Network Working Group J. Schoenwaelder, Ed.
Internet-Draft Jacobs University
Intended status: Standards Track June 10, 2016
Expires: December 12, 2016

A Revised Conceptual Model for YANG Datastores
draft-schoenw-netmod-revised-datastores-01

Abstract

Datastores are a fundamental concept binding the YANG data modeling language to protocols transporting data defined in YANG data models, such as NETCONF or RESTCONF. This document defines a revised conceptual model of datastores based on the experience gained with the initial simpler model and addressing requirements that were not well supported in the initial model.

Table of Contents

1. Introduction

This document provides a revised architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [I-D.ietf-netconf-restconf] and the YANG [RFC6020] data modeling language. Datastores are a fundamental concept binding management data models to network management protocols and agreement on a common architectural model of datastores ensures that data models can be written in a network management protocol agnostic way. This architectural framework identifies a set of conceptual datastores but it does not mandate that all network management protocols expose all these conceptual datastores. Furthermore, the architecture does not detail how data is encoded by network management protocols.

2. Background

NETCONF [RFC6241] provides the following definitions:

- datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.

- configuration datastore: The datastore holding the complete set of configuration data that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [I-D.ietf-netmod-rfc6020bis] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG did exist):

- datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.

- configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration data.

RFC 6244 defined operational state data as follows:

- Operational state data is a set of data that has been obtained by the system at runtime and influences the system's behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

Section 4.3.3 of RFC 6244 discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as a separate data tree is the recommended approach.

Implementation experience and requests from operators indicate that the datastore model initially designed for NETCONF and refined by YANG needs to be extended. In particular, the notion of intended configuration and applied configuration has developed. Furthermore, separating operational state data from configuration data in a separate branch has been found operationally complicated. The relationship between the branches is not machine readable and filter expressions operating on configuration data and on related operational state data are different.

3. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:

```
     +-------------+                 +-----------+
     | <candidate> |                 | <startup> |
     |  (ct, rw)   |<---+       +--->| (ct, rw)  |
     +-------------+    |       |    +-----------+
            |           |       |           |
            |         +------------+        |
            +-------->| <running>  |<-------+
                      | (ct, rw)   |
                      +------------+
                            |
                            v
            +---------------------------------+
            | operational-state               |<-- control plane
            | (cf, ro)                        |
            +---------------------------------+
```

ct = config true; cf = config false; rw = read-write; ro = read-only

Note that read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for the <candidate> and <startup> datastores is optional and the <running> datastore does not have to be writable. Furthermore, the <startup> datastore can only be modified by copying <running> to <startup> in the standardized NETCONF datastore editing model. The RESTCONF protocol does not expose these differences and instead provides only a writable unified datastore, which hides whether edits are done through a <candidate> datastore or by directly modifying the <running> datastore or via some other implementation specific mechanism. RESTCONF also hides how configuration is made persistent. Note that implementations may also have additional datastores that can propagate changes to the <running> datastore. NETCONF explicitly mentions so called named datastores.

Some observations:

o Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.

o The NETCONF <get/> operation returns the content of the <running/> configuration datastore together with the operational state. It is therefore necessary that config false data is in a different branch than the config true data. This is in particular relevant if operational state data can have a different lifetime compared to configuration data or if configuration data is not immediately or successfully applied.

- Several implementations have proprietary mechanisms that allow clients to store inactive data in the <running> datastore; this inactive data is only exposed to clients that indicate that they support the concept of inactive data; clients not indicating support for inactive data receive the content of the <running> datastore with the inactive data removed. Validation always happens on the <running> datastore with inactive data removed.

- Some operators have reported that it is essential for them to be able to retrieve the configuration that has actually been successfully applied, which may be a subset or a superset of the <running> configuration.

