Yaml

What is yaml

YAML is a human readable mechanism for storing data that is easily mapped to common data types.

It supports hierarchical structures

Whitespace delimited (like Python) so files have consistent look.

Program language agnostic – supported by most.

Yaml - Example

```
---
ipsec_monitor_interval: 5
ipsec_monitor_threshold: 5
zones:
  untrust:
    name: untrust
    interfaces:
      - lo0.0
      - ge-0/0/1.0
      - ge-0/0/4.0
      - ge-0/0/5.0
```

YAML files always start with 3 hyphens

Yaml – data types

- YAML automatically detects data types. You can do programmy stuff with them, but we won't be covering that.

```
---
boolean: true
float: 105.7
integer: 33
string: This is a string
string2: "This is also a string"
```

Yaml - lists

- Lists can be in blocks using a hyphen+space to begin new items

```
---
- MX
- SRX
- QFX
```

- Or inline delimited by comma+space and enclosed in square brackets

```
---
[MX, SRX, QFX]
```

Yaml – Dictionary/Associate Array

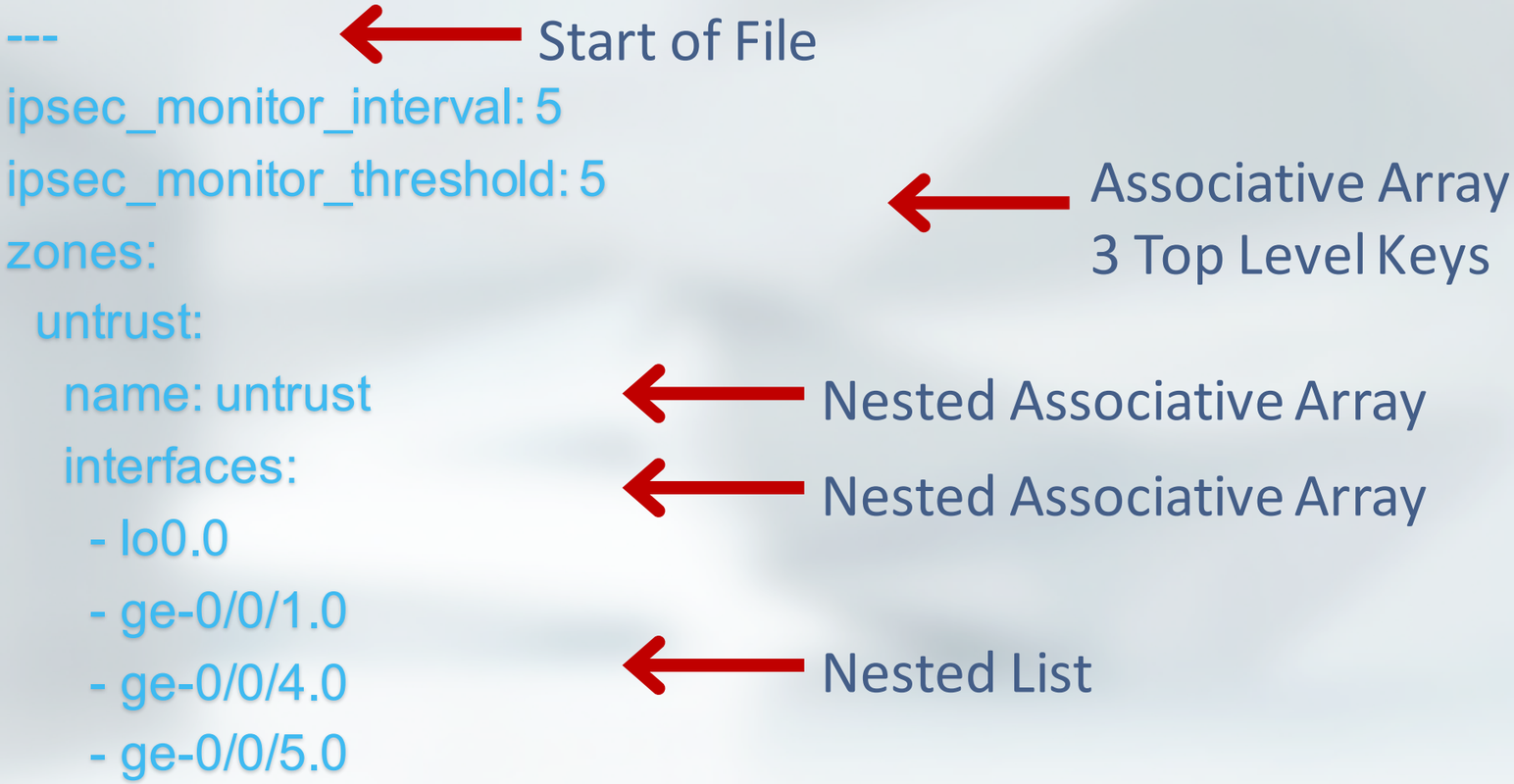
Associate Arrays can be in blocks with new lines and indention with keys and values separated by colon+space

```
---
    name: lo0
      logical_unit: 185
      inet_address: 10.1.1.185/32
```

Or inline delimited by comma+space with keys and values separated by colon+space enclosed in curly brackets

```
---
    {name: lo0, logical_unit: 185, inet_address: 10.1.1.185/32}
```

