Network Migration and configuration translation

Antón Bernal – ESNOG Talk 2018/04/10

Juniper Networks Professional Services

A bit about me @ Juniper

Several years supporting customer networks

Roles in Juniper Professional Services

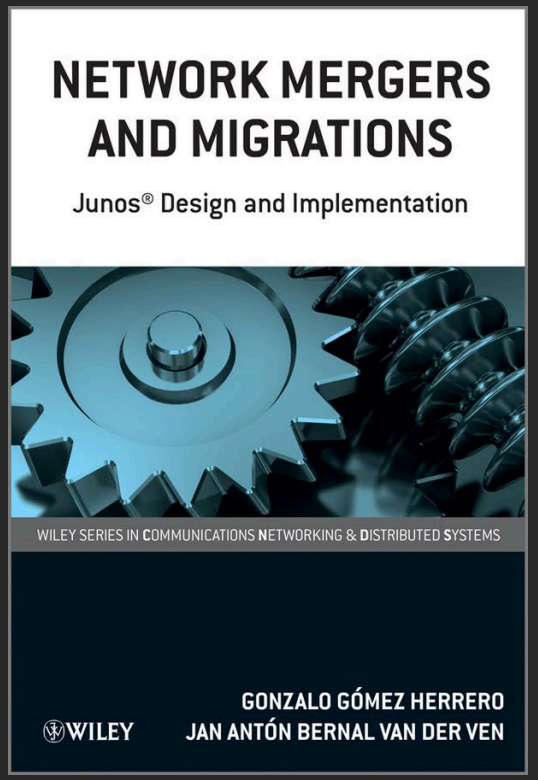
- Roaming Consultant
- Resident Engineer
- Proactive Engineer
- Team lead

