Automating Network Configuration

Brent Chapman
brent@netomata.com

Netomata, Inc.
www.netomata.com

NANOG 49 — 13 June 2010

netomata

Introduction

- Who I am
- What I'm here to talk about

Why automate network configuration?

Because automated networks are

* More reliable
* Easier to maintain
* Easier to scale

For example...

Imagine you're managing a moderately complex web site

* Multiple real and virtual hosts
* Several "environments" (production, testing, development, etc.)
* Separate VLAN for each environment

For example...

What networking devices & services need to be managed?

* Routers
* Switches
* Load Balancers
* Firewalls
* Real-time status monitoring (i.e., Nagios)
* Long-term usage monitoring (i.e., MRTG)

For example...

How to add new virtual host to existing load balancer pool?

* Set up host itself, using Puppet or cfengine or whatever
* Add host to VLAN defs on switches
* Add host to ACLs on routers
* Add host to pool on load balancers
* Add host to NAT and ACLs on firewalls
* Add host to real-time monitoring (i.e., Nagios)
* Add host to usage monitoring (i.e., MRTG)

For example...

What's the problem with doing all that by hand?

* You have to remember how to manage all those very different devices (and you probably don't do it very often)
* It takes a lot of time
* Every step is a chance to make a mistake
* You might get distracted, and never finish

Over time, these small mistakes add up, leading to inconsistent networks that are unreliable and difficult to troubleshoot

Why are automated networks better?

Because automated networks are

* More reliable
* Easier to maintain
* Easier to scale

More reliable

Network device & service configs are highly cross-dependent

* What you do over here is related to what you do over there; think of IP addresses, subnet masks, SNMP community strings, etc.
* Lots that has to be consistent

More reliable

Network devices and services are typically hand-configured

People make mistakes

* Tyops, omissions, accidentally reusing same IP address, extra stuff that shouldn't be there

These inconsistencies cause problems

More reliable

Costs of manually-induced inconsistencies

* Direct costs -- outages, intermittent failures, service degradation, time spent troubleshooting
* Indirect costs of unreliability

More reliable

Indirect costs of unreliability

* Reduces utility of network -- people don't trust it, and treat it as "fragile"
* People become reluctant to make changes, introduce new services, etc.
* Network staff spends more time trouble-shooting, less being proactive

Easier to maintain

- Easier to trouble-shoot
- Easier to test changes
- Easier to deploy changes network-wide
- Faster to make changes
- Easier to roll back changes, if needed

Easier to maintain

Easier to trouble-shoot

* Hand-maintained configs are inconsistent
* Inconsistencies make trouble-shooting harder
    * Does this particular inconsistency matter?
    * Is it relevant to the problem at hand?
    * Is everything here, that should be?
    * Is anything extra here, that shouldn't be?

Easier to maintain

Easier to test changes

* Changes can be prototyped in a lab or testbed
* Automation system can reliably reproduce same changes in production

Easier to maintain

Easier to deploy changes network-wide

* Easy to generate new configs with a change for all devices and services
    * Parameter change
    * Policy change
    * Template change
* Automated tools to deploy new configs

Easier to maintain

Faster to make changes

* Automated tools quickly generate new configs, rather than changing each by hand
* Automated tools quickly deploy new configs, rather than installing each by hand

Easier to maintain

Easier to roll back changes, if needed

* Automated configs can be stored under version control
    * So can the parameters & templates used to generate them
* Automated installation tools can roll back quickly and easily

Easier to scale

Easy to add more systems of existing types (horizontal scaling)

Easy to add new types of systems, or new features/services (vertical scaling)

Easier to scale

Easy to add more systems of existing types (horizontal scaling)

- Generate config for new system
- Generate cross-dependent configs for other systems (monitoring, etc.)
- Deploy configs using automated tools

Easier to scale

Easy to add new types of systems, or new features/services (vertical scaling)

* Add or modify templates for new feature/service/type
* Generate new configs for all affected devices/services (both old & new)
* Deploy configs using automated tools

Levels of Automation

No automation

Probably most of today's networks

Configs are edited manually, in place

Configs are backed up by hand, haphazardly (if at all)

Depends on people to follow procedures, but people are lousy at that.

Read-only automation

Still managing configs by hand

Somebody has set up a tool like RANCID to automatically back up running configs

Maybe it emails them when changes are detected

Only checks systems it knows about

Automated discovery

Still managing configs by hand

Use RANCID to back up configs, and email about changes noticed in configs

Use a tool like Netdisco or OpenNMS to discover devices on the network, and tell RANCID what to back up

Automated config deployment

Configs still maintained by hand

Use something to distribute configs to devices & services

* RANCID in "push" mode
* Custom "expect" scripts
* rsync

Can help to have centralized out-of-band console access, via Conserver or similar

Automated config generation

Use a tool like Netomata Config Generator (NCG) to generate configs for devices & services

Configs still deployed by hand, perhaps backed up by RANCID

Automation of both generation & deployment

Use NCG to generate configs

Use RANCID to distribute configs

* Perhaps include ability to limit distribution, for testing/validation

Automated auditing of running configs

Use RANCID to check running configs against "approved" configs

* Perhaps automatically revert to "approved" config

Use Netdisco or OpenNMS to identify new (unmanaged) devices that appear

Approval wrapped around automation

Develop & test new configs in lab

Request & receive formal approval before limited or full deployment

- Approval process could be web-based
- Tied to deployment system

Would make auditors very happy

Aspects of Automation

1: Knowing your network

Have to know what you're managing

* Devices
* Interfaces
* IP addresses
* Domains & sub-domains
* Services
* ...

Need an "inventory" or "description" of your network

2: Generating configs

Once you know what you're configuring (devices & services), you need to generate the configs for those devices and services

3: Getting configs to & from devices & services

Once you have the configs, you need

* Methods for installing & activating them
* Methods for verifying what's running

4: Change management and control

Once you're reliably getting configs to/from your devices/services, you need a way to manage the changes

* Version control
* Authorization

Netomata Config Generator (NCG)

Netomata Config Generator (NCG)

Generates config files that are

* for various devices and services
* complete, consistent, ready-to-install

Open source (GPLv3), written in Ruby (>1.8.5)

http://www.netomata.com/products/ncg

http://www.netomata.com/docs/programs/ncg

NCG inputs

Two sets of input

* Model describing network (devices, interfaces, interface parameters, etc.)
* Templates for various types of devices (routers, switches, firewalls, etc.) and services (DNS, DHCP, MRTG, etc.)

NCG phases

Works in two phases

* First phase builds up completely populated model of network in memory
* Second phase combines model and templates to generate configs for all devices and services

Generated configs are then installed manually, or via RANCID, ssh, rsync, etc.

NCG user roles

Designed for two distinct user roles:

* Architects define model, create & modify templates, etc.
* Engineers add/modify/delete data in model, then "turn the crank" to produce configs

Running NCG

First, NCG builds in-memory model with all the details about your network:

* Devices
* Interfaces
* VLANs
* IP addresses
* etc.

Then, NCG combines this model with config templates, and generates ready-to-install configs

Creating NCG templates

Start with working configs

* From devices/services in production
* From lab tests or trial setups

Think about the data

* Figure out what data needs to be in tree
* Make sure it's there

Experience setting up NCG templates

Moderately complex network can be done in a couple of weeks

* Several days to set up data and templates for first type of device or service
* Day or two for second type of device/service
* Less for each additional type of device/service

You'll find lots of errors and inconsistencies in existing network as you do this

More about NCG at end of talk, if there's time...

