REAWNZ

NETWORK AUTOMATION AT REANNZ

AARON MURRIHY
aaron.murrihy@reannz.co.nz

REANNZ

WHY AUTOMATE?

- Save time; make operations scalable
- Improve reliability of the network
- Improve understandability of the network
- Makes documentation and monitoring easier to maintain

STEPS TO AUTOMATING A PROBLEM

1. Identify the problem
2. Decide if it’s worth automating
3. Write tool to solve problem
   - Generate configuration
   - Apply configuration to the network
4. Socialise your new tool

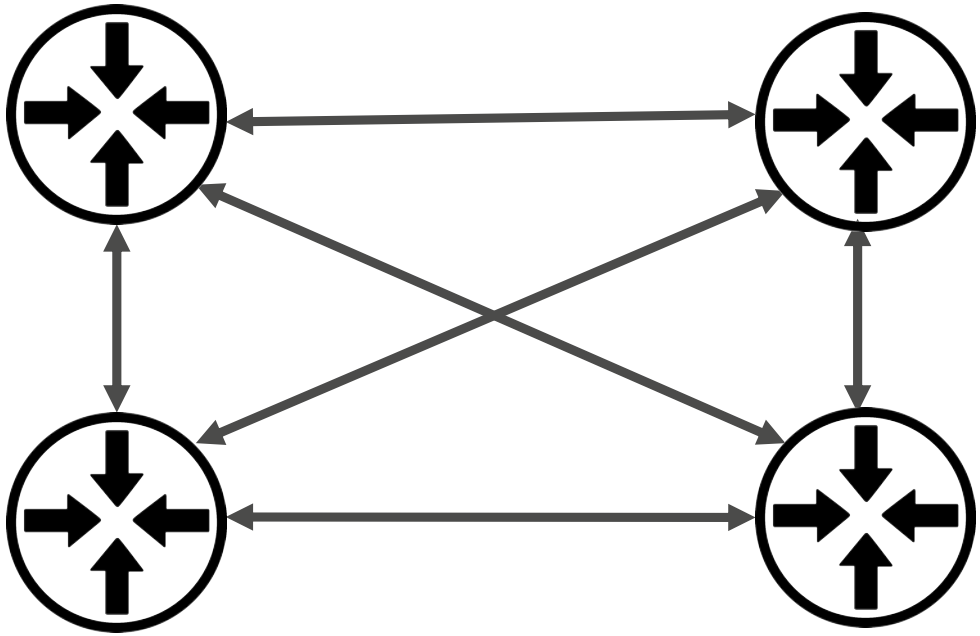
IDENTIFY THE PROBLEM

IDENTIFY THE PROBLEM

For historical reasons REANNZ uses LDP-signalled VPLSes.

IDENTIFY THE PROBLEM

This means every VPLS instance on every host must be configured with neighbours to be fully meshed with every other host with the same VPLS ID.



IDENTIFY THE PROBLEM

HOST1:
set routing-instances vpls-5000 protocols vpls neighbor 172.24.2.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.3.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.4.1

