



Grant Agreement
 Call:
 Topic:
 Type of action: RIA

No.: 732638
 H2020-ICT-2016-2017
 ICT-13-2016



D2.4: Testbed requirements, developments and integrations

Work package	WP 2
Task	Task 2.2
Due date	31/12/2017
Submission date	10/11/2018
Deliverable lead	Imec
Version	4
Authors	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU)
Reviewers	Peter Van Daele (imec)

Abstract	This deliverable describes the specific requirements, developments and integrations that were done in the first 21 months to add new testbeds to the federation.
Keywords	Testbed integration, federation, testbed support

D2.4: Testbed requirements, developments and integrations

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1	15/08/2018	TOC	Brecht Vermeulen (imec)
V2	1/11/2018	First complete version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU)
V3	8/11/2018	Almost final version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU)
V4	10/11/2018	Final version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU)

DISCLAIMER

The information, documentation and figures available in this deliverable are written by the **Federation for FIRE Plus (Fed4FIRE+)**; project's consortium under EC grant agreement **732638** and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

COPYRIGHT NOTICE

© 2017-2021 Fed4FIRE+ Consortium

ACKNOWLEDGMENT



Co-funded by the
European Union



Co-funded by the
Swiss Confederation

This deliverable has been written in the context of a Horizon 2020 European research project, which is co-funded by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.



D2.4: Testbed requirements, developments and integrations

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:		R
Dissemination Level		
PU	Public, fully open, e.g. web	<input checked="" type="checkbox"/>
CL	Classified, information as referred to in Commission Decision 2001/844/EC	<input type="checkbox"/>
CO	Confidential to FED4FIRE+ project and Commission Services	<input type="checkbox"/>

* *R*: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

D2.4: Testbed requirements, developments and integrations

EXECUTIVE SUMMARY

This deliverable describes the specific requirements, developments and integrations that were done in the first 21 months to add new testbeds to the federation. In total 21 testbeds were added to the federation, of 13 different types/implementations.

The federation template that we support together with Geni (Geni Control Framework, GCF) has been further extended, a.o. with general GDPR support. This results in a Docker AM which is fully functional for docker resources (including IPv6).



TABLE OF CONTENTS

DISCLAIMER	2
COPYRIGHT NOTICE	2
ACKNOWLEDGMENT	2
1 TESTBED FEDERATION EFFORTS	8
1.1 GENERAL DEVELOPMENT FOR FEDERATING TESTBEDS	8
1.2 FEDERATED TESTBEDS	9
1.2.1 FUTEBOL Brazil/UFES	9
1.2.2 FUTEBOL Brazil/UFMG	9
1.2.3 Futebol Brazil/UFRGS	10
1.2.4 FUTEBOL VTT	10
1.2.5 OWL TUD	10
1.2.6 ARNO Sant'Anna Pisa Testbed.....	11
1.2.7 FIT R2Lab	11
1.2.8 ESAT MM Testbed.....	11
1.2.9 City of Things Antwerp	11
1.2.10 Other testbeds.....	12
2 SUPPORT FOR TESTBEDS WANTING TO FEDERATE	13
3 INFORMATION FOR AM DEVELOPERS	14
3.1 ADDING AN AM TO JFED	14
3.1.1 Requirements.....	14
3.1.2 Server X509 Certificate	16
3.1.3 Allowing access to federated users	17
3.1.4 Configuring the AM GetVersion call.....	18
3.1.5 Using the jfed scanner tool.....	19
3.1.6 Local files for adding testbeds	21
3.1.7 Adding to jFed for all users.....	21
3.1.8 Adding your AM to one of the experiment GUI “icons”	23
3.2 HANDLING TESTBED SPECIFIC TERMS & CONDITIONS (I.E. HANDLING GDPR)	23
3.2.1 jFed support for Testbed Specific Terms & Conditions (T&C).....	23
3.2.2 jFed T&C integration javascript details	24
3.2.3 Example: standalone T&C site	25
3.2.4 Example: AM with T&C site built in.....	26
3.2.5 Add T&C site info to central jFed config.....	26
3.2.6 Make the AM reject users that did not approve the Terms & Conditions.....	27
3.3 DEBUGGING YOUR AM	27
3.3.1 jFed Probe GUI	28
3.3.2 jFed automated tester	28
3.3.3 Permanent monitoring: Fedmon	33
3.4 RSPEC DETAILS FOR AM DEVELOPERS	34
3.4.1 General	34
3.4.2 Choosing your component manager urn.....	34
3.4.3 RSpec basics: sliver_type and exclusive	36
3.4.4 RSpec basics: Disk Images	37
3.4.5 RSpec basics: Specific nodes	38
3.4.6 RSpec basics: Hardware type	38
3.4.7 Advertisement Examples: Bare metal access.....	39
3.4.8 Advertisement Examples: Simple VM's	40

D2.4: Testbed requirements, developments and integrations

3.4.9	Advertisement Examples: Complex VM's: multiple sizes	41
3.4.10	Advertisement Examples: Specialised hardware connected to single VM box.....	41
3.4.11	Advertisement Examples: Combining Bare metal and VMs on a single node	42
3.4.12	Request RSpec.....	43
3.5	STITCHING DETAILS FOR AM DEVELOPERS	44
3.5.1	General	44
3.5.2	“Real” Stitching: automated multiple hops and VLAN translation	44
3.5.3	Stitching alternative: Dedicated Ext. Network Connection	46
3.5.4	Stitching alternative: “tunnels” over internet using link_type	47
4	DOCKER-AM AS EXAMPLE AM.....	49
4.1	SUPPORTED AGGREGATE MANAGER FEATURES	49
4.2	HOW TO INSTALL THE AM ?	49
4.2.1	Dependencies	49
4.2.2	Download source code.....	50
4.2.3	Configure AM	50
4.2.4	Configure a DockerMaster.....	51
4.2.5	Generate certificate and key.....	52
4.3	STARTING THE AM.....	53
4.4	TRUST YOUR C-BAS INSTALLATION	53
4.5	CONFIGURING A REMOTE DOCKERMANAGER (OPTIONAL).....	53
4.5.1	Configure the remote.....	53
4.5.2	Configure the AM	54
4.6	HOW TO ADAPT THIS AM TO YOUR INFRASTRUCTURE ?	54
4.7	DEVELOPMENT NOTES	55
4.8	ADDITIONAL INFORMATIONS	56
4.9	TROUBLESHOOTING	56



D2.4: Testbed requirements, developments and integrations

LIST OF FIGURES

FIGURE 1: OVERVIEW OF JFED RESOURCE ICONS, INCLUDING NEW ICONS FOR RASPBERRY, 5G, IOT AND GTS (GEANT TESTBED AS A SERVICE)	8
FIGURE 2: THE NEW RASPBERRY PI AND IOT ICONS IN JFED.....	10
FIGURE 3: THE NEW ICONS IN JFED, INCLUDING THE 5G ICON	11
FIGURE 4: A LINK BETWEEN CITY OF THINGS NODES IN JFED.....	11
FIGURE 5: JFED AUTOMATED TESTER: OVERVIEW OF TESTS.....	30
FIGURE 6: JFED AUTOMATED TESTER: RUN A SPECIFIC TEST	31
FIGURE 7: JFED AUTOMATED TESTER: TEST RESULTS	32
FIGURE 8: JFED AUTOMATED TESTER: TEST RESULT DETAILS.....	33



1 TESTBED FEDERATION EFFORTS

1.1 GENERAL DEVELOPMENT FOR FEDERATING TESTBEDS

The documentation aimed at assisting testbed owners when federating their testbeds, was extended on many points, based on feedback from the testbeds. This documentation can be found at: <https://doc.ilabt.imec.be/jfed-documentation-5.9/amdevelopers> (see also section 3)

Inside the jFed experiment GUI tool, support for different link types was extended to manage the needs of the Futebol testbeds. jFed will now automatically select the right type of link, based on which nodes (of which testbeds) are connected with each other. This makes it much easier for users, and reduces the need to know all technical details about some testbed links.

Figure 1 gives an overview of the current support resource types (icons on the left), and examples of the new types in the canvas at the right.

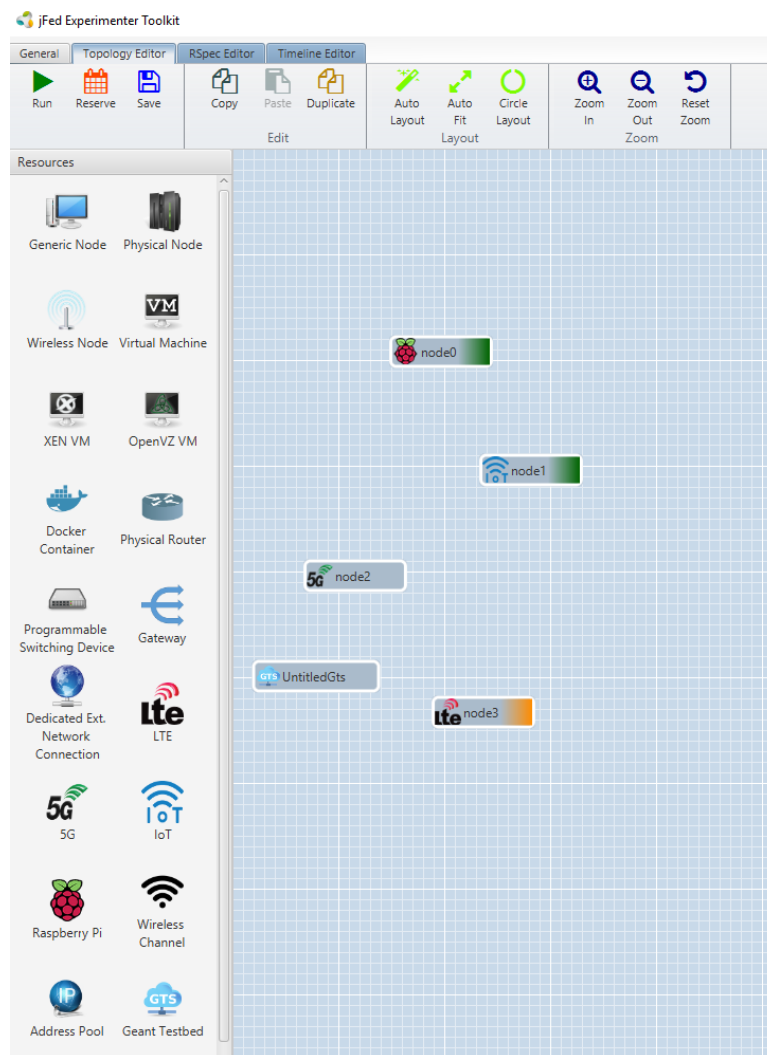


Figure 1: Overview of jFed resource icons, including new icons for Raspberry, 5G, IoT and GTS (Geant testbed as a service)

D2.4: Testbed requirements, developments and integrations

1.2 FEDERATED TESTBEDS

1.2.1 FUTEBOL Brazil/UFES

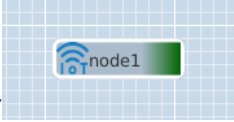
The FUTEBOL testbed of the Federal University of Espírito Santo was federated during April 2017 to Dec 2017. This testbed offers access to VMs. A lot of assistance was required, but no new jFed developments were needed.

1.2.2 FUTEBOL Brazil/UFMG

The FUTEBOL testbed of the Universidade Federal de Minas Gerais was federated begin 2017. This testbeds offers access to a wide range of experimental hardware: USRP (software defined radio) hardware, TelosB sensors, bare metal wifi nodes, and bare metal Raspberry Pi nodes.

jFed was extended with more complex support for hardware and sliver types. This allowed

adding more flexible “icons” in the jFed GUI. A Raspberry Pi () and an IoT

icon () were then added and linked to the appropriate configuration of the UFMG testbed.

