FUTEBOL

Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory

FUTEBOL UFRGS User Manual

Draft

Revision 0.1

Authors	Hugo Santos - Universidade Federal do Rio Grande do Sul Gustavo Araújo - Universidade Federal do Rio Grande do Sul
Version	0.1
Abstract	This document is a manual for the end-users of the FUTEBOL UFRGS testbed. It describes how to reserve the resources available at the UFRGS testbed, and also presents simple experiments that can be performed using those resources. Using those examples, the user will be able to build his/her own experiments.

This project has received funding from the European Union's Horizon 2020 for research, technological development, and demonstration under grant agreement no. 688941 (FUTEBOL), as well from the Brazilian Ministry of Science, Technology and Innovation (MCTI) through RNP and CTIC.

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	12/11/2017	Description of how to allocate Raspberry Pis, COPA and ZigBee	

Table of Contents

1 Testbed Architecture

In this testbed, we address the needs of an optical-wireless infrastructure to provide ubiquitous Internet of Things (IoT) communication, involving devices ranging from low complexity sensors and actuators (e.g., luminosity and smart light bulbs) to more advanced ones (e.g., multimedia sensors and smart glasses). We employ 16 Raspberries Pi connected to 8 XBee ZigBee RF modules through XBee Explorer USB adapters. The raspberries are connected with 16 Arduinos equipped with XBee Arduino shields, temperature, luminosity and humidity sensors. Together these IoT devices enable the realization of data collection, data processing, computing routine delegation, event creation based on monitoring results and data storage in the cloud. Therefore, this facility is ideally equipped to investigate the combination of various physical layer approaches into coexisting or coherent networks.

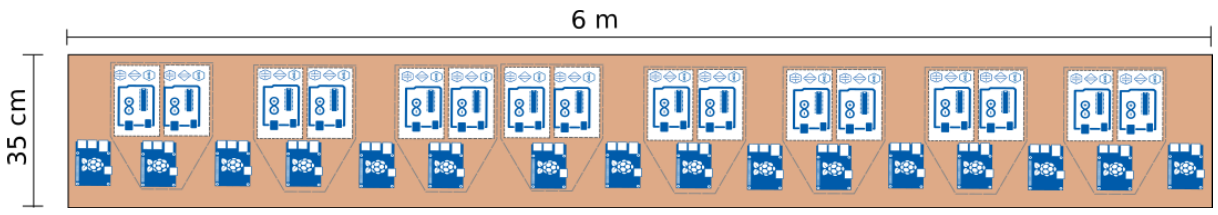


Fig 1. IoT Test layout

2 Functional Layers

Logically, one can think of the testbed as consisting of four layers: The bottom layer provides the physical elements, such as servers, raspberries, arduino, XBees, storage, etc., that can be controlled through one hypervisors. The next layer corresponds to the virtualized testbed, comprising containers/VM associated to the physical IoT devices. Finally, at the top sits the definition of each experiment that uses the resources provided by the lower layer.