Technical

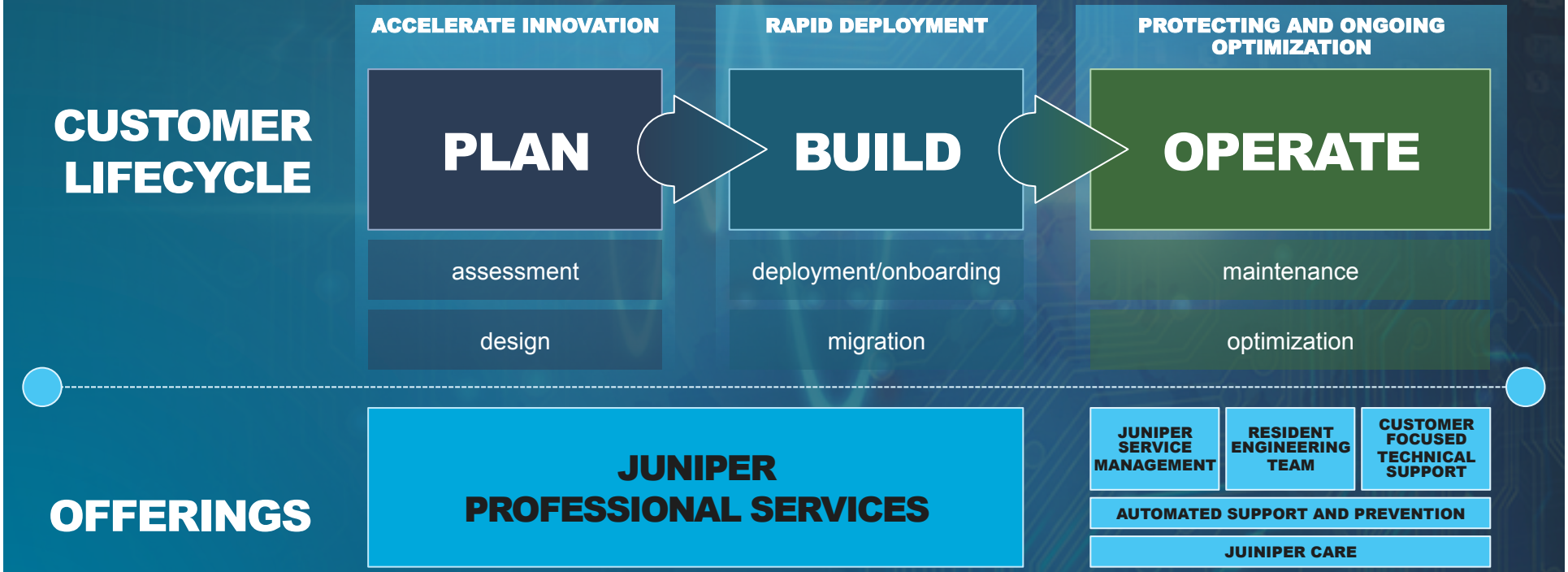
- Manager

Scoping, proposals

- Core&Edge Practice Team Manager



PS Service Offerings Across the Lifecycle



Professional Services / Planning
Network Design and Implementation Plan

- Assessment
- Architecture design
  - High Level Design – what
  - Low level design – how
  - Design Validation – what if…
- Implementation plan
  - Per node implementation
    - Node inspection health checks
    - Target node configuration
      - Jinja template + YML parameters (inventory)

Greenfield
Brownfield

Digression
LLD using Python JINJA2 template engine

Automated configuration generation to enhance quality
repeatability, no fat-fingering of configurations

template.j2

```
{% for intf in con_ext[connection].interfaces %}
   set interfaces {{intf}} description "{{con_ext[connection].interfaces[intf].description}}"
   set interfaces {{intf}} gigether-options 802.3ad {{connection}}
   {% if con_ext[connection].interfaces[intf].wan_mode is defined %}
       set interfaces {{intf}} framing wan-phy
   {% endif %}
{% endfor %}
```

Inventory.yml

```
con_ext:
  ae1:
    interfaces:
      et-0/0/1:
        description “first member link”
        wan_mode ”yes”
```

config.cfg

```
set interfaces et-0/0/1 description ”first member link”
set interfaces et-0/0/1 gigether-options 802.3ad ae1
set interfaces et-0/0/1 framing wan-phy
```

