
BonFIRE User Documentation

Release 4.0.5

BonFIRE Staff

January 29, 2014

CONTENTS

1	Background	3
1.1	About BonFIRE	3
1.2	An overview of BonFIRE features	3
1.3	Infrastructure	8
2	Before You Start	15
2.1	Steps To Getting started	15
2.2	Registering a BonFIRE Account	15
2.3	Setting Up SSH	16
2.4	Uploading SSH Key to BonFIRE	21
2.5	SSH Gateway Configuration	23
2.6	Getting Support	25
3	Running Experiments in BonFIRE	27
3.1	Introduction	27
3.2	Using the BonFIRE Testbeds	30
3.3	Quota and Usage	34
3.4	Creating Experiments	35
3.5	Tutorials and Experiment Scenarios	39
4	Client Tools	81
4.1	Overview of Client Tools	81
4.2	Experiment Descriptors	81
4.3	BonFIRE Portal	113
4.4	OCCI/API via HTTP(cURL)	118
4.5	Command Line Interface Tools	126
4.6	Restfully	135
5	Compute	141
5.1	Overview of Compute	141
5.2	Contextualisation	141
5.3	On Request Resources	148
5.4	Deploying Compute Resources in BonFIRE	155
6	Storage	167
6.1	Overview of Storage	167
6.2	Datablock Storage	167
6.3	Network File System	181
6.4	Advanced Storage Features	182

7 Networking	189
7.1 Overview of Networking in BonFIRE	189
7.2 Basic network resources	191
7.3 Emulated Network at the Virtual Wall	196
7.4 Controlled public networking with FEDERICA	197
7.5 Controlled Bandwidth with AutoBAHN	209
8 Monitoring	223
8.1 Overview of Monitoring Options in BonFIRE	223
8.2 Managing Data Storage for Monitoring	226
8.3 How To Set Up Monitoring	228
8.4 Accessing and Downloading Monitoring Data	254
9 Elasticity	261
9.1 BonFIRE EaaS - Elasticity as a Service	261
9.2 How To Use the BonFIRE EaaS	261
10 Experiment Services	265
10.1 Notifications	265
10.2 COCOMA	279
10.3 Provenance	321
11 Reference Docs	323
11.1 BonFIRE Architecture	323
11.2 BonFIRE API Specification	325
11.3 Media Type	333
11.4 Testbed Specificities	335
11.5 Deploying resources on Amazon EC2	337
11.6 Instance Types	347
11.7 VM Images	348
11.8 XML Schema Definition	349
11.9 Experiment Lifecycle	359
11.10 States of OCCI Resources	359
11.11 BonFIRE OVF Schema Definition	361
11.12 Default Monitoring Metrics	363
11.13 Release Notes	365
12 FAQ	371
12.1 FAQ	371
Index	375

This is the user documentation of the BonFIRE service.

BACKGROUND

1.1 About BonFIRE

1.1.1 What is BonFIRE?

BonFIRE is an EU project which is designing, building and operating a multi-site cloud-based facility on top of six infrastructure testbeds operated by six project partners. The infrastructure sites offer heterogeneous Cloud resources, including compute, storage and network. For more information about the infrastructure offerings, click [here](#).

The *BonFIRE Architecture* is designed to support research on applications, services and systems targeting in particular, but not exclusively, the Internet of Services (IoS) community. It has key functionalities such as monitoring at infrastructure and virtual machine level, experiment management with a single declarative experiment descriptor, elasticity, and resource management for deployment of application software over a variety of differently configured resources (compute, storage, and network). More information about the various features in BonFIRE is please see the *Overview of Features* page.

To find out what experimenters have done with BonFIRE already, please check out <http://www.bonfire-project.eu/innovation>.

1.1.2 Using BonFIRE

BonFIRE is geared towards experimentation and research into Cloud/IoS, and offers the facilities to easily create, manage and monitor experiments, whilst giving the experimenters more information and control of Cloud resources than what is offered by other public Cloud providers. To better explain what we mean by an experiment, have a look at the *Introduction*.

Interactions with BonFIRE are done via a RESTful interface based on **OCCI**, referred to as the *BonFIRE API*. Each of the infrastructure sites in BonFIRE are accessed through this API, which makes it very easy to conduct multi-site, geographically distributed, experiments. Moreover, all resources on all infrastructure sites are accessed with a single BonFIRE sign-on that you set up when you register an account (see *Registering a BonFIRE Account*).

1.2 An overview of BonFIRE features

An overview of features of BonFIRE are discussed below with links for more information.

Please note that not all features are supported by all testbeds in BonFIRE. For an overview of the features supported by each testbed and more information about using the different testbeds, please see the page on *Using the BonFIRE Testbeds*.

1.2.1 Multiple client tools

To create experiments and Cloud resources in BonFIRE, it is possible to issue raw HTTP commands to the BonFIRE API via, for example, [cURL](#). However, we also offer multiple client tools for interacting easily and effectively with BonFIRE:

- BonFIRE Portal: a graphical client tool that operates via your web browser, which is very intuitive to use. The Portal also gives you the option to set up a Managed Experiment in a step-by-step procedure to define the initial deployment of compute, storage and networking resources for your experiment. As part of this procedure, the Portal will create an *Experiment Descriptor*, which you can save and use again.
- BonFIRE Experiment Descriptor: instead of following a step-by-step procedure on the Portal, you can write/edit an experiment descriptor in either JSON or OVF. These descriptors are interpreted by an Experiment Manager, allowing you to describe all of the resources for an initial deployment of an experiment without needing to know OCCI or how to form HTTP messages.
- Restfully: a general-purpose client library for RESTful APIs, which is written in Ruby. Its goal is to abstract the nitty-gritty details of exchanging HTTP requests between the user-agent and the server. Restfully also allows you to write scripts for your experiment deployment which can be automatically executed.
- Command Line Interface Tools: BonFIRE also offers CLI tools that are meant to provide users with a way to interact with the BonFIRE API from a command line interface. This could be used in an interactive, manual, fashion or programmatically (i.e., scripted).

1.2.2 Infrastructure sites and resources

BonFIRE integrates multiple Infrastructure that are geographically distributed across Europe, offering heterogeneous resources: EPCC in Edinburgh, Scotland; HP Cells in Bristol, England; Inria in Rennes, France; HLRS in Stuttgart, Germany; iMinds Virtual Wall in Ghent, Belgium; PSNC in Poznan, Poland; Wellness Telecom in Seville Spain.

Each infrastructure site offers compute, storage and network resources, which can be deployed within an experiment in BonFIRE. Some infrastructure sites offer additional functionalities, such as on-request resources (at Inria) and advanced network emulation (at the Virtual Wall). The resources are accessed with single sign-on via SSH.

1.2.3 On-request compute resources

Inria currently offers on-request resources in BonFIRE, allowing experimenters to reserve large quantities of physical hardware (162 nodes/1800 cores available). This gives experimenters the flexibility to perform large-scale experimentation, as well as providing greater control of the experiment variables as exclusive access to the physical hosts is possible. It is also possible to control which physical host Virtual Machines are deployed to.

1.2.4 Virtual machines and instance types

BonFIRE offers several Virtual Machine (VM) images based on Debian Squeeze, which vary in storage size. The storage size can be further extended with block storage, as discussed below.

Experimenters can create new VM images by deploying a BonFIRE base image, installing and configuring software, and saving this with a desired name. The size of the VM image will be the same as the base image. BonFIRE does not currently offer the ability for experimenters to create VM images locally and upload to BonFIRE because a certain amount of configuration is required for the images to integrate in BonFIRE.

When deploying compute resources, an instance type must be set. The instance types are labelled according to a small/medium/large type of taxonomy, which vary the number of virtual CPU cores and RAM size. Please note that the availability of instance types vary across the different infrastructure sites. It is, however, also possible to deploy

custom instance types, giving you fine-grained control of the compute resource specifications (including percentage of a CPU).

1.2.5 Storage

As mentioned above, the VM images come with a certain OS storage size. If these images do not provide enough storage, the VMs can be extended with storage resources. These are DATA BLOCKS that you can define the file system of (e.g., ext3, jfs, reiserfs) and the desired storage size, which are mounted in the VMs. It is possible to make this storage extend the root partition of your compute node instead of being mounted as a separate partition.

By default, storage resources that are created in BonFIRE are duplicated when they are deployed with compute resources, and are destroyed after the experiment duration has finished. Whilst other resources (compute and network) must be created within the context of an experiment, storage resources can be created outside of an experiment and will therefore outlive the duration of any experiments that use that storage resource. Moreover, it is possible to persist the storage resources on the OpenNebula testbeds in BonFIRE (see [Infrastructure](#):), so that data written to the storage during the execution of an experiment is saved when the compute resource is shut down.

Another special type of storages is the shared storage. At the moment they can only be created at the `be-ibbt` testbed. Unlike the other storage types, Shared storages are accessible by multiple computes at the same time. If the storage was created outside the scope of an experiment, it can even be accessed by computes from different experiments.

1.2.6 Networking

The infrastructure sites in BonFIRE are interconnected via the public Internet, but not all providers are able to provision public IPv4 addresses. Therefore, BonFIRE operates a VPN to tunnel traffic between sites over what we call the BonFIRE WAN. For more information, please have a look at the [Networking Overview](#).

One of the usage scenarios in BonFIRE is [Cloud With Emulated Network Implications](#), for which it is possible to control the networking on the Virtual Wall. As an experimenter it is possible to control the networking in three ways:

- **Network topologies:** the Virtual Wall allows you to set up any network topology you want and, by default, provides shortest path routing between any compute resources installed on the network nodes, with Gigabit connectivity.
- **Network impairments:** if you want to study the influence of network impairments on the performance of your system under test, you can specify link characteristics such as bandwidth, latency and loss rate on each link, both statically and dynamically.
- **Background traffic:** it is also possible to introduce TCP or UDP streams on the links to represent background traffic, with parameters to specify the packet size and the throughput (#packets/second).

For more information about the controlled networking at the Virtual Wall, see [Emulated Network at the Virtual Wall](#):

In the next release of BonFIRE, there will be more control of the networking between certain sites. An integration with [AutoBAHN](#) will give Bandwidth-on-Demand between VMs on the EPCC and PSNC sites. Interconnection with [FEDERICA](#) is also in progress.

1.2.7 Monitoring

BonFIRE provides fine-grained monitoring information about the virtual resources in your experiments as well as the physical hosts (on the OpenNebula testbeds) that the virtual resources are deployed on. The latter is a unique feature of BonFIRE, which allows you to correlate observations from the data collected in your VMs (VM performance metrics or applications/services running in your VMs) with events on the physical host.

The monitoring solution offered by BonFIRE is based on the open source monitoring software [Zabbix](#). The software comprises two major software components: Zabbix server and Zabbix agent. The server is referred to as an ‘Aggregator’ in BonFIRE, which is deployed on a separate resource, whilst the agents reside in the deployed VMs. The Zabbix Aggregator collects monitoring information reported by the Zabbix Agents, giving you a single view of the data for all your experiment resources.

There is easy access to the Zabbix GUI via the BonFIRE Portal, where you can view the monitoring data and make any configurations of the metrics you want to monitor. The default configuration in BonFIRE will monitor over 100 VM metrics and 16 infrastructure metrics. In addition to these, you can add your own metrics, which makes it easy to monitor your application/services if desired. Moreover, you can also specify the monitoring metrics in OCCI when you deploy compute resources.

1.2.8 Elasticity

It is the goal to make Cloud features easy to use in BonFIRE, and Elasticity as a Service is a new feature in release 3 to do just that. Elasticity refers to dynamically increasing or decreasing resources according to load/demand, which is a popular selling point of ‘the Cloud’. BonFIRE provides a VM image with an Elasticity Engine that performs load balancing based on either [HAProxy](#) or [Kamailio](#).

The EaaS in BonFIRE uses the monitoring information from the Zabbix Aggregator (see above) deployed in an experiment for retrieving information regarding the load of the compute resources. It interoperates with the BonFIRE API for dynamically adding and removing compute resources based on elasticity triggering rules specified by the user.

