The Single Source of Truth for Network Automation
Most network engineers begin their automation journey by producing some simple reporting software. It is low-risk, has a positive useful impact, and a good introduction to network scripting and the many libraries that support network automation.
Most network engineers begin their automation journey by producing some simple reporting software. It is low-risk, has a positive useful impact, and a good introduction to network scripting and the many libraries that support network automation.
Eventually, tasks which are repetitive, and simple to automate start to look like great candidates to automate. Engineers discover that the great libraries that integrate with software tools can be used to write as well as read configuration, and simple standalone tools are created.
Eventually, tasks which are repetitive, and simple to automate start to look like great candidates to automate. Engineers discover that the great libraries that integrate with software tools can be used to write as well as read configuration, and simple standalone tools are created.
More complex tools are eventually produced. Engineers begin to “configure the network and not the device”, so state becomes a problem (I mean state becomes properly managed). This takes the look and feel of a proper application.
More complex tools are eventually produced. Engineers begin to “configure the network and not the device” so state becomes a problem (I mean state becomes properly managed). This takes the look and feel of a proper application.
The ultimate place to reach is a fully automated and integrated business with a set of processes enforced and delivered by software. “Configure the product, not the network”.

Generally solved by businesses with scale challenges (mass access, hosting) but now a commonplace medium sized ISP/IXP requirement.
This presentation..

• Offer a technical perspective/thoughts on architecture on Greenfield deployment at the ‘automated business’ end of the spectrum

• What motivated this decision?
  • Replication - “as a service” product
  • Efficiency, leaness
  • Service assurance (rapid provisioning, ongoing high availability)
  • Integration with third party peering networks, Euro-IX, PeeringDB
  • Experience in this field, and frustration with traditional model
  • Chance to align business and technical process from the start - in our “DNA”
This presentation .. (2)

- Data model
  - Why and how to build a data model to support integrated automated business
- Software architecture for network centric businesses
  - Abstraction
    - APIs & API integration with customers
- Software testing
- Useful third party tools
What I mean, “data model”?

- A description of the things your business needs to ‘know’ in order to operate

- Start with the steady state of the business
What I mean, “data model”?

- Then model the interactions between those “things”
Why to care from an engineering point of view?

- People
- Organisations
- Products
- Configured Services
- Infrastructure Elements

Monitoring

Quotes

Configurations
Where does/shall data live?

Finance

Sales

Engineering

Support
Where does/shall data live?

Fundamentally it is *fine* for data to “live” in different tools and databases.
Where does/shall data live?

- Finance: Xero, Excel
- Sales: Search Engine, Inc.
- Engineering: MySQL, influxdb
- Support: Confluence

Search Engine (Netherlands) B.V.

Search Engine
Where does/shall data live?

Finance

Search Engine (Netherlands) B.V.

Sales

Search Engine, Inc.

We just deal with Fred

Search Engine

Support
Fundamentally it is *not fine* for more than one data place to be authoritative for any single type of record. The other databases must refer to the key (id) of a single authoritative source.
Where does/shall data live?

Fundamentally it is not fine for more than one data place to be authoritative for any single type of record.

The other databases must refer to the key (id) of a single authoritative source.

We will talk about how to configure and enforce that shortly.
Rules of Engagement

• Store any item of data **ONCE**
  • Easy to ensure that it is correct
  • “Third normal form”

• Give every record a **unique** ID which has nothing to do with the record
  • (ASN is not to be used as ID!)

• Decide **where** it will be authoritative

• Requires buy in and planning from across the business.
Separate your customer/infrastructure data

Port

port_id

customer_id

bridge_id

port_name

Service

service_id

port_id

service_item1

product_id

Ensure infrastructure centric and customer-centric data is not in the same table. This will make your data substantially easier to maintain in terms of portability.
Database Fashions

- Document store -vs- RDBMS
  - Developers like document stores because they are very extensible and less strict
    - “Storage” cost reduced, so now we can be lazy
  - Strict is a benefit / feature
Common Data Stores in Engineering

- SQL - Truths about users, ports, services, ‘state’, e.g. MySQL

- Time Series - e.g. Port utilisation, light level, error count, e.g. InfluxDB

- Third Party - Someone else’s sorted data, e.g. CRM, e.g. EuroIX/PeeringDB
General Architecture

A single API layer makes it simpler to develop and monitor your platform, and easier to make changes to back end services as time goes by.
General Architecture

Client Utilities (scripts, portals, even customers)

API

Worker

Device A

Device A

Device W

It also makes it easier to expose your tools and data to customers. This is a good thing!
General Architecture

Client Utilities (scripts, portals, even customers)

API

Worker

Device A

Device A

Device W

API can export data, no matter about back end storage format in a single format (pick JSON)
Worker, BIRD

