runcible

Grayson Head

Feb 26, 2020

CONTENTS:

1 Introduction **1**
- 1.1 What is Runcible? . . . 1
- 1.2 What is a Runcible? . . . 1
- 1.3 How is This Different from Ansible and Others? . . . 1
- 1.4 I Want to Get Involved . . . 2

2 Getting Started **3**
- 2.1 Installation . . . 3
- 2.2 YAML File . . . 3
- 2.3 MergeDB Datasource . . . 6

3 Components **15**
- 3.1 Modules . . . 15
- 3.2 Schedulers . . . 16
- 3.3 Needs . . . 16

4 Driver Index **19**
- 4.1 Cumulus . . . 19
- 4.2 Modules . . . 19

5 API **25**
- 5.1 API Device Example . . . 25

6 Contribution Guide **27**
- 6.1 Current State of Runcible . . . 27
- 6.2 Roadmap . . . 27

7 Indices and tables **29**

CHAPTER

ONE

INTRODUCTION

1.1 What is Runcible?

Runcible is a framework and CLI application to allow for declarative switch management. It intakes a declared state for a device in YAML or JSON and turns it into a list of idempotent commands to configure that device, it then runs them over SSH, Telnet, or RS-232 (plugins can be written for any text based terminal, or REST API however.)

Runcible provides a high-level API for Python developers to leverage to manage devices programmatically, and also provides a YAML interface with configuration layering and inheritance for network engineers with any amount of programming experience.

1.2 What is a Runcible?

Depending on who you ask, its either a nonsensical word, a word for spork, or a faster than light transportation network.

1.3 How is This Different from Ansible and Others?

Runcible was created to solve three major problems in regards to Network Automation:

1.3.1 Interface Commonality

One of the core components of Runcible is a datatype known as *Modules*. Modules are plugin-independent interfaces that allow data with a common schema to be passed into multiple types of plugins. This allows for a large amount of configuration re-use between similar devices produced by different vendors. I.E. the vlans module should be implemented by any switch plugin that supports vlans, and the configuration should be identical (assuming the device supports the entire featureset of the vlans module.)

1.3.2 Topology Awareness

One important aspect of any kind of network automation is ensuring that bad automation runs are dealt with, and that you stage your changes in a topology-aware manner. You wouldn't want a bad change to propagate to your entire core switch fabric and take your network down. Runcible provides *Schedulers* that allow for intelligent automation runs allow you to ensure that your automated changes are made intelligently, and also control rollback and failure behavior.

1.3.3 Protocol Agnosticism

Runcible doesn't operate on a defined set of protocols. While most providers will go with a text based protocol (SSH, telnet, RS232), any protocol is supported. Runcible provides some sane default protocol modules based on paramiko for SSH, and pyserial for RS232 terminals, but has loose shim classes that allow plugin writers to implement any protocol they deem necessary without inhibiting any of Runcible's features. This allows users to use their same automation repository for both bootstrapping devices via a serial connection, as well as making changes via SSH, Telnet, REST, or even protocols that haven't been invented yet.

1.4 I Want to Get Involved

Great! Head over to the *Contribution Guide* or give it a spin: *Getting Started*.

CHAPTER

TWO

GETTING STARTED

2.1 Installation

To install Runcible via pip, run:

```
pip3 install runcible
```

Note: Runcible is compatible with python 3.5 and higher

2.2 YAML File

The easiest way to get started with Runcible is to write some definition files in YAML. This will allow you to become familiar with all of Runcible's *Modules* and try out some configurations. If you plan on managing a large infrastructure, it is highly recommended to utilize Runcible with it's sister project, *Setting Up MergeDB*.

For example, to configure a single Cumulus switch, you could generate a yaml file like so:

```
# examples/yaml/cumulus_switch.yaml
---
core:
  meta:
    device:
      default_management_protocol: ssh
      driver: cumulus
      ssh:
        hostname: 192.168.122.166
        username: cumulus
  bonds:
  - mtu: 9000
    name: po4
    pvid: 25
    slaves:
    - swp4
    vlans:
    - 20
    - 50
  interfaces:
  - name: swp1
    ipv4_addresses:
      - 192.168.2.2/24
```

(continues on next page)

(continued from previous page)

```
    vlans: []
  - name: swp2
    vlans:
    - 10
    - 20
    - 30
    - 40
    - 50
  - name: swp3
    vlans:
    - 10
    - 20
    - 30
    - 40
    - 50
  - name: swp5
    vlans:
    - 10
    - 20
    - 30
    - 40
    - 50
  ntp_client:
    interface: eth0
    servers:
    - 0.cumulusnetworks.pool.ntp.org
    - 1.cumulusnetworks.pool.ntp.org
    - 2.cumulusnetworks.pool.ntp.org
    - 3.cumulusnetworks.pool.ntp.org
  system:
    hostname: core
  vlans:
  - id: 10
    name: vlan10
  - id: 20
    name: vlan20
  - id: 30
    name: vlan30
  - id: 40
    name: vlan40
  - id: 50
    name: vlan50
```

To understand the contents of this file in depth, visit the *Modules* section of the documentation.