1.2.9 Notifications

As your experiment is executing, particularly if elasticity is used, it may be important to track changes/events in the experiment. BonFIRE offers a way for clients to subscribe to notifications of experiment state changes, as well as resource state changes, such as when they are created, updated, or destroyed. The state changes are available as events on a message queue in BonFIRE, which uses [RabbitMQ](#).

The experiment states are in accordance with the Experiment Lifecycle, and the resource (compute, storage and network) states are in accordance with the specified OCCI States.

1.2.10 Contextualisation

Contextualisation elements in OCCI can be used to pass initialisation values for an experiment in the form of simple key-value pairs. This is used by the BonFIRE testbeds and the BonFIRE API, but is also available to the users for specifying, for example:

- The IP address of the Zabbix Aggregator when deploying compute resources (the Portal will do this automatically).
- Any custom monitoring metrics.
- Elasticity trigger rules, if the EaaS is used.
- Post-install scripts, which can run after the VM has been deployed.
- Any additional SSH keys, to allow multiple users access to experiment VMs.

The contextualisation element is generic, so that any key-value pairs could be defined. The contextualisation variables are written to `/etc/default/bonfire`, so you can source this file to access the variables within your experiment VMs.

1.2.11 Advanced features

As discussed above, BonFIRE offers the features you would expect from a Cloud provider. However, BonFIRE goes beyond that to offer a facility for research that gives maximum control and observability. BonFIRE exceeds the standard by offering the following advanced features:

- **Experiment descriptors:**

As discussed above, there are many client tools available in BonFIRE. An important feature of BonFIRE is the availability of experiment descriptors, which can be used to specify the deployment of compute, storage and network resources, along with contextualisation such as specifying monitoring metrics and elasticity rules. The experiment descriptor can be specified in either JSON or OVF, depending on preference. Thereby, it is very easy to share and extend upon existing experiment deployments.

- **Infrastructure monitoring:**

Unlike other public Cloud providers, BonFIRE is able to expose much more information about what is going on “behind the scenes” to help maximise the observability in the experiments. Infrastructure monitoring is available on the OpenNebula testbeds in BonFIRE, exposing metrics such as the number of VMs and CPU load of the physical host. This source of monitoring information can be essential to explain observations made in experiments that could be caused events outside the experimenter’s control.

- **Time-stamped information for internal BonFIRE processes:**

In its quest for advanced monitoring, BonFIRE provides the timestamps of specific experiment events to the experimenters. Triggered by specific experimenter requirements, the timestamps of VM requests going through the Experiment Manager are logged and served via HTTP.

- **Infrastructure and VM logs:**

To further increase the observability in BonFIRE, the testbeds publish hypervisor information about the state of the site hosts. Additionally, in-depth timestamped information of the status changes of each VMs is available; easily accessible from the VM log linked to on the VM page on the BonFIRE Portal.

- **Shared storage:**

Another special type of storages is the shared storage. At the moment these storage resources can only be created at the be-iibbt testbed but may be used by compute resources at any of the BonFIRE testbeds. Unlike the other storage types, shared storages are accessible by multiple, potentially distributed, computes at the same time. If the storage was created outside the scope of an experiment, it can even be accessed by computes from different experiments.

- **Custom instance types:**

Central to VM instantiation is the notion of instance types; the combination of CPUs and RAM available to the VM created. BonFIRE lets you control the make up of your VM by allowing to specify any such combination. Although combinations exceeding the available resources will not be instantiated, this feature is another example of BonFIRE functionality particularly conducive to experimentation.

- **Exclusive access to physical hosts:**

Contention between VMs running on the same physical host is to be expected, and although infrastructure monitoring can help interpret experiment results, BonFIRE offers experimenters control of this by gaining exclusive access to physical hosts. This is possible via the on-request resources.

- **Control the deployments of VMs on physical clusters:**

BonFIRE has always allowed its users to specify which site they want their VMs to run on. Since Release 3, all BonFIRE interfaces allow to also specify on which specific host to deploy that VM. This is essential for using on-request resources, but also provides fine-grained control of your experiment. Therefore, in combination with exclusive access to physical hosts, it is possible to run controlled experiments to, for example, ensure that there is no contention from other VMs, or to actually induce contention by varying the load of the physical host.

- **Background network traffic emulation:**

The BonFIRE Virtual Wall testbed supports configuration of the network topology and network impairments such as bandwidth, latency and loss rate on each network link. New in Release 3 is the ability to also introduce background traffic on a user-specified network link. The background traffic can be configured to be a TCP or UDP stream, with parameters to specify the packet size and the throughput (#packets/second).

1.3 Infrastructure

This section provides an overview of the infrastructures specifications at the individual sites in BonFIRE. The features that each site supports is not covered here; instead refer to the page on [Using the BonFIRE Testbeds](#). Also, note that each site supports different *Instance Types* that define the specification of the compute resources you can deploy in terms of CPU and RAM.

Each of the local infrastructures offers a set of permanent and on-request resources. The permanent resources can be used for small-scale experiments and are available permanently during the BonFIRE project lifetime. The on-request infrastructure is available for large-scale testing, but is not exclusively dedicated to the BonFIRE project. These resources can only be reserved by requesting them directly from the infrastructure provider. The BonFIRE infrastructure map depicted below provides an overview of the permanent and on-request resources available.

BonFIRE provides an infrastructure [Health Map](#) to give an overview about the operation status of all testbed sites. For more details about the infrastructure offer at each site, see below.



Figure 1.1: BonFIRE infrastructure map for permanent and on-request resources

Permanent Resources:

Provider	Core	RAM	Storage	nodes
EPCC	176	416GB	6-16TB	7 workers
HP	128	136GB	5TB	32 workers
iMinds	192	384GB	4TB	16 workers
Inria	96	256GB	2.4TB	4 workers
HLRS	344	1068GB	12TB	36 workers
PSNC	48	192GB	600GB	4 workers
WT	48	64GB	3TB	8 workers

On Request Resources:

Provider	Core	RAM	Storage	nodes
EPCC	X	X	X	X
HP	384	144GB	32TB	96 workers
iMinds	816	3TB	17TB	68 workers
Inria				>160 workers
HLRS	X	X	X	X
PSNC	X	X	X	X
WT	X	X	X	X

The BonFIRE infrastructure providers are:

- The University of Edinburgh, United Kingdom (EPCC, `uk-epcc`)
- Hewlett-Packard Labs, United Kingdom (HP, `uk-hplabs`)
- Interdisciplinary Institute for Broadband Technology, Belgium (iMinds, `be-ibbt`)
- *Inria* (`fr-inria`)
- Universität Stuttgart, Germany (USTUTT-HLRS, `de-hlrs`)
- The Poznan Supercomputing and Networking Centre, Poland (PSNC, `pl-psnc`)
- The Wellness Telecom, Spain (WT, `es-wellness`)

1.3.1 EPCC

General configuration

`uk-epcc` runs OpenNebula, in a version derived from OpenNebula 3.0 for BonFIRE.

- **Hypervisor used** Nodes run XEN 3.0.3
- **Image management** Block devices are managed using a custom TM driver.
- **Image storage** Images are stored using the “raw” format
- **OpenNebula scheduler configuration:** These values are subject to frequent changes. Their meaning can be explored in <http://opennebula.org/documentation:archives:rel3.0:schg>
 - `-t` (seconds between two scheduling actions): 30
 - `-m` (max number of VMs managed in each scheduling action): 300
 - `-d` (max number of VMs dispatched in each scheduling action): 30
 - `-h` (max number of VMs dispatched to a given host in each scheduling action): 1

Permanent resources

EPCC provides eight dedicated nodes as permanent resources. Two of these nodes offer four, 12-core AMD Opteron 6176 (2.3GHz) with 128GB of memory each. Five of these nodes offer two, 8-core AMD Opteron 4274 HE (2.5GHz, 2.8GHz turbo frequency) with 32GB of memory each (total core count is 176; total memory count is 416GB of which approximately 400GB is available for VMs). The final dedicated node hosts the EPCC front-end and service VMs. It currently offers 6 TB of storage to BonFIRE, with another 6-10TB available, subject to RAID configuration.

Information about Physical Host Details at EPCC .

On-request resources

EPCC decided to provide a substantial infrastructure, permanently available to the BonFIRE users. No on-request resources will be made available.

Networking

The nodes are connected by Gigabit ethernet. 8 public IP addresses are presently available for VMs (more may be made available in future). There are no firewall restrictions on public interfaces.

Other networking features available are a bandwidth-on-demand service to PSNC via AutoBAHN, interconnection with FEDERICA and private (to an experiment) internal networks. IPv6 networking is planned for a future release.

1.3.2 HP

Permanent resources

HP offers 32 dedicated dual processor nodes as permanent resources. These 64 processors are Xeon X5450 (3GHz) and the nodes have 5TB shared storage. Switched Gigabit Ethernet networking interconnections are available between these nodes.

On-request resources

HP may provide a maximum of 96 additional nodes and a maximum of an additional 32 TB of storage in the SAN dedicated to BonFIRE. This additional infrastructure will be provided upon request and will be strictly limited to the available capacity in the HP cloud research infrastructure.

1.3.3 iMinds

General configuration

be-ibbt operated by iMinds runs Emulab for configuration of the network topology and impairments. No virtualization technologies are used as each image is mapped onto one hardware node.

Permanent resources

be-ibbt makes 16 nodes available as dedicated resources. These Dual Intel Xeon E5645 six-core nodes each have 250 GB storage. Up to 6 Gigabit Ethernet networking interfaces are available per node. These interfaces allow interconnecting the nodes in an emulated network, with controlled link characteristics for delay, bandwidth capacity and packet loss rate.

On-request resources

Up to 84 nodes can be reserved for an experiment for a limited period, as long as the reservation is made sufficiently in advance. Regarding the storage capacity, the use of those 84 nodes would make a total of 21 TB available.

1.3.4 Inria

General configuration

fr-inria runs OpenNebula, in a version derived from OpenNebula 3.6 for BonFIRE.

- **Hypervisor used** Nodes run XEN 3.2
- **Image management** Inria's setup has been described in a blog entry on OpenNebula's blog platform: <http://blog.opennebula.org/?author=59>

Basically, NFS is configured on the hypervisor of the service machine and mounted on the OpenNebula frontend and on workers. TM drivers are modified to use dd to copy VM images from the nfs mount to the local disk of the worker node (local LV to be precise), and cp to save MV images to NFS. This way, we :

 - have a efficient copy of images to workers (no ssh tunneling)
 - may have significant improve thanks to NFS cache
 - don't suffer of concurrent write access to NFS because VMs are booted on a local copy
- **Image storage** Images are stored using the “raw” format
- **OpenNebula scheduler configuration:** These values are subject to frequent changes. Their meaning can be explored in <http://opennebula.org/documentation:archives:rel3.0:schg>
 - `-t` (seconds between two scheduling actions): 10
 - `-m` (max number of VMs managed in each scheduling action): 300
 - `-d` (max number of VMs dispatched in each scheduling action): 30
 - `-h` (max number of VMs dispatched to a given host in each scheduling action): 2

Available resources

Permanent resources

Inria provides 4 dedicated worker nodes (DELL PowerEdge C6220 machines) as permanent resources. These worker nodes have the following characteristics:

- **CPU** 2 Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz, Hyperthreading enabled, with 6 cores each
- **Memory** 64GiB, in 8*8GiB, DDR3 1600 MHz memory banks
- **Local storage.** * 2* 300GB SAS storage.
- **Network.** 2* 1GB/s ethernet linked bonded together

One server nodes with 2+8 disks (RAID1 for system, RAID 5 on 8 SAS 10k 600G disks), 6 cores and 48GB of RAM, 2 cards of 4 Gbps ports to host the different services needed to run the local testbed. Gigabit Ethernet networking interconnections are available between these nodes, with bonds to increase performance (2GB/s on nodes, 4GB/s on the server).

This infrastructure is monitored for power consumption using eMAA12 PDUs from Eaton.

Information about Physical Host Details at Inria .

