256**: If present, the SHA256sum that IsoFile should have.
  - IsoUrl: The URL that IsoFile can be downloaded from.
- **Kernel**: If present, a partial path to the kernel that should be used to boot a machine over the network. The kernel must be specified as a relative path – no leading / or .. characters are allowed. As an example, the Kernel parameter for the community provided ubuntu-16.04-install boot environment is *install/netboot/ubuntu-installer/amd64/linux*, the path to the kernel relative to the root of the Ubuntu install ISO.
- **Initrds**: If present, a list of partial paths to initrds that should be loaded along with the Kernel when booting a machine over the network. Initrd paths follow the same rules as kernel paths.
- **BootParams**: If present, a string that will undergo template expansion as if it were a *Template*, and passed as arguments to the kernel when it boots.
- **RequiredParams**: A list of parameters that are required to be present (directly or indirectly) on a Machine to use this BootEnv. Only applicable to bootenvs that do not have the OnlyUnknown flag set. It is used to verify that a Machine has all the parameters it needs to be able to boot using this BootEnv.
- **OptionalParams**: A list of parameters that the BootEnv may use if present (directly or indirectly) on a Machine.
- **Templates**: A list of templates that will be expanded and made available via static HTTP and TFTP for this BootEnv. Each entry in this list must have the following fields:

  All bootenvs should include entries in their Templates list for the *pxelinux*, *elilo*, and *ipxe* bootloaders. If the OnlyUnknown flag is set, their Paths should expand to an appropriate location to be loaded as the fallback config file for each bootloader type, otherwise their Paths should expand to an approriate location to be used as a boot file for the loader based on the IP address of the machine. Good examples for each are the discovery and the sledgehammer bootenvs.

Rendering the unknownBootEnv

The BootEnv for the unknownBootEnv preference is rendered whenever *dr-provision* starts up or the BootEnv for the preference is changed. It is the only time that templates are rendered without a Machine being referenced, which is why BootEnvs that can be rendered this way must have the OnlyUnknown flag set.

All of the templates referred to by the Templates section of the BootEnv will be rendered as static files available over the http and tftp services of the provisioner at the paths indicated by each entry in the Templates section. All referenced templates can refer to each other by their ID (if referring to a Template object directly), or by the TemplateInfo Name (if the TemplateInfo object), in addition to all the Template objects by ID.

Rendering a BootEnv for a Machine

The BootEnv for a Machine is rendered whenever *dr-provision* starts up, whenever a Machine changes to a different boot environment, or whenever a boot environment referred to by a machine changes.

All of the templates referred to by the Templates section of the BootEnv will be rendered as static files available over the http and tftp services of the provisioner at the paths indicated by each entry in the Templates section. All referenced templates can refer to each other by their ID (if referring to a Template object directly), or by the TemplateInfo Name (if the TemplateInfo object), in addition to all the Template objects by ID.

Workflow

A Workflow defines a series of Stages that a Machine should go through. It replaces the old change-stage/map mechanism of orchestrating stage changes, which had the following drawbacks:

- change-stage/map is implemented as a map of currentStage -> nextStage:Action pairs. This make it impossible for a machine to go through the same stage twice when going through a workflow.
- It was very easy to get the Action that the runner should perform wrong, leading to unexpected reboots or apparent hangs while walking through the Stages. This has been replaced by making the Runner be smart enough to know that it must reboot on BootEnv changes to a machine, and by having the runner always wait for more tasks unless it is in an OS install BootEnv, or the Runner is directed to exit by a Task exit state.
- The Machine Tasks field only contained tasks for the current Stage, making it hard to see at a glance what Tasks will be executed for the entire workflow.

Workflows have the following fields:

- **Name**: The unique Name of the workflow.
- **Stages**: A list of Stages that any machine with this Workflow must go through.

When the Workflow field on a machine is set, the current task list on the machine is replaced with the results of expanding each Stage in the Workflow using the following items:

- stage:stageName
- bootenv:bootEnvName (if the stage has a non-empty BootEnv field)
- task0…taskN (the content of the Stage Tasks field)

Additionally, the Stage and BootEnv fields of the Machine become read-only, as Stage and BootEnv transitions will occurr as dictated by the machine Task list, and when the Stage changes it does not affect the Task list.

Machine

Machines are what DigitalRebar Provison uses to model a system as it goes through the various stages of the provisioning process. As such, Machine objects have many fields used for different tasks:

- **Name**: A user-chosen name for the machine. It must be unique, although it can be updated at any point via the API. It is a good idea for the Name field to be the same as the FQDN of the Machine in DNS, although nothing enforces that convention.
- **Uuid**: A randomly-chosen v4 UUID that uniquely identifies the machine. It cannot be changed, and it what everything else in dr-provison will use to refer to a machine.
- **Address**: The IPv4 address that third-party systems should expect to be able to use to access the Machine. *dr-provision* does not manage this field – specifically, this does not have to be the same as an existing Lease or Reservation.
- **BootEnv**: The boot environment the Machine should PXE boot to the next time it reboots. When you change the BootEnv field on a machine or change the BootEnv that a Machine wants to use, all relavent templates on the provisioner side are rerendered to reflect the updates. The BootEnv field is read-only if the Workflow field is set.
- **Params**: A map containing parameter names and their associated values. Params set directly on a machine override params from any other source when templates using those params are rendered.
- **Profiles**: An ordered list of profile names that the template render process will use to look up values for Params. At render time, Profiles on a machine are looked at in the order that they appear in this list, and the first one that is found wins (assuming the Param in question is not provided directly on the Machine).
- **OS**: The operating system that the Machine is running. It is only set by *dr-provision* when the Machine is moved into a BootEnv that has -install in the name.
- **Secret**: A random string used when generating auth tokens for this machine. Changing this field will invalidate any existing auth tokens for this machine.
- **Runnable**: A flag that indicates whether the machine agent is allowed to create and execute Jobs against this Machine.
- **Workflow**: The name of the Workflow that the Machine is going through. If the Workflow field is not empty, the Stage and BootEnv fields are read-only.
- **Tasks**: The list of tasks that the Machine should run or that have run. You can add and remove Tasks from this list as long as they have not already run, they are not the current running Task, or they are beyond the next Stage transition present in the Tasks list.
- **CurrentTask**: The index in Tasks of the current running task. A CurrentTask of -1 indicates that none of the Tasks in the current Tasks list have run, and a CurrentTask that is equal to the length of the Tasks list indicates that all of the Tasks have run. The machine agent always creates Jobs based on the CurrentTask. If the Workflow field is non-empty, setting this field to -1 will instead set this field to the most recent Stage in the Tasks list that did not initiate a BootEnv change.
- **Stage**: The current Stage the Machine is in. Changing the Stage of a Machine has the following effects:
  - If the new Stage has a new BootEnv, the Machine Runnable flags will be set to False and the BootEnv on the Machine will change.
  - If the Machine Workflow field is empty, the Machine Tasks list will be replaced by the task list from the new Stage, and CurrentTask will be set back to -1.

  Note that the Stage field is read-only when the Workflow field is non-empty.

Job

Jobs are what *dr-provision* uses to track the state of running individual Tasks on a Machine. There can be at most one current Job for a Machine at any given time. Job objects have the following fields:

- **Uuid**: The randomly generated UUID of the Job.
- **Previous**: The UUID of the Job that ran prior to this one. The Job history of a Machine can be traced by following the Previous UUIDs until you get to the all-zeros UUID.
- **Machine**: The UUID of the Machine that the job was created for.
- **Task**: The name of the Task that the job was created for.
- **Workflow**: The name of the Workflow that the job was created in
- **Stage**: The name of the Stage that the job was created in.
- **BootEnv**: The name of the BootEnv that the job was created in.
- **State**: The state of the Job. State must be one of the following:
  - **created**: this is the state that all freshly-created jobs start at.
  - **running**: Jobs are automatically transitioned to this state by the machine agent when it starts executing this job's Actions.
  - **failed**: Jobs are transitioned to this state when they fail for any reason.
  - **finished**: Jobs are transitioned to this state when all their Actions have completed successfully.
  - **incomplete**: Jobs are transitioned to this state when an Action signals that the job must stop and be restarted later as part of its action.
- **ExitState**: The final disposition of the Job. Can be one of the following:
  - **reboot**: Indicates that the job stopped executing due to the machine needing to be rebooted.
  - **poweroff**: Indicates that the job stopped executing because the machine needs to be powered off.
  - **stop**: Indicates that the job stopped because an action indicated that it should stop executing.
  - **complete**: Indicates that the job finished.
- **StartTime**: The time the job entered the *running* state.
- **EndTime**: The time the Job entered the *finished* or *failed* state.
- **Archived**: Whether it is possible to retrieve the log the Job generated while running.
- **Current**: Whether this job is the most recent for a machine or not.
- **CurrentIndex**: The value of the Machine CurrentTask field when this Job was created.
- **NextIndex**: CurrentIndex++

Job Actions

Once a Job has been created and transitioned to the running state, the machine agent will request that the Templates in the Task for the job be rendered for the Machine and placed into JobActions. JobActions have the following fields:

- **Name**: The name of the JobAction. It is present for informational and troubleshooting purposes, and the name does not effect how the JobAction is handled.
- **Content**: The result of rendering a specific Template from a Task against a Machine.

- **Path**: If present, the Content will be written to the location indicated by this field, replacing any previous file at that location. If Path is not present or empty, then the Contents will be treated as a shell script and be executed.

3.25.6 DHCP Models

These models manage how the DHCP server built into dr-provision. They determines what IP addresses it can hand out to which systems, what values to set for DHCP options, and how to handle DHCP requests at various points in the lifecycle of any given DHCP lease.

Interface

Interface objects are provided by dr-provision as an easy means for the UX and the CLI to enumerate the network interfaces on the dr-provision server and provide some basic information for building local subnets. Interface objects have the following fields:

- Name: the name of the interface that the OS assigned.
- Index: The index of the interface. This is an OS specific index, and does not mena anything to dr-provision directly.
- Addresses: A list of CIDR addresses that are bound to the interface.
- ActiveAddress: The CIDR address that you should use as the Subnet address if you want to create a Subnet specifically for this interface.

DHCP Option

The DHCP Option object holds templated values for DHCP options that should be returned to clients in response to requests. It has the following fields:

- Code: A byte that holds the numeric DHCP option code. See RFC 2132 and friends for what these codes can be.
- Value: A string that will be template-expanded to form a valid value to return as the DHCP option. Template expansion happens in the context of the source options.

Subnet

The Subnet Object defines the configuration of a single subnet for the DHCP server to process. Multiple subnets are allowed. The Subnet can represent a local subnet attached to a local interface (Broadcast Subnet) to the Digital Rebar Provision server or a subnet that is being forwarded or relayed (Relayed Subnet) to the Digital Rebar Provision server. Subnet objects have the following fields:

- Name: The unique name of this Subnet.
- Enabled: A boolean value that indicates whether this subnet is available to hand out new Leases, and whether to allow renewals of any Leases in its range. Setting this to *true* allows the subnet to operate normally, and setting it to *false* will cause dr-provision to refuse to hand out new Leases for addresses in its range and will cause lease renewals for already existing Leases in its address range to fail.
- Strategy: A string that determines how the subnet will uniquely identify part of the DHCP request for address assignment.
- Proxy: A boolean value that indicates that dr-provision should respond to requests for addresses in this address range as if it was a proxy DHCP server (as defined in section 2 of the PXE specification).

- Unmanaged: A boolean value that indicates whether dr-provision will send any DHCP options required for a system to boot over the network. If this bit is set, dr-provision will never send any DHCP options that may be needed to network boot a system.
- Subnet: The network address in CIDR form of this Subnet. Subnets may not have overlapping address ranges.
- ActiveStart: This is the start of the IP address range that this subnet will hand out. It must be within the address range the Subnet is responsible for, and it must be less than ActiveEnd.
- ActiveEnd: This is the end of the IP address range that this subnet will hand out. It must be within the address range the Subnet is responsible for, and it must be greater than ActiveStart.
- ActiveLeaseTime: This is the time (in seconds) that a lease created in this subnet will be valid for.
- ReservedLeaseTime: This is the time (in seconds) that a lease created by a Reservation in this subnet will be valid for. It overrides ActiveLeaseTime.
- OnlyReservations: If set to *true*, then Leases in this subnet range can only be created if there is a reservation created for the requested address.
- Options: A list of DhcpOption objects that should be returned in any replies to dhcp requests.

Reservation

Reservations are what the dr-provision DHCP service uses to ensure that an IP address is always issued to the same device. Reservations have the following fields:

- Strategy: The strategy that the DHCP service should use to determine whether this reservations should be used.
- Token: The string that the Strategy uniquely identifies a network interface with.
- Address: The IP address that is being reserved.
- Options: The DHCP options that should be returned when creating or renewing a Lease based on this Reservation.