D2.4: Testbed requirements, developments and integrations

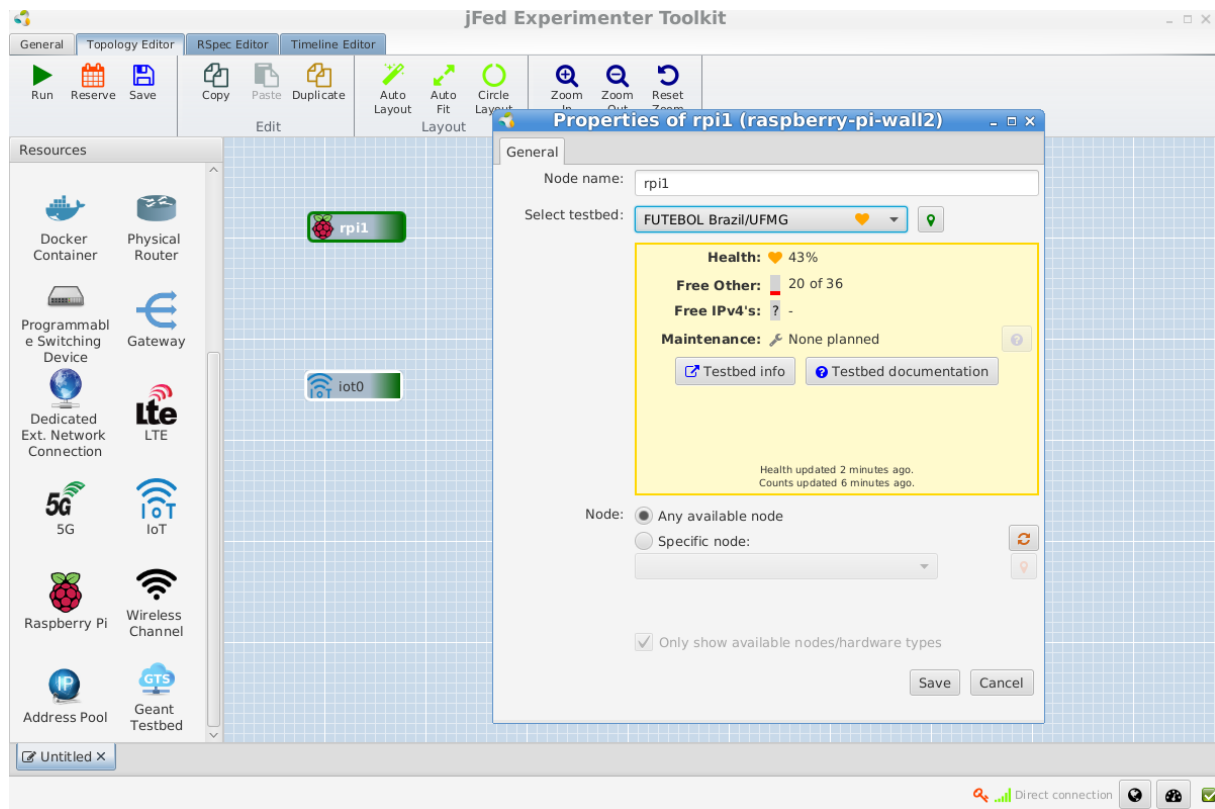


Figure 2: The new Raspberry Pi and IoT icons in jFed

1.2.3 Futebol Brazil/UFRGS

The FUTEBOL testbed of the Federal University of Rio Grande do Sul was federated begin 2017. This testbed offers access to USRP (software defined radio) hardware, VMs, and bare metal Raspberry Pi nodes.

The federation work required for FUTEBOL UFGM was developed in parallel for this testbed, which had the same requirements. The new Raspberry Pi icon in jFed was linked to the appropriate configuration of the UFGRS testbed.

1.2.4 FUTEBOL VTT

The FUTEBOL VTT testbed of the Technical Research Centre of Finland was federated in April 2018. Little federation support was needed, and no new jFed development was required.

1.2.5 OWL TUD

The Online Wireless Lab (OWL) testbed of the Technische Universität Dresden was federated in March 2018. A lot of testbed side debugging was required.

The biggest issue turned out to be the very long image load time. jFed was modified to handle long load times in a more user friendly way.

A 5G icons was added to jFed, and the testbed was linked to this icon.

D2.4: Testbed requirements, developments and integrations



Figure 3: The new icons in jFed, including the 5G icon

1.2.6 ARNO Sant'Anna Pisa Testbed

The ARNO Testbed at Sant'Anna Pisa was federated between June and Aug 2017. The testbed is based on the docker AM, and it took some testbed side debugging to get it federated. The testbed offers access to LTE equipment. No jFed development was required for this tested.

1.2.7 FIT R2Lab

The R2Lab testbed of FIT was federated May 2018. This testbed allows access to wireless hardware. No new jFed development was needed, and the testbed was added under the 5G icon.

1.2.8 ESAT MM Testbed

The ESAT Massive MIMO Testbed of KU Leuven was federated between October and December 2017.

We helped develop the modified GCF AM for this testbed, as some complex feature were needed. jFed was also extended to support some of these features, in particular, support for connecting to windows nodes using RDP was added, and support for working with gateway nodes added by the testbed (not requested by the user) was added. The testbed was added under the 5G icon in jFed.

1.2.9 City of Things Antwerp

The City of Things testbed in Antwerp was federated. This testbed required little federation work. Because the testbed uses special links by default (gre links), jFed was extended to clearly differentiate between different link types in the GUI.

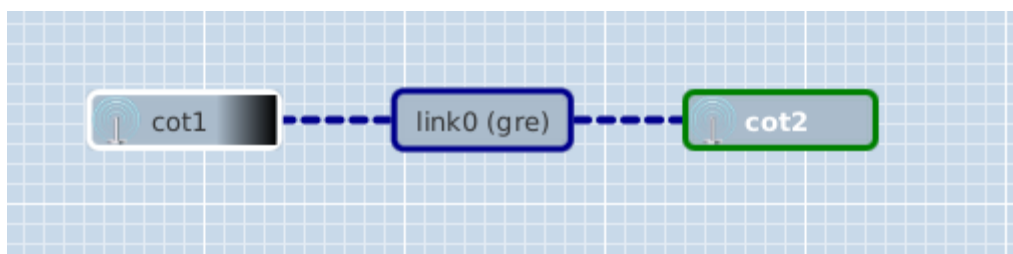


Figure 4: A link between City of Things nodes in jFed

D2.4: Testbed requirements, developments and integrations

1.2.10 Other testbeds

Two development and test servers were added (for University Bristol and UFMG). These are not production testbeds that can be used by federation users.

A lot of new US ExoGeni and InstaGeni testbeds were federated. Adding these to the federation required almost no effort:

- ExoGENI CIENA HQ
- InstaGENI ODU
- InstaGENI Hawaii
- InstaGENI VT
- InstaGENI UVM
- InstaGENI Louisiana
- InstaGENI UTDallas
- InstaGENI UCSD
- InstaGENI Utc
- InstaGENI University of Washington
- InstaGENI Colorado



D2.4: Testbed requirements, developments and integrations

2 SUPPORT FOR TESTBEDS WANTING TO FEDERATE

For testbeds wanting to federate we have the following things in place:

- Documentation how to start: <https://doc.ilabt.imec.be/jfed-documentation-5.9/amdevelopers> (see also section 3).
 - This includes specific information on GDPR inclusion for testbeds: <https://doc.ilabt.imec.be/jfed-documentation-5.9/amdevelopers#handling-testbed-specific-terms-conditions-i-e-handling-gdpr>
- Tools for testing the Aggregate Manager: jFed Probe and jFed Automatic tester (<https://doc.ilabt.imec.be/jfed-documentation-5.9/otherjfedtools.html#overview>)
- An example/template Aggregate Manager with docker containers. This is used as well as front-end for an existing testbed. See <https://github.com/open-multinet/docker-am>
- Documentation on the AM API: <https://github.com/open-multinet/federation-am-api> and <https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/federation-am-api.html>
- Documentation on the Slice Authority and Member Authority APIs: <https://github.com/open-multinet/CommonFederation-SA-MA-API> and <https://geni-nsf.github.io/CommonFederationAPI/CommonFederationAPIv2.html>
- Continuous monitoring and testing, including detailed information when clicking through: <https://fedmon.fed4fire.eu> (see also Deliverable 3.2 for more information on this).

D2.4: Testbed requirements, developments and integrations

3 INFORMATION FOR AM DEVELOPERS

If you develop an AM (typically supporting the [Geni v3 AM API](#)), to enable your testbed to be used from tools such as jFed, there are some practical things you need to be aware of.

This documentation goes into details about how to add your AM to jFed, how to test your AM with jFed, and gives some info on the advertisement and request RSpec that your AM might want to support.

3.1 ADDING AN AM TO JFED

This section explains how to add details about your AM to jFed. This is needed in order to use the AM from the jFed software.

First of all, note that this section is about adding an AM, an Aggregate Manager (sometimes also known as CM, “Component Manager”). If you want to setup and use your own user and slice authority (SA and/or MA), there are some extra complications not mentioned here. (Feel free to ask us for additional info if you are in this case.) Note that if you federate your AM with the fed4fire SA and MA, there is no need to configure jFed to use the SA/MA of your testbed (if you have one), so there is also no need to configure it in jFed.

In short, this section assumes that you have a fed4fire account, and you will login to jFed with this account, in order to use your AM.

3.1.1 Requirements

You need a few things before you get started:

- A server to run the AM software on.
- A publicly reachable IP for that server. This needs to be either an IPv4 or an IPv6 address. We recommend both.
- A DNS name for that server, that resolves to the publicly reachable IP addresses of the server. (Recommendation: It’s nice if the DNS name refers to your testbed and is specific for your AM. Example: *am.mytestbed.example.com*) (Note: A DNS name is not strictly required, but *very highly* recommended)
- Choose a URN for your AM. This is of the form: *urn:publicid:IDN+DNSNAME+authority+am* where you replace *DNSNAME* by the DNS name of your AM. (Example: *urn:publicid:IDN+mytestbed.example.com+authority+am*). If you don’t have

D2.4: Testbed requirements, developments and integrations

a DNS name for your server use a globally unique name for your testbed instead. It's not recommended to use an IP address instead, though that can work. Never use *localhost* or *127.0.0.1*, and never use the default value of the AM software you use. (For more info, see [Choosing your component manager urn](#))

- Choose a port at which you server will run. There is no standard port in the specification, so a lot of different ports are used in practice (12369, 8010, ...). We recommend using port 443, if that is not in use by anything else. The advantage of using the default https port is that it is reachable through most firewalls, and the protocol is in fact using https.
- You need a X509 Server Certificate, because the AM server uses https. This can be a self signed certificate (jFed stores a list of these to make it work safely). However, in that case, make sure you configure the fields in your self signed certificate correctly. See the next section for more details.
- You probably have testbed resources that you want to make reachable to experimenters using SSH. There are 2 options, either you need to public IP addresses that you can assign to these nodes when needed (IPv4 or IPv6), or you need to have one machine with a publically reachable IP address (IPv4 recommended) act as a gateway. In both cases, make sure you have these address(es) available. You can combine the gateway machine with the machine on which the AM runs, but for security reasons, this is not recommended.

An example of these choices for the imec Virtual Wall2 testbed:

- IP: *193.191.148.179*
- DNS name: *wall2.ilabt.iminds.be*
- URN: *urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm* So the URN including the DNS name as TLD/TLA part of the URN. "authority" as "type" of the URN, and "cm" as "name" part of the URN. (We actually recommend "am" as name instead of "cm", but both are fine.)
- Server certificate:
 - Not self signed in our case, but signed by a trusted root.
 - "Subject Name" contains *CN=www.wall2.ilabt.iminds.be*, as well as details about our address and organisation
 - "Subject Alternative Name" contains *DNS:www.wall2.ilabt.iminds.be*, *DNS:boss.wall2.ilabt.iminds.be*, *DNS:authority.ilabt.iminds.be*

D2.4: Testbed requirements, developments and integrations

- Our server runs at port 12369, because emulab already uses port 443 for another webserver, with different SSL settings.
- The full URL at which our AM is reachable is: `https://www.wall2.ilabt.iminds.be:12369/protogeni/xmlrpc/am/3.0`
- Experimenters can reserve servers at our AM. All of these are then reachable with SSH using public IPv6 addresses. We also have a pool of IPv4 addresses which users can assign to servers if needed. In addition, we have a gateway (`bastion.test.iminds.be`, which has both an IPv4 and IPv6 address), which can be used by experimenters that do not have IPv6.

3.1.2 Server X509 Certificate

An AM uses a message format specified in [Geni AM specification](#), which is sent over XML-RPC, which is sent over HTTPS with client authentication. So an AM runs at an HTTPS server (= HTTP over SSL) that is configured to require client authentication.

This means that the first step is to correctly setup the SSL server. The first step is to configure the server's X509 certificate. This certificate is used to identify the server to the users. Because only the server has the private key matching the certificate, users can setup a secure connection.

For SSL on the public internet, it is required that your server's X509 certificate is signed by a trusted root. All browsers have a list of these trusted roots. This is in short how the internet is made secure.

jFed also uses the internet's trusted roots, so if your server certificate is signed by these, that should be enough. However, for convenience, jFed also allows you to use a self signed X509 certificate. You can then add this self signed certificate to the list of certificates that jFed trusts, and everything will work without requiring the hassle of acquiring a "real" X509 certificate signed by a trusted root.

There are some best practices to take into account, regarding the self signed X509 certificate that you generate for your AM:

- "Subject" field of the certificate must contain a "CN" that is the hostname of the server (NOT and IP, the DNS hostname!)
- The "X509v3 Subject Alternative Name" section, must contain a "DNS" entry, which is the hostname of your server (NOT and IP, the DNS hostname!)

D2.4: Testbed requirements, developments and integrations

Another best practice derived from these, is that your AM needs a DNS name, not just an IP address.

Note that you can also add your AM to jFed if these guidelines are not respected. Try to at least have a meaningful “Subject” “CN”, and avoid a “Subject” “CN” like “localhost” at all cost.

3.1.3 Allowing access to federated users

3.1.3.1 Federate with fed4fire User and Slice Authority

Your AM needs to be configured so it is federated with the fed4fire user and slice authority. How to do this depends on the AM software.

General instructions: You need to add [the fed4fire root certificate](#) to the list of trusted authorities.