RANCID

RANCID

"Really Awesome New Cisco confIg Differ"

Logs in to device, runs "expect" script, processes result

* Also works for many types of devices besides Cisco
* Default script shows current device config, and emails about differences from last time program was run
* Other scripts available, such as to install new configs

See talk: http://www.shrubbery.net/rancid/NANOG29/

Installing RANCID

Available from

* http://www.shrubbery.net/rancid/
* Also available as installable package (.deb, .rpm, etc.) on many distributions

Installing RANCID

Helpful HowTo's:

* http://homepage.mac.com/duling/halfdozen/RANCID-Howto.html
* http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch1_:_Network_Backups_With_Rancid
* Google "RANCID howto"

FAQ: http://www.shrubbery.net/rancid/FAQ

RANCID config files

`/etc/rancid.conf` — main config file

* Where other files and directories are
* Whether to use CVS or Subversion
* List of "group" names (for different sets of devices)

RANCID config files

`$BASEDIR/$GROUP/router.db` — list of devices for a particular group, and what type of device they are (Cisco, Juniper, etc.)

`$HOME/.cloginrc` — Access credentials (logins, passwords, enable keys, etc.)

* Can include other files, so info can be shared among multiple users
* Keys to the kingdom, so GUARD CAREFULLY!

Running RANCID

Two sets of programs (see "`man rancid`" for list):

* `*rancid` to get configs
* `*login` to access & run commands (called by `*rancid` programs)

Platform	`*rancid`	`*login`
Cisco	`rancid`	`clogin`
Juniper	`jrancid`	`jlogin`
Nortel	`brancid`	`blogin`
Foundry	`francid`	`flogin`
NetScaler	`nsrancid`	`nslogin`
...	...	...

Using RANCID to execute commands

RANCID's `*login` programs can be used to execute commands on devices (via "`-c`" flag)

* Handles details of logging in, becoming superuser, and issuing commands
* Can issue multiple commands, separated by semi-colon

Using RANCID to execute commands

RANCID `*login` programs have limited ability for interactive responses

* If you can predict response needed, you can include in command
* For instance, add a "`\r`" for a carriage return to confirm certain Cisco commands), i.e.:
    * `clogin -c 'reload\r\r' cisco0.lab`

Using RANCID to execute commands

For more complex interactions (including processing output):

* Write an "`expect`" script
    * RANCID distro includes a couple of examples: `cisco-load.exp` and `cisco-reload.exp`

Run it using RANCID `*login`'s "`-s`" flag

* Variables can be passed to it with "`-E`" flags

Integrating NCG and RANCID

Use NCG to generate

* device configs
* RANCID configs with per-device info (`router.db` and `.cloginrc` files)

Then use RANCID to install device configs

Integrating NCG and RANCID

```
brent% clogin -c 'copy flash:config.text flash:config.text.bak\r\r' \
   switch-1.mgmt.ht.netomata.com
switch-1.mgmt.ht.netomata.com
spawn ssh -c 3des -x -l access switch-1.mgmt.ht.netomata.com
Password:

switch-1>enable
Password:
switch-1#
switch-1#term length 0
switch-1#copy flash:config.text flash:config.text.bak
Destination filename [config.text.bak]?
%Warning:There is a file already existing with this name
Do you want to over write? [confirm]
Copy in progress...C
16504 bytes copied in 0.232 secs (71138 bytes/sec)
switch-1#exit
Connection to switch-1.mgmt.ht.netomata.com closed.
```

First, make a backup copy of the current config

* On Cisco, "`startup-config`" is "`flash:config.text`"

Integrating NCG and RANCID

```
brent% cp configs/switch-1.config /var/www
brent% clogin -c 'copy http://10.5.16.11/switch-1.config flash:config.text\r\r'
switch-1.mgmt.ht.netomata.com
switch-1.mgmt.ht.netomata.com
spawn ssh -c 3des -x -l access switch-1.mgmt.ht.netomata.com
Password:

switch-1>enable
Password:
switch-1#
switch-1#term length 0
switch-1#copy http://10.5.16.11/switch-1.config flash:config.text
Destination filename [config.text]?
%Warning:There is a file already existing with this name
Do you want to over write? [confirm]
Loading http://10.5.16.11/switch-1.config !
16594 bytes copied in 0.256 secs (64820 bytes/sec)
switch-1#exit
Connection to switch-1.mgmt.ht.netomata.com closed.
```

Then, copy the new config to the device

Integrating NCG and RANCID

```
brent% clogin -c 'config replace flash:config.text\rY' \
   switch-1.mgmt.ht.netomata.com
switch-1.mgmt.ht.netomata.com
spawn ssh -c 3des -x -l access switch-1.mgmt.ht.netomata.com
Password:
switch-1>enable
Password:
switch-1#
switch-1#term length 0
switch-1#config replace flash:config.text
This will apply all necessary additions and deletions
to replace the current running configuration with the
contents of the specified configuration file, which is
assumed to be a complete configuration, not a partial
configuration. Enter Y if you are sure you want to proceed. ? [no]: Y
The rollback configlet from the last pass is listed below:
...
Rollback aborted after 5 passes
switch-1#exit
Connection to switch-1.mgmt.ht.netomata.com closed.
```

Finally, activate new config on device

About Cisco's "config replace"

Output from 'config replace' is misleading

* seems to say that it failed ("aborted after 5 passes")
* but it seems that it has actually succeeded (if you do "show run" and look for changes)

Safer (but more disruptive) to "reload" device

* Helps if you have something on serial console to capture any complaints about problems in the generated startup-config (aka "flash:config.text")

Reloading Cisco via RANCID

```
brent% clogin -c 'reload\rno\r' \
  switch-1.mgmt.ht.netomata.com
```

You do NOT want to save current (modified) config

* Whole point is to replace it with new startup-config

Command above ASSUMES Cisco will prompt to save config

* If not, use "`... -c 'reload\r' ...`"
* Unfortunately, no one-command-for-all-situations

Reloading Cisco via RANCID

```
brent% clogin -s cisco-reload.exp -Ereload_arg="testing" \
  switch-1.mgmt.ht.netomata.com
```

Better yet, use RANCID "`cisco-reload.exp`" script

* Does the right thing regardless of whether device thinks config needs saving
* Is one of the sample scripts in the RANCID distribution

Requires '`-Ereload_arg="something"`'

* Can specify delay ("`in MMM`" min) or time ("`at HH:MM`")
* Otherwise, specify reason for reload (for log message)

ZipTie

(aka AlterPoint NAI)

ZipTie / AlterPoint NAI

NAI == "NetworkAuthority Inventory"

Discovers network devices, backs up configs, maps network

* Discovers neighbors; find devices by hostname, IP, or MAC
* Can also config many types of devices
* Reports, graphs, etc.