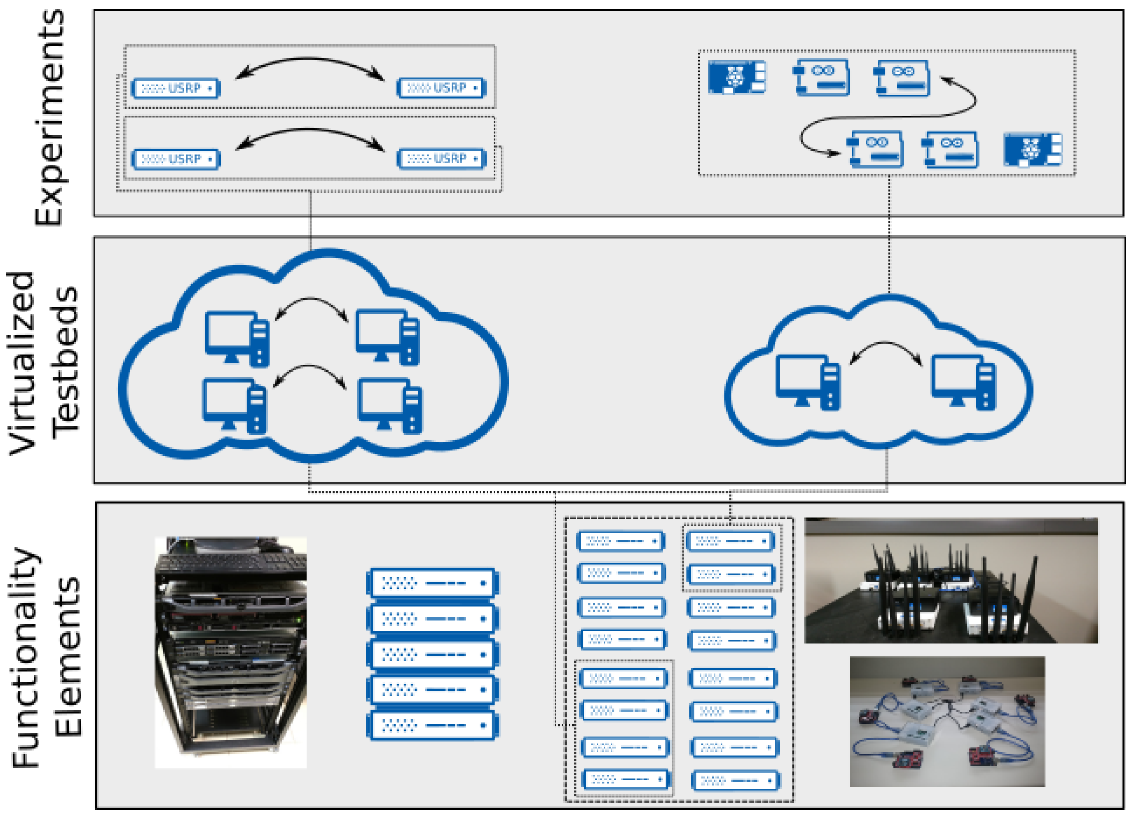


Fig 2. Functional layers

3 Setting up an Experiment

With their own hardware, users will be able to setup an experiment using jFed. This allows users to send information that will communicate with the Aggregate Manager (AM) gateway for authentication and to receive information about what resources are available. The AM will interact with the Cloud Based Testbed Manager (CBTM) to setup Experimentation Units. The CBTMs act as software defined radio transmitters and receivers.

After the user send information, the AM will authenticate users, tell them which resources will be available to them through RSpecs, and interact with the CBTM on behalf of the users in order to instantiate the available resources. The AM uses the GENI v3 API which is written as a wrapper of the reference AM.

The principal responsibility of the CBTM is to create virtual machines in the servers, where the users will be able to run experiment specific software. The CBTM will interact with the KVM Hypervisor. We are currently running CBTM v2.0, which is written in Python and use libvirt libraries to interact with the KVM hypervisor.

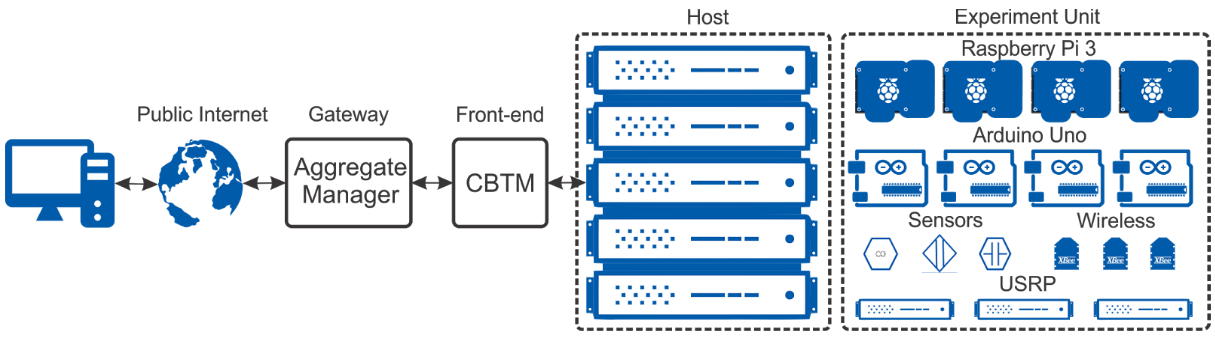


Fig 3. IoT testbed diagram

4 IoT Experimentation Units

An IoT Experimentation Unit consists of **one** Raspberry Pi and **two** Arduinos with the following capabilities:

The hardwares for experimentation are:

- Raspberry Pi 3 Model B
  - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
  - 1GB RAM
  - BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- Arduino Uno
  - ATmega328P 16MHz
  - Flash Memory 32KB
  - SRAM 2KB
  - EEPROM 1KB
- XBee S2C with Explorer USB Adapter
- XBee S2C Arduino Shield
- Humidity, luminosity and temperature sensors

The Experimentation unit are as following:

- Raspberry equipped with 2 Arduinos sensor set and a Arduino XBee Shield labelled as **raspberry-arduino-sensor**
- Raspberry equipped with 2 Arduinos and a XBee Shield labelled as **raspberry-arduino**
- Raspberry equipped with 1 XBee Explorer USB labelled as **raspberry-xbee**

Up to 6 units as raspberry-arduino-sensor, up to 2 raspberry-arduino and up to 8 raspberry-xbee can be used in an experiment, and users simply SSH into the raspberry to run the experiments.

5 IoT Federation and RSpec Description

The UFRGS testbed is managed by a software stack, constituted by the Aggregate Manager, the Coordinator and the Cloud Based Testbed Manager (CBTM). This stack is accessed by the jFed Experimenter GUI at the entry point of the testbed, which is the Aggregate Manager. The resources provided by the testbed include: virtual machines and LAN connectivity. Their slicing is performed as follows:

- For **virtual machines:** The selection of which type of virtualization will be based on the sliver type, which separates the available resources by the available hardware, and the operating system image. According to the selected sliver, a different hardware configuration will be made available, for example, a Raspberry Pi. An OS image can also be specified in order to further customize the node, for example, selecting an image with a specific software pre-installed.
- **LAN connectivity:** interconnectivity is provided transparently to the experimenter either using virtual or hardware switches. For the hardware switches, the slicing is performed using a divisor (e.g. FlowVisor, OpenVirteX [FLOWVISOR, OPENVIRTEX]), in order to isolate the networks from different experimenters.

