This tutorial was supported in part by the EC IST-EMANICS Network of Excellence (#26854).
NETCONF is a network management protocol specifically designed to support configuration management. It provides the following features:

- distinction between configuration and state data
- multiple configuration datastores (running, startup, . . .)
- support for configuration change transactions
- configuration testing and validation support
- selective data retrieval with filtering
- streaming and playback of event notifications
- extensible remote procedure call mechanism
YANG is a data modeling language designed to write (configuration) data models for the NETCONF protocol. It provides the following features:

- human readable easy to learn representation
- hierarchical configuration data models
- reusable types and groupings
- extensibility through augmentation mechanisms
- supports the definition of operations (RPCs)
- formal constraints for configuration validation
- data model modularity through features
- versioning rules and development support
- translations to XSD, RelaxNG and YIN
## NETCONF and YANG Timeline

<table>
<thead>
<tr>
<th>Year</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun 2002</td>
<td>IAB network management workshop</td>
</tr>
<tr>
<td>May 2003</td>
<td>NETCONF WG established</td>
</tr>
<tr>
<td>Dec 2006</td>
<td>NETCONF core RFCs published</td>
</tr>
<tr>
<td>2007</td>
<td>YANG design team creates YANG proposal</td>
</tr>
<tr>
<td>Apr 2008</td>
<td>NETMOD WG established</td>
</tr>
<tr>
<td>2009</td>
<td>YANG core RFCs published</td>
</tr>
</tbody>
</table>

## IETF WGs considering YANG
- IP Flow Information Export (IPFIX)
- . . .
The one and only most popular NETCONF figure...
STOP
$ ssh -s broccoli netconf
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:confirmed-commit:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>http://tail-f.com/ns/aaa/1.1</capability>
<capability>http://tail-f.com/ns/execd/1.1</capability>
</capabilities>
<session-id>123</session-id></hello>]]>]]>

<?xml version="1.0" encoding="UTF-8"?>
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
</capabilities>
</hello>]]>]]>
#!/usr/bin/env python2.6
#
# Connect to the NETCONF server passed on the command line and
# display their capabilities. This script and the following scripts
# all assume that the user calling the script is known by the server
# and that suitable SSH keys are in place. For brevity and clarity
# of the examples, we omit proper exception handling.
#
# $ ./nc01.py broccoli

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user):
    with manager.connect(host=host, port=22, username=user) as m:
        for c in m.server_capabilities:
            print c

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"))
#! /usr/bin/env python2.6
#
# Retrieve the running config from the NETCONF server passed on the
# command line using get-config and write the XML configs to files.
#
# $ ./nc02.py broccoli

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user):
    with manager.connect(host=host, port=22, username=user) as m:
        c = m.get_config(source='running').data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"))
#!/usr/bin/env python2.6
#
# Retrieve a portion selected by an XPATH expression from the running
# config from the NETCONF server passed on the command line using
# get-config and write the XML configs to files.
#
# $ ./nc03.py broccoli "aaa/authentication/users/user[name='schoenw']"

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, expr):
    with manager.connect(host=host, port=22, username=user) as m:
        assert(":xpath" in m.server_capabilities)
        c = m.get_config(source='running', filter=('xpath', expr)).data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2])
CONTINUE
NETCONF Operations

- get-config(source, filter)
  Retrieve a (filtered subset of a) configuration from the configuration datastore source.

- edit-config(target, default-operation, test-option, error-option, config)
  Edit the target configuration datastore by merging, replacing, creating, or deleting new config elements.

- copy-config(target, source)
  Copy the content of the configuration datastore source to the configuration datastore target.

- delete-config(target)
  Delete the named configuration datastore target.
lock(target)
Lock the configuration datastore target.

unlock(target)
Unlock the configuration datastore target.

get(filter)
Retrieve (a filtered subset of a) the running configuration and device state information.

close-session()
Gracefully close the current session.

close-session(session)
Force the termination of the session session.
NETCONF Operations (cont.)

- **discard-changes()**
  Revert the candidate configuration datastore to the running configuration (:candidate capability).

- **validate(source)**
  Validate the contents of the configuration datastore source (:validate capability).

- **commit(confirmed, confirm-timeout)**
  Commit candidate configuration datastore to the running configuration (:candidate capability).