ZipTie / AlterPoint NAI

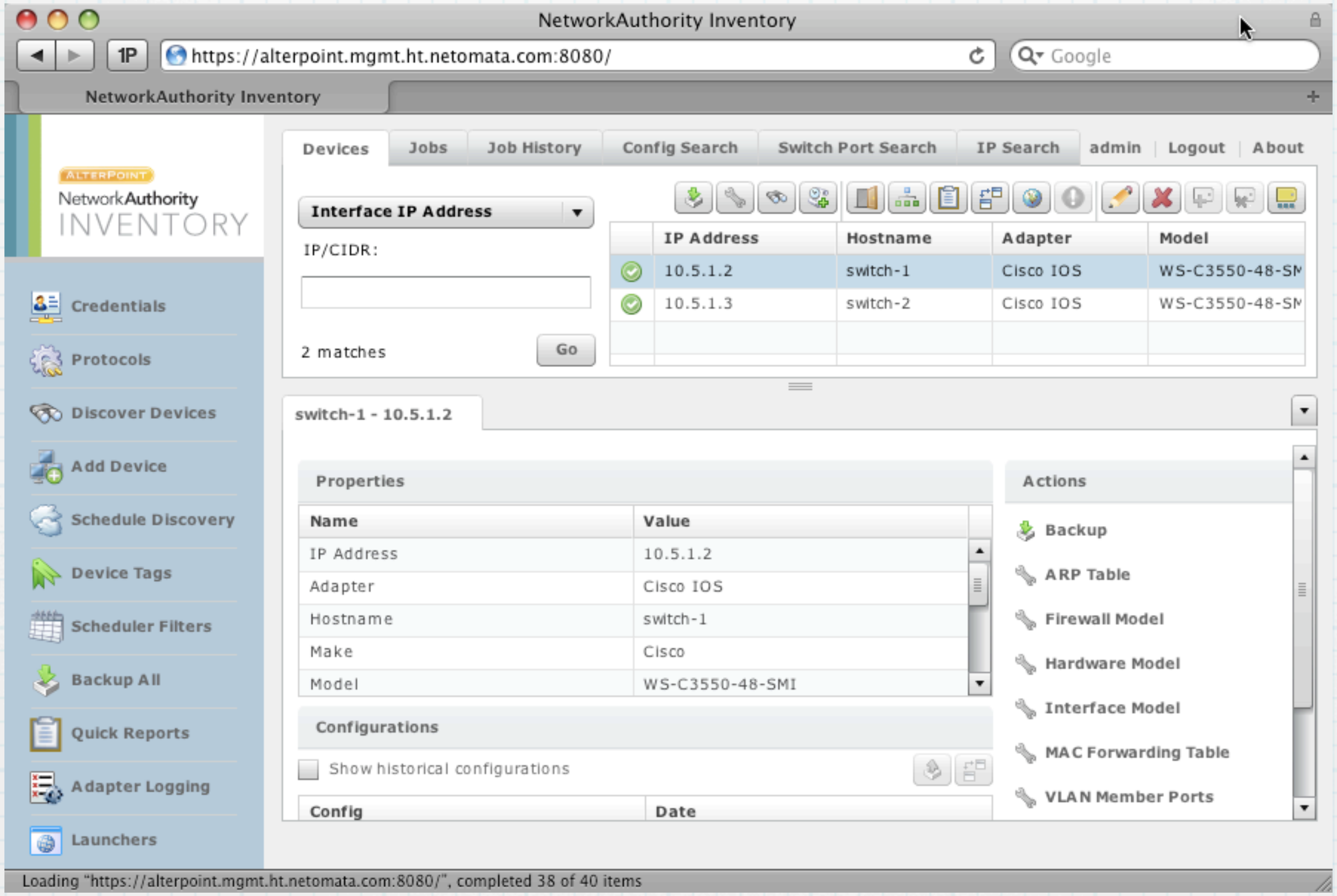
Web-based user interface (AJAX or similar)

Includes framework for plug-in tools

* Many plug-ins contributed by community

Can create schedule for many jobs (discovery, backup, etc.)

ZipTie/NAI



NetworkAuthority Inventory

https://alterpoint.mgmt.ht.netomata.com:8080/

Devices | Jobs | Job History | Config Search | Switch Port Search | IP Search | admin | Logout | About

Credentials
Protocols
Discover Devices
Add Device
Schedule Discovery
Device Tags
Scheduler Filters
Backup All
Quick Reports
Adapter Logging
Launchers

Interface IP Address

IP/CIDR:

2 matches

	IP Address	Hostname	Adapter	Model
✓	10.5.1.2	switch-1	Cisco IOS	WS-C3550-48-SM
✓	10.5.1.3	switch-2	Cisco IOS	WS-C3550-48-SM

switch-1 - 10.5.1.2

Properties

Name	Value
IP Address	10.5.1.2
Adapter	Cisco IOS
Hostname	switch-1
Make	Cisco
Model	WS-C3550-48-SMI

Configurations

Show historical configurations

Config	Date

Actions

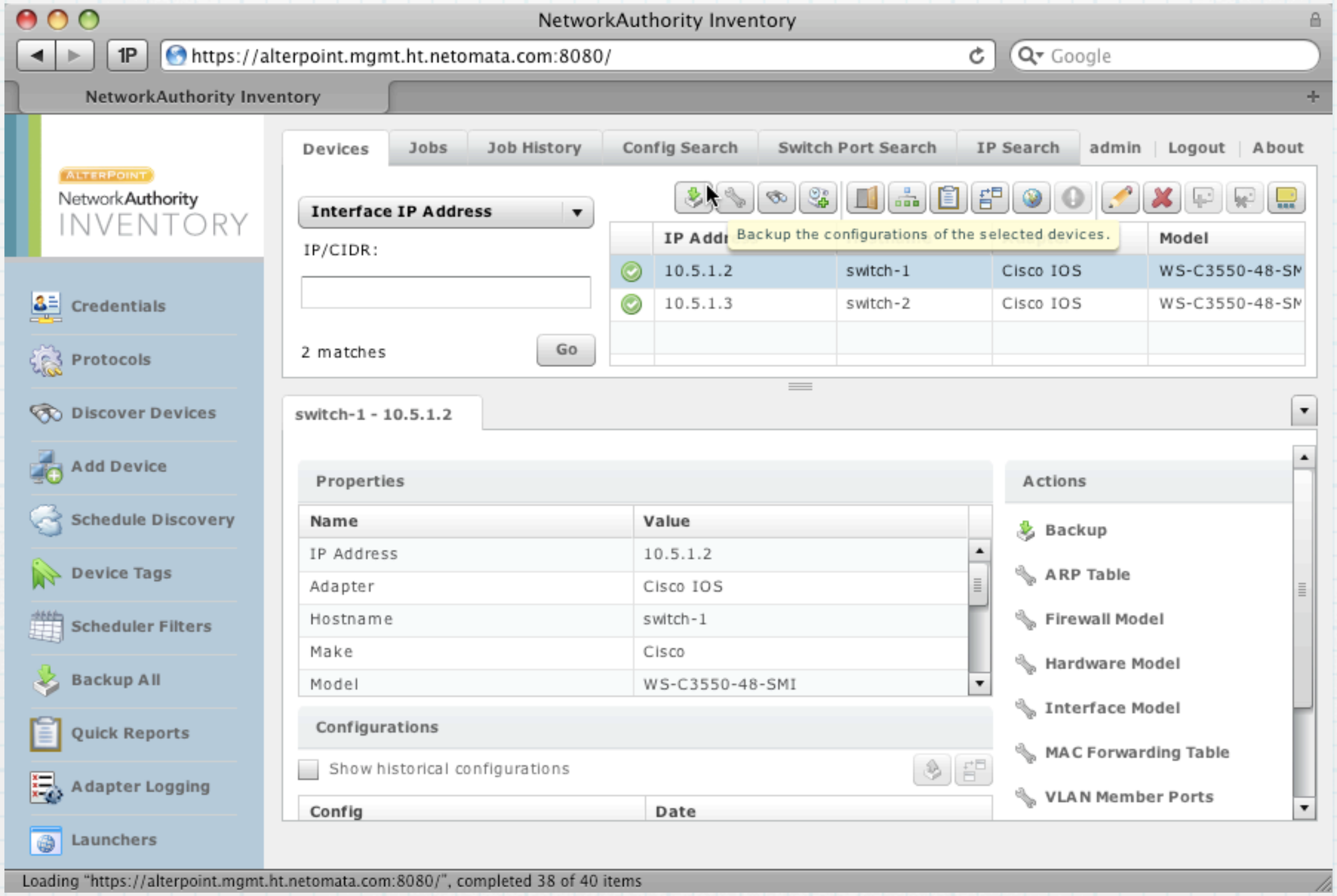
- Backup
- ARP Table
- Firewall Model
- Hardware Model
- Interface Model
- MAC Forwarding Table
- VLAN Member Ports

Loading "https://alterpoint.mgmt.ht.netomata.com:8080/", completed 38 of 40 items

Main window

* List of devices at top right
* Device details at bottom

ZipTie/NAI Backups



https://alterpoint.mgmt.ht.netomata.com:8080/

Interface IP Address

IP/CIDR:

2 matches

Backup the configurations of the selected devices.

