Subscribing to datastore push updates
draft-netmod-clemm-datastore-push-00.txt

Abstract

This document defines a subscription and push mechanism for
datastores. This mechanism allows client applications to request
updates from a datastore, which are then pushed by the server to the
client per a subscription policy, without requiring additional client
requests.

Status of This Memo

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF). Note that other groups may also distribute
working documents as Internet-Drafts. The list of current Internet-
Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 30, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
1. Introduction

YANG datastores, i.e. datastores that contain data modeled according using YANG [RFC6020], are not restricted to configuration data, but can also contain operational data. It is therefore reasonable to expect that data in YANG datastores will increasingly be used to support applications that are not focused on managing configurations but that are, for example, related to service assurance.

Service assurance applications typically involve monitoring operational state of networks and devices; of particular interest are changes that this data undergoes over time. Likewise, there are applications in which data and objects from one datastore need to be made available to applications in other systems and to remote datastores [peermount-req], requiring mechanisms that allow remote...
systems to become quickly aware of any updates to allow to validate and maintain cross-network integrity and consistency.

Traditional approaches rely heavily on polling, in which data is periodically explicitly retrieved by a client from a server.

There are various issues associated with polling-based management:

- It introduces additional load on network and devices. Each polling cycle requires a separate yet arguably redundant request that results in an interrupt, requires parsing, consumes bandwidth.
- It lacks robustness. Polling cycles may be missed, requests may be delayed or get lost, often particularly in cases when the network is under stress and hence exactly when the need for the data is the greatest.
- Data may be difficult to calibrate and compare. Polling cycles may undergo slight fluctuations, resulting in intervals of different lengths which makes data hard to compare. Likewise, pollers may have difficulty issuing requests that reach all devices at the same time, resulting in offset polling intervals which again make data hard to compare.

More effective is an alternative in which an application can request to be automatically updated of current content of the datastore (such as a subtree, or data in a subtree that meets a certain filter condition), and in which the server subsequently pushes those updates.

The need to perform polling-based management is typically considered an important shortcoming of management applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores using protocols such as NETCONF [RFC6241] or RESTCONF [restconf] will be any more effective, as they would follow the same request/response pattern.

While YANG allows to define notifications, such notifications are generally intended to indicate the occurrence of certain well-specified event conditions, such as the onset of an alarm condition or the occurrence of an error. Likewise, a capability to define configuration change events has been defined in [RFC5277]. However, these change events pertain only to configuration information, not to operational state. RFC 5277 furthermore predates YANG and does not provide tie-in with YANG-defined datastore contents.
Service Assurance applications are not the only applications benefiting from a push- and subscription-based alternative to polling. Another example is Peer Mount [peermount]. Peer Mount allows a datastore to incorporate data from remote datastores by reference, resulting in virtual datastores that are federated across a network and offer different local views. Various use cases indicate the usefulness of introducing caching in conjunction with Peer Mount, which benefits greatly if updates can automatically be pushed from a mount server to a mount client.

The way in which the updates are to occur can be directed by policy. For example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

Because not every server may support every requested interval for every piece of data, it is furthermore necessary for a server to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate subscription parameters.

Finally, a mechanism is needed to communicate the updates themselves. One option is to use existing NETCONF and RESTCONF mechanisms, by defining special notifications with which to carry those updates. Other alternatives are conceivable, such as use of a dedicated publish/subscribe mechanism that provides an alternative to a NETCONF or RESTCONF transport.

This document specifies a YANG data model for the configuration and management of subscriptions to data in YANG datastores. It also defines a notification that can be used to carry data updates and thus serve as push mechanism.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in a hierarchical manner.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

Mount client: The system at which a mount point resides, into which the remote subtree is mounted.

Mount point: A data node that receives the root node of the remote datastore being mounted.
Mount server: The server with which the mount client communicates and which provides the mount client with access to the mounted information. Can be used synonymously with mount target.

Mount target: A remote server whose datastore is being mounted.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

Peer Mount: An extension to the YANG management framework that allows local YANG datastores to incorporate data from remote (peer) YANG datastores.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node

SNMP: Simple Network Management Protocol

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Solution Overview

This document specifies a solution that allows clients to subscribe to information updates in a YANG datastore, which are subsequently pushed from the server to the client. The solution encompasses several components:

- The configuration and management of the subscriptions.
- An ability to negotiate subscription parameters where a subscription policy desired by a client cannot be supported.
- The datastream of the push updates.

