Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2017

R. Wilton, Ed.
Cisco Systems

July 6, 2016

Refined YANG datastores with Meta-data
draft-wilton-netconf-opstate-metadata-00

Abstract

draft-wilton-netmod-refined-datastores defines refined YANG datastore definitions to provide an explicit Operational State Datastore and a clean separation between the ’intended configuration’ for a device and the ’applied configuration’ that is in effect on a device. This draft builds on draft-wilton-netmod-refined-datastores by describing a YANG Metadata based extension that can be used by protocols such as NETCONF and RESTCONF to allow the key information from the various operational state related datatstores to be made available without requiring the client to explicitly read and monitor each abstract datastore separately.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes an optional extension to NETCONF/RESTCONF, based on With-defaults Capability for NETCONF [RFC6243], and Metadata with YANG [I-D.ietf-netmod-yang-metadata], that allow servers to communicate the content of the Applied Configuration Abstract Datastore and Operational State Datastore defined in

[I-D.wilton-netmod-refined-datastores] to clients in an efficient way.

Operator requirements state that there needs to be a clear separation between the configuration that has been sent to the device and the actual configuration that the device is actually acting on.

For some simple ’opstate unware’ devices, if it is sufficient to assume that the configuration is applied instantaneously and also that none of the configuration can fail, then the intended and applied configuration can be regarded as being exactly the same, and a formal split between intended and applied configuration is unnecessary.

For other more complex ’opstate aware’ devices, the assumptions above do not always hold: Devices may take several minutes (or even tens of minutes) to apply the configuration, some of the configuration may fail, or it may not be possible to apply some of the configuration due to either insufficient resources or due to a requisite piece of hardware not being present in the device.

This extension is primarily aimed at this latter class of opstate aware devices, but to improve interoperability it is anticipated that it could also fairly easily be implemented on opstate unaware devices as well.

The extension uses YANG Metadata to allow a client to quickly determine whether the configuration has or has not been successfully applied. Semantically, this is similar to performing a "diff" between two related configuration datastores, but with the content of the diff returned relative to one of those datastores.

1.1. Definitions

Definitions of the following terms are taken from [I-D.wilton-netmod-refined-datastores]:

opstate aware device - a device that implements the requirements specified in [I-D.ietf-netmod-opstate-reqs], in particular the device must expose the split between the device’s ’intended configuration’ vs ’applied configuration’.

opstate unaware device - a device that is not an ’opstate aware device’. In particular, it does not draw a clear distinction between ’intended configuration’ vs ’applied configuration’, and generally treats them as having exactly the same contents.

abstract datastore - a new variant of YANG datastore that represents a particular common property of the contents (e.g. ’applied configuration’). Servers could allow it to be external referenced as a named datastore, but generally that is not expected or required.

The following datastore definitions are taken from NETCONF [RFC6241]:

- candidate ds - represents candidate configuration

- startup ds - represents startup configuration

The following datastore definitions are taken from [I-D.wilton-netmod-refined-datastores]:

- Persistent Configuration - holds the client provided configuration that is written to the Startup Datastore and is recovered after device reboot.

- Ephemeral Configuration - an optional datastore that holds client provided transient configuration that is discarded after device reboot.

- Operational State - a read-only datastore that holds all of the operational state of the device. Specifically it holds: the exact configuration that has been applied, along with any system controlled configuration, and all system state (including statistics and any ephemeral state).

- Intended Configuration - abstractly represents the combined desired configuration of the device

- Applied Configuration - abstractly represents the actual applied configuration of the device

- Running Configuration - abstractly represents the combined intended and applied datastores, logically equivalent to the definition given in [RFC6241]

1.2. Objectives

The objectives of this draft are:

to minimize the number of explicit datastores that NETCONF/ RESTCONF servers must implement and clients must manage, whilst providing a clean separation between intended configuration, applied configuration, and operational state.

to provide an efficient mechanism to query and monitor whether all of the intended configuration has been applied, and view any configuration that has not been applied.

to provide an efficient mechanism to query and monitor the operational state of the device.

