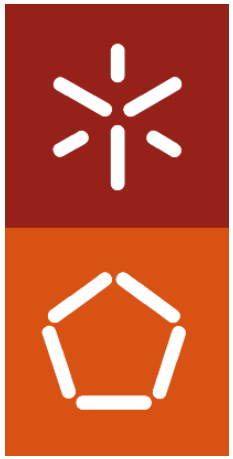
GESTÃO DE REDES / NETWORK MANAGEMENT

Notas complementares / Complementary notes

NETCONF + YANG
RESTCONF



Introduction

- Networks Operators complained that **configuration management** was not being addressed by IETF! (RFC3535)
- But, can't we do it with existing management technology? Do we need a new one?

SNMP / SMIv2 / MIBs

COPS-PR /SPPI /PIBs

SIM / MOP / UML / PCIM

CLI / TELNET / SSH

HTTP / HTML

XML

Introduction

- What about SNMP?

SNMP / SMI / MIBs

+ Works very well for device monitoring
+ The stateless nature of SNMP is useful for statistical and status polling.
+ Core MIBs implemented everywhere!
+ Many proprietary MIBs available & supported
+ Important for event correlation, alarm detection, and root cause analysis

- Too little deployment of writable MIB modules! (devices are not fully configurable by SNMP)
- The transactional model and the protocol constraints make it more complex to implement MIBs
- Scaling problems with regard to the number of objects in a device
- Semantic mismatch between the task-oriented view and data-centric view (no high level proc.)

- Standard MIB modules often lack writable MIB objects which can be used for configuration
- SNMP requires applications to be useful:
  Designed as a programmatic interface between management applications and devices.

Introduction

- What about CLI / TELNET / SSH ?

CLI / TELNET / SSH

+ A saved sequence of textua… commands can easily be replayed
+ It is natural to use the same interface users already use and same abstractions for automating configuration change
+ Command line interface (CLI) does not require any special purpose applications
+ CLI are task-oriented by nature…
+ Most command line interfaces provide context sensitive help (reduced learning curve)

- …I interfaces lack a common data model (different behavior between vendors)
- Command line interface is primarily targeted to humans (who can adjust to changes easily)
- CLI lack of proper version control for the syntax and the semantic (difficult to maintain)
- Not efficient in multi-vendor environments
- Access control is often ad-hoc and insufficient

It is still a common practice to use TELNET/SSH to access and configure devices using their CLI

Introduction

- What about HTTP / HTML ?

HTTP / HTML

+ Embedded web servers for configuration are end-user friendly and solution oriented
+ Suitable for configuring consumer devices by inexperienced users
+ No need for specialized applications (only a web browser is required)

- Embedded web servers are management application hostile (difficult to use by programs)
- Replay of configuration is often problematic (different versions of same device may use different interface)
- The access control facilities are often ad-hoc and sometimes insufficient.

Many devices have an embedded web server which can be used to configure the device and to obtain status information. Access and configuration through HTTP/HTML are common.

Requirements

- **Network Operators identified Requirements (RFC3535):**
  1. Easy to use (a key requirement!)
  2. Clear distinction between *configuration data*, *operational state* and *statistics* (which parameters were really administratively configured?)
  3. To fetch separately configuration data, operational state data, and statistics…
  4. Enable operators to concentrate on the configuration of the entire network as a whole, rather than individual devices
  5. Support configuration transactions across a number of devices…
  6. Operational change from config A to config B with minimal state changes. Minimize the possible impact of config changes.
  7. Need standards for pulling and pushing configurations from devices
  8. It must be easy to do consistency checks of configurations over time, per link…
  9. (…)

Requirements

(…)

9. Network wide configurations are typically stored in central master databases (…) standardize the common parts of these network wide configuration database schemas
10. Use text tools, like diff or version control (…) devices should not reorder data
11. A role-based access control model and the principle of least privilege…
12. To do consistency checks of access control lists across devices
13. Distinguish between the distribution of configurations and the activation of a certain configuration. Devices should be able to hold multiple configurations
14. it is a requirement to support both data-oriented and task-oriented access control

NETCONF / YANG

Writable MIB Module IESG Statement

March 2, 2014

The IESG is aware of discussions in the OPS area and in a number of working groups about the current practice for standards-based approaches to configuration. The OPS area has shown strong support for the use of NETCONF/YANG while many working groups continue to specify MIB modules for this purpose. The IESG wishes to clarify this situation with this statement:

- IETF working groups are therefore encouraged to use the NETCONF/YANG standards for configuration, especially in new charters.
- SNMP MIB modules creating and modifying configuration state should only be produced by working groups in cases of clear utility and consensus to use SNMP write operations for configuration, and in consultation with the OPS ADs/MIB doctors.