Lease

Leases track what IP addresses the system has handed out to what Strategy/Token pairs. Leases contain the following fields:

- Strategy: The strategy that the DHCP service used to allocate the IP address this lease handed out.
- Token: The string that the Strategy uniquely identifies a network interface with.
- Address: The IP address that was handed out.
- State: The state the lease is in. State can be one of the follosing values:
  - PROBE: The IP address is being probed via ICMP Echo Requests to make sure it is not in use by another system.
  - OFFER: The IP address was offered to a system in response to a DHCP discover.
  - ACK: The IP address was offered in response to a DHCP Request.
- ExpireTime: The time at which the Lease expires.

3.25.7 Sledgehammer Overview

Sledgehammer is our system discovery/inventory/configuration in-memory OS image. It is based on the latest CentOS 7.x live CD, with any unneeded content stripped out, tools we usually use preinstalled, and rebacled to me more space and network bandwidth efficient. https://github.com/digitalrebar/sledgehammer is the Git repo that contains the code needed to build Sledgehammer.

Sledgehammer Build

RackN provides pre-built Sledgehammer images that should function properly on the vast majority of x86_64 based servers. However, if you need to include support for custom network hardware, you can rebuild Sledgehammer to have it include the extra drivers that are needed.

This process should mostly consist of modifying sledgehammer.ks to have it pull in any extra drivers or utilities that should be present, and then rebuilding Sledgehammer using build_sledgehammer.sh.

Sledgehammer Design

Sledgehammer is delivered in three parts:

- vmlinuz0: This is the initial kernel that the system PXE boots to when booting to Sledgehammer. This file is usually around 5 - 6 megs in size.
- stage1.img: This is the first-stage initramfs image. It is responsible for loading any required network drivers and then loading the second-stage initramfs over HTTP. This is a standard Linux initramfs, and is usually around 15 megs in size, mainly due to including drivers and firmware for large amounts of network hardware.
- stage2.img: This is the second-stage initramfs. It contains the final running Sledgehammer image as a highly compressed squashfs image. This image is usually between 400 and 500 megs in size.

The two separate stages allows us to minimize the amount of data that gets loaded over TFTP and over the relatively inefficient network stack that the PXE layer in the BIOS and NIC firmware uses.

How Sledgehammer Boots

1. The system PXE boots from a dr-provision server, either into the discovery bootenv, the sledgehammer bootenv, or some other boot environment that boots into Slegehammer. This causes the system to load vmlinuz0 and stage1.img and boot to vmlinuz0.
2. stage1.img loads network drivers for all available network adaptors present in the system, finds the one that the system PXE booted from, uses DHCP to get the IP address to use, then fetches the second-stage image from the location specified by the provisioner.web kernel parameter.
3. stage1.img then mounts the squashfs that stage2.img contains, mounts an in-memory overlayfs on top of that image to allow normal read-write operations, mounts a few other required filesystems, and then pivots over to systemd on the stage2 image to allow for a normal Centos 7 bootup.
4. Once the network has been configured, The Sledgehammer service starts up. It verifies that the system has an IP address on the expected network interface, then fetches the machine-agnostic startup script from the dr-provision static file server, verifies that it is sane, and executes that script.
5. The machine-agnostic startup script (which must be provided by the discovery bootenv and any other bootenv that boots unknown machines into Sledgehammer) examines the options that the kernel was booted with, sets the system hostname, fetches a copy of drpcli from the provisioner, and determines whether it needs to create a machine in dr-provision for this system based on whether a machine UUID was passed as a kernel parameter.

If it was not, a new machine is created, and has its initial Stage and BootEnv set to the default Stage and the default BootEnv. The machine-specific startup script is then downloaded, validated, and executed.

6. The machine-specific startup script (which must be provided the sledgehammer bootenv and any bootenv that boots known machines to Sledgehammer) then starts the machine agent, which starts executing tasks on the machine.

3.25.8 Workflow

dr-provision implements a basic Workflow system to help automate the various tasks needed to provision and decommission systems. The workflow system is the results of several other components in dr-provision interacting. The rest of this section goes over those parts in more detail.

The Bits and Bobs

Tasks

The basic unit of work that dr-provision sequences to drive Machines through a workflow is a *Task*. Individual Tasks are executed against Machines by creating a Job for them.

Tasks contain individual Templates that are expanded for a Machine whenever a Job is created. Each of these individual Templates can expand to either a script to be executed (if the Path parameter is empty or not present), or a file to be placed on the filesystem at the location indicated by template-expanding the Path parameter.(if the Path parameter is not empty).

Jobs

A *Job* is used to track the execution history of Tasks against a specific Machine. A Job is created every time a Task is executed against a machine – Jobs keep track of their state of execution. The history of what has been executed (including all log output from scripts) is stored as a chain of Jobs, and the exit status of the Job determines what a machine agent will do next.

Stages

A *Stage* is used to provide a list of Tasks that should be run on a Machine along with the BootEnv the tasks should be run in.

Machine Agent

The Machine Agent is responsible for creating Jobs, writing out or executing any JobActions, streaming job Logs back to dr-provision for archival purposes, and updating the Job state based on the exit state of any Actions. The Machine Agent is also responsible for rebooting, powering off, and changing to a different Stage as indicated by the Job exit status or the change stage map. Unless directed to exit, the Machine Agent watches the event stream for the Machine it is running on and will execute new tasks as they come to be available.

Change Stage Map (OLD AND BUSTED)

The change-stage/map parameter defines what stage to change to when you finish all the tasks in the current stage. The change stage map is a map whose keys correspond to the stage the machine is currently in and whose values indicate the next stage to transition to and what to have the runner do on the stage transition.

The change-stage/map mechanism has been replaced by the Workflow mechanism, but will be maintained for the forseeable future. You are encouraged to migrate to using Workflows.

Workflows (NEW HOTNESS)

A *Workflow* is used to provide a list of Stages that a Machine should run through to get to a desired end state. When a Workflow is added to a Machine, it renders the BootEnv and Stage for the Machine read-only, replaces the task list on the Machine with one that will step through all the Stages, BootEnvs, and Tasks needed to drive the machine through the Workflow.

How They Work Together

Machine Agent (client side)

The Machine Agent runs on the Client and is responsible for executing tasks and rebooting the Machine as needed. It is structured as a finite state machine for increased reliability and auditability. The Machine Agent always starts in the AGENT_INIT state.

AGENT_INIT Initializes the Agent with a fresh copy of the Machine data, marks the current Job for the machine as *failed* if it is *created* or *running*,and creates an event stream that recieves events for that Machine from dr-provision. If an error was recorded, the Agent prints it to stderr and then clears it out.

If an error occurrs during this, the agent will sleep for a bit and transition back to AGENT_INIT, otherwise it will transition to AGENT_WAIT_FOR_RUNNABLE.

AGENT_WAIT_FOR_RUNNABLE Waits for the Machine to be both Available and Runnable. Once it is, the Agent transitions to AGENT_REBOOT if the machine changed BootEnv, AGENT_EXIT if the Agent recieved a termination signal, AGENT_INIT if there was an error waiting for the state change, and AGENT_RUN_TASK otherwise.

AGENT_RUN_TASK Tries to create a new Job to run on the machine.

If there was an error creating the Job, transitions back to AGENT_INIT.

If there was no job created, the Agent transitions to AGENT_CHANGE_STATE if the Machine does not have a Workflow, and AGENT_WAIT_FOR_CHANGE_STAGE if it does.

If a Job was created, the Agent attempts to execute all the steps in the Task for which the Job was created, and updates the Job depending on the exit status of the steps.

If there was an error executing the Job, the agent will transition back to AGENT_INIT.

If the Job signalled that a reboot is needed, the Agent transitions to AGENT_REBOOT.

If the Job signalled that the system should be powered off, the Agent transitions to AGENT_POWEROFF.

If the Job signalled that the Agent should stop processing Jobs, the Agent transitions to AGENT_EXIT.

Otherwise, the Agent transitions to AGENT_WAIT_FOR_RUNNABLE.

AGENT_WAIT_FOR_STAGE_CHANGE Waits for the Machine to be Available, and for any of the following fields on the Machine to change:

- CurrentTask
- Tasks
- Runnable
- BootEnv
- Stage

Once those conditions are met, follows the same rules as AGENT_WAIT_FOR_RUNNABLE.

AGENT_CHANGE_STAGE Checks the change-stage/map to determine what (and how) to transition to the next Stage when AGENT_RUN_TASK does not get a Job to run from dr-provision.

The Agent first tries to retrieve the change-stage/map Param for the Machine from dr-provision. If it fails due to connection issues, the Agent will transition to AGENT_INIT. If there is no change-stage map, the Agent uses an empty one.

If there is a key in the change-stage/map for the current Stage, the Agent saves the corresponding value as val for further processing.

If there is no next entry for the current Stage in the change-stage/map and the Machine is in a BootEnv that ends in -install, the Agent assumes that val is "local", otherwise the Agent transitions to AGENT_WAIT_FOR_STAGE_CHANGE.

The Agent splits val into nextStage and targetState on the first ':' character in val.

If targetState is empty, it is set according to the following rules:

- If the BootEnv for nextStage is not empty or different from the current BootEnv, targetState is set to "Reboot"
- Otherwise targetState is set to "Success"

The Agent changes the machine Stage to the value indicated by nextStage. If an error occurs during that process, the Agent transitions to AGENT_INIT.

If targetState is "Reboot", the agent transitions to AGENT_REBOOT. if targetState is "Stop", the agent transitions to AGENT_EXIT. If targetState is "Shutdown", the agent transitions to AGENT_POWEROFF. If targetState is anything else, the agent transitions to AGENT_WAIT_FOR_RUNNABLE.

AGENT_EXIT Exits the Agent.

AGENT_REBOOT Reboots the system.

AGENT_POWEROFF Cleanly shuts the system down.

dr-provision (server side)

In dr-provision, the machine Agent relies on these API endpoints to perform its work:

- GET from */api/v3/machines/<machine-uuid>* to get a fresh copy of the Machine during AGENT_INIT.
- PATCH to */api/v3/machines/<machine-uuid>* to update the machine Stage and BootEnv during the AGENT_CHANGE_STAGE.
- GET from */api/v3/machines/<machine-uuid>/params/change-stage/map* to fetch the change-stage/map for the system during AGENT_CHANGE_STAGE.
- POST to */api/v3/jobs* to retrieve the next Job to run during AGENT_RUN_TASK.
- PATCH to */api/v3/jobs/<job-uuid>* to update Job status during AGENT_RUN_TASK and during AGENT_INIT.
- PUT to */api/v3/jobs/<job-uuid>/log* to update the job log during AGENT_RUN_TASK.

- UPGRADE to */api/v3/ws* to create the EventStream websocket that recieves Events for the Machine from dr-provision. Each Event contains a copy of the Machine state at the point in time that the event was created.

Retrieving the next Job

Out of all those endpoints, the one that does the most work is the *POST /api/v3/jobs* endpoint, which is responsible for figuring out what (if any) is the next Job that should be provided to the Machine Agent. It encapsulates the following logic:

1. dr-provision recieves an incoming POST on */api/v3/jobs* that contains a Job with just the Machine filled out.

   If the Machine does not exist, the endpoint returns an Unprocessable Entity HTTP status code.

   If the Machine is not Runnable and Available, the endpoint returns a Conflict status code.

   If the Machine has no more runnable Tasks (as indicated by CurrentTask being greater than or equal to the length of the Machine Tasks list), the endpoint returns a No Content status code, indicating to the Machine Agent that there are no more tasks to run.

2. dr-provision retrieves the CurrentJob for the Machine. If the Machine does not have a CurrentJob, we create a fake one in the Failed state and use that as CurrentJob for the rest of this process.
3. dr-provision tentatively sets *nextTask* to CurrentTask + 1.
4. If the CurrentTask is set to -1 or points to a *stage:* or *bootenv:* entry in the machine Task list, we mark the CurrentTask as *failed* if it is not already *failed* or *created*.
5. If CurrentTask is set to -1, we update it to 0 and set *nextTask* to 0.
6. If CurrentTask points to a *stage:* or a *bootenv:* entry in the Tasks list, and the Machine is not already in the appropriate Stage or BootEnv, we skip the next step. Otherwise we skip past these entries in the Tasks list until we get to an entry that refers to a Task and update CurrentTask and *nextTask* to point to that entry.
7. Depending on the State of the CurrentJob, we take one of the following actions:
   - "incomplete": This indicates that CurrentJob did not fail, but it also did not finish. dr-provision returns CurrentJob unchanged, along with the Accepted status code.
   - "finished": This indicates that the CurrentJob finished without error, and dr-provision should create a new Job for the next Task in the Tasks list. dr-provision sets CurrentTask to *nextTask*.
   - "failed": This indicates that the CurrentJob failed. Since updating a Job to the *failed* state automatically makes the Machine not Runnable, something else has intervened to make the machine Runnable again. dr-provision will create a new Job for the current Task in the Tasks list.
8. dr-provision creates a new Job for the Task in the Tasks list pointed to by CurrentTask. If CurrentTask points to a *stage:* or a *bootenv:* task entry, the new Job is created in the *finished* state, otherwise it is created in the *created* state. The Machine CurrentJob is updated with the UUID of the new Job. The new Job and the Machine are saved.
9. If the new Job is in the *created* state, it is returned along with Created HTTP status code, otherwise nothing is returned along with the NoContent status code.