Yaml – Example Deconstructed



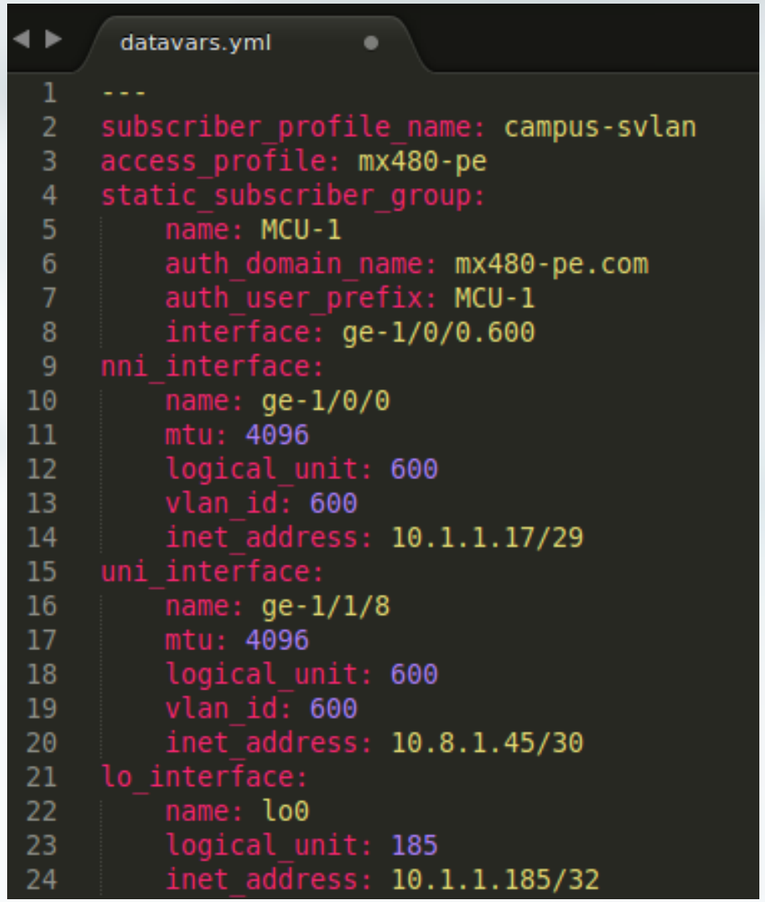
Yaml - gotchas

- Whitespace indented, but DOES NOT MIXED WHITESPACE!
  - Suggested format is to use (2) spaces for indents
    - Not hard requirement, but must be the same throughout a file

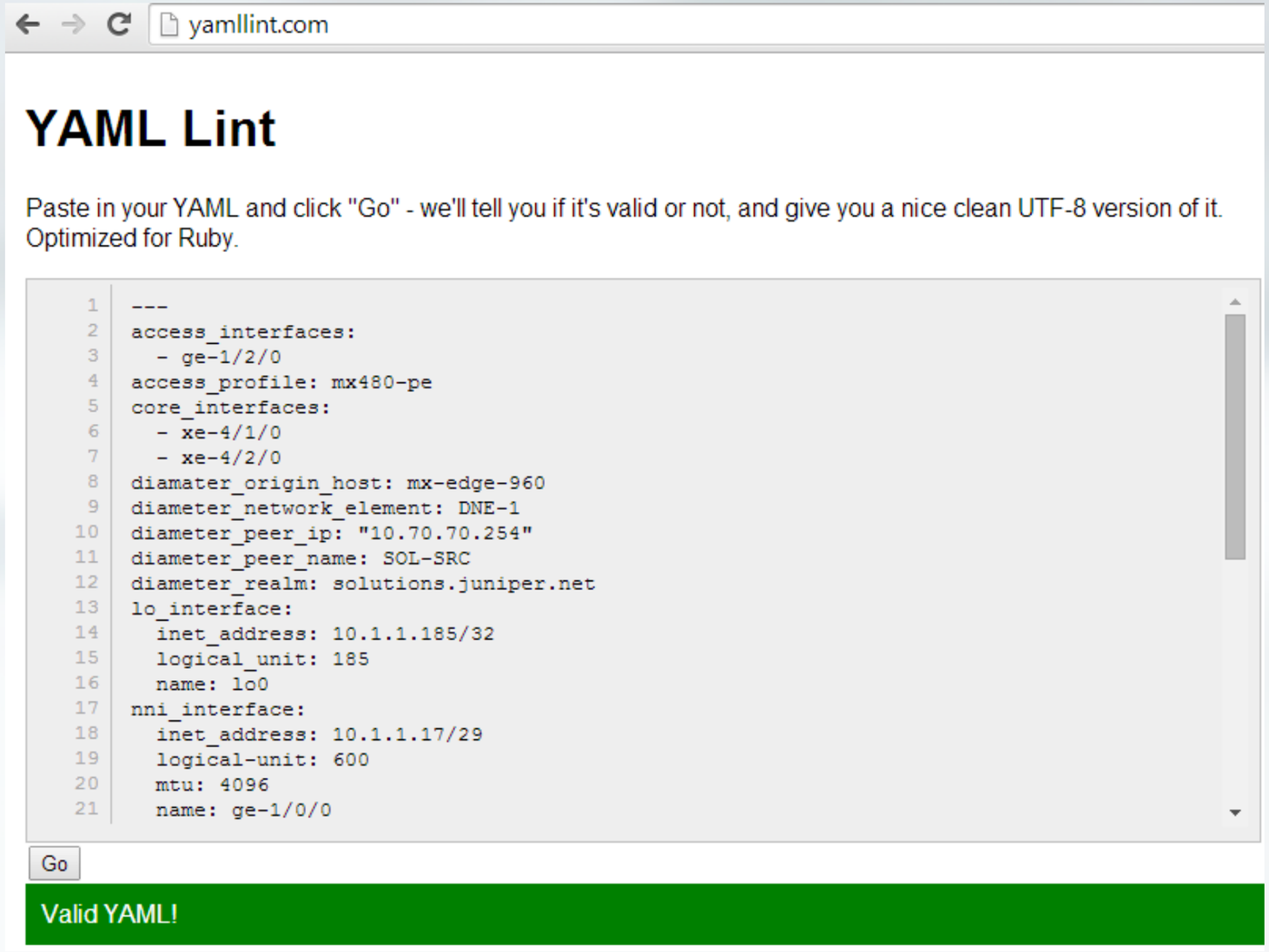
- Comments begin with a number sign # and separated by whitespace

Yaml - editor

- Using an editor with YAML support can make your life easier. Here is an example of Sublime 3



Yaml – Check validity (Linting)



NETCONF

Compatible with:
All Junos Platforms

NETCONF Explained

What is NETCONF?