On-request resources

fr-inria can expand over the 160 nodes of Grid'5000 located in Rennes. When using on-request resources of Grid'5000, BonFIRE users have a dedicated pool of machines that can be reserved in advance for better control of experiment conditions, but nevertheless accessible using the standard BonFIRE API. The interface of the reservation system is documented in *the dedicated page*

When requesting resources, a description of the nodes made available is shown on the web interface so the user can choose between the 4 available types of nodes. Parapluie nodes are instrumented for power consumption with the same pdus as the permanent infrastructure

1.3.5 USTUTT

Like most of the other providers, de-hlrs runs OpenNebula 3.6 in a version derived for BonFIRE.

General configuration

- **Hypervisor used** Nodes run XEN 3.1.2
- **Image management** Block devices are managed using the same modified version of the LVM manager as used by *Inria*.
- **Image storage** Images are stored using the “raw” format
- **OpenNebula scheduler configuration:** These values are subject to frequent changes. Their meaning can be explored in <http://opennebula.org/documentation:archives:rel3.0:schg>
 - -t (sconds between two scheduling actions): 30
 - -m (max number of VMs managed in each scheduling action): 300
 - -d (max number of VMs dispatched in each scheduling action): 30
 - -h (max number of VMs dispatched to a given host in each scheduling action): 1

Permanent resources

USTUTT provides 36 dedicated worker nodes with different hardware combinations:

- **14 nodes:** 2x Quad Core Intel Xeon @ 2.83 GHz, 16GB RAM
- **14 nodes:** 2x Quad Core Intel Xeon @ 2.83 GHz, 32GB RAM
- **6 nodes:** 2x Dual Core Intel Xeon @ 3.2 GHZ, 2GB RAM
- **2 nodes:** 4x AMD Opteron 12 cores @ 2.6GHz, 196GB RAM

These 4 types of various architectures offer users a lot of capabilities for performing their experiments. Every node can be accessed directly by specifying the <host> element during the creation of a compute resource. For a better overview of all available resources, USTUTT provides a current summary of the worker nodes at <http://nebulosus.rus.uni-stuttgart.de/one-status.txt>. In addition, a storage server with a total amount of 12TB disk space supplies the experimenters of the BonFIRE project. All these nodes are connected via Gigabit Ethernet network interconnections.

Information about Physical Host Details at HLRS .

On-request resources

Currently, USTUTT is not providing any additional on-request infrastructure.

1.3.6 PSNC

Currently p1-psnc provides OpenNebula 3.0.0 in a version derived from OpenNebula 3.0 for BonFIRE.

General configuration

- **Hypervisor used** Nodes run QEMU KVM 1.0
- **Image management** The images are managed via SSH
- **Image storage** Images are stored using the “raw” format
- **OpenNebula scheduler configuration:** These values are subject to frequent changes. Their meaning can be explored in <http://opennebula.org/documentation:archives:rel3.0:schg>
 - -t (seconds between two scheduling actions): 10
 - -m (max number of VMs managed in each scheduling action): 10
 - -d (max number of VMs dispatched in each scheduling action): 2
 - -h (max number of VMs dispatched to a given host in each scheduling action): 1

Permanent resources

PSNC provides four dedicated nodes as permanent resources. Each of nodes offers two six-core Intel Xeon E5645 with a total amount of 600GB storage. Switched Gigabit Ethernet networking interconnections are available between multiple interfaces on these nodes.

- **4 nodes:** 2x Intel Xeon E5645 @ 2.4 GHz, 48GB RAM, 600 GB disk (RAID1)

These nodes are used for both production and integration testbeds including management node.

On-request resources

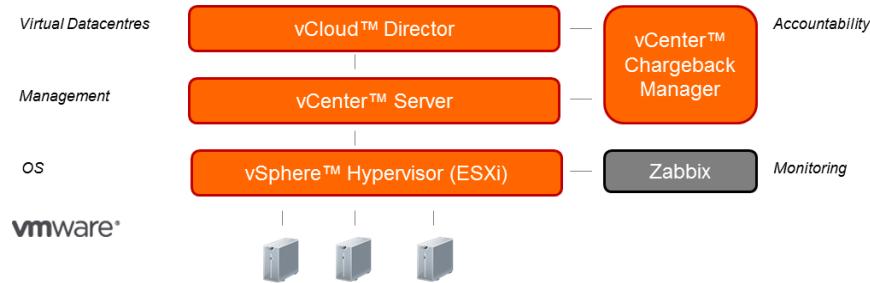
PSNC decided to provide a substantial infrastructure, permanently available to the BonFIRE users and no additional on-request resources will be made available.

1.3.7 WT

General configuration

es-wellness operated by Wellness Telecom runs VMware vCloud technology. Overall, VMware stands out as the cloud leader due to the reach and maturity of its offerings. In a recent Gartner market analysis of cloud services providers, four out five top service providers work with VMware technologies.

- **Hypervisor used :** vSphere™ Hypervisor (ESXi) Version 5.1 (Servers’ operating system).
- **Management:** vCenter™ Server Version 5.1 System management (Provides a centralized layer for the servers).
- **Virtual Datacentres:** vCloud Director Version 1.5 (Provides the API base for controlling the system).



Permanent resources

Wellness Telecom provides 8 dedicated nodes as permanent resources with a total pool of 120 GHz (Intel® Xeon® Processor E5-2640 2.50 GHz 95W 6C/15MB Cache/DDR3 1333MHz). Each node has 8GB of RAM memory (DDR3-1600-MHz RDIMM/PC3-12800/dual rank/1.35v) with a total of 3TB of storage capacity in the Storage Area Network (SAN).

On-request resources

Wellness Telecom is not offering any on-request resources at this moment.

BEFORE YOU START

2.1 Steps To Getting started

To get started with using BonFIRE, please follow these steps:

1. Register an account
2. Basic experiment lifecycle
3. The video tutorial about First hour in BonFIRE
4. BonFIRE general webinar section
5. Tips and best practices for experimenting on BonFIRE
6. Hands on examples
7. Read about how to run experiments in BonFIRE

The following are some important information which should be known at the getting started stage:

- Set up SSH and configure SSH to access VMs through the BonFIRE SSH Gateways
- Upload your SSH key to the Portal
- Read about how to access VMs in BonFIRE
- The VPN for users is an alternative to SSH in order to access the VMs running on the BonFIRE private network (BoNFIRE WAN).
- The BonFIRE Portal may the quickest and easiest starting point to get a feel for running experiments in BonFIRE!

2.2 Registering a BonFIRE Account

You need to apply for an account to use BonFIRE. To apply for an account, go to <http://portal.bonfire-project.eu/register> and fill in the requested fields.

Once your account has been approved (this can take up to 2 business days), you can proceed with the *Setting Up SSH* tutorial to create and/or upload your **SSH** key.

2.3 Setting Up SSH

2.3.1 How to Create an SSH Key

Linux/Mac OSX

Note: You may or may not be familiar with command-line tools and/or SSH. So that's why we adopt a very guided style for that first tutorial.

To complete this tutorial, complete the following tasks:

1. Find your “Terminal” program. Search for it. You will find it.
2. Run your Terminal program. It won’t look like much.
3. In your Terminal program, run `ssh-keygen -t dsa`. You run things in Terminal by just typing their name and hitting RETURN.
4. You’ll be asked several questions about where to store your key, a passphrase, etc. Enter `~/.ssh/id_bonfire` when asked for where to save the key.
5. After you finished answering the questions, the program should have created two new files in `~/.ssh`: the private part of your SSH key (`~/.ssh/id_bonfire`), and the public part (`~/.ssh/id_bonfire.pub`). The private part must remain secret and never live your computer. The public part can be publicly shared (see next point).
6. The next thing to do is to upload the public part of the SSH key to BonFIRE, so that BonFIRE recognize you when you try to access the testbeds. Go to <https://portal.bonfire-project.eu/en/user-details>, you’ll be asked for your BonFIRE credentials. Enter them and then you should see a page with your account details.

Click on “Edit This User Account” and copy/paste the content of the `~/.ssh/id_bonfire.pub` file into the Public Key text area. To display the content of the `~/.ssh/id_bonfire.pub` file, you can use the `cat` command-line tool in the Terminal program:

```
$ cat ~/.ssh/id_bonfire.pub
```

Once you pasted in the content of `~/.ssh/id_bonfire.pub`, click “Submit”.

Here’s me doing the above on my computer in Terminal:

```
crohr@parachute:~ $ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/Users/crohr/.ssh/id_dsa): .ssh/id_bonfire
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in .ssh/id_bonfire.
Your public key has been saved in .ssh/id_bonfire.pub.
The key fingerprint is:
51:f5:59:c2:19:f3:31:ab:6e:f4:cb:25:54:b4:78:0c crohr@parachute.irisa.fr
The key's randomart image is:
---[ DSA 1024]---
|       ...E+=o|
|       . .*B= |
|       . .o=o |
|       .   o.  |
|      S   o.  |
|       o..  |
|       o... |
|       . ..o|
|           o |
```

```
+-----+
crohr@parachute:~ $ ls -al ~/.ssh/id_bonfire*
-rw----- 1 crohr staff 668 14 Jun 14:36 /Users/crohr/.ssh/id_bonfire
-rw-r--r-- 1 crohr staff 614 14 Jun 14:36 /Users/crohr/.ssh/id_bonfire.pub
crohr@parachute:~ $ cat ~/.ssh/id_bonfire.pub
ssh-dss AAAAB3NzaC1kc3MAAACBAM2OThC0Afz9K4C3ymq35FOSoSEtzN3lmwui9801+KYPjL0YdAv/qE2rxsdOh/1lfrCWNHlp
```

Windows

The needed software is PuTTYgen which is used to generate a key pair with PuTTY as follow:

1. Download and start the puttygen.exe generator (download from <http://www.putty.org/>)
2. In the automatically opened window, press Generate.
3. Move your mouse randomly in the small screen in order to generate the key pairs.
4. Enter a key comment, which will identify the key (could be generated by default).
5. Type the key passphrase and confirm it. The passphrase is used to protect your key. You will be asked for it when you connect via PuTTY.
6. Click “Save private key” to save your private key.
7. Click “Save public key” to save your public key, or you can copy/paste the public key which is shown in the window in the Key section.

After having generated the key pair, use the generated public key on BonFIRE tesbed. Go to your account setting and edit your account paste your public key (see the next section), it should look:

```
ssh-rsa AAAAB3NzaC1yc2EAAAQAAIBLFC4jemnEJewqPGhym0SgvHDyj4OSTcAwnLGgxZYnj9YmZ3HfbUcoXy
    KDJuZTj5Td1MRfkje6jXcMj/iBo3AxttSOXZRxUQTv/bEFtPE9hc+jA0mgf0JLtw3cI1wkX9ZL0vW04PjHrVQG
    tBLXha5BUZt2i8zTYR4MGE0vGmWHQ== rsa-key-20110622
```

3. If everything went well, you should now be connected to the gateway. Make Normally, it takes time till the public key gets distributed on BonFIRE testbed.

2.3.2 How to access a BonFIRE SSH Gateway

You'll now learn how to use the SSH key you just created to connect to one of the *SSH Gateway Configuration*.

It is from one of those machines that you'll be able to access compute resources located at one of the BonFIRE testbeds, using SSH.

Note: If you just uploaded your key to your account, you may have to wait a few minutes before you can connect to the SSH gateway.

MacOS/Linux

1. Open your Terminal program.
2. Run the following command (replace BONFIRE_USER with your BonFIRE username):

```
$ ssh -i ~/.ssh/id_bonfire BONFIRE_USER@ssh.bonfire.grid5000.fr
```

3. If everything went well, you should now be connected to the gateway. Make sure it is the case by running the hostname command:

```
$ hostname
```

Here's me doing the above on my computer in Terminal:

```
crohr@parachute:~ $ ssh -i ~/.ssh/id_bonfire crohr@ssh.bonfire.grid5000.fr
Warning: Permanently added 'ssh.bonfire.grid5000.fr,131.254.204.5' (RSA) to the list of known hosts.
Last login: Wed Jun  8 11:17:43 2011 from parachute.irisa.fr
[crohr@ssh ~]$ hostname
ssh.bonfire.grid5000.fr
```

Windows

Basic setup

For windows user, PuTTY is suitable software to get access to the BonFIRE gateways and BonFIRE resources.