	IP Addr			Model
✓	10.5.1.2	switch-1	Cisco IOS	WS-C3550-48-SM
✓	10.5.1.3	switch-2	Cisco IOS	WS-C3550-48-SM

switch-1 - 10.5.1.2

Properties

Name	Value
IP Address	10.5.1.2
Adapter	Cisco IOS
Hostname	switch-1
Make	Cisco
Model	WS-C3550-48-SMI

Configurations

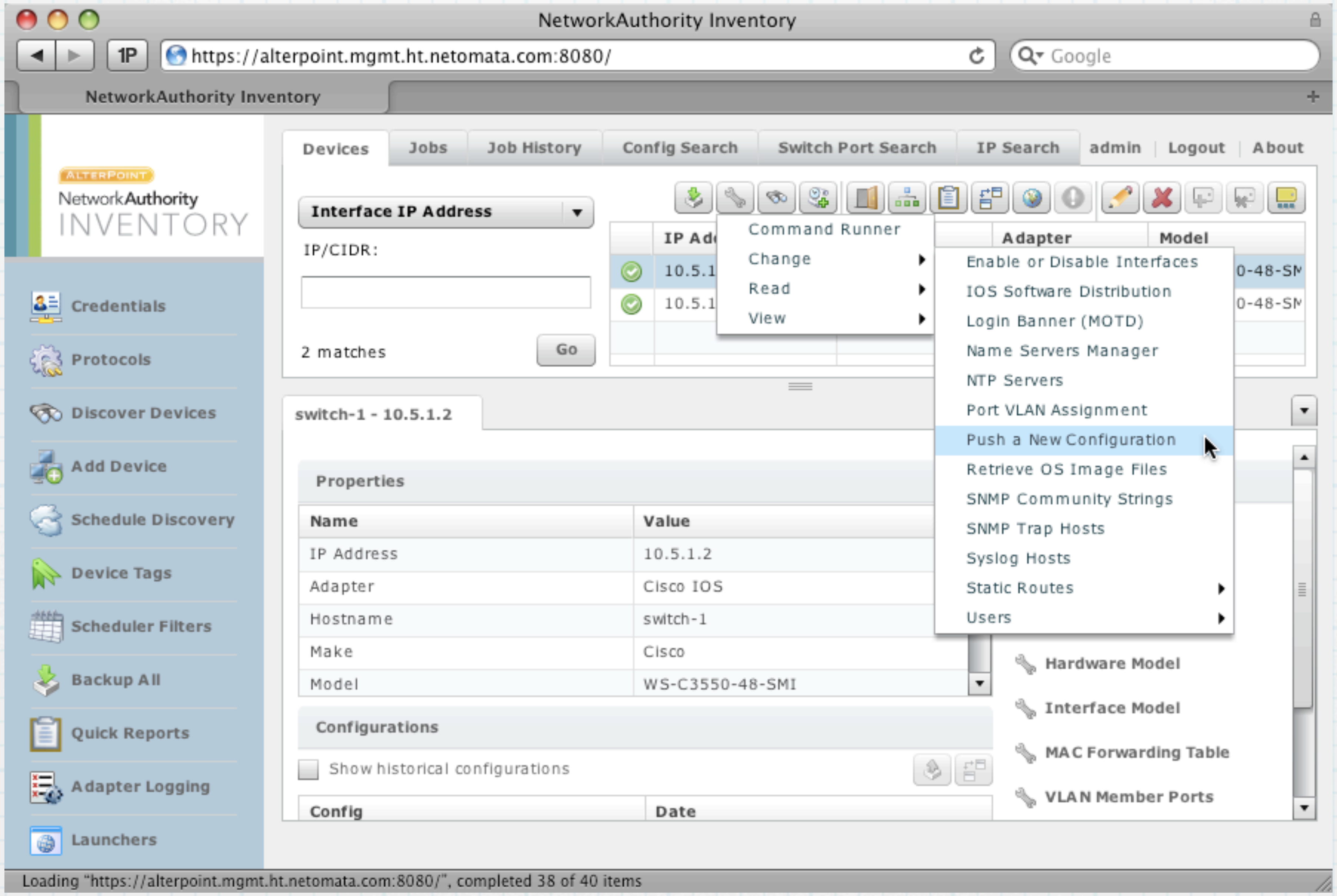
Show historical configurations

Config	Date

Loading "https://alterpoint.mgmt.ht.netomata.com:8080/", completed 38 of 40 items

Can backup and restore configs of selected devices

Push new config

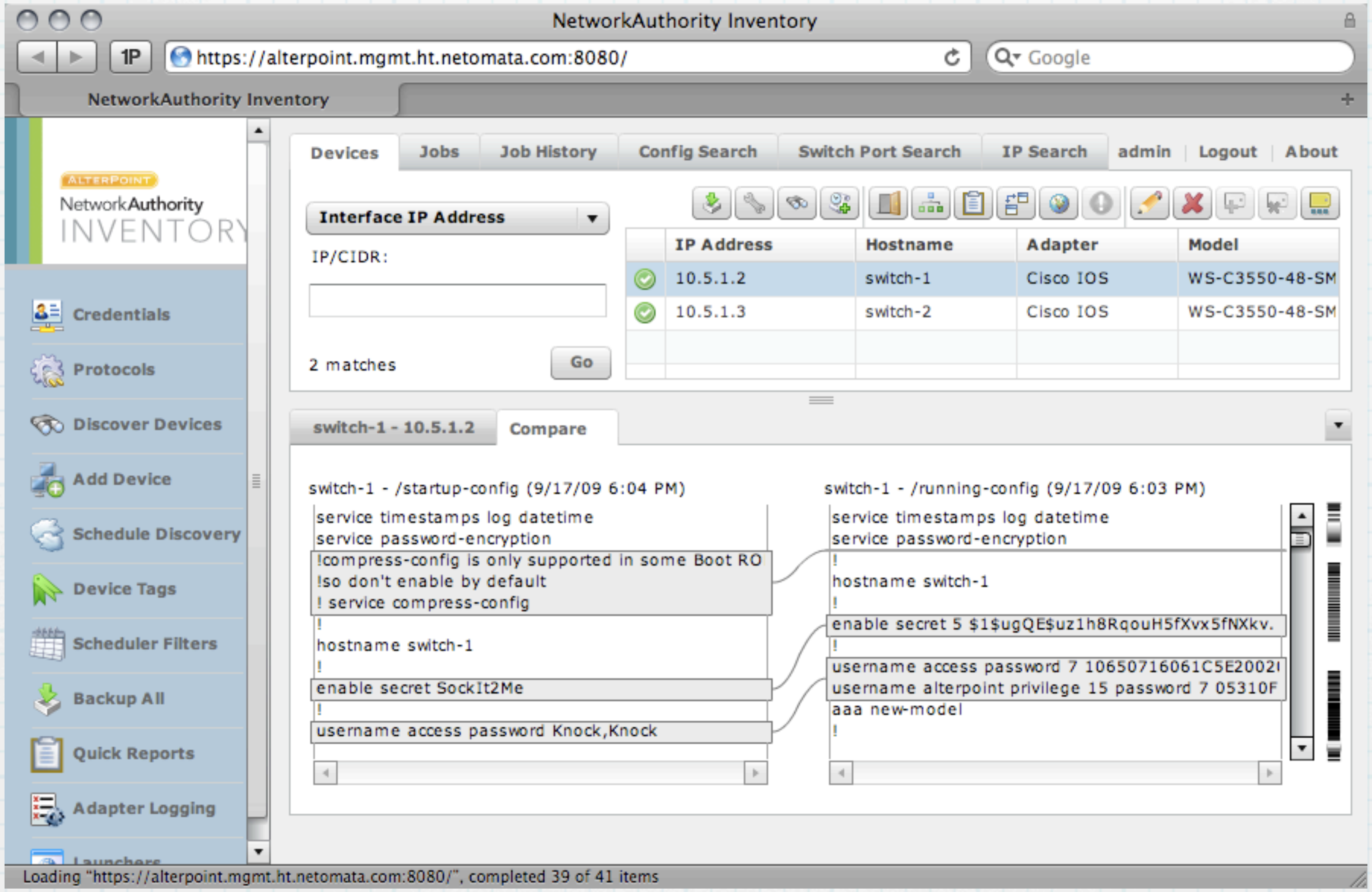


Name	Value
IP Address	10.5.1.2
Adapter	Cisco IOS
Hostname	switch-1
Make	Cisco
Model	WS-C3550-48-SMI

