Verifoo RestAPI and XML Docs

Antonio Varvara, Raffaele Sommese

07/12/2017

Rest API Description

Deployment API

This is the main API of the Verifoo Service. It provides, for each graph, the verification of the validity of the network model and the optimised deployment on the hosts.
Example request

- **POST** http://localhost:8080/verifoo/deployment
- Accept: **APPLICATION_XML**;
- Content: XML file with the topology.

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: XML file with integrated deployment information.

Error Response

- 400: **BAD_REQUEST**
- Content-Type: **TEXT_PLAIN**;
- Content: Error Reason.

Translator API

This API provides a converter for verifoo output, you must fill the **<parsingstring>** attribute in the XML file, with the verifoo output. *Example request*

- **POST** http://localhost:8080/verifoo/converter
- Accept: **APPLICATION_XML**;
- Content: XML file with the topology and the output model provided by verifoo in the Parsing String element

Example response

- 200: **OK**
- Content-Type: **APPLICATION_XML**;
- Content: Converted Schema with integrated deployment information.

Error Response

- 400: **BAD_REQUEST**
- Content-Type: **TEXT_PLAIN**;
- Content: Error Reason.

Log API

This API provide a convenient way for accessing the log of log4j2 for debugging purposes.
Example request

- **GET** http://localhost:8080/verifoo/log

Example response

- 200: **OK**
- Content-Type: **TEXT_HTML**;

Z3 Install Note

For the correct operation of the application, you must provide the Z3 native library and include it to Java Library Path. The most convenient way to do this is add the path that the library to the dynamic linking library path.

- In Linux is LD_LIBRARY_PATH
- In MacOS is DYLD_LIBRARY_PATH
- In Windows is PATH

Make sure that you have download the correct version of Z3 according to your OS and your JVM endianness. In any case we have provide a mechanism for add automatically the Z3 library to the path when it is deployed to a WebServer. It is tested on Tomcat and on Webspere and it works for the following distribution:

- Ubuntu x32
- Ubuntu x64
- Debian x64

Install and Testing Note

This project came with an Ant Script for compiling and testing purpose. For generate the war file you can use:

```
ant war
```

For test the internal component (Verifoo Proxy and Unmarshaller):

```
ant test
```

For test the WebService:

```
ant testWS
```

☞ **Warning**: You must have Tomcat installed and you must set $CATALINA_HOME according.

XML Schema

Listing 1: XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<NFV xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    ↪ noNamespaceSchemaLocation="nfvInfo.xsd">
 <graphs>
   <graph id="0">
     <node functional_type="FIREWALL" name="node1">
       <neighbour name="nodeA"/>
       <neighbour name="node2"/>
       <configuration description="A simple description" name="conf1">
         <firewall>
           <elements>
             <source>nodeC</source>
             <destination>nodeD</destination>
           </elements>
         </firewall>
       </configuration>
     </node>
   </graph>
 </graphs>
 <CapacityDefinition>
   <CapacityForNode node="node1" capacity="10"/>
 </CapacityDefinition>
 <PropertyDefinition>
   <Property graph="0" name="IsolationProperty"/>
 </PropertyDefinition>
 <Hosts>
  <Host diskStorage="10" name="host1" type="MIDDLEBOX"/>
  <Host diskStorage="10" name="host2" type="MIDDLEBOX"/>
 </Hosts>
 <Connections>
  <Connection sourceHost="host1" destHost="host2" avgLatency ="-1"/>
  <Connection sourceHost="host1" destHost="host3" avgLatency ="-10"/>
 </Connections>
 <ParsingString></ParsingString>
</NFV>
```

NFV

NFV is the root element of the XML schema, it must contain:

- A **Graphs** element that contains a list of **Graph**
- A list of **Capacity Definition**
- A list of **Property Definition** (one or more)
- An **Hosts** element that contains a list of **Host**
- A **Connections** element that contains a list of **Connection** between hosts
- An optional **Parsing String** used for the converter service.

Graph

A Graph is a chain of service that will be deployed in the network, it is contained inside a list of Graphs.
Verifoo can check and deploy multiple graphs.
Graph is characterised by

- A *unique* **ID**
- A list of **Nodes**

☞ **Warning**: You must define a graph that have at least 1 Client and 1 Server otherwise an exception will be thrown.

Listing 2: Graphs Example