1.3. Overview of a refined model of datastores

The following diagram, taken from [I-D.wilton-netmod-refined-datastores], illustrates how all of the abstract and concrete datastores (except running) relate to each other:

```
      +-------------+                  +-----------+
      | <candidate> |                  | <startup> |
      |  (ct, rw)   |<---+        +--->| (ct, rw)  |
      +-------------+    |        |    +-----------+
             |           |        |           |
             |      .....|........|........   |
             |      . +-----------------+ .   |
             +------->|<persistent cfg> |<---+
                    . | (ct, rw)        | .
      Intended      . +-----------------+ .
       Config   ==> .          v          .
      Datastore     . +-----------------+ .
      (abstract)    . |<ephemeral cfg>  |<--- Can override persistent
                    . | (ct, rw)        | .   cfg. Optional
                    . +-----------------+ .
                    ..........|............
                              |
                    +---------v-----------+
                    | ................... |
                    | . <applied cfg>   .<--- Actual cfg in effect
     Operational    | . (ct, ro)        . |
        State   ==> | ................... |
      Datastore     |         +           |
                    |    system cfg      <--- System created config
                    |         +           |
                    |    system state    <--- All config false nodes
                    +---------------------+

    Key
    Solid boxes (-----) indicate normal datastores:
      (i.e Startup, Persistent Cfg, Ephemeral Cfg, Operational State)

    Dotted boxes (.....) indicate abstract datastores:
      (i.e. Intended Config and Applied Config)

    ct = config true, rw = read/write, ro = read/only
```

2. Applied Configuration Metadata Option to NETCONF and RESTCONF

The applied configuration metadata option is an optional extension to NETCONF/RESTCONF, loosely based on With-defaults Capability for NETCONF [RFC6243], that uses YANG Metadata [I-D.ietf-netmod-yang-metadata] to annotate the Persistent Configuration, Ephemeral Configuration, Operational State, or Running Configuration Datastores with additional metadata that partially reflects the contents of the abstract datastores illustrated in Section 1.3.

Principally, the metadata annotations allow the client to see whether the configuration has been applied or failed; the reason for the failure if it has failed; or whether the configuration node only exists because it has been created by the system. This is achieved without the client having to query or explicitly monitor any of the extra abstract datastores.

2.1. 'with-applied-config-metadata' modes query parameter

The 'with-applied-config-metadata' option supports further refinement of the nodes and metadata that is returned through a query parameter that supports three modes.

2.1.1. selectively-annotate

For a given NETCONF/RESTCONF query, the 'selectively-annotate' option returns all the nodes what would have been returned for that query anyway, but with additional metadata annotations for any nodes in the request datastore that differ from the equivalent node in the applied configuration datastore (i.e. it would exclude metadata annotations for nodes that would be reported as "cfg-status=applied").

This is the default option and MUST be supported by all servers implementing the with-applied-config option.

This option allows a client to both monitor the exact configuration that the device is trying to converge to, and also the status of whether that configuration has been applied. This is achieved in a way that uses a single request, and in the context of a single datastore. In the case that the applied configuration exactly matches the intended configuration then clearly no additional metadata annotations are returned.

2.1.2. annotate-all

For a given NETCONF/RESTCONF query, the 'annotate-all' option returns all the nodes what would have been returned for that query anyway, but with opstate metadata annotations for all nodes that are returned. I.e. this is the same data that would be returned using the 'selectively-annotate' option except that it also includes metadata for nodes that are reported as "cfg-status=applied".

This option MAY be supported by servers implementing the opstate metadata option. It may be useful for clients that prefer explicit applied configuration annotations for every node.

2.1.3. annotated-diff

For a given NETCONF/RESTCONF query, the 'annotated-diff' option filters the nodes returned in the standard query, so that only nodes that are not reported as "cfg-status=applied" are returned. Applied configuration metadata annotations are included for all nodes that are returned.

This option makes it easy for clients to determine whether their configuration is fully applied in a concise way and SHOULD be supported by all servers implementing the with-applied-config option.

2.2. Datastore specific handling

The Applied Configuration Metadata could be used with any of following configuration datastores (persistent, ephemeral, intended, applied, running), or the Operational State Datastore. However, not all values of the "cfg-status" make sense for all datastores, so the values that may be used for particular datastores are restricted as follows:

Persistent Cfg Ds - applying, applied, applied-deviation, overridden, blocked, failed

Ephemeral Cfg Ds - applying, applied, applied-deviation, overridden, blocked, failed

Intended Cfg Ds - applying, applied, applied-deviation, blocked, failed

Applied Cfg Ds - applying, applied, applied-deviation, failed

Operational State Ds - applying, applied, applied-deviation, system-controlled, failed

If the Running Configuration Datastore is handled as described in [I-D.wilton-netmod-refined-datastores], then NETCONF <get> requests are handled as if they are against the Operational State Datastore, and <get-config> requests are handled as if they are made against the Intended Configuration Datastore, which indicates which metadata values can be used.