Loading "https://alterpoint.mgmt.ht.netomata.com:8080/", completed 38 of 40 items

Can push a new config to a device

Compare configs



	IP Address	Hostname	Adapter	Model
✓	10.5.1.2	switch-1	Cisco IOS	WS-C3550-48-SM
✓	10.5.1.3	switch-2	Cisco IOS	WS-C3550-48-SM

switch-1 - /startup-config (9/17/09 6:04 PM)

```
service timestamps log datetime
service password-encryption
!compress-config is only supported in some Boot RO
!so don't enable by default
! service compress-config
!
hostname switch-1
!
enable secret SockIt2Me
!
username access password Knock,Knock
```

switch-1 - /running-config (9/17/09 6:03 PM)

```
service timestamps log datetime
service password-encryption
!
hostname switch-1
!
enable secret 5 $1$ugQE$uz1h8RqouH5fXvx5fNXkv.
!
username access password 7 10650716061C5E2002
username alterpoint privilege 15 password 7 05310F
aaa new-model
!
```

Loading "https://alterpoint.mgmt.ht.netomata.com:8080/", completed 39 of 41 items

Cool tool for comparing configs

* on same device (before/after, startup/running)
* across devices

Installing ZipTie/NAI

http://inventory.alterpoint.com/

* Supported packages available for Ubuntu and Windows
* Unsupported packages also available for other Linux/UNIX variants and Mac OS X
* Also available as a VMware Virtual Appliance

Installing ZipTie/NAI

Requires

* JDK (1.5 or later)
* Perl 5.8.8 or later
* Bunch of Perl CPAN modules (which, in turn, need "make")

Installing configs with ZipTie/NAI

Configs need to be in "`tool-file-store`" subdir

* In "`/usr/share/networkauthority-inventory/`" in Ubuntu package

"Credentials" need to be set in ZipTie/NAI

* Username used needs SCP or FTP access

BEWARE: check carefully that it works; seems to fail silently (reports success, but doesn't work)

ZipTie/NAI's muddled licensing status

ZipTie developed largely by AlterPoint

- Community provided plug-ins and testing

ZipTie originally released as open source under Mozilla license

ZipTie/NAI's muddled licensing status

In 2008, AlterPoint rebranded ZipTie as "NetworkAuthority Inventory" and released under new license

* "Open Core" model
* AlterPoint says "The product is downloadable and can be used freely for internal business purposes"
* http://inventory.alterpoint.com/license

Vendor-specific device configuration tools

Some vendors offer tools to manage their devices

Problem is, they only manage <u>their</u> devices...

Few networks are single-vendor

So, these tools usually aren't much help

Automating Configuration of Network Services

SNMP status monitoring (e.g., Nagios)

Use NCG to generate config files for Nagios

* Walk list of devices
  * Generate device-level monitoring directives: CPU load, memory usage, temperature, etc.
  * Walk list of interfaces on device
    * Generate interface-level monitoring directives: up/down, traffic load, etc.

I.e., `web_hosting/templates/services/nagios`

SNMP trend monitoring (e.g., MRTG)

Use NCG to generate config files for MRTG

* Use MRTG's "`cfgmaker`" tool to generate basic MRTG config for sample device, then convert to NCG template
  * For each device, generate device-level monitoring directives: CPU load, memory usage, temperature, etc.
  * For each interface on each device, generate interface-level monitoring directives: bytes in/out, errors, etc.
* Also generate main "`index.html`" file
  * Start with index created by MRTG's "`indexmaker`" tool

Example in `web_hosting/templates/services/mrtg`

MRTG cfgmaker and indexmaker

Why not always use MRTG's "`cfgmaker`" and "`indexmaker`" to create MRTG config and `index.html` files?

* `cfgmaker` and `indexmaker` query devices and generate configs for the network as it is, not as it should be or will be
* Presumes that the device is up, and fully-configured
* Needs to be re-run every time there's a change in devices or interfaces (including name/description)

DNS

```
!services!dns!files {
    (+) {
        # DNS forward lookups
        ncg_template = dns/templates/dns.ncg
        ncg_output = configs/dns/dns.txt
    }
    (+) {
        # DNS IP reverse lookups
        ncg_template = dns/templates/in-addr.arpa.ncg
        ncg_output = configs/dns/in-addr.arpa.txt
    }
    (+) {
        # DNS IPv6 reverse lookups
        ncg_template = dns/templates/ip6.arpa.ncg
        ncg_output = configs/dns/ip6.arpa.txt
    }
}
```

Use NCG to generate

* DNS forward data (A, AAAA, CNAME, etc. records)
* DNS IPv4 reverse data (`in-addr.arpa` records)
* DNS IPv6 reverse data (`ip6.arpa` records)

DNS: templates/dns.ncg

```
<%
    (@target["!"].keys_having_key("dns_name") +
      @target["!"].keys_having_key("dns_cname")).
      uniq.each { |key|
        node = @target[key]
     if (node.has_key?("dns_name")) then
         if (node.has_key?("ip_address")) then
-%>
<%= node["dns_name"] %>.    IN    A     <%= node["ip_address"] %>
<%          end
            if (node.has_key?("ipv6_address")) then
-%>
<%= node["dns_name"] %>.    IN    AAAA  <%= node["ipv6_address"] %>
<%          end
        end
        if (node.has_key?("dns_cname")) then
            if (node.has_key?("dns_name")) then
-%>
<%= node["dns_cname"] %>.   IN    CNAME <%= node["dns_name"] %>.
<%          else
              raise "node #{node.key} has 'dns_cname', but no 'dns_name'"
            end
        end
    }
-%>
```

For all nodes with "dns_name" or "dns_cname" keys:

* If node has "dns_name" and "ip_address" keys, generate A record
* If node has "dns_name" and "ipv6_address" keys, generate AAAA record
* If node has "dns_cname" and "dns_name" keys, generate CNAME record

DNS: templates/ in-addr.arpa.ncg

```
<%
    @target["!"].keys_having_key("dns_name").each { |key|
        node = @target[key]
        if (node.has_key?("ip_address")) then
-%>
<%= IPAddr.new(node["ip_address"]).reverse %>.    IN   PTR  <%= node["dns_name"] %>.
<%      end
    }
-%>
```

For all nodes having "`dns_name`" keys:

* if node also has an "`ip_address`" key, generate a `PTR` record

DNS: templates/ ip6.arpa.ncg

```
<%
    @target["!"].keys_having_key("dns_name").each { |key|
        node = @target[key]
     if (node.has_key?("ipv6_address")) then
-%>
<%= IPAddr.new(node["ipv6_address"]).reverse %>.  IN   PTR  <%= node["dns_name"] %>.
<%      end
    }
-%>
```

For all nodes having "`dns_name`" keys:

* if node also has an "`ipv6_address`" key, generate a `PTR` record

DHCP

Use NCG to generate DHCP config files

- Static assignments of addresses to particular hostnames
- Just like walking tree to generate DNS configs

ACLs

Use NCG to generate ACLs (Access Control Lists) for various devices

`web_hosting` example:

* ACLs in switch configs block packets between environments (Production, QA, Dev, etc.)
* Prevents accidental cross-environment dependencies

`web_hosting` example demonstrates that ACLs can be distributed across generated config

ACLs

```
... statements to generate "interface VLANxxx" stanza ...
!
ip access-list extended MATCH_SUBNET_<%= vlan_id %>
  permit ip <%= vlan_ip %> 0.0.0.255 any
!
! add this VLAN to the "MATCH_SUBNET_ALL_ENV"
! access-list, which should already
! have other env subnets on it as well.
ip access-list extended MATCH_SUBNET_ALL_ENV
  permit ip <%= vlan_ip %> 0.0.0.255 any
!
vlan access-map VLAN<%= vlan_id %>_MAP 10
  match ip address MATCH_SUBNET_<%= vlan_id %>
  action forward
vlan access-map VLAN<%= vlan_id %>_MAP 20
  match ip address MATCH_SUBNET_ALL_ENV
  action drop
vlan access-map VLAN<%= vlan_id %>_MAP 30
  match ip address MATCH_ANY
  action forward
vlan filter VLAN<%= vlan_id %>_MAP vlan-list <%= vlan_id %>
```

`_v_env.ncg` template:

* Primarily configures environment's VLAN interface
* Also generates & applies ACLs for the environment
* Also adds environment's IP addresses to shared `MATCH_SUBNET_ALL_ENV` ACL used by all environment ACLs

"`show run`" will group all these together

VLANs

NCG `web_hosting` example shows good use of VLANs

* Multiple environments (Production, QA, Dev, etc.), each with its own VLAN
* Shared VLAN for management

VLANs

NCG `web_hosting` example shows how

* VLANs can be consistently named across multiple devices
* Consistent VLAN names can be used in descriptions in
  * devices (i.e., included in interface descriptions)
  * services (i.e., labels in MRTG and Nagios)

VPNs

Can use NCG to define

* List of devices
* List of tunnels between devices

Then generate configs for all devices (plus DNS, monitoring, etc.)