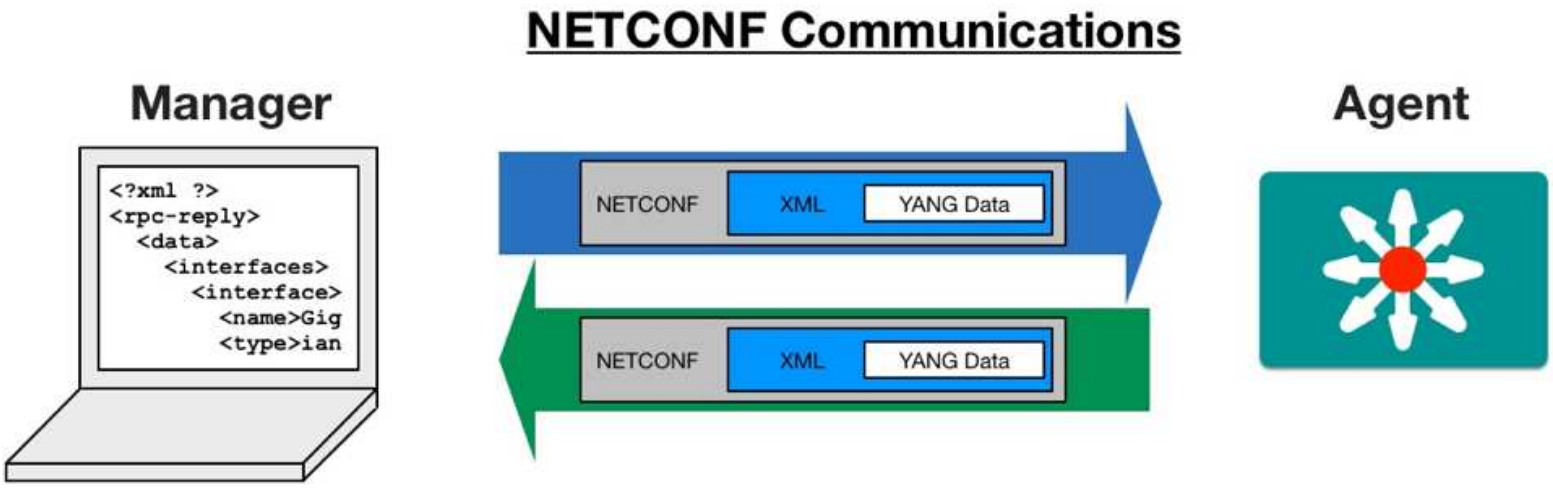


Image from https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28

NETCONF / YANG

- **NETCONF: Network Configuration Protocol**
  - This protocol defines a simple mechanism where network management applications, acting as clients, can invoke operations on the devices, which act as servers.
  - Configuration data information can be retrieved, and new configuration data can be uploaded and manipulated.
  - NETCONF specification [RFC4741] defines a small set of operations
  - Avoid making any requirements on the data carried on those operations → Data Agnostic
- **YANG: Yet Another Next Generation**
  - YANG is a data modeling language used to model configuration and state data manipulated by NETCONF [RFC6020]
  - Allowing both standard and proprietary data models to be published (human readable)
  - YANG allows a modeler to create a data model, to define the organization of the data in that model, and to define constraints on that data.