IoT Nodes: the Experimenter can choose between access to a:

- Raspberry with XBee USB Explorer Adapter and Raspbian OS. There are up to **8** and can be allocated informing the RSpec the sliver type “**raw-raspberry**”.

The **Raspberry Pi mote** with **XBee USB Explorer Adapter** RSPEC can be set as follows:

```
<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+32">
   <sliver_type name="raw-raspberry"/>
   <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="331.0" y="127.0"/>
  </node>
```

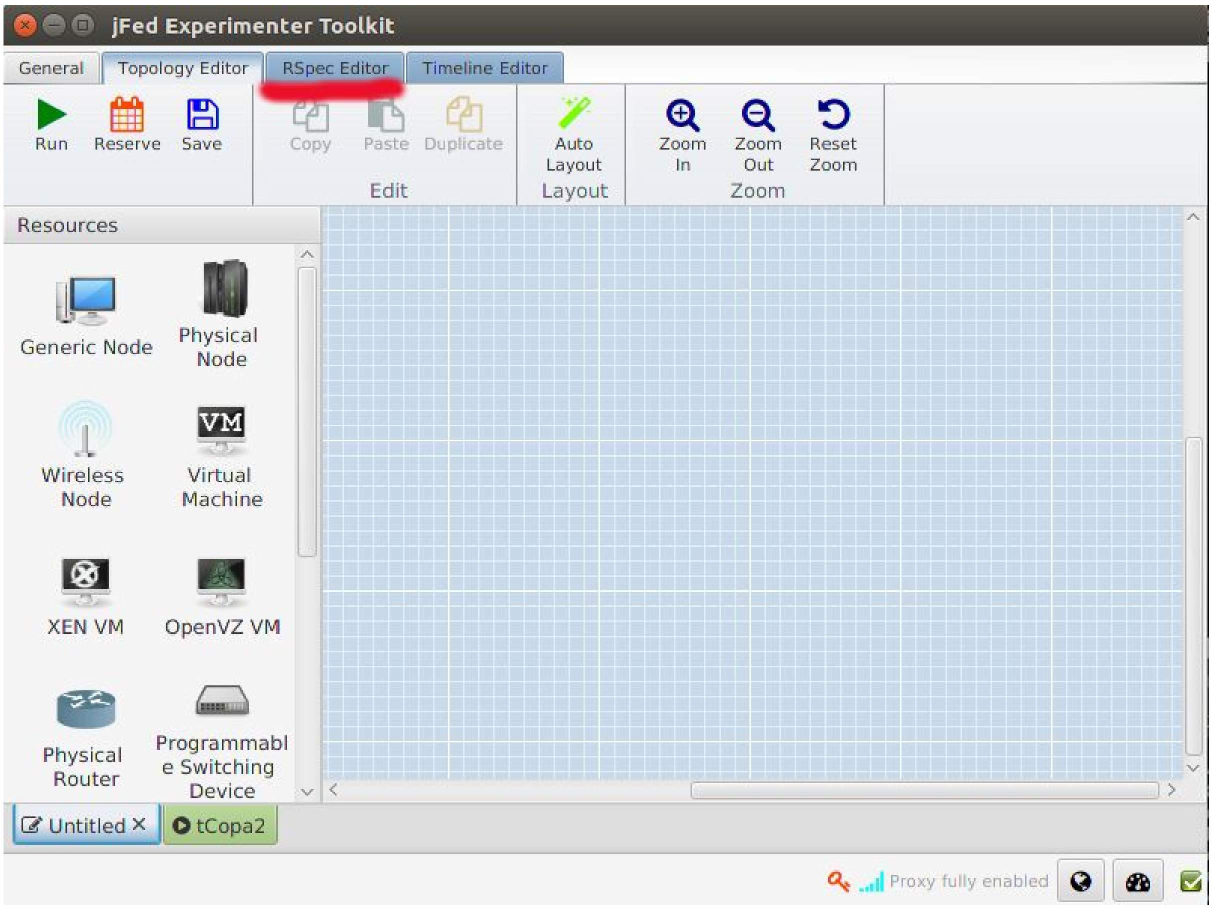
6 IoT Tutorial

6.1 Experiment Description

An example to deploy a simples experiment with IoT is to reserve two **raw-raspberry** units to communicate between each other. The raspberry #1 runs a python serial data sender application to the XBee module. The XBee module of raspberry #2 collects the data through a python serial data receiver application and prints on a terminal screen.