- XML Device API implemented over SSH subchannel
- SSH –p 830 user@device –s netconf
- Communicate using XML RPC
- Configuration Commands & Operational Commands

Off-Box Workflow Automation

NETCONF – XML over SSH

- Secure and connection oriented … SSHv2 as transport
- Structured and transaction based … XML as RPC request / response
- User-class privilege aware … Native to Junos

Management System

NETCONF XML PROTOCOL (RFC4741)

Automate config changes, remote invocation of operational commands, collection of logs

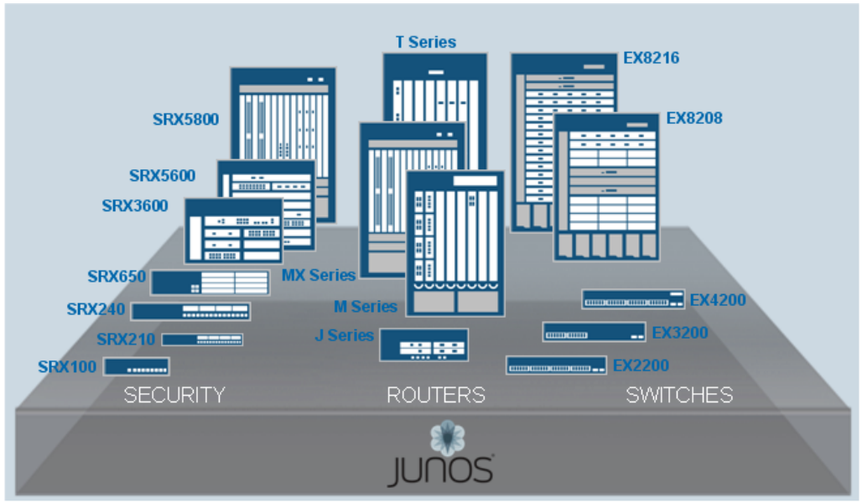
Secure TCP/IP connections via SSHv2 (RFC4742)

XML

NETCONF client libraries exist for a number of programming languages such as Java, Perl, Ruby, Python, and even SLAX !

JUNOS

Security | **Routing** | **Switching**



Junos Configuration

- The following illustrates the Junos configuration as an XML document for a VLAN named “Accounting”

```
[edit]
admin@EX# show vlans Accounting | display xml
```

CLI

```
[edit]
admin@EX-dc1# show vlans Accounting
description "Accounting Department";
vlan-id 500;
interface {
    ge-0/0/12.0;
    ge-0/0/13.0;
    ae0.0;
}
```

XML

```
<configuration>
   <vlans>
      <vlan>
         <name>Accounting</name>
         <description>Accounting Department</description>
         <vlan-id>500</vlan-id>
         <interface>
            <name>ge-0/0/12.0</name>
         </interface>
         <interface>
            <name>ge-0/0/13.0</name>
         </interface>
         <interface>
            <name>ae0.0</name>
         </interface>
      </vlan>
   </vlans>
</configuration>
```

Junos Operational Commands

- The following illustrates the "`show vlans`" operational command in XML:

```
admin@EX> show vlans | display xml rpc
```

```
admin@EX> show vlans | display xml
```

XML RPC COMMAND

```
<rpc>
  <get-vlan-information>
</rpc>
```

XML RESPONSE

```
<rpc-reply>
  <vlan-information …>
    <vlan-terse/>
    <vlan>
        <vlan-instance>0</vlan-instance>
        <vlan-name>VLAN-blue</vlan-name>
        <vlan-create-time>Wed Nov 16 12:42:03 2011
        </vlan-create-time>
        <vlan-status>Enabled</vlan-status>
        <vlan-owner>static</vlan-owner>
        <vlan-tag>500</vlan-tag>
        <vlan-tag-string>500</vlan-tag-string>
        <vlan-index>2</vlan-index>
        <vlan-protocol-port>Port Mode</vlan-protocol-port>
        <vlan-members-count>1</vlan-members-count>
        <vlan-members-upcount>0</vlan-members-upcount>
        <vlan-detail>
          <vlan-member-list>
            <vlan-member>
              <vlan-member-interface>ge-0/0/19.0</vlan-member-interface>
            </vlan-member>
          </vlan-member-list>
        </vlan-detail>
    </vlan>
    …
</rpc-reply>
```

Demo

Documentation:
http://www.juniper.net/documentation/en_US/junos13.2/information-products/pathway-pages/netconf-guide/netconf.html

SLAX & JUISE

LIBSLAX

- Junos OnBox scripting language
- Based on XSLT
- Exported SLAX from Junos
  - Standalone language
  - Open-Source (githib.com/juniper/libslax)
- Imported back into JUNOS-12.2, 12.3
  - Re-imported on each release

SLAX Scripts

Operation Script

- Create Custom Commands
- Diagnose Network Problems
- Controlled Configuration Change