To apply this configuration to the switch in question, simply run:

```
runcible -y {path_to_yaml_file} '*.' apply
```

You will get output that looks like:

```
The following changes will be applied:
Device core:
========================================
bonds needs:
    bonds.po4.CREATE
    bonds.po4.mtu.SET: 9000
    bonds.po4.slaves.ADD: swp4
```

(continues on next page)

(continued from previous page)

```
    bonds.po4.vlans.ADD: 20
    bonds.po4.vlans.ADD: 50
    bonds.po4.pvid.SET: 25
interfaces needs:
    interfaces.swp1.vlans.ADD: 10
    interfaces.swp1.vlans.ADD: 20
    interfaces.swp1.vlans.ADD: 30
    interfaces.swp1.vlans.ADD: 40
    interfaces.swp1.vlans.ADD: 50
    interfaces.swp2.vlans.ADD: 10
    interfaces.swp2.vlans.ADD: 20
    interfaces.swp2.vlans.ADD: 30
    interfaces.swp2.vlans.ADD: 40
    interfaces.swp2.vlans.ADD: 50
    interfaces.swp3.vlans.ADD: 10
    interfaces.swp3.vlans.ADD: 20
    interfaces.swp3.vlans.ADD: 30
    interfaces.swp3.vlans.ADD: 40
    interfaces.swp3.vlans.ADD: 50
    interfaces.swp5.vlans.ADD: 10
    interfaces.swp5.vlans.ADD: 20
    interfaces.swp5.vlans.ADD: 30
    interfaces.swp5.vlans.ADD: 40
    interfaces.swp5.vlans.ADD: 50
ntp_client needs no changes.
system needs no changes.
vlans needs:
    vlans.10.CREATE
    10.name.SET: vlan10
    vlans.20.CREATE
    20.name.SET: vlan20
    vlans.30.CREATE
    30.name.SET: vlan30
    vlans.40.CREATE
    40.name.SET: vlan40
    vlans.50.CREATE
    50.name.SET: vlan50
Would you like to apply the changes? y/[n]y
```

Accepting these changes will apply them to the device.

```
Device core
========================================
    bonds.po4.CREATE
    bonds.po4.slaves.ADD: swp4
    bonds.po4.mtu.SET: 9000
    bonds.po4.vlans.ADD: 20
    bonds.po4.vlans.ADD: 50
    bonds.po4.pvid.SET: 25
    interfaces.swp1.vlans.ADD: 10
    interfaces.swp1.vlans.ADD: 20
    interfaces.swp1.vlans.ADD: 30
    interfaces.swp1.vlans.ADD: 40
    interfaces.swp1.vlans.ADD: 50
    interfaces.swp2.vlans.ADD: 10
    interfaces.swp2.vlans.ADD: 20
    interfaces.swp2.vlans.ADD: 30
```

(continues on next page)

(continued from previous page)

```
    interfaces.swp2.vlans.ADD: 40
    interfaces.swp2.vlans.ADD: 50
    interfaces.swp3.vlans.ADD: 10
    interfaces.swp3.vlans.ADD: 20
    interfaces.swp3.vlans.ADD: 30
    interfaces.swp3.vlans.ADD: 40
    interfaces.swp3.vlans.ADD: 50
    interfaces.swp5.vlans.ADD: 10
    interfaces.swp5.vlans.ADD: 20
    interfaces.swp5.vlans.ADD: 30
    interfaces.swp5.vlans.ADD: 40
    interfaces.swp5.vlans.ADD: 50
    vlans.10.CREATE
    10.name.SET: vlan10
    vlans.20.CREATE
    20.name.SET: vlan20
    vlans.30.CREATE
    30.name.SET: vlan30
    vlans.40.CREATE
    40.name.SET: vlan40
    vlans.50.CREATE
    50.name.SET: vlan50
```