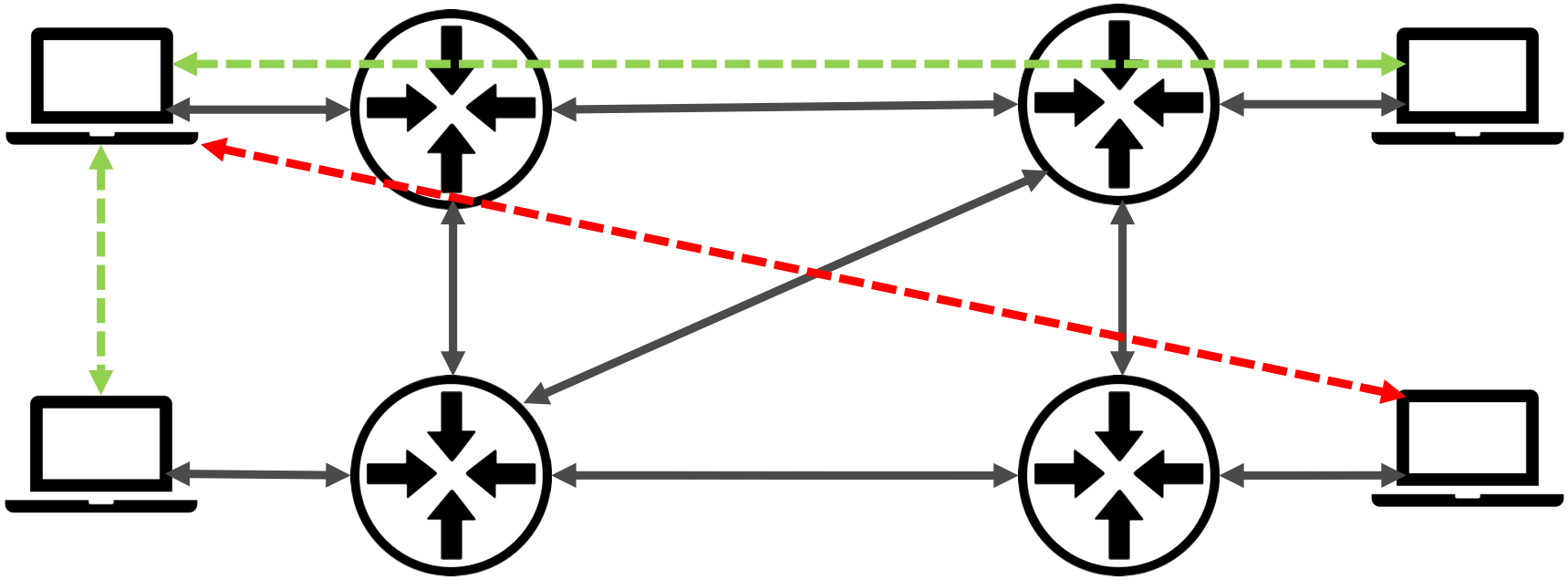
HOST2:
set routing-instances vpls-5000 protocols vpls neighbor 172.24.1.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.3.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.4.1

HOST3:
set routing-instances vpls-5000 protocols vpls neighbor 172.24.1.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.2.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.4.1

HOST4:
set routing-instances vpls-5000 protocols vpls neighbor 172.24.1.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.2.1
set routing-instances vpls-5000 protocols vpls neighbor 172.24.3.1

IDENTIFY THE PROBLEM

If a VPLS is not fully meshed. Some parts of the network will be unable to talk to other parts of the network, potentially only unidirectionally. Awful to diagnose.



DECIDE IF A PROBLEM IS WORTH AUTOMATING

IS IT WORTH AUTOMATING?

- How much time does it take to do it by hand?
- What are the consequences of it being wrong?
- How easy is it to tell if it’s configured incorrectly?
- Are there multiple ways to configure the service?
- Will homogenous configurations make the network easier to understand?
- What do other members of your team think?

IS IT WORTH AUTOMATING?