AM specific instructions:

- For GCF, you need to put the root certificate file in the `gcf trusted_roots` directory. Example command to do this

```
curl http://users.atlantis.ugent.be/bvermeul/wall2.pem > ~/.gcf/trusted_roots/wall2.pem
```

The result of configuring this is:

- Your AM will allow (client-authenticated) SSL connections from fed4fire users. If not configured correctly, your AM might terminate SSL handshake from fed4fire users (so no HTTP data is even sent). If correctly configured, your AM will know for each connection that an fed4fire user is connected. This is called “authentication”.
- Your AM will accept user and slice “credentials” that is received from fed4fire users. It will know what fed4fire allows these users to do (typically, a slice credential will tell the AM a user can reserve resources). This is called “authorization”.

3.1.3.2 Firewall

You’ll need to make sure some ports are reachable:

- Your AM is listening on one or two ports, and these should be publicly reachable.

D2.4: Testbed requirements, developments and integrations

- If your testbed nodes are reachable directly, the necessary ssh port on each node should be publicly reachable. If you use a testbed specific ssh proxy/gateway, only that proxy should be publically reachable.
- Your AM should be ping-able from the fed4fire monitoring domain, so it should allow ICMP for at least that domain.
- Your testbed OML monitoring should be able to create outgoing connections to the fed4fire monitoring domain. (Allow outgoing TCP to flsmonitor.ilabt.iminds.be:3003)

Note: The fed4fire monitoring connections will come from:

- `2001:6a8:1d80:2031:225:90ff:fe1d:1c68`
- `193.191.148.194`

Only for ICMP you could choose to allow only these (but you can also allow public access). For AM and SSH access, the ports need to be publicly reachable anyway. Don't rely too much on these not changing, you probably need to allow the whole subnet.

3.1.4 Configuring the AM GetVersion call

Your AM needs to be configured to return the correct info in the GetVersion reply. This is not a critical step, and it can be skipped. But it is a good practice to get this right. It also makes the jFed scanner work better.

The `GetVersion` call is a call that the scanner uses to retrieve basic server info. It is defined in the AM specification. The results contains a lot of info, most of which will be automatically filled in correctly by the server software. Experience shows that the following fields have to be checked for correctness:

```
"urn": "urn:publicid:IDN+example.com+authority+am",
"geni_api_versions": {
  "2": "https://example.com/am/2",
  "3": "https://example.com/am/3"
}
```

You need to make sure that:

- The URN is correct. That includes the hostname part (`example.com` in the example above), and the name part `am` in the example above. The name should be `am`, or `cm`. The URN must always match the expected URN in the `component_manager_urn` attribute in the request RSpec.
- The `geni_api_versions` URLs must point to the correct server URL(s). This must be the URL at which the AM is publically available. Make sure that:

D2.4: Testbed requirements, developments and integrations

- The protocol is `https`
- The port is correct
- The path is correct
- The hostname is **not** `localhost` or `127.0.0.1`.
- Preferably, the hostname is not an IP address but a DNS hostname.
- Multiple URLs are allowed if the server supports multiple AM versions. But this is not required. Do make sure however that at least the version contacted is listed. That is, if you connect to `https://example.com/am/2` it is at least expected that `"2": "https://example.com/am/2"` is listed.

3.1.5 Using the jfed scanner tool

Note

This section assumes you are using linux or MAC. Instructions for windows are very similar.

The jFed scanner can be used to automatically detect basic AM server settings, and store them for use by the other jFed tools. This way, you can add use an experiment AM to your local jFed, and test it.

While the scanner can be a usefull tool, it fails in some scenarios (as it is hard to test all strange AM configurations). It is also not frequently tested, so it can break in some releases. Finally, it is not aimed at end users, and is thus not a user friendly tool. For these reasons, if you experience problems with it, don't hesitate to contact us.

You can find the current stable jFed Scanner GUI [in this archive](#).

After downloading the archive, and extract it to a directory. Then execute `scanner-gui.jar`.

Example commands:

```
$ wget -nv http://jfed.iminds.be/releases/develop/VERSION/jar/jfed_gui.tar.gz
2017-01-10 11:27:28 URL:http://jfed.iminds.be/releases/develop/VERSION/jar/jfed_gui.tar.gz
[25654644/25654644] -> "jfed_gui.tar.gz" [1]
$ tar xzf jfed_gui.tar.gz
$ cd jfed_gui
$ java -jar scanner-gui.jar
```

Once the scanner tool has started, log in using your fed4fire account.

In the next screen, fill in the `Aggregate Manager URL`, and click the `Scan` button.



D2.4: Testbed requirements, developments and integrations

Requirements for the URL:

- It needs to be HTTPS
- Do not forget the correct port
- Do not forget to add the full path to the AM endpoint
- Preferably, use a DNS hostname, not an IP address

When the scan is completed, you end up on the `Scan Overview` tab. Here, in the ideal case, you'll see 4 green `OK` statuses. If not, something went wrong. In the `Call Logs` and `Debug Logs` tab, you can find debug info which can help when something goes wrong.

In the `Scan Output` tab, all information the scanner gathered is shown. It is sometimes needed to uncheck the checkboxes next to any incorrect or unneeded information. But usually, you do not need to do anything here.

In the `Security` tab, you will find the X509 certificate of the server. You need to verify if this is indeed the certificate of your server, and if that is so, you need to check the checkbox next to `I trust all of the certificates above`.

Then, click on the `Show Results` button.

You will see a JSON snippet containing the info that jFed will use. You can edit the following fields:

- `id`: This should be a very short unique ID for the testbed. It should only contain letters, no spaces or special characters.
- `longName`: This is the "human readable" name of your testbed. This name may contain spaces and special characters.
- `server/name`: This is the "human readable" of the testbed server. This name may contain spaces and special characters. (It can be the same as the testbed longName)
- `defaultComponentManagerUrn`: This is the URN of your AM. Make sure it is correct.

If you click on `Add to local jFed`, the JSON snippet will be stored in `~/.jFed/extra_testbeds/<id>.json` (where `<id>` is the ID you chose for the testbed). All jFed tools will load files from that dir to add additional testbeds.

D2.4: Testbed requirements, developments and integrations

3.1.6 Local files for adding testbeds

Using local files, you can add your AM to jFed.

When the jFed scanner has successfully scanned a server, it shows a button “Add to local jFed”, which will write such a file.

These files are stored in the jFed config dir, in the *extra_testbeds* dir. On Linux this is in *~/jFed/extra_testbeds/*.

Below is an example of such a file with dummy values. You can also look at the global jFed configuration (see link in next section) for real examples.

Content of *~/jFed/extra_testbeds/mytestbed.json*:

```
{
  "id": "mytestbed",
  "longName": "My Brand New Testbed",
  "defaultComponentManagerUrn": "urn:publicid:IDN+mytestbed.org+authority+am",
  "allowLinks": false,
  "servers": [
    {
      "name": "My Brand New Testbed",
      "certificateChain": "-----BEGIN CERTIFICATE-----\nMII... SERVER CERTIFICATE HERE
... \n-----END CERTIFICATE-----\n",
      "urnTld": "mytestbed.org",
      "baseUrl": "https://am.mytestbed.org/",
      "flags": [
        "featureExecuteAndInstallService",
      ],
      "services": [
        {
          "api": "Geni.AM",
          "apiVersion": "3",
          "url": "https://am.mytestbed.org/",
          "urn": "urn:publicid:IDN+mytestbed.org+authority+am",
          "@type": "Service"
        }
      ],
      "defaultComponentManagerUrn": "urn:publicid:IDN+mytestbed.org+authority+am",
      "@type": "Server"
    }
  ],
  "@type": "Testbed"
}
```

3.1.7 Adding to jFed for all users

The jFed team can also add the same info about your AM, to the “global” jFed config. That enables the use of your AM for all jFed users. The global jFed configuration is available at <https://flsmonitor-api.fed4fire.eu/testbed?embed=true>

Contact us to add your AM.

D2.4: Testbed requirements, developments and integrations

You do not need to have a local working config, but that helps.

The minimal info you need to send us is:

- The URL at which your AM runs (the full URL, including hostname, port and path)
- The name you want to see for your testbed inside jFed and at our monitoring site
- An example request RSpec (which shows us the sliver_type(s) used and other info)
- Does your testbed support (or even require) specifying disk images in the request RSpec?
- Does your testbed support (or even require) assigning specific nodes (component_id attribute) in the request RSpec?
- Does your testbed support (or even require) specifying hardware types in the request RSpec?
- Are links between nodes of your testbed supported?
- Does your testbed support “stitched” links? (If you’re unsure what that is, your testbed doesn’t support them.)

Other info which is usefull to send us is:

- Which “icon” in the jFed experiment GUI is the best match for your testbed nodes (ex: physical node, VM, wireless, ...)?
- For security reasons, it is recommended to also send your server certificate (in PEM format). We can also retrieve the certificate ourself, but theoretically, that could be intercepted.
- The coordinates of the testbed (latitude, longitude and country)
- Info on the organisation running the testbed (full name, latitude, longitude, country, address, link to logo, link to website)
- Additional info about your testbed: link to documentation, link to testbed description and/or other general testbed information.
- If your testbed has a web interface (where users can login and control the testbed) in addition to the AM, you can send us that link as well.
- The email address(es) of the “primary” contact(s) of the testbed. If we need to contact someone about the testbed, we use these emails.
- The email address(es) of the “technical” contact(s) of the testbed. For technical questions about the testbed, these emails are used. These emails also are used for automatic mails about the tests, and in the future, testbed specific bugreports will also be sent here. Let us know wether or not you want the technical contact(s) of the testbed to receive automated emails when the testbed goes down.

D2.4: Testbed requirements, developments and integrations

- The email address(es) of the GDPR contact(s) of the testbed. Anything related to the GDPR we will send here.
- The URN(s) of the users that may have admin access to some of the data we store about your testbed. This is currently not used, but we plan on using this later to allow you to restart tests, enable/disable tests or edit testbed info. This is typically the URN of the users you use to login to jFed.
- Which software does your testbed AM use? Typical options are: emulab, gcf, openstack, ...

(A lot of email addresses are mentioned above, but typically, they are all just the same single email address.)

3.1.8 Adding your AM to one of the experiment GUI “icons”

The jFed Experimenter GUI has a graphical editor, where you can drag “icons” to the canvas and in this way easily configure an experiment. Each of these icons has their own list of relevant testbeds.

Note that you do not need these icons to test your AM using the jFed Experimenter GUI: after you added your testbed to jFed, you can directly edit the XML Rspec, and everything will work.

It is currently not possible to manually add a testbed to one of these jFed icons in your local jFed install. Adding testbeds to the “icons” can only be done by the jFed developers. When you contact us to add your testbed to jFed for all users, we will discuss which icon(s) it needs to be added too, and add it for you.

3.2 HANDLING TESTBED SPECIFIC TERMS & CONDITIONS (I.E. HANDLING GDPR)

3.2.1 jFed support for Testbed Specific Terms & Conditions (T&C)

jFed has extensive support for sites with Testbed Specific Terms & Conditions. These sites can integrate with jFed, but this is not required. T&C sites need to be registered in the jFed central config.

The basic requirements for such site are that they store if a user has consented to the T&C. So some form of DB is typically needed. When a user has not yet consented, the testbed can choose to disallow the user access. For this functionality, the testbed AM needs to access the

D2.4: Testbed requirements, developments and integrations

consent DB. Users need to identify themselves to T&C sites, the only logical method of doing is, is using the fed4fire method: https client certificates.

T&C sites can use jFed specific javascript, to:

- Detect if they are running inside jFed: `if (window.jfed)`
- Report to jFed if the user has consented (and for how long), or not: `jfed.decline();` and `jfed.approveWithDateISO8601("\2018-05-19T13:05:00+02:00\");` (and other date options)
- Request jFed to close the browser window: `jfed.close();`

jFed also works with sites that do not use any of the specific javascript. In this case, jFed will open the site in a window, and when it is closed will assume that the user has accepted the terms..

3.2.2 jFed T&C integration javascript details

When jFed loads the T&C site, it is useful to let jFed know if the user has already accepted the T&C before, or not. However, when the site javascript runs, jFed has not yet had time to inject its code into the site. To solve this, you can use this method:

```
function initJFed() {
  if (window.jfed && window.jfed.decline) {
    //Here, you would check if the user has accepted the terms and conditions, or not.
    //if the user has accepted, let jFed know
    //window.jfed.approve(); //uses default timeframe configured in the jFed central config

    //if the user hasn't accepted yet, let jFed know
    window.jfed.decline();
  }
}

//run this to automatically contact jFed when its javascript becomes available
if (window.jfed) {
  initJFed();
} else {
  //window.jfed is not (yet) available
  //trick to make browser call initJFed() when window.jfed becomes available.
  Object.defineProperty(window, 'jfed', {
    configurable: true,
    enumerable: true,
    writable: true,
    get: function() {
      return this._jfed;
    },
    set: function(val) {
      this._jfed = val;
      initJFed();
    }
  });
}
```

There are multiple methods to signal to jFed that the user has accepted the T&C:

D2.4: Testbed requirements, developments and integrations

```
//report approval of the T&C to jFed (without specifying end date of approval)
// (This will cause the default duration configured for this specific testbed in the central jFed
config to be used)
jfed.approve();

//report approval of the T&C to jFed, and specify a date in ms since epoch (javascript
Date.getTime() returns this)
jfed.approveWithDateInMillisecondsSinceEpoch(Date.now()+(24*3600*30*1000));

//report approval of the T&C to jFed, and specify the relative date in days from now
jfed.approveDaysFromNow(7);

//report approval of the T&C to jFed, and specify a data in ISO8601 format (RFC3339 is a subset
of ISO8601)
jfed.approveWithDateISO8601(\"2018-05-19T13:05:00+02:00\");
```

You can also report that the T&C are not accepted:

```
//report decline of the T&C to jFed
jfed.decline();
```

You can also request jFed to close the browser window:

```
//close the jFed T&C window from within javascript
jfed.close();
```

When something goes wrong, and no javascript is used, jFed reverts to the fallback. This means that when you close the window, jFed will assume that the user has accepted the T&C. Because of this, it's not a bad idea to initially call `jfed.decline();`.