6.2 Resource Allocation

At jFed, the following RSPEC allocates two raspberries in the UFRGS testbed.



```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request" generated_by="jFed
RSpec Editor" generated="2017-11-17T10:15:21.848-02:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vlan/1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+31">
    <sliver_type name="raw-raspberry"/>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="131.5" y="128.5"/>
  </node>
  <node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+32">
    <sliver_type name="raw-raspberry"/>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="331.0" y="127.0"/>
  </node>
</rspec>
```

6.3 Coding

For this to occur, firstly the XBee module are pre-configured with PAN ID 1234, API mode 1 and all of them runs as coordinator to simplify the network creation. The main part of the receiver application sets the given USB port of the USB Xplorer Xbee and the bandwidth rate. After that, a Xbee demands a initialization in bypass mode to properly work at this configuration. The describe part of the is as follows:

```
private static final String PORT = "/dev/ttyUSB0";
private static final int BAUD_RATE = 9600;
...
XBeeDevice myDevice = new XBeeDevice(PORT, BAUD_RATE);
try {
        SerialPortRxTx serPort = new SerialPortRxTx(PORT, BAUD_RATE);
        serPort.open();
        serPort.writeData("\n\n".getBytes()); //wake boot menu up
        Thread.sleep(1000);
        serPort.writeData("B".getBytes()); //initiate bypass mode
        Thread.sleep(1000);
        serPort.close();

        myDevice.open();
        myDevice.addDataListener(new MyDataReceiveListener());
        System.out.println("\n>> Waiting for data...");
        ...
```

The Receiver Java project can be found on path:
~/projeto/XBJL-1.2.1/examples/communication/ReceiveDataSample

The sender application also sets USB port and serial bandwidth rate, also initialize at bypass mode and periodically sends the message “Hello! I am RECEIVER, USB1!”. The main part of the code are as follows:

```
private static final String PORT = "/dev/ttyUSB1";
private static final int BAUD_RATE = 9600;
private static final String DATA_TO_SEND = "Hello! I am RECEIVER, USB1!";
private static final String REMOTE_NODE_IDENTIFIER = "SENDER";

public static void main(String[] args) {
    ...
    XBeeDevice myDevice = new XBeeDevice(PORT, BAUD_RATE);
    byte[] dataToSend = DATA_TO_SEND.getBytes();
    try {
        SerialPortRxTx serPort = new SerialPortRxTx(PORT, BAUD_RATE);
        serPort.open();
        serPort.writeData("\n\n".getBytes()); //wake boot menu up
        Thread.sleep(1000);
        serPort.writeData("B".getBytes()); //initiate bypass mode
        Thread.sleep(1000);
        serPort.close();

        myDevice.open();
        // Obtain the remote XBee device from the XBee network.
        XBeeNetwork xbeeNetwork = myDevice.getNetwork();
        RemoteXBeeDevice remoteDevice =
xbeeNetwork.discoverDevice(REMOTE_NODE_IDENTIFIER);
        if (remoteDevice == null) {
            System.out.println("Couldn't find the remote XBee device with '" +
REMOTE_NODE_IDENTIFIER + "' Node Identifier.");
            System.exit(1);
        }

        System.out.format("Sending data to %s >> %s | %s... ",
remoteDevice.get64BitAddress(),
           HexUtils.prettyHexString(HexUtils.byteArrayToHexString(dataToSend)),
           new String(dataToSend));
        myDevice.sendData(remoteDevice, dataToSend);
        System.out.println("Success");
        ...
```

The Sender Java project can be found on path:
~/projeto/XBJL-1.2.1/examples/communication/SendDataSample

6.4 Generating and running the executables

Firstly, the code need to be interpreted by the JDK and the steps for the **Receiver** application are as follows:

```
cd ~/projeto/XBJL-1.2.1/examples/communication/ReceiveDataSample

javac -sourcepath src -classpath
"libs/xbee-java-library-1.2.1.jar:libs/rxtx-2.2.jar:libs/slf4j-api-1.7.12.jar:libs/slf4j-nop-1.7.12.j
```

```
ar:libs/android-sdk-5.1.1.jar:libs/android-sdk-addon-3.jar" -d bin
src/com/digi/xbee/api/receivedata/*.java

cd bin/

jar cvfm ../myReceiveDataSample.jar ../manifest.mf com/digi/xbee/api/receivedata/*.class

cd ..
```

Set run permission for the generated JAR:

```
chmod +x myReceiveDataSample.jar
```

And finally run it with:

```
java -jar myReceiveDataSample.jar
```

A similar process occurs for the Sender application and the terminal are as follows:

```
cd ~/projeto/XBJL-1.2.1/examples/communication/SendDataSample

javac -sourcepath src -classpath
"libs/xbee-java-library-1.2.1.jar:libs/rxtx-2.2.jar:libs/slf4j-api-1.7.12.jar:libs/slf4j-nop-1.7.12.j
ar:libs/android-sdk-5.1.1.jar:libs/android-sdk-addon-3.jar" -d bin
src/com/digi/xbee/api/senddata/MainApp.java

cd bin/

jar cvfm ../mySendDataSample.jar ../manifest.mf
com/digi/xbee/api/senddata/MainApp.class

cd ..
```

Set run permission as:

```
chmod +x mySendDataSample.jar
```

And run it with:

```
java -jar mySendDataSample.jar
```

To test this scenario, open two ssh sessions on the **raspberry-xbee** unit and run the program senddata.java on raspberry #1 and receivedata.java on raspberry #2.

