Getting to Know NETCONF

NETCONF is an XML-RPC based API. The current RFC 6241 specifies the RFC standard. However you can also see the original RFC 4741 as well to see where it all started.

NETCONF Introduction via the RFC

Quoted from RFC 6241 Introduction

The NETCONF protocol defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated. The protocol allows the device to expose a full, formal application programming interface (API). Applications can use this straightforward API to send and receive full and partial configuration data sets.

The NETCONF protocol uses a remote procedure call (RPC) paradigm. A client encodes an RPC in extensible markup language (XML) (TODO is this suppose to link to http://www.w3.org/TR/2000/REC-xml-20001006) [W3C.REC-xml-20001006] and sends it to a server using a secure, connection-oriented session. The server responds with a reply encoded in XML. The contents of both the request and the response are fully described in XML document type definitions (DTDs) or XML schemas, or both, allowing both parties to recognize the syntax constraints imposed on the exchange.

A key aspect of NETCONF is that it allows the functionality of the management protocol to closely mirror the native functionality of the device. This reduces implementation costs and allows timely access to new features. In addition, applications can access both the syntactic and semantic content of the device's native user interface.

NETCONF allows a client to discover the set of protocol extensions supported by a server. These "capabilities" permit the client to adjust its behavior to take advantage of the features exposed by the device. The capability definitions can be easily extended in a noncentralized manner. Standard and non-standard capabilities can be defined with semantic (meaning of expressions, statements, and program units) and syntactic (form of expessions, statements, and program units) rigor. Capabilities are discussed in Section 8.

The NETCONF protocol is a building block in a system of automated configuration.

XML is the lingua franca of interchange, providing a flexible but fully specified encoding mechanism for hierarchical content. NETCONF can be used in concert with XML-based transformation technologies, such as XSLT TODO : should a link be added? [W3C.REC-xslt-19991116], to provide a system for automated generation of full and partial configurations. The system can query one or more databases for data about networking topologies, links, policies, customers, and services. This data can be transformed using one or more XSLT scripts from a task-oriented, vendor-independent data schema into a form that is specific to the vendor, product, operating system, and software release. The resulting data can be passed to the device using the NETCONF protocol.

What does this all mean?

NETCONF was designed to achieve a few specific goals

- To closely mirror the native functionallity of the device

This keeps the way we interact with the device close to how the device is configured. This simplifies the API usage because the way you interact with it is the same as doing the native configuration

- To use an extensible interchange language that can be easily manipulated

Computers do a horrible job of interpreting data that does not have context. XML was chosen as the interchange format for NETCONF. It allows for data exchange in a way that makes sense to a computer and it can easily be parsed. XML is highly extensible and does a great job of representing the data so it can be managed.

- Secure access to the API

NETCONF requires secure access to the API as defined in RFC 6241 Section 2. The goal is the communication must be authenticated, have data integrity, confidentiality, and replay protection. Junos uses SSH as this transport mechanism. NETCONF while it can support other protocols such as TLS or BEEP it MUST support SSH as per RFC 6241 section 2.3.

Interacting with NETCONF manually

Canonical Data Terminator

The character set "]]>]]>" is used to specify the end of a NETCONF message. This is used to tell the host that the message is completed. While some protocols such as HTTP use content length for this, by defining a specific terminator that is all that an application needs to look for at the end of a message. This is typically done by looking for this set of bytes as "5D 5D 3E 5D 5D 3E". Because XML can contain new lines, tabs, and carriage returns this byte set terminator allows the usage of these typical end elements to be included freely in the XML sent to the user. You can read more here at the Juniper documentation NETCONF end of document.

Connecting to NETCONF

Connecting to NETCONF host can easily be done from your NetDevOps VM.

- Connect into your host with "vagrant ssh ndo"
- Run the command "ssh root@172.16.0.1 -s netconf" to request an SSH session with the device. The "-s netconf" flag is used to specify the NETCONF subsystem.
    - We can also connect to the NETCONF port directly using "ssh root@172.16.0.1 -p 830 -s netconf" by specifying the port of 830
- Use the root password of "Juniper" to connect to the device

```
ssh root@172.16.0.1 -s netconf
```

This should connect you to the NETCONF service on the vSRX. Once connected you will see it spit out a bunch of XML back to you. We now are bound to a socket connection that we can freely send and receive data over.

Each NETCONF session is given a session ID. This is used to tell you which process id or PID is being used on the server. If you have multiple NETCONF sessions active it will help you determine which one you are connected to. The session ID is included as part of the hello that is part of the initial connection.

Response

```
<!-- No zombies were killed during the creation of this user interface -->
<!-- user root, class super-user -->
<hello>
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
    <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0</ca
    <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-commit:
    <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0</cap
    <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?protocol=
    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>1985</session-id>
</hello>
]]>]]>
```

The hello response also includes a set of capabilities on the device. These included URLs are used as part of the namespace for the document. These aren't active websites that are used.