See http://www.netomata.com/wiki/point_to_point_links_example

Host Automation Systems

Puppet, cfengine, chef, etc.

Q: How to integrate with NCG?

A: Pick one to be "king", and either have

* NCG generate HAS config files
* or HAS generate NCG config files

Which should be "on top" depends on situation

Tips, Tricks, & Random Observations

Copying configs to/from Cisco IOS via SCP

```
brent% scp configs/switch-1.config \
  privuser@switch-1.mgmt.ht.netomata.com:startup-config
Password:
switch-1.config                                        100%   16KB  16.2KB/s   00:01
```

You can use SCP to copy configs to and from an IOS device

* SSH must be configured and working
* AAA authentication & authorization must be set
* User must have "privilege 15" (i.e., "enable" privs)
* SCP server must be explicitly enabled

```
aaa new-model
aaa authentication login default local
aaa authorization exec default local
username privuser privilege 15 password SockIt2Me
ip scp server enable
```

Cisco's "config replace"

Late-enough versions of IOS include a "`config replace`" command

Command lets you replace running config with new one, without rebooting

Device figures out changes needed to go from old to new config, and does them

Generally considered "not quite reliable yet"

Cisco's "config replace"

Roll-back and optional "`timeout`" parameter

* Lets device revert to old config if new config isn't confirmed within a specified period of time
* Can save you from cutting yourself off with a busted change

Requires "config archive" to be set up

* Which requires off-device storage for old configs

Automation-friendly devices and services

Configured via CLI or file, not GUI

Config can be replaced in whole, not simply edited

For example, for SNMP monitoring

* MRTG is good: config file is text, can be regenerated and replaced
* Cacti is bad: config is via GUI, no good way to load from text or replace entire config

The database is always right

If it's not, fix the database

* Make changes by fixing the database (or the templates, or whatever) rather than by changing devices directly
* If you need to make an emergency or experimental change to a device by hand, you aren't done until the automation system is updated to reproduce & propagate that change

Don't "fix" what you don't understand

If automated audit finds something unexpected (i.e., changed running config), flag it and alert humans

Automation shouldn't simply overwrite anomalies

Human needs to review anomalies and determine appropriate course of action

* Fix database: incorporate changes from device
* Fix device: return to "expected" config

Have one source of "truth"

Have one source of "truth" for any given fact (name, IP address, etc.)

If you're entering the same bit of data in multiple places (i.e., IP address), it opens up possibility for typos, duplicates, omissions, etc.

Better to have one source of "truth", and derive everything else from that

Everybody wants to be a hero

In most orgs, management rewards "heroism"

* Something broke, somebody saved the day by fixing

If you reward heroism, you get more of it

* More broken stuff, needing heroic fixes

If somebody had to be a hero, something went wrong

Instead, you should reward folks for <u>preventing</u> problems

Would-be heros will push back against automation, because it limits their opportunity to be a hero

Introducing automation to existing networks

Start with just one type of service/device/location, preferably something new and non-essential

- New VPN deployment?
- New VoIP phone deployment?
- New SNMP monitoring system?
- New site/room/rack?

Gain experience with the tools and methods

- Be able to show "See how much easier this is?"

Special Circumstances

QA labs, testbeds, and dev environments

Network automation lets you build dev/test/QA environments

* Quickly
* Easily
* That closely mimic production

Which leads to

* better dev/test/QA practices & experience
* easier deployment
* better production

IPv6

Enabling IPv6 requires lots of little changes to just about every device

* Enabling IPv6 on device
* Adding IPv6 addresses to interfaces
* Enabling IPv6-specific services, like RA

Much easier to do, if you're generating all configs

Cloud computing

If you're a cloud provider

* You need to automate network configuration along with VM provisioning
* Regardless of whether your customers are "internal" or "external"

CoBIT, ITIL, etc.

CoBIT, ITIL, and the like are all about

* Best practices
* Repeatability of process

Network automation gives you a way to repeatably, reliably configure your network devices and services

Strategies for Promoting Automation

Arguments to convince management

Emphasize

* Improved reliability through consistency & completeness
* Easier (and therefore cheaper) troubleshooting
  * Quicker return-to-service when problems occur
  * Less staff time spent chasing down problems
* Easier/cheaper maintenance
* Easier/cheaper upgrades, growth, and scalability

Arguments to convince staff

Emphasize

* Easier to work on, because more consistent
* More reliable, because more consistent
  * Less time firefighting, more time extending
  * Fewer middle-of-the-night problems
* More predictable
* Easier to extend

Resources & Tools

Resources

Netomata blog:
http://www.netomata.com/blog

Netomata community (wiki & lists):
http://www.netomata.com/community

"Cisco IOS Hints and Tips" blog:
http://blog.ioshints.info

Tools

Netomata Config Generator (NCG):
http://www.netomata.com/products/ncg

RANCID:
http://www.shrubbery.net/rancid/

AlterPoint NetworkAuthority Inventory (NAI, aka "ZipTie"):
http://inventory.alterpoint.com/

Netdisco: http://www.netdisco.org/

OpenNMS: http://www.opennms.org/

Nagios: http://www.nagios.org/

MRTG: http://oss.oetiker.ch/mrtg/

Automating Network Configuration

Brent Chapman
brent@netomata.com

Netomata, Inc.
www.netomata.com

More About Netomata Config Generator (NCG)

Netomata Config Generator (NCG)

Generates config files that are

* for various devices and services
* complete, consistent, ready-to-install

Open source (GPLv3), written in Ruby (>1.8.5)

http://www.netomata.com/products/ncg

http://www.netomata.com/docs/programs/ncg

NCG inputs

Two sets of input

* Model describing network (devices, interfaces, interface parameters, etc.)

* Templates for various types of devices (routers, switches, firewalls, etc.) and services (DNS, DHCP, MRTG, etc.)

NCG phases

Works in two phases

* First phase builds up completely populated model of network in memory
* Second phase combines model and templates to generate configs for all devices and services

Generated configs are then installed manually, or via RANCID, ssh, rsync, etc.

NCG user roles

Designed for two distinct user roles:

* Architects define model, create & modify templates, etc.
* Engineers add/modify/delete data in model, then "turn the crank" to produce configs

NCG phase 1: modeling

Build model with all the details about your network:

* Devices
* Interfaces
* VLANs
* IP addresses
* etc.