Changing the Workflow on a Machine

Changing a Workflow on the Machine has the following effects:

- The Stages in the Workflow are expanded to create a new Tasks list. Each Stage gets expanded into a List as follows:

  - *stage:<stageName>*
  - *bootenv:<bootEnvName>* if the Stage specifies a non-empty BootEnv.
  - The Tasks list in the Stage

  The Tasks list on the Machine are replaced with the results of the above expansion.

- The CurrentTask index is set directly to -1.
- The Stage and BootEnv fields become read-only from the API. Instead, they will change in accordance with any *stage:* and *bootenv:* elements in the Task list resulting from expanding the Stages in the Workflow. Any Stage changes that happen during processing a Workflow do not affect the Tasks list or the CurrentTask index.

Removing a Workflow from a Machine

To remove a workflow from a Machine, set the Workflow field to the empty string. The Stage field on the Machine is set to *none*, the Tasks list is emptied, and the CurrentTask index is set back to -1.

Changing the Stage on a Machine

Changing a Stage on a Machine has the folowing effects when done via the API and the Machine does not have a Workflow:

- The Tasks list on the Machine is replaced by the Tasks list on the Stage.
- If the BootEnv field on the Stage is not empty, it replaces the BootEnv on the Machine.
- The CurrentTask index is set to -1
- If the Machine has a different BootEnv now, it is marked as not Runnable.

Resetting the CurrentTask index to -1

If the Machine does not have a Workflow, the CurrentTask index is simply set to -1. Otherwise. it is set to the most recent entry that would not occur in a different BootEnv from the machine's current BootEnv.

3.25.9 Multi-Machine Cluster Pattern

A clear pattern has emerged for Digital Rebar Provision to build application clusters. This pattern allows machines to coordinate activities using atomic storage operations against the Digital Rebar Provision API. Having a safe, atomic shared storage area allows scripts on machines working in parallel to share data and synchronize activities.

The critcal elements of this pattern are using 1) a shared cluster profile on all the machines in the cluster and 2) using JSON PATCH with a reference model to ensure atomic updates.

RackN Task Library content has a set of *Cluster Stages* that implement this behavior in a generic way.

Cluster Profile

The pattern using a shared Profile that has been assigned to all Machines in the cluster. The profile is self-referential: it must contain the name of the profile in a parameter so that machine action will be aware of the shared profile.

For example, if we are using the Profile `example` to create a cluster, then we need to include the Param `cluster-profile:  example` in the Profile. While this may appear redundant, it is essential for the machines to find the profile when they are operating against it. Typically, all cluster scripts start with a "does my cluster profile exist" stanza:

```
{{if .ParamExists "cluster-profile" -}}
CLUSTER_PROFILE={{.Param "cluster-profile"}}
PROFILE_TOKEN={{.GenerateProfileToken (.Param "cluster-profile") 7200}}
{{else -}}
echo "Missing cluster-profile on the machine!"
exit 1
{{end -}}
```

The Digital Rebar API has special behaviors allow machines to modify these templates including an extention for Golang template rendering (see *Rendering Templates*) to include `.GenerateProfileToken`. This special token must be used when updating the shared template.

Atomic Updates with JSON PATCH

The Digital Rebar CLI and UX use JSON PATCH (https://tools.ietf.org/html/rfc6902) instead of PUT extensively. PATCH allows atomic field-level updates by including tests in the update. This means that simulataneous upates do not create "last in" race conditions. Instead, the update will fail in a predictable way that can be used in scripts.

The DRPCLI facilitates use of PATCH for atomic operations by allowing scripts to pass in a reference (aka pre-modified) object. If the `-r` reference object does not match then the update will be rejected.

This allows machines take actions that require synchronization among the cluster such as electing leaders or waiting on operations to finish on other machines. For example, it is typical for clustered machines to poll a param waiting for it to be set by a cluster leader.

The following example shows code that runs on all maachines but only succeeds for the cluster leader. It assumes the Param `my/leader` is set to default to "none".

```
{{template "setup.tmpl" .}}
cl=$(get_param "my/leader")
while [[ $cl = "none" ]]; do
  drpcli -r "$cl" -T "$PROFILE_TOKEN" profiles set "cluster-name" param "my/
↪leader" to "$RS_UUID" 2>/dev/null >/dev/null && break
  # sleep is is a hack but it allows for backoffs
  sleep 1
  # get the cluster info
  cl=$(get_param "my/leader")
done
```

Collecting Data in Profile

As automation is run on the cluster profile, it is common to update Params in the cluster profile. This is a good place to make information available for operators or other machines in the script instead of storing on the specific machine elected leader.

Showing Progress and Leaders

To show cluster install progress, common practice is to set the Machine.Meta icon and color. This provides a fast reference back to operators about the state of the cluster without having to open the profile.

:: drpcli machines update $RS_UUID "{"Meta":{"color":"purple", "icon": "anchor"}}" | jq .Meta

Scripts are often updated so that the elected leader(s) have a distinct icon or color from the rest of the cluster.

3.26 Integrations

Digital Rebar Provision provides many growing integration points.

3.27 Terraform Provider for Digital Rebar Provision

NOTICE: Moved to *Terraform Provider for Digital Rebar Provision*. Please update references.

3.28 Moved: KRIB (Kubernetes Rebar Integrated Bootstrapping)

NOTICE: Moved to *KRIB (Kubernetes Rebar Integrated Bootstrapping)*. Please update references.

3.29 Content Packages

Digital Rebar Provision provides many content packages.

3.29.1 ansible

The following documentation is for ansible content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

params

The content package provides the following params.

ansible/groups

Creates Ansible Host Groups when using the dynamic inventory generator for Ansible.

ansible/groups-members

Inventory of Rebar Machines into Ansible Host Groups when using the dynamic inventory generator for Ansible. Often generated by automation or the UX.

ansible/groupvars

Maps Rebar Profiles to Ansible Host Groups when using the dynamic inventory generator for Ansible.

ansible/hostvars

Sets additional variables in the Ansible dynamic inventory generator for Ansible.

ansible/parent-groups

Creates Ansible Nested Host Groups when using the dynamic inventory generator for Ansible. There is no matching concept in Digital Rebar.

3.29.2 bios

The following documentation is for bios content package at version Unspecified.

Unspecified

3.29.3 burnin

The following documentation is for burnin content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

3.29.4 certs

The following documentation is for certs content package at version Unspecified.

Unspecified

params

The content package provides the following params.

certs/root-pw

This is used to restrict access to the signing ability of the root cert. Stores values for certificates in *certs-data* profile

certs/csr

This certificate signing request with the authkey of the root can be used to generate signed certificate.

NOTE: Stores values for certificates in *certs-data* profile

certs/data

This is the data store for the certs plugin. Stores values for certificates in *certs-data* profile

certs/profile

This allows for defining the internal profile to use for signing the cert. Values are currently: default, client, server, peer.

Stores values for certificates in *certs-data* profile

certs/root

This is used to set or define the root certificate to manipulate or create. Stores values for certificates in *certs-data* profile

plugins

The content package provides the following plugins.

certs

Extends machines with CA management verbs including: makeroot, deleteroot, signcert, getca, list & show.

For production deployments, should be combined with encrypted parameters feature.

Stores values for certificates in *certs-data* profile

3.29.5 classify

The following documentation is for classify content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

params

The content package provides the following params.

classify/disable-classifier

In some use cases, you may need to disable the classifier system from running when machines are being provisioned. This parameter enables a fast disable, without having to remove the classifier itself.

Apply this Param with the value "true" to disable the classifier, on any machines or profiles applied to machines, to effect disabling.

For example, you could apply this Param to "global" profile to disable it globally. NOTE that normal precedence with Paramaters applies, so if a Machine has the Param applied also, the Machines Param value overrides the global value.

classify/classification-data

This object defines a series of tests to determine the classification of a Machine followed by a list of actions to apply to that Machine if the test is successful.

The order of precedence is important! A Machine will be evaluated in the order you specify in the tests below. Generally speaking, you should order the MOST SPECIFIC tests first, followed by general tests next. The very last test in the example below would be a default in the event no previous tests matched.

If none of the tests match, then the Machine will not be modified.

The object should look like this.

- test: "has_mac 00:11:22:33:44:55" actions: - "add_profile profile_name_1" - "change_workflow workflow_name_1"
- test: "in_subnet 192.168.0.0/24" actions: - "set_parameter param_name value" - "add_profile profile_name_2" - "change_workflow workflow_name_2"
- test: "always" actions: - "change_workflow wait-for-input"

stages

The content package provides the following stages.

classify

-| Moves machines into new workflows based upon classification.

tasks

The content package provides the following tasks.

classify

-| This will change the machines workflow and add profiles.

3.29.6 dell-support

The following documentation is for dell-support content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

params

The content package provides the following params.

dell-dsu-base-url

The Dell System Update repositories contain BIOS and firmware updates for all supported Dell servers (in the os_independent repo) as well as OpenManage Server Administrator (in the os_dependent repo).

You can find more information on DSU and the update process we take advantage of at https://linux.dell.com/repo/hardware/dsu/

dell-dsu-block-release

Dell releases updates to its firmware on a block schedule (usually once a month) and the tooling assumes that all firmware in a block should be installed together. You can use this param to pin firmware and tools at a specific release, or you can leave it unspecified to allow tools and updates to be installed from the latest block release.

Block releases are named according to the following format:

YY.MM.NN

YY is the last two digits of the year MM is the month NN is the nth release in a month, starting at 00

3.29.7 dev-library

The following documentation is for dev-library content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

params

The content package provides the following params.

dev/wait-time

Used by the wait-time stage for development use. This can be very helpful to troubleshoot timing issues in provisioning. It should NOT be used in production!

stages

The content package provides the following stages.

wait-time

Uses the dev/wait-time param to determine delay (default=5). This can be very helpful to troubleshoot timing issues in provisioning. It should NOT be used in production!

tasks

The content package provides the following tasks.

workflow-reset

Handy for debugging, this task will clear the current Workflow value form a machine. This allows machines to reset their workflow so THE SAME workflow can be reapplied. Typically, this is a Workflow builder use case only.

always-fails

Handy for debugging, this task will always fail to allow for testing of fault conditions

wait-time

Handy for dev/test, this task will sleep for a programmable amount of time

3.29.8 drp-community-content

The following documentation is for drp-community-content content package at version v1.10.1-tip-53-4f6b983b0f46fd5ae72c030eae353c14397b78b7.

Unspecified

bootenvs

The content package provides the following bootenvs.

ubuntu-16.04-install

NOTE: Default Ubuntu ISOs will attempt to check internet repositories, this can cause problems during provisioning if your environment does not have outbound access. Workaround this by defining Options 3 (Gateway) and 6 (DNS) for your machines' Subnet.

ubuntu-18.04-install

NOTE: Default Ubuntu ISOs will attempt to check internet repositories, this can cause problems during provisioning if your environment does not have outbound access. Workaround this by defining Options 3 (Gateway) and 6 (DNS) for your machines' Subnet.

params

The content package provides the following params.

provisioner-default-password-hash

This specifies the password hash to use for the install process. This is the root password on CentOS-based installs and the default user on the Debian-based installs.

To generate a hash, use the following command:

> echo 'import crypt,getpass; print crypt.crypt(getpass.getpass(), "$6$16_CHARACTER_SALT_HERE")' | python -

Set your 16 character salt in the correct place. This generate sha512 hash which should work on both operating system types.

select-kickseed

The name of a custom kickstart or preseed template to use. If not defined, the default for each platform will be used, as follows

> 'net-seed.tmpl' for Debian/Ubuntu platforms 'centos-7.ks.tmpl' for CentOS 7 platforms

erase-hard-disk-set

This string defines the set of disks to erase. Space separated dev names.

e.g. "/dev/sda /dev/sdb"

local-security-repo

The string value is either a URL for Ubuntu systems or a host/path string for Debian systems. This will override the default security repos if specified.

kexec-ok

Allows the machine agent to call kexec to switch boot environments as long as the machine is currently running Linux, and the new environment has a template named 'kexec' that contains the kernel, initrds, and command line to use.

proxy-servers

This is an array of URLs where each string is an HTTP proxy server to references.

The URLs can be names or IPs with ports and schemas.

provisioner-default-uid

Used in the Debian/Ubuntu installers to specify the uid of the default user.