3.2.3 Example: standalone T&C site

A standalone T&C site must:

- Use https with client authentication enabled. Since the site has no purpose without a valid user, it makes sense to make SSL user authentication mandatory (this is a setting in the server config).
- The site must be configured to allow users of the fed4fire authority access. (This is done by trusting the root certificate for wall2 in the server config)
- The site must extract the user URN from the certificate used to authenticate.
- When a user declines or approves the T&C, this decision must be stored in a DB at the server.
- Optionally, the site can use the jFed T&C integration javascript mentioned above.

Alternatively, you could use a regular http/https site, and using a separate service that takes care of the 4 first points above.

D2.4: Testbed requirements, developments and integrations

The technology chosen for the site backend (php, python, java, ...) and the specific DB used are up to the testbed to decide.

(We might add an example of such a standalone site in the future)

You can use the example in the next section as a partial example for a standalone site: Take a look at `terms_conditions.html` and `terms_conditions.js` in https://github.com/wvdemeer/docker-am/tree/gdpr/gcf_docker_plugin/terms_conditions

3.2.4 Example: AM with T&C site built in

It's also possible to integrate the site directly in the AM code. One of the advantages is that the site can then easily run at the same domain and port of the AM. The SSL authentication is also already configured as the AM requires the same.

To demo this, we've modified a GCF based site, the docker AM. Looking at these changes, you should be able to modify any gcf based site in the same way. The main files for this add-on are here: https://github.com/wvdemeer/docker-am/tree/gdpr/gcf_docker_plugin/terms_conditions The exact changes that were needed can be seen here: <https://github.com/open-multinet/docker-am/compare/master...wvdemeer:gdpr>

3.2.5 Add T&C site info to central jFed config

Send us the following info:

- What is the URL of the T&C site for your testbed?
- Do these T&C need to be approved per user? Or per project or even slice? (we recommend per user if possible)
- For per user approval, how long does the approval remain valid (i.e. how long before the user needs to re-approve)? (in days)

We will add this info to the central jFed config, so that jFed will send users to the T&C site when they start an experiment.

In case you update the conditions on the T&C site, let us know. We can change the version in the config, which will cause jFed to require all users to visit the T&C site again.

Technical details: We add the following to the testbed config:



D2.4: Testbed requirements, developments and integrations

```
"gdprInfo": {  
  "grpUrl": "https://example.com/terms_conditions/index.html",  
  "acceptPeriodInDays": 365,  
  "acceptSubject": "USER",  
  "version": "1",  
  "@type": "GDPRInfo"  
}
```

Here's an example: <https://flsmonitor-api.fed4fire.eu/testbed/iminds-docker>

3.2.6 Make the AM reject users that did not approve the Terms & Conditions

It should be sufficient to only add a check to the `Allocate` call.

In case the user is not known to have approved the T&C of the testbed, we strongly recommend using this error message:

```
{  
  "output": "[T&C-APPROVAL-MISSING] Approval of the Terms & Conditions is required in order to use this testbed. Please visit https://example.com/termsandconditions",  
  "code": { "geni_code": 7 }  
}
```

Notes:

- The above is a JSON representation of the XML-RPC reply, the actual reply is off course in XML. (But JSON is much easier to read.)
- `geni_code` 7 is used. This code is the general code for “REFUSED” “Operation Refused”
- Be sure to include “[GDPR-CONSENT-MISSING]” for automatic detection of this error by jFed. jFed can then show the appropriate error dialog.
- It is also important to include a human readable message, and a link to the actual terms and conditions of the testbed. This is *essential* for users that do not use jFed.
- The `value` field is omitted above, but it may be included. It is optional because this is an error reply.
- An AM may optionally include an `am_type` and `am_code`. You are free to use what you want for these.

3.3 DEBUGGING YOUR AM

jFed offers some tools that can help a lot when debugging your AM.

Once your AM is running, we can add it to our fedmon monitor.

D2.4: Testbed requirements, developments and integrations

3.3.1 jFed Probe GUI

The jFed Probe GUI is a tool for manually calling servers. Very low level calls can be made, and low level replies are visible. This tool requires knowledge of the communication APIs, and the tool itself is not a very user friendly GUI. It can be very valuable for testbed developers that wish to debug their server.

While you can execute all calls by filling in all details manually, for convenience, the probe GUI can automatically execute user and slice related calls with minimal user input:

- Start the Probe GUI and login
- Expand “API Wrappers”
- Expand “Automatic User and Slice API Wrapper”
- Select “getLocalUserCredentials”
- Click the big “Call” Button. After a while, you’ll see that a call was done, and it’s results will appear. This call contains the user credential. (the jFed probe has stored it and will automatically fill it in for next calls.)

Now that you have a user credential, you can use it to create a slice:

- Go to the “Automatic User and Slice API Wrapper” methods, and select “createSlice”
- In the form to the left, fill in “sliceName”, choose an expire time (2 hours by default)
- Click to check the checkbox before “subAuthName”, and fill in the EXACT name of your project as “subAuthName”. (“sub authority” and “project” are the same thing)
- Click the big “Call” Button. After a while, you’ll see that one or more call are done, and their results will appear. This last call contains the slice credential. (the jFed probe has stored it and will automatically fill it in for next calls.)

If you go to any AM call now, you will see that the slice name and credential are filled in by default.

3.3.2 jFed automated tester

The Automated Testing GUI: This tool is used to run tests scenarios on servers. This is only useful for server developers, and it is not a very user friendly GUI. It includes simple tests, such as verifying if the `GetVersion` reply of an AM server is correct, and more complex tests, such as verifying if an AM can correctly provision a node, and if that node can be logged in to using SSH.

D2.4: Testbed requirements, developments and integrations

First start the automated tester. You can find the latest jFed releases [here](#). Download the “jFed GUI (archive)” from that page.

After downloading the archive, extract it to a directory. Then execute `automated-testing-gui.jar`.

Example commands:

```
$ wget -nv http://jfed.iminds.be/releases/develop/VERSION/jar/jfed_gui.tar.gz
2017-01-10 11:27:28 URL:http://jfed.iminds.be/releases/develop/VERSION/jar/jfed_gui.tar.gz
[25654644/25654644] -> "jfed_gui.tar.gz" [1]
$ tar xzf jfed_gui.tar.gz
$ cd jfed_gui
$ java -jar automated-testing-gui.jar
```

You can also use the installer, and install the automated tester that way.

When you start the automated tester, you need to log in with your user account.

On the next page, you need to select the testbed you want to test, and the test you want to run. In the example below, we are testing the “Virtual Wall 1” testbed. The chosen test is “TestNodeLogin”. This test will create a new slice at the logged in user’s authority. Then it will contact the tested testbed, it will execute `Allocate` and `Provision` calls, wait for the nodes to become ready, and then try to log in to the nodes. Finally, no matter whether the tests fails or succeeds, a `Delete` call is made to free any resource reservations.

D2.4: Testbed requirements, developments and integrations

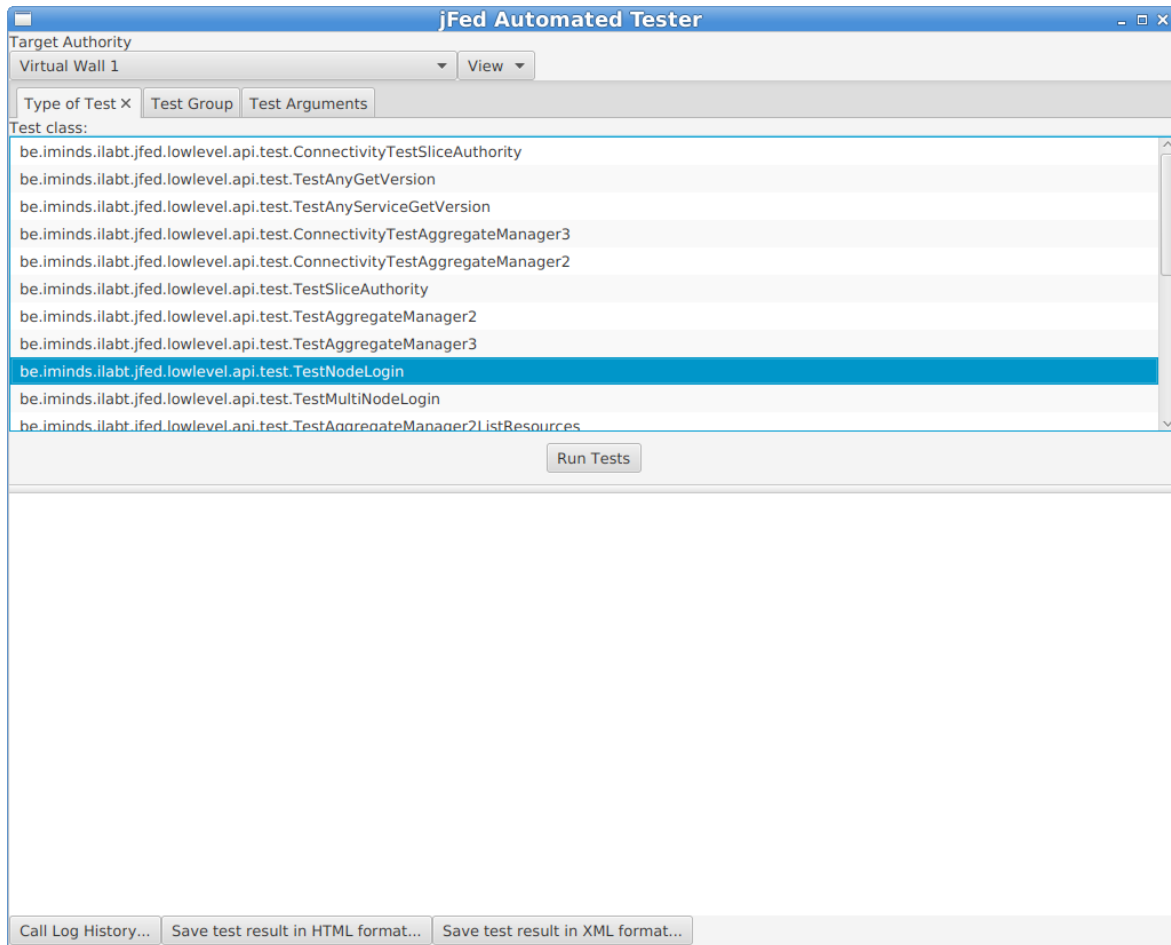


Figure 5: jFed automated tester: overview of tests

When the testbed (“Target Authority” at the top of the window) and the test are selected (in the “Type of Test” tab), you can optionally go to the “Test Arguments” tab. This tab lists all possible options for the tests. For a lot of tests, these are a lot of options, and they are undocumented. For the “TestNodeLogin”, you typically do not need to change anything. Some interesting configuration options are:

- `fixed_rspec`: You can specify a custom RSpec here, if the automatically generated RSpec is not ok.
- `fixed_rspec_url`: You can specify the URL to a custom RSpec here, instead of entering the RSpec directly as with the `fixed_rspec` option.