- **create-subscription(stream, filter, start, stop)**
  Subscribe to a notification stream with a given filter and the start and stop times.
## Editing Configuration

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>merge</strong></td>
<td>The configuration data is merged with the configuration at the corresponding level in the configuration datastore.</td>
</tr>
<tr>
<td><strong>replace</strong></td>
<td>The configuration data replaces any related configuration in the configuration datastore identified by the target parameter.</td>
</tr>
<tr>
<td><strong>create</strong></td>
<td>The configuration data is added to the configuration if and only if the configuration data does not already exist.</td>
</tr>
<tr>
<td><strong>delete</strong></td>
<td>The configuration data identified by the element containing this attribute is deleted in the configuration datastore.</td>
</tr>
</tbody>
</table>
STOP
#!/usr/bin/env python2.6
#
# Create a new user to the running configuration using edit-config
# and the test-option provided by the :validate capability.
#
# $ ./nc04.py broccoli bob 42 42

demo(sys.argv[1], os.getenv("USER"), sys.argv[2], sys.argv[3], sys.argv[4])
#!/usr/bin/env python2.6
#
# Delete an existing user from the running configuration using
# edit-config and the test-option provided by the :validate
# capability.
#
# $ ./nc05.py broccoli bob

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, name):
    snippet = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
            <authentication>
                <users>
                    <user xc:operation="delete">
                        <name>%s</name>
                    </user>
                </users>
            </authentication>
        </aaa>
    </config>""" % name

    with manager.connect(host=host, port=22, username=user) as m:
        assert(":validate" in m.server_capabilities)
        m.edit_config(target='running', config=snippet,
                      test_option='test-then-set')

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2])
CONTINUE
A configuration datastore is the complete set of configuration information that is required to get a device from its initial default state into a desired operational state.

- The `<running>` configuration datastore represents the currently active configuration of a device and is always present.
- The `<startup>` configuration datastore represents the configuration that will be used during the next startup.
- The `<candidate>` configuration datastore represents a configuration that may become a `<running>` configuration through an explicit commit.
Some operations (edit-config) may support different error behaviours, including rollback behaviour.
STOP
#!/usr/bin/env python2.6
#
# Delete a list of existing users from the running configuration using
# edit-config; protect the transaction using a lock.
#
# $ ./nc06.py broccoli bob alice

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

template = """"<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
        <authentication>
            <users>
                <user xc:operation="delete">
                    <name>%s</name>
                </user>
            </users>
        </authentication>
    </aaa>
</config>"""

def demo(host, user, names):
    with manager.connect(host=host, port=22, username=user) as m:
        with m.locked(target='running'):
            for n in names:
                m.edit_config(target='running', config=template % n)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2:])
#!/usr/bin/env python2.6
#
# Delete a list of existing users from the running configuration using
# edit-config and the candidate datastore protected by a lock.
#
# $ ./nc07.py broccoli bob alice

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

template = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
        <authentication> <users> <user xc:operation="delete">
            <name>%s</name> </user></users></authentication></aaa></config>""

def demo(host, user, names):
    with manager.connect(host=host, port=22, username=user) as m:
        assert(":candidate" in m.server_capabilities)
        m.discard_changes()
        with m.locked(target='candidate'):
            for n in names:
                m.edit_config(target='candidate', config=template % n)
                m.commit()

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2:])
CONTINUE
## NETCONF Implementations

### Commercial Toolkits (not necessarily complete)

<table>
<thead>
<tr>
<th>Toolkits</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netconf Central</td>
<td><a href="http://www.netconfcentral.org/">http://www.netconfcentral.org/</a></td>
</tr>
<tr>
<td>Tail-f (ConfD)</td>
<td><a href="http://www.tail-f.com/">http://www.tail-f.com/</a></td>
</tr>
</tbody>
</table>

### Device Vendors (not necessarily complete)

<table>
<thead>
<tr>
<th>Vendors</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juniper Networks</td>
<td><a href="http://www.juniper.net/">http://www.juniper.net/</a></td>
</tr>
</tbody>
</table>
## NETCONF Implementations (cont.)

### Open Source (not necessarily complete)

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
</table>
YANG’s purpose

YANG is an extensible NETCONF data modeling language able to model configuration data, state data, operations, and notifications. YANG definitions directly map to XML content.

YANG vs. YIN

YANG uses a compact SMI-like syntax since readability is highest priority. YIN is an XML version of YANG (lossless roundtrip conversion).

YANG vs. XSD or RELAX NG

YANG can be translated to XML Schema (XSD) and RELAX NG so that existing tools can be utilized.
Built-in Data Types

<table>
<thead>
<tr>
<th>Category</th>
<th>Types</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integral</td>
<td>{u,}int{8,16,32,64}</td>
<td>range</td>
</tr>
<tr>
<td>Decimals</td>
<td>decimal64</td>
<td>range, fraction-digits</td>
</tr>
<tr>
<td>String</td>
<td>string</td>
<td>length, pattern</td>
</tr>
<tr>
<td>Enumeration</td>
<td>enumeration</td>
<td>enum</td>
</tr>
<tr>
<td>Bool and Bits</td>
<td>boolean, bits</td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td>binary</td>
<td>length</td>
</tr>
<tr>
<td>References</td>
<td>leafref</td>
<td>path</td>
</tr>
<tr>
<td>References</td>
<td>identityref</td>
<td>base</td>
</tr>
<tr>
<td>References</td>
<td>instance-identifier</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>empty</td>
<td></td>
</tr>
</tbody>
</table>