In addition, there are a number of additional considerations, such as the tie-in of the mechanisms with security mechanisms. Each of those aspects will be discussed in the following subsections.

3.1. Subscription Model

Yang allows modeling the content of notifications. The contents are a set of explicitly stated data nodes forming a hierarchy. For modeling updates in a datastore, a new generic notification is introduced, the "push-update". This notification has the following
semantics. The contents of the notification are not explicitly stated. They are the union of the data nodes in the Yang modules supported by the server, excluding the following statements: "mandatory", "must", "min-elements", "max-elements", "when", and "default". Note that the notification contents are dynamic, depending on the modules supported by the server.

Subscriptions to the "push-update" are initiated by clients. Servers respond to a subscription request explicitly positively or negatively. Negative responses include information describing the reason for the subscription rejection.

Datastore-push subscriptions are defined using a data model. This model is based on the subscriptions defined in [RFC-5277], which is also reused in RESTCONF. The model is extended with a subscription type a set of parameters for each type. The complete set of subscription parameters is:

- The name of the stream to subscribe to. The stream is called "push-update".
- The identity of the subscriber.
- An optional filter. It describes the subset of stream events of interest to the subscriber. The server should only send to the subscriber the events that match the filter, when present. The absence of a filter indicates that all events in the stream are of interest to the subscriber and all events in it must be sent to the subscriber. Two filtering mechanisms are considered: subtree filtering and XPath filtering, with the semantics described in [RFC5277].
- An optional start time. Used to trigger replays starting at the provided time. Its semantics are those in [RFC5277].
- An optional stop time. Used to limit temporarily the events of interest. Its semantics are those in [RFC5277].
- A notification trigger definition. The trigger can be periodic or based on change. For periodic subscriptions, the trigger is defined by the interval with which to push updates. For on-change subscriptions, the trigger is defined using the dampening interval with which to push repeated changes, an indicator for the magnitude of changes, etc.

The following figure depicts the data model.
module: ietf-datastore-push
  +--rw datastore-push-subscription
    +--rw stream string
    +--rw subscription-id subscription-identifier
    +--rw (filter)?
      | +--:(substree)
      | | +--rw subtree-filter
      | +--:(xpath)
      |   +--rw xpath-filter yang:xpath1.0
    +--rw (notification-trigger)
      | +--:(periodic)
      | | +--rw period yang:timeticks
      | +--:(on-change)
      |   +--rw (change-policy)
      |     +--:(delta-policy)
      |     | +--rw delta uint32
    +--rw start-time? yang:date-and-time
    +--rw stop-time? yang:date-and-time

Figure 1: Model structure

The example below illustrates a subscription for a periodic push of all data under a container called foo.

<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>push-update</stream>
    <subscription-id>foo</subscription-id>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/dspush/1.0"
      select="/ex:foo"/>
    <period>500</period>
  </filter>
</create-subscription>
</netconf:rpc>

Figure 2: Subscription example

3.2. Negotiation of Subscription Policies

A subscription rejection can be caused by the inability of the server to provide a stream with the requested semantics. Providing "on-change" updates for operational data can be computationally expensive and an agent may decide not to support them or supporting them for a small number of subscribers or for a limited set of data nodes.
Datastore-push supports a simple negotiation between clients and servers for subscription parameters. The negotiation is limited to a single pair of subscription request and response. For negative responses, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters are no guarantee for subsequent requests for this client or others.

The example below illustrates a subscription response, where an agent does not support frequent periodic updates, and suggests a different sampling rate to the client.

```xml
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>push-update</stream>
    <subscription-id>foo</subscription-id>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/dspush/1.0"
      select="/ex:foo"/>
    <period>500</period>
  </filter>
</create-subscription>
</netconf:rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>operation-not-supported</error-type>
    <error-tag>application</error-tag>
    <error-severity>error</error-severity>
    <error-info>
      <supported-subcription>
        <period>3000</period>
      </supported-subcription>
    </error-info>
  </rpc-error>
</rpc-reply>
```

Figure 3: Subscription negotiation example

### 3.3. Push Data Stream and Transport Mapping

Pushing data based on a subscription could be considered analogous to a response to a data retrieval request, e.g. a "get" request. However, contrary to such a request, multiple responses to the same request may get sent over a longer period of time. Likewise, clients
need to be able to distinguish between data updates and state update regarding the subscription itself, for example when a subscription can no longer be serviced.