D2.4: Testbed requirements, developments and integrations

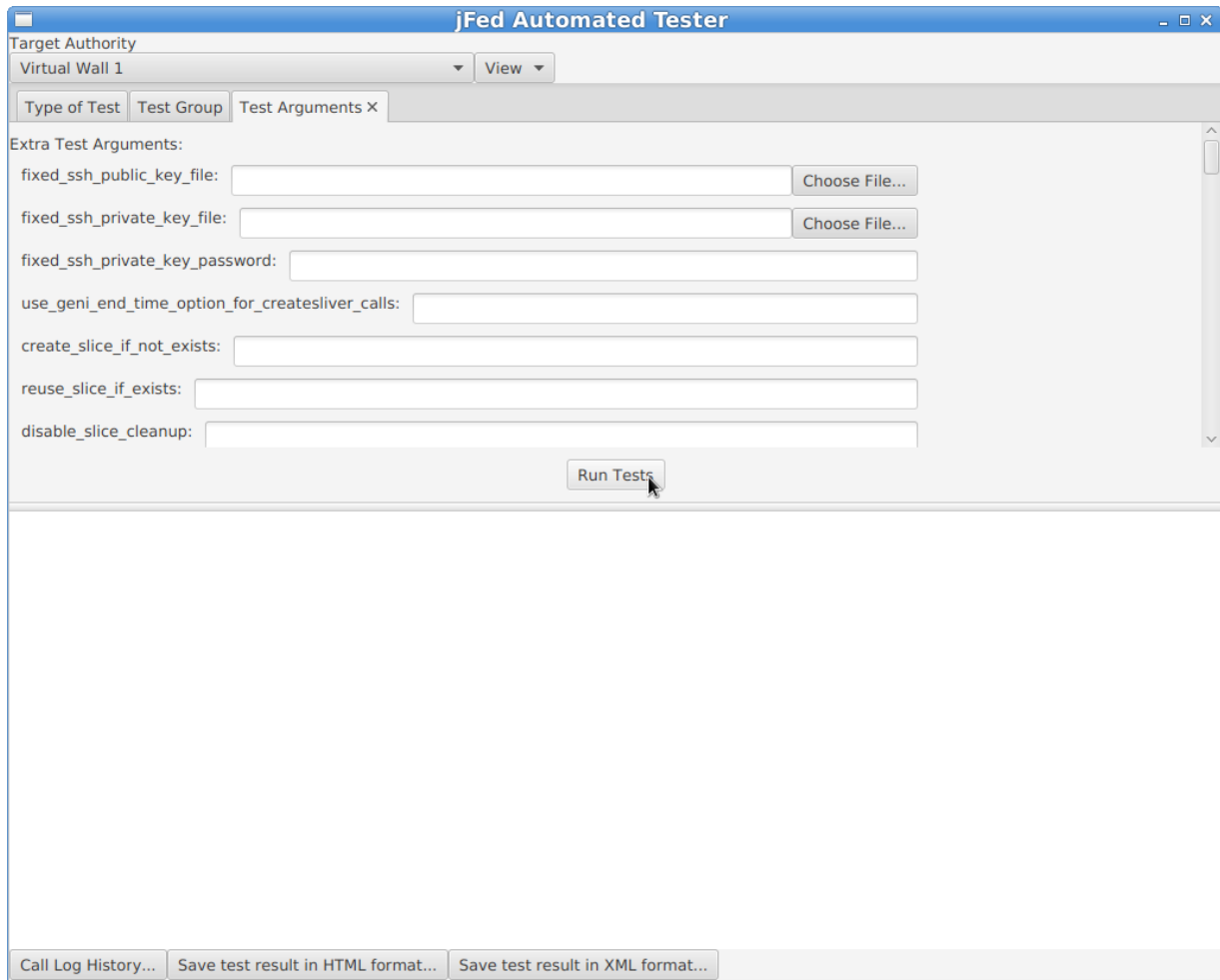
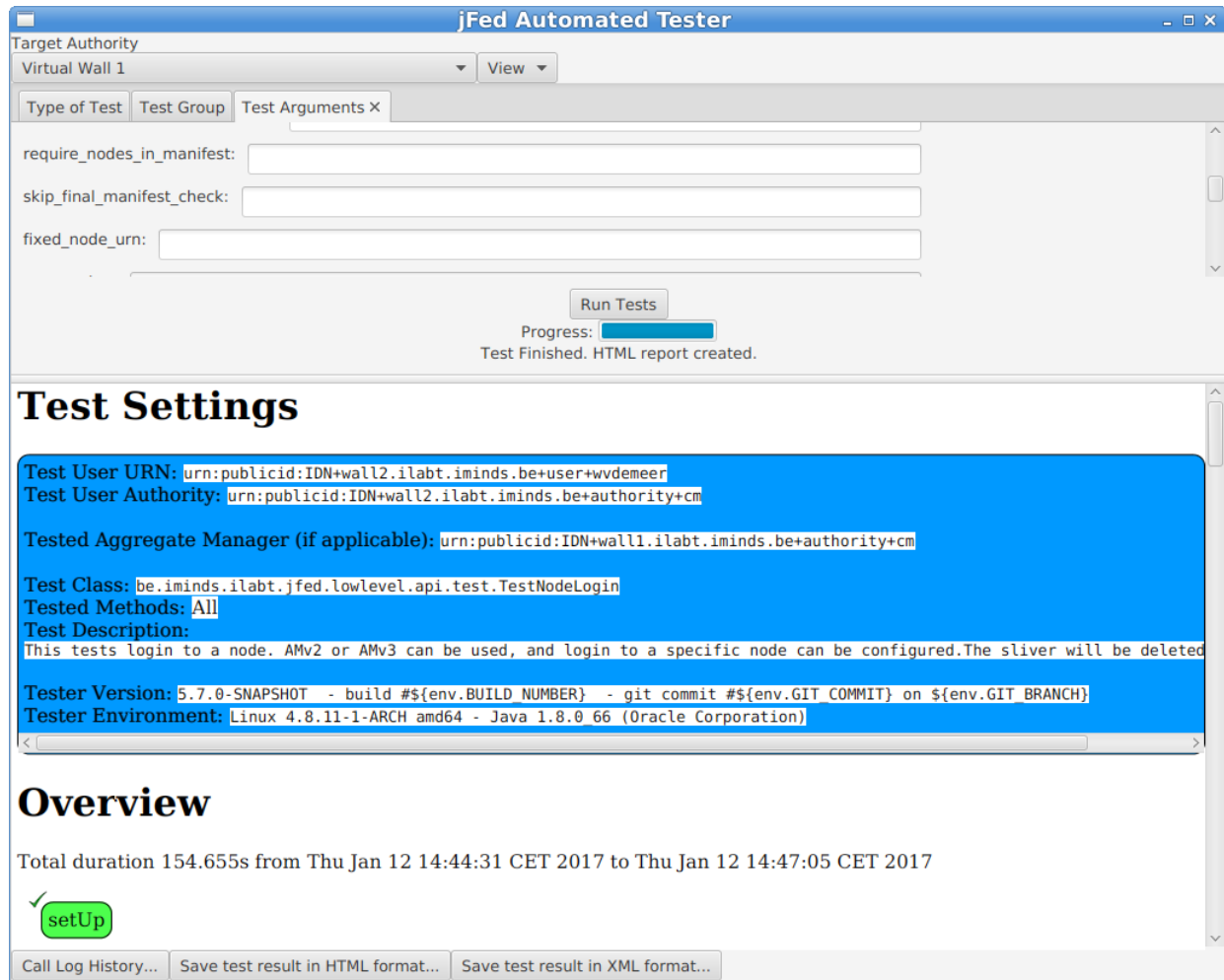


Figure 6: jFed automated tester: run a specific test

Next, click on the “Run Tests” button. You will see a progress bar while the test runs. A “TestNodeLogin” can take a few minutes, to more than 20 minutes, depending on how fast the AM is, and whether the nodes come online or not.

A test report will appear in the bottom half of the window when the test is finished. You can look into the details of each step. Note that you can click the “Hide/Show” buttons to show additional call details.

D2.4: Testbed requirements, developments and integrations



The screenshot displays the 'jFed Automated Tester' application window. At the top, it shows the 'Target Authority' as 'Virtual Wall 1'. Below this, there are input fields for 'Type of Test', 'Test Group', and 'Test Arguments'. The main configuration area includes fields for 'require_nodes_in_manifest', 'skip_final_manifest_check', and 'fixed_node_urn'. A 'Run Tests' button is present, along with a progress indicator and a status message: 'Test Finished. HTML report created.'

Test Settings

Test User URN: `urn:publicid:IDN+wall2.ilabt.iminds.be+user+wvdemeer`
 Test User Authority: `urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm`
 Tested Aggregate Manager (if applicable): `urn:publicid:IDN+wall1.ilabt.iminds.be+authority+cm`
 Test Class: `be.iminds.ilabt.jfed.lowlevel.api.test.TestNodeLogin`
 Tested Methods: `All`
 Test Description:
 This tests login to a node. AMv2 or AMv3 can be used, and login to a specific node can be configured. The sliver will be deleted
 Tester Version: `5.7.0-SNAPSHOT - build #${env.BUILD NUMBER} - git commit #${env.GIT COMMIT} on ${env.GIT BRANCH}`
 Tester Environment: `Linux 4.8.11-1-ARCH amd64 - Java 1.8.0_66 (Oracle Corporation)`

Overview

Total duration 154.655s from Thu Jan 12 14:44:31 CET 2017 to Thu Jan 12 14:47:05 CET 2017

✓ **setUp**

Call Log History... Save test result in HTML format... Save test result in XML format...

Figure 7: jFed automated tester: test results

D2.4: Testbed requirements, developments and integrations

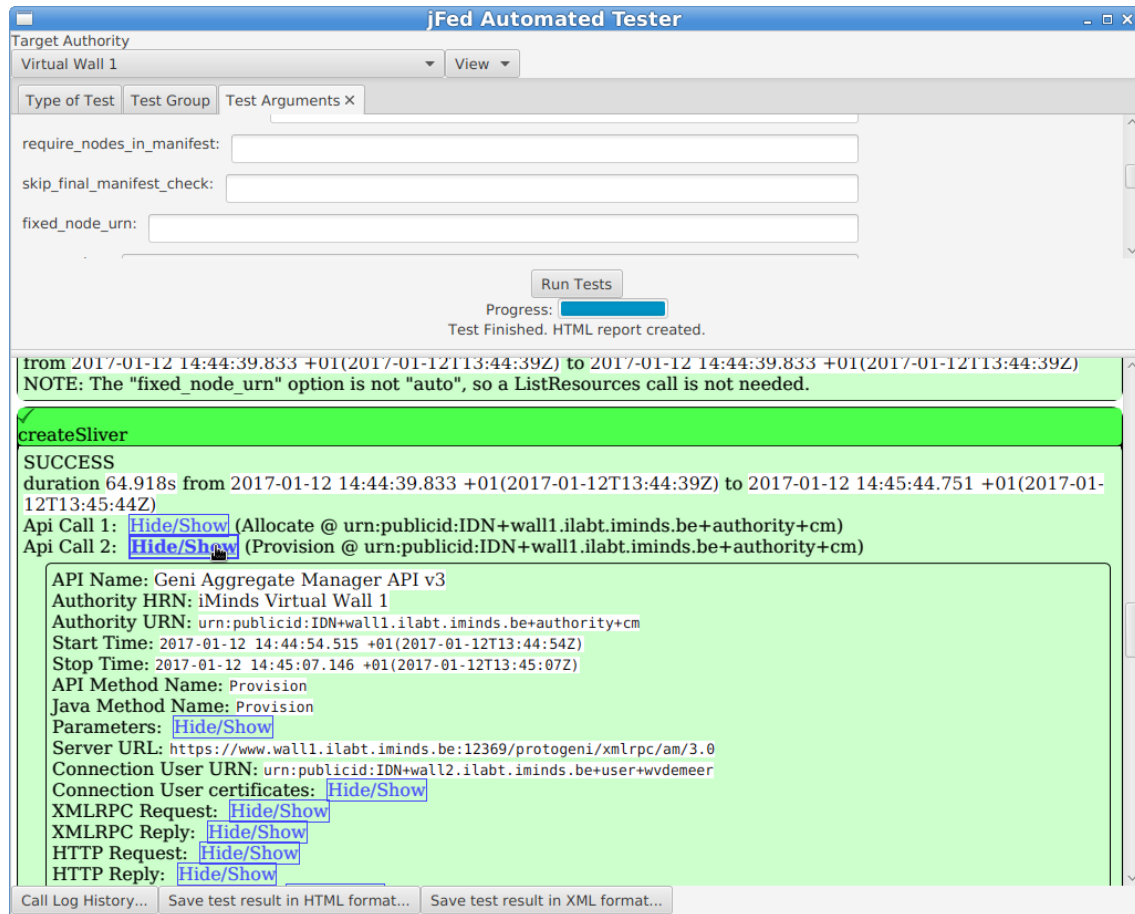


Figure 8: jFed automated tester: test result details

At the bottom of the window, there are some buttons of interest:

- The “Call Log History” button allows you to see the call details in the same way as the jFed probe shows them. This is sometimes easier than the HTML view above.
- You can use “Save test result in HTML format” to store the result to disk, which can be handy to send them to others.

Todo

this section needs to be improved

3.3.3 Permanent monitoring: Fedmon

The jFed team can add your AM to the “federation monitor”. Ask us about this.

D2.4: Testbed requirements, developments and integrations

As an example, here is the monitoring overview for all monitored imec/iMinds testbeds: <https://flsmonitor.fed4fire.eu/fls.html?testbedcategory=iMinds&showlogintests>

3.4 RSPEC DETAILS FOR AM DEVELOPERS

3.4.1 General

There are 3 types of RSpecs, request, manifest and advertisement RSpecs.

All RSpecs must be valid XML, and for each type a slightly different schema is used. Not that “valid XML” is defined as:

- It is well formed XML (closing and ending tags match, etc.). This means the XML can be parsed without error by an XML parser.
- The XML follows the specified schema. This means that all rules specified in the schema are followed, and the rules concerning namespace are followed.

The XSD files that define the RSpec schema for “geni version 3 RSpecs” can be found at the URI used to identify the namespace: <http://www.geni.net/resources/rspec/3> .. note:: There are 3 root xsd files at that location: one for the request RSpec schema, one for the manifest RSpec schema and one for the advertisement RSpec schema. .. note:: The URI used to define an XML namespace does NOT need to host any XSD files. This is often done as it is a convenient location, but the URI can also be **only an identifier**. Research XML namespaces if this is confusing.

The Rspec schema’s allow a lot of freedom: they allow adding custom elements at different places in the RSpec. However, this is only allowed if the additions are in a *different* namespace than the “geni v3 rspec” namespace. This means you need to define a custom namespace for any RSpec extension you make!