The needed softwares is PuTTY client. You can download putty.exe from <http://www.putty.org/>

The following shows how to use an authenticated access to BonFIRE machines through PuTTY client:

1. Open PuTTY client and set the Host Name field with any of the gateways (list of BonFIRE SSH gateways) in the “Session” page
2. On the “Connection/Data” page you can (optionally) put your BonFIRE username as the “Auto-login” username
3. On the “Connection/SSH/Auth” page, tick the “Attempt authentication using Pageant”, and the “Allow agent forwarding” options, after that browse the generated private key file and upload the private key
4. Make sure you are running pageant (the Putty ssh agent) and add your private ssh key (pageant.exe downloadable from <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>)
5. Click Open
6. Open the session to the proxy - it should automatically do the username and authenticate with your key.

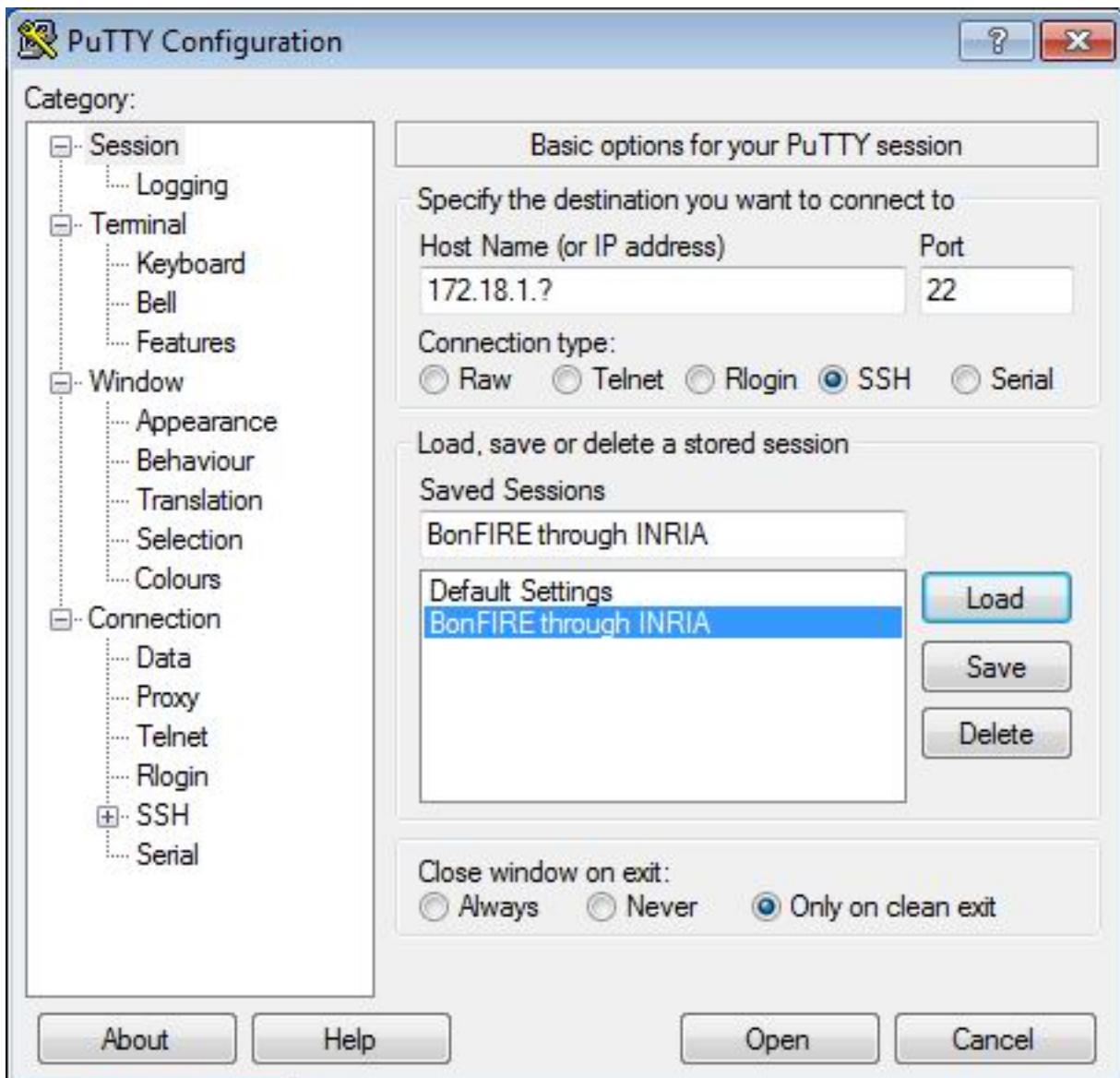
After logging into one of the gateways, you can access to any BonFIRE machine through ssh linux command and the connection to the VM will also be authenticated using your key (because the agent forwarding option is used):

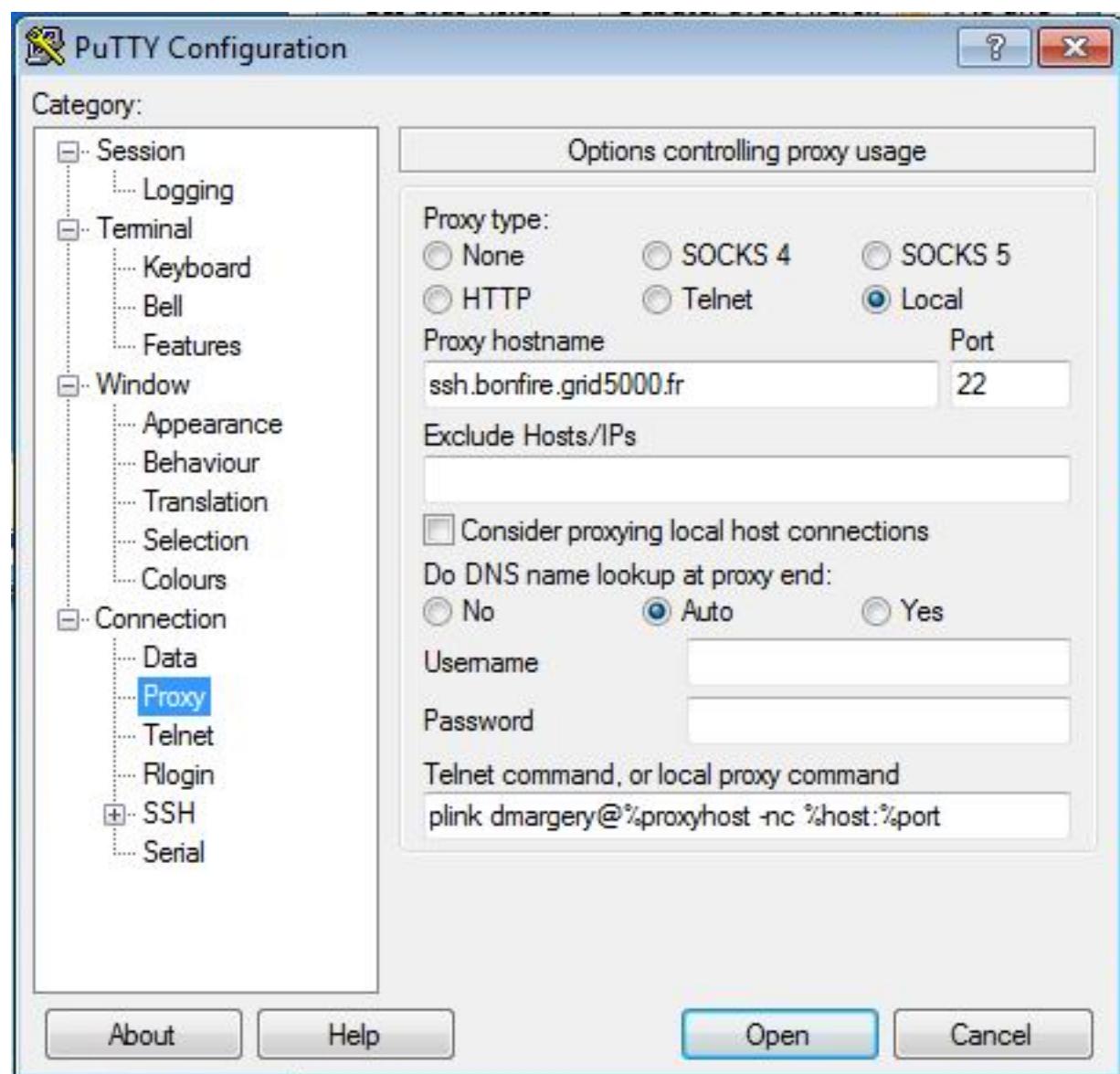
```
ssh root@<ip address of any machine>
```

Advanced setup

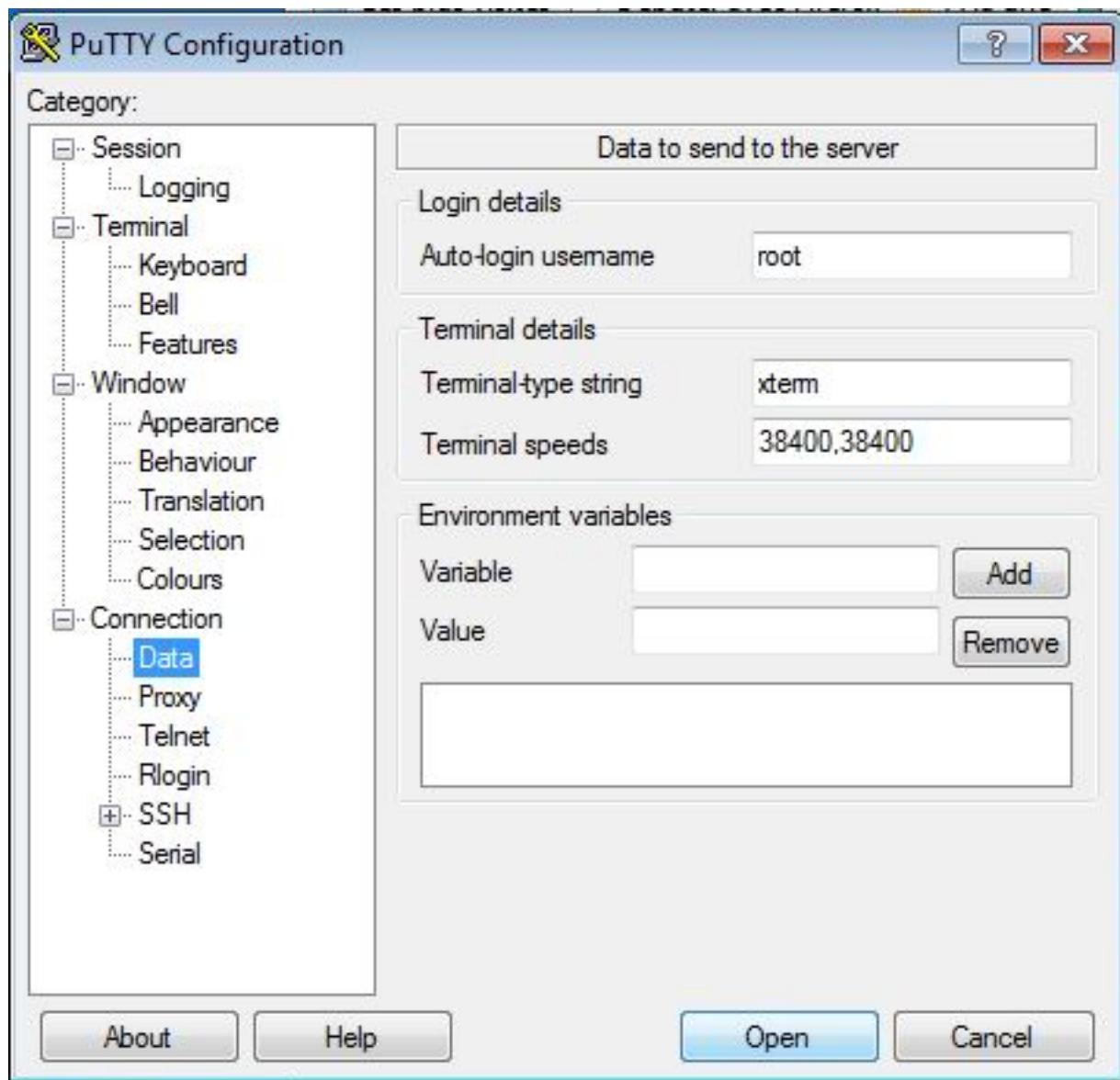
This setup aims at giving a one click access to VMs, presume that you are using pageant (pageant.exe: downloadable from:<http://www.putty.org/>) to store your keys. This setup is illustrated below:

1. For this you need to create a stored session. In the opening screen you should
 - create a session named BonFIRE
 - put a fake IP starting with the BonFIRE WAN prefix: 172.18.?.? and specify you will be using an ssh connection
2. Fill in the proxy information, in the connection field
 - specify local proxy
 - specify the name of the ssh gateway you will be using to connect to BonFIRE (in this example ssh.bonfire.grid5000.fr)
 - local proxy command: plink <bonfire-login>@%proxyhost -nc %host:%port
3. Enter your username on the destination VM





- auto-login username should be set to root.