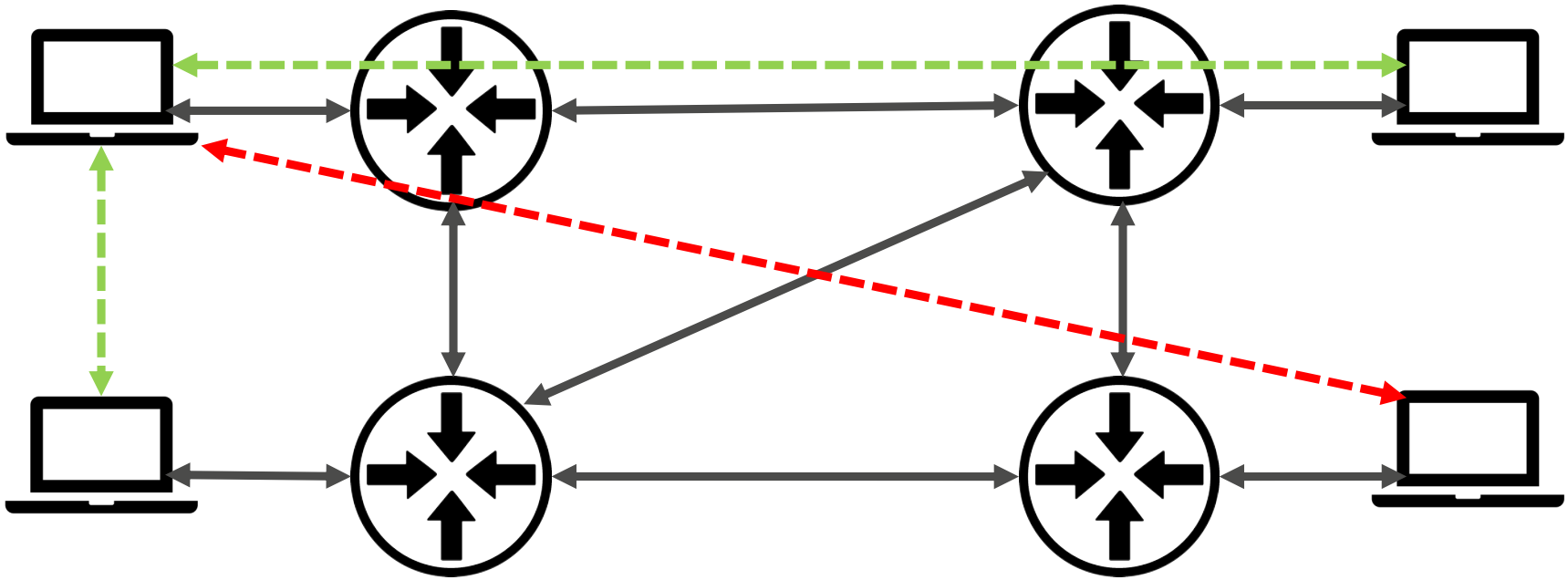
On the REANNZ network today:

- Number of VPLSes = 244
- Number of neighbours configured = 4248
- Our largest VPLS has 31 hosts and 870 neighbour statements

Imagine someone asking you to make sure everything is fully meshed…

Worth Automating

IS IT WORTH AUTOMATING?



Automate, Automate, Automate

IS IT WORTH AUTOMATING?

VPLSes can be configured to

- Tear off vlan headers on ingress and insert new ones on egress
- Pass packets leaving their vlan header intact
  - Vlan IDs much match at all ends
- Translate/normalise vlan IDs
- Pass Q-in-Q packets
  - Create multiple mac-learning domains
- Any number of other ways I haven't mentioned or don't know about.

Please, just do it already!

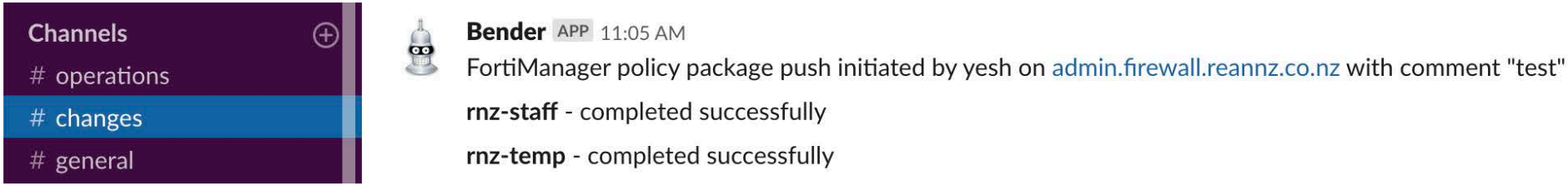
QUICK INTRODUCTION TO THE REANNZ OPERATIONAL AUTOMATION ENVIRONMENT

MEET BENDER

- Monolithic repo
  - Gives easy access to all internal tools and libraries
  - Branch, write patch, code review, merge
  - No hidden ”pet” projects
  - Easy(er) to refactor code and change APIs
- Miniconda Python environment
  - “make install” to build
  - Installs external packages without affecting the main system
  - Environment is reliably built on any host that needs it

MEET BENDER

- Implements Python libraries for accessing the APIs of our “sources of truth” and other services
  - NetDB
  - phpIPAM
  - FreshDesk
  - Google Apps
  - PagerDuty
  - Slack



MEET BENDER

- Implements wrappers around NETCONF for Junipers
  - Pushes configuration
    - Config lock on every host being configured
    - commit check
    - show | compare
    - rollback
    - TODO: commit confirmed
  - Grabs running state
    - BGP/OSPF/LLDP Neighbours
    - Interfaces
    - Hardware
    - Alarms
    - Etc.
  - Upgrades devices
    - Compares pre and post state
    - Cleans up storage and takes a backup disk image
    - Initiates firmware upgrade
    - Notifies an engineer of any issues