2.3. Applied Configuration Metadata YANG Definition

Defines the applied configuration datastore metadata that is used in both NETCONF and RESTCONF

```
   <CODE BEGINS> file "ietf-yang-opstate-metadata@2016-07-06.yang"
   module ietf-yang-opstate-metadata {
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-yang-opstate-metadata";

     prefix "opstate";

     import ietf-yang-metadata {
       prefix "md";
     }

     organization
       "IETF NETCONF (Network Configuration Protocol) Working Group";

     contact
       "WG Web:   <http://tools.ietf.org/wg/netconf/>

        WG List:  <netconf@ietf.org>

        WG Chair: Mahesh Jethanandani
                  <mjethanandani@gmail.com>

        WG Chair: Mehmet Ersue
                  <mehmet.ersue@nsn.com>

        Editor:   Robert Wilton
                  <rwilton@cisco.com>";

     description
       "This module defines YANG metadata to allow the reason why
        a config true node exists in the operational state datastore
        to be determined using YANG metadata.

        Copyright (c) 2016 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust’s Legal Provisions
        Relating to IETF Documents
         (http://trustee.ietf.org/license-info).

        This version of this YANG module is part of
        draft-wilton-netconf-opstate-metadata-00; see the Internet
        draft itself for full legal notices.";

     revision 2016-07-06 {
       description "Initial revision";

       reference
         "Internet draft: draft-wilton-netconf-opstate-metadata-00";
     }

     md:annotation cfg-status {
       type enumeration {
         enum applying {
           description
             "The configuration for the annotated node is currently
              changing (i.e. being created, deleted or changing in
              value) as part of an ongoing configuration operation";
         }
         enum applied {
           description
             "The configuration is fully applied.  The node exists in
              both the intended and applied datastores and has exactly
              the same value in both.";
         }
         enum applied-deviation {
           description
             "The configuration has been applied to the extend the
              server is able to, but the value in the applied
              configuration datastore does not exactly match the value
              in the intended configuration datastore.";
         }
         enum overridden {
           description
             "The configuration node value has been overridden by the
              same node in another configuration datastore.";
         }
         enum system-controlled {
           description
             "The configuration node only exists in the Operational
              State Datastore because it is system controlled. It is
              not present in the abstract applied configuration
              datastore.";
         }
         enum blocked {
           description
             "The system cannot apply the configuration because
              the required hardware resources are not present.  The
              configuration node does not exist in the applied
              configuration datastore.";
         }
         enum failed {
           description
```

```
             "The system cannot apply the configuration due to an
              error.  The configuration node does not exist in the
              applied configuration datastore.";
         }
       }
       description
         "Status indicates why a configuration node (i.e. config=true)
          in the operational-state datastore does not match the
          corresponding node in the intended config datastore";
     }

     md:annotation cfg-status-reason {
       when "../status = 'blocked' or ../status = 'failed'" {
         description
           "An optional status reason can be provided for blocked or
            failed configuration";
       }
       type string;
       description
         "Indicates the reason why the applied configuration node is
          blocked or failed";
     }
   }
   <CODE ENDS>
```

2.4. :with-applied-cfg-metadata Capability

The :with-applied-cfg-metadata capability for NETCONF and RESTCONF indicates that the server is capable of returning applied configuration metadata.

2.4.1. Capability Identifier

Equivalent capabilities are defined for both NETCONF and RESTCONF:

NETCONF: urn:ietf:params:netconf:capability:with-applied-cfg-metadata:1.0

RESTCONF: urn:ietf:params:restconf:capability:with-applied-cfg-metadata:1.0

Note that this protocol capability URI is separate from the YANG module capability URI for the YANG module is Section XXX. A server that implements this module MUST also advertise a YANG module capability URI according to the rules specified in [RFC6020].

2.4.1.1. datastore parameter

The identifier MUST have a parameter: "supported-datastores". This indicates which datastores the server allows the :with-applied-cfg-metadata option to be used on.

The datastores that may be supported are described in Section 1.1. The allowed values of this parameter include 'persistent', 'ephemeral', 'opstate', 'running', and any other appropriate configuration datastores supported by the server. Both 'persistent' and 'operational state' datastores MUST be supported.