The complete code is available on annex.That is it! Enjoy!

7 COPA Experiment units

Virtualized resources are divided in container pool VM, that is the container support system and, COPA VM, which is the web-based administration interface. The virtualialized resource for experimentation are:

- Virtual Machine
    - Virtualized single core CPU
    - 4GB RAM memory
    - Storage size 12GB

The Experimentation unit are as following:

- VM carrying COPA tool with:
    - sliver_type: default-vm
    - disk_image: urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+copa
- VM carrying a container pool with:
    - sliver_type: default-vm
    - disk_image: urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+pool

Both COPA VM and container pool VM are using the same amount of virtualized resources (e.g. CPU, memory, etc).

8 Federation and RSpec Description

The UFRGS testbed is managed by a software stack, constituted by the Aggregate Manager, the Coordinator and the Cloud Based Testbed Manager (CBTM). This stack is accessed by the jFed Experimenter GUI at the entry point of the testbed, which is the Aggregate Manager. The resources provided by the testbed include: virtual machines and LAN connectivity. Their slicing is performed as follows:

- For **virtual machines:** The selection of which type of virtualization will be based on the sliver type, which separates the available resources by the available hardware, and the operating system image. According to the selected sliver, a different hardware configuration will be made available, for example, a COPA VM. An OS image can also be specified in order to further customize the node, for example, selecting an image with a specific software pre-installed.

Those resources provide some parameters that can be controlled remotely, e.g. by a SDN controller. Each type of device will provide different parameters, which are listed below, as well as how those are accessed:sources provide some parameters that can be controlled remotely, e.g. by a SDN controller.

The **COPA VM** RSPEC can be set as follows:

```
<node client_id="copa" exclusive="true"
```

```
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+5">
    <sliver_type name="default-vm">
      <disk_image name="urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+copa"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="103.0" y="99.5"/>
  </node>
```

The **Pool VM** RSPEC can be set as follows:

```
<node client_id="central" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+4">
    <sliver_type name="default-vm">
      <disk_image name="urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+pool"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="259.0" y="99.5"/>
  </node>
```