Note that there are certain rules on how an AM should handle unknown RSpec extensions in RSpec requests. See the section on requests RSpecs below.

3.4.2 Choosing your component manager urn

In the RSpecs, resources belong to a certain “component manager”. This is the authority responsible for these resources, and thus it is typically your AM itself. This binding is represented in each RSpec, using the `component_manager_id` attribute, which has an URN as value.

D2.4: Testbed requirements, developments and integrations

More info on the exact format of the URNs used can be found at <http://groups.geni.net/geni/wiki/GeniApIdentifiers>. Put simply, the URN using for the `component_manager_id` attribute is either of the form `urn:publicid:IDN+TOPLEVELAUTHORITY:SUBAUTHORITY+authority+cm` or of the form `urn:publicid:IDN+TOPLEVELAUTHORITY:SUBAUTHORITY+authority+am`. (both ending, `cm` and `am` are in use at different testbeds. We recommend `am`, but it doesn't really matter.)

It is best NOT to use `:SUBAUTHORITY` at all, unless you really know you need it. So the typical urn is: `urn:publicid:IDN+TOPLEVELAUTHORITY+authority+cm`

The `TOPLEVELAUTHORITY` part of the URN is preferably the top level DNS name of your AM. In some cases, a nickname is also ok. Both `localhost` or `127.0.0.1` are **not** allowed. IP addresses should be avoided.

This URN is used in the following places:

- * In the ```component_manager_id``` attribute of node elements in the advertisement RSpec.
- * In the ```component_manager_id``` attribute of node and link elements in the request and manifest RSpec.
- * Optionally: in the ```GetVersion``` reply, in the ```value``` ```urn``` field.

A few examples of real component manager URNs:

- * `urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm`
- * `urn:publicid:IDN+utah.cloudlab.us+authority+cm`
- * `urn:publicid:IDN+fuseco.fokus.fraunhofer.de+authority+cm`
- * `urn:publicid:IDN+instageni.cs.princeton.edu+authority+cm`
- * `urn:publicid:IDN+exogeni.net+authority+am`

D2.4: Testbed requirements, developments and integrations

* `urn:publicid:IDN+exogeni.net:bbnvmsite+authority+am`

3.4.3 RSpec basics: `sliver_type` and `exclusive`

The basic rspec format allows expressing raw bare metal hardware nodes as well as VMs/container nodes. The `sliver_type` element is used, as well as the `exclusive` attribute. Support for both is mandatory in each of the RSpec types.

The `sliver_type` element is used to specify which sort of node is requested or available. The 2 common cases are bare metal, and virtual machines. You are free to pick a `sliver_type` name that makes sense for your testbed.

For bare metal nodes, emulab uses `raw-pc` as `sliver_type` name.

Some examples of `sliver_type` names for virtual machines:

- `default-vm` is defined within `geni` as a “convenience” `sliver_type`. Each AM that supports VMs is supposed to replace this by the default VM `sliver_type` for that AM. This makes it easier to write portable RSpecs that can be executed on multiple testbeds, without changing the `sliver_type` for each testbed. It is useful to support this feature if you AM supports VMs, but it is not mandatory.
- `emulab-xen` and `emulab-openvz` are the VM `sliver_type` used by emulab (and thus instageni).
- `xo.tiny`, `xo.small`, `xo.medium`, `xo.large`, `xo.xlarge` are the types used by exogeni. Note that they map to the “size” of the VM.
- `docker-container` is used by the `ilab.t` docker AM.

The `exclusive` attribute is always true for bare metal hardware, since you always get exclusive access to it. Example:

```
<node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm">
  <sliver_type name="raw-pc"/>
</node>
```

For VMs, the user can specify the node `exclusive` attribute of the node to be either false or true. `exclusive="false"` means that other users can get a VM hosted on the same physical machine. `exclusive="true"` means that other users can *not* get a VM hosted at the same physical machine. For most cases with VMs or containers, `exclusive="false"` is always used.

D2.4: Testbed requirements, developments and integrations

Example:

```
<node                                client_id="node1"                                exclusive="false"
component_manager_id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm">
  <sliver_type name="default-vm"/>
</node>
```

A request with `exclusive="false"` for a node where that is not supported by the testbed, should not result in a failure. The testbed should just change it to true in the manifest. However, a request for `exclusive="true"` for a node for which the testbed does not support it, should result in an error.

In the advertisement RSpec, the AM needs to list each allowed `sliver_type` for each node. The `exclusive` attribute takes a different meaning in the advertisement. `exclusive="false"` means that a request may never request exclusive access to a node. `exclusive="true"` means a request may request exclusive access.

3.4.4 RSpec basics: Disk Images

Disk images are added as part of a sliver type, because they can differ depending on sliver type.

In an advertisement RSpec, the AM should list all possible disk images within each sliver element of each node. Note that this means there will be a lot of repetition! (That is not a nice feature of the “geni version 3 rspec” schema, but there is no way around it.)

It is not required that an AM supports this functionality. In case an AM does not support it, it should fail with the correct error when a `disk_image` is specified in a request RSpec.

Example in a request:

```
<node                                client_id="node0"                                exclusive="true"
component_manager_id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm">
  <sliver_type name="raw-pc">
    <disk_image name="urn:publicid:IDN+wall2.ilabt.iminds.be+image+emulab-ops:CENTOS65-64-STD-BIG2"/>
  </sliver_type>
</node>
```

In practice, the `name` field of a disk image contains an URN. jFed currently can only handle disk images containing an URN. The `authority` part of the URN should refer to the testbed, the `type` part of the URN is always “image”.

D2.4: Testbed requirements, developments and integrations

Note that the geni version 3 rspec schema allows the following optional attributes to be specified in the advertisement RSpec (they are allowed in the request RSpec, but don't make much sense there):

- `os`: The name of the OS
- `version`: The version of the disk image, or OS (not really specified anywhere which one).
- `description`: A textual description of the disk image. You can put anything that is helpful for users here.
- `url`: TODO: emulab supports disk images from other testbeds. This is not yet explained here.

To have jFed support disk images for a testbed, the jFed central config needs to be updated. Contact the jFed developers for this.

3.4.5 RSpec basics: Specific nodes

Optionally, an AM can allow (or require) requests that demand a specific piece of hardware. This is done using the `component_id` attribute. If this attribute is not specified, the testbed has to either fail the request (informing the user that `component_id` is mandatory), or pick a suitable piece of hardware automatically.

Note that the advertisement RSpec can contain an additional `component_name` attribute, which has a nickname for a node. This is typically the same as the last part of the `component_id`. This `component_name` attribute should NOT be used in a request RSpec. An AM should not require it, nor use it to assign specific nodes, only `component_id` should be used for that.

Example in a request:

```
<node client_id="node0" exclusive="true"
  component_manager_id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm"
  component_id="urn:publicid:IDN+wall2.ilabt.iminds.be+node+n082-01">
  <sliver_type name="raw-pc"/>
</node>
```

3.4.6 RSpec basics: Hardware type

Optionally, an AM can allow (or require) requests that demand a specific **type** of hardware, but that do not specify the specific piece of hardware. This is done using the `hardware_type` element. If this element is specified and no `component_id` is

D2.4: Testbed requirements, developments and integrations

specified, the testbed has to either fail the request (informing the user that `hardware_type` is not supported), or pick a suitable piece of hardware, matching the type, automatically.

Note

It is important to understand the difference between `hardware_type` and `sliver_type`.

Example in a request:

```
<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm">
  <sliver_type name="raw-pc"/>
  <hardware_type name="gpunode"/>
</node>
```

In an advertisement RSpec, the `hardware_type` should be specified, if this feature is supported in the request RSpec. Note that it is allowed to specify multiple `hardware_type` elements in an advertisement RSpec. It is ok to do so if that makes sense for your AM. But if possible, it's nice to keep it simple and specify only 1 `hardware_type` per node.

3.4.7 Advertisement Examples: Bare metal access

Scenario: You want to give users “bare metal access” to nodes. This means they get full access, not access to a VM or container. The user is the only one getting access to the machine, typically as “root” or as a user with sudo privileges.

Advertisement RSpec example:

```
<node component_id="urn:publicid:IDN+example.com+node+pi1"
component_manager_id="urn:publicid:IDN+example.com+authority+am" component_name="pi1"
exclusive="true">
  <sliver_type name="raw-pc">
    <disk_image name="urn:publicid:IDN+example.com+image+raspbian"/>
    <disk_image name="urn:publicid:IDN+example.com+image+arch"/>
  </sliver_type>
  <hardware_type name="pc-raspberry-pi"/>
  <available now="true"/>
  <location country="NU" latitude="0.0" longitude="0.0"/>
</node>
```

Things to note:

- `exclusive` is true, because users get full access to the node.
- `sliver_type` is `raw-pc`. This is the typical sliver type used to represent bare metal access to “PC like” hardware.

D2.4: Testbed requirements, developments and integrations

- `hardware_type` is `pc-raspberry-pi`. The hardware type name should be something that identifies the type of hardware to a user. In this case, the hardware is a Raspberry Pi computer. Another example are the `pcgen01`, `pcgen02` and `pcgen03` types, which are used on the imec virtual wall, and means “A PC of generation 1, 2 or 3”, users can then look up in the testbed documentation what the full specifications of each “generation” is.
- `available now="true"` means that this hardware is currently available.
- `location` is used to specify the location of the node. `country` is the country code, in this case, and invalid code, referring to “null island”. Of course you should use the real coordinate of your testbed node here. It is OK to use the same coordinate for all testbed nodes.

3.4.8 Advertisement Examples: Simple VM's

This example shows how you can offer a single type of VM.

Advertisement RSpec example:

```
<node xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
component_id="urn:publicid:IDN+example.com+node+vmhost1"
component_manager_id="urn:publicid:IDN+example.com+authority+am" component_name="vmhost1"
exclusive="false" >
  <sliver_type name="xen-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu14"/>
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
  <hardware_type name="pc-vmhost">
    <emulab:node_type type_slots="20"/>
  </hardware_type>
  <available now="true"/>
  <location country="NU" latitude="0.0" longitude="0.0"/>
</node>
```

Matching Request RSpec example:

```
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1">
  <node client_id="vm-one" component_manager_id="urn:publicid:IDN+example.com+authority+am"
exclusive="false">
    <sliver_type name="xen-vm">
      <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="100.0" y="100.0"/>
  </node>
  <node client_id="vm-two" component_manager_id="urn:publicid:IDN+example.com+authority+am"
exclusive="false">
    <sliver_type name="xen-vm">
      <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
    </sliver_type>
    <jfed:location x="100.0" y="100.0"/>
  </node>
</rspec>
```


D2.4: Testbed requirements, developments and integrations

3.4.9 Advertisement Examples: Complex VM's: multiple sizes

This example shows how you can offer different “sizes” of VM, where each size has a different number of CPU cores, memory, etc.

Each size will typically take a different number of `type_slots`, for example, a “tiny-vm” will take 1 type slot, and a “large-vm” will take 5 type slots.

Advertisement RSpec example:

```
<node
    xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
    component_id="urn:publicid:IDN+example.com+node+vmhost1"
    component_manager_id="urn:publicid:IDN+example.com+authority+am"
    component_name="vmhost1"
    exclusive="false" >
  <sliver_type name="vm-tiny">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu14"/>
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
  <sliver_type name="vm-medium">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu14"/>
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
  <sliver_type name="vm-big">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu14"/>
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
  <hardware_type name="pc-vmhost">
    <emulab:node_type type_slots="30"/>
  </hardware_type>
  <available now="true"/>
  <location country="NU" latitude="0.0" longitude="0.0"/>
</node>
```

Matching Request RSpec example:

```
<rspec
    xmlns="http://www.geni.net/resources/rspec/3"
    type="request"
    xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1">
  <node client_id="vm-one" component_manager_id="urn:publicid:IDN+example.com+authority+am"
    exclusive="false">
    <sliver_type name="vm-tiny">
      <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="100.0" y="100.0"/>
  </node>
  <node client_id="vm-two" component_manager_id="urn:publicid:IDN+example.com+authority+am"
    exclusive="false">
    <sliver_type name="vm-tiny">
      <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
    </sliver_type>
    <jfed:location x="100.0" y="100.0"/>
  </node>
</rspec>
```

3.4.10 Advertisement Examples: Specialised hardware connected to single VM box

```
<node
    component_id="urn:publicid:IDN+example.com+node+usrp2"
    component_manager_id="urn:publicid:IDN+example.com+authority+am"
    component_name="usrp2"
    exclusive="true">
  <sliver_type name="usrp-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+plain"/>
  </sliver_type>
</node>
```

D2.4: Testbed requirements, developments and integrations

```

    <disk_image name="urn:publicid:IDN+example.com+image+gnuradio"/>
  </sliver_type>
  <hardware_type name="pc-usrp"/>
  <available now="true"/>
  <location country="NU" latitude="0.0" longitude="0.0"/>
</node>

```

3.4.11 Advertisement Examples: Combining Bare metal and VMs on a single node

Hint: look at emulab advertisement RSpecs.

Note that not all possible scenarios can be expressed in this format. Also, for some scenarios that can be expressed, not all info is specified. RSpec has grown historically, and does not offer every flexibility.