4. Revised Model of Datastores

Below is a new conceptual model of datastores extending the original model in order reflect the experience gained with the original model.

```
  +-------------+                  +-----------+
  | <candidate> |                  | <startup> |
  |  (ct, rw)   |<---+        +--->| (ct, rw)  |
  +-------------+    |        |    +-----------+
         |           |        |           |
         |         +------------+         |
         +-------->| <running>  |<--------+
                   | (ct, rw)   |
                   +------------+
                         |
                        (A)       // e.g., removal of 'inactive' nodes
                         v
                   +------------+
                   | <intended> | // subject to validation
                   | (ct, ro)   |
                   +------------+
                         |
                        (B)       // e.g., missing resources or delays
                         v
                   +------------+
                   | <applied>  |
                   | (ct, ro)   |
                   +------------+
                         |
                        (C)       // e.g., autodiscovery, control-plane
                         |        // protocols, control-plane datastores
                         v
         +--------------------------------+
         | <operational-state>            |
         | (ct + cf, ro)                  |
         +--------------------------------+
```

ct = config true; cf = config false; rw = read-write; ro = read-only

The main changes are:

- The original <running> configuration datastore has been split into the <running> configuration datastore and the <intended> configuration datastore. The <intended> configuration datastore contains the configuration that is intended to be applied to the device. On a traditional NETCONF implementation, <running> and <intended> are always the same. However, implementations that support inactive configuration usually expose <running> to clients that understand inactive configuration and they expose <intended> to clients that do not understand inactive configuration. The introduction of an <intended> datastore makes this difference explicit.

- A new <applied> configuration datastore has been introduced that reflects the configuration currently active on the device. This may be a subset or a superset of the <intended> configuration. Possible reasons for differences are situations where intended configuration can't be applied due to missing resources or where configuration changes take noticeable time to become applied.

- The operational state is considered to reside in a conceptual <operational-state> datastore. This new read-only datastore consists of config true and config false nodes; in the original model the operational state only had config false nodes. The reason for incorporating config true nodes here is to be able to expose all operational settings without having to replicate definitions in the data models.

- The model foresees control-plane datastores that are by definition not part of the persistent configuration of a device. In some contexts, these have been termed ephemeral datastores since the information is ephemeral, i.e., lost upon reboot. The control-plane datastores interact with the rest of the system like any other control-plane mechanisms (e.g., routing protocols, discovery protocols). Note that the ephemeral datastore discussed in I2RS documents maps to a control-plane datastore in the revised datastore model described here.

5. Discussion

5.1. Missing Resources

Sometimes some parts of <intended> configuration refer to resources that are not present and hence parts of the <intended> configuration cannot be applied. A typical example is an interface configuration that refers to an interface that is not currently present. In such a situation, the interface configuration remains in <intended> but the interface configuration will not appear in <applied>.

5.2. System-controlled Resources

Sometimes resources are controlled by the device and such system controlled resources appear in (and disappear from) the operational state dynamically. If a system controlled resource has matching configuration in <intended> when it appears, the system will try to apply the configuration, which causes the configuration to appear in <applied> eventually (if application of the configuration was successful).

5.3. Auto-configured or Auto-negotiated Values

Sometimes configuration leafs support special values that instruct the system to automatically configure a value. An example is an MTU that is configured to 'auto' to let the system determine a suitable MTU value. Another example is Ethernet auto-negotiation of link speed. In such a situation, the <intended> and <applied> configuration datastores report the value 'auto' while the corresponding leaf in the operational state datastore will report the actual MTU value or the auto-negotiated link speed.

Since a config true leaf may be used both for configuration and for reporting operational state, the value set of a leaf allowed in a configuration datastore may be different from the value set of the corresponding leaf in the operational state datastore.

5.4. Operational State with Different Origins

Sometimes a single list is used to report operational state values that originate from difference sources, i.e., configuration, control plane protocols, or internal processing. An example are IP addresses of an interface that can originate from configuration, from DHCP, or may be dynamically auto-configured. In this case, the operational state datastore will report all IP addresses that are assigned to an interface while the applied configuration datastore only lists the successfully configured addresses that have originated from the intended configuration datastore.