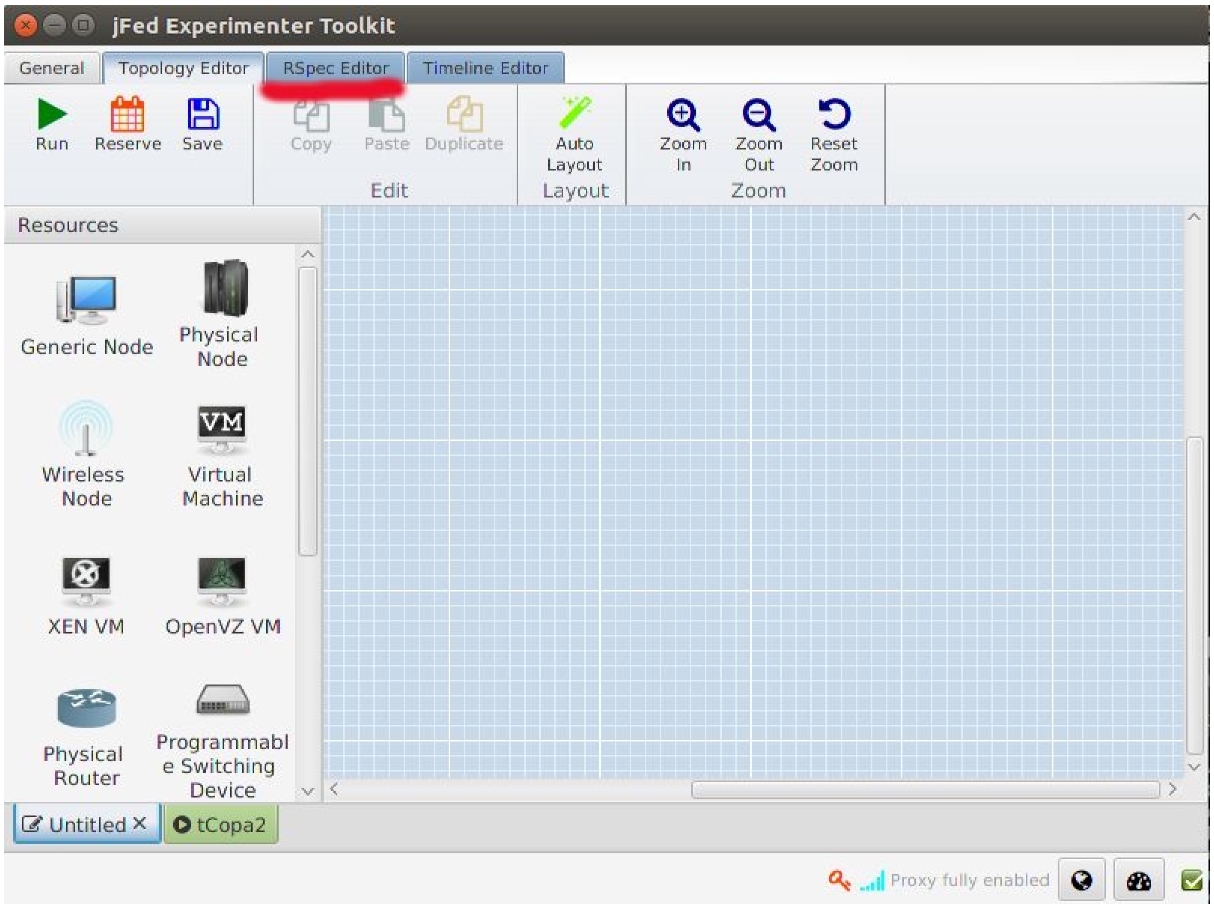
9 COPA Tutorial

9.1 Experiment Description

An example to deploy a simple experiment with COPA is to reserve two VMs units to manage through COPA. The VM #1 runs the COPA tool. The VM #2 runs an ordinary VM and the management is made by the web interface of COPA.

9.2 Resource Allocation

At jFed, edit the RSpec in the RSpec Editor tab. This shows how to set RSPECs Editor at jFed.



And edit the text as follows:

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request" generated_by="jFed
RSpec Editor" generated="2017-10-24T12:44:31.024-02:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vlan/1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="central" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+4">
    <sliver_type name="default-vm">
      <disk_image name="urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+pool"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="259.0" y="99.5"/>
  </node>
  <node client_id="edge" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
```

```
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+6">
    <sliver_type name="default-vm">
      <disk_image name="urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+pool"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="429.0" y="100.5"/>
  </node>
  <node client_id="copa" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am"
component_id="urn:publicid:IDN+futebol.inf.ufrgs.br+authority+am+node+5">
    <sliver_type name="default-vm">
      <disk_image name="urn:publicid:IDN+futebol-cbtm.inf.ufrgs.br+image+copa"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="103.0" y="99.5"/>
  </node>
</rspec>
```

9.3 Register VMs

Double click on the edge and central nodes to open up a terminal window. Accept the ssh connection and type **ifconfig** to obtain the ip address on the **lxdbr0** interface.

In this example case, the ip addresses were 192.168.5.83 for the central node. For the edge node was 192.168.5.84.

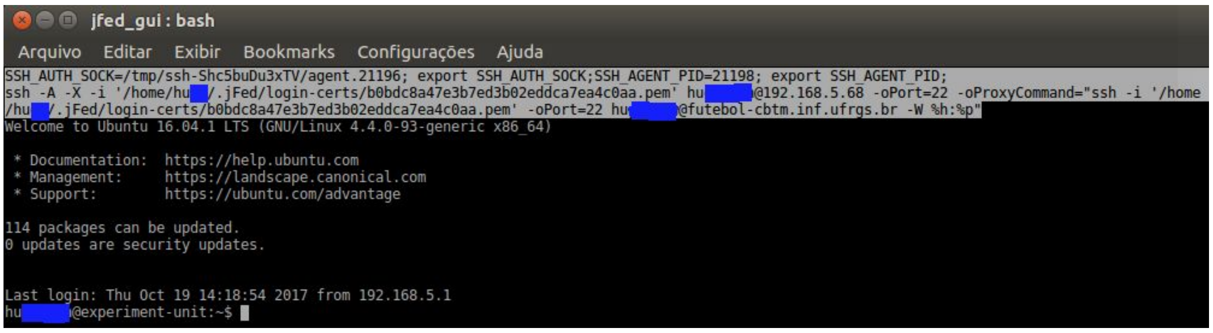
Open up a terminal window on the copa node, accept the ssh connection and edit the file located at **/copa/server.txt 99 53**

In this example case, the content of the file is as follows:

```
Central;192.168.5.83:8443
Edge;192.168.5.84:8443
```

9.4 Tunneling the COPA Web Interface

For this to occur,open up a terminal on the COPA VM, and copy the highlighted command from the terminal screen. Note that the parameter of this messages everytime you load an experiment at jFed. You need to follow some steps. The Figure X shows the highlighted command on the terminal screen.



```
SSH_AUTH_SOCK=/tmp/ssh-Shc5buDu3xTV/agent.21196; export SSH_AUTH_SOCK;SSH_AGENT_PID=21198; export SSH_AGENT_PID;
ssh -A -X -i '/home/hu[redacted]/.jFed/login-certs/b0bdc8a47e3b7ed3b02eddca7ea4c0aa.pem' hu[redacted]@192.168.5.68 -oPort=22 -oProxyCommand="ssh -i '/home
/hu[redacted]/.jFed/login-certs/b0bdc8a47e3b7ed3b02eddca7ea4c0aa.pem' -oPort=22 hu[redacted]@futebol-cbtm.inf.ufrgs.br -W %h:%p"
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-93-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

114 packages can be updated.
0 updates are security updates.