Professional Services / Planning
Network Migration Plan and Execution

Brownfield

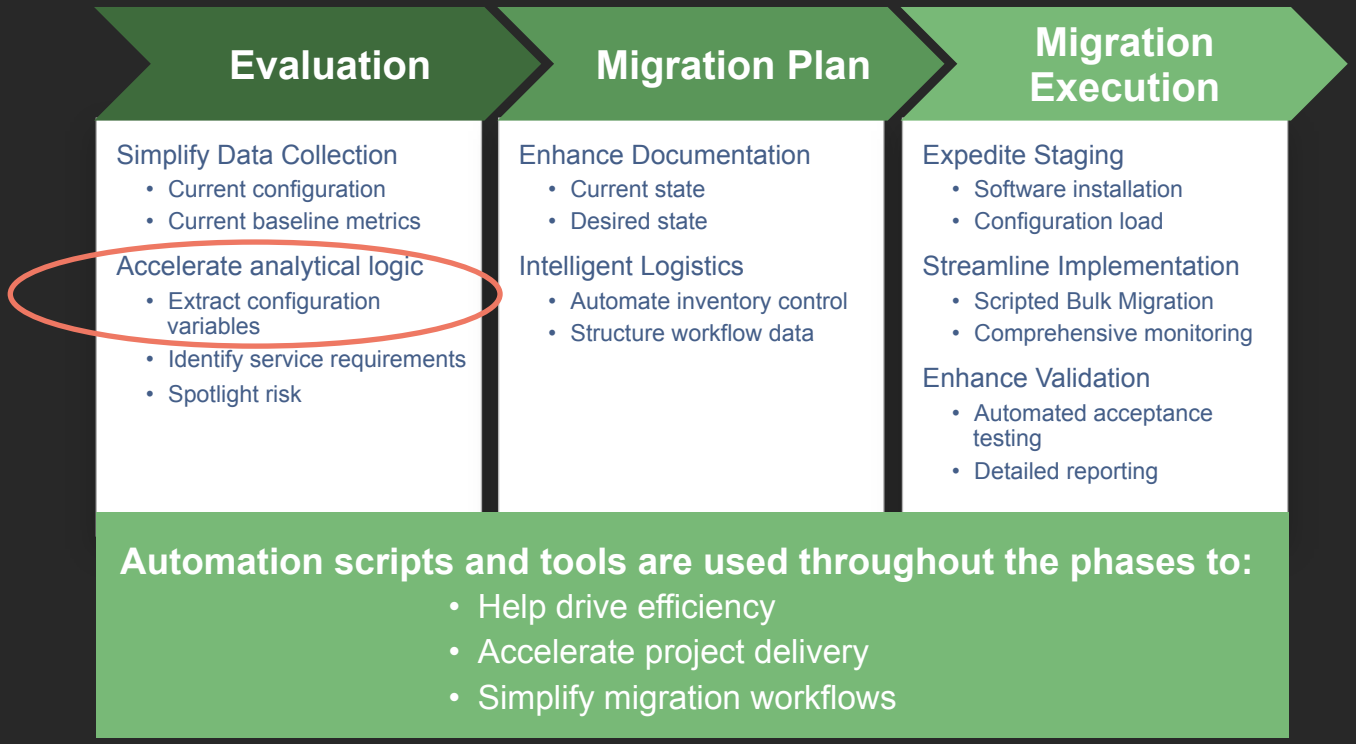
Migration Strategy – approach

- Motivation
- Scale,
- Customer impact
- hot-cut, or incremental transition?
  - old/new architecture compatibility
- Interim scenario

Migration Validation

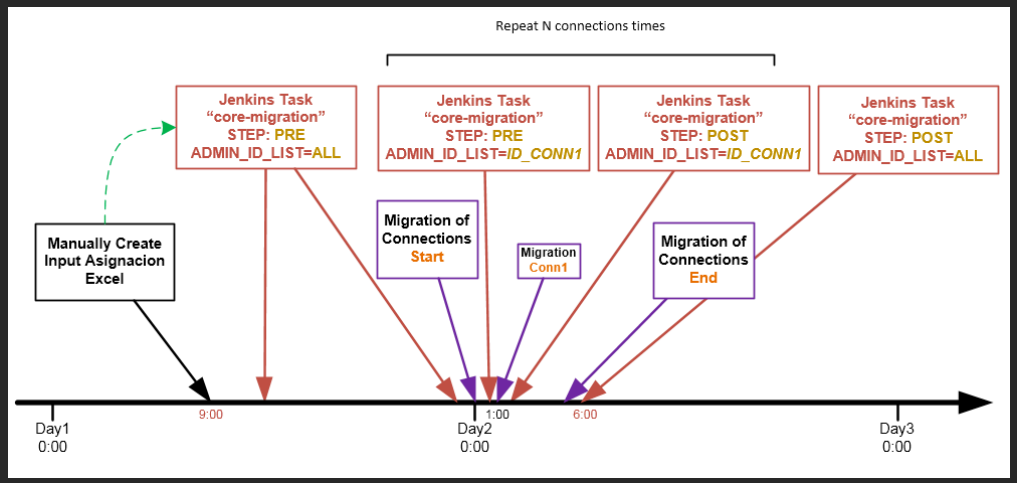
Migration Execution

Network Migration Methodology
Automation benefits



Digression
Network Maintenance activities for service migration

- Network node integration
- Old network – new network gateway
- Internal infrastructure bringup (critical protocols)
- Move of customer connections
  - port mappings
- Pre-/Post- checks
  - Tooling for automation



Ad-hoc implementation and Migration

- Large customers
  - interested in rigorous planning
    - Assessment, HLD, LLD, DVT, First Implementation
  - Inventory-based provisioning system
- Small/Med Enterprises
  - Network-is-master (provisioning ad-hoc)
  - Mostly interested in smooth deployment
  - Implementation and migration Plan and execution

Configuration conversion workflow

- Collect and analyze source device configuration
  - Parse semantics
  - Identify parameters
- Extract inventory in interim format (csv, xml, yaml)
- Transform inventory
  - Source to destination semantics
- Build target configuration