Internal SQL

Worker, Arista

InfluxDB
Models <> Templates

• Once you have confidence in your data model you can harness the power of templated configuration

• Once your data model extends across the business you can do that with greater accuracy and devolved control

• e.g. at Asteroid, our sales people can deliver exchange ports directly from the quotation

  • So can customers

• Simultaneous delivery of monitoring from the quotation
Automation fire triangle

Structured Data

Templates

Automation
Templates - Jinja

- Generate any kind of configuration
- Takes variables from your JSON API
- Facilitates programatic methods in configuration strophe
  - Loops
  - Conditionals
Automation - Ansible

```yaml
- name: get route-server settings for this ixp
  uri: url="https://sputnik.asteroidhq.com/export/euroix/{{ exchange_id }}/participants.json" body=yes
  register: exchange_data
  tags:
    - softconfig

- name: install bird IPv4 config
  template: >
    src=templates/collector4.conf.j2
    dest=/etc/bird/bird.conf
    owner=root
    group=root
    mode=0640
  vars:
    exchange_data: {{ exchange_data }},
  register: bird4_changed
  tags:
    - softconfig

- name: restart bird (IPv4)
```
Conditional Logic without script

- name: Collect Prefix information
  url: url='http://46.51.199.18/prefix/{{ item["connection_list"][0]["vlan_list"][0]["ipv4"]["as_macro"] }}' body=yes
  register: prefix
  with_items: "{{ exchange_data["json"]['member_list'] }"
  when: item["connection_list"][0]["vlan_list"][0]["ipv4"]["routeserver"] == true
  tags:
    - softconfig
Advantages

- API layer lightweight
  - Retrieve and update database records
  - Write in a familiar type-safe language (I chose Python)
- Automation layer lightweight
  - Essentially Ansible configuration files
  - Configuration “easier” than coding?
Business Logic

Client Utilities (scripts, portals, even customers)

Worker

Device A

Device A

Device W

API

Ensure your API choices allow you to store, retrieve and process business logic as well as your technical products. Example: Asteroid Campaign logic.
Worker Architecture

Must consider:
- Inter Process Communication
- Job and network state
- Device independent
- Vendor failure behaviour
- Device swap-outs
Inter-Process communication

- Message Queue based?
  - e.g. RabbitMQ
    - 👍 Quite good support in major scripting languages
    - 👍 Fault tolerant, order matters, guaranteed delivery, HA
    - 👎 Extra software to support & Centralised

- Web Services
  - 👍 Same technology stack as central API
  - 👍 Inherently extensible
  - 👍 Decentralised
  - 👎 Extra software to write and more state to manage
Device Independence

• I chose to write a different worker per back end technology

• 👎 A bit of copy/paste code, which is an anti-pattern

• 👍 No stress trying to treat different vendors generically

• 👍 NAPALM allows me to continue with Ansible

• 👍 Can swap out a switch/server architecture for sure
Device Swap-outs

• Using Ansible/NAPALM for switch configuration allows a process for rolling full configuration in event of device failure

• No need for specific software feature, an operational process is ok
Software Testing

- Write the test first

- Red, Green, Refactor mantra
Integration vs Unit Testing

- If you are like me, you will prefer Integration tests
- Write lots, and remember to cover desired exceptions
- Run on your development instance after every change
- “Back to Zero” testing catches unexpected failures
The Joy of Errors

```
tatou:sputnik andy$ pytest
=================================== test session starts =================================
platform darwin -- Python 2.7.10, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: /Users/andy/src/sputnik, infile:
plugins: cov-2.4.0
collected 188 items

tests/test_api_asteroid.py .................................................................
tests/test_api_whitelabel.py ............................................................
tests/test_regular.py .................................................................
tests/test_unit.py .................................................................

================================== FAILURES ===================================

[UnitTests.test_can_load_organisations_quotes]

self = <test_unit.UnitTests testMethod=test_can_load_organisations_quotes>

def test_can_load_organisations_quotes(self):
    ql = QuoteList()
    testquote = ql.list_quotes_organisation(2)[0]
    self.assertTrue(testquote["quote_currency"] == "EUR")
    self.assertTrue(testquote["quote_lineitems"])[0]["quote_lineitem_desc"] == "10Gbit Peering Port, Stellar Internet Exchange"

IndexError: tuple index out of range

tests/test_unit.py:391: IndexError

================================== 1 failed, 187 passed in 5.82 seconds ===========================
```
Ubiquity of JSON for an ISP

- Especially in Peering!
- PeeringDB
- Euro-IX IXF-DB
- Asteroid JSON
Summary

- Single source of truth under the control of all departments
- Which is used to configure services and network
- Accessible to all departments
  - Customer self service
  - Provision from quote
  - “Information in one place and tool”
- Account Managers can do troubleshooting
Any Questions?

Andy Davidson <andy@asteroidhq.com>
www.asteroidhq.com