4. Save your session
5. Each time you need to connect to one of your VMs, change the IP in the startup script after having loaded your session, and you are done!

2.4 Uploading SSH Key to BonFIRE

1. Log in to the BonFIRE Portal
2. Go to the icon on the up-right of the page (the little person): account settings and click on it
3. Click on Add Public Key

The screenshot shows the BonFIRE homepage. At the top, there is a dark header bar with icons for a folder, a test tube, and a document on the left, and user information ('yliang') on the right. The main content area has a light gray background. It starts with the text "Hello yliang!" in orange. Below it is a message in black: "Please choose your destination: There are Resources and of course Experiments — the two main items, which can be accessed via the icons top left. If you have any questions, please consult the FAQ. We are always open for a dialogue and hungry for feedback, just Contact us." A link "Further documentation on how to use the BonFIRE services is available here." is also present. At the bottom of the content area, there is a section titled "We've got News" with a single item: "We've got news. Right on the front page. You might have noticed already."

The screenshot shows the "User Details" page for the user "yliang". The top navigation bar includes icons for a folder, a test tube, and a document. The main title is "User Details - yliang". Below the title, the user's details are listed in a table-like format:

UID	yliang
Full Name	Yongzheng Liang
Email Address	liang@rus.uni-stuttgart.de
Numeric UID	2017
Login Shell	/bin/sh
Home Directory	/home/yliang

At the bottom of the page are three green buttons with white text:

- ▶ edit this user account
- ▶ set password
- ✚ Add Public Key

- Add your public key as seen in the following picture, and submit it

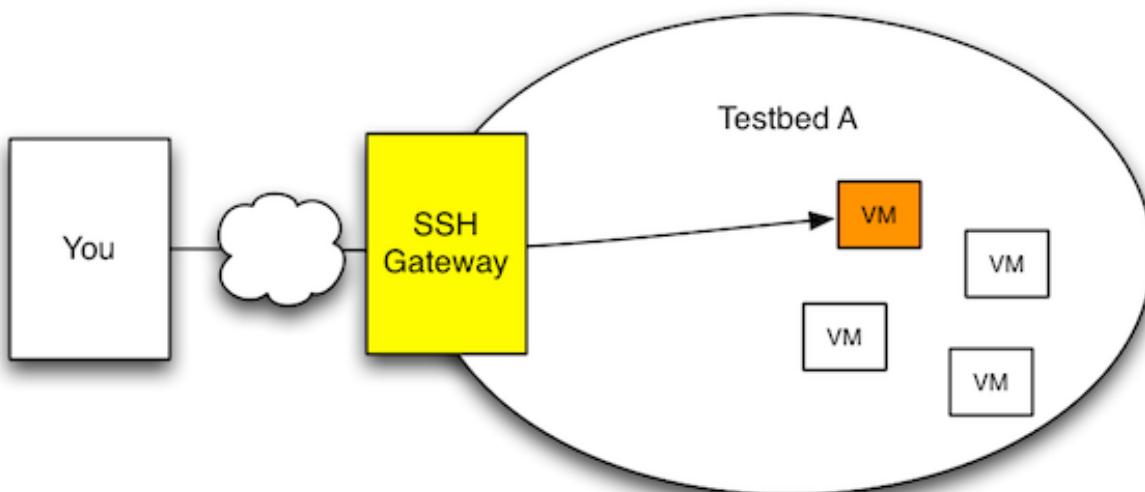
Please enter your Public Key ×

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABJQAAIBLFC4jemnEJewqPGhym0SgvHDyj40STcAwnLGgxZYnj9YmZ3HfbUcoXykDJuzTj5TdIMRfkyc6jXcmj/iBo
3AxttSOXZRxUQTv/bEFTPE9hc+jA0mgf0JLtw3cllwkX9ZLqvW04PjHrVQG+tlBLxha5BUZt2i8zTYR4MGE0vGmWHQ== rsa-key-20110622
```

Abort Submit

2.5 SSH Gateway Configuration

Only a handful of BonFIRE testbeds have public IPs available for VMs. Therefore most of the time you will deploy VMs (for instance in the BonFIRE WAN) that are not publicly accessible.



SSH gateways are used to access the VMs located within the BonFIRE WAN. That means you have to go through an additional hop to access your VMs via SSH:

```
$ ssh -A -i ~/.ssh/id_bonfire ssh.fr-inria.bonfire-project.eu
Last login: Wed Jul  6 10:25:00 2011 from 131.254.15.115
[crohr@ssh ~]$
$ ssh root@172.18.2.136
```

2.5.1 List of BonFIRE SSH gateways

- EPCC (uk-epcc)

```
ssh.uk-epcc.bonfire-project.eu
```

- HLRS (de-hlrs)

Not available.

- HP Labs (uk-hplabs)

```
ssh.uk-hplabs.bonfire-project.eu
```

- IBBT (be-ibbt)

```
ssh.be-ibbt.bonfire-project.eu
```

- INRIA (fr-inria)

```
ssh.fr-inria.bonfire-project.eu
```

2.5.2 Putting some Magic into your SSH Configuration

When the machines you want to access are hidden behind a gateway, as is the case with most of the BonFIRE resources, SSHing to them can prove a bit cumbersome: first you have to SSH into the gateway as you (with agent forwarding enabled), then ssh to the machine as `root`. Below you'll learn how to SSH into your machines in one command:

1. Locate your SSH configuration file. On Linux/MaxOS it is located at `~/.ssh/config`. If it does not exist, create it and append the following lines to it (replace `BONFIRE_USER` with your BonFIRE username):

```
# BonFIRE SSH Gateway
Host bonfire-gateway
    Hostname ssh.fr-inria.bonfire-project.eu
    User BONFIRE_USER
    ForwardAgent yes
# BonFIRE WAN IPs:
Host 172.18./*
    User root
    ProxyCommand ssh bonfire-gateway "nc %h %p"
    StrictHostKeyChecking no
# BonFIRE Public IPs at INRIA
Host 131.254./*
    User root
    ProxyCommand ssh bonfire-gateway "nc %h %p"
    StrictHostKeyChecking no
```

The following is an example regarding each testbed site.

```
#VMs at INRIA
Host 172.18.1.* 172.18.7.* 172.18.248.* 172.18.249.* 172.18.250.* 172.18.251.* 172.18.252./*
    ProxyCommand ssh [login@]ssh.fr-inria.bonfire-project.eu "nc %h %p"
    IdentityFile ~/.ssh/id_bonfire
    StrictHostKeyChecking no
    HashKnownHosts no
    User root

#VMs at HLRS
Host 172.18.2.*
```

```
ProxyCommand ssh [login@]ssh.fr-inria.bonfire-project.eu "nc %h %p"
IdentityFile ~/.ssh/id_bonfire
StrictHostKeyChecking no
HashKnownHosts no
User root

#VMs at EPCC
Host 172.18.3.* 172.18.240.* 172.18.241.* 172.18.242.* 172.18.243.* 172.18.244./*
ProxyCommand ssh [login@]ssh.uk-epcc.bonfire-project.eu "nc %h %p"
IdentityFile ~/.ssh/id_bonfire
StrictHostKeyChecking no
HashKnownHosts no
User root

#Hosts at IBBT
Host 172.18.4./*
ProxyCommand ssh [login@]ssh.be-ibbt.bonfire-project.eu "nc %h %p"
IdentityFile ~/.ssh/id_bonfire
StrictHostKeyChecking no
HashKnownHosts no
User root

#VMs at PSNC
Host 172.18.8.* 172.18.9./*
ProxyCommand ssh [login@]ssh.uk-epcc.bonfire-project.eu "nc %h %p"
IdentityFile ~/.ssh/id_bonfire
StrictHostKeyChecking no
HashKnownHosts no
User root

#VMs at HP
Host 172.18.5.* 10.25.* 10.26./*
ProxyCommand ssh [login@]ssh.uk-hplabs.bonfire-project.eu "nc %h %p"
IdentityFile ~/.ssh/id_bonfire
StrictHostKeyChecking no
HashKnownHosts no
User root