NETCONF

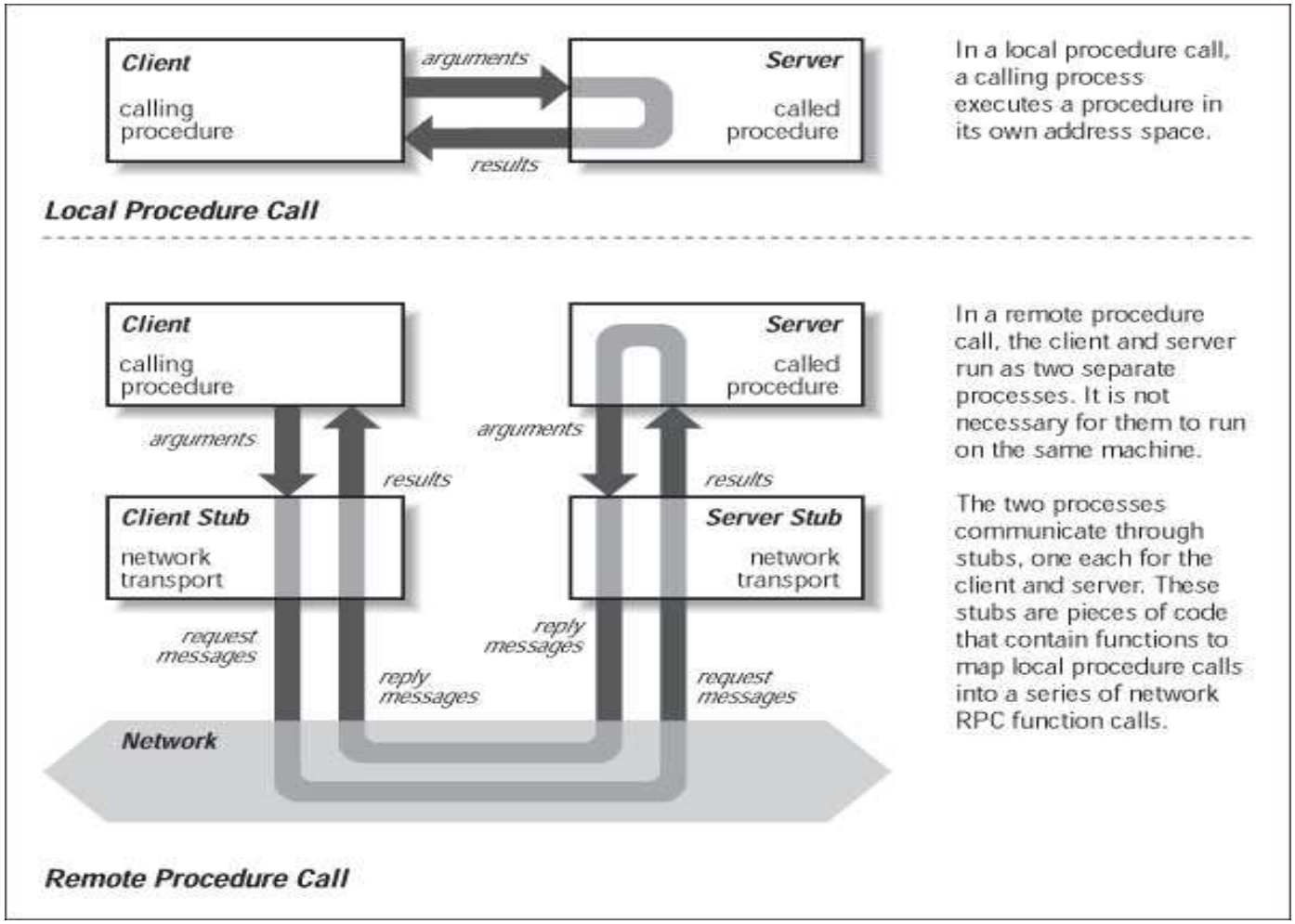
- **Different paradigm:**

  XML-based remote procedure call (RPC) mechanism

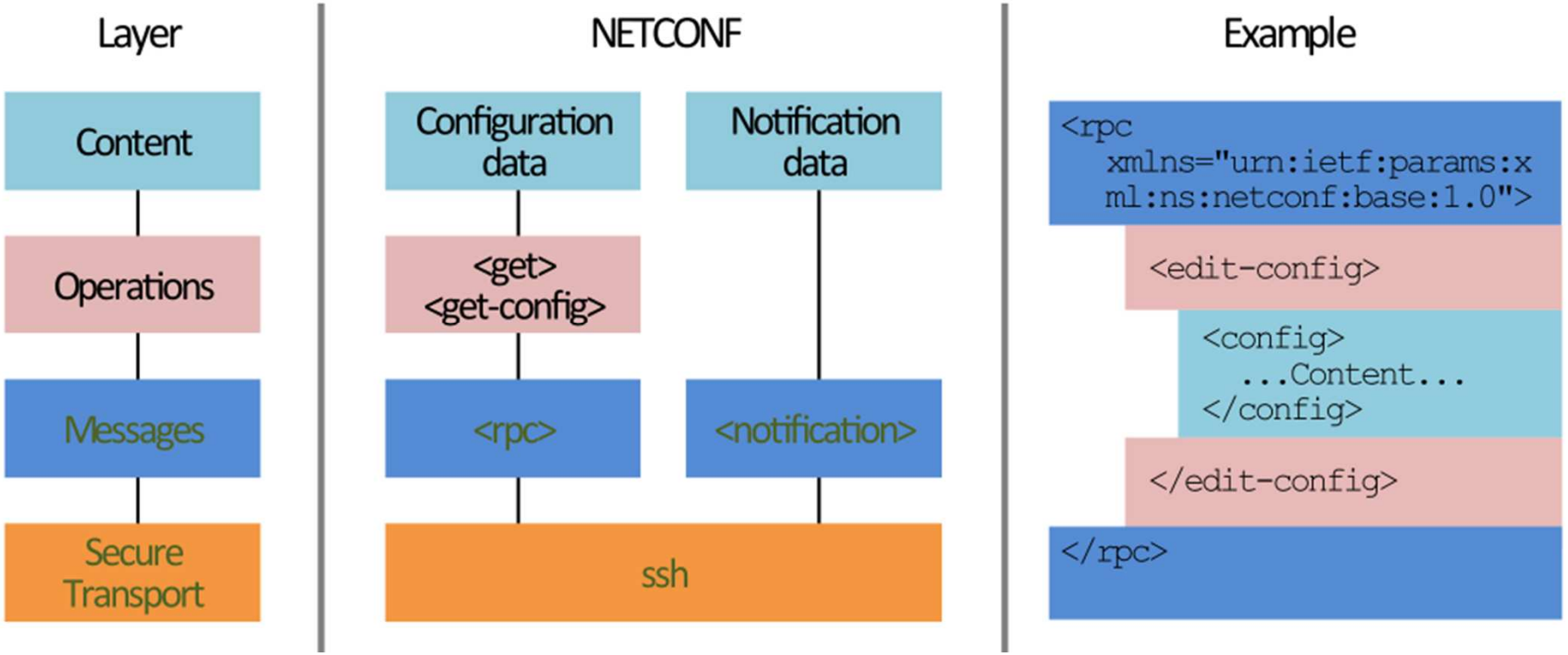
  - task-oriented approach → not data-centric as in SNMP
  - Over a connection-oriented transport service (usually SSH)
  - Includes mechanisms for controlling configuration datastores
  - A NETCONF session is the logical connection between a network administrator or network configuration application and a network device.
  - All NETCONF operations are carried out within a session
  - The NETCONF server is required to authenticate the entity requesting a session before processing any request
  - NETCONF messages MUST be well-formed XML, encoded in UTF-8

NETCONF: XML-based RPC

- What is RPC (Remote Procedure Call)?



NETCONF: Layers



Notice the colors: they show relation between the layers and the encapsulation of data

Image from https://www.ietf.org/slides/slides-edu-network-configuration-with-netconf-00.pdf

NETCONF: Operations

- The NETCONF standard (RFC 4741) defined the following operations that can be executed against the different data store

Operation	Description
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of specified configuration datastore
<edit-config>	Loads all or part of a configuration to the specified configuration datastore
<copy-config>	Replace an entire configuration datastore with another
<delete-config>	Delete a configuration datastore
<commit>	Copy candidate datastore to running datastore
<lock> / <unlock>	Lock or unlock the entire configuration datastore system
<close-session>	Graceful termination of NETCONF session
<kill-session>	Forced termination of NETCONF session

- RFC6241 updated with some variants like a **confirmed commit** and a **validate**

Operation	Description
Confirmed commit	A confirmed <commit> operation MUST be reverted if a confirming commit is not issued within the timeout period
Validate	Validation consists of checking a complete configuration for syntactical and semantic errors before applying the configuration to the device.
Xpath	The XPath capability indicates that the NETCONF server supports the use of XPath expressions in the <filter> element

NETCONF: Configuration Databases

- There may exist **several configuration databases** on devices
- Not all devices implement them all (not required)

- NETCONF operations are targeted to **lock – retrieve – edit – save – copy – unlock** configs on any of those data stores