Collect and analyze source device configuration

- What is the volume/complexity of the work?
  - Multiple Software releases (schema variations)
  - Multiple vendors (implementation variations)
  - as-is vs. service redesign
- Source material
  - Device configuration (ascii) text files
  - Design documentation and network drawings (outdated?)
  - Engineering rules (addressing plans, numbering schemes)

Extract Inventory from configuration

Configurations may be multi-line structures

Delimited by braces {}, Exclamation (!), indentation spaces, …

Approaches presented here:

- LALR parsing using a compiler
- Use of regular expression (regexp) language

```
“regular expressions can add, remove, isolate,
and generally fold, spindle, and mutilate all
kinds of text and data” — Mastering RegExp, Jeff Friedl
```

- A “reverse-jinja” experiment

Configuration extraction – compiler

As a compiler – grammar using (Lex/yacc)

$ More C2j.yy

```
%start cisco_object_start
%%
cisco_object_start:
        cisco_object_list;

cisco_object_list:
        |cisco_object_list cisco_interface
        |cisco_object_list vrf
        |cisco_object_list hostname
        |cisco_object_list bgp
        |cisco_object_list vrf_ospf
        |cisco_object_list ospf
          { yyerrok; }
        |cisco_object_list error
```

```
vrf: TOK_IP_VRF TEXTSTRING
    description
    RD
    vpn_id
    import_map
    route_target_export
    route_target_import
    max_routes
     {
      // vrfName, description, rd, vpnId, import, rtE, rtI, maxrte
      addVrf ($2,  $3,  $4,  $5,  $6,  $7,  $8,  $9 );
     }
```

Configuration extraction/translation – compiler

Use an (in-)complete grammar to parse the source co(de)nfiguration

Compile Source configuration to object configuration

```
$ c2j Router.CiscoConfig router.txt -pPortMapFile   -uUnitMapFile

portmap:
    GigabitEthernet,0/2,1/0/1
    GigabitEthernet,0/3,1/1/0
    ATM,1/0,0/1/0
Unitmap:
    GigabitEthernet,0/3.221,
    GigabitEthernet,0/3.227,
    ATM,1/0.12040,12266
```

Usage

1. First run – extract interfaces from the source configuration
2. Edit the interface mapping file
3. Second run – convert configuration using the mapping file
4. Load the global configuration onto any router
5. Merge the compiler-generated configuration with it
6. Go through "Manual Conversion Steps v1.3” procedure

Configuration extraction – Python regexp

```
REGEX_VRF = re.compile(r"""^s*
(?:rd\s+\S+:(?P<id>\d+))?
(?:export\s+map\s+(?P<export>\S+))?
(?:route\-target\s+export\s+(?P<rt_export>\S+))?
(?:route\-target\s+import\s+(?P<rt_import>\S+))?
""", re.VERBOSE)
```

```
"""Cisco translator"""
import argparse
{…}
class Data(object):
"""Data passed between Classes and functions"""
def __init__(self, filename):
    self.parsed_config = self._load_config(filename)
    self.communities_vrf = dict()
{…}
class Vrf(object):
def __init__(self, name):
    self.name = name
    self.id = None

{…}
```

```
def parse_vrf(data, verbose=True):
    """Parse VRF configuration"""
    vrf_list = data.parsed_config.find_objects('^ip vrf')

temp = re.match(utils.REGEX_VRF, aux)
 if temp.group('id'):
    vrf.id = temp.group('id')
```

EXTRACTED YML

```
vrfs:
    NAME:
            name: "NAME"
            id:  12345
```

Configuration extraction / ruminations

Current challenges

- Network engineer with right skill mix (networking, programming)
- Maintenance of the parsing schema (vendor, s/w release)

⇒ Innovation: Create *iterative process* where network engineers use a *simplistic parsing* schema to analyze (extract) configuration inventory into a YML structure for further processing

Configuration extraction – jinja-like?