In phase 2, NCG combines this model with templates, to generate ready-to-install configs

NCG input files

Two types of input files:

* .neto files — hierarchical data files
* .neto_table files — tabular data files

.neto files

Hierarchical data files

Curly-bracket nested

* look sort of like BIND or DHCP config files

http://www.netomata.com/docs/formats/neto

.neto files

```
base_ip = 172.16.0.0
base_ip_netmask = 255.255.0.0
devices {
  (+) {
    name = switch-1
    make = Cisco
    model = 3550-48
    interfaces {
      # ...
    }
  }
}
```

key — string that names a particular value

value — either a node or a string

node — contains list of key/value pairs; can be nested

.neto files

```
base_ip = 172.16.0.0
base_ip_netmask = 255.255.0.0
devices {
  (+) {
    name = switch-1
    make = Cisco
    model = 3550-48
    interfaces {
      # ...
    }
  }
}
```

(+) creates new "anonymous" key (sort of like a pointer in C)

# marks comments (to end of line)

.neto files

```
base_ip = 172.16.0.0
base_ip_netmask = 255.255.0.0
devices {
   (+) {
      name = switch-1
      make = Cisco
      model = 3550-48
      interfaces {
         # ...
      }
   }
}
```

.neto files

```
base_ip = 172.16.0.0
base_ip_netmask = 255.255.0.0
devices {
   (+) {
      name = switch-1
      make = Cisco
      model = 3550-48
      interfaces {
         # ...
      }
   }
}
```

→

```
!base_ip = 172.16.0.0
!base_ip_netmask = 255.255.0.0
!devices <node>
!devices!@000000001 <node>
!devices!@000000001!name = switch-1
!devices!@000000001!make = Cisco
!devices!@000000001!model = 3550-48
!devices!@000000001!interfaces <node>
...
```

.neto example

```
base_ip          = 10.5.0.0
admin_ip         = [%= ip_union(@target["(...)!base_ip"], "0.0.16.0") %]
syslog_ip        = [%= ip_union(@target["(...)!admin_ip"], "0.0.0.26") %]
syslog_facility = local5
snmp_ip          = [%= ip_union(@target["(...)!admin_ip"], "0.0.0.27") %]
snmp_community  = public
domain           = example.com

table vlans.neto_table
```

`[%= ... %]` is an embedded Ruby (ERB) block

* Treated as a tiny little stand-alone file of Ruby code

`ip_union()` is an NCG-provided utility method

* Combines 2 IP addresses via a bit-wise OR

`@target` is a Ruby variable referring to current node

So, to decode:

* `admin_ip = [%= ip_union("10.5.0.0", "0.0.16.0") %]`
* `admin_ip = "10.5.16.0"`

.neto example

```
base_ip          = 10.5.0.0
admin_ip         = [%= ip_union(@target["(...)!base_ip"], "0.0.16.0") %]
syslog_ip        = [%= ip_union(@target["(...)!admin_ip"], "0.0.0.26") %]
syslog_facility = local5
snmp_ip          = [%= ip_union(@target["(...)!admin_ip"], "0.0.0.27") %]
snmp_community  = public
domain           = example.com

table vlans.neto_table
```

Incorporates a .neto_table file

.neto_table files

.neto_table format is just a shortcut

* Any .neto_table file can be expressed as a .neto file
* Each .neto_table file has a header with directives for essentially converting data to a .neto file

.neto_table files

Tabular data files

* Each line is a row of data
* Columns are tab-separated
* All data lines must have same number of columns

http://www.netomata.com/docs/formats/neto_table

.neto_table example

```
@ !vlans!(+)!id = %{id}
@ !vlans!(id=%{id})!active = %{active}
@ !vlans!(id=%{id})!type = %{type}
@ !vlans!(id=%{id})!vlan_ip = [%= ip_union(@target["(...)!base_ip"], "0.0.%{id}.0") %]
@ !vlans!(id=%{id})!netmask = %{netmask}
@ !vlans!(id=%{id})!name = %{name}
@ !vlans!(id=%{id})!description = %{description}
#
% id      active  type     netmask          name            description
# --      ------  ----     --------         ----            -----------
2         yes     switch   255.255.255.0    ISP-FW          ISP-to-Firewall
3         yes     switch   255.255.255.0    FW-LB           Firewall-to-LoadBal
4         yes     admin    255.255.255.0    LB-Rtr          LoadBal-to-Router
16        yes     admin    255.255.240.0    Management      Management
32        yes     admin    255.255.240.0    Bulk            Bulk
```

# lines are comments

.neto_table example

```
@ !vlans!(+)!id = %{id}
@ !vlans!(id=%{id})!active = %{active}
@ !vlans!(id=%{id})!type = %{type}
@ !vlans!(id=%{id})!vlan_ip = [%= ip_union(@target["(...)!base_ip"], "0.0.%{id}.0") %]
@ !vlans!(id=%{id})!netmask = %{netmask}
@ !vlans!(id=%{id})!name = %{name}
@ !vlans!(id=%{id})!description = %{description}
#
% id     active  type    netmask          name             description
# --     ------  ----    --------         ----             -----------
2        yes     switch  255.255.255.0    ISP-FW           ISP-to-Firewall
3        yes     switch  255.255.255.0    FW-LB            Firewall-to-LoadBal
4        yes     admin   255.255.255.0    LB-Rtr           LoadBal-to-Router
16       yes     admin   255.255.240.0    Management       Management
32       yes     admin   255.255.240.0    Bulk             Bulk
```

@ lines are directives that are re-processed for every line of data, with variables substituted from current data line

.neto_table example

```
@ !vlans!(+)!id = %{id}
@ !vlans!(id=%{id})!active = %{active}
@ !vlans!(id=%{id})!type = %{type}
@ !vlans!(id=%{id})!vlan_ip = [%= ip_union(@target["(...)!base_ip"], "0.0.%{id}.0") %]
@ !vlans!(id=%{id})!netmask = %{netmask}
@ !vlans!(id=%{id})!name = %{name}
@ !vlans!(id=%{id})!description = %{description}
#
% id      active  type    netmask         name            description
# --      ------  ----    -------         ----            -----------
2       yes     switch  255.255.255.0   ISP-FW          ISP-to-Firewall
3       yes     switch  255.255.255.0   FW-LB           Firewall-to-LoadBal
4       yes     admin   255.255.255.0   LB-Rtr          LoadBal-to-Router
16      yes     admin   255.255.240.0   Management      Management
32      yes     admin   255.255.240.0   Bulk            Bulk
```

% line names columns (separated by 1 or more tabs)

%{name} references data from a given column

* For example, in processing first line of data:
    * `%{id} = 2, %{active} = yes, %{type} = switch,` etc.

.neto_table example

```
@ !vlans!(+)!id = %{id}
@ !vlans!(id=%{id})!active = %{active}
@ !vlans!(id=%{id})!type = %{type}
@ !vlans!(id=%{id})!vlan_ip = [%= ip_union(@target["(...)!base_ip"], "0.0.%{id}.0") %]
@ !vlans!(id=%{id})!netmask = %{netmask}
@ !vlans!(id=%{id})!name = %{name}
@ !vlans!(id=%{id})!description = %{description}
#
% id     active  type     netmask          name            description
# --     ------  ----     --------         ----            -----------
2        yes     switch   255.255.255.0    ISP-FW          ISP-to-Firewall
3        yes     switch   255.255.255.0    FW-LB           Firewall-to-LoadBal
4        yes     admin    255.255.255.0    LB-Rtr          LoadBal-to-Router
16       yes     admin    255.255.240.0    Management      Management
32       yes     admin    255.255.240.0    Bulk            Bulk
```

Processing first line of data (i.e., doing "@" lines) is same as:

```
!vlans!(+)!id = 2
!vlans!(id=2)!active = yes
!vlans!(id=2)!type = switch
!vlans!(id=2)!vlan_ip = [%= ip_union(@target["(...)!base_ip"], "0.0.2.0") %]
!vlans!(id=2)!netmask = 255.255.255.0
!vlans!(id=2)!name = ISP-FW
!vlans!(id=2)!description = ISP-to-Firewall
```

.neto_table tips

Separate headers from data

- "table" directive in .neto file takes multiple arguments (including globs)
- Concatenates files before processing; behaves as if they were a single file

.neto_table tips

Uses for separating .neto_table headers from data:

* Use same header for different data files (for example, list switch interfaces in a separate file for each switch, with the same header for each)
* Use different header for same data files (for example, use one header for file listing VLANs to create VLAN table for switches, and another header for same file to create VLAN interfaces on switches)