This is a complex case. It's good to note that each `sliver_type` works with one or more `hardware_types`, and will not work with certain other `hardware_types`. This info is not specified anywhere.

In the example below, the `hardware_type` "pc-gen1" and "pc-gen2" have a `sliver_type` "raw-pc", and the `hardware_type` "pc-vmhost" has `sliver_type` "small-xen-vm" and "big-xen-vm" (these match resources allocated to the VM, such as cores and memory. As an example, "small-xen-vm" takes 2 type slots, and "big-xen-vm" takes 5 type slots).

Example advertisement:

```

<node
  xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
  component_id="urn:publicid:IDN+example.com+node+nodeA"
  component_manager_id="urn:publicid:IDN+example.com+authority+am"
  exclusive="true" >
  <sliver_type name="small-xen-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu14"/>
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
  <sliver_type name="big-xen-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu14"/>
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
  <sliver_type name="raw-pc">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
    <disk_image name="urn:publicid:IDN+example.com+image+arch"/>
  </sliver_type>
  <hardware_type name="pc-vmhost">
    <emulab:node_type type_slots="20"/>
  </hardware_type>
  <hardware_type name="pc-gen2">
    <emulab:node_type type_slots="1"/>
  </hardware_type>
  <available now="true"/>
  <location country="NU" latitude="0.0" longitude="0.0"/>
</node>

```

D2.4: Testbed requirements, developments and integrations

Example request for any bare metal node of type “pc-gen2” (there could be bare metal nodes of type “pc-gen1” offering `sliver_type` “raw-pc” as well):

```
<node client_id="gen2nodeA" component_manager_id="urn:publicid:IDN+example.com+authority+am"
exclusive="true">
  <sliver_type name="raw-pc">
    <disk_image name="urn:publicid:IDN+example.com+image+arch"/>
  </sliver_type>
  <hardware_type name="pc-gen2"/>
</node>
```

Example request for bare metal access to “nodeA”:

```
<node client_id="nodeA" component_id="urn:publicid:IDN+example.com+node+nodeA"
component_manager_id="urn:publicid:IDN+example.com+authority+am" exclusive="true">
  <sliver_type name="raw-pc">
    <disk_image name="urn:publicid:IDN+example.com+image+arch"/>
  </sliver_type>
</node>
```

Example request for a “big” VM on any VM node:

```
<node client_id="vm1" component_manager_id="urn:publicid:IDN+example.com+authority+am"
exclusive="false">
  <sliver_type name="big-xen-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
</node>
```

Example request for 2 VM nodes on “nodeA”, with exclusive hardware access:

```
<node client_id="small-vm-a" component_id="urn:publicid:IDN+example.com+node+nodeA"
component_manager_id="urn:publicid:IDN+example.com+authority+am" exclusive="true">
  <sliver_type name="small-xen-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
</node>
<node client_id="big-vm-b" component_id="urn:publicid:IDN+example.com+node+nodeA"
component_manager_id="urn:publicid:IDN+example.com+authority+am" exclusive="true">
  <sliver_type name="big-xen-vm">
    <disk_image name="urn:publicid:IDN+example.com+image+ubuntu16"/>
  </sliver_type>
</node>
```

3.4.12 Request RSpec

This section describes which request RSpec features you can use for your AM. jFed will automatically offer a lot of functionality when this is done correctly.

Some things to take into account in request RSpecs:

- Each node will have exactly one `sliver_type` in a request.

D2.4: Testbed requirements, developments and integrations

- Each `sliver_type` will have zero or one `disk_image` elements. If your testbed requires `disk_image` or does not support it, it should handle bad requests RSpecs with the correct error.
- The `exclusive` element is specified for each node in the request. Your testbed should check if the specified value (in combination with the `sliver_type`) is supported. and return the correct error if not.
- The request RSpec might contain links that have a `component_manager` element that matches your AM. If your AM does not support links, it should return the correct error.

Some information will be in a request RSpec, that needs to be ignored and copied to the manifest RSpec unaltered. This is important to do correctly.

- A request RSpec can contain nodes that have a `component_manager_id` set to a different AM. You need to ignore these nodes, and copy them to the manifest RSpec unaltered.
- A request RSpec can contain links that do not have a `component_manager` matching your AM (links have multiple `component_manager_id` elements!). You need to ignore these links, and copy them to the manifest RSpec unaltered.
- A request RSpec can contain XML extensions in nodes, links, services, or directly in the `rspec` element. Some of these your AM will not know. It has to ignore these, and preferably also pass them unaltered to the manifest RSpec.

3.5 STITCHING DETAILS FOR AM DEVELOPERS

3.5.1 General

“Stitching” and “Dedicated Ext. Network Connection” are both methods to connect testbeds using dedicated physical links. Thus a fixed delay and bandwidth is available between nodes.

An alternative to both is to connect nodes over the internet, but this only offers “best-effort” delay and bandwidth (this typically uses “virtual links”, created gre-tunnels or other methods).

3.5.2 “Real” Stitching: automated multiple hops and VLAN translation

Stitching is used to create links between different AMs and testbeds. The user just draws a link between 2 nodes of a different testbed in the jfed experimenter GUI. Behind the scenes, VLAN connections are setup over dedicated link(s) between the different testbeds. Possibly, the

D2.4: Testbed requirements, developments and integrations

testbeds are not directly connected, and a few hops are needed to make the connection. On each of these hops, VLAN translation may be required to change to the VLAN used in the next link. Each involved AM must keep track of the used VLANs on each external link.

The user does not need to know any of these details, but the client software needs to send VLAN info to each VLAN, and work as an intermediate to find a working configuration.

To start this setup procedure, the client first needs to contact the SCS: the “Stitching Computation Service”. This service has info on the interconnections between all AMs, so it can find a path from AM to AM, passing through the needed AMs on the way. The client needs to receive info from the SCS about which AMs to contact in which order.

The SCS will then request the advertisement RSpec of the AMs periodically. To allow stitching, the advertisement RSpec must contain info about external links.

At imec/iMinds we run an SCS at <http://scs.atlantis.ugent.be:8081/geni/xmlrpc>. There is also a geni SCS. The SCS's must be configured with info about the AMs that offer external links.

Stitching is a very complicated topic. Here are a few points that might be good to know:

- The SCS is needed, because in some cases, a stitched link that goes from one AM to another, will actually cross one or more other AMs. Only the central SCS can know about this topology, as it talks to all involved AMs. It is this topology information that the SCS will send to clients.
- You can run your own SCS, and jFed can be configured to use this specific SCS for specific testbeds. However, it's probably easier and better in the long run if we just add the info to our SCS.
- The advertisement RSpec contains extra information related to stitching. This information is in a separate namespace. See the example below.
- The SCS gets a request RSpec as input, and appends stitching information to it. This request RSpec is then sent to each involved AM.
- The manifest RSpec contains cross-AM info about the current stitching setup.
- There is a special error code and message that AMs should return for failures related to VLANs not available for stitching.
- There are 2 important features that improve stitching setup time:
 - Support for “any” VLAN in the request RSpec: instead of specifying a VLAN, the client is allowed to set the special keyword “any” instead, and let the AM choose



D2.4: Testbed requirements, developments and integrations

- Up to date advertisement RSpecs: The advertisement RSpec always contains up to date info on which VLANs are currently available.
- The client code in the jFed experiment GUI that manages stitching is very complex, to handle all cases. The client must “negotiate” between the different AMs until a stitching configuration (= which VLANs on which AM) is found that works on all AMs.

Below is an extract from the wall2 stitching info in the advertisement request:

```

<stitching lastUpdateTime="2017-02-15T06:19:30Z"
xmlns="http://hpn.east.isi.edu/rspec/ext/stitch/0.1/">
  <aggregate
    id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm"
    url="https://www.wall2.ilabt.iminds.be:12369/protogeni/xmlrpc/am">
    <aggregatetype>protogeni</aggregatetype>
    <stitchingmode>chainANDTree</stitchingmode>
    <scheduledservices>false</scheduledservices>
    <negotiatedservices>false</negotiatedservices>
    <lifetime id="life">
      <start type="time">2017-02-15T06:19:30Z</start>
      <end type="time">2017-02-15T06:19:30Z</end>
    </lifetime>
    <node id="urn:publicid:IDN+wall2.ilabt.iminds.be+node+c300b">
      <port id="urn:publicid:IDN+wall2.ilabt.iminds.be+stitchport+c300b:4.2">
        <capacity>1000000</capacity>
        <maximumReservableCapacity>1000000</maximumReservableCapacity>
        <minimumReservableCapacity>1000</minimumReservableCapacity>
        <granularity>1</granularity>
        <link id="urn:publicid:IDN+wall2.ilabt.iminds.be+interface+c300b:4.2">
          <remoteLinkId>urn:publicid:IDN+wall1.ilabt.iminds.be+interface+c300a:4.1</remoteLinkId>
          <trafficEngineeringMetric>10</trafficEngineeringMetric>
          <capacity>1000000</capacity>
          <maximumReservableCapacity>1000000</maximumReservableCapacity>
          <minimumReservableCapacity>1000</minimumReservableCapacity>
          <granularity>1</granularity>
          <switchingCapabilityDescriptor>
            <switchingcapType>l2sc</switchingcapType>
            <encodingType>ethernet</encodingType>
            <switchingCapabilitySpecificInfo>
              <switchingCapabilitySpecificInfo_L2sc>
                <interfaceMTU>1500</interfaceMTU>
                <vlanRangeAvailability>197,300-349,750-
1000</vlanRangeAvailability>
                <vlanTranslation>false</vlanTranslation>
              </switchingCapabilitySpecificInfo_L2sc>
            </switchingCapabilitySpecificInfo>
          </switchingCapabilityDescriptor>
        </link>
      </port>
      ....
    </node>
  </aggregate>

```

3.5.3 Stitching alternative: Dedicated Ext. Network Connection

If you only need single hop stitching, you can make things easy, by using links to fake nodes, which represent certain VLANs. The disadvantage is that this is less transparent for end users. It is however, much easier to implement on the AM.

D2.4: Testbed requirements, developments and integrations

In the jFed GUI, this is implemented with the “Dedicated Ext. Network Connection” icon. You drag in this icon, and then make a link between a real node and this “virtual” node.

3.5.4 Stitching alternative: “tunnels” over internet using link_type

In this scenario, there are no dedicated links between testbeds. Instead, virtual links will be set up in software, that will run over the internet. Of course, this implies best-effort bandwidth and delay, instead of guaranteed bandwidth and delay.

These links can typically be used within a testbed as between testbeds. (Using them within testbeds is useful if the testbed doesn’t support dedicated links between nodes.)

In this scenario, you draw a link in jFed and change it’s type to match the type of virtual link. This link type is set using the `<link_type>` element in the RSpecs `<link>`.

Currently, the know link types are:

- Regular link: no link type, or `<link_type name="lan"/>` (between nodes of the same testbed)
- Sticked link: no link type, or `<link_type name="lan"/>` (between nodes of the different testbeds)
- GRE: `<link_type name="gre-tunnel"/>`
- EGRE: `<link_type name="egre-tunnel"/>`
- Proposed: VXLAN: `<link_type name="vxlan"/>`

Example RSpec:

```
<?xml version='1.0'?>
<rspec                                xmlns="http://www.geni.net/resources/rspec/3"                type="request"
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+testbed-
a.example.com+authority+am">
    <sliver_type name="raw-pc"/>
    <interface client_id="node0:if0">
      <ip address="192.168.0.1" netmask="255.255.255.0" type="ipv4"/>
    </interface>
  </node>
  <node client_id="node1" exclusive="true" component_manager_id="urn:publicid:IDN+testbed-
b.example.com+authority+am">
    <sliver_type name="raw-pc"/>
    <interface client_id="node1:if0">
      <ip address="192.168.0.2" netmask="255.255.255.0" type="ipv4"/>
    </interface>
  </node>
  <link client_id="link0">
    <component_manager name="urn:publicid:IDN+testbed-a.example.com+authority+am"/>
    <component_manager name="urn:publicid:IDN+testbed-b.example.com+authority+am"/>
    <interface_ref client_id="node0:if0"/>
    <interface_ref client_id="node1:if0"/>
  </link>
</rspec>
```

D2.4: Testbed requirements, developments and integrations

```
<link_type name="gre-tunnel"/>
<property source_id="node0:if0" dest_id="node1:if0" capacity="10000"/>
<property source_id="node1:if0" dest_id="node0:if0" capacity="10000"/>
</link>
</rspec>
```



D2.4: Testbed requirements, developments and integrations

4 DOCKER-AM AS EXAMPLE AM

This section describes the features and install instructions for the Docker AM (supporting IPv6 resources as well !), that we advise as example/template of an Aggregate Manager. Code for this can be found at <https://github.com/open-multinet/docker-am> .