Last login: Thu Oct 19 14:18:54 2017 from 192.168.5.1
hu[redacted]@experiment-unit:~$ 
```

In this example case, the copied command is:

```
SSH_AUTH_SOCK=/tmp/ssh-Shc5buDu3xTV/agent.21196; export
SSH_AUTH_SOCK;SSH_AGENT_PID=21198; export SSH_AGENT_PID;
ssh -A -X -i
'/home/hu/.jFed/login-certs/b0bdc8a47e3b7ed3b02eddca7ea4c0aa.pem'
hu@192.168.5.68 -oPort=22 -oProxyCommand="ssh -i '/home
/hu/.jFed/login-certs/b0bdc8a47e3b7ed3b02eddca7ea4c0aa.pem' -oPort=22
hu@futebol-cbtm.inf.ufrgs.br -W %h:%p"
```

Open up a terminal on your computer and adds the string to the end of the copied command:

```
-L 8000:localhost:8000 -L 8100:localhost:8100
```

The tunneling then will be made with the command bellow:

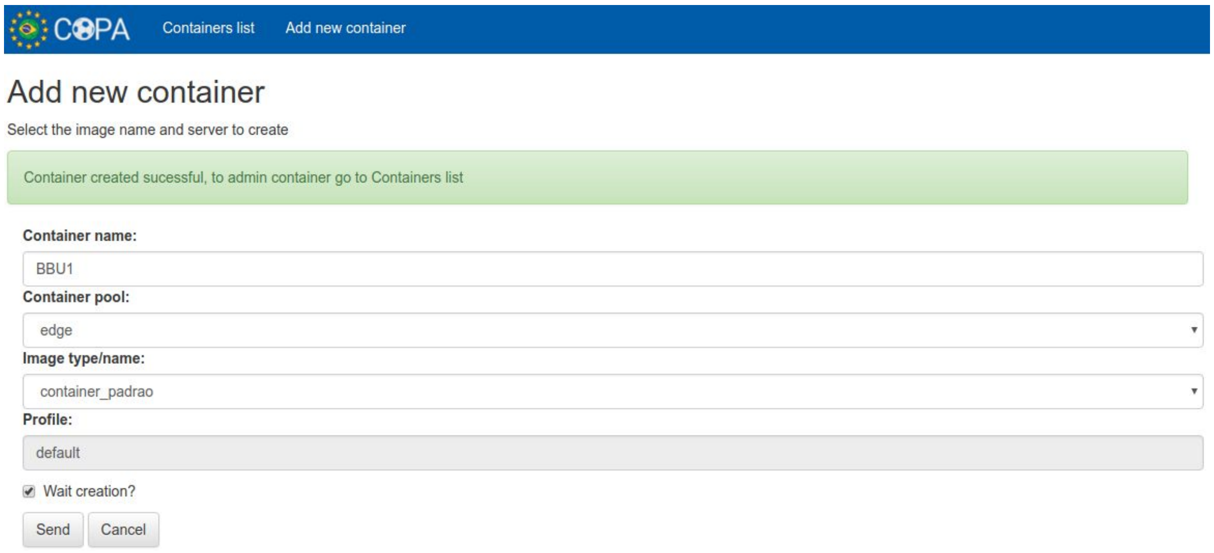
```
SSH_AUTH_SOCK=/tmp/ssh-Shc5buDu3xTV/agent.21196; export
SSH_AUTH_SOCK;SSH_AGENT_PID=21198; export SSH_AGENT_PID;
ssh -A -X -i
'/home/hu/.jFed/login-certs/b0bdc8a47e3b7ed3b02eddca7ea4c0aa.pem'
hu@192.168.5.68 -oPort=22 -oProxyCommand="ssh -i '/home
/hu/.jFed/login-certs/b0bdc8a47e3b7ed3b02eddca7ea4c0aa.pem' -oPort=22
hu@futebol-cbtm.inf.ufrgs.br -W %h:%p" -L 8000:localhost:8000 -L
8100:localhost:8100
```

Next, open up your browser and type the address:

```
http://localhost:8000/core/welcome/
```

9.5 Creating container in COPA Web Interface

Browse to the add container tab and add two containers. Name your container and pick which node will host the container in the Container pool dropdown. In this example we added BBU1 to the edge node, shown on Figure 4, and GW1 to central node, shown on Figure 5. Remember to wait up to 2 minutes for the container creation message on the top of the screen.



Add new container

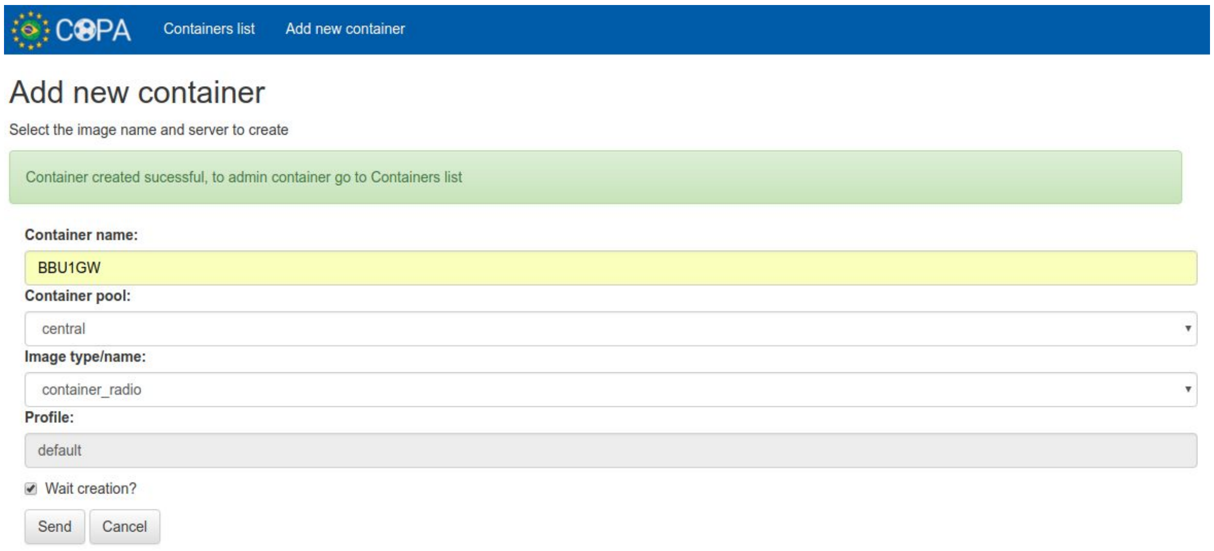
Select the image name and server to create

Container created sucessful, to admin container go to Containers list

Container name:	BBU1
Container pool:	edge
Image type/name:	container_padrao
Profile:	default

☑ Wait creation?

Figure 4. Creation of BBU1 on the edge Container pool.



Add new container

Select the image name and server to create

Container created sucessful, to admin container go to Containers list

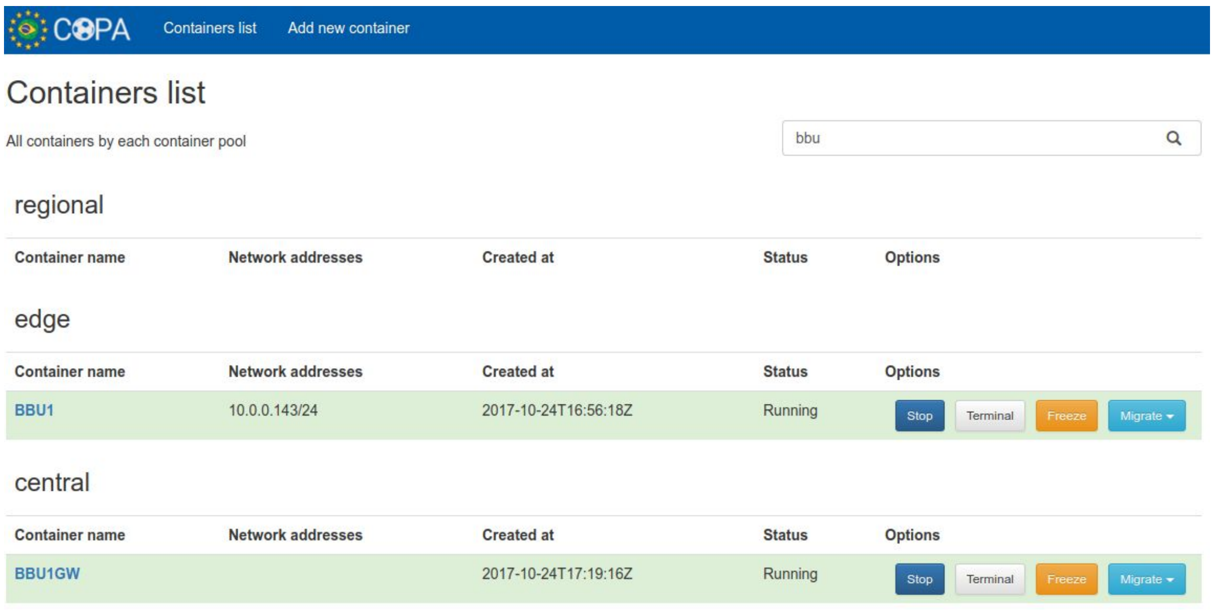
Container name:	BBU1GW
Container pool:	central
Image type/name:	container_radio
Profile:	default

☑ Wait creation?

Figure 5. Creation of GW1 on the central Container pool.

9.6 COPA Web Interface

The COPA web interface shows some informations about the created containers and it provides some operations, such as, start or stop container, open a terminal window, freeze or migrate a container to another container pool. Figure 6 shows the COPA web interface.



Containers list

All containers by each container pool

regional

Container name	Network addresses	Created at	Status	Options

edge

Container name	Network addresses	Created at	Status	Options
BBU1	10.0.0.143/24	2017-10-24T16:56:18Z	Running	Stop Terminal Freeze Migrate

central

Container name	Network addresses	Created at	Status	Options
BBU1GW		2017-10-24T17:19:16Z	Running	Stop Terminal Freeze Migrate

Figure 6. Container list