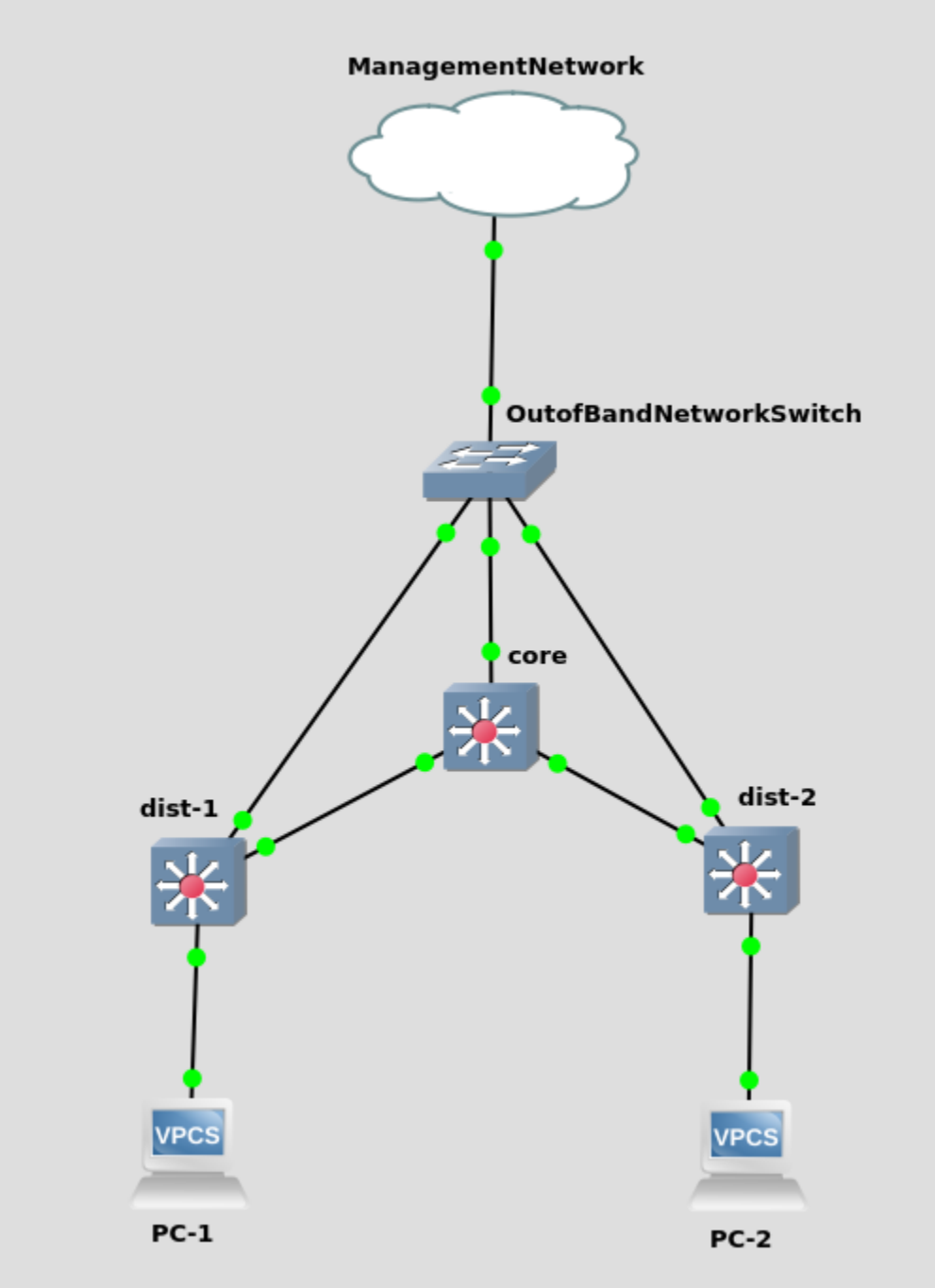
2.3 MergeDB Datasource

2.3.1 Setting Up MergeDB

MergeDB is a project created for Runcible to make declaration of configurations easier, as a result MergeDB is a preferred mechanism for defining Runcible declarations (although you can also use flat YAML or JSON files as well.)

For this example, I will build a simple 3 switch setup inside of GNS3 using Cumulus VX for the operating system.

Here is the topology:



Warning: It is highly recommended to use Runcible with switch fabrics only when you have a dedicated out-of-band management network that won't become inaccessible in the event of misconfiguration.

First, create a directory that contains a file named `mdb.yaml`, which we will leave blank for now. This file indicates to MergeDB that we are creating a MergeDB database in this directory. Next we will create two folders, one for our device declarations, and one for our configuration layers. We will call these folders `devices` and `layers`. In the base of those two directories, create a file called `dir.yaml` in each directory and leave them blank for now. These files inform MergeDB that the .yaml files we will create in these directories are valid MergeDB declarations.

Note: This directory structure is completely arbitrary. MergeDB is designed in a manner that lets you organize your declarations in whatever way makes sense to you.

Next, lets create a few configuration layers that define our switch configurations. Firstly, our switches all have the default U: `cumulus` P: `CumulusLinux!` credentials, so lets create a layer that adds those attributes to the meta object:

```
# examples/mergedb_getting_started/layers/ssh_auth.yaml
---
meta:
  device:
    ssh:
      username: cumulus
      password: CumulusLinux!
```

Next, our switches should all contain the same VLANS in this example, so lets make a layer that defines those:

```
# examples/mergedb_getting_started/layers/vlans.yaml
---
vlans:
  {% for i in [10, 20, 30, 40, 50] %}
  - id: {{ i }}
    name: vlan{{ i }}
  {% endfor %}
```

Note that we are using jinja2 templating to avoid needing to duplicate the vlan definitions.