Image from https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28

NETCONF: Configuration Databases

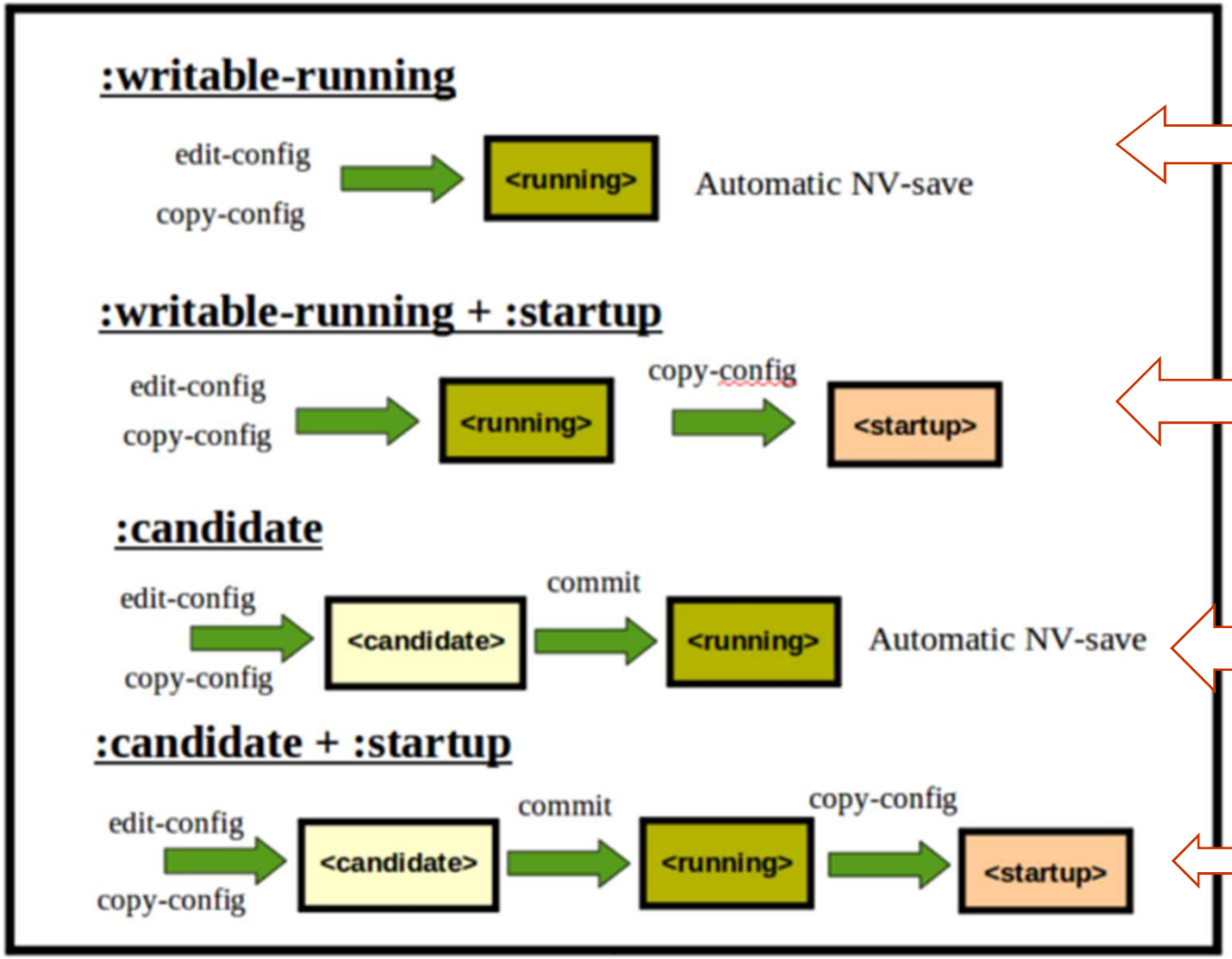
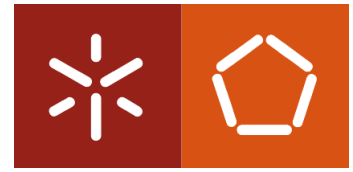
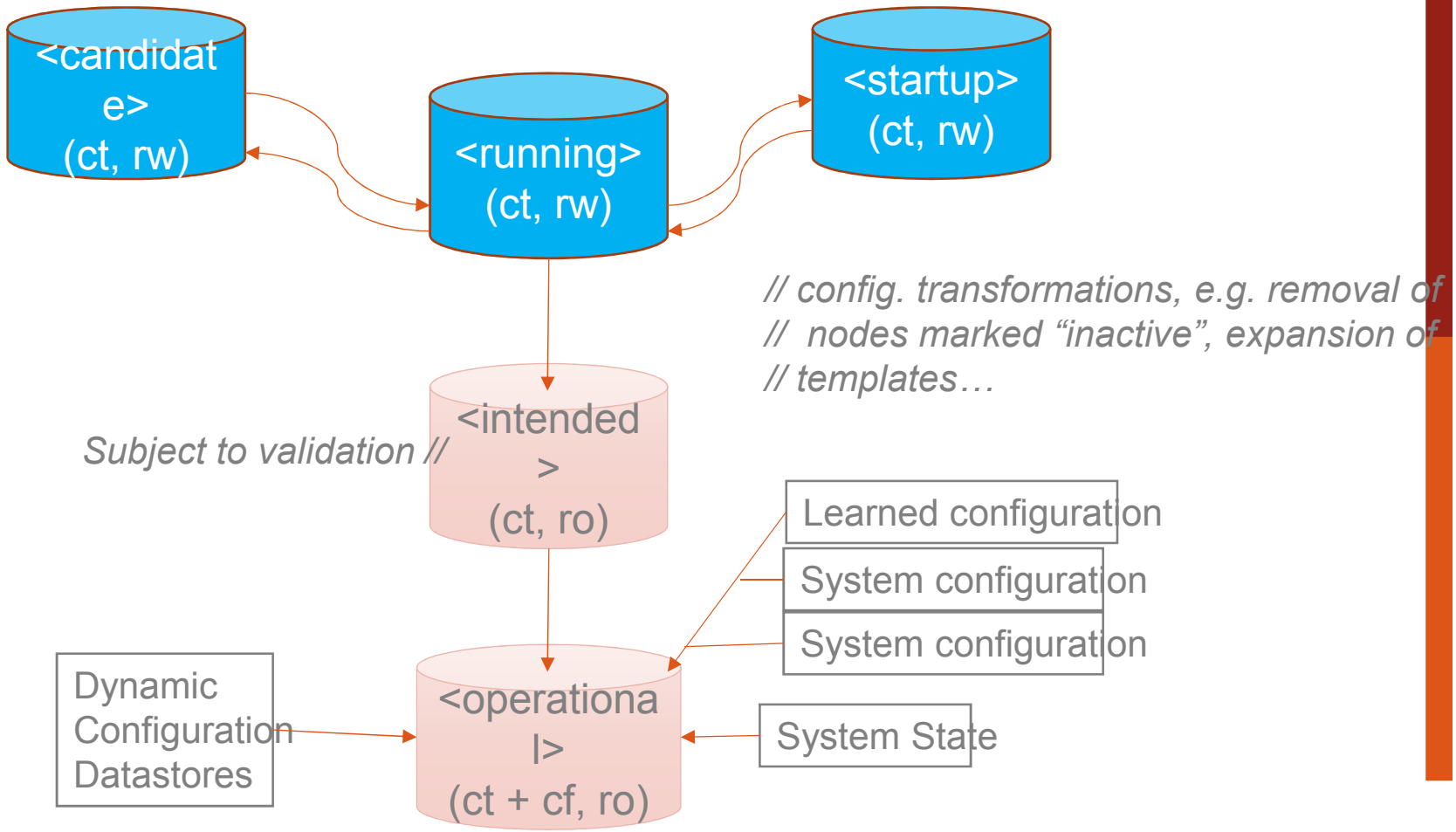