Event Script

- Automate Event Responses
- Correlate Events

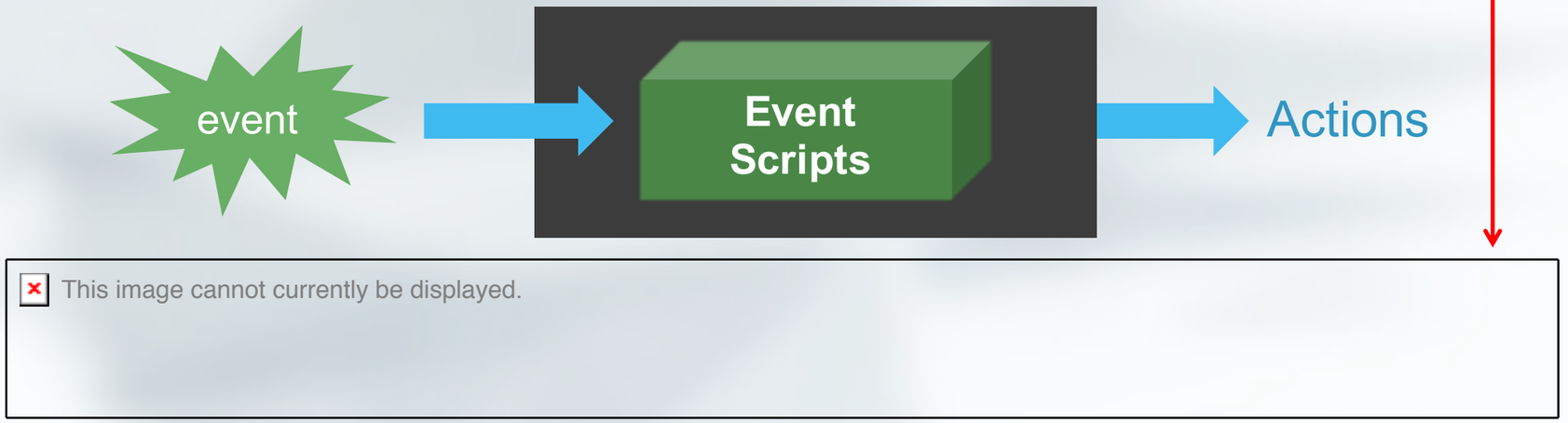
Commit Configuration Script

- Assure compliance to business rules network/security policies

Event Automation

Event scripts are triggered by events on the device

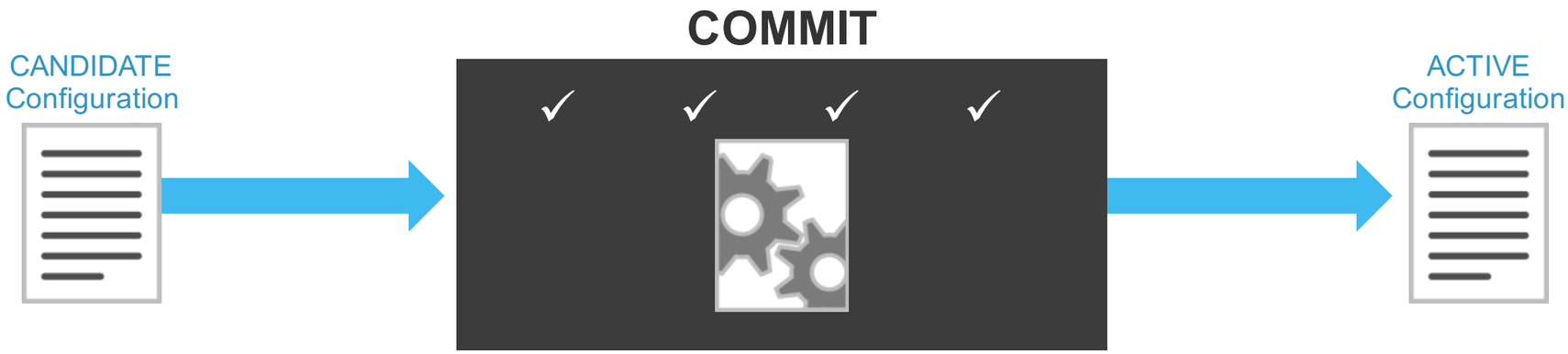
- Gather relevant troubleshooting info and correlate events from leading indicators
- Automate event responses with a set of actions

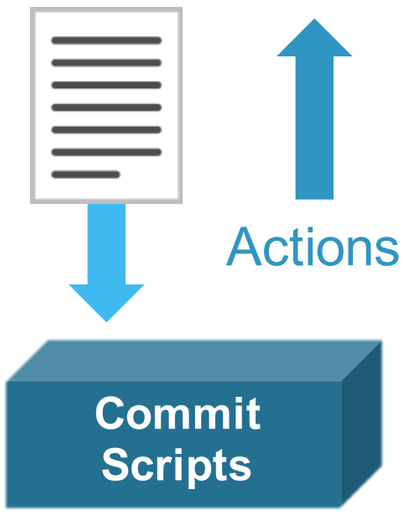


Benefits

- ✓ Automate time-of-day configuration changes
- ✓ Speed time-to-resolve to reduce the downtime
- ✓ Automate response to leading indicators to minimize impact