#VMs at WT
Host 172.18.232./*
ProxyCommand ssh [login@]ssh.uk-epcc.bonfire-project.eu "nc %h %p"
IdentityFile ~/.ssh/id_bonfire
StrictHostKeyChecking no
HashKnownHosts no
User root
```

2. You can now access compute resources directly by typing their IP from your machine.

\$ ssh 172.18.2.136

Note: This only works for compute resources connected to one of the predefined networks, i.e. “BonFIRE WAN” or “Public Network”.

2.6 Getting Support

In addition to the availability of this user documentation, there are two other ways of getting support in BonFIRE:

Forum: <https://forum.bonfire-project.eu/>

Email: support@bonfire-project.eu

The forum is a good support channel for discussing how to best do something that the online documentation does not cover. You may find that your questions have already been addressed in existing threads, so please do a search for your question before creating a new thread. Feel free to support fellow community members if you know the answers.

If email support is used, please specify the problem in as much detail as possible. A support ticket will be opened for you and a person will be assigned to assisting you.

RUNNING EXPERIMENTS IN BONFIRE

3.1 Introduction

We consider an experiment in BonFIRE to be more than just the deployment and execution of Cloud resources (compute, storage and network). We see the lifecycle as 8 phases, starting with the design of the experiment and ending in publication of results. The complete lifecycle is described in more detail below in [The BonFIRE Experiment Lifecycle](#).

To get started with the notion of running an experiment in BonFIRE, see [A General Workflow For Conducting An Experiment](#) below. Running an experiment in BonFIRE is very straight-forward, even if it is federated across multiple testbeds. BonFIRE provides single sign-on using SSH to any VMs on the different testbeds, and the deployment of resources via the OCCI-based API is largely identical for all testbeds. There are some differences in the capabilities that the different testbeds offer and policies for accessing them, which you can read more about on the [Using the BonFIRE Testbeds](#) page.

When you get accepted to use BonFIRE, you will have a quota on the usage of the resources, and the access to different resources on different testbeds may be constrained due to finite resources available in the project. To find out more about this and how to view your usage, please see the [Quota and Usage](#) page.

For more technical details about how to create an experiment in BonFIRE, see the [Creating Experiments](#) and [Tutorials](#) pages.

3.1.1 The BonFIRE Experiment Lifecycle

In BonFIRE, we consider experiments as being comprised of eight phases, which are shown in [Figure 1](#).

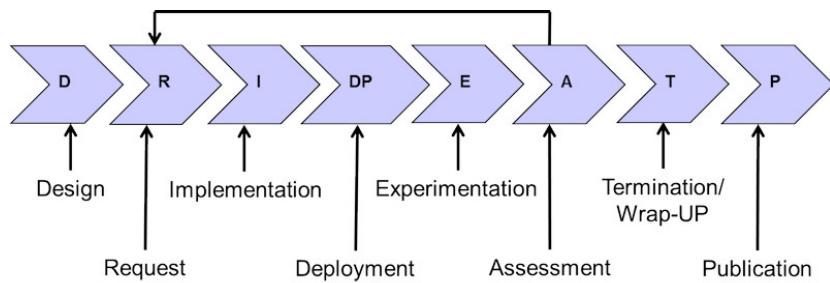


Figure 3.1: [Figure 1: The BonFIRE Experiment lifecycle](#)

Phase 1: Design

In the first phase, the experiment is defined and designed with an analysis of which BonFIRE resources will be required.

Phase 2: Request

Within this phase, the experimenters interact with BonFIRE by submitting the experiment design together with an application for resources. In this part of the lifecycle, the BonFIRE service will take over the task of analyzing the needs of the experiment and planning the scheduling of the experiment on the BonFIRE testbeds. This may also include a reservation of resources either on the testbed side or by the BonFIRE service (see [Figure 2](#)).

Phase 3: Implementation

Here, the experiment is implemented and all the necessary actions are taken to allow later deployment within the BonFIRE system.

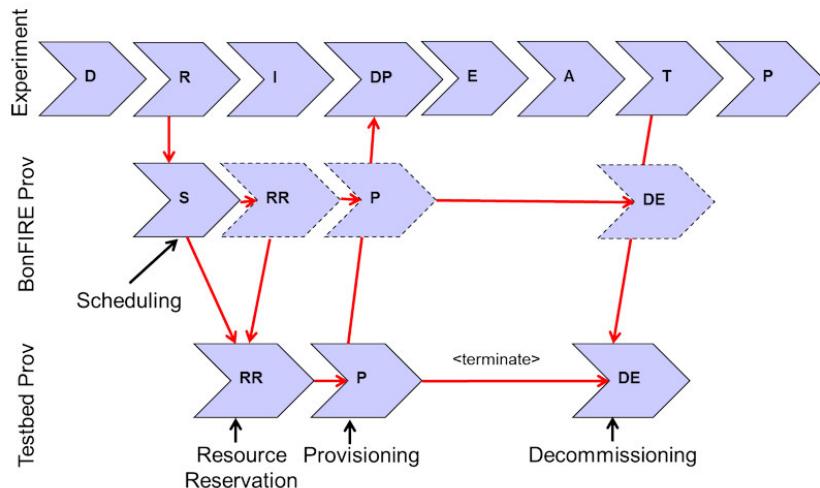


Figure 3.2: *Figure 2: The Experiment lifecycle and dependencies to BonFIRE and the Testbed Provider*

Phase 4: Deployment

In this phase, the experiment is deployed in BonFIRE. This implies that in response to the request, the testbeds and/or BonFIRE Provider have provided all necessary resources to execute the experiment.

Phase 5: Experimentation

This covers the execution of the experiment.

Phase 6: Assessment

After an experiment has run, it can be assessed on basis of its results. At this point, the experiment lifecycle foresees a loop to allow for fine-tuning of the experiment and another run (including the request of resources of BonFIRE, etc).

Phase 7: Termination/Wrap-Up

The termination of an experiment can occur in two different situations:

1. “natural” termination. This is the case when the experiment has been performed as planned, all the results are available and the experimenters are ready for a wrap-up.
2. “forced” termination. This happens when the BonFIRE service or the testbed terminates the experiment before the experiment is fully executed. A forced termination may occur if a termination condition is reached such as over-consumption of the requested resources or misbehaviour of the experimenters

In both cases, decommissioning needs to take place either by the BonFIRE service or by the testbed, in order to release the assigned resources. Experiment results may be transferred to persistent storage.

Phase 8: Publication

In the final phase of an experiment, the results are published.

3.1.2 A General Workflow For Conducting An Experiment

This section considers the general steps to conduct a basic experiment, but does not go into details of setting up things like network emulation on the Virtual Wall. The following steps is merely a guide to help clarify the process you need to go through to perform typical actions in an experiment in BonFIRE.

1. Register with BonFIRE and set up SSH
 - See the [Steps To Getting started](#) page.
2. Connect to BonFIRE
 - Log onto the BonFIRE portal using the userID (<https://portal.bonfire-project.eu/>), or
 - Log onto the BonFIRE Broker API using the userID via other [client tools](#)
3. Set-up and create VM images for an experiment
 - Create an experiment container
 - Select [instance type](#)
 - Deploy [base image](#) (from each testbed that is used)
 - Install and configure software (and set up monitoring agents) (via SSH)
 - Save image
 - Terminate VM
4. Execute experiment
 - If monitoring is used, need to deploy Aggregator first to get its IP address (used for contextualisation)
 - Set up and deploy any [storage](#) resources needed
 - Set up and deploy any [network](#) resources needed (e.g., network emulation at the Virtual Wall)
 - **Set up and deploy [compute](#) resources**

- (a) Set *context*
 - (b) Set *instance type*
 - (c) Set *VM image*
- Configure/ start/stop services, etc (via SSH)
 - Change experiment conditions during run-time, such as adjusting network parameters, increasing load on VMs, etc
5. Terminate / wrap up experiment
 - Get/download the data (output data from VMs and *monitoring data* from Zabbix Aggregator)
 - Terminate VMs

3.2 Using the BonFIRE Testbeds

The lowest layer of the *BonFIRE Architecture* are the testbeds. There are several testbeds in BonFIRE with different *Infrastructure* offerings. This is not only in the form of heterogeneous resources, but there are some differences in the services, functionality and some minor differences in how to access some of the testbeds.

3.2.1 The Different Testbeds

The testbeds are the Cloud infrastructures that provide compute, network and storage resources that can be used by BonFIRE experiments. BonFIRE has four types of testbeds provided by project partners:

- four [OpenNebula](#) infrastructures operated by EPCC, HLRS, Inria and PSNC;
- the Cells cloud infrastructure provided by [HP](#); and
- the Virtual Wall network emulation infrastructure provided by [iMinds](#).
- a VMware cloud infrastructure provided by Wellness Telecom at WT.

Additionally, BonFIRE will soon connect to external facilities such third-party cloud solutions like [Amazon EC2](#), as well other FIRE and networking services such as [FEDERICA](#) and GEANT [AutoBAHN](#).

Testbeds may expose their functionality via any API they choose, but to simplify the initial implementation all the embedded testbeds expose their basic resource manipulation functionality using a version of the Open Cloud Computing Interface ([OCCI](#)). The external facilities all expose their functionality via other, proprietary, interfaces. However because of different semantics or working model in each underlying system, some [OCCI deviations](#) between them for implementation need to be taken into account by experimenters in describing their experiments.

3.2.2 Features Supported By Each Testbed Site

Table 3.1: List of features on each site

Features	EPCC	Inria	HLRS	PSNC	HP	iMinds	WT
Custom instance types	x	x	x				
Controlled deployment on physical hosts	x	x	x	x			
Exclusive access to physical hosts		x	x	x			
Access to VM logs	x	x	x	x			
Access to status logs	x	x	x	x			
Public IPs	x	x		x	x	limited	
In-depth VM monitoring	x	x	x	x	x	x	x
Infrastructure monitoring	x	x	x	x			x
User-specified application monitoring	x	x	x	x	x	x	x
Controlled network topologies					x	x	
Network background traffic						x	
Network impairments	x	x	x	x	x	x	x
<i>Bandwidth on Demand</i>	x			x			
Controlled network slices with <i>FEDERICA</i>	x			x			
<i>Elasticity engine</i>	x	x	x	x	x	x	
<i>Shared storage</i>						x	

3.2.3 Differences Between Testbeds

Although BonFIRE offers a uniform way in which to interact with the different testbeds, there are a few differences in features and functionality offered by the different testbeds:

- Each testbed offers heterogeneous resources. Also, within some testbeds, there are heterogeneous resources (USTUTT, Inria on-request and HP).
- Each testbed supports different *Instance Types* that specify the CPU and RAM of the compute resources.
- To run an experiment on the Virtual Wall (iMinds), a **GO** command needs to be issued to deploy the experiment resources. Once deployed, more cannot be added. For all other testbeds, this is not the case.
- Public IP addresses are limited at all testbeds, with the exception of Inria and HP. EPCC and PSNC offer a small set of Public IPs and the other testbeds do not offer any.
- VMs with a public IP at HP can be accessed on ports 80, 443 and 22. It is also possible to connect via a BonFIRE SSH Gateway using the *BonFIRE WAN*.
- Private subnets possible at HP, EPCC and PSNC.
- Advanced network emulation only available at iMinds (control of network topologies, network impairments and background traffic).
- Shared (NFS) storage only available at the Virtual Wall (but can be used in different experiments and accessed by VMs on different testbeds).
- Persisted storage resources are not available at HP.
- On-request (compute) resources are only available at Inria, HP, USTUTT and iMinds). Please note that the process for using the on-request resources is different. See *On Request Resources* for more information.
- Dynamic reservation of on-request resources only available at Inria. On other testbeds, experimenters need to make a request to BonFIRE for the testbed administrators to manually make available additional resources.
- The BonFIRE WAN at HP does not provide access to Internet. A typical solution to install software in a VM is to create an instance with a public IP, install all software needed and then use the SAVE_AS to store the image. Then the experimenter can boot from that image any number of instances needed.

- Private subnets are not possible at WT.
- Update storage parameters (OS) at WT: these types of disk are attached to a virtual machine and cannot be disaggregated. This means that the only way to update them is by updating the entire virtual machine.
- Share storage (OS) at WT: System storage is associated to a template and it is impossible to extract it. Furthermore, it does not have an URL and the only way to contact it is by throwing the virtual machine. This is a limitation of vCloud Director.
- WT does not support DATABLOCKS. Therefore only OS disks can be used as storage. In this way, the maximum storage possible on VMs is 10GB and on maximum storage possible on Aggregators is 6GB
- It is important to point out that aggregator data at WT are lost at the end of an experiment. Hence, the user must perform a SAVE_AS command during the experiment lifetime in order to keep data.
- Because of VMware limitations, the customization scripts of the virtual machines created at Wellness Telecom have a fixed size. Therefore, the number of SSH public keys that can be transferred to a VM is limited to 70 keys. This number includes all the public keys stored in user's profiles belonging to the group of the virtual machine.
- When an user performs a 'Save As' command at WT over a VM (Name: 'MACHINE_1' / Group: 'GROUP_1') and shares it with the group 'GROUP_2', the resultant VM will belong to both groups ('GROUP_1' and 'GROUP_2'). This behaviour is different from other testbeds where the resultant VM would only belong to 'GROUP_2'.
- When a VM is deployed at WT it shows the STOPPED state until it gets running.

3.2.4 Creating Resources on the BonFIRE Testbeds

A resource, be it storage, network, or compute, is created on a specific location (testbed). To discover the list of available testbeds, you can use the API via different Client Tools. Below are examples of listing the available testbeds and services using OCCI. This can also be viewed on the *BonFIRE Portal*.

Listing Available Testbeds

Listing available testbeds is easily achieved with a simple GET request:

```
$ curl -kni https://api.bonfire-project.eu/locations

HTTP/1.1 200 OK
Date: Thu, 08 Dec 2011 13:32:25 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=120
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "6cdf753edb840409981b901908ffb97f"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.022719
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations">
  <items offset="0" total="4">
    <location href="/locations/be-ibbt">
      <name>be-ibbt</name>
      <url>https://bonfire.test.atlantis.ugent.be</url>
      <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
      <link rel="computes" href="/locations/be-ibbt/computes" type="application/vnd.bonfire+xml"/>
```