Image from http://www.netconfcentral.org/netconf_docs

NETCONF: Configuration Databases

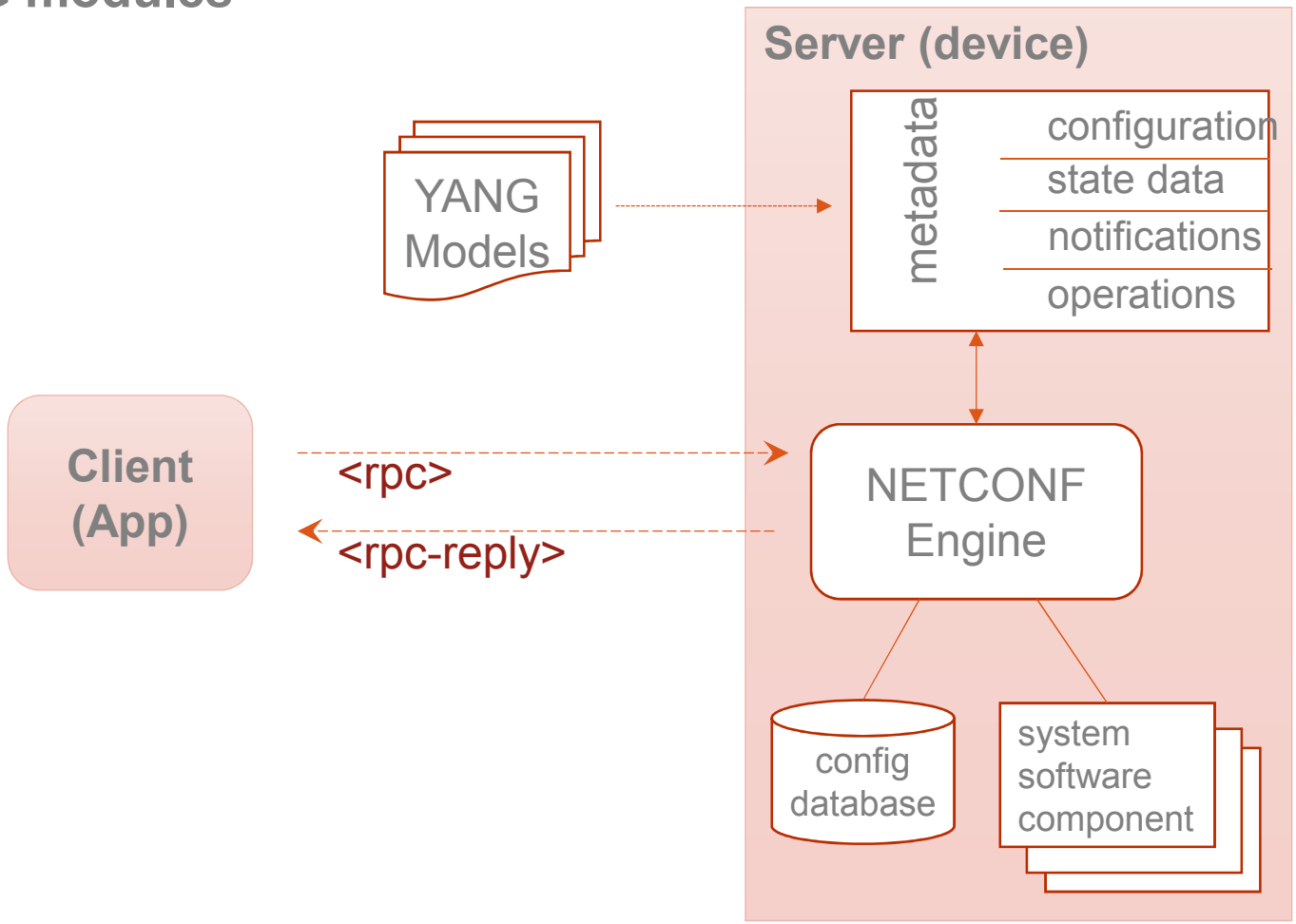


- **New model** recently proposed (RFC 8342, March 2018)