2.4.1.2. supported-modes parameter

The identifier MUST also have the parameter: "supported-modes". This indicates which particular operations are supported.

Possible modes are 'selectively-annotate', 'annotate-all', and 'annotated-diff' as defined in Section 2.1. The 'selectively-annotate' option MUST be supported, and the 'annotated-diff' option SHOULD be supported.

2.4.1.3. Examples

NETCONF and RESTCONF capability examples:

```
urn:ietf:params:netconf:capability:with-applied-cfg-
metadata:1.0?supported-datastores=persistent,intended&supported-
modes=selectively-annotate

urn:ietf:params:restconf:capability:with-applied-cfg-
metadata:1.0?supported-datastores=running&supported-
modes=selectively-annotate,annotated-diff
```

2.5. NETCONF specifics

Further details for the <with-applied-config-metadata> option need to still be fleshed out here:

> The option would only be supported for NETCONF <get> and <get-config> requests.
>
> NETCONF would need to add support for the new datastores (as least "persistent", "opstate").

2.6. RESTCONF specifics

Further details for the <with-applied-config-metadata> option need to still be fleshed out here:

The option would only be supported for RESTCONF GET requests.

The option could be used when accessing the default combined datastore view, or with new datastore specific REST paths (e.g. as least "persistent", "opstate").

3. Discussion Points

This section lists some points that may warrant further discussion:

The proposal is written in terms of NETCONF/RESTCONF "get" requests but it is desirable if the metadata could also apply to YANG Pub/Sub as well.

Should the extension target adding opstate specific metadata, or is it just applied configuration metadata?

The proposed YANG model merges several different opstate properties into a single ’cfg-status’ leaf, possibly these could be separated out into separate leaves.

One of the aims of the approach described in [I-D.wilton-netmod-refined-datastores] and in this document is to allow opstate unaware servers to fairly easily add basic support for the operational state extensions. This provides an opportunity to improve interoperability with NETCONF/RESTCONF clients because they can treat opstate aware and opstate aware servers in exactly the same way. Does the approach in this draft achieve this goal?

4. Acknowledgements

This document arose through discussions to find a consensual solution to the operational state problem. The following people were actively involved in the discussions that indirectly led to this document:

- Lou Berger, Labn Consulting, <lberger@labn.net>

- Martin Bjorklund, Tail-f Systems, <mbj@tail-f.com>

- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>

- Acee Lindem, Cisco Systems, <acee@cisco.com>

- o Juergen Schoenwaelder, Jacobs University Bremen <j.schoenwaelder@jacobs-university.de>

- o Rob Shakir, Jive Communications, <rjs@rob.sh>

- o Kent Watsen, Juniper Networks, <kwatsen@juniper.net>

The author would also like the thank the following people who have kindly provided feedback on this draft: Matt Hall, Einar Nilsen-Nygaard.

5. IANA Considerations

None

6. Security Considerations

TBD.

7. References

7.1. Normative References

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

7.2. Informative References

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-10 (work in progress), June 2016.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-07 (work in progress), March 2016.

[I-D.schoenw-netmod-revised-datastores]
Schoenwaelder, J., "A Revised Conceptual Model for YANG Datastores", draft-schoenw-netmod-revised-datastores-01 (work in progress), June 2016.

[I-D.wilton-netmod-refined-datastores]
Wilton, R., "Refined YANG datastores", draft-wilton-netmod-refined-datastores-00 (work in progress), June 2016.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <http://www.rfc-editor.org/info/rfc6242>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

Appendix A. Usage examples

A sample encoding of the <with-applied-cfg-metadata> enhancement is described below.

A simple example module is provided to illustrate the subsequent examples. This is not a real module, and is not intended for any real use.

```
module example-interfaces {

  namespace "http://example.com/ns/interfaces";

  prefix exam;

  container interfaces {
    description "Example interfaces group";

    list interface {
      key name;
      description "Example interface entry";
```

```
         leaf name {
           type string {
             length "1 .. max";
           }
           description
             "The administrative name of the interface.";
         }

         leaf mtu {
           type uint32;
           default 1514;
           description
             "The maximum transmission unit (MTU) value assigned to
              this interface.";
         }

         leaf enabled {
           type boolean;
           default "true";
           description "Enable the interface";
         }

         leaf oper-status {
           config false;
           type enumeration {
             enum up {
               description
                 "Ready to pass packets.";
             }
             enum down; {
               description
                   "The interface does not pass any packets.";
             }
           }
         }
       }
     }
   }
```