MEET BENDER

- Generates network configuration
  - Standardised configurations and user accounts
  - Member edge firewall ACLs
  - Member edge route policy
  - Standardised circuit types
  - Managed Firewall route policy

NETWORK AUTOMATION AT REANNZ

AARON MURRIHY
aaron.murrihy@reannz.co.nz

REANNZ

WRITE THE TOOL

DESIGN AN INTERFACE

- Gives you an outcome to work towards
- Gives you an idea of what options to cater for

```
aaron@nms-wlg:~$ vpls --help
Usage: vpls [OPTIONS] COMMAND [ARGS]...

Options:
  -v, --verbose  Verbose output, -vv for debug info
  --help         Show this message and exit.

Commands:
  mesh  Ensures all VPLSes on the network are...
```

```
aaron@nms-wlg:~$ vpls mesh --help
Usage: vpls mesh [OPTIONS]

  Ensures all VPLSes on the network are meshed. If not, generates
  configuration to fully mesh them.

  Example usage:
      vpls mesh -w

Options:
  -w, --write  Write config output to per-host file
  --help       Show this message and exit.
```

DESIGN AN INTERFACE

- Also makes it easier to integrate with your existing tooling environment

```
aaron@nms-wlg:~$ vpls --help
Usage: vpls [OPTIONS] COMMAND [ARGS]...

Options:
  -v, --verbose  Verbose output, -vv for debug info
  --help         Show this message and exit.

Commands:
  add       Creates a new VPLS on the network.
  allocate  Allocates the next available VPLS in IPAM...
  mesh      Ensures all VPLSes on the network are...
  remove    Delete ports from a VPLS.
  wipe      Remove all instances of a VPLS from the...
```

DESIGN AN INTERFACE

- Use click to implement CLI interfaces in Python

```
@click.group()
@click.option('-v', '--verbose', default=0, count=True,
              help='Verbose output, -vv for debug info')
def main(verbose):
    bender.logging.configure(level=bender.logging.verbose_to_level(verbose))

@main.command('mesh')
@click.option('-w', '--write', is_flag=True, help='Write config output to per-host file')
def mesh(write):
    """
    \b
    Ensures all VPLSes on the network are meshed. If not, generates
    configuration to fully mesh them.
    \b
    Example usage:
        vpls mesh -w
    """
```

GATHER STATE

Gather current (configured) VPLS state

```
def get_vplses_participants():
    """
    Looks through the configs of all core MPLS hosts and builds up a list of
    all VPLSes and all hosts that participate in that VPLS.

    VPLS current state is grabbed by parsing rancid backed-up set config.

    Returns dict structure of the form:
    {
        5000: ['host1', 'host2', 'host3'],
        5001: ['host2', 'host8']
    }
    """
```