A more suitable mechanism is therefore that of a notification. Contrary to notifications associated with alarms and unexpected event occurrences, push updates are solicited, i.e. tied to a particular subscription which triggered the notification. (An alternative conceptual model would consider a subscription an "opt-in" filter on a continuous stream of updates.)

The notification contains several parameters:

- A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.
- A data node that contains a representation of the datastore subtree containing the updates. The subtree is filtered per access control rules to contain only data that the subscriber is authorized to see. Also, depending on the subscription type, i.e., specifically for on-change subscriptions, the subtree contains only the data nodes that contain actual changes. (This can be simply a node of type string or, for XML-based encoding, anyxml.)

Notifications are sent using <notification> elements as defined in [RFC5277]. Alternative transports are conceivable but outside the scope of this specification.

3.4. Other considerations

3.4.1. Authorization

A client may only receive updates to data that the client has proper authorization for. Normal authorization rules apply. Data that is being pushed therefore needs to be subjected to a filter that applies the corresponding rules, removing any non-authorized data as applicable.

The authorization model for data in YANG datastores is described in the Netconf Access Control Model [RFC6536].

3.4.2. Subscription status and subscription monitoring

It is possible that a server may no longer be able to serve a subscription that had been previously accepted. For example, a server may have run out of resources, or internal errors may have
occurred. When this is the case, a server needs to be able to temporarily suspend the subscription, or even to terminate it.

For this reason, a server SHALL maintain status information for each subscription that indicates the current status of the subscription.

In addition, a server needs to indicate any changes in status to the subscriber through a notification. Specifically, subscribers need to be informed of the following:

- A subscription has been temporarily suspended, including the reason. (See subscription-suspended in the model below.)
- A subscription (that had been suspended earlier) is once again operational. (See subscription-resumed in the model below.)
- A subscription has been abnormally terminated, including the reason. (See subscription-terminated in the model below.)

Finally, a server might provide additional information about subscriptions, such as statistics about the number of data updates that were sent. However, such information is currently outside the scope of this specification.

3.4.3. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, it should be noted that monitoring of operational state changes inside a system can be associated with significant implementation challenges.

Even periodic retrieval of operational state alone, to be able to push it, can consume considerable system resources. Configuration data may in many cases be persisted in an actual database or a configuration file, where retrieval of the database content or the file itself is reasonably straightforward and computationally inexpensive. However, retrieval of operational data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, if is important for an implementation to understand what subscriptions it can or cannot support. It is far preferable to decline a subscription request, than to accept it only to result in subsequent failure later.

Whether or not a subscription can be supported will in general be determined by a combination of several factors, including the
subscription policy (on-change or periodic, with on-change in general
being the more challenging of the two), the period in which to report
changes (1 second periods will consume more resources than 1 hour
periods), the amount of data in the subtree that is being subscribed
to, and the number and combination of other subscriptions that are
concurrently being serviced.

4. YANG module

<CODE BEGINS>
file "ietf-datastore-push@2014-10-27.yang"

module ietf-datastore-push {
  // RFC Ed.: replace XXXX with 'ietf' and remove this note
  namespace "urn:XXXX:params:xml:ns:yang:ietf-datastore-push";
  prefix "datastore-push";

  import ietf-yang-types { prefix yang; }

  organization
    "IETF";

  contact
    "Editor: Alexander Clemm
<mailto:alex@cisco.com>

    Editor: Alberto Gonzalez Prieto
<mailto:albertgo@cisco.com>

    Editor: Eric Voit
<mailto:evoit@cisco.com>";

  description
    "This module contains conceptual YANG specifications
    for datastore push.";

  revision 2014-10-27 {
    description
      "Initial revision.";
    reference
      "Datastore push.";
  }

  // Typedefs
typedef datastore-contents {
    type string;
    description
"The encoding of the contents adheres to the subscription parameters. It corresponds to the filtered datastore subtree."