Now, lets create some layers that define our switch environment. In this example, we want all of the uplinks from the dist1 and dist2 switches to be tagged on all vlans, and the downlinks from the switches to the PCs to be untagged. As a result, we will create two different layers called `core.yaml` and `dist.yaml`.

```
# examples/mergedb_getting_started/layers/core.yaml
{% set vlans = [10, 20, 30, 40, 50] %}
---
interfaces:
  {% for i in range(1,4) %}
  - name: swp{{ i }}
    vlans: {{ vlans }}
  {% endfor %}
  {% for i in range(5,6) %}
  - name: swp{{ i }}
    vlans: {{ vlans }}
  {% endfor %}
```

```
# examples/mergedb_getting_started/layers/dist.yaml
{% set vlans = [10, 20, 30, 40, 50] %}
---
interfaces:
  {% for i in range(1,3) %}
  - name: swp{{ i }}
    pvid: 10
  {% endfor %}
  - name: swp6
    vlans: {{ vlans }}
```

As you can see, our core has all tagged interfaces, whereas the first two ports on the dist switch are untagged, and the

last port is tagged.

Now we need to create the declarations for our switches. In the device directory, create a .yaml for each of the devices:

```
# examples/mergedb_getting_started/switches/core.yaml
---
mergedb:
  inherit:
    - layers/core.yaml
meta:
  device:
    ssh:
      hostname: 192.168.122.166
    default_management_protocol: ssh
    driver: cumulus
system:
  hostname: core
```

```
# examples/mergedb_getting_started/switches/dist1.yaml
---
mergedb:
  inherit:
    - layers/dist.yaml
meta:
  device:
    ssh:
      hostname: 192.168.122.149
    default_management_protocol: ssh
    driver: cumulus
system:
  hostname: dist1
```

```
# examples/mergedb_getting_started/switches/dist2.yaml
---
mergedb:
  inherit:
    - layers/dist.yaml
meta:
  device:
    ssh:
      hostname: 192.168.122.231
    default_management_protocol: ssh
    driver: cumulus
system:
  hostname: dist2
```

At this point, if you were to run MergeDB, you would get blank output because we haven't added anything to the build list. So lets add the rest of our inheritance structure and the build list to the dir.yaml inside the devices directory:

```
# examples/mergedb_getting_started/switches/dir.yaml
---
inherit:
  - layers/ssh_auth.yaml
  - layers/vlans.yaml
build:
  - dist1.yaml
  - dist2.yaml
  - core.yaml
```

Now, run mergedb and inspect the built configs.

```
$ mergedb ../examples/mergedb_getting_started build
core:
  interfaces:
  - name: swp1
    vlans:
    - 10
    - 20
    - 30
    - 40
    - 50
  - name: swp2
    vlans:
    - 10
    - 20
    - 30
    - 40
    - 50
  - name: swp3
    vlans:
    - 10
    - 20
...
```

You can also check the build process for each built declaration to see how MergeDB constructed it at each step.

```
$ mergedb ../examples/mergedb_getting_started detail core.yaml
Initial Layer ../examples/mergedb_getting_started/layers/ssh_auth.yaml:
====================================
meta:
  device:
    ssh:
      password: CumulusLinux!
      username: cumulus

Merge Layer ../examples/mergedb_getting_started/layers/vlans.yaml:
====================================
  meta:
    device:
      ssh:
        password: CumulusLinux!
        username: cumulus
+ vlans:
+ - id: 10
+   name: vlan10
+ - id: 20
+   name: vlan20
+ - id: 30
+   name: vlan30
+ - id: 40
+   name: vlan40
+ - id: 50
+   name: vlan50

Merge Layer ../examples/mergedb_getting_started/layers/core.yaml:
====================================
+ interfaces:
```

(continues on next page)

(continued from previous page)

```
+ - name: swp1
+   vlans:
+   - 10
+   - 20
+   - 30
+   - 40
+   - 50
+ - name: swp2
+   vlans:
+   - 10
...
```