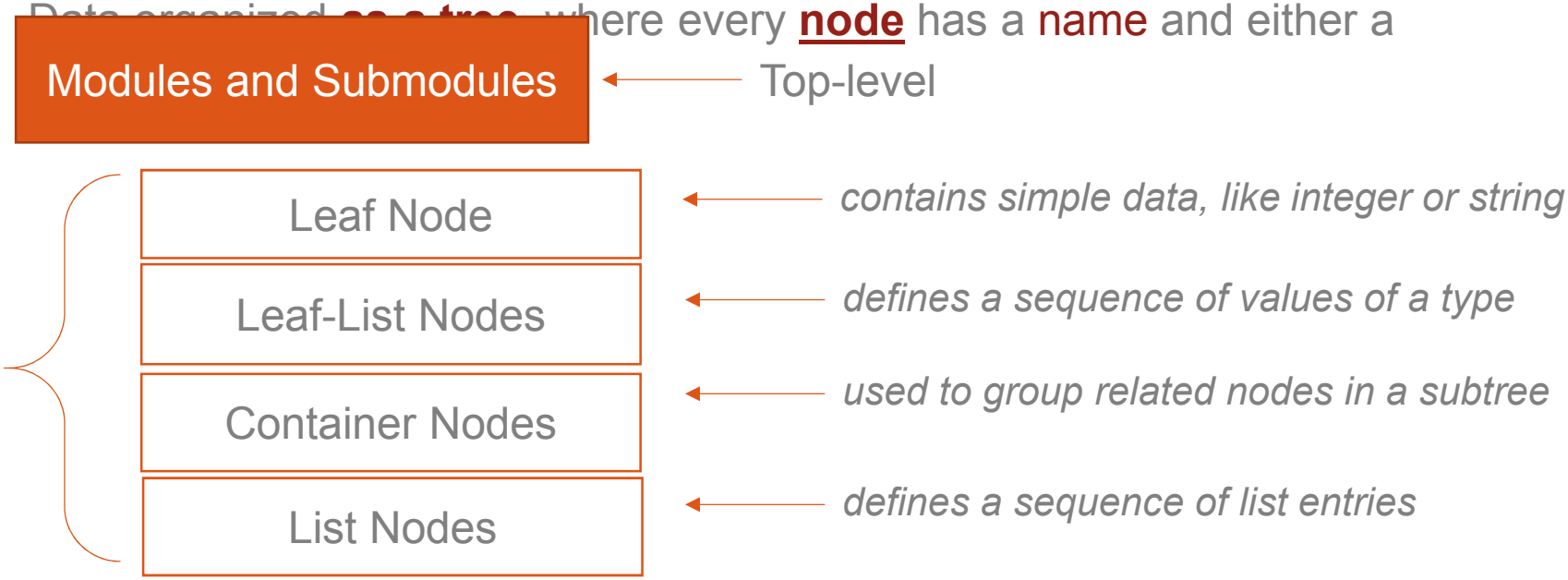
NETCONF & YANG

- The NETCONF client and server are driven by the content of YANG modules



YANG version 1.1

- **YANG is a language originally designed to model data for the NETCONF protocol (RFC 7950, august 2016)**
  - Standard Modules and Proprietary Models: define *configuration data*, RPC, *notifications*
  - Defines hierarchies of data that can be used in operations (configuration, state data, etc.)
  - Data organized **as a tree**, where every **node** has a name and either a



YANG version 1.1

- **example-system model**

YANG

```
module example-system {
        yang-version 1.1;
        namespace "urn:example:system";
        prefix "sys";

        organization "Example Inc.";
        contact "joe@example.com";
        description
          "The module for entities implementing the Example system.";

        revision 2007-06-09 {
          description "Initial revision.";
        }
      .....
}
```

$ pyang –f **yin** *example-system.yang*

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="example-system"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:sys="urn:example:system">
  <yang-version value="1.1"/>
  <namespace uri="urn:example:system"/>
  <prefix value="sys"/>
  <organization>
    <text>Example Inc.</text>
  </organization>
  <contact>
    <text>joe@example.com</text>
  </contact>
  <description>
    <text>The module for entities implementing the Example system.</text>
  </description>
  <revision date="2007-06-09">
    <description>
      <text>Initial revision.</text>
    </description>
  </revision>
...
```

YANG version 1.1

- **Leaf Node** example

YANG

```
leaf host-name {
        type string;
        description
          "Hostname for this system.";
      }
```

XML

```
<host-name>my.example.com</host-name>
```

- Leaf node has:
  - one value; no children; one instance

- **Leaf-List Node** example

YANG

```
leaf-list domain-search {
        type string;
        description
          "List of domain names to search.";
      }
```

XML

```
<domain-search>high.example.com</domain-search>
<domain-search>low.example.com</domain-search>
<domain-search>every.example.com</domain-search>
```

- Leaf-List node has:
  - one value; no children; multiple instances;

YANG version 1.1

- **Container Node** example

YANG

```
container system {
  container login {
    leaf message {
      type string;
      description
        "Message given at start of login session.";
    }
}
```

XML

```
<system>
  <login>
    <message>Good morning</message>
  </login>
</system>
```

- Container node has:
  - no value; only child nodes; any number of child nodes of any type;

YANG version 1.1

- **List Node example**

YANG

```
list user {
        key "name";
        leaf name {
          type string;
        }
        leaf full-name {
          type string;
        }
        leaf class {
          type string;
        }
 }
```

XML

```
<user>
  <name>glocks</name>
  <full-name>Goldie Locks</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow White</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzell</name>
  <full-name>Rapun Zell</full-name>
  <class>tower</class>
</user>
```

- List node has:
  - each entry is like a container identified by key; defines multiple key leafs; any number of child nodes of any type;

YANG version 1.1

- **RPC definition**

Example 1:
An RPC operation with one input
parameter and one output result

```
rpc activate-software-image {
      input {
        leaf image-name {
          type string;
        }
      }
      output {
        leaf status {
          type string;
        }
      }
    }
```

YANG

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <activate-software-image xmlns="http://example.com/system">
      <image-name>example-fw-2.3</image-name>
    </activate-software-image>
  </rpc>

  <rpc-reply message-id="101"
             xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <status xmlns="http://example.com/system">
      The image example-fw-2.3 is being installed.
    </status>
  </rpc-reply>
```

XML

YANG version 1.1

- **RPC definition**

Example 2:
An RPC operation tied to a list!

```
list interface {
    key "name";

    leaf name {
      type string;
    }

    action ping {
      input {
        leaf destination {
          type inet:ip-address;
        }
      }
      output {
        leaf packet-loss {
          type uint8;
        }
      }
    }
  }
```

YANG

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <interface xmlns="http://example.com/system">
      <name>eth1</name>
      <ping>
        <destination>192.0.2.1</destination>
      </ping>
    </interface>
  </action>
</rpc>