.neto_table tips

More uses for separate .neto_table headers and data:

* Header files define data model; changes can be limited to "architect" role
* Data files can be simplified for editing by non-architects, without confusion of directives in headers
* Could create simple web interface for data files

.neto_table tips

Structure data model and input files so that common changes are easily done through a simple .neto_table data file edit

For example:

* Add a device by adding one line to data file
* Change the type of an interface by changing a column in an existing data line

.neto_table tips

.neto_tables are a very important feature of NCG:

* Make it possible to structure input so that "easy changes are easy"
* Make it possible to separate "architect" from "engineer" role
  * Architects set up data model (via .neto_table headers)
  * Engineers manage data in .neto_table data files
* Foundation for web UI integration

NCG phase 2: generation

Once tree has been fully populated, NCG moves on and starts generating config files

* Find all nodes in tree having "`ncg_output`" key
  * Value is filename to write generated config to
  * File is created or overwritten, as necessary
* Generate each config file (named in "`ncg_output`") using template file named in "`ncg_template`" key in same node
  * If node has an "`ncg_output`" key, it <u>must</u> have an "`ncg_template`" key

.ncg templates

Embedded Ruby (ERB)

* http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/classes/ERB.html
* Also known as "eRuby"

See also http://www.netomata.com/docs/formats/ncg

Embedded Ruby (ERB)

Much like PHP or other embedded templating languages

Allows you to embed programming logic into a template for a given config file

* For example, iterate through list of interfaces to produce interface statements for device config

Embedded Ruby (ERB)

```
<%
my_hostid = 17
(0..2).each do |i|
-%>
auto eth<%= i %>
interface eth<%= i %> inet static
    address 192.168.<%= i %>.<%= my_hostid %>
    netmask 255.255.255.0
<% if (i == 0) then -%>
    gateway 192.168.<%= i %>.1
<% end -%>
    broadcast 192.168.<%= i %>.255
<%# blank line %>
<% end -%>
```

`<% ... %>` is executed, but nothing is output

`<%= ... %>` is executed, and value is output

`<%# ... %>` is a comment

`... -%>` "swallows" trailing newline in output

`<%% ... %%>` is replaced with `<% ... %>`

Embedded Ruby (ERB)

```
<%
my_hostid = 17
(0..2).each do |i|
-%>
auto eth<%= i %>
interface eth<%= i %> inet static
    address 192.168.<%= i %>.<%= my_hostid %>
    netmask 255.255.255.0
<% if (i == 0) then -%>
    gateway 192.168.<%= i %>.1
<% end -%>
    broadcast 192.168.<%= i %>.255
<%# blank line %>
<% end -%>
```

$\rightarrow$

```
auto eth0
interface eth0 inet static
    address 192.168.0.17
    netmask 255.255.255.0
    gateway 192.168.0.1
    broadcast 192.168.0.255

auto eth1
interface eth1 inet static
    address 192.168.1.17
    netmask 255.255.255.0
    broadcast 192.168.1.255

auto eth2
interface eth2 inet static
    address 192.168.2.17
    netmask 255.255.255.0
    broadcast 192.168.2.255
```

Running NCG

```
brent% cd examples/web_hosting
brent% ls -lR configs
brent%
```

Nothing up my sleeve...

Running NCG

```
brent% ncg -v web_hosting.neto
NETOMATA_LIB=/usr/local/lib/netomata/lib
Reading file(s) web_hosting.neto
Reading file(s) ./web_hosting.pre.neto
Reading table(s) ./vlans.neto_table_hdr ./vlans.neto_table
Reading file(s) ./secrets.neto
Reading table(s) ./interfaces.neto_table
Reading table(s) ./vlan-interfaces.neto_table_hdr ./vlans.neto_table
Reading table(s) ./users.neto_table
Reading table(s) ./vlan-interfaces.neto_table_hdr ./vlans.neto_table
Reading table(s) ./users.neto_table
Reading file(s) ./web_hosting.post.neto
Generating ./configs/switch-1.config for !devices!@000000001
Generating ./configs/nagios/switch-1.cfg for !devices!@000000001!services!nagios
Generating ./configs/switch-2.config for !devices!@000000002
Generating ./configs/nagios/switch-2.cfg for !devices!@000000002!services!nagios
Generating ./configs/mrtg/mrtg.cfg for !services!mrtg!config
Generating ./configs/mrtg/index.html for !services!mrtg!index
```

Running NCG

```
brent% ls -lR configs
total 40
drwxr-xr-x   5 brent   staff     170 Sep 14 19:04 mrtg
drwxr-xr-x   5 brent   staff     170 Sep 14 19:04 nagios
-rw-r--r--   1 brent   staff   16566 Sep 14 19:04 switch-1.config
-rw-r--r--   1 brent   staff   16567 Sep 14 19:04 switch-2.config

configs/mrtg:
total 56
-rw-r--r--   1 brent   staff   17610 Sep 14 19:04 index.html
-rw-r--r--   1 brent   staff   25798 Sep 14 19:04 mrtg.cfg
-rw-r--r--   1 brent   staff    7882 Sep 14 19:04 netomata.logo.
160x80.jpg

configs/nagios:
total 28
-rw-r--r--   1 brent   staff    1235 Sep 14 19:04 COMMON.cfg
-rw-r--r--   1 brent   staff   10575 Sep 14 19:04 switch-1.cfg
-rw-r--r--   1 brent   staff   10575 Sep 14 19:04 switch-2.cfg
```

Running NCG

```
brent% more configs/switch-1.config
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Generated by Netomata Config Generator (ncg)
!!     http://www.netomata.com/
!! --------
!! Target: switch-1
!! Date: Mon Sep 14 19:04:02 -0700 2009
!! User: brent
!! Host: Brent-MacBookPro.local
!! Directory: /Users/brent/Netomata/software/ncg/examples/web_hosting
!! Command: /Users/brent/Netomata/software/ncg/bin/ncg web_hosting.neto
!! --------
...
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
no service pad
service timestamps debug datetime
service timestamps log datetime
service password-encryption
service compress-config
!
hostname switch-1
!
enable secret SockIt2Me
!
username access password Knock,Knock
...
```

Running NCG

```
brent% diff -U 1 configs/switch-[12].config
--- configs/switch-1.config2009-09-14 19:04:02.000000000 -0700
+++ configs/switch-2.config2009-09-14 19:04:02.000000000 -0700
@@ -4,3 +4,3 @@
 !! --------
-!! Target: switch-1
+!! Target: switch-2
 !! Date: Mon Sep 14 19:04:02 -0700 2009
@@ -23,3 +23,3 @@
 !
-hostname switch-1
+hostname switch-2
 !
@@ -53,3 +53,3 @@
 spanning-tree extend system-id
-spanning-tree vlan 1-4094 priority 24576
+spanning-tree vlan 1-4094 priority 28672
 !
@@ -329,3 +329,3 @@
 interface FastEthernet0/25
- description host-1 [ipmi]
+ description host-2 [ipmi]
  switchport access vlan 48
```

Installing NCG-generated config files

NCG generates config files, but doesn't install them

You still need:

* A method to review them
    * diff against previously-installed files?
* A mechanism to install them
    * scp, RANCID, ZipTie, ...

Should probably automate your whole review/approve/install process with a Makefile or Rakefile or something

Creating NCG templates

Start with working configs

- From devices/services in production
- From lab tests or trial setups

Think about the data

- Figure out what data needs to be in tree
- Make sure it's there

Experience setting up NCG templates

Setting up data and templates for first type of device or service takes several days

* Get all the data together
* Massage it into .neto and .neto_table format, and structure tree
* Template config

Experience setting up NCG templates

Second type of device/service takes a day or two

* Figure out what else you should have put into the tree
* Fix the form of things that are in the tree, but not in a useful way

Experience setting up NCG templates

Each successive type of device or service gets easier and faster

* Less need to "fix" tree
* Data is already there, and in useful form