YANG Configuration Templates

draft-tt-netmod-yang-config-templates-00

Templates Team

Robert Wills, Cisco
Qiufang Ma, Huawei
Deepak Rajaram, Nokia

Timeline

Two virtual interim meetings

Template Team meetings and drafting work

IETF 121
Gathering ideas on templates

IETF 122
Update to WG from interim meetings; high-level requirements written

IETF 123
-00 draft from Template Team

The problem

- Datastores often contain multiple subtrees of identical/similar config
  - For example: multiple interfaces with similar config
- Currently, the client has to repeatedly configure the same nodes in each subtree
  - Complex and error-prone
  - Masks the intent of the client

YANG Configuration Templates solution

- A **Configuration Template** is a fragment of configuration, stored in the running datastore
- A node in <running> can **apply** a template to instantiate its configuration on descendant nodes
- Config from templates can be **overriden** by regular config

Pieces of the template solution

Config to define templates

Regular config

Metadata to apply templates

Regular config
+
config expanded from templates

Key concepts

Defining and applying templates

- Templates have their own Yang model
  - A list of templates:
    - Name
    - Content
  - Templates are manipulated just like any other config in the datastore
  - The template's content can be any config nodes in the device's schema
  - The content can use a pattern on list keys to restrict which list entries the template takes effect for
- Templates are applied using RFC 7952 metadata
  - Multiple templates can be applied to a node
  - Templates can be applied at any level in the hierarchy

apply-templates: RFC 7952 metadata

- Introduce a new piece of metadata: **apply-templates**
- Value is a space-separated list of templates to apply to the node

- No existing mechanism for a client to edit RFC 7952 metadata?
- The draft proposes a mechanism to do this, by specifying the metadata alongside the regular config in the edit request

Expansion and precedence rules

- The value of a node in <intended> is determined by using precedence rules to decide where to take the value from
  - The draft specifies the precedence rules
- When the contents of a template is changed, <intended> is also changed
  - “living templates”

Validation

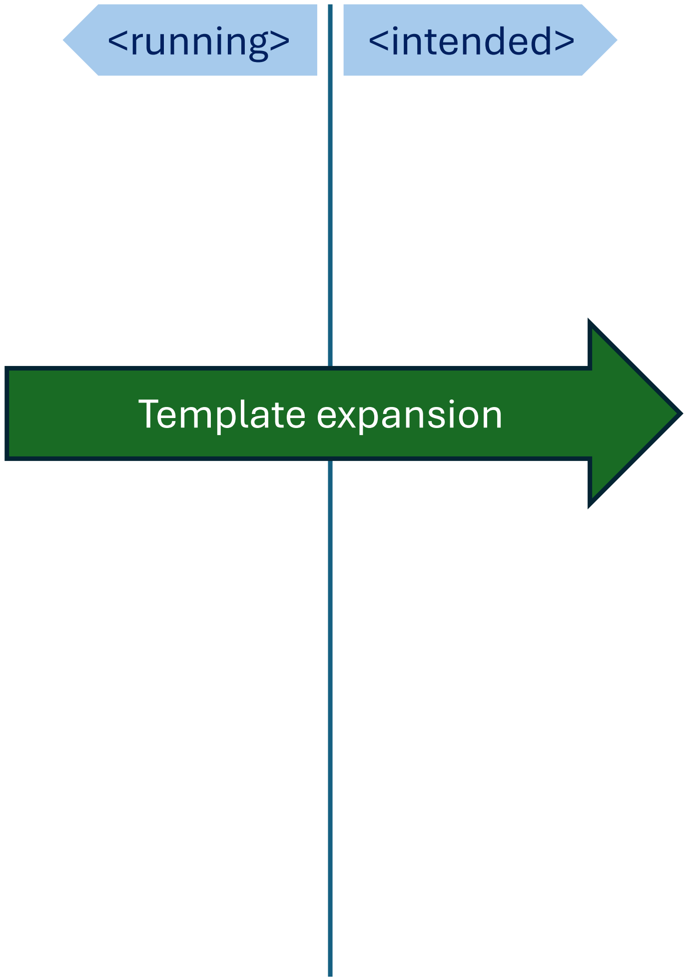
- In <running>:
  - Template content does not have to be fully valid
    - eg mandatory nodes can be missing
  - Where validation is possible (eg range, pattern, length), the server SHOULD enforce these constraints in the template
- The contents of <intended> (after template expansion) must be valid at commit time