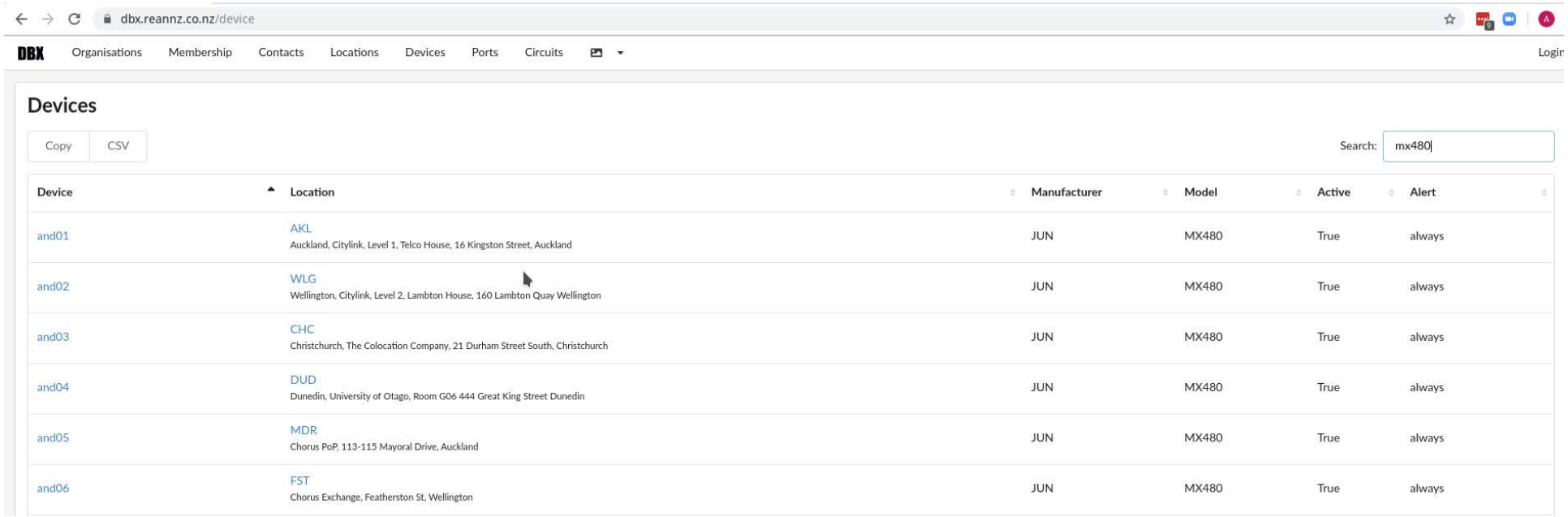
GATHER STATE

Get the list of core MPLS hosts

```
# Grab the list of Juniper core MPLS devices for NetDB
netdb = bender.api.netdb.NetDB()
hosts = netdb.get_hosts_names(vendor='juniper', device_class='core')
```

GATHER STATE

We have a database, but no problem with a CSV or YAML file



Device	Location	Manufacturer	Model	Active	Alert
and01	AKL Auckland, Citylink, Level 1, Telco House, 16 Kingston Street, Auckland	JUN	MX480	True	always
and02	WLG Wellington, Citylink, Level 2, Lambton House, 160 Lambton Quay Wellington	JUN	MX480	True	always
and03	CHC Christchurch, The Colocation Company, 21 Durham Street South, Christchurch	JUN	MX480	True	always
and04	DUD Dunedin, University of Otago, Room G06 444 Great King Street Dunedin	JUN	MX480	True	always
and05	MDR Chorus PoP, 113-115 Mayoral Drive, Auckland	JUN	MX480	True	always
and06	FST Chorus Exchange, Featherston St, Wellington	JUN	MX480	True	always

GATHER STATE

For each host

- Iterate config file line-by-line
- For each VPLS
  - Add host to VPLS list of participants

```
vplses_participants = {}
for host in hosts:
    with open('%s/%s.set' % (CONFIG_DIR, host), 'r') as f:
        for line in f:
            # Regex parses Juniper "set" config that looks like this:
            # set routing-instances vpls-5000 protocols vpls vpls-id 5000
            vpls_id_regex = 'set routing-instances vpls-([0-9]+) ' \
                            'protocols vpls vpls-id ([0-9]+)'
            m = re.match(vpls_id_regex, line)
            if m:
                vpls_designation = int(m.group(1))
                vpls_id = int(m.group(2))
                # Just do a quick sanity check here
                assert vpls_designation == vpls_id

                # Create the VPLS info data structure if it doesn't
                # already exist
                if vpls_id not in vplses_participants:
                    vplses_participants[vpls_id] = []

                # This host participates in this VPLS. Add it to the list
                # of participants
                vplses_participants[vpls_id].append(host)

return vplses_participants
```

MESH VPLSES

```
# all_configs will be built with the structure:
# {
#   host1: [config_to_add_vpls_neighbour,
#           config_to_delete_vpls_neighbour],
#   host2: [config_to_add_vpls_neighbour]
# }
all_configs = {}
for vpls_id,participants in vplses_participants.items():
    log.info('Meshing vpls-{id_}'.format(id_=vpls_id))

    configs = mesh_vpls(vpls_id, participants)

    # Add any vpls meshing configs for this VPLS into all_configs
    for host,lines in configs.items():
        if host not in all_configs:
            all_configs[host] = []
        all_configs[host] = all_configs[host] + lines
```