2.3.2 Running Runcible from CLI

Now that we have a database constructed with some switch configuration, we can run Runcible to configure our test environment.

```
[mergedb_getting_started]$ runcible ".*" apply -m .
The following changes will be applied:
Device core:
========================================
    WARNING: need 10.name.SET: vlan10 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e1258d0>
    WARNING: need 20.name.SET: vlan20 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e1258d0>
    WARNING: need 30.name.SET: vlan30 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e1258d0>
vlans needs:
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
interfaces needs:
    interfaces.swp1.vlans.ADD: 10
    interfaces.swp1.vlans.ADD: 20
    interfaces.swp1.vlans.ADD: 30
    interfaces.swp2.vlans.ADD: 10
    interfaces.swp2.vlans.ADD: 20
    interfaces.swp2.vlans.ADD: 30
    interfaces.swp3.vlans.ADD: 10
    interfaces.swp3.vlans.ADD: 20
    interfaces.swp3.vlans.ADD: 30
    interfaces.swp4.vlans.ADD: 10
    interfaces.swp4.vlans.ADD: 20
    interfaces.swp4.vlans.ADD: 30
    interfaces.swp5.vlans.ADD: 10
    interfaces.swp5.vlans.ADD: 20
    interfaces.swp5.vlans.ADD: 30
    interfaces.swp6.vlans.ADD: 10
    interfaces.swp6.vlans.ADD: 20
    interfaces.swp6.vlans.ADD: 30
Device dist2:
========================================
    WARNING: need 10.name.SET: vlan10 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e1259e8>
    WARNING: need 20.name.SET: vlan20 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e1259e8>
```

(continues on next page)

(continued from previous page)

```
    WARNING: need 30.name.SET: vlan30 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e1259e8>
vlans needs:
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
interfaces needs:
    interfaces.swp1.pvid.SET: 10
    interfaces.swp2.pvid.SET: 10
    interfaces.swp6.vlans.ADD: 10
    interfaces.swp6.vlans.ADD: 20
    interfaces.swp6.vlans.ADD: 30
Device dist1:
========================================
    WARNING: need 10.name.SET: vlan10 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e125358>
    WARNING: need 20.name.SET: vlan20 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e125358>
    WARNING: need 30.name.SET: vlan30 is not supported by provider <runcible.
↪providers.cumulus.vlans.CumulusVlansProvider object at 0x7fe24e125358>
vlans needs:
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
interfaces needs:
    interfaces.swp1.pvid.SET: 10
    interfaces.swp2.pvid.SET: 10
    interfaces.swp6.vlans.ADD: 10
    interfaces.swp6.vlans.ADD: 20
    interfaces.swp6.vlans.ADD: 30
Would you like to apply the changes? y/[n]
```

Once you click yes, Runcible will apply all of the changes listed. By default, the naive scheduler will be used which will run Runcible against the devices one after the other in the order specified.

```
Device core
========================================
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
    interfaces.swp1.vlans.ADD: 10
    interfaces.swp1.vlans.ADD: 20
    interfaces.swp1.vlans.ADD: 30
    interfaces.swp2.vlans.ADD: 10
    interfaces.swp2.vlans.ADD: 20
    interfaces.swp2.vlans.ADD: 30
    interfaces.swp3.vlans.ADD: 10
    interfaces.swp3.vlans.ADD: 20
    interfaces.swp3.vlans.ADD: 30
    interfaces.swp4.vlans.ADD: 10
    interfaces.swp4.vlans.ADD: 20
    interfaces.swp4.vlans.ADD: 30
    interfaces.swp5.vlans.ADD: 10
    interfaces.swp5.vlans.ADD: 20
    interfaces.swp5.vlans.ADD: 30
    interfaces.swp6.vlans.ADD: 10
```

(continues on next page)

(continued from previous page)

```
    interfaces.swp6.vlans.ADD: 20
    interfaces.swp6.vlans.ADD: 30
Device dist2
========================================
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
    interfaces.swp1.pvid.SET: 10
    interfaces.swp2.pvid.SET: 10
    interfaces.swp6.vlans.ADD: 10
    interfaces.swp6.vlans.ADD: 20
    interfaces.swp6.vlans.ADD: 30
Device dist1
========================================
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
    interfaces.swp1.pvid.SET: 10
    interfaces.swp2.pvid.SET: 10
    interfaces.swp6.vlans.ADD: 10
    interfaces.swp6.vlans.ADD: 20
    interfaces.swp6.vlans.ADD: 30
```

CHAPTER

THREE

COMPONENTS

3.1 Modules

Modules are the primary data-type in Runcible. They are used to represent both the user's desired state for a given device as well as tracking the current state of a device. They can be generated from JSON, YAML, or a Python Dict (if you are using the *API*.)

In general, Modules have a very simple structure, they have a name and attributes:

```
module_name:
    attribute1: value1
    attribute2: value2
```

An example of the `ntp_client` module as implemented by the *Cumulus* driver:

```
ntp_client:
    interface: eth0
    servers:
    - 0.cumulusnetworks.pool.ntp.org
    - 1.cumulusnetworks.pool.ntp.org
    - 2.cumulusnetworks.pool.ntp.org
    - 3.cumulusnetworks.pool.ntp.org
```

There are also some special modules that behave differently.