Configuration Automation



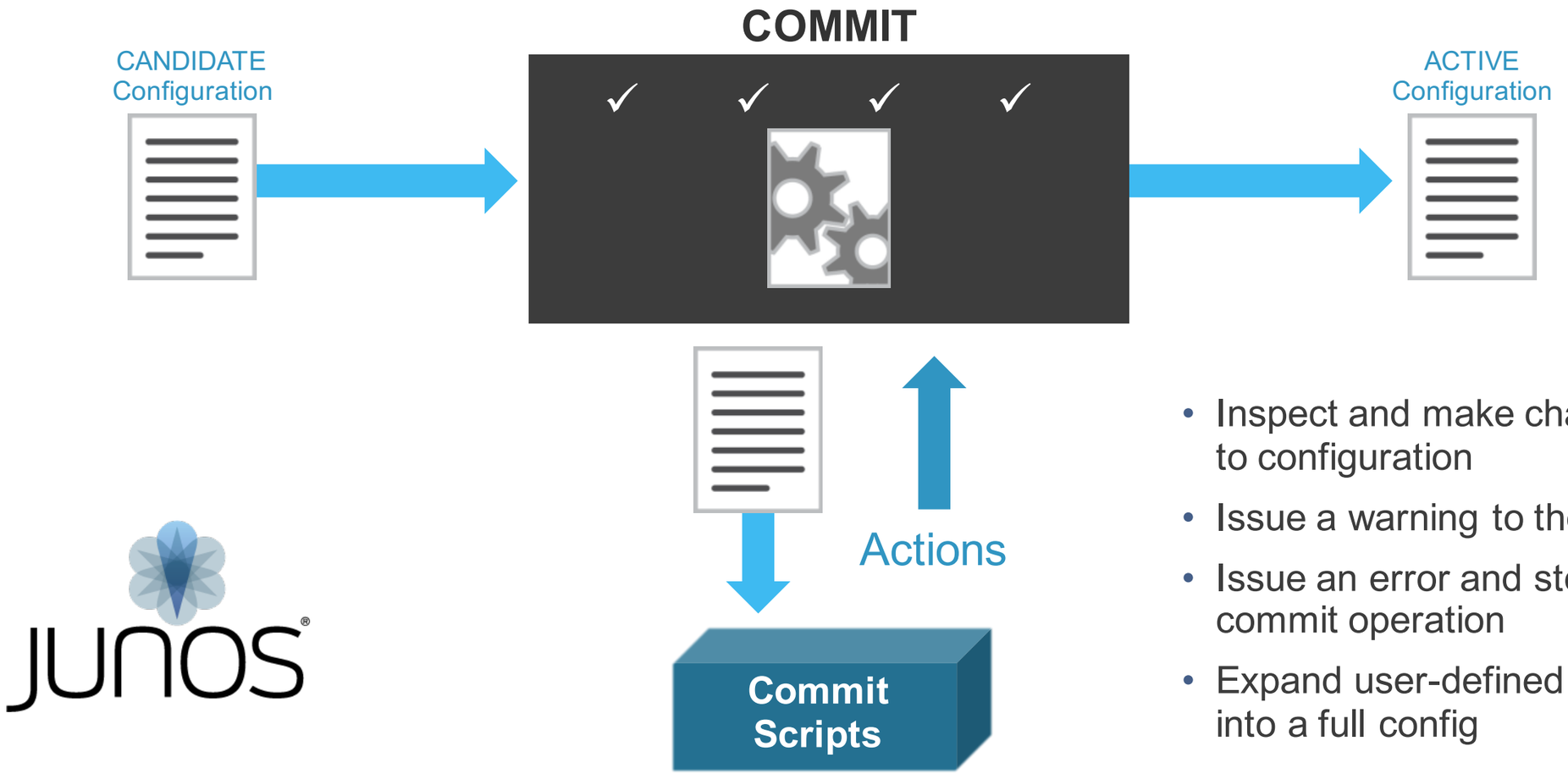


- Abstract complex configuration into a simple set of base commands
- Enforce best practices and business rules

Benefits

- ✓ Assure compliance to business rules and policies
- ✓ Provide change management to avert and correct errors
- ✓ Simplify and speed setup of complex configuration

Configuration Automation



- Inspect and make changes to configuration
- Issue a warning to the user
- Issue an error and stop the commit operation
- Expand user-defined macros into a full config