<rpc-reply message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:sys="http://example.com/system">
    <sys:packet-loss>60</sys:packet-loss>
</rpc-reply>
```

XML

YANG version 1.1

- **Notification** example

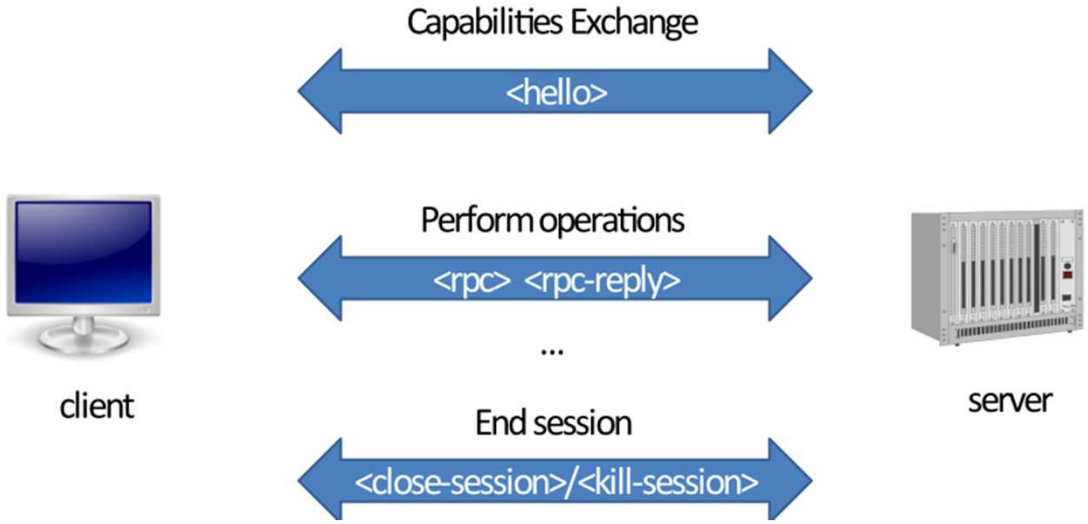
YANG

```
notification link-failure {
    description
      "A link failure has been detected.";
    leaf if-name {
      type leafref {
        path "/interface/name";
      }
    }
    leaf if-admin-status {
      type admin-status;
    }
    leaf if-oper-status {
      type oper-status;
    }
}
```

XML

```
<notification
 xmlns="urn:ietf:params:netconf:capability:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <link-failure xmlns="urn:example:system">
      <if-name>so-1/2/3.0</if-name>
      <if-admin-status>up</if-admin-status>
      <if-oper-status>down</if-oper-status>
    </link-failure>
</notification>
```

NETCONF: Example workflow



- **Client**: open an SSH connection to port 830 (NETCONF PORT) or *to any other port and request NETCONF subsystem*
- **Client**: say “Hello” and provide supported capabilities
- **Server**: answers with its own supported capabilities
- **Client**: call remote procedures…
- **Client/Server**: close the session

Image from https://www.ietf.org/slides/slides-edu-network-configuration-with-netconf-00.pdf

NETCONF simple example

- **STEP 1:** <hello>

C

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.1">
   <capabilities>
      <capability>urn:ietf:params:netconf:base:1.1</capability>
   </capabilities>
</hello>
```

S

C

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.1">
   <capabilities>
      <capability>urn:ietf:params:netconf:base:1.1</capability>
      <capability> urn:ietf:params:netconf:capability:writable-running:1.0 </capability>
     <capability>urn:ietf:params:netconf:capability:candidate:1.0  </capability>
   </capabilities>
   <session-id>5</session-id>
</hello>
```

S

NETCONF simple example

- **STEP 2: <get>** all model data from device

C

```
<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get/>
</rpc>
```

S

C

```
<rpc-reply message-id="102"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <data>
        <!-- ... entire set of data returned ... -->
      </data>
</rpc-reply>
```

S

NETCONF simple example

- **STEP 3: <get>** filter model data from device

C

S

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <t:top xmlns:t="http://example.com/schema/1.2/stats">
        <t:interfaces>
          <t:interface t:ifName="eth0"/>
        </t:interfaces>
      </t:top>
    </filter>
  </get>
</rpc>
```

C

S

```
<rpc-reply message-id="103"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <t:top xmlns:t="http://example.com/schema/1.2/stats">
      <t:interfaces>
        <t:interface t:ifName="eth0">
          <t:ifInOctets>45621</t:ifInOctets>
          <t:ifOutOctets>774344</t:ifOutOctets>
        </t:interface>
      </t:interfaces>
    </t:top>
  </data>
</rpc-reply>
```

NETCONF simple example

- **STEP 4:** <close-session>

C

```
<rpc message-id="104"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>
```

S

C

```
<rpc-reply message-id="104"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <ok/>
</rpc-reply>
```

S

NETCONF Advanced example

- **Example1: configuration update against a single device**
  1. Acquiring the configuration lock.
  2. Checkpointing the running configuration.
  3. Loading and validating the incoming configuration.
  4. Changing the running configuration.
  5. Testing the new configuration.
  6. Making the change permanent (if desired).
  7. Releasing the configuration lock.

NETCONF Advanced example

- **Step 1: A lock should be acquired to prevent simultaneous updates from multiple sources**

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <lock>
         <target>
           <running/>
         </target>
       </lock>
</rpc>
```

If **:candidate** capability is available, the candidate configuration should be locked

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
      <target>
        <candidate/>
      </target>
    </lock>
</rpc>
```

NETCONF Advanced example

- **Step 2: The running configuration can be saved into a local file as a checkpoint**

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <copy-config>
     <target>
       <url>file://checkpoint.conf</url>
     </target>
     <source>
       <running/>
     </source>
   </copy-config>
 </rpc>
```

To restore the checkpoint file, reverse the <source> and <target> parameters

NETCONF Advanced example

- Step 3 If the :candidate capability is supported, the configuration can be loaded onto the device without impacting the running system

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <!-- place incoming configuration changes here -->
    </config>
  </edit-config>
</rpc>
```

If the device supports the :validate:1.1 capability, validate it:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <validate>
    <source>
      <candidate/>
    </source>
  </validate>
</rpc>
```

NETCONF Advanced example

- **Step 4: Changing the Running Configuration**

  If the device supports the :candidate, commit the changes to running

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <commit>
        <confirmed/>
        <confirm-timeout>120</confirm-timeout>
      </commit>
    </rpc>
```

  If not, the configuration change is loaded directly into running

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <edit-config>
        <target>
          <running/>
        </target>
        <config>
          <!-- place incoming configuration changes here -->
        </config>
      </edit-config>
    </rpc>
```

NETCONF Advanced example

- **Step 5: Testing the New Configuration**

  To gain confidence, the application can run tests of the operational state of the device.

- **Step 6: Making the Change Permanent**

  If the device supports the :startup capability, make changes permanent

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
      <target>
        <startup/>
      </target>
      <source>
        <running/>
      </source>
    </copy-config>
  </rpc>
```

  If device supports :candidate, and a confirmed commit was request, the confirming commit must be send before timeout or it will be reverted! (Why?)