```xml
<graphs>
   <graph id="0">
     <node ....>
   </graph>
   <graph id="1">
     ....
   </graph>
   <graph id="2">
     ....
   </graph>
</graphs>
```

Node

A Node is a logical network element that implements a Network Function. A node is characterised by:

- A *Unique* **Name**
- A **Functional Type**
- A List of **Neighbour Node Names**
- A **Configuration** for the Functional Types

☞ **Warning**: Pay attention when you define the neighbours of a node, remember the graph must be a chain otherwise an exception will.

Listing 3: Node Example

```
<node functional_type="FIREWALL" name="node1">
   <neighbour name="node2"/>
   <configuration ...>
      ....
   </configuration>
</node>
```

Functional Type

A Node can be a:

- **FIREWALL**
- **ENDHOST**
- ~~**ENDPOINT**~~
- **ANTISPAM**
- **CACHE**
- **DPI**
- **MAILCLIENT**
- **MAILSERVER**
- **NAT**
- **VPNACCESS**
- **VPNEXIT**
- **WEBCLIENT**
- **WEBSERVER**
- **FIELDMODIFIER**

☞ **Warning**: ENDPOINT is not implemented in Verifoo.

☞ **Warning**: You cannot have in the same graph more than one server or more than one client, or a server and a client that use different protocols. If you don't respect this condition an exception will be thrown.

Configuration

In this section we describe the different type of configurations that can be provided. A configuration is characterized by an *unique* name and by an *optional* description.

Firewall

A Firewall Configuration contains a list of ACLs (elements). The ACL defines a tuple of source node and destination node that represents the connection that will be blocked.
Due to Verigraph Schema Design is necessary to provide at least 1 ACL, if you don't want to set an ACL provide a configuration with dummy node name.

Listing 4: Firewall Configuration Example

```xml
<configuration description="A simple description" name="conf1">
  <firewall>
    <elements>
      <source>nodeC</source>
      <destination>nodeD</destination>
    </elements>
  </firewall>
</configuration>
```

Cache

A Cache Configuration contains a list of resources. A resource is a node, and cache must include all nodes behind the cache in the chain.
Remember: Cache needs the notion of internal and external networks.

Listing 5: Cache Configuration Example

```xml
<configuration description="A simple description" name="conf3">
  <cache>
    <resource>nodeA</resource>
    <resource>node1</resource>
  </cache>
</configuration>
```

NAT

A NAT Configuration contains a list of internal nodes. The source defines the list of nodes behind the Nat.

Listing 6: NAT Configuration Example

```xml
<configuration description="A simple description" name="conf4">
 <nat>
   <source>nodeA</source>
 </nat>
</configuration>
```

DPI

A DPI Configuration contains a list of notAllowed elements, that defines the string that, if found inside the packet, will make the packet to be dropped.

Listing 7: Cache Configuration Example

```xml
<configuration description="A simple description" name="conf2">
  <dpi>
    <notAllowed>SomeString</notAllowed>
  </dpi>
</configuration>
```

Antispam

An Antispam Configuration contains a list of source nodes that represent the blacklisted mail clients and servers.

Listing 8: Antispam Configuration Example

```xml
<configuration description="A simple description" name="conf5">
 <antispam>
   <source>nodeA</source>
 </antispam>
</configuration>
```

MailServer

A Mail Server Configuration contains the Mail Server names.

Listing 9: MailServer Configuration Example