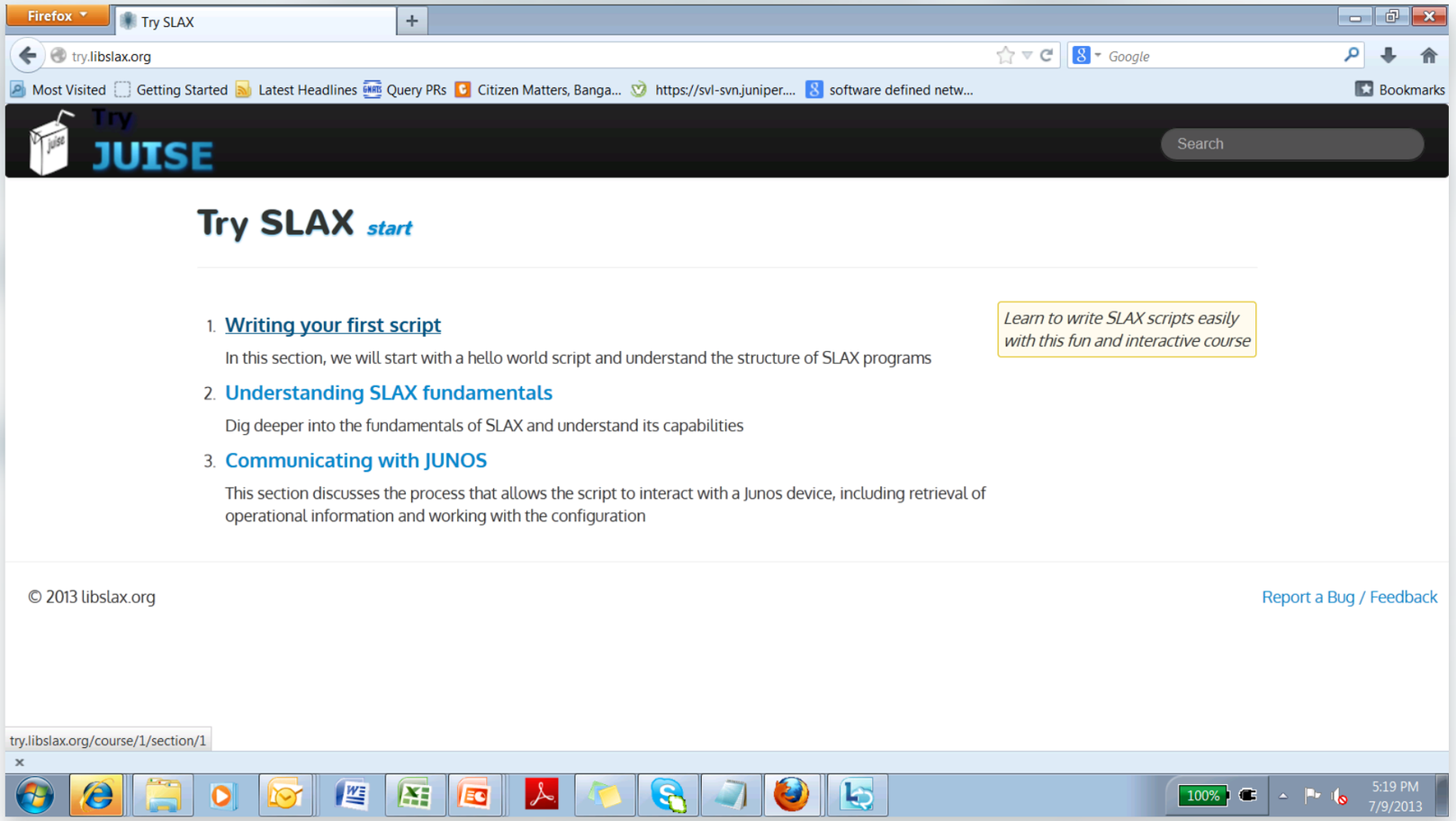
JUISE

- JUNOS UI Scripting Environment
- Open-Source (githib.com/juniper/juise)
- The JUISE project allows scripts to be written, debugged, and executed outside the JUNOS environment
- Tools for developers are available, including a debugger, a profiler, a call-flow tracing mechanism, and trace files

```
% juise @my-router my-script name1 val1 name2 val2
```

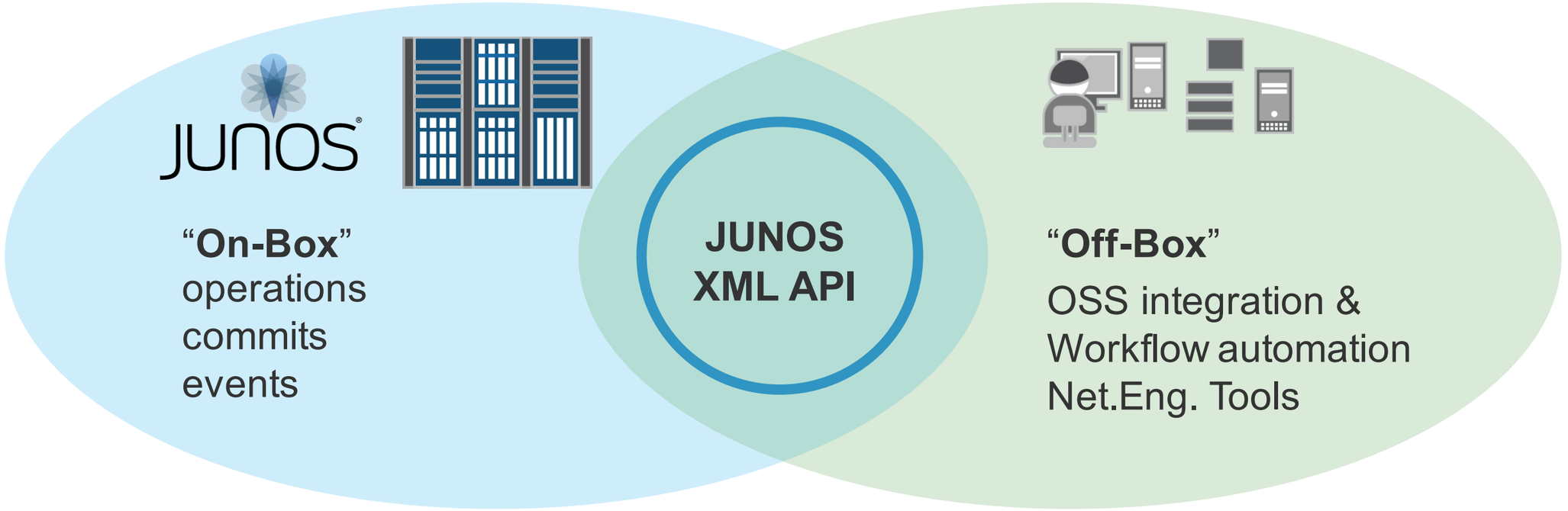
Try SLAX

- http://try.libslax.org
- Interactively learn and try SLAX programming language from web browser



Junos Workflow Automation

Bootstrap your automations with On-Box scripting, complete them with Off-Box



Code Libraries

Programming Language Support

- NETCONF
  - PERL
  - Java
  - Ruby
  - Python
- SLAX
  - JUISE (JUNOS UI Scripting Environment)
- EZ API
  - Python
  - Ruby