The value is a string for of the integer value.

provisioner-default-user

Used in the Debian/Ubuntu installers to specify the username of the default user.

rs-debug-enable

Boolean value that enables Bash Script debugging - essentially by turning on 'set -x' globally. Scripts can (and probably do) enable/disable this flags in various sections. In those cases we are not overriding those values.

Additionally, the shell variable 'RS_DEBUG_ENABLE' is set to 1 (on) for Script authors to use. This allows a construct like

> (($RS_DEBUG_ENABLE)) && run_debug_function

package-repositories

This provides a list of repositories to install packages from. It includes dedicated OS installation repositories and more general ones.

An example:

- tag: "centos-7-install" # Every repository needs a unique tag. # A repository can be used by multiple operating systems. # The usual example of this is the EPEL repository, which # can be used by all of the RHEL variants of a given generation. os:

  - "centos-7"

  # If installSource is true, then the URL points directly # to the location we should use for all OS install purposes # save for fetching kernel/initrd pairs from (for now, we will # still assume that they will live on the DRP server). # When installSounrce is true, the os field must contain a single # entry that is an exact match for the bootenv's OS.Name field. installSource: true # For redhat-ish distros when installSource is true, # this URL must contain distro, component, and arch components, # and as such they do not need to be further specified. url: "http://mirrors.kernel.org/centos/7/os/x86_64"

- tag: "centos-7-everything" # Since installSource is not true here, # we can define several package sources at once by # providing a distribution and a components section, # and having the URL point at the top-level directory # where everything is housed. # DRP knows how to expand repo definitions for CentOS and # ScientificLinux provided that they follow the standard # mirror directory layout for each distro. os:

  - centos-7

  url: "http://mirrors.kernel.org/centos" distribution: "7" components:

  - atomic
  - centosplus
  - cloud
  - configmanagement
  - cr
  - dotnet
  - extras
  - fasttrack
  - opstools
  - os
  - paas
  - rt
  - sclo
  - storage
  - updates

- tag: "debian-9-install" os:

  - "debian-9"

  installSource: true # Debian URLs always follow the same rules, no matter # whether the OS install flag is set. As such, # you must always also specify the distribution and # at least the main component, although you can also # specify other components. url: "http://mirrors.kernel.org/debian" distribution: stretch components:

  - main
  - contrib
  - non-free

- tag: "debian-9-backports" os:
  - "debian-9"

  url: "http://mirrors.kernel.org/debian" distribution: stretch-updates components:
  - main
  - contrib
  - non-free
- tag: "debian-9-security" os:
  - "debian-9"

  url: "http://security.debian.org/debian-security/" securitySource: true distribution: stretch/updates components:
  - contrib
  - main
  - non-free

kernel-console

This string defines the console tty string for the kernel boot string.

e.g. console=ttyS1,115200

local-repo

DEPRECATED: Do not use.

Boolean value that tells the install steps to only use the local exploded iso on the DRP server as the only installation repo.

ntp-servers

This is an array of strings where each string an IP address or Name of an NTP server.

provisioner-default-fullname

Used in the Debian/Ubuntu installers to specify the full name of the default user.

zero-hard-disks-for-os-install

By default, the erase disks for os install task tries to only erase any metadata on the disks that may confuse a next OS install, along with (optionally) attempting to discard all sectors on devices that support discard. If this is set to true, the task will also zero all sectors on any non-SSD drives.

custom-ipxe

You can use this whenever you need a custom iPXE boot action, such as booting from a remote URL, booting to an iPXE prompt for troubleshooting, or simply playing around with different ipxe tools. This param defaults to launching an iPXE shell.

gohai-inventory

This provides an untyped dictionary of values from Gohai.

This is fairly raw data. Other parameters are distilled from this.

part-scheme

This string contains the name of a template that holds the Debian installer partitioning commands for use during installation.

The string will be expanded into this template name:

> part-seed-<string>.tmpl

e.g. softraid

extra-packages

This is an array of strings where each string is an additional package to install during the initial OS install.

last-boot-macaddr

Keeps track of the MAC address (in PXELINUX format) that the system most recently PXE booted from.

machine-plugin

The plugin that should manage this machine.

operating-system-disk

Defines the disk the installer should use for OS installation. The usage of this parameter inside a template should add a /dev/ if required. The value should just be the disk simple name.

e.g. sda

access-keys

This map is used to put ssh public keys in place for the root user.

The key of the map is a arbitrary name and the value is the ssh public key for that name.

gohai/skip

Allows machines to stop using the discover-nogohai stage. When true, the gohai part of the discovery stage will be skipped

dns-domain

This is used currently in the Ubuntu/Debian preseed file to specify the DNS Domain Name of the host.

This may be in flux.

access-ssh-root-mode

This string defines the login policy for the root user.

Possible values are: without-password - default yes no forced-commands-only

change-stage/map

This map is used to select the next stage based upon the current stage.

The form is current stage as the key with the value being a string with a colon seperated next stage and return action (success or reboot).

stages

The content package provides the following stages.

discover-no-gohai

DEPRECATED! Use the discover Stage with gohai/skip Param instead.

Pre gohai/skip Parameter, used to run discovery without gohai action.

complete-nowait

This is deprectated and leaves the runner running, but will exit install bootenvs correctly. The use of this was to exit install workflows. This will continue to work for that, but should be replaced by finish-install.

finish-install

Originally, this stage was used with the STOP runner action in the change-stage/map and this will continue to work.

Going forward, the STOP action is not required. The changing of bootenv from something-install to local will cause the runner to exit.

The runner will also continue to run regardless of the RunnerWait flag.

tasks

The content package provides the following tasks.

always-pxe-in-uefi-first

Certian Linux distributions reorder the UEFI boot options to always locally boot from their install first, which is not generally what dr-provision wants, as it makes regaining control of the machine by PXE booting it to Sledgehammer harder. This task rewrites the UEFI boot order to have whatever device we booted from be the first.

gohai

Sets Param: gohai-inventory

Collect inventory from machines using drpcli gohai command and store the result in the gohai-inventory Param on the machine.

If you want to disable this behavior, set the gohai/skip Param to true.

Hint: this can be A LOT of data added to the machine param! You may want to use ?slim in the API to skip returning it on list requests.

3.29.9 drp-community-contrib

The following documentation is for drp-community-contrib content package at version v1.10.1-tip-53-4f6b983b0f46fd5ae72c030eae353c14397b78b7.

Unspecified

3.29.10 drp-prom-mon

The following documentation is for drp-prom-mon content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

tasks

The content package provides the following tasks.

add-node-exporter-param

The 'drp-prom-mon/node-exporter' Param defines the IP Address of the Node Exporter that will be scraped by the Prometheus instance. This value will subsequently be used by the Prometheus configuration steps to add a scrape rule to config files for collecting metrics.

cfg-grafana

This task configures the drp-prom-mon Grafana component. The Prometheus Datasource will be setup, the default admin credentials will be changed, and the grafana shared dashboard (ID 405) will be installed.

install-node-exporter

This task installs the Node Exporter package on a given machine. It also will record the Node Exporter systems IP address in the param 'drp-prom-mon/node-exporter'. This value will subsequently be used by the Prometheus Stage to collect (scrape) metrics from this machine.

The Param 'drp-prom-mon/cluster-profile' MUST BE SET on the machine so that the Stage can record the Pram information correctly.

If the Param 'drp-prom-mon/cluster-profile' is set to the value

> prom-mon-single-node

Then the Stage will assume this is an all-in-one DRP/Prometheus/ Node Exporter/Grafana platform - and you DO NOT have to set the cluster profile.

cfg-drp-prefs

Set the Global Preferences for a DRP instance. If the associated OptionalParams are set, then the default settings will be overridden with the value specified. The defaults for the BootEnv and Stages will assume Discover/Sledgehamer/Discovery.

params

The content package provides the following params.

drp-prom-mon/cluster-profile

This param will be set to the users cloned profile name for the cluster. It is used to identify the Profile to write Param data to for subsequent use during install and configuration stages.

This Params should be used in a cloned Profile which contains the name of the Clone in this Param.

drp-prom-mon/drp-version

This param will set the version of Digital Rebar Provision to install on the Machine.

Set this to any value supported by the DRP installer script. Typically these values would be like the following

> stable = current production 'stable' release tip = current 'tip' release (usually tracks a few
>
> > commits behind 'master'
>
> **v3.7.3 = an explicit version to install - the leading** 'v' is required in the version string

drp-prom-mon/installer-version

This param will set the version of the 'install.sh' script to install this DRP instance with.

The only options supported are

> stable tip

drp-prom-mon/node-exporter

This param will be set to the Machine IP address that the Node Exporter will be running on. Node Exporter will feed metrics to the Prometheus/Grafana instance

Prometheus pulls stats, so no need to configure the Node Exporter to know about it. However, Prometheus DOES need to be configured to know about Node Exporter to scrape stats.

This Param is used in the Prometheus install task.

3.29.11 filebeat

The following documentation is for filebeat content package at version Unspecified.

3.29.12 Filebeat Forwarder

This plugin is used to send events to Filebeat so that it can forward the event to logstash or elasticsearch.

Installation / Configuration Filebeat

Install filebeat:

```
curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-6.
↪3.2-x86_64.rpm
rpm -vi filebeat-6.3.2-x86_64.rpm
```

Configure filebeat:

Inside the /etc/filebeat/filebeat.yml file. You will need to add a filebeat.inputs stanza.

```
- type: tcp
  enabled: true
  max_message_size: 10MiB
  host: "localhost:9008"
  fields_under_root: true
```

You will also need to configure your output section. See the *logstash* or *elasticsearch* as needed.

Configuration DRP Filebeat Plugin

A plugin will need to be created for the filebeat.

The *filebeat/url* parameter can be used to specify the IP:Port pair that filebeat is listening on. This defaults to 127.0.0.1:9008.

params

The content package provides the following params.

filebeat/path

Filebeat log file path. The plugin writes the events to this log file. This file should be under log rotate.

3.29.13 flash

The following documentation is for flash content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

3.29.14 honeycomb

The following documentation is for honeycomb content package at version Unspecified.

Unspecified

3.29.15 image-builder

The following documentation is for image-builder content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

tasks

The content package provides the following tasks.

image-update-packages

Should be run after fixing repo packages, but before installing cloud-init.

3.29.16 image-deploy

The following documentation is for image-deploy content package at version Unspecified.

Unspecified

params

The content package provides the following params.

curtin/partitions

The curtin partition as an object. It would look like this:

storage: version: 1 config:

- id: disk0 type: disk ptable: msdos path: /dev/sda name: main_disk wipe: superblock
- id: disk0-part1 type: partition number: 1 device: disk0 size: 139G flag: boot
- id: disk0-part1-format-root type: format fstype: ntfs quiet: True volume: disk0-part1
- id: disk0-part1-mount-root type: mount path: / device: disk0-part1-format-root

This example is for a wim-like root fs for windows, but other custom partitions could be used.

If unspecified, the default action to setup a simple partition table suitable for linux rootfs installs.

image-deploy/additional-tarballs

A list of additional tarballs to install into the image.

image-deploy/image-file

The file image for image-deploy to install. This will have the Provisioner's URL prepended to the filename.

You may use the drpcli isos upload command to place an image into the system. If this method is used, please prepend 'isos/' to your filename. You should not have any other directories in the filename.

You may use the drpcli files upload command to place an image into the system. If this method is used, prepend 'files/' to your filename. You may use additional directories in the filename.

You may also just place the image file under the DRP tftpboot directory. If this method is used, use the path relative to the tftpboot directory.

cloud-init/user-data

This is the data to return to the node on cloud init calls.

image-deploy/image-os

The file image contains the install contents for an operating system. This variable defines the image OS at a high level.

Valid values are:

- linux
- windows

image-deploy/image-url

The url for the image for image-deploy to install.

image-deploy/install-disk

The disk curtin should install upon

image-deploy/windows-license-key

The license key for the windows operating system as a string.

cloud-init/x509-certs

This contains all the x509 certs for the system as a single string

image-deploy/windows-unattend-template

The template to use for unattend.xml file.

image-deploy/image-type

The file image may be in many formats. This tells image-deploy what method to use to install.

Validated methods are:

- tgz = root file system in compressed tar file.
- dd-tgz = Compressed tar file with a single raw disk image.

image-deploy/windows-unattend-path

The path to the unattend.xml file.

3.29.17 ipmi

The following documentation is for ipmi content package at version Unspecified.