MESH VPLS

```
def mesh_vpls(vpls_id, participants):
    """
    For each of the participants in the VPLS, checks the configured VPLS
    neighbours against the list of VPLS participants. If the configured list
    is different (wrong) in any way, generates config to make it right.

    Returns dict of configs to add/remove neighbours keyed by host
    {
        'host1': ['set routing-instances vpls-5000 protocols vpls neighbour 172.24.2.1'],
        'host2': ['set routing-instances vpls-5000 protocols vpls neighbour 172.24.1.1',
                  'delete routing-instances vpls-5000 protocols vpls neighbour 172.24.8.1']
    }
    """
```

MESH VPLS

- For each host
  - Build correct list of neighbours
  - Get the configured list of neighbours

```
# Regex parses Juniper "set" config that looks like this:
# set routing-instances vpls-5000 protocols vpls neighbor 172.24.1.1
neighbour_regex = 'set routing-instances vpls-%s protocols vpls ' \
                  'neighbor (([0-9]{1,3}\.){3}[0-9]{1,3})' % vpls_id

configs = {}
for host in participants:
    # To fully mesh a VPLS, I must have a neighbour statement for every
    # host that participates in the VPLS except for myself.
    neighbours = [p for p in participants if p != host]

    # Use a DNS lookup to get the loopback address for each host.
    # We only care about referencing neighbours by loopback address now.
    neighbour_loopbacks = set(get_loopbacks_from_dns(neighbours))

    configured_neighbour_loopbacks = set()
    with open('%s/%s.set' % (CONFIG_DIR, host), 'r') as f:
        for line in f:
            m = re.match(neighbour_regex, line)
            if m:
                configured_neighbour_loopbacks.add(m.group(1))
```

MESH VPLS

- Looks for mismatches between configured and correct state
  - Generate config to add what doesn't exist, but should
  - Generate config to delete what does exist, but shouldn't

```
    # Now lets look for any differences in the two sets. Any differences
    # are a misconfiguration of the mesh.
    mismatches = neighbour_loopbacks.symmetric_difference(
            configured_neighbour_loopbacks)
    if len(mismatches) > 0:
        configs[host] = []
    for mismatch in mismatches:
        if mismatch in neighbour_loopbacks:
            # This is an unconfigured neighbour. Add it.
            set_neighbour = 'set routing-instances vpls-{id_} protocols ' \
                            'vpls neighbor {addr}'.format(id_=vpls_id,
                                                          addr=mismatch)
            configs[host].append(set_neighbour)
        else:
            # This neighbour no longer participates in the VPLS. Remove it.
            delete_neighbour = 'delete routing-instances vpls-{id_} protocols ' \
                               'vpls neighbor {addr}'.format(id_=vpls_id,
                                                             addr=mismatch)
            configs[host].append(delete_neighbour)

return configs
```

OUTPUT CONFIGURATIONS

- To STDOUT or file

```
# Now return all meshing config to the user
for host,lines in all_configs.items():
    output_config(lines, host, write)
```

OUTPUT CONFIGURATIONS

```
aaron@nms-wlg:~/generated$ vpls mesh -w
aaron@nms-wlg:~/generated$ ls
and03  and06  and17
aaron@nms-wlg:~/generated$ cat *
#################
#     and03     #
#################
delete routing-instances vpls-5004 protocols vpls neighbor 172.24.2.1
#################
#     and06     #
#################
set routing-instances vpls-5004 protocols vpls neighbor 172.24.3.1
#################
#     and17     #
#################
set routing-instances vpls-5004 protocols vpls neighbor 172.24.3.1
```

APPLY CONFIGURATIONS TO THE NETWORK

```
aaron@nms-wlg:~/generated$ apply_config
--- and03: config diff follows ---

[edit routing-instances vpls-5004 protocols vpls]
-       neighbor 172.24.2.1;

--- and06: config diff follows ---

[edit routing-instances vpls-5004 protocols vpls]
        neighbor 172.24.17.1 { ... }
+       neighbor 172.24.3.1;

--- and17: config diff follows ---

[edit routing-instances vpls-5004 protocols vpls]
        neighbor 172.24.6.1 { ... }
+       neighbor 172.24.3.1;

3 hosts will be updated
0 hosts already match proposed config
Commit changes? [y/N] (N in 600s): y
Confirmed yes
All config changes committed.
```