Programming Language Support

- Perl
  - Available from Juniper Support Software download:
  - http://www.juniper.net/support/downloads/?p=netconf#sw
- Ruby
  - Available from RubyGems.org, maintained by Juniper Networks
  - gem install netconf
- Python
  - Available from open source project, not affiliated with Juniper Networks
  - https://github.com/vbajpai/ncclient
- Java
  - Available from Juniper Support Software download:
  - http://www.juniper.net/support/downloads/?p=netconf#sw
- SLAX
  - Available from Google code, maintained by Juniper Networks
  - http://github.com/Juniper/libslax/
  - http://github.com/Juniper/juise/

Also all found on http://github.com/Juniper

Introducing "JUNOS PyEZ"

Open and Extensible "micro-framework"

- Remote device management and "fact" gathering
- Troubleshooting, Audit and Reporting
  - Operational data
  - Configuration data
- Configuration Management
  - Unstructured config snippets and templates
  - Structured abstractions
- Generalized utilities for file-system, software-upgrade, secure file copy (scp), etc.
- RPC Meta-Programming

Pyez – a layered approach

Python Shell

interactive

Python script

simple → complex

IT Frameworks

Custom Applications

- Native Python data types (hash/list)
- Junos specific not required
- XML not required

junos-pyez

open-source, Juniper

- Junos specific
- Abstraction Layer
- micro-framework

ncclient

open-source, Community

- NETCONF transport only
- Vendor Agnostic
- No abstractions

Pyez – configuration changes

Unstructured

Structured

Junos config in text, set, or XML format

"snippets" that contain variables

Jinja2 is template engine

"snippets"
(no variables)

"templates"
(merge variables)

Resources

Structured abstractions defined by the junos-ez micro-framework

Juniper + Community

write-only

Junos Configuration

read-write

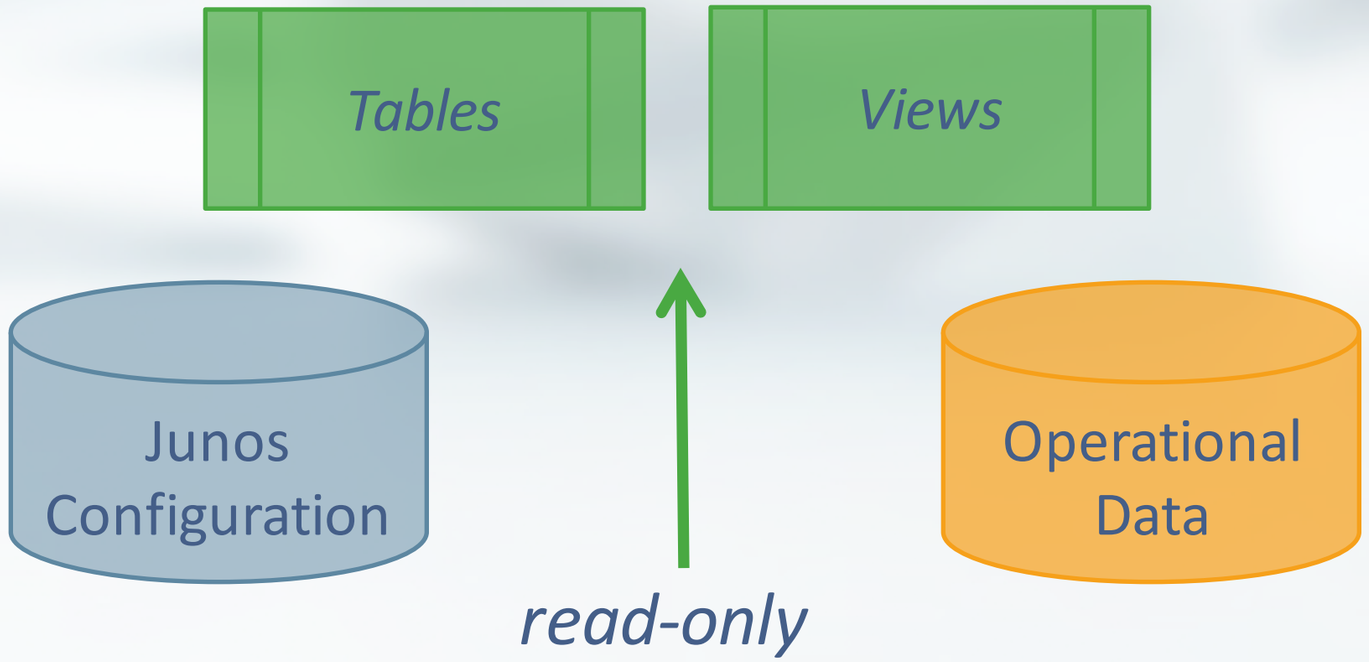
Pyez – troubleshooting, audit, reporting

Easily retrieve data and extract as native Python

Conceptually like database tables and views that define the fields of data you want

Structured abstractions defined by the Junos PyEZ micro-framework

No coding required to create abstractions



Juniper + Community

Pyez – “Hello world”

```
from jnpr.junos import Device
dev = Device(host='srx', user='eddie', passwd='password123')
dev.open()
Device(srx)
dev.facts
{'domain': None, 'hostname': 'firefly', 'ifd_style': 'CLASSIC', 'version_info':
junos.version_info(major=(12, 1), type=X, minor=(46, 'D', 15), build=3), '2RE': False,
'serialnumber': 'aaf5fe5f9b88', 'fqdn': 'firefly', 'virtual': True, 'switch_style': 'NONE', 'version':
'12.1X46-D15.3', 'HOME': '/cf/var/home/rick', 'srx_cluster': False, 'model': 'FIREFLY-PERIMETER',
'RE0': {'status': 'Testing', 'last_reboot_reason': 'Router rebooted after a normal shutdown.', 'model':
'FIREFLY-PERIMETER RE', 'up_time': '8 minutes, 51 seconds'}, 'personality': 'SRX_BRANCH'}
dev.close()
```

Pyez – Connecting To devices

- PyEZ supports connecting to Junos devices using standard user/pass, SSH keys, Agents and Forwarding.