Key concepts

Using interface config for this example, but templates work with any schema

Worked example

Templates are defined and applied in the <running> datastore



The expanded config appears in the <intended> datastore

Defining a Template

<running> | <intended>

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <mtu>1500</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>
```

The template contains a fragment of config using the ietf-interfaces model

Templates are defined in a list in the <running> datastore

Defining a Template

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <mtu>1500</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>
```

```
<interfaces>
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

<running>

<intended>

```
<interfaces>
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

The template has **not** been applied, so <intended> matches <running>

Applying a Template

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <mtu>1500</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>

<interfaces
 ct:apply-templates="interface-mtu">
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

<running> | <intended>

Now, the template is applied using the **apply-templates** metadata

Applying a Template

<running>

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <mtu>1500</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>
```

Now, the template is applied using the **apply-templates** metadata

```
<interfaces
 ct:apply-templates="interface-mtu">
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

<intended>

```
<interfaces>
 <interface>
  <name>loopback0</name>
  <mtu>1500</mtu>
 </interface>
 <interface>
  <name>eth0</name>
  <mtu>1500</mtu>
 </interface>
 <interface>
  <name>eth1</name>
  <mtu>1500</mtu>
 </interface>
</interfaces>
```

Config from the template appears in **<intended>**

Using simple patterns to restrict templates

<running>

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <name>eth.*</name>
     <mtu>1500</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>
```

The list key has a **pattern**, restricting which list entries the template takes effect for

```
<interfaces
 ct:apply-templates="interface-mtu">
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

<intended>

```
<interfaces>
 <interface>
  <name>loopback0</name>
  <mtu>1500</mtu>
 </interface>
 <interface>
  <name>eth0</name>
  <mtu>1500</mtu>
 </interface>
 <interface>
  <name>eth1</name>
  <mtu>1500</mtu>
 </interface>
</interfaces>
```

Only **ethernet** interfaces have the template applied

Updating a template

<running> | <intended>

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <name>eth.*</name>
     <mtu>9000</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>

<interfaces
 ct:apply-templates="interface-mtu">
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

The value of the MTU is updated in the template

```
<interfaces>
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
  <mtu>9000</mtu>
 </interface>
 <interface>
  <name>eth1</name>
  <mtu>9000</mtu>
 </interface>
</interfaces>
```

The change **takes effect** in <intended>

Overriding a template

```
<templates xmlns="...">
 <template>
  <id>interface-mtu</id>
  <content>
   <interfaces xmlns="...">
    <interface>
     <name>eth.*</name>
     <mtu>9000</mtu>
    </interface>
   </interfaces>
  </content>
 </template>
</templates>

<interfaces
 ct:apply-templates="interface-mtu">
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
  <mtu>2000</mtu>
 </interface>
 <interface>
  <name>eth1</name>
 </interface>
</interfaces>
```

The eth0 subtree overrides the value of the MTU leaf by explicitly setting it

<running> | <intended>

```
<interfaces>
 <interface>
  <name>loopback0</name>
 </interface>
 <interface>
  <name>eth0</name>
  <mtu>2000</mtu>
 </interface>
 <interface>
  <name>eth1</name>
  <mtu>9000</mtu>
 </interface>
</interfaces>
```

The explicitly set value **takes priority** for eth0

Open issues / next steps

- Non-NMDA clients/servers
- Precisely define the semantics of template expansion
- The meaning/semantics of keys in a template
- Define which patterns are valid in template list keys
- Schema for template contents – use Schema Mount?

Backup slides

Expansion and precedence rules

- The value of a node in <intended> is determined by using precedence to decide where to take the value from
  - **Non-template config** always has the highest precedence
  - When templates are applied to multiple ancestors, the **innermost ancestor** takes precedence
  - When multiple templates are applied to a node, the **order they appear** determines the precedence
- Expansion is "additive":
  - A node from a template can be **modified** using non-template config, but cannot be **deleted**
- When a template is changed, <intended> is changed
  - "living templates"

Editing RFC 7952 metadata

- Recall: the apply-templates metadata is used to apply one or more templates to a node
- No existing mechanism for a client to edit RFC 7952 metadata?
- The draft proposes that in an edit request:
  - If apply-templates is **specified and non-empty**, the metadata is set to that value
    - ie the client provides the new list of templates to apply to the node
  - If apply–templates is **specified and empty**, the metadata is deleted
    - ie the client can remove all template applications
  - If apply-templates is **not specified**, no change to the metadata
    - ie the client can edit the children of this node without changing which templates are applied