3.1.1 Special Modules

Module Array

Module Arrays are a wrapper that allows a number of modules to exist under a parent module. For example:

```
parent_module:
    - key: sub_module_1
      attribute: value
    - key: sub_module_2
      attribute: value
```

The most common use case for Module Arrays are interfaces, bonds, and vlans, for example here is an example of the `bonds` module array as implemented by the *Cumulus* driver:

```
bonds:
- name: po1
  pvid: 1
  slaves:
  - swp1
  - swp2
- name: po2
  pvid: 1
  slaves:
  - swp3
  - swp4
```

Meta Modules

TBD

3.2 Schedulers

Schedulers determine the run-order of Runcible's execution stage (when changes are actually made). They govern many aspects of the "bigger picture" when configuring multiple devices, such as:

- The order in which the devices are configured
- Ensuring that certian groups of devices aren't configured at the same time
- Controlling failure behavior (I.E. failure of a core device results in a stop/rollback of the execution process)
- Rollback and failure behavior

3.3 Needs

3.3.1 What is a need?

A need is a datatype that abstracts state changes. Needs can be simplified into a formatted string:

`<parent_module>.<module>.<attribute>.<command> <value>` Where both the parent module and value are optional. For example:

`vlans.12.ipv4_addr.SET: 10.1.2.3/24` Is a need that represents setting the ipv4_addr of the vlan sub_module for vlan 12 to 10.1.2.3.

`ntp_client.servers.DEL: 0.pool.ntp.org` Is a need that represents removing the value `0.pool.ntp.org` from the list of ntp servers.

Needs are tightly couple to module attribute pathing, so the attribute:

```
system:
  hostname: test
```

Would be modified by a need of `system.hostname.SET: newhostname`.

3.3.2 How do they get generated?

Needs are generated by Runcible automatically when you provide a desired state and call the command `apply`. For instance:

```
[mergedb_getting_started]$ runcible ".*" apply -m .
The following changes will be applied:
Device core:
========================================
vlans needs:
    vlans.10.CREATE
    vlans.20.CREATE
    vlans.30.CREATE
interfaces needs:
    interfaces.swp1.vlans.ADD: 10
    interfaces.swp1.vlans.ADD: 20
    interfaces.swp1.vlans.ADD: 30
    interfaces.swp2.vlans.ADD: 10
    interfaces.swp2.vlans.ADD: 20
    interfaces.swp2.vlans.ADD: 30
    interfaces.swp3.vlans.ADD: 10
    interfaces.swp3.vlans.ADD: 20
    interfaces.swp3.vlans.ADD: 30
```

What occured is that Runcible examined the current state and desired state, and generated a list of needs that are required to bring the two into alignment. Needs provide idempotency in a way that doesn't require Plugin developers to worry about state, as Runcible abstracts the state into Needs.

3.3.3 Needs and ad-hoc commands

You can also apply needs from the command line. By specifying a need string in lieu of special_functions, you can easily make changes without having to create a desired state data source.

For instance:

```
$ runcible -m /home/grayson/PycharmProjects/runcible/examples/cumulus_mclag 'spine1a'␣
↪ntp_client.interface.GET
Device spine1a:
========================================
eth0
```

3.3.4 Need Operations

Need objects support the following operations:

SET

Boolean: set sets the boolean to either True or False List: set replaces the entire list with a new list String: replaces the string with the new string Integer: replaces the integer with the new integer

DELETE

List: must be specified with a value, and only deletes the value specified String: removes the string Integer: removes the integer

CLEAR

List: deletes the whole list

GET

Only used by ad-hoc commands, returns the value of the attribute

ADD

List: adds a new value (or values) to the list

CREATE

Module: Creates a sub-module within a module array

REMOVE

Module: Deletes a sub-module within a module array

CHAPTER

FOUR

DRIVER INDEX

4.1 Cumulus

4.2 Modules

system: Type: Module

Supported Attributes:

Attribute	Type	Allowed Operations	Description	Examples
hostname	string	• SET	This attribute defines the system's hostname, it can be either a short name or fully qualified	• hostname • hostname.domain.com

```
# Example Runcible system module
---
system:
  hostname: hostname
```

interfaces: Type: Module Array

Supported Attributes:

Attribute	Type	Allowed Operations	Description	Examples
name	string	• CREATE • REMOVE	The name of the interface	• swp1 • ge01/0/1
portfast	boolean	• SET	Enables spanning tree portfast on the interface	• False • True
pvid	integer	• SET • DELETE	Vlan ID of the untagged PVID vlan for this interface.	• 10 • 20 • 30
bpduguard	boolean	• SET	Enables BPDU Guard on the interface	• False • True
vlans	list	• SET • ADD • DELETE • CLEAR	A list of the tagged VLANS trunked to this interface. Depending on the switch and interface mode, this may be mutually exclusive with pvid.	• [1, 10, 15, 50] • [20, 30, 40] • [20]
ipv4_addresses	list	• SET • ADD • DELETE • CLEAR	A list of IPV4 addresses of the interface in CIDR notation	• ['192.168.1.2/24', '192.168.1.3/24'] • ['10.2.3.2/24']

```
# Example Runcible interfaces module array
---
interfaces:
- bpduguard: false
  ipv4_addresses:
  - 192.168.1.2/24
  - 192.168.1.3/24
  name: swp1
  portfast: false
  pvid: 10
  vlans:
  - 1
  - 10
  - 15
  - 50
```

vlans: Type: Module Array

Supported Attributes:

Attribute	Type	Allowed Operations	Description	Examples
id	integer	• CREATE • REMOVE	The VLAN id of the VLAN	• 2 • 20 • 4094
name	string	• DELETE • SET	The symbolic name of the vlan	• office_vlan
ipv4_addresses	list	• DELETE • SET • ADD • CLEAR	A list of IPV4 addresses of the interface in CIDR notation	• ['192.168.1.2/24', '192.168.1.3/24'] • ['10.2.3.2/24']
ipv4_gateway	string	• DELETE • SET	The IPV4 default gateway for the interface	• 192.168.1.1 • 10.2.3.1
mtu	integer	• DELETE • SET	The maximum MTU of the interface	• 1500 • 9000

```
# Example Runcible vlans module array
---
vlans:
- id: 2
  ipv4_addresses:
  - 192.168.1.2/24
  - 192.168.1.3/24
  ipv4_gateway: 192.168.1.1
  mtu: 1500
  name: office_vlan
```

ntp_client: Type: Module

Supported Attributes:

Attribute	Type	Allowed Operations	Description	Examples
interface	string	• DELETE • SET	Interface used for NTP	• swp1 • eth0
servers	list	• DELETE • SET • ADD • CLEAR	A list of servers hostname or IP addresses used for NTP	• ['0.pool.ntp.org', '1.pool.ntp.org', '2.pool.ntp.org']

```
# Example Runcible ntp_client module
---
ntp_client:
  interface: swp1
  servers:
  - 0.pool.ntp.org
  - 1.pool.ntp.org
  - 2.pool.ntp.org
```

cumulus_mclag: Type: Module

Supported Attributes:

Attribute	Type	Allowed Operations	Description	Examples
interface_ip	string	• SET • DELETE	The IP address that will be assigned to the peerlink bond for state syncing	• 169.254.2.1/30
peerlink_interfaces	list	• SET • ADD • DELETE • CLEAR	The interfaces used to create the peerlink bond	• ['swp47', 'swp48']
system_mac_address	string	• SET • DELETE	The emulated mac address of the CLAG cluster	• 44:38:39:ff:01:01
peer_ip	string	• SET • DELETE	The CLAG peers ip address	• 169.254.2.2
priority	integer	• SET • DELETE	The priority of this device in the CLAG cluster	• 1000 • 0
backup_ip	string	• SET • DELETE	The backup ip used for the CLAG cluster if the peer_ip is unreachable	• 192.168.122.18
clagd_args	list	• SET • ADD • DELETE • CLEAR	Additional arguments passed to the CLAG daemon on startup (such as –vm to enable CLAG in a virtual machine	• –vm

```
# Example Runcible cumulus_mclag module
---
cumulus_mclag:
  backup_ip: 192.168.122.18
  clagd_args: --vm
  interface_ip: 169.254.2.1/30
  peer_ip: 169.254.2.2
```

(continues on next page)

(continued from previous page)

```
  peerlink_interfaces:
  - swp47
  - swp48
  priority: 1000
  system_mac_address: 44:38:39:ff:01:01
```

bonds: Type: Module Array

Supported Attributes:

Attribute	Type	Allowed Operations	Description	Examples
name	string	• CREATE • REMOVE	Name of the bond	• po1 • bond0
slaves	list	• SET • ADD • DELETE • CLEAR	A list of member interfaces that are slaves in the bond	• [‘swp1’, ‘swp2’] • [‘ge0/0/1’, ‘ge0/0/2’]
mtu	integer	• SET • DELETE	Sets the maximum allowed MTU for the bond	• 1500 • 9000
ipv4_addresses	list	• SET • ADD • DELETE • CLEAR	A list of IPV4 addresses of the bond in CIDR notation	• [‘192.168.1.2/24’, ‘192.168.1.3/24’] • [‘10.2.3.2/24’]
ipv4_gateway	string	• SET • DELETE	The IPV4 default gateway for the bond	• 192.168.1.1 • 10.2.3.1
vlans	list	• SET • ADD • DELETE • CLEAR	A lit of tagged vlans on the bond	• [1, 2, 3, 4] • [200, 201, 202]
pvid	integer	• SET • DELETE	The untagged or PVID vlan on the bond	• 1 • 2 • 3 • 4000
clag_id	integer	• SET • DELETE	The CLAG ID of the bond	• 1 • 2

```
# Example Runcible bonds module array
---
```

(continues on next page)