Hello RPC

Once connected we need to send back a hello message to the the NETCONF server. On Junos this is not required to be sent. However the best practice is to send this information. Also in the event that we DO enforce it in the future, it is better to be compliant to the specification of the protocol.

Request

```
<hello>
    <capabilities>
        <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
        <capability>urn:ietf:params:xml:ns:netconf:capability:candidate:1.0
        <capability>urn:ietf:params:xml:ns:netconf:capability:confirmed-com
        <capability>urn:ietf:params:xml:ns:netconf:capability:validate:1.0<
        <capability>urn:ietf:params:xml:ns:netconf:capability:url:1.0?proto
        <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
        <capability>http://xml.juniper.net/dmi/system/1.0</capability>
    </capabilities>
</hello>
]]>]]>
```

Response

```
Junos will not send a response
```

Message ID

When sending RPC messages it is possible to include a message-id in the request. When the response is returned it will allow you to match the response to the request. This is hugely helpful for when you need to troubleshoot a connection in which you are sending multiple messages at the same time. It helps you map a response to a request. This is optional and it is up to the implementor to use.

Gathering some information

Now that we have an active NETCONF connection to the device we can start sending commands to the host.

Request

```
<rpc message-id="1">
    <get-software-information/>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <software-information>
        <host-name>NetDevOps-SRX01</host-name>
        <product-model>firefly-perimeter</product-model>
        <product-name>firefly-perimeter</product-name>
        <package-information>
            <name>junos</name>
            <comment>JUNOS Software Release [12.1X47-D10.4]</comment>
        </package-information>
    </software-information>
</rpc-reply>
]]>]]>
```

Adding to the configuration via a manual RPC

Now we are going to make a simple configuration change. We do this just as we would from the command line: Open configuration, load change, and commit. For our example we will simply change the hostname.

- Open Configuration
- Load Configuration
- Commit Configuration

Opening the configuration

To change the configuration we first must open up a candidate configuration. This is just as if you were to type "edit" or "configure" from the CLI.

Request

```
<rpc message-id="2">
  <open-configuration>
  </open-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
</rpc-reply>
]]>]]>
```

Loading a configuration

Now that the configuration is open we will load our configuration in the standard XML format that is expected. We can also load other formats such as "set commands" or "Junos config". But to best demonstrate the API we will use the standard XML formatting.

Request

```
<rpc message-id="3">
    <load-configuration action="merge">
    <configuration>
        <system>
            <host-name>NETCONFED</host-name>
        </system>
    </configuration>
    </load-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <load-configuration-results>
        <ok/>
    </load-configuration-results>
</rpc-reply>
]]>]]>
```

Committing the configuration

We must now commit the configuration just as we would by do by hand. But again in this case we will use the XML formatting.

Request

```
<rpc message-id="4">
    <commit-configuration>
        <log>Committed via NETCONF by hand!</log>
    </commit-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <ok/>
</rpc-reply>
]]>]]>
```

Validating the change

Now we want to validate that our change successfully occured. There are numerous ways to do this, but lets reuse an RPC we did earlier in the lab. We will get the software information and this response also includes the hostname. The best practice would be to look at the diff of configuration or by comparing the entire configuration.

Request

```
<rpc message-id="5">
    <get-software-information/>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <software-information>
        <host-name>NETCONFED</host-name>
        <product-model>firefly-perimeter</product-model>
        <product-name>firefly-perimeter</product-name>
        <package-information>
            <name>junos</name>
            <comment>JUNOS Software Release [12.1X47-D10.4]</comment>
        </package-information>
    </software-information>
</rpc-reply>
]]>]]>
```

Rolling back the change

To keep the lab intact we want to rollback to the previous configuration. This will keep us all on the same page as we progress, plus it gives us an excuse to run a few more RPCs. In this request we are going to rollback to the previous configuration and then commit the change.

Request

```
<rpc message-id="6">
    <get-rollback-information>
        <rollback>1</rollback>
    </get-rollback-information>
</rpc>
```

Response

The response is long and contains the entire configuration. For this example the response has been truncated.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <rollback-information>
        <ok/>
        <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm" junos:cha
            <version>12.1X47-D10.4</version>
            <system>
                <host-name>NetDevOps-SRX01</host-name>
            </system>
            <!-- RESPONSE TRUNCATED -->
        </configuration>
    </rollback-information>
</rpc-reply>
]]>]]>
```

Lastly commit the configuration to apply the rollback.

Request

```
<rpc message-id="7">
    <commit-configuration>
        <log>Committed via NETCONF by hand!</log>
    </commit-configuration>
</rpc>
]]>]]>
```

Response

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="htt
    <ok/>
</rpc-reply>
]]>]]>
```

Review

In this section we reviewed how to make a manual connection to the NETCONF service. This is not ideal to do by hand. Humans aren't very good at parsing XML data by hand. This is why the best way to interact with a Junos device manually is through the CLI. It gives you the exact same capabilities of NETCONF but in a human friendly format. Although you are welcome to use NETCONF in this manner it probably won't be a very efficent way to interact with the device.