typedef subscription-identifier {
  type string {
    length "1 .. max";
  }
  description
    "A client-provided identifier for the subscription."
}

// Identities
// Subscription error
identity subscription-errors {
  description
    "Base identity for subscription errors."
}

typedef subscription-term-reason {
  type identityref {
    base "subscription-errors";
  }
  description
    "Reason for a server to terminate a subscription."
}

typedef subscription-susp-reason {
  type identityref {
    base "subscription-errors";
  }
  description
    "Reason for a server to suspend a subscription."
}

identity internal-error {
  base "subscription-errors";
  description
    "Subscription failures caused by server internal error."
}

identity no-resources {
  base "subscription-errors";
  description
    "Lack of resources, e.g. CPU, memory, bandwidth";
}
identity other {
    base "subscription-errors";
    description
        "Fallback reason - any other reason";
}

// Notifications
notification push-update {
    description
        "This notification contains an update from a datastore";

    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description
            "This references the subscription because of which the
             notification is sent.";
    }

    leaf datastore-contents {
        type datastore-contents;
        description
            "This contains datastore contents
             per the subscription.";
    }
}

notification subscription-suspended {
    description
        "This notification indicates a suspension of the
        subscription by the server has occurred. No further
datastore updates will be sent until subscription resumes.";

    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description
            "This references the affected subscription.";
    }

    leaf reason {
        type subscription-susp-reason;
        description
            "Provides a reason for why the subscription was
            suspended.";
    }
}
notification subscription-resumed {
    description "This notification indicates that a subscription that had previously been suspended has resumed. Datastore updates will once again be sent.";
    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description "This references the affected subscription.";
    }
}

notification subscription-terminated {
    description "This notification indicates that a subscription has been terminated."
    leaf subscription-id {
        type subscription-identifier;
        mandatory true;
        description "This references the affected subscription.";
    }
    leaf reason {
        type subscription-term-reason;
        description "Provides a reason for why the subscription was terminated.";
    }
}

container datastore-push-subscription {
    description "Content of a yang-push subscription."
    leaf stream {
        type string;
        mandatory true;
        description "The name of the stream to subscribe to.";
    }
}
leaf subscription-id {
    type subscription-identifier;
    mandatory true;
    description
    "Identifier to use for this subscription."
}

choice filter {
    description
    "Subset of stream events of interest."
    case substree {
        container subtree-filter {
            description
            "Datastore subtree of interest."
        }
    }
    case xpath {
        leaf xpath-filter {
            type yang:xpath1.0;
            mandatory true;
            description
            "Xpath defining the events of interest."
        }
    }
}

choice notification-trigger {
    mandatory true;
    description
    "Defines necessary conditions for sending an event to
    the subscriber."
    case periodic {
        description
        "The agent is requested to notify periodically the
        current values of the datastore or the subset
        defined by the filter."
        leaf period {
            type yang:timeticks;
            mandatory true;
            description
            "Elapsed time between notifications."
        }
    }
    case on-change {
        description
        "The agent is requested to notify changes in
values in the datastore or a subset of it defined by a filter.

choice change-policy {
  mandatory true;
  description
    "Policy describing necessary conditions for sending an event to the subscriber.";
  case delta-policy {
    leaf delta {
      type uint32;
      mandatory true;
      description
        "For integer, minimum difference between current and last reports values that can trigger an update.";
    }
  }
  }
}

leaf start-time {
  type yang:date-and-time;
  description
    "Starting time for replays."
  reference "RFC 5277, Section 2.1.1";
}

leaf stop-time {
  type yang:date-and-time;
  description
    "Time limit for events of interest."
  reference "RFC 5277, Section 2.1.1";
}

5. Security Considerations

Subscriptions could be used to attempt to overload servers of YANG datastores. For this reason, it is important that the server has the ability to decline a subscription request if it would deplete its resources. In addition, a server needs to be able to suspend an existing subscription when needed. When this occur, the subscription
status is updated accordingly and the clients are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

6. References

6.1. Normative References


6.2. Informative References


Authors’ Addresses

Alexander Clemm
Cisco Systems
EMail: alex@cisco.com

Alberto Gonzalez Prieto
Cisco Systems
EMail: albertgo@cisco.com

Eric Voit
Cisco Systems
EMail: evoit@cisco.com