4.1 SUPPORTED AGGREGATE MANAGER FEATURES

- Every basic feature (Allocate, Provision, Delete, Status, ListResources, Describe, Renew)
- Some PerformOperationalAction call are supported
 - `geni_update_users` : Update SSH authorized keys or add a user
 - `geni_reload` : If you want to "reset" your container
 - Other options have no effect
- You can provide a sliver-type to get different kind of containers (for example limited memory or CPU container). Check the advertisement RSpec, and have a look at `gcf_to_docker.py` for details.
- Install a custom docker image by providing a name from a DockerHub or a URL to a Dockerfile or a ZipFile containing a Dockerfile and dependencies.
- Restart the AM without losing the state of existing slivers: Running docker containers will keep running when the AM stops, and can be controlled again when the AM restarts. (You can safely remove `am-state-v1.dat` to clear the state and thus force config reload. You will need to kill any running docker containers manually in that case.)
- Multiple physical host for Docker. That means you can increase the scalability easily by setting up a new "DockerMaster" on remote host. To scale the setup, integration with kubernetes is probably preferable.
- `install` and `execute` can be used to install a zipfile in a specific directory and execute commands automatically when the container is ready.
- IPv6 per container can be configured in addition to the IPv4 port forwarding of the host.
- The is demo code that can be used as a basis to customize the AM. Two features are demonstrated in this code: **** Supporting custom non-container external resources.** (See `resourceexample.py`) **** Automatically adding a gateway proxy per slice.** (See "proxy" in the configuration parsing)

4.2 HOW TO INSTALL THE AM ?

4.2.1 Dependencies

```
apt-get install -y python2.7 python-lxml git python-m2crypto python-dateutil python-  
openssl libxmlsec1 xmlsec1 libxmlsec1-openssl libxmlsec1-dev python-pip
```

Pyro4 is also required (for remote dockermasters):



D2.4: Testbed requirements, developments and integrations

```
pip install pyro4
```

4.2.2 Download source code

- First clone the git repository

```
git clone https://github.com/open-multinet/docker-am.git
```

- Initialize submodule (geni-tools source code)

```
cd docker-am
cd geni-tools
git submodule init
git submodule update
git checkout develop
```

4.2.3 Configure AM

Two files are used to configure the AM.

4.2.3.1 gcf_config

This file is for the generic GCF configuration. This contains the basic AM setup. The docker AM specific functionality is activated by setting `delegate` to `testbed.DockerAggregateManager`

Path of this file : `docker-am/gcf_docker_plugin/gcf_config`

- `base_name` : Typically the name of your machine. This is the name used in the URN (`urn:publicid:IDN+docker.example.com+authority+am`)
- `rootcadir` : A directory where your trusted root certificates are stored. These are the certificates of the MA/SA servers who's users the AM will trust. (`wall2.pem` for example)
- `host` : This should be the DNS name of the server. It is used for binding the server socket. `0.0.0.0` is often a good choice if the hostname is not correctly configured on the server.
- `port` : You are free to choose a port. 443 is recommended (because it is infrequently blocked by client side firewalls).
- `delegate` : To activate the docker AM code, this must be `testbed.DockerAggregateManager`
- `keyfile` and `certfile` = Private key and certificate of the AM server. This is used for SSL authentication. See the section below.

4.2.3.2 docker_am_config

This is the configuration that is specific for the docker AM.

D2.4: Testbed requirements, developments and integrations

Path of this file : `docker-am/gcf_docker_plugin/docker_am_config`

The `[general]` section currently contains one parameter.

- `public_url`: the URL to the AM, as advertised in the `Getversion` reply. This URL must contain the FQDN of the host. A raw IP address is discouraged. The following values for the hostname are forbidden here: `0.0.0.0` `127.0.0.1` `localhost`

A `[proxy]` section is also allowed, but not mandatory (no automatic proxy is used if not specified). Check the example config for details.

Each other "section" in the config (a section start with `[name_of_the_section]`) in this file represents a DockerMaster (a dockermaster host one or more containers). If you want to configure multiple DockerMaster just duplicate the first section and change the name. Then, configure parameters :

- `max_containers` : The maximum number of container hosted by your DockerMaster
- `ipv6_prefix` : If you have an IPv6 address on your host, set the prefix in /64 or /80 (for example : `2607:f0d0:1002:51::`) and each container will be assigned an IPv6 in this range
- `dockermaster_pyro4_host`, `dockermaster_pyro4_password` and `dockermaster_pyro4_port` : Parameters to connect to the dockermanager using pyro4 RPC (only when using a remote dockermanager, to use a local docker service, skip these options)
- `node_ipv4_hostname` : The IPv4 of your DockerMaster host (will be used to expose an SSH port on the containers)
- `starting_ipv4_port` : This is the first range used by docker for port forwarding. For example if you set 12000, the first container should be reachable on port 12000, the second on port 12001, ... The AM uses the first port available from 12000 to `12000+max_containers`

4.2.4 Configure a DockerMaster

You can run the docker service on either the same node as the AM, or use the remote DockerMaster feature (which uses pyro4 for RPC) to run the service on another node.

On the node where docker needs to run, do the following:

First, install the docker engine: <https://docs.docker.com/engine/installation/>

Then, make sure the daemon is running (with systemd) : `systemctl start docker.service`
If you want the docker daemon restart automatically after reboot : `systemctl enable docker.service`

See the section "Configuring a remote DockerManager (Optional)" for more details on a remote DockerManager

D2.4: Testbed requirements, developments and integrations

4.2.4.1 Configure IPv6 for Docker

If you want to use IPV6 on your container, you have to configure Docker bridge to use a specified prefix.

Create a new file (this path is available on Debian based distribution) `/etc/systemd/system/docker.service.d/docker.conf` :

```
[Service]
ExecStart=
ExecStart=/usr/bin/docker daemon --ipv6 --fixed-cidr-v6="2607:f0d0:1002:51::/64" -
H fd://
```

By replacing the IPv6 example by your own with the proper prefix length (64 or 80)

Then restart your docker daemon : `systemctl restart docker.service`

4.2.5 Generate certificate and key

You need to get a server key and certificate for the AM. You can either get a real one (using your regular way to get SSL Certificate, or using "Let's Encrypt"), or you can create a self signed AM server certificate. In the later case, you will need to add the self signed certificate to the trust store of all clients (which is not a big deal, as you need to add other server info anyway).

It is advised not to use the `bootstrap-geni-am/geni-tools/src/gen-certs.py` script provided by gcf, it does not generate a good AM server certificate. A valid server certificate should have:

- A Subject Name containing a CN equal to the server hostname
- One or more Subject Alternative Names of type DNS matching the server hostname(s)
- The server hostname mentioned in the 2 points above, should be a DNS name, never a raw IP address
- There is no real need for the certificate to contain a Subject Alternative Name of type URI that contains the URN of the AM. But it is off course no problem if it is included.

Also note that your AM server certificate has nothing to do with the root certificate of your clearinghouse (= MA/SA). The clearinghouse root certificate is used for trusting credentials and for SSL client authentication, it has nothing to do with SSL *server* authentication!

To generate a self signed server certificate, you can use the provided script:

```
cd generate-AM-server-cert/
./generate-certs.sh
```

D2.4: Testbed requirements, developments and integrations

Make sure the location of the server key and certificate matches the keyfile and certfile options specified in `bootstrap-geni-am/gcf_docker_plugin/gcf_config`
You can find some more details on how a certificate can be generated at <https://stackoverflow.com/a/27931596/404495>

4.3 STARTING THE AM

```
sh docker-am/run_am.sh
```

Or with the systemd service : `cp am_docker.service /etc/systemd/system/`

Edit the WorkingDirectory according to your installation, reload the systemd daemon

```
: systemctl daemon-reload, then start the AM : systemctl start am_docker.service
```

Check the status : `systemctl status am_docker.service`

4.4 TRUST YOUR C-BAS INSTALLATION

If you use C-BAS as Member Authority (MA) and Slice Authority (SA), you have to trust credentials from this. To do, just copy certificates used by C-BAS in your "rootcadir" (configured in `gcf_config`), usually there stored `C-BAS/deploY/trusted/certs`.

Restart your AM, if you check the output of the server you should have this at the beginning :

```
INFO:cred-verifier:Adding trusted cert file sa-cert.pem
INFO:cred-verifier:Adding trusted cert file ma-cert.pem
INFO:cred-verifier:Adding trusted cert file ch-cert.pem
INFO:cred-verifier:Adding trusted cert file ca-cert.pem
INFO:cred-verifier:Combined      dir      of      4      trusted      certs      /root/C-
BAS/deploY/trusted/certs/      into      file      /root/C-
BAS/deploY/trusted/certs/CATedCACerts.pem for Python SSL support
```

Of course this AM is not C-BAS dependent and you can trust the certificate of any MA/SA. For example you can trust the MA/SA from iMinds, so you will be able to create a slice on wall2 and use it on your AM.

4.5 CONFIGURING A REMOTE DOCKERMANAGER (OPTIONAL)

You can set up several DockerManager hosted on different physical machine in order to increase scalability (for example).

4.5.1 Configure the remote

First of all you need to install dependancies on the remote host :

D2.4: Testbed requirements, developments and integrations

```
apt-get install python2.7 python-pip git
```

```
pip install pyro4
```

And install docker-engine : <https://docs.docker.com/engine/installation/>

Now download the source code repository :

```
git clone https://github.com/open-multinet/docker-am.git
```

```
And try : python2 docker-am/gcf_docker_plugin/daemon_dockermanager.py --host 127.0.0.1
```

You should get a warning about not using any password and a URI the server listening on.

You can use it in this way but it's more convenient to configure a systemd service. To do this, just copy the service file :

```
cp bootstrap-gei-am/dockermanager.service.sample /etc/systemd/system/dockermanager.service
```

Then edit the WorkingDirectory and ExecStart line in this file to match to your configuration. The "--host" parameter should be an IP reachable from the AM, so a public IP or, if your AM is on the same network a private IP.

Finally, do `systemctl daemon-reload && systemctl start dockermanager.service` and check with `systemctl status dockermanager.service`

4.5.2 Configure the AM

On the AM, edit `docker-am/gcf_docker_plugin/docker_am_config` and add or edit a section to match the three parameters (`dockermaster_pyro4_host`, `dockermaster_pyro4_password`, `dockermaster_pyro4_port`) with the parameters set on the remote

Then delete `am-state-v1.dat` (to force configuration reload) and restart your AM.

4.6 HOW TO ADAPT THIS AM TO YOUR INFRASTRUCTURE ?

If you want to test the AM with your hardware (not with Docker or in addition to Docker) you have to develop your own Python Class which manages your hardware.

You can follow the docker model as example. It is based on three classes : `DockerMaster` (`dockermaster.py`), `DockerContainer` (`dockercontainer.py`), and `DockerManager` (`gcf_to_docker.py`).

- `DockerMaster` is more or less just a pool of `DockerContainers`, because a `DockerMaster` should be a unique physical machine

D2.4: Testbed requirements, developments and integrations

- DockerContainer represents a single docker container, with some information like to ssh port, the IPv6, ...
- DockerManager is a generic class to manage docker from Python

So, if you want to represent a physical machine which can be reserved by a user the Python class should be a merge between `DockerMaster` and `DockerContainer`, you should inherit your class from `ExtendedResource`. This class is formed of all used methods by the AM, so you have to implement at least those methods (also have a look to `geni-tools/src/gcf/geni/am/resource.py`)

To kickstart coding this, the class "ResourceExample" is provided. It's dummy external resource manager, which can act as a starting template. You must write configuration processing code in `testbed.py` > `_init_` to enable the resource. The dummy resource does nothing, to get started, edit the file `resourceexample.py` and find lines with `ssh="exit 0;"` and follow the instruction on the lines above. Note that the kickstart code assumes that your AM has SSH access to the external resource.

Once your resources are ready, you have to init them in `testbed.py` in the `_init_` method by adding them to the aggregate configuration parsing. Be sure to delete `am-state-v1.dat` when testing, to force configuration reload.

Note : You should probably implement a generic wrapper for your infrastructure like `DockerManager`, it's easier to maintain, especially if you have different kinds of resources.

4.7 DEVELOPMENT NOTES

If you want make some contribution to this software, here there is a quick explanation of each file :

- `testbed.py` : The aggregate manager main class, handle calls from the API
- `dockermaster.py` : Docker Master is a pool of docker container, it is called to get some `DockerContainer` instances
- `dockercontainer.py` : Represents a Container with methods to manage it
- `gcf_to_docker.py` : The `DockerManager` class, used as generic wrapper for Docker in Python, mostly used by `DockerContainer`
- `resourceexample.py` : A dummy resource to kickstart you to develop your own resource
- `extendedresource.py` : A generic resource class which adds some usefull methods to the base `Resource` class (which is in `resource.py`, in the `geni-tools` repo)
- `daemon_dockermanager.py` : The daemon used to create a remote `DockerMaster` using `Pyro4` framework.



D2.4: Testbed requirements, developments and integrations

4.8 ADDITIONAL INFORMATIONS

- Objects are serialized in `docker-am/am-state-v1.dat`, so you can restart the AM without consequence
- Slivers expiration is checked every 5 minutes, and on each API call
- Warning : If you restart the host, docker containers are lost, to keep consistent state delete `am-state-v1.dat` before restarting the AM.
 - It will mostly work without deleting the file but you could have some unexpected behaviors

4.9 TROUBLESHOOTING

- If you get the error "Objects specify multiple slices", you probably made a typo in `component_manager_id` (during allocate call)
- If your configuration is not taken in account, delete `docker-am/am-state-v1.dat` and remove all running containers `docker rm -f $(docker ps -a -q)`
- If you get an SSL error (like host not authenticated) check if you correctly add your AM/SA certs in trusted root