"I am Vinz, Vinz Clortho, Keymaster of Gozer...Volguus Zildrohoar, Lord of the Seboullia. Are you the Gatekeeper?"

Pyez – Connect examples

Username and Password

```
dev = Device(host='srx', user='eddie', passwd='password123')
```

Host, user, and passwd

Connecting to a device via the SSH Agent or Actively loaded environment key

```
dev = Device('srx')
```

No user, key, or password
host keyword can also be omitted

Connecting to a device via the default SSH key + Key password

```
dev = Device(host='srx', passwd='juniper')
```

Host and passwd only

Pyez – Connect examples cont.

Keyfile + password

```
dev = Device(host='srx', ssh_private_key_file='/keys/pkey',
 passwd='juniper')
```

Host, key file and password

Keyfile w/out password

```
dev = Device(host='srx',
    ssh_private_key_file='/keys/nopass‘)
```

Host and key file only

Pyez – disable facts

- The automatic gathering of facts can be disabled. This is useful in situations where bugs or missing device types prevent facts from gathering.

```
dev = Device(host='srx', gather_facts=False)
dev.open()
Device(srx)
dev.facts
{}
```

Pyez – check connection (Probe)

- To check if NETCONF can be reached the device can be probed. This is particularly useful in re-connecting after a reboot.

Default timeout of 5 seconds

- dev.probe()
- False
- dev.probe(timeout=30)
- False

Specify timeout

- dev = Device(host='srx', auto_probe=5)
- dev.open()
- jnpr.junos.exception.ProbeError: ProbeError(srx)

Probe timeout before opening

Pyez – Config utilities

- Included in the framework are various utilities. We will specifically cover some of the more common Configuration functions.
  - Load
  - Pdiff
  - Commit_check
  - Commit
  - Rollback

Pyez – load

- Loads changes into the candidate configuration.
- Changes can be in the form of strings (text, set, xml), XML objects, and files.
- Files can be either static snippets of configuration or Jinja2 templates.

load() does **NOT** commit the change, it only loads it.
To commit changes use **commit()**

Set is not supported in JUNOS < 11.4

Pyez – config set

```
from jnpr.junos import Device
from jnpr.junos.utils.config import Config
dev = Device('srx')
dev.open()
set_commands = """
set system host-name myRouter
set system domain-name shermdog.com
"""
cu = Config(dev)
cu.load(set_commands, format='set')
<Element load-configuration-results at 0xb5f2048c>
```

Import Config util

Set commands in multi-line string

Create Config object

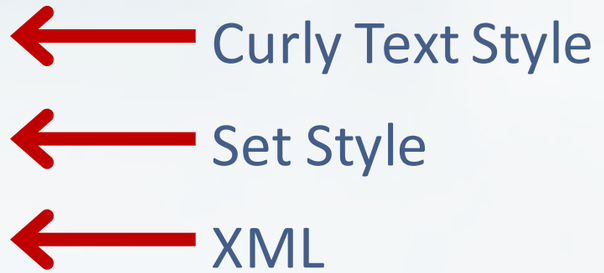
Load configuration onto device

Pyez – config from file

- Configuration data can be loaded from a file on the local system.
- The file extension can be used to determine the format, or it can be specified.

```
cu.load(path='config-example.conf')
cu.load(path='config-example.set')
cu.load(path='config-example.xml')
```


```
cu.load(path='config-example', format='text')
cu.load(path='config-example', format='set')
cu.load(path='config-example', format='xml')
```



Pyez – config from template

- The framework can render templates and load them.
- Variables can be stored in Python dictionary or loaded from a file.

```
tvars = dict(host_name='rick',
    domain_name='shermdog.com')
```

Data for variables

```
cu.load(
    template_path="config-example-template.conf",
    template_vars=tvars)
```

Path to template and vars

Pyez – pdiff / diff

- After a configuration has been loaded the diff (patch-format) between the current candidate and the provided rollback can be retrieved.

- pdiff automatically calls *print* for debugging

cu.pdiff()

By default compare rollback 0

```
[edit system]
-  host-name firefly;
+  host-name rick;
+  domain-name shermdog.com;
```

Pyez – pdiff / diff cont.

- cu.pdiff(3) ← A rollback id can be used to compare different rollbacks

```
[edit system]
-  host-name firefly;
+   host-name rick;
+   domain-name shermdog.com;
+   time-zone America/Chicago;
```

- The diff can also be stored into an object using diff()

```
diff = cu.diff()
```

Pyez – rollback

- The candidate config can be rolled back to either the last active config or a specific rollback number.

```
cu.rollback()
  True
cu.pdiff()
  None
cu.rollback(1)
  True
cu.pdiff()
[edit system]
-  time-zone America/Chicago;
```

By default, roll back to last active - 0

This shows the config *is* rolled back

A rollback id can also be used

Pyez – commit_check & commit

- Configuration changes can be checked prior to commit

```
cu.commit_check()
  True
```

If the commit-check results in warnings, they are not reported (at this time). A **jnpr.junos.exception.CommitError** is thrown

```
cu.commit ()
  True
cu.commit(comment="Doc",
    confirm=2)
True
```

← Comments and confirm timeout can be set

Demo

Documentation:
http://www.juniper.net/documentation/en_US/junos13.2/information-products/pathway-pages/netconf-guide/netconf.html

http://github.com/Juniper