Type system

The data type system is mostly an extension of the SMIng type system, accommodating XML and XSD requirements.
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leaf</td>
<td>A leaf has one value, no children, one instance.</td>
</tr>
<tr>
<td>leaf-list</td>
<td>A leaf-list has one value, no children, multiple instances.</td>
</tr>
<tr>
<td>container</td>
<td>A container has no value, holds related children, has one instance.</td>
</tr>
<tr>
<td>list</td>
<td>A list has no value, holds related children, has multiple instances, has a key property.</td>
</tr>
</tbody>
</table>
STOP
module jacobs-fake-aaa-module {

    namespace "http://tail-f.com/ns/aaa/1.1";
    prefix aaa;

    organization
        "Jacobs University Bremen";

    contact
        "Juergen Schoenwaelder";

    description
        "This module contains a fake YANG module for some tail-f data models and it should only be used for educational purposes."

    revision 2009-07-30 {
        description "Initial revision."
    }

    feature ssh-keys {
        description
            "This feature indicate the support of SSH key storage."
    }

    // ...
}

container aaa {
  container users {
    list user {
      key "name";

      leaf name {
        type string {
          pattern "[a-zA-Z0-9]+";
        }
        description "The name of an account on the system. Note that the name root is often associated with special privileges.";
      }

      leaf uid {
        type uint32;
        mandatory true;
        description "The id used by the system to identify a user.";
      }

      leaf gid {
        type uint32;
        mandatory true;
        description "The id used by the system to identify the user’s group.";
      }
    }
  }
}
leaf password {
  type hashed-password;
  description
    "The hashed password of a user. The special value * means no access to the system.";
}
leaf ssh_keydir {
  type string;
  if-feature ssh-keys;
  description
    "The storage location of SSH keys."
}
leaf homedir {
  type string;
  default "/";
  description
    "The home directory of the user.";
}
CONTINUE
**augment**

The `augment` statement can be used to place nodes into an existing hierarchy using the current module’s namespace.

**must**

The `must` statement can be used to express constraints (in the form of XPATH expressions) that must be satisfied by a valid configuration.

**when**

The `when` statement can be used to define sparse augmentations where nodes are only added when a condition (expressed in the form of an XPATH expression) is true.
grouping

A grouping is a reusable collection of nodes. The grouping mechanism can be used to emulate structured data types or objects. A grouping can be refined when it is used.

choice

A choice allows one alternative of the choice to exist. The choice mechanism can be used to provide extensibility hooks that can be exploited using augments.

- Should a grouping be considered a template mechanism or a structured data type mechanism?
Notifications and Operations

**notification**

The `notification` statement can be used to define the contents of notifications.

**rpc**

The `rpc` statement can be used to define operations together with their input and output parameters carried over the RPC protocol.
# YANG Implementations

## Commercial Toolkits (not necessarily complete)

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netconf Central</td>
<td><a href="http://www.netconfcentral.org/">http://www.netconfcentral.org/</a></td>
</tr>
<tr>
<td>SNMP Research</td>
<td><a href="http://www.snmp.com/">http://www.snmp.com/</a></td>
</tr>
<tr>
<td>Tail-f (ConfD)</td>
<td><a href="http://www.tail-f.com/">http://www.tail-f.com/</a></td>
</tr>
</tbody>
</table>

## Open Source (not necessarily complete)

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>jYang</td>
<td><a href="http://jyang.gforge.inria.fr/jYang/">http://jyang.gforge.inria.fr/jYang/</a></td>
</tr>
<tr>
<td>libsmi</td>
<td><a href="http://www.ibr.cs.tu-bs.de/projects/libsmi/">http://www.ibr.cs.tu-bs.de/projects/libsmi/</a></td>
</tr>
<tr>
<td>pyang</td>
<td><a href="http://code.google.com/p/pyang">http://code.google.com/p/pyang</a></td>
</tr>
</tbody>
</table>

## Utilities

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yang.el</td>
<td>emacs mode for editing yang</td>
</tr>
</tbody>
</table>
**Final Words...**

**Acknowledgements**

- Martin Björklund, Tail-f
- Phil Shafer, Juniper Networks
- Andy Bierman, Netconf Central
- Shikhar Bhushan (ncclient), Jacobs University
- Siarhei Kuryla (yang for libsmi), Jacobs University
- Ha Manh Tran (netconf testing), Jacobs University

**Disclaimer**

All errors on the slides are mine.