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit/>
  </rpc>
```

NETCONF Advanced example

- **Step 7: Releasing the Configuration Lock**

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <unlock>
        <target>
          <running/>
        </target>
      </unlock>
    </rpc>
```

If the :candidate capability is supported, the unlock operation is performed on the candidate…

```
<rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <unlock>
        <target>
          <candidate/>
        </target>
      </unlock>
    </rpc>
```

NETCONF Advanced example

- **Example 2: Operations on Multiple Devices**
  - **Transaction** semantics is required!
  - NETCONF protocol contains sufficient primitives upon which transaction-oriented operations can be built
  - Complete transactional support among all devices is prohibitively expensive!
  - But… we may reduce the size and number of windows for failure scenarios!

  - **Classes of multidevice operation**:
    - the operation *can fail on individual devices* without requiring all devices to revert to their original state!

      → Repeat the operation later… on that device only
    - the *operation should complete on all devices or be fully reversed*.

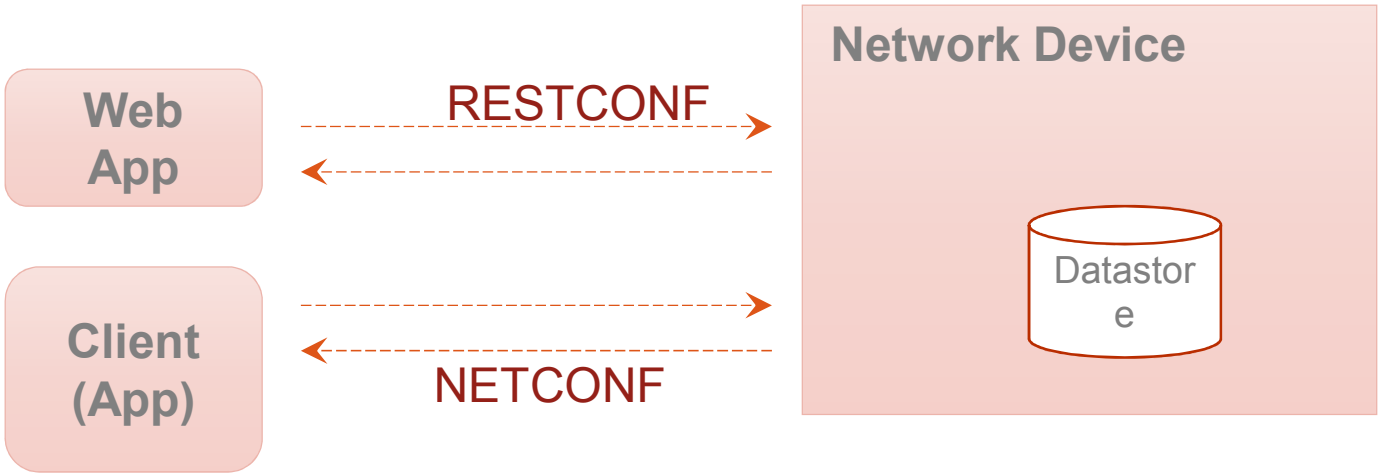
      → The same steps used in single-device operations are used, but are performed in parallel across all devices! If any step fails, the previous

RESTCONF

- **RESTCONF**
  - RESTCONF uses HTTP methods to implement the equivalent of NETCONF operations, enabling basic **CRUD** operations on a hierarchy of conceptual resources.
    - Create (POST), Read (GET), Update (PUT), Delete (DELETE)
  - RESTCONF is not intended to replace NETCONF, but rather to provide an HTTP interface
  - The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by YANG data model
  - RESTCONF does not need to mirror the full functionality of the NETCONF protocol

RESTCONF

- **Coexistence with NETCONF**



- t is possible that locks are in use on a RESTCONF server, even though RESTCONF cannot manipulate locks. In such a case, the RESTCONF protocol will not be granted write access to data resources within a datastore.

RESTCONF Example

- **Retrieve the Top-Level API Resource**

```
GET /.well-known/host-meta HTTP/1.1
Host: example.com
Accept: application/xrd+xml
```

```
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn

    <XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
        <Link rel='restconf' href='/restconf'/>
    </XRD>
```

RESTCONF Example

- The client may then retrieve the top-level API resource

```
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang-data+json
```

In JSON or XML…

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Content-Type: application/yang-data+json

  {
    "ietf-restconf:restconf" : {
      "data" : {},
      "operations" : {},
      "yang-library-version" : "2016-06-21"
    }
  }
```

RESTCONF Example

- Retrieve the Server Capability Information

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/capabilities HTTP/1.1
Host: example.com
Accept: application/yang-data+xml
```

```
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
Last-Modified: Thu, 26 Jan 2017 16:00:14 GMT
Content-Type: application/yang-data+xml

<capabilities xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
      <capability>\
                  urn:ietf:params:restconf:capability:defaults:1.0?\
                  basic-mode=explicit\
      </capability>
                  ....

</capabilities>
```

References

- http://www.netconfcentral.org/
- https://ripe68.ripe.net/presentations/181-NETCONF-YANG-tutorial-43.pdf
- https://datatracker.ietf.org/wg/netconf/documents/
- https://www.cisco.com/c/en/us/support/docs/storage-networking/management/200933-YANG-NETCONF-Configuration-Validation.html
- http://www.yumaworks.com/support/download-yumapro-client/
- https://tools.ietf.org/html/rfc3535
- https://tools.ietf.org/html/rfc6241
- https://tools.ietf.org/html/rfc6020
- https://tools.ietf.org/html/rfc6244
- https://tools.ietf.org/html/rfc8040
- https://www.ietf.org/slides/slides-edu-network-configuration-with-netconf-00.pdf
- https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28