NOT SHOWN HERE

- Write unit tests!
  - Ensure it's generating config as expected
  - Ensure it's parsing VPLS "set" config correctly
  - Ensure it's finding mesh mismatches
- Get peer-reviewed!
  - Ask a network-focused colleague to review functionality
  - Ask a software-focused colleague to review coding style and readability

SOCIALISE THAT ITS IN PRODUCTION

SOCIALISE YOUR NEW TOOL

aaron 2:45 PM

@here Just pulled a new firewall failover utility (based on the standardised firewall architecture) into production.

Untitled

```
aaron@nms-wlg:~$ fwl_failover --help
Usage: fwl_failover [OPTIONS] LOCATION

  Generate config to fail-over firewalls. Optionally, --failback will allow
  you to fail primary back to that firewall. Only a single location can be
```

👍 3

Thanks @yesh and @rprocter for reviewing the code

rprocter 2:49 PM

@aaron nice work

yesh 2:52 PM

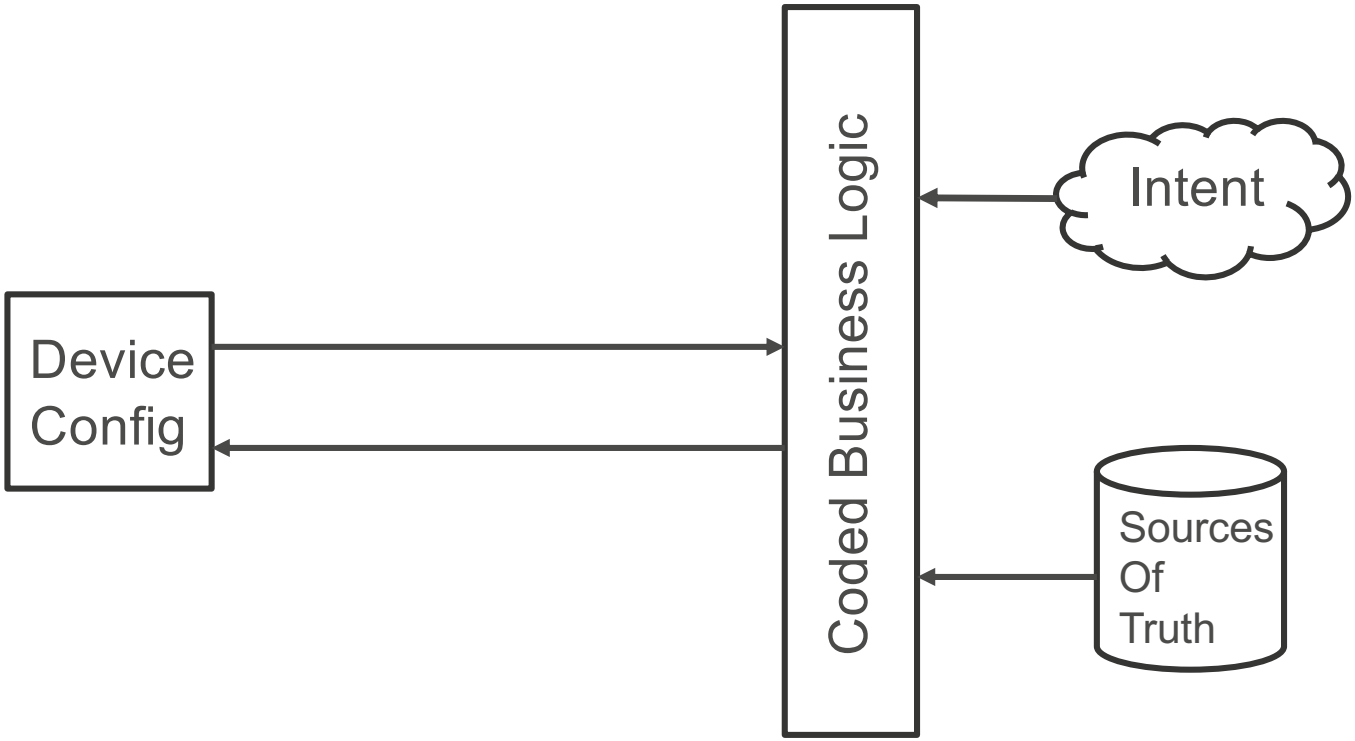