```

<link rel="networks" href="/locations/be-ibbt/networks" type="application/vnd.bonfire+xml"/>
<link rel="storages" href="/locations/be-ibbt/storages" type="application/vnd.bonfire+xml"/>
<link rel="configurations" href="/locations/be-ibbt/configurations" type="application/vnd.bonfire+xml"/>
<link rel="services" href="/locations/be-ibbt/services" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/fr-inria">
  <name>fr-inria</name>
  <url>https://api.bonfire-project.eu/</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/fr-inria/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/fr-inria/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/fr-inria/storages" type="application/vnd.bonfire+xml"/>
  <link rel="configurations" href="/locations/fr-inria/configurations" type="application/vnd.bonfire+xml"/>
  <link rel="services" href="/locations/fr-inria/services" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/uk-hplabs">
  <name>uk-hplabs</name>
  <url>https://occisvr-vif0.occi.ext8.bonfire.hpl.hp.com:443</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/uk-hplabs/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/uk-hplabs/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/uk-hplabs/storages" type="application/vnd.bonfire+xml"/>
  <link rel="configurations" href="/locations/uk-hplabs/configurations" type="application/vnd.bonfire+xml"/>
  <link rel="services" href="/locations/uk-hplabs/services" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/uk-uedin">
  <name>uk-uedin</name>
  <url>https://bonfire.epcc.ed.ac.uk:8443</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/uk-uedin/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/uk-uedin/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/uk-uedin/storages" type="application/vnd.bonfire+xml"/>
  <link rel="configurations" href="/locations/uk-uedin/configurations" type="application/vnd.bonfire+xml"/>
  <link rel="services" href="/locations/uk-uedin/services" type="application/vnd.bonfire+xml"/>
</location>
</items>
<link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

Listing Services Available on Each Testbed

Every testbed maintains services for the user. For instance, most of the testbeds provide a machine that acts as an SSH Gateway (see [SSH Gateway Configuration](#)). Others also provide an aggregator machine that monitors the physical resources on which your virtual resources will be run. You can easily list all these services, and know their IP, using the API:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/services
```

```

HTTP/1.1 200 OK
Date: Thu, 08 Dec 2011 13:35:43 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=3600
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "c8a39531f389008f56b0b1a3c4aef9d0"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.021710
Vary: Authorization,Accept
```

```
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/services">
  <items offset="0" total="1">
    <service href="/locations/fr-inria/services/1">
      <name>aggregator</name>
      <ip>131.254.204.19</ip>
      <link rel="parent" href="/locations/fr-inria" type="application/vnd.bonfire+xml"/>
    </service>
  </items>
  <link href="/locations/fr-inria" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

Note: The ip element can be an IPv4 or IPv6 address, as well as a plain hostname.

3.3 Quota and Usage

On all the testbeds, you can see what is your current quota usage and limitations by issuing a request to the ‘account’ resource. Here is how you would do it, for the IBBT testbed for instance:

```
crohr@parachute:~ $ curl -kni https://api.bonfire-project.eu/locations/be-ibbt/account

HTTP/1.1 200 OK
Date: Thu, 01 Mar 2012 15:42:25 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "e53e922c85f002c7b549227f35d36e67"
Cache-Control: max-age=0, private, must-revalidate
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.933386
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<account xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/be-ibbt/account">
  <id>10</id>
  <name>crohr</name>
  <quota>
    <storage>409600</storage>
  </quota>
  <usage>
    <storage>0</storage>
  </usage>
  <link rel="parent" href="/locations/be-ibbt" type="application/vnd.bonfire+xml"/>
  <link rel="self" href="/locations/be-ibbt/account" type="application/vnd.bonfire+xml"/>
</account>
```

Unit is XXX.

3.4 Creating Experiments

To do an experiment in BonFIRE and be able to use the three different types of resources (*compute*, *storage* and *network*), it is necessary to create an experiment first. Any resources will be added to an experiment instance, with the exception of *persistent* and *shared* storage resources, and any VM images that are saved.

Each of the resource pages linked to above describe with examples how to deploy and manage the different resources in an experiment. There are also several *tutorials* available, but first let's see how to create an experiment container.

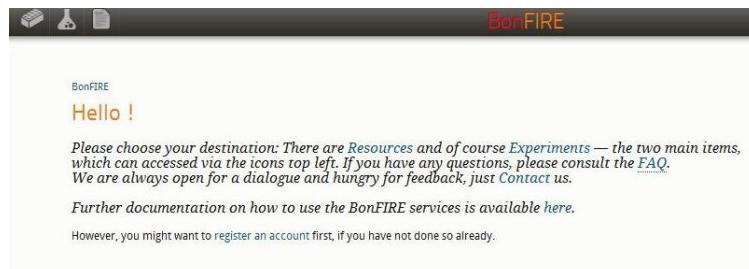
An experiment is defined by a name, a description and a walltime. The walltime is the duration of the experiment, after which all the experiment resources (compute, storage and network) created within the experiment are destroyed. The walltime is defined in seconds for all client tools except for the Portal, which expects a value in minutes (it then calculates the value in seconds when it creates the OCCI message to send to the BonFIRE API).

The subsections below illustrate how to create an experiment with the different client tools available in BonFIRE!

3.4.1 Portal

Assuming that you have logged onto the BonFIRE Portal at <https://portal.bonfire-project.eu/>, this is how you create an experiment.

1. press the button for Experiments in the top left corner:



2. press the button Add Experiment:



3. enter information for the experiment, and press Continue:

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Create Experiment

Name ?
Tutorial

Description ?
used for tutorial

Walltime (Lifetime in Minutes) ?
120

Add a Monitoring Aggregator at this Site ?
don't create an aggregator

Add Infrastructure Metrics ?

→ Continue ► Finish

4. review the details of the experiment data in OCCI format, and click the button **Finish** to submit it:

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Review OCCI Request for Experiment Creation

```
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Tutorial</name>
  <description>used for tutorial</description>
  <walltime>7200</walltime>
  <status>ready</status>
</experiment>
```

Ignore XML Validation Errors

→ Continue ▶ Finish

You should then be forwarded to another screen where you can add resources to your experiment.

3.4.2 Experiment Descriptor

There are several options for creating an experiment with the experiment descriptors.

- 1: As a **Managed Experiment** via the *BonFIRE Portal*.
- 2: On the command line with a *JSON* descriptor.
- 3: On the command line with an *OVF* descriptor.

It does not make sense to just create an experiment with an experiment descriptor, as the purpose is to define the complete initial deployment. Therefore, please refer to the pages above for more information with code examples.

3.4.3 OCCI

Assuming that you have *installed and set up curl* to post OCCI HTTP messages to the BonFIRE API, this is how you create an experiment.

```
$ curl -k -i https://api.bonfire-project.eu/experiments -u BONFIRE_USER \
-H 'Content-Type: application/vnd.bonfire+xml' -X POST -d \
'<?xml version="1.0" encoding="UTF-8"?>
```

```
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>My First Experiment using cURL</name>
  <description>Experiment description</description>
  <walltime>3600</walltime>
</experiment>'
```

3.4.4 Command Line Tool

Assuming you have *installed and set up the command-line tools*, you can use the `bfexperiment` tool to query and manipulate existing experiments as well as to create new ones. Its usage is as follows:

```
# bfexperiment create <name> [-D <description>] [-W <walltime>] [-G <group> ...]
```

The following positional argument must be specified when creating experiments:

- `<name>` - The name for this experiment.

The following options are applicable when creating experiments:

- `-D <description>, --description <description>` - description of the experiment. Empty by default.
- `-W <walltime>, --walltime <walltime>` - lifetime of the experiment in seconds. Defaults to one day.
- `-G <group>, --group <group>` - A user group this experiment will be accessible by. This option can be specified multiple times.

For example:

```
# bfexperiment create "My First Experiment using CMD Tool" -D "Experiment description" -W 3600
```

3.4.5 Restfully

Assuming you have *installed and configured Restfully*, you first need to start the interactive Restfully shell (replace LOGIN and PASSWORD with your BonFIRE credentials, see the *Restfully FAQ* for an alternative way of passing parameters):

```
$ restfully https://api.bonfire-project.eu/ -u LOGIN -p PASSWORD -r ApplicationVndBonfireXml
```

You should get back a prompt like the following:

```
Restfully/0.8.1 - The root resource is available in the 'root' variable.  
ruby-1.8.7-p249 >
```

You now have access to the `root` of the BonFIRE API, which you need to use to issue Restfully commands to BonFIRE. To create an experiment, type the following:

```
root.experiments.submit(:name => "My First Experiment using Restfully", :description => "Experiment description")
```

To create experiments and resources in Restfully it may be quicker to do this in a Ruby script. For more information about how to do this, please click [here](#).

3.5 Tutorials and Experiment Scenarios

This section includes some tutorials for things you are likely to do when you conduct experiments in BonFIRE, as well as some scenarios for using the different testbeds (OpenNebula, Cells and Virtual Wall).

3.5.1 Access Virtual Machines in BonFIRE

Because of lack of public IPv4 addresses, most of the BonFIRE testbeds cannot provide a public IPv4 address for each virtual machine running on their infrastructure. These virtual machines are assigned with BonFIRE addresses which are private IPv4 addresses, and they are uniquely defined in BonFIRE. The different BonFIRE testbed sites are interconnected via BonFIRE WAN.

Secure Shell (SSH) is a network protocol for secure date communication, remote shell services or command execution and other secure network services between two networked computers that it connects via a secure channel over an insecure network http://en.wikipedia.org/wiki/Secure_Shell. A SSH Gateway is an object that provides a tunneled network connection through a publicly visible host. To access a host hidden behind the public one is by first logging into the publicly visible host, and from thence logging into the hidden ones.

The VMs running on four BonFIRE testbed sites can be reached via a SSH gateway. They are EPCC, HLRS, INRIA, and IBBT. HP does not provide an SSH gateway, all VMs created there have a public IP.

The picture below shows the gateways located in the BonFIRE infrastructure.

Access a VM via a SSH gateways

For accessing a virtual machine via a ssh gateway for the first time, the following steps should be taken:

- create SSH key
- upload the public key onto the BonFIRE Portal
- access a SSH gateway
- access a virtual machine from the SSH gateway

How to create SSH key

SSH uses public-key to authenticate a remote computer. You can find here how to create SSH key

How to upload a public key onto the BonFIRE Portal

On this page you can see how to upload SSH key .

BonFIRE SSH gateways

- EPCC (uk-epcc)

ssh.uk-epcc.bonfire-project.eu
129.215.62.205

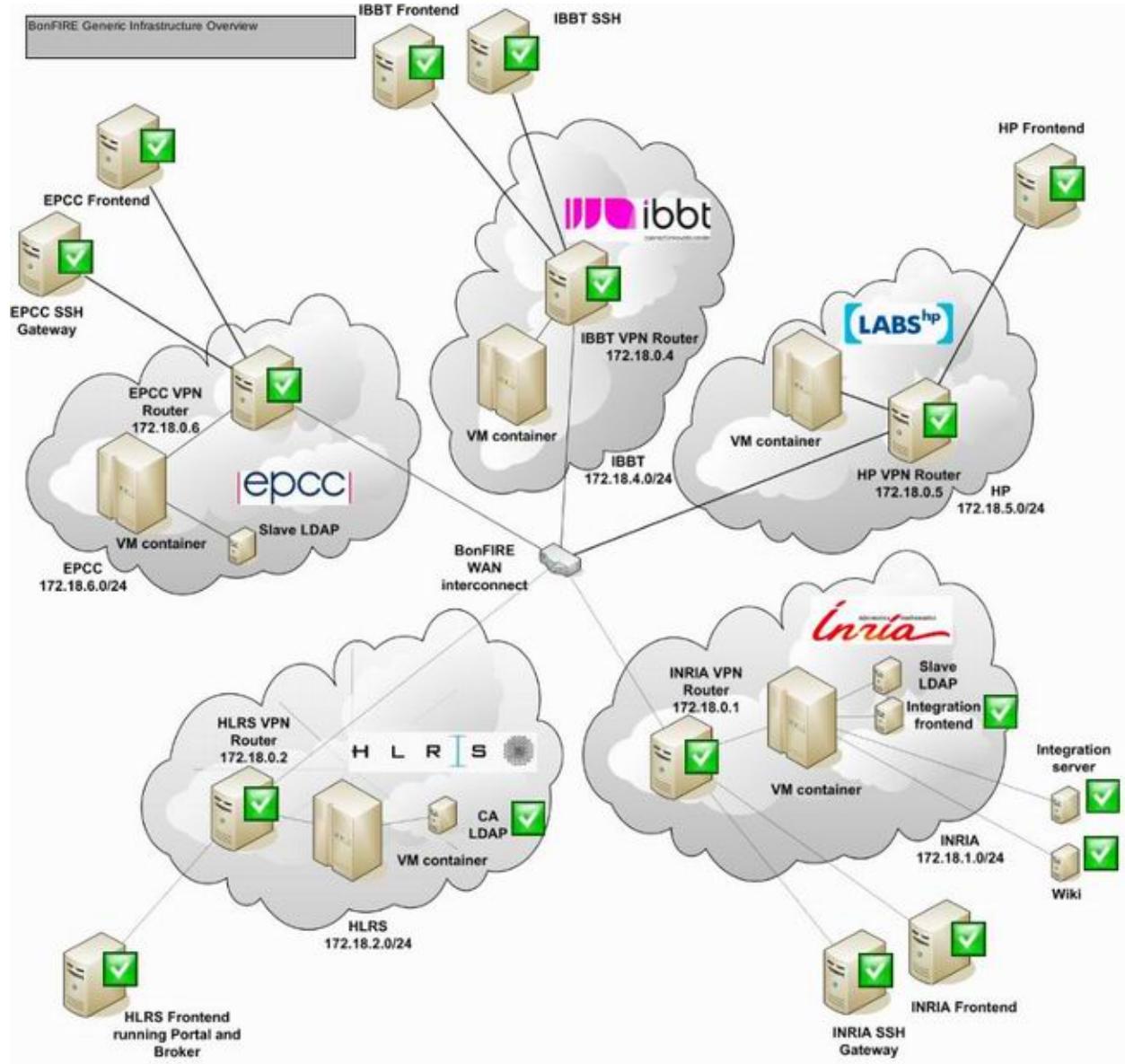
- INRIA (fr-inria)

ssh.fr-inria.bonfire-project.eu
131.254.204.5

- IBBT (be-ibbt)

ssh.be-ibbt.bonfire-project.eu
193.191.148.155

- HLRS (de-hlrs)



HLRS does not have its own gateway. The INRIA gateway can be used for accessing VMs running on HLRS.