(continued from previous page)

```
bonds:
- clag_id: 1
  ipv4_addresses:
  - 192.168.1.2/24
  - 192.168.1.3/24
  ipv4_gateway: 192.168.1.1
  mtu: 1500
  name: po1
  pvid: 1
  slaves:
  - swp1
  - swp2
  vlans:
  - 1
  - 2
  - 3
  - 4
```

CHAPTER

FIVE

API

5.1 API Device Example

Create a dict representing the desired state of your device (In this case, a switch using the Cumulus provider)

```
from runcible.api import Device, CBType
conf = {
    "meta": {
        "device": {
            "ssh": {
                "hostname": "192.168.122.41",
                "username": "cumulus",
                "password": "CumulusLinux!",
            },
            "default_management_protocol": "ssh",
            "driver": "cumulus"
        }
    },
    "system": {
        "hostname": "switch-test"
    },
    "interfaces": [
        {"name": "swp1", "pvid": 22, "bpduguard": False},
        {"name": "swp2", "pvid": 23, "bpduguard": True, "portfast": True},
    ],
    "vlans": [
        {"id": 22},
        {"id": 23}
    ]
}
d = Device("switch1", conf)
```

The Device class provides two main methods, .plan() and .execute().

Plan generates a list of needs and displays them to the user as callbacks, execute applies those changes. Each can be re-run without re-creating the instance as many times as desired, but you must run them in the order .plan() -> .execute().

```
>>> d.plan()
{'has_fatal': False, 'has_errors': False, 'log': [{'message': 'system needs no␣
↪changes.', 'callback_type': 'INFO'}, {'message': 'interfaces needs:', 'callback_type
↪': 'INFO'}, {'message': 'vlans needs:', 'callback_type': 'INFO'}]}
>>> d.execute()
{'has_fatal': False, 'has_errors': False, 'log': [{'message': 'interfaces.swp1.pvid.
↪SET: 22', 'callback_type': 'SUCCESS'}, {'message': 'interfaces.swp2.pvid.SET: 23',
↪'callback_type': 'SUCCESS'}, {'message': 'interfaces.swp2.bpduguard.SET: True',
↪'callback_type': 'SUCCESS'}, {'message': 'interfaces.swp2.portfast.SET: True',
↪'callback_type': 'SUCCESS'}, {'message': 'vlans.module.CREATE: 20', 'callback_type
↪': 'SUCCESS'}, {'message': 'vlans.module.CREATE: 4044', 'callback_type': 'SUCCESS'}
↪]}
```

(continues on next page)

(continued from previous page)

```
>>> d.plan()
{'has_fatal': False, 'has_errors': False, 'log': [{'message': 'system needs no␣
↪changes.', 'callback_type': 'INFO'}, {'message': 'interfaces needs no changes.',
↪'callback_type': 'INFO'}, {'message': 'vlans needs no changes.', 'callback_type':
↪'INFO'}]}
>>> d.execute()
{'has_fatal': False, 'has_errors': False, 'log': [{'message': 'No changes needed',
↪'callback_type': 'SUCCESS'}]}
```

If you don't want JSON callbacks, you can also change the callback method to terminal to show what the CLI will look like.

```
>>> d = Device("switch1", conf, callback_method=CBMethod.TERMINAL)
>>> d.plan()
system needs no changes.
interfaces needs:
interfaces.swp1.pvid.SET: 22
interfaces.swp2.pvid.SET: 23
interfaces.swp2.bpduguard.SET: True
interfaces.swp2.portfast.SET: True
vlans needs:
vlans.module.CREATE: 20
vlans.module.CREATE: 4044
>>> d.execute()
interfaces.swp1.pvid.SET: 22
interfaces.swp2.pvid.SET: 23
interfaces.swp2.bpduguard.SET: True
interfaces.swp2.portfast.SET: True
vlans.module.CREATE: 20
vlans.module.CREATE: 4044
```

CHAPTER

SIX

CONTRIBUTION GUIDE

6.1 Current State of Runcible

Runcible is very early in it's devleopment; however our goal is to provide a full layer of functionality at a time, which means it might be useful for certian tasks long before development is complete. It's also a great time to get involved, as your input will shape the direction and structure of the project.

During the early stages of development, the most up to date information should be found in the *Getting Started* guide.

6.2 Roadmap

6.2.1 Phase 1

- Device level API Classes (done)
- Feature complete Providers for Cumulus Switches (80% done)
- Feature complete Providers for Ubiquity Switches
- Builtin common modules for switches (done)
- Basic Naive Executor/Schedulers for running against multiple devices (done)
- YAML Configuration Layering and Inheritance (done)
- YAML directory structure (done)

6.2.2 Phase 2

- Network level API Classes (For performing actions against multiple devices)
- Topology Aware Executor/Schedulers

6.2.3 Phase 3

- Self Testing/Configuration Rollback functionality

CHAPTER

SEVEN

INDICES AND TABLES

- search