yeah good work

Dan T 2:53 PM

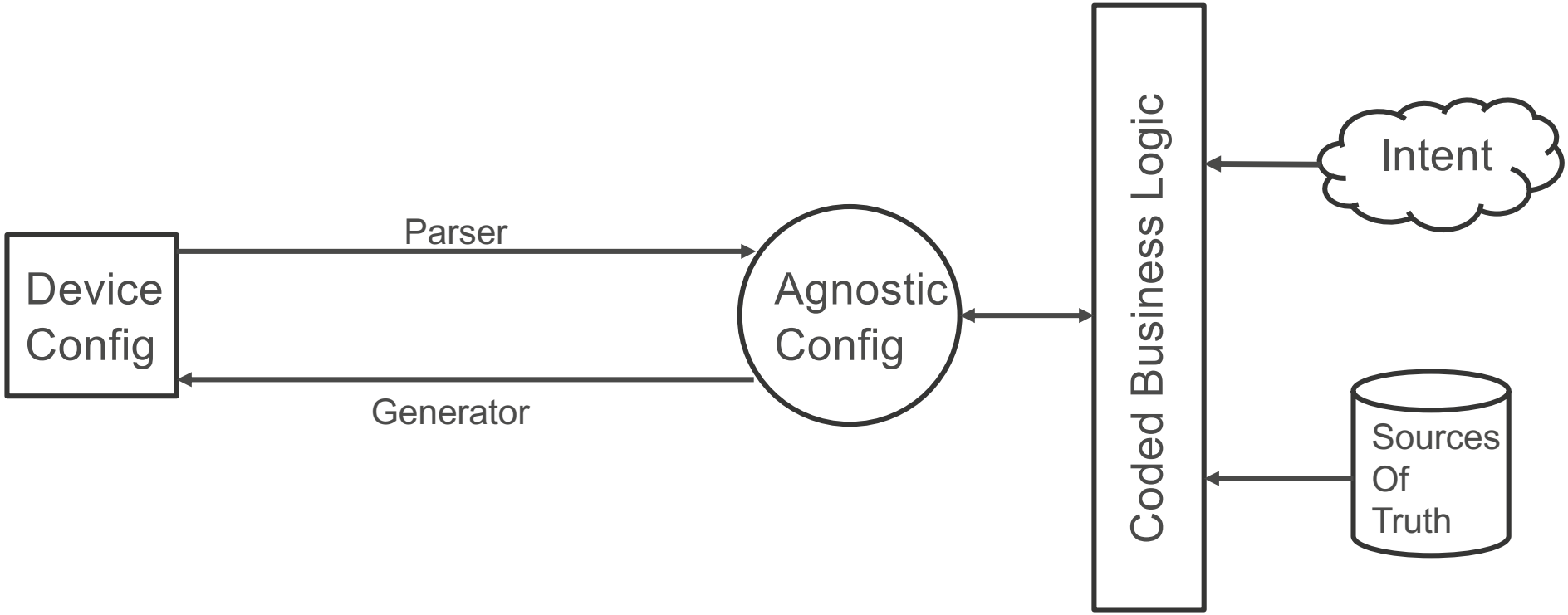
very cool

THE FUTURE

CURRENT AUTOMATION



NEAR FUTURE AUTOMATION



VENDOR AGNOSTIC CONFIG

- Python class tree structure
- Looks similar to a very, very cut-down version of OpenConfig
  - With some key differences
- (In theory) trivial to automate new vendor devices
  - Just implement a new parser and generator
  - All existing “Coded Business Logic” continues to work

PARSER AND GENERATOR

- All parsing/generating implemented in one place
- Unit testing is easy
- Can deal with multiple device configuration types (e.g. EX, MX)
- Can deal with version specific functionality and syntax

GENERATOR

- Diffs pre and post-change agnostic config
- Generates the minimum configuration to implement that change
  - Better co-existence with hand-configured configuration

THE END

QUESTIONS?

AARON MURRIHY
aaron.murrihy@reannz.co.nz
help@reannz.co.nz

REAWNZ