Unspecified

params

The content package provides the following params.

ipmi/configure/ip-mode

This parameter defines which method the BMC should use to get its IP address. Valid values are: dhcp or static

ipmi/configure/reserve-ip

Set to true to enable the DRP to create a Reservation for the BMC's MAC address and IP.

ipmi/enabled

This parameter indicates if the ipmi-configure task enabled IPMI.

ipmi/password

This parameter is used by the IPMI Plugin to access the BMC

ipmi/configure/address

This is required for static ip-mode.

ipmi/configure/no-bmc-fatal

Set to true to cause a failure if the BMC doesn't exist.

ipmi/configure/user

Set this value to true to have the system setup a user on the BMC

ipmi/configure/username

Username to create on the BMC

ipmi/configure/gateway

This is recommended for static ip-mode.

ipmi/configure/port

For Dell servers, this optional parameter will set the iDrac port mode. Many values are supported. Validated values are shared or dedicated.

ipmi/configure/userid

The user id (numeric) to use for the configured user

ipmi/configure/password

Password to create on the BMC for the configured user

ipmi/configure/netmask

This is required for static ip-mode.

ipmi/configure/network

Set to true to enable the IPMI configuring of network address.

ipmi/identify-duration

Duration in seconds to leave the identify light on

ipmi/username

This parameter is used by the IPMI Plugin to access the BMC

ipmi/address

This parameter is used by the IPMI Plugin to access the BMC

tasks

The content package provides the following tasks.

ipmi-configure

This task uses the ipmi.configure parameters to configure the system BMC.

The administrator may choose to configure a user, the network pieces, or both.

Defaults: impi/configure/address = unset impi/configure/gateway = unset impi/configure/ip-mode = dhcp impi/configure/netmask = unset impi/configure/network = true impi/configure/no-bmc-fatal = false impi/configure/reserve-ip = true impi/configure/port = unset impi/configure/user = true impi/configure/username = root impi/configure/password = cr0wBar! impi/configure/userid = unset

3.29.18 krib

The following documentation is for krib content package at version v1.10.1-tip-53-4f6b983b0f46fd5ae72c030eae353c14397b78b7.

3.29.19 KRIB (Kubernetes Rebar Integrated Bootstrapping)

License: KRIB is APLv2

This document provides information on how to use the Digital Rebar *KRIB* content add-on. Use of this content will enable the operator to install Kubernetes in either a Live Boot (immutable infrastructure pattern) mode, or via installed to local hard disk OS mode.

KRIB uses the kubeadm cluster deployment methodology coupled with Digital Rebar enhancements to help proctor the cluster Master election and secrets management. With this content pack, you can install Kubernetes in a zero-touch manner.

KRIB does also support production, highly available (HA) deployments, with multiple masters. To enable this configuration, we've chosen to manage the TLS certificates and etcd installation in the Workflow instead of using the kubeadm process.

This document assumes you have your Digital Rebar Provisioning endpoint fully configured, tested, and working. We assume that you are able to properly provision Machines in your environment as a base level requirement for use of the KRIB content add-on use. See _rs_krib for step by step instructions.

Note: documentation source: https://github.com/digitalrebar/provision-content/blob/master/krib/._Documentation.meta

KRIB Video References

The following videos have been produced or presented by RackN related to the Digital Rebar KRIB solution.

- KRIB Zero Config Kubernetes Cluster channel on YouTube.
- KubeCon: Zero Configuration Pattern on Bare Metal on YouTube - RackN presentation at 2017 KubeCon/Cloud NativeCon in Austin TX

Online Requirements

KRIB uses community *kubeadm* for installation. That process relies on internet connectivity to download containers and other components.

Immutable -vs- Local Install Mode

The two primary deployment patterns that the Digital Rebar KRIB content pack supports are:

1. Live Boot (immutable infrastructure pattern - references[1][2])
2. Local Install (standard install-to-disk pattern)

The *Live Boot* mode uses an in-memory Linux image based on the Digital Rebar Sledgehammer (CentOS based) image. After each reboot of the Machine, the node is reloaded with the in-memory live boot image. This enforces the concept of *immutable infrastructure* - every time a node is booted, deployed, or needs updating, simply reload the latest Live Boot image with appropriate fixes, patches, enhancements, etc.

The Local Install mode mimics the traditional "install-to-my-disk" method that most people are familiar with.

KRIB Basics

KRIB is a Content Pack addition to Digital Rebar Provision. It uses the *Multi-Machine Cluster Pattern* which provides atomic guarantees. This allows for Kubernetes master(s) to be dynamically elected, forcing all other nodes to wait until the kubeadm on the elected master to generate an installation token for the rest of the nodes. Once the Kubernetes master is bootstrapped, the Digital Rebar system facilitates the security token hand-off to rest of the cluster so they can join without any operator intervention.

Elected -vs- Specified Master

By default, the KRIB process will dynamically elect a Master for the Kubernetes cluster. This masters simply win the *race-to-master* election process and the rest of the cluster will coalesce around the elected master.

If you wish to specify a specific machines to be the designated masters, you can do so by setting a *Param* in the cluster *Profile* to the specific *Machine* that will be come the master. To do so, set the `krib/cluster-masters` *Param* to a JSON structure with the Name, UUID and IP of the machines to become masters. You may add this *Param* to the *Profile* in the below specifications, as follows:

```
# JSON reference to add to the Profile Params section
"krib/cluster-masters": [{"Name":"<NAME>", "Uuid":"<UUID>", "Address": "
↪<ADDRESS>"}]

# or drpcli command line option
drpcli profiles set my-k8s-cluster param krib/cluster-master to <JSON>
```

[1] Immutable Infrastructure Reference: Making Server Deployment 10x Faster – the ROI on Immutable Infrastructure
[2] Immutable Infrastructure Reference: Go CI/CD and Immutable Infrastructure for Edge Computing Management

The Kubernetes Master will be built on this Machine specified by the <*UUID*> value.

Note: This *MUST* be in the cluster profile because all machines in the cluster must be able to see this parameter.

Install KRIB

KRIB is a Content Pack and is installed in the standard method as any other Contents. We need the `krib.json` content pack to fully support KRIB and install the helper utility contents for stage changes.

Please review _rs_krib for step by step instructions.

CLI Install

KRIB uses the Certs plugin to build TLS, you can install that from the RackN library

```
# download cert plugin provider (installs the plugin autmatically)
curl -o certs https://s3-us-west-2.amazonaws.com/rebar-catalog/certs/v2.4.0-
↪0-02301d35f9f664d6c81d904c92a9c81d3fd41d2c/amd64/linux/certs
# install plugin provider
drpcli plugin_providers upload certs from certs
# verify it worked - should return true
drpcli plugins show certs | jq .Available
```

Using the Command Line (*drpcli*) utility configured to your endpoint, use this process:

```
    # Get code
    git clone https://github.com/digitalrebar/provision-content
    cd krib

# KRIB content install
drpcli contents bundle krib.yaml
drpcli contents upload krib.yaml
```

If KRIB is already installed, you should replace the upload with *drpcli contents update krib krib.yaml*

UX Install

In the UX, follow this process:

1. Open your DRP Endpoint: (eg. https://127.0.0.1:8092/)
2. Authenticate to your Endpoint
3. Login with your `` `RackN Portal Login` `` account (upper right)
4. Go to the left panel “Content Packages” menu
5. Select `Kubernetes (KRIB: Kubernetes Rebar Immutable Bootstrapping)` from the right side panel (you may need to select *Browser for more Content* or use the *Catalog* button)
6. Select the *Transfer* button for both content packs to add the content to your local Digital Rebar endpoint

Configuring KRIB

The basic outline for configuring KRIB follows the below steps:

1. create a *Profile* to hold the *Params* for the KRIB configuration (you can also clone the `krib-example` profile)
2. add a *Param* of name `krib/cluster-profile` to the *Profile* you created
3. add a *Param* of name `etcd/cluster-profile` to the *Profile* you created
4. apply the Profile to the Machines you are going to add to the KRIB cluster
5. change the Workflow on the Machines to `krib-live-cluster` for memory booting or `krib-install-cluster` to install to Centos. You may clone these reference workflows to build custom actions.
6. installation will start as soon as the Workflow has been set.

There are many configuration options available, review the `krib/*` and `etcd/*` parameters to learn more.

Configure with Terraform

Please review the `intergrations/krib` for example Terraform plans.

Configure with the CLI

The configuration of the Cluster includes several reference *Workflow* that can be used for installation. Depending on which Workflow you use, will determine if the cluster is built via install-to-local-disk or via an immutable pattern (live boot in-memory boot process). Outside of the Workflow differences, all remaining configuration elements are the same.

You must writeable create a *Profile* from YAML (or JSON if you prefer) with the Params stagemap and param required information. Modify the *Name* or other fields as appropriate - be sure you rename all subsequent fields appropriately.

```
echo '
---
Name: "my-k8s-cluster"
Description: "Kubernetes install-to-local-disk"
Params:
  krib/cluster-profile: "my-k8s-cluster"
  etcd/cluster-profile: "my-k8s-cluster"
Meta:
  color: "purple"
  icon: "ship"
  title: "My Installed Kubernetes Cluster"
' > /tmp/krib-config.yaml

drpcli profiles create - < /tmp/krib-config.yaml
```

Note: The following commands should be applied to all of the Machines you wish to enroll in your KRIB cluster. Each Machine needs to be referenced by the Digital Rebar Machine UUID. This example shows how to collect the UUIDs, then you will need to assign them to the `UUIDS` variable. We re-use this variable throughout the below documentation within the shell function named *my_machines*. We also show the correct `drpcli` command that should be run for you by the helper function, for your reference.

Create our helper shell function *my_machines*

```
function my_machines() { for U in $UUIDS; do set -x; drpcli machines $1 $U $2;␣
↪set +x; done; }
```

List your Machines to determine which to apply the Profile to

```
drpcli machines list | jq -r '.[] | "\(.Name) : \(.Uuid)"'
```

IF YOU WANT to make ALL Machines in your endpoint use KRIB, do:

```
export UUIDS=`drpcli machines list | jq -r '.[].Uuid'`
```

Otherwise - individually add them to the *UUIDS* variable, like:

```
export UUIDS="UUID_1 UUID_2 ... UUID_n"
```

Add the Profile to your machines that will be enrolled in the cluster

```
my_machines addprofile my-k8s-cluster

# runs example command:
# drpcli machines addprofile <UUID> my-k8s-cluster
```

Change stage on the Machines to initiate the Workflow transition. YOU MUST select the correct stage, dependent on your install type (Immutable/Live Boot mode or install-to-local-disk mode). For Live Boot mode, select the stage `ssh-access` and for the install-to-local-disk mode select the stage `centos-7-install`.

```
# for Live Boot/Immutable Kubernetes mode
my_machines workflow krib-live-cluster

# for intall-to-local-disk mode:
my_machines workflow krib-install-cluster

# runs example command:
# drpcli machines workflow <UUID> krib-live-cluster
# or
# drpcli machines workflow <UUID> krib-install-cluster
```

Configure with the UX

The below example outlines the process for the UX.

RackN assumes the use of CentOS 7 BootEnv during this process. However, it should theoretically work on most of the BootEnvs. We have not tested it, and your mileage will absolutely vary...

1. create a *Profile* for the Kubernetes Cluster (e.g. `my-k8s-cluster`) or clone the `krib-example` profile.

2. add a *Param* to that *Profile*: `krib/cluster-profile` = `my-k8s-cluster` 2. add a *Param* to that *Profile*: `etcd/cluster-profile` = `my-k8s-cluster` 3. Add the *Profile* (eg `my-k8s-cluster`) to all the machines you want in the cluster. 4. Change workflow on all the machines to `krib-install-cluster` for install-to-local-disk, or to `krib-live-cluster` for the Live Boot/Immutable Kubernetes mode

Then wait for them to complete. You can watch the Stage transitions via the Bulk Actions panel (which requires RackN Portal authentication to view).

Note: The reason the *Immutable Kubernetes/Live Boot* mode does not need a reboot is because they are already running *Sledgehammer* and will start installing upon the stage change.

Operating KRIB

Who is my Master?