```xml
<configuration description="A simple description" name="confB">
  <mailserver>
    <name>nodeB</name>
  </mailserver>
</configuration>
```

MailClient

A Mail Client Configuration contains the Mail Server name.

Listing 10: MailClient Configuration Example

```xml
<configuration description="A simple description" name="confB">
    <mailclient mailserver="nodeB"/>
</configuration>
```

WebServer

A Web Server Configuration contains the Web Server names.

Listing 11: WebServer Configuration Example

```xml
<configuration description="A simple description" name="confB">
  <webserver>
    <name>nodeB</name>
  </webserver>
</configuration>
```

WebClient

A Web Client Configuration contains the Web Server name.

Listing 12: Web Client Configuration Example

```xml
<configuration description="A simple description" name="confB">
    <webclient webserver="nodeB"/>
</configuration>
```

VpnAccess

A VpnAccess Configuration contains the VpnExit name.

Listing 13: VpnAccess Configuration Example

```
<configuration description="A simple description" name="conf1">
  <vpnaccess vpnexit="node2" />
</configuration>
```

VpnExit

A VpnExit Configuration contains the VpnAccess name.

Listing 14: Vpn Exit Configuration Example

```
<configuration description="A simple description" name="conf2">
  <vpnexit vpnaccess="node2"/>
</configuration>
```

EndHost

A EndHost Configuration contains a Packet Model.

Listing 15: End Host Configuration Example

```
<configuration description="A simple description" name="conf2">
  <endhost body="thisisarequest"/>
</configuration>
```

Capacity Definition

The Capacity Definition element is a list that contains for each node, its disk requirement that will be used by Verifoo for the deployement.

☞ **Warning**: If a node doesn't have a capacity associated, the code infers that it is 0.

Listing 16: Capacity Definition Example

```
<CapacityDefinition>
   <CapacityForNode node="node1" capacity="10"/>
 </CapacityDefinition>
```

Property Definition

The Property Definition element is a list that contains the properties that will be checked by Verifoo for a specific graph. For now, only the isolation property is supported by Verifoo.

Listing 17: Property Definition Example

```xml
<PropertyDefinition>
   <Property graph="0" name="IsolationProperty"/>
 </PropertyDefinition>
```

Host

An host is a physical machine present in the network infrastructure. An host is characterised by:

- A *Unique* **Name**
- A **Type** to distinguish client and server from middleboxes
- The **Disk Storage** available on the host

After the Rest API has been called, the host will contain also a list of **NodeRef** sub-elements that represent the nodes that will be deployed on that host.

☞ **Warning**: When you try to deploy a graph on a physical network you have to indicate one special host on which the client node will be deployed, another one on which the server node will be deployed and at least one other host on which the other nodes should be deployed, otherwise an exception will be thrown.

Listing 18: Hosts Example

```xml
<Hosts>
  <Host diskStorage="10" name="host1" type="CLIENT"/>
  <Host diskStorage="20" name="host2" type="MIDDLEBOX"/>
  <Host diskStorage="10" name="host3" type="SERVER"/>
 </Hosts>
```

Connection

A connection element represent the physical connection between two hosts. A connection is characterised by:

- A **Source**
- A **Destination**
- The **avgLatency** that represent the average latency on the physical link between the source and the destination.

Listing 19: Connections Example

```
<Connections>
 <Connection sourceHost="host1" destHost="host2" avgLatency ="-1"/>
 <Connection sourceHost="host1" destHost="host3" avgLatency ="-10"/>
</Connections>
```

Parsing String

It's the raw output of Verifoo execution (*model.toString()*). It is used by the converter service.

Listing 20: An extract of ParsingString Example

```
<ParsingString>
...
(define-fun check_isolation_n_0_nodeA_nodeB () Node
  node5)
(define-fun integer_host1 () Int
  1)
(define-fun node3@host7 () Bool
  false)
(define-fun node3@host2 () Bool
  true)
  ....
</ParsingString>
```