A.1. NETCONF Persistent datastore get-config request using with-applied-cfg-metadata

This example illustrates a <get-config> request made against the Persistent Configuration Datastore using the with-applied-cfg-metadata selectively-annotate option using the example YANG module above:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <persistent/>
    </source>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-applied-cfg-metadata>
     xmlns="urn:...:ietf-netconf-with-applied-cfg-metadata">
      selectively-annotate
    </with-applied-cfg-metadata>
  </get-config>
</rpc>
```

The response indicates that at the time of the reply:

The system has failed to apply the MTU configuration of 9001 on eth0/0 due to a hardware programming error.

The request to change the MTU leaf on eth0/1 to 9000 is still in the process of being applied.

The MTU configuration on eth0/2 has been successfully applied.

```
   <rpc-reply message-id="101"
              xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
              xmlns:opstate="urn:...:ietf-yang-opstate-metadata">
     <data>
       <interfaces xmlns="http://example.com/ns/interfaces">
         <interface>
           <name>eth0/0</name>
           <mtu opstate:cfg-status="failed"
                opstate:cfg-status-reason="hardware programming error">
             9001
           </mtu>
         </interface>
         <interface>
           <name>eth0/1</name>
           <mtu opstate:cfg-status="applying">
             9000
           </mtu>
         </interface>
         <interface>
           <name>eth0/2</name>
           <mtu>2000</mtu>
         </interface>
       </interfaces>
     </data>
   </rpc-reply>
```

A.2. NETCONF Operational State Datastore get request using with-applied-cfg-metadata

This example illustrates a <get> request made against the Operational State Datastore using the with-applied-cfg-metadata selectively-annotate option using the example YANG module above:

```
   <rpc message-id="102"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get>
       <source>
         <opstate/>
       </source>
       <filter type="subtree">
         <interfaces xmlns="http://example.com/ns/interfaces"/>
       </filter>
       <with-applied-cfg-metadata>
        xmlns="urn:...:ietf-netconf-with-applied-cfg-metadata">
         selectively-annotate
       </with-applied-cfg-metadata>
     </get>
   </rpc>
```

An example response is given below. This response assumes that the device is in the same state as for the <get-config> example given above, and assumes that the device uses NETCONF with-default "explicit" mode. The response indicates that at the time of the reply:

The MTU on eth0/0 is still at the YANG default of 1514. This differs from the intended configuration because the configuration failed to be applied.

The configuration request to change the MTU leaf on eth0/1 from 1514 to 9000 is still in the process of being applied, and the device is still currently using an MTU of 1514.

The MTU configuration of 2000 on eth0/2 has been successfully applied.

eth0/3 has not been configured, but exists in the Operational State Datastore because it is automatically created by the device. The MTU leaf (which has the default value) must also be marked as system-controlled because there is no implicit or explicit MTU leaf in the intended configuration for eth0/3. The enabled leaf is also system-controlled but with a non default value, since the device does not allow the interface to be enabled without being explicitly configured.

```
   <rpc-reply message-id="101"
              xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
              xmlns:opstate="urn:...:ietf-yang-opstate-metadata">
     <data>
       <interfaces xmlns="http://example.com/ns/interfaces">
         <interface>
           <name>eth0/0</name>
           <mtu opstate:cfg-status="failed"
                opstate:cfg-status-reason="hardware programming error">
             1514
           </mtu>
           <enabled>true</enabled>
           <oper-status>up</oper-status>
         </interface>
         <interface>
           <name>eth0/1</name>
           <mtu opstate:cfg-status="applying">
             1514
           </mtu>
           <enabled>true</enabled>
           <oper-status>up</oper-status>
         </interface>
         <interface>
           <name>eth0/2</name>
           <mtu>2000</mtu>
           <enabled>true</enabled>
           <oper-status>up</oper-status>
         </interface>
         <interface>
           <name opstate:cfg-status="system-controlled">
             eth0/3
           </name>
           <mtu opstate:cfg-status="system-controlled">
             1514
           </mtu>
           <enabled opstate:cfg-status="system-controlled">
             false
           </enabled>
           <oper-status>down</oper-status>
         </interface>
       </interfaces>
     </data>
   </rpc-reply>
```

Author's Address

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com