If you have not specified who the Kubernetes Master should be; and the master was chosen by election - you will need to determ

```
# returns the Kubernetes cluster Machine UUID
drpcli profiles show my-k8s-cluster | jq -r '.Params."krib/cluster-masters"'
```

Use `kubectl` - on Master

You can log in to the Master node as identified above, and execute `kubectl` commands as follows:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
kubectl get nodes
```

Use `kubectl` - from anywhere

Once the Kubernetes cluster build has been completed, you may use the `kubectl` command to both verify and manage the clus

```
# get the Admin configuration and tokens
drpcli profiles get my-k8s-cluster param krib/cluster-admin-conf > admin.conf

# set our KUBECONFIG variable and get nodes information
export KUBECONFIG=`pwd`/admin.conf
kubectl get nodes
```

Advanced Stages - Helm and Sonobuoy

KRIB includes stages for advanced Kubernetes operating support.

The reference workflows already install Helm using the *krib-helm* stage. To leverage this utility simply define the required JSON syntax for your charts as shown in krib_helm.

Sonobuoy can be used to validate that the cluster conforms to community specification. Adding the *krib-sonobuoy* stage will start a test run. It can be rerun to collect the results or configured to wait for them. Storing test results in the *files* path requires setting the *unsafe/password* parameter and is undesirable for production clusters.

Ingress/Egress Traffic, Dashboard Access, Istio

The Kubernetes dashboard is enabled within a default KRIB built cluster. However no Ingress traffic rules are set up. As such, you must access services from external connections by making changes to Kubernetes, or via the *Kubernetes Dashboard via Proxy*.

These are all issues relating to managing, operating, and running a Kubernetes cluster, and not restrictions that are imposed by Digital Rebar Provision. Please see the appropriate Kubernetes documentation on questions regarding operating, running, and administering Kubernetes (https://kubernetes.io/docs/home/).

For Istio via Helm, please consult krib_helm for a reference install

Kubernetes Dashboard via Proxy

Once you have obtained the `admin.conf` configuration file and security tokens, you may use `kubectl` in Proxy mode to the M

```
kubectl proxy
```

Now, in a local web browser (on the same machine you executed the Proxy command) open the following URL:

https://127.0.0.1:8001/ui

Multiple Clusters

It is absolutely possible to build multiple Kubernetes KRIB clusters with this process. The only difference is each cluster should have a unique name and profile assigned to it. A given Machine may only participate in a single Kubernetes cluster type at any one time. You can install and operate both Live Boot/Immutable with install-to-disk cluster types in the same DRP Endpoint.

Footnotes

params

The content package provides the following params.

etcd/servers

Param is set (output) by the etcd cluster building process

krib/cluster-profile

Part of the Digital Rebar Cluster pattern, this parameter is used to identify the machines used in the Kubernetes cluster This parameter is REQUIRED for KRIB and etcd cluster contruction

krib/operate-options

This Param can be used to pass additional flag options to the 'kubectl' operation that is specified by the 'krib/operate-action' Param. By default, the 'drain' operation will be called if no action is defined on the Machine.

This Param provides some customization to how the operate operation functions.

For 'kubectl drain' documentation, see the following URL: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#drain

For 'kubectl uncordin' doc, see the URL: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#uncordon

NOTE NOTE the following flags are set as default options in the Template for drain operations:

> –ignore-daemonsets –delete-local-data

For 'drain' operations, if you override these defaults, you MOST LIKELY need to specify them for the drain operation to be successful. You have been warned.

No defaults provdided for 'uncordon' operations (you shouldn't need any).

krib/cluster-api-vip-port

The VIP API port to use for multi-master clusters. Each master will bind to 'krib/cluster-api-port' (6443 by default), but HA services for the API will be services by this port number.

Defaults to '8443'.

krib/cluster-admin-conf

Param is set (output) by the cluster building process

etcd/server-count

Allows operators to set the number of machines required for the etcd cluster. Machines will be automatically added until the number is met. NOTE: should be an odd number

etcd/servers-done

Param is set (output) by the etcd cluster building process

krib/cluster-domain

Defines the cluster domain for kublets to operate in by default. Default is 'cluster.local'.

krib/label-env

Used for node specification, labels should be set.

sonobuoy/wait-mins

Default is -1 so that stages do not wait for complete.

Typical runs may take 60 minutes.

If <0 then code does wait and assumes you will run it again to retrieve the results. Task is idempotent so you can re-start a run after you have started to check on results.

etcd/client-port

Allows operators to set the port used by etcd clients

krib/operate-action

This Param can be used to:

> 'drain', 'delete', 'cordon', or 'uncordon'

a node in a KRIB built Kubernetes cluster. If this parameter is not defined on the Machine, the default action will be to 'drain' the node.

Each action can be passed custom arguments via use of the 'krib/operate-options' Param.

provider/calico-config

Set this string to an HTTP or HTTPS reference for a YAML configuration to use for the Calico network provider. If Calico is not installed, this Param will have no effect on the cluster.

provider/flannel-config

Set this string to an HTTP or HTTPS reference for a YAML configuration to use for the Flannel network provider. If Flannel is not installed, this Param will have no effect on the cluster.

krib/cluster-service-dns-domain

Allows operators to specify the Service DNS Domain that will be used by CoreDNS during the 'kubeadm init' process of the cluster creation.

By default we do not override the setting from kubeadm default behavior.

krib/cluster-cri-socket

This Param defines which Socket to use for the Container Runtime Interface. By default KRIB content uses Docker as the CRI, however our goal is to support multiple container CRI formats.

krib/kubeadm-cfg

Set this string to an HTTP or HTTPS reference for a YAML configuration to use for the 'kubeadm.cfg' used during the 'kubeadm init' process.

The default behavior is to use the Parameterized 'kubeadm.cfg' from the template named 'krib-kubeadm.cfg.tmpl'. This config file is used in the 'krib-config.sh.tmpl' which is the main template script that drives the 'kubeadm' cluster init and configuration.

etcd/version

Allows operators to determine the version of etcd to install Note: Changes should be coordinate with KRIB Kubernetes version

krib/cluster-masters

List of the machine(s) assigned as cluster master(s). If not set, the automation will elect leaders and populate the list automatically.

rook/data-dir-host-path

This should be set to match the desired location for Ceph storage.

Default: /docker

In future versions, this should be calculated or inferred based on the system inventory

etcd/server-ca-name

Allows operators to set the CA name for the server certificate Requires Cert Plugin

etcd/server-ca-pw

Allows operators to set the CA password for the server certificate Requires Cert Plugin

krib/cluster-bootstrap-token

Defines the bootstrap token to use. Default is 'fedcba.fedcba9876543210'.

krib/cluster-kubeadm-cfg

Once the cluster initial master has completed startup, then the KRIB config task will record the bootstrap configuration used by 'kubeadm init'. This provides a reference going forward on the cluster configurations when it was created.

krib/cluster-master-vip

For High Availability (HA) configurations, a floating IP is required by the load balancer.

krib/operate-on-node

This Param specifies a Node in a Kubernetes cluster that should be operated on. Currently supported operations are 'drain' and 'uncordon'.

The drain operation will by default maintain the contracts specified by PodDisruptionBudgets.

Options can be specified to override the default actions by use of the 'krib/operate-options' Param. This Param will be passed directly to the 'kubectl' command that has been specified by the 'krib/operate-action' Param setting (defaults to 'drain' operation if nothing specified).

The Node name must be a valid cluster member name, which by default in a KRIB built cluster; the fully qualified value of the Machine object 'Name' value.

krib/cluster-api-port

The API bindPort number for the cluster masters. Defaults to '6443'.

krib/networking-provider

This Param can be used to specify either 'flannel' or 'calico' network providers for the Kubernetes cluster. This is completed using the provider specific YAML definition file.

The only supported providers are:

> flannel calico

The default is 'flannel'.

helm/version

Allows operators to determine the version of etcd to install Note: Changes should be coordinate with KRIB Kubernetes version

krib/cluster-masters-on-etcds

For development clusters, allows running etcd on the same machines as the Kubernetes masters RECOMMENDED: set to false for production clusters

krib/cluster-name

Allows operators to set the Kubernetes cluster name

etcd/client-ca-name

Allows operators to set the CA name for the client certificate Requires Cert Plugin

etcd/peer-port

Allows operators to set the port for the cluster peers

krib/cluster-image-repository

Allows operators to specify the location to pull images from for Kubernetes. Defaults to 'k8s.gcr.io'.

krib/cluster-master-count

Allows operators to set the number of machines required for the Kubernetes cluster. Machines will be automatically added until the number is met. NOTE: should be an odd number

sonobuoy/binary

Downloads tgz with compiled sonobuoy executable. The full path is included so that operators can choose the correct version and archtiecture

etcd/name

Allows operators to set a name for the etcd cluster

etcd/peer-ca-name

Allows operators to set the CA name for the peer certificate Requires Cert Plugin

krib/cluster-service-subnet

Allows operators to specify the service subnet CIDR that will be used during the 'kubeadm init' process of the cluster creation.

Defaults to "10.96.0.0/12".

krib/dashboard-config

Set this string to an HTTP or HTTPS reference for a YAML configuration to use for the Kubernetes Dashboard.

docker/working-dir

Allows operators to change the Docker working directory

krib/cluster-pod-subnet

Allows operators to specify the podSubnet that will be used by CoreDNS during the 'kubeadm init' process of the cluster creation.

krib/cluster-dns

Allows operators to specify the DNS address for the cluster name resolution services.

Set by default to "10.96.0.10".

WARNING: This IP Address must be in the same range as the "krib/cluster-service-cidr" specified addresses.

krib/cluster-is-production

By default the KRIB cluster mode will be set to dev/test/lab (whatever you wanna call it). If you set this Param to true then the cluster will be tagged as in Production use.

If the cluster is in Production mode, then the state of the various Params for new clusters will be preserved, preventing the cluster from being overwritten.

If NOT in Production mode, the following Params will be wiped clean before building the cluster. This is essentially a destructive pattern.

> krib/cluster-admin-conf - the admin.conf file Param will be wiped krib/cluster-join - the Join token will be destroyed

This allows for "fast reuse" patterns with building KRIB clusters, while also allowing a cluster to be marked Production and require manual intervention to wipe the Params to rebuild the cluster.

krib/packages-to-prep

List of packages to install for preparation of KRIB install process. Designed to be used in some processes where pre-prep of the packages will accelerate the KRIB install process. More specifically, in Sledgehammer Discover for Live Boot cluster install, the 'docker' install requires several minutes to run through selinux context changes.

Simple space separated String list.

etcd/cluster-profile

Part of the Digital Rebar Cluster pattern, this parameter is used to identify the machines used in the etcd cluster This parameter is REQUIRED for KRIB and etcd cluster contruction

krib/cluster-join-command

Param is set (output) by the cluster building process

helm/charts

Install Helm Charts

Array of charts to install via Helm. The list will be followed in order.

Work is idempotent: No action is taken if charts are already installed.

Fields: chart and name are required.

Options exist to inject additional control flags into helm install instructions:

- name: name of the chart (required)
- chart: reference of the chart (required) - may rely on repo, path or other helm install [chart] standard
- namespace: kubernetes namespace to use for chart (defaults to none)
- params: map of parameters to include in the helm install (optional). Keys and values are converted to –[key] [value] in the install instruction.
- sleep: time to wait after install (defaults to 10)
- wait: wait for name (and namespace if provided) to be running before next action
- prekubectl (optional) array of kubectl [request] commands to run before the helm install
- postkubectl (optional) array of kubectl [request] commands to run after the helm install
- targz (optional) provides a location for a tar.gz file containing charts to install. Path is relative.

- templates (optional) map of DRP templates keyed to the desired names (must be uploaded!) to render before doing other work.
- repo (optional) adds the requested repo to the helm using *helm repo add* before installing helm. syntax is *[repo name] [repo path]*.

example:

```
[
  {
    "chart": "stable/mysql",
    "name": "mysql"
  }, {
    "chart": "istio-1.0.1/install/kubernetes/helm/istio",
    "name": "istio",
    "targz": "https://github.com/istio/istio/releases/download/1.0.1/istio-1.
↪0.1-linux.tar.gz",
    "namespace": "istio-system",
    "params": {
      "set": "sidecarInjectorWebhook.enabled=true"
    },
    "sleep": 10,
    "wait": true,
    "kubectlbefore": ["get nodes"],
    "kubectlafter": ["get nodes"]
  }
]
```

krib/cluster-master-certs

Requires Cert Plugin

etcd/peer-ca-pw

Allows operators to set the CA password for the peer certificate If missing, will be generated Requires Cert Plugin

krib/cluster-bootstrap-ttl

How long BootStrap tokens for the cluster should live. Default is ‘24h0m0s’.

Must use a format similar to the default.

krib/cluster-kubernetes-version

Allows operators to specify the version of Kubernetes containers to pull from the ‘krib/cluster-image-repository’.

krib/labels

Used for adhoc node specification, labels should be set.

NOTES: * Use krib/label-env to set the env label! * Use inventory/data to set physical characteristics

etcd/client-ca-pw

Allows operators to set the CA password for the client certificate Requires Cert Plugin

stages

The content package provides the following stages.

krib-helm

Installs and runs Helm Charts after a cluster has been constructed. This stage is idempotent and can be run multiple times. This allows operators to create workflows with multiple instances of this stage. The charts to run are determined by the helm/charts parameter.

Due to helm downloads, this stage requires internet access.

This stage also creats a tiller service account. For advanced security, this configuration may not be desirable.

krib-pkg-prep

Simple helper stage to install prereq packages prior to doing the kubernetes package installation. This just helps us get a Live Boot set of hosts (eg Sledgehammer Discovered) prepped a little faster with packages in some use cases.

krib-set-time

Helper stage to set time on the machine - DEV

krib-sonobuoy

Installs and runs Sonobuoy after a cluster has been constructed. This stage is idempotent and can be run multiple times. The purpose is to ensure that the KRIB cluster is conformant with standards

If credentials are required so that the results of the run are pushed back to DRP files.

Roadmap items: * eliminate need for DRPCLI credentials * make "am I running" detection smarter

krib-operate-drain

This stage runs an Drain operation on a given KRIB built Kubernetes node. It uses the 'krib-operate-drain' Profile

In addition - you may set the following Params on a Machine object to override the default behaviors of this stage:

> krib/operate-action - action to take (drain or uncordon) krib/operate-on-node - a Kubernetes node name to operate on krib/operate-options - command line arguments to pass to the
>
> > 'kubectl' command for the action

If the 'krib/operate-on-node' Param is empty, the node that is currently running the Stage will be operated on. Otherwise, specifying an alternate Node allows remote draining a node.

DRAIN NOTES: this Stage does a few things that MAY BE VERY BAD !!

1. service pods are ignored for the drain operation

2. –delete-local-data is used to evict pods using local storage

Default options are '–ignore-daemonsets –delete-local-data' to the drain operation. If you override these values (by setting 'krib/operate-options') you MAY NEED to re-specify these values, otherwise, the Node will NOT be drained properly.

These options may mean your data might be nuked.

krib-operate

This stage runs an Operation (drain|uncordon) on a given KRIB built Kubernetes node. You must specify action you want taken via the 'krib/operate-action' Param. If nothing specified, the default action will be to 'drain' the node.

In addition - you may set the following Params to alter the behavior of this stage:

> krib/operate-action - action to take (drain or uncordon) krib/operate-on-node - a Kubernetes node name to operate on krib/operate-options - command line arguments to pass to the
>
> > 'kubectl' command for the action

DRAIN NOTES: this Stage does a few things that MAY BE VERY BAD !!

1. service pods are ignored for the drain operation
2. –delete-local-data is used to evict pods using local storage

Default options are '–ignore-daemonsets –delete-local-data' to the drain operation. If you override these values (by setting 'krib/operate-options') you MAY NEED to re-specify these values, otherwise, the Node will NOT be drained properly.

These options may mean your data might be nuked.

UNCORDON NODES: typically does not require additional options

krib-operate-cordon

This stage runs a Cordon operation on a given KRIB built Kubernetes node. It uses the 'krib-operate-cordon' Profile.

In addition - you may set the following Params on a Machine object to override the default behaviors of this stage:

> krib/operate-action - action to take (cordon or uncordon) krib/operate-on-node - a Kubernetes node name to operate on krib/operate-options - command line arguments to pass to the
>
> > 'kubectl' command for the action

If the 'krib/operate-on-node' Param is empty, the node that is currently running the Stage will be operated on. Otherwise, specifying an alternate Node allows remote cordon a node.

krib-operate-delete

This stage runs an Delete node operation on a given KRIB built Kubernetes node. It uses the 'krib-operate-delete' Profile

In addition - you may set the following Params on a Machine object to override the default behaviors of this stage:

> krib/operate-action - action to take krib/operate-on-node - a Kubernetes node name to operate on krib/operate-options - command line arguments to pass to the
>
> > 'kubectl' command for the action

If the 'krib/operate-on-node' Param is empty, the node that is currently running the Stage will be operated on. Otherwise, specifying an alternate Node allows remote delete a node.

WARNING: THIS OPERATE DESTROYS A KUBERNETES NODE!

Presumably, you want to 'krib-operate-drain' the node first to remove it from the cluster and drain it's workload to other cluster workers prior to deleting the node.

krib-operate-uncordon

This stage runs an Uncordon operation on a given KRIB built Kubernetes node. This returns a Node back to service in a Kubernetes cluster that has previously been drained. It uses the 'krib-operate-uncordon' Profile

In addition - you may set the following Params on a Machine object to override the default behaviors of this stage:

> krib/operate-action - action to take (drain or uncordon) krib/operate-on-node - a Kubernetes node name to operate on krib/operate-options - command line arguments to pass to the
>
> > 'kubectl' command for the action

If the 'krib/operate-on-node' Param is empty, the node that is currently running the Stage will be operated on. Otherwise, specifying an alternate Node allows remote uncordon on a node.

Default options are '' (empty) to the uncordon operation.

tasks

The content package provides the following tasks.

docker-install

Installs Docker using O/S packages

krib-config

Sets Param: krib/cluster-join, krib/cluster-admin-conf Configure Kubernetes using Kubeadm This uses the Digital Rebar Cluster pattern so krib/cluster-profile must be set

krib-sonobuoy

Installs Sonobuoy and runs it against the cluster on the leader. This uses the Digital Rebar Cluster pattern so krib/cluster-profile must be set.

NOTE: Sonobuoy may take over an HOUR to complete. The task will be in process during this time.

The tasks only run on the leader so it must be included in the workflow. All other machines will be skipped so it is acceptable to run the task on all machines in the cluster.

kubernetes-install

Downloads Kubernetes installation components from repos This task relies on the O/S packages being updated and accessible. NOTE: Access to update repos is required!

krib-pkg-prep

Installs prerequisite OS packages prior to starting KRIB install process. In some use cases this may be a faster pattern than performing the steps in the standard templates.

For example - Sledgehammer Discover nodes, add 'krib-prep-pkgs' stage. As machine is finishing prep - you can move to setting up other things, before kicking off the KRIB workflow.

Uses packages listed in the 'default' Schema section of the Param 'krib/packages-to-prep'. You can override this list by setting the Param in a Profile or directly on the Machines to apply this to.

Packages MUST exist in the repositories on the Machines already.

etcd-config

Sets Param: etcd/servers If installing Kubernetes via Kubeadm, make sure you install a supported version! This uses the Digital Rebar Cluster pattern so etcd/cluster-profile must be set

krib-helm

Installs Helm and runs helm init (which installs Tiller) on the leader. This uses the Digital Rebar Cluster pattern so krib/cluster-profile must be set.

The install checks to see if tiller is running and may skip initialization.

The tasks only run on the leader so it must be included in the workflow. All other machines will be skipped so it is acceptable to run the task on all machines in the cluster.

krib-dev-reset

Clears Created Params: krib/*, etcd/*

3.29.20 kubespray

The following documentation is for kubespray content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

params

The content package provides the following params.

ansible/groups-members

Inventory of Rebar Machines into Ansible Host Groups when using the dynamic inventory generator for Ansible. Often generated by automation or the UX.

ansible/groupvars

Maps Rebar Profiles to Ansible Host Groups when using the dynamic inventory generator for Ansible.

ansible/hostvars

Sets additional variables in the Ansible dynamic inventory generator for Ansible.

ansible/parent-groups

Creates Ansible Nested Host Groups when using the dynamic inventory generator for Ansible. There is no matching concept in Digital Rebar.

kube_api_version

Overrides default password for user kube

kube_version

Overrides default for admin user

ansible/groups

Creates Ansible Host Groups when using the dynamic inventory generator for Ansible.

3.29.21 kvm-test

The following documentation is for kvm-test content package at version Unspecified.

Unspecified

3.29.22 license

The following documentation is for license content package at version Unspecified.

Unspecified

params

The content package provides the following params.

rackn/license

The Base64 encoded signed license json blob

rackn/license-object

The user viewable form of the license object

3.29.23 os-other

The following documentation is for os-other content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

3.29.24 packet-ipmi

The following documentation is for packet-ipmi content package at version Unspecified.

Unspecified

params

The content package provides the following params.

packet/always-pxe

This parameter indicates if the node should always pxe.

packet/api-key

The API implies the user and the Project that the plugin will act as.

packet/ipxe-script-url

This parameter is the URL of the default ipxe file. If unspecified, the option will default to the current DRP endpoint default.ipxe file.

packet/name

This parameter is the name to use for the packet machine.

packet/facility

This parameter describes the Packet facility for this node.

packet/import-existing

This parameter tells the plugin to import existing VMs or not.

packet/plan

This parameter is the Packet plan for this machine.

packet/project-id

This parameter describes the Packet Project ID for this node.

packet/sos

This parameter is the ssh hostname to use to access the serial console of this machine.

packet/uuid

This parameter is used by the Packet IPMI Plugin to access the API

3.29.25 raid

The following documentation is for raid content package at version Unspecified.

Unspecified

3.29.26 slack

The following documentation is for slack content package at version Unspecified.

Unspecified

params

The content package provides the following params.

slack/bot-key

The Key implies the user and the channel that the bot will act as.

3.29.27 sledgehammer-builder

The following documentation is for sledgehammer-builder content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

Unspecified

3.29.28 task-library

The following documentation is for task-library content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

3.29.29 RackN Task Library

This content package is used to collect useful stages and operations for Digital Rebar.

Cluster Stages

These stages implement the *Multi-Machine Cluster Pattern*.

Allows operators to orchestrate machines into sequential or parallel operation.

Inventory Stage

Convert gohai and other JSON data into a map that can be used for analysis, classification or filtering.

params

The content package provides the following params.

inventory/check

Using BASH REGEX, define a list of inventory data fields to test using Regular Expressions. Fields are tested in sequence, the first to fail will halt the script

inventory/collect

Map of commands to run to collect Inventory Input Each group includes the fields with jq maps to store For example: adding "drpcli gohai" will use gohai JSON as input Then jq will be run with provided values to collect inventory into a inventory/data as a simple map.

To work correctly, Commands should emit JSON

Gohai example:

> **::**
>
> > **{**
> >
> > > **gohai: {** Command: "drpcli gohai", Fields: {
> > >
> > > > RAM: ".System.Memory.Total / 1048576 | floor", NICs: ".Networking.Interfaces | length"
> > >
> > > }
> >
> > }
>
> }

inventory/CPUs

From inventory/data, used for indexable data

inventory/CPUCores

From inventory/data, used for indexable data

inventory/CPUType

From inventory/data, used for indexable data

inventory/Manufacturer

From inventory/data, used for indexable data

inventory/SerialNumber

From inventory/data, used for indexable data

inventory/integrity

Allows operators to compare new inventory/data to stored inventory/data on the machine. If true and values not match (after first run) then Stage will fail

inventory/ProductName

From inventory/data, used for indexable data

inventory/data

Stores the data collected by the fieds set in inventory/collect If inventory/integrity true, then used for comparision data

inventory/flatten

Creates each inventory value as a top level Params This is needed if you want to filter machines in the API by inventory data. For example, using Terraform filters. WARNING: This will create LOTS of Params on machines!

inventory/NICs

From inventory/data, used for indexable data

inventory/RAM

From inventory/data, used for indexable data

inventory/CPUSpeed

From inventory/data, used for indexable data

inventory/Family

From inventory/data, used for indexable data

stages

The content package provides the following stages.

inventory

Collects selected fields from Gohai into a simpler, flat list. The process uses JQ filters, defined in inventory/fields, to build inventory/data on each machine. Also, applies the inventory/check map to the data and will fail if the checks do not pass.

tasks

The content package provides the following tasks.

cluster-step

Waits until machine is in Nth (or earlier) position Replies on cluster-remove to advance queue Typical cluster workflow is add->step->remove->sync Where step or sync are optional. See cluster-add task for more details

stage-chooser

This task uses the stage-chooser/next-stage and stage-chooser/reboot parameters to change the stage of the current machine and possibly reboot.

This is not intended for use in a stage chain or workflow. But a transient stage, that can be used on a machine that is idle with a runner executing.

cluster-add

Injects the Machine Uuid into the cluster/machines param. This is the first step in a the cluster workflow Typical cluster workflow is add->step->remove->sync Where step or sync are optional.

The cluster-* stages provide basic queue management for machines with a shared profile. That profile must have a cluster/profile field that names the profile. Once the queue is running, machines are tracked in the cluster/machines parameter.

cluster-add enqueues the machine cluster-step services the queue (takes the next cluster/step in line) cluster-remove dequeues the machine (should be done AFTER servicing) cluster-sync forces all machines to wait until the queue is empty

If you are doing a rolling upgrade, use the following sequence: cluster-add -> cluster-step -> [do work] -> cluster-remove

If you are coalescing work over multiple machines, use the following sequence: cluster-add -> [do work] -> cluster-remove -> cluster-sync

The cluster-add process will automatically elect cluster/leader-count leaders during the enqueue step. These cluster/leaders will be be maintained after the run is over and require manual editing to remove. It is possible to pre-create the leasder list.

It is possible to do multiple add/remove steps in a single workflow

If your cluster is NOT behaving, you can rescue or escape from the cluster stages by setting cluster/escape on the profile. This will interrupt a cluster in process. Using escape=0 will allow the cluster stages to exit normally and noop while using escape=1 will exit with an error and the stages will stop progressing. Escape=1 is safter since downstream actions are blocked.

cluster-remove

Removes the Machine Uuid into the cluster profile Typical cluster workflow is add->step->remove->sync Where step or sync are optional. See cluster-add task for more details

cluster-sync

Waits until no machines remain. Use with cluster-remove Typical cluster workflow is add->step->remove->sync Where step or sync are optional. See cluster-add task for more details

inventory-check

Using the inventory/collect parameter, filters the JSON command output into inventory/data hashmap for rendering and validation. Will apply inventory/check for field valation

network-lldp

Assumes Sledgehammer has LLDP daemon installed so that we can collect data. Also requires the system to have LLDP enabled on neighbors including switches to send the correct broadcast packets.

3.29.30 terraform

The following documentation is for terraform content package at version v1.11.0-0-48c1838f5b6343685226ff09045d49cd285e628e.

3.29.31 Terraform Provider for Digital Rebar Provision

The following instructions to install the needed Digital Rebar Provision integrations to support Terraform. Terraform integrations require that machines can reboot.

The Terraform system is not synchronized with Digital Rebar, this documentation is only intended to cover the provider.

Note: As of DRP v3.8 (workflows enabled), changes to bootenvs will automatically cause reboots if the DRP runner/agent is allowed to keep running. Otherwise, an IPMI plugin is required. We recommend using both Workflow and IPMI based reboots to ensure that the systems return to a consistent state.

Source Code: https://github.com/rackn/terraform-provider-drp

Video Demos:

- v3.8 DRP https://youtu.be/RtuZQHKmd9U
- v3.2 DRP https://www.youtube.com/watch?v=6MLyUVgnVo4

License

The Terraform Provider for Digital Rebar Provision is APLv2 licensed. Advanced features or workflow may require capabilities that use different licenses.

Prereqs

Before starting this process, a Digital Rebar Provision (DRP) server is required, along with the ability to provision machines. These machines could be VMs, Packet servers or physical servers in a data center.

You must also have installed Terraform and secured the Digital Rebar Provision Terraform Provider. You can build the provider from source or retrieve a compiled version from the Github releases area, https://github.com/rackn/terraform-provider-drp/releases.

Basic Operation

The DRP Terraform Provider uses a pair of Machine Parameters to create an inventory pool. Only machines with these parameters will be available to the provider.

The *terraform/managed* parameter determines the basic inventory availability. This flag must be set to true for Teraform to find machines.

The *terraform/allocated* parameter determines when machines have been assigned to a Terraform plan. When true, the machine is now being managed by Terraform. When false, the machine is available for allocation.

Using the RackN *terraform-ready* stage will automatically set these two parameters.

The Terraform Provider can read additional fields when requesting inventory. In this way, users can request machines with specific characteristics.

DRP Machine Configuration

Note: these instructions assume that you are using the RackN extension; however, it is not required for Terraform operation.

Install the *Terraform* content package from RackN. This content provides the parameters (described above) and a stage (*terraform-ready*) to configure Terraform properties on machines.

Create a workflow that includes the *terraform-ready* stage to set the two parameters. Once these values are set they do not have to be run again, but there is no harm in leaving the stage in place.

Before testing Terraform, it is good practice to make sure you can manually cycle machines by changing their workflow (or change stage and reboot). These changes should result in a machine provisioning cycle.

Make sure that you know your endpoint URI, user and password.

Installing The Terraform Digital Rebar Provision Provider

Please install Terraform on your system: https://www.terraform.io/intro/getting-started/install.html

Download (or build) the DRP Provider from https://github.com/rackn/terraform-provider-drp/releases.

Initialize the plugin using *terraform init*

Create a Plan using the DRP Provider Resources

The DRP Provider exposes all the objects in Digital Rebar Project so you can script against any operation.

The critical block is the provider block which identifies the provider and login information (shown here with default values):