This is future thinking… (not even ´experimental´)

Reverse the JINJA+YML => CONFIG process

```
{% intfs = con_ext[connection].interfaces[] %}
set interfaces {{ &intfs}} gigether-options 802.3ad {{connection}}
set interfaces {{ &intfs}} description "{{ &intfs.description}}”

 set interfaces et-0/0/1 gigether-options 802.3ad ae1
 set interfaces et-0/0/1 description ”first member link”
 set interfaces et-0/0/1 framing wan-phy
```

add parsing-rules

to

source configuration

extracted.yml

```
con_ext:
  ae1:
     interfaces:
        et-0/0/1:
           description: “first member link”
           connection: 1
```

Configuration extraction – embedded&iterative

Embedding the parsing process in the configuration enables iteration

```
set interfaces {{intf}} gigether-options 802.3ad {{connection}}                    template
set interfaces {{intf}} description "{{con_ext[connection].interfaces[intf].description}}”

set interfaces {{intfs[]=“et-0/0/1”}} description "{{intfs[].description}}”
set interfaces {{intfs[]=“et-0/0/1”}} gigether-options 802.3ad {{connection=“ae1”}}   parsed

set interfaces et-0/0/1 framing wan-phy
                                                                                    unparsed
```

Separate parser engine from networking semantics

Provides for rapid prototyping of templates and YML structures

Use cases

Inventory scan, Configuration auditing, LLD template extraction

Digression – template auto-generation

Facilitate templating by heuristics

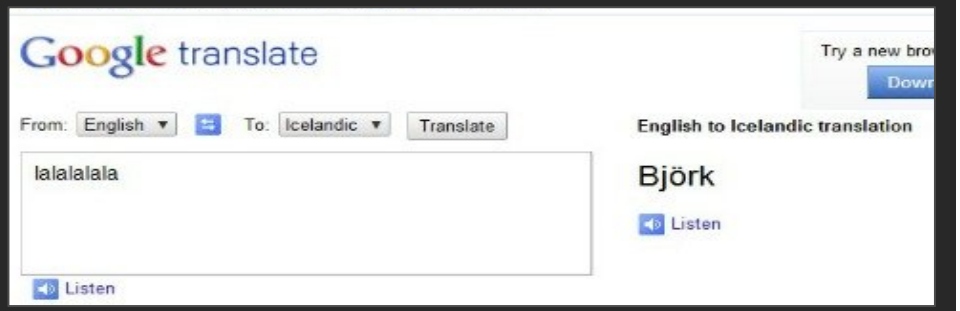
Configuration grammar pretty static accross S/W versions
Use network live configuration data and *infer* what is a parameter
Build a possible template prototype


```
set interfaces et-0/0/1 gigether-options 802.3ad ae1
set interfaces et-0/1/0 gigether-options 802.3ad ae1
set interfaces xe-2/1/1 gigether-options 802.3ad ae2
set interfaces xe-2/0/1 gigether-options 802.3ad ae2
set interfaces xe-0/0/1 gigether-options 802.3ad ae3
=>
set interfaces {{VAR1}} gigether-options 802.3ad {{VAR2}}
```

Configuration translation

More than just ‘direct translation’ - need a new YML file

- Functionality map
  - Some features not directly supported
- Implementation differences
  - Behavior models may not be identical
  - class of service
  - static route next-hops
  - route preference handling
- Diverse ‘default’ behavior
  - No configuration implies adding configuration in target



Proper translation requires to write down challenges identified, document mapping decisions, and reach agreement on final implementation

Build target configuration

Ingredients

- Inventory
- Design documentation templates

Recipe

- Define templates into a provisioning system
- Jinja LLD templates and YAML inventory files

Full or partial configuration excerpts can be constructed in this fashion

Closing remarks and opening feedback

Network Migration is a comprehensive activity
- Planning, Testing, Deploy

Fully automated translation is a bold challenge
- Despite massive automation, fine-tuning is always required
  - Professional Services at your service
- Divide and conquer
  - Analyze, Extract, Transform, Build
- Separate software and networking skills
  - Allow quick adaptations by network engineers

Appreciate your comments and ideas! (anton@juniper.net)