6. Implications

6.1. Implications on NETCONF

o A mechanism is needed to announce support for <intended> and <applied> configuration datastores.

o Support for <intended> and <applied> datastores should be a feature (optional to implement).

o For systems supporting <intended> or <applied> configuration datastores, the <get-config/> operation may be used to retrieve data stored in these new datastores.

o A new operation should be added to retrieve the operational state data store (e.g., <get-state/>). (In principle <get-config/> could work but it would be a confusing name.)

o The <get/> operation will be deprecated since it returns data from two datastores that may overlap in the revised datastore model.

- Invoking <get-config/> on <running> will return <intended> for backwards compatibility. [XXX: How to deal with <edit-config/> for old and new clients with inactive nodes? XXX]

6.2. Implications on RESTCONF

- The {+restconf}/data resource represents the combined configuration and state data resources that can be accessed by a client. This is effectively bundling <running> together with <operational-state>, much like the <get/> operation of NETCONF. The RESTCONF design should change such that the {+restconf}/data resource does not return the content of multiple datastores. Instead, it should return the <running> datastore by default.

- The "content" query parameter can be used to select whether config, nonconfig or all data is returned. It defaults to all. The "content" query parameter should be changed to allow the selection of other datastores, e.g., <operational-state>.

6.3. Implications on YANG

- Some clarifications may be needed if this revised model is adopted. YANG currently describes validation in terms of the <running> configuration datastore while it really happens on the <intended> configuration datastore.

6.4. Implications on Data Models

- Since the NETCONF <get/> operation returns the content of the <running/> configuration datastore and the operational state together in one tree, data models were often forced to branch at the top-level into a config true branch and a structurally similar config-false branch that replicated some of the config true nodes and added state nodes. With the revised datastore model, this is not needed anymore since the different datastores handle the different lifetimes of data objects and together with the deprecation of the <get/> operation it is not possible to write simpler models.

- There may be some differences in the value set of some objects that are used for both configuration and state. At this point of time, these are considered to be rare cases that can be dealt with using text in description statements. Future versions of YANG may consider whether it is reasonable to allow value sets of schema nodes to be partitioned into config true and config false value sets.

o It is desirable to be able to obtain information why a certain value exists in the operational state datastore. Metadata annotations [I-D.ietf-netmod-yang-metadata] should be defined that allow to report the origin of data in the operational state datastore. Note that the definition needs to be flexible and incrementally deployable since not all systems today maintain information about the origin with the operational state.

7. IANA Considerations

None.

8. Security Considerations

This document discusses a conceptual model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

9. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([I-D.wilton-netmod-opstate-yang], [I-D.bjorklund-netmod-operational], [I-D.wilton-netmod-opstate-yang], [I-D.ietf-netmod-opstate-reqs], [I-D.kwatsen-netmod-opstate], [I-D.openconfig-netmod-opstate]) and [RFC6244] touched on some of the problems of the original datastore model. The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

o Andy Biermann, YumaWorks, <andy@yumaworks.com>

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

o Marcus Hines, Google, <hines@google.com>

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

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

o Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>

o Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>

o Anees Shaikh, Google, <aashaikh@google.com>

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

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

o Robert Wilton, Cisco Systems, <rwilton@cisco.com>

10. References

10.1. Normative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-13 (work in progress), April 2016.

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-13 (work in progress), June 2016.

[I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-05 (work in progress), March 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>.

10.2. Informative References

[I-D.bjorklund-netmod-operational]
Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.

[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.

[I-D.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[I-D.wilton-netmod-opstate-yang]
Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in progress), December 2015.

[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>.

Author's Address

Juergen Schoenwaelder (editor)
Jacobs University

Email: j.schoenwaelder@jacobs-university.de