```
variable "api_url" {
  default = "https://127.0.0.1:8092"
}
variable "api_user" {
  default = "rocketskates"
}
variable "api_password" {
  default = "r0cketsk8ts"
}
provider "drp" {
  api_user     = "${var.api_user}"
  api_password = "${var.api_password}"
  api_url      = "${var.api_url}"
}
```

Once you have the provider block, you can name resource blocks using the normal object key values. For example, a machine resource looks like this:

```
resource "drp_machine" "one_random_node" {
  count = 1
  Workflow    = "centos-7"
  Description = "updated description"
  Name        = "greg2"
  userdata    = "yaml cloudinit file"
}
```

You can set any Machine property by naming the property with correct capitalization. You can use the Terraform syntax to create more complex models like Meta, Profiles.

Note: If you set Profiles, Params or Meta you will override other existing information in the machine! The following helpers have been defined to avoid this:

- *add_profiles*: allows you to add profiles to the machine without override other profiles.

There are many options to set including filters, parameters and profiles. For a full example, please look at https://github.com/rackn/terraform-provider-drp/blob/master/test.tf.example

Picking Machines with a Pool

You can add a *pool* block into the plan that will select machines based on the pool name. This is helpful if you want to partiation your machines. Pools use the *terraform/pool* Param on the machines and will be assumed to be *default* if omitted.

For example:

```
pool = "deep_eddy"
```

Picking Machines with Filter

You can add a *filters* block into the plan that will select machines based on criteria. This is helpful if you want to select specific types of machines based on Param data. Filters use the API filters definition and are JSON formatted (types are guessed so numbers and bools are coerced). See *API Filters*.

For example:

```
filters = [{
        name = "Name"
        jsonvalue = "greg2"
    }]
```

You can only filter on indexed fields and defined Params. Further, you cannot search deeply into Params, only the first level value is matched.

Special Complete and Decommissioning Fields

The provider watches until the machine reaches the *complete* or *complete-no-wait* stages; however, you can customize this behavior by setting the *completion_stage* to the plan.

You can override the default the decommissioning flow (set workflow or stage back to *discover*) by adding *decommission_workflow* = *"my_decom_workflow"* to the plan.

You can also override the return icon (*map outline*) and color ('black') by adding *decommission_icon* and *decommission_color* to the plan. Machine icons are handy ways to quickly show status of a provisioning cycle.

Users can set icons using

```
Meta {
    icon = "leaf"
    color = "green"
}
```

Creating RAW Machines using Cloud IPMI plugins

The *drp_machine* resource relies on having a pool of machines already configured; however, you can use the *drp_raw_machine* resource to create machines in Digital Rebar Provision. If you are using an IPMI plugin that supports creating machines, such as Packet or Virtualbox, and set the *machine-plugin* value then the plugin will create (and destroy) the associated machine in the target platform. This can be a very powerful way to build and manage clusters.

It is possible to use raw and pooled machines together by also setting the *terraform/managed* and *terraform/allocated* parameters when creating machines. This will allow Terraform to treat newly created machines as a pool. It's important to include chained *depends_on* in the resource blocks when using this approach in a single plan.

You may also set *terraform/pool* to something. The default behavior assumes *default* but you can use this Param to manage multiple pools of resources. Select pools using *pool* in the *drp_machine* resources.

Note: Unlike the *drp_machine* resource, this resource does not wait until the workflow has completed. It will return when the machine has been create API returns.

An example of the *drp_raw_machine* resource with correct parameter values is

```
resource "drp_raw_machine" "packet-machines" {
  Description = "Terraform Added RAW"
  Workflow = "discover"
  Name = "packet_machine"
  Params {
    "machine-plugin" = "packet-ipmi"
    "packet/plan" ="baremetal_0"
    "terraform/managed" = "true"
    "terraform/allocated" = "false"
    "terraform/pool" = "default"
  }
```

Running Terraform

Just use *terraform apply* and *terraform destroy* and as normal!

Note: the examples above use variables for endpoint login. The syntax for overriding these variables to set environment variables starting with *export TF_VAR_my_var=* and the variable name or pass *-var 'api_url=https://[ip address]:8092'*. User names and passwords should never be hard coded into plan files!

Extending the Features

Using the *terraform/owner* parameter helps administrators track who is using which machines. You may also choose to create multiple DRP users to help track activity.

It is highly recommended that you include decommissioning steps (disk scrub, bios reset, etc) and additional burn-in to validate systems during the recovery cycle.

Using IPMI to reset machines is a safer bet than relying on the DRP runner to soft reboot systems. If you want to make sure that you have a consistent recovery process, IPMI is highly recommended.

To improve delivery time:

1. Keep the machines running
2. Use image based provisioning instead of netboot.

Note: If you are relying on the DRP Running workflow to start allocation and recovery, make sure that you have your tokens set to never expire!

Summary

Now that these steps are completed, the Digital Rebar Provision Terraform Provider will integrate like any cloud provider.

3.29.32 virtualbox-ipmi

The following documentation is for virtualbox-ipmi content package at version Unspecified.

Unspecified

params

The content package provides the following params.

virtualbox/name

This is the name used by virtualbox for the machine.

virtualbox/user

This is the user that the virtualbox machine is running as

virtualbox/vm-path

This is the path to store vms.

virtualbox/vram-size-mb

The size of the virtual Video Ram in MB.

virtualbox/cpus

Number of cpus to create for this virtualbox machine

virtualbox/disk-size-mb

The size of the virtual disk in MB.

virtualbox/id

This parameter is used by the IPMI Plugin to access the BMC

virtualbox/mem-size-mb

The size of the memory disk in MB.

3.30 Contributing to Digital Rebar Provision

Before submitting pull requests, please make sure to read and understood the Apache license. Submitting a pull is considered to be accepting the project's license terms.

3.30.1 Guidelines for Pull Requests

We follow typical Github fork/pull request processes.

- Must be Apache 2 license
- For bugs & minor items (20ish lines), we can accept the request at our discretion
- Does not inject vendor information (Name or Product) into Digital Rebar except where relevant to explain utility of push (e.g.: help documentation & descriptions).
- Passes code review by Digital Rebar team reviewer
- Does not degrade the security model of the product
- Does not reduce code coverage
- Items requiring more scrutiny
  - Major changes
  - CLI/API changes, especially breaking compatability
  - New technology
- Pull requests should be against a defined feature branch in the Digital Rebar repo

3.30.2 Timing

- Accept no non-bug fix push requests within 2 weeks of a release fork
- No SLA - code accepted at PTLs discretion. No commitment to accept changes.

3.30.3 Coding Expectations

- Copyright & License header will be included in files that can tolerate headers
- At least 1 line comments as header for all methods
- Documentation for API calls concurrent with pull request

3.30.4 Testing/ Validation

- For core functions, the push will be validated to ensure it does NOT break build, deploy, or our commercial products
- For operating systems that are non-core, we will *not* validate on the target OS for the push.
- We expect that a pull request will be built and tested in our CI system before the push can be accepted.

3.31 Trademark

The Digital Rebar name and mark are maintained by RackN until the project goverance moves to an independent body. Users of the project are welcome to use the name and mark. For future project management purposes, RackN requests vendors obtain permission for commercial uses.

3.31.1 Name

Digital Rebar usage options are as follows:

- The project name is “Digital Rebar” as two words, both capitalized
- Acceptable alternatives:
  - DigitalRebar (avoid in written text in favor of Digital Rebar)
  - rebar.digital (the “.” is required in this format)
  - dR or DR (do not use Dr)
  - Internally within the project, it is acceptable to just say “Rebar”

3.31.2 Logos

It is acceptable to use the Digital Rebar logos when referencing the project or workloads that leverage the project.

Large Icon:

Small Icon

3.31.3 Mascot

Cloud Native Metal Bear

3.32 License

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

 5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

 6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

 7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

 8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
    result of this License or out of the use or inability to use the
    Work (including but not limited to damages for loss of goodwill,
    work stoppage, computer failure or malfunction, or any and all
    other commercial damages or losses), even if such Contributor
    has been advised of the possibility of such damages.

 9. Accepting Warranty or Additional Liability. While redistributing
    the Work or Derivative Works thereof, You may choose to offer,
    and charge a fee for, acceptance of support, warranty, indemnity,
    or other liability obligations and/or rights consistent with this
    License. However, in accepting such obligations, You may act only
    on Your own behalf and on Your sole responsibility, not on behalf
    of any other Contributor, and only if You agree to indemnify,
```

(continues on next page)

(continued from previous page)

CHAPTER 4

License

DigitalRebar Provision code is available is available from multiple authors under the Apache 2 license.

Digital Rebar Provision Documentation is available from multiple authors under the Creative Commons license with Attribution.

```
Work licensed under a Creative Commons license is governed by applicable copyright␣
↪law.
This allows Creative Commons licenses to be applied to all work falling under␣
↪copyright,
including: books, plays, movies, music, articles, photographs, blogs, and websites.
Creative Commons does not recommend the use of Creative Commons licenses for software.

However, application of a Creative Commons license may not modify the rights allowed␣
↪by
fair use or fair dealing or exert restrictions which violate copyright exceptions.
Furthermore, Creative Commons licenses are non-exclusive and non-revocable.
Any work or copies of the work obtained under a Creative Commons license may continue
to be used under that license.

In the case of works protected by multiple Creative Common licenses,
the user may choose either.
```

Index