Access a BonFIRE gateway

A Linux user could have a look at putting Magic into SSH Configuration and access compute resource on Linux

A windows user could have a look at access compute resource on Windows

Access a virtual machine from a SSH gateway

The virtual machine can be reached now directly by typing its IP, e.g.

```
$ ssh 172.18.2.136
```

Access a VM running on HP

The HP site does not require an SSH gateway since all VMs created there may be requested to have public ips. Once the public key has been uploaded onto the BonFIRE portal, a VM with a public ip will be reachable from the internet.

The HP site allows a user to request VMs with a public ip or an internal ip. Only VMs with a public ip can be reached from the internet. If several VMs are requested, at least one of them must have a public ip in order to gain access from that VM to those with internal ips.

3.5.2 Configuring Software and Saving VMs

Saving VMs

When you deploy one of the base VMs in BonFIRE and SSH into it, you can install software and configure it as you like. You can save a new VM image based on this, so you do not have to repeat this process. This is done by issuing a SAVE_AS command and then shutting down the VM.

Saving VMs using OCCI

For saving VM using OCCI, please find a guideline on Save As using OCCI

Saving VMs using Portal

For saving VM using Portal, please find a guideline on Save As using Portal

3.5.3 An Experiment Scenario Example

This introduction imposes an experiment scenario and a general workflow on conducting an experiment. It aims to give an overview of the basic operations that can be done with BonFIRE.

An experiment scenario

An experimenter wants to run three experiments on different BonFIRE infrastructure sites (see *Infrastructure*)

1. an experiment with a client at HLRS, a server and an Aggregator for monitoring at EPCC
2. an experiment with a client at EPCC, a server and an aggregator at Cells

3. an experiment with a client, a server and an aggregator at Virtual Wall

For the experimenter, there are several possibilities to conduct the experiments:

- via BonFIRE Portal, which interacts graphically with the BonFIRE functionalities (see [BonFIRE Portal](#))
- via BonFIRE Broker API, which interacts directly with the BonFIRE functionalities (see [OCCI/API via HTTP\(cURL\)](#))
- via BonFIRE Experiment Manager (see [Experiment Descriptors](#))

First hour in BonFIRE

The [First hour in BonFIRE](#) videotutorial shows how to launch an experiment on BonFIRE using the Portal. The tutorial shows a video stream application between a server and a client, then the traffic is checked through the BonFIRE Zabbix monitoring service. The following machines are configured:

- A web backend hosting some video files (Debian + Nginx)
- A web frontend hosting a web page and a js video player requesting the videos from the web backend (Debian + Nginx + MediaElement)

If you are interested in trying the examples by yourself, you may download everything in the following sections.

Manual deployment

- web backend: `web_backend.tar.gz`
- web frontend: `web_frontend-sites-available.tar.gz`, `web_frontend-www.tar.gz`

Automatic deployment

- Non-Virtual Wall experiment: `experiment_descriptor.json`
- Virtual Wall experiment: `experiment_descriptor_vw.json`

For those experimenters interested in using Restfully, they can have a look at this page:

- Video Demonstrating Control and Observability

3.5.4 How to run experiments on Virtual Wall (`ibbt-vw`)

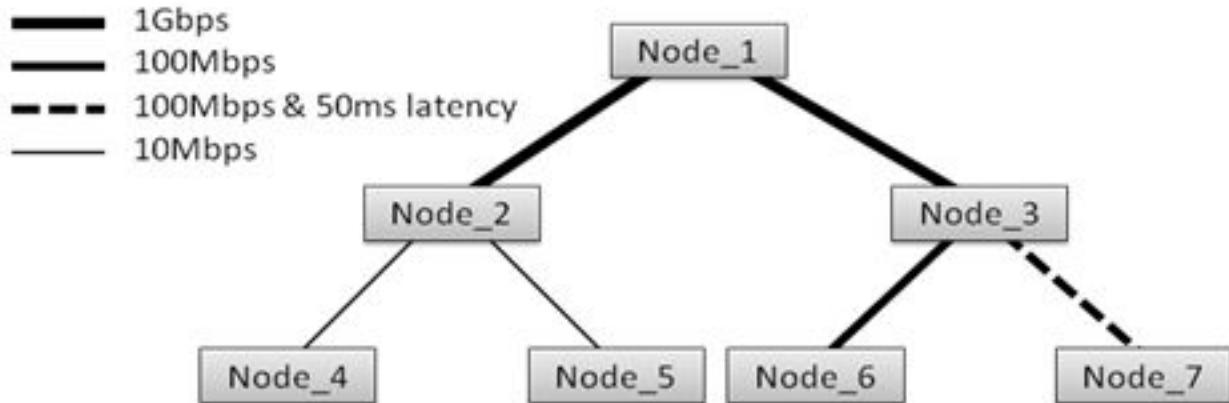
Experiments can be launched via the Portal, or directly using the API. See respectively [BonFIRE Portal](#) and [OCCI/API via HTTP\(cURL\)](#) to learn how to do this step by step. Below you'll find general information about the virtual wall specificities in how they handle the creation of resources.

Why use the Virtual Wall?

For the VW the networking part of the experiment is the most important. If you need a specific networking topology or you have specific needs for the properties of your network links, i.e. you want to specify the bandwidth, latency and lossrate of the network links, then you will need to use the Virtual Wall.

Preliminary work

Before you can start creating an experiment on the VW, you need to think about the resources you will need. Most important are the network resources, which are the very reason one uses the VW.



The picture above shows a certain network topology which needs to be translated to resources. On top you have a server and on the bottom you have 4 clients. These are connected to the server via two routers in the middle.

You have to translate this topology in a number of network resources. Intuitively you would create four subnets:

- Subnet1 (172.16.1.0): Node_1 - Node_2
- Subnet2 (172.16.2.0): Node_1 - Node_3
- Subnet3 (172.16.3.0): Node_2 - Node_4 & Node_2 - Node_5
- Subnet4 (172.16.4.0): Node_3 - Node_6 & Node_3 - Node_7

Each of these subnets translates to a network resource. However, since the link between Node_3 and Node_7 has different characteristics than the link between Node_3 and Node_6, a separate network resource needs to be created for that link. In total we thus have 5 network resources:

- Network1: Node_1 - Node_2
- Network2: Node_1 - Node_3
- Network3: Node_2 - Node_4 & Node_2 - Node_5
- Network4: Node_3 - Node_6
- Network5: Node_3 - Node_7

Furthermore we will need a total of 7 compute resources.

For more information about subnets, please check this document: <http://www.bradreese.com/how-to-subnet-a-network.pdf>

Creating experiments via OCCI

As mentioned in the beginning of this page, experiments can also be started using the API directly (see [OCCI/API via HTTP\(cURL\)](#)). Make sure to create your network resources before adding compute resources and start the experiment by setting it to the running state.

```
PUT /experiments/{{experiment_id}} HTTP/1.1
Host: api.bonfire-project.eu
Content-Type: application/vnd.bonfire+xml
Accept: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">action=run</experiment>
```

Creating experiments on the Portal

Create an experiment

When creating an experiment on the Portal you need to provide some information first:

- A name for the experiment
- A description of the experiment
- The experiment’s “walltime” in minutes: this is the time the experiment will run on the testbed
- You have to choose if you want to create a “Monitoring Aggregator” on one of the testbed sites

Add resources to the experiment

Once the experiment container has been created, you will need to add resources to it. For experiments on the VW, it is advisable to first add the network and storage resources and later add the compute resources, which refer to these network and storage resources.

Please take into account that a network must have at least 2 computes in it, otherwise the whole experiment will not swap in on the Virtual Wall. If the network has traffic generation, i.e. it is an “Active Network”, it must contain only 2 computes.

Take to hand the information you acquired during the preliminary work and add the desired number of network resources. Be sure to specify the correct bandwidth, latency and loss rate for every network resource.

You can also add a traffic generator to your network if you want. You have to set the type of the network to “active” and provide the desired protocol, throughput and packet size.

Furthermore, you can choose a queue strategy. In addition to default DropTail, the VW supports RED and GRED (Gentle RED). RED/GRED queuing is handled via the insertion of a traffic shaping delay node, in much the same way that bandwidth, delay, and packet loss is handled. If you choose to use RED/GRED you will be asked to provide additional parameters as you can see on the screenshot above. Hover over the questionmark to get more information about the different parameters.

In addition to creating network resources, you can also create some datablock storages and/or some shared storages to be used in your computes. A datablock resource can be either persistent or non-persistent. A non-persistent resource will be gone once the experiment has finished, while a persistent resource will still be available on the BonFIRE NFS server at IBBT after the experiment is no longer running. The location of this storage can be found in the mount point property of the storage. Shared storages can be shared between different experiments and are always persistent. They also have the mount point property pointing to their location on the BonFIRE NFS server at IBBT

After you created the necessary network and storage resources, you can start adding compute resources to experiment. You will again be asked to provide a name for every node, as well as a description. Choose an operating system from the list of VmImages, and add the desired network and storage resources to the compute. Be sure to add the BonFIRE

The screenshot shows the BonFIRE web application interface. At the top, there is a dark header bar with the BonFIRE logo in the center. To the left of the logo are three small icons: a brick, a flask, and a document. To the right are several user interface icons: a gear, a speech bubble, an info symbol, a camera, and a person profile. The person icon has the name "smelis" next to it.

Below the header, the URL "BonFIRE >> Experiments >> Create Experiment" is displayed. The main content area is titled "Create Experiment".

The form fields are as follows:

- Name ?**: Test Experiment
- Description ?**: A test to see if everything works
- Walltime (Lifetime in Minutes) ?**: 7200
- Group ?**: <private>
- Add a Monitoring Aggregator at this Site ?**: don't create an aggregator
- Add Infrastructure Metrics ?**:
- Let the Aggregator monitor itself ?**:
- Elasticity Capable Aggregator ?**:

At the bottom of the form are two buttons: a blue "Continue" button with a right-pointing arrow, and a green "Finish" button with a right-pointing triangle.

The screenshot shows the BonFIRE web interface with a dark header bar containing icons for home, experiments, details, and help, along with a user profile for 'smelis'. The main title 'BonFIRE' is centered above the content area. Below the header, the breadcrumb navigation shows 'BonFIRE >> Experiments >> Experiment Details >> Create Network'. The main content is titled 'Create Network Resource' and contains a form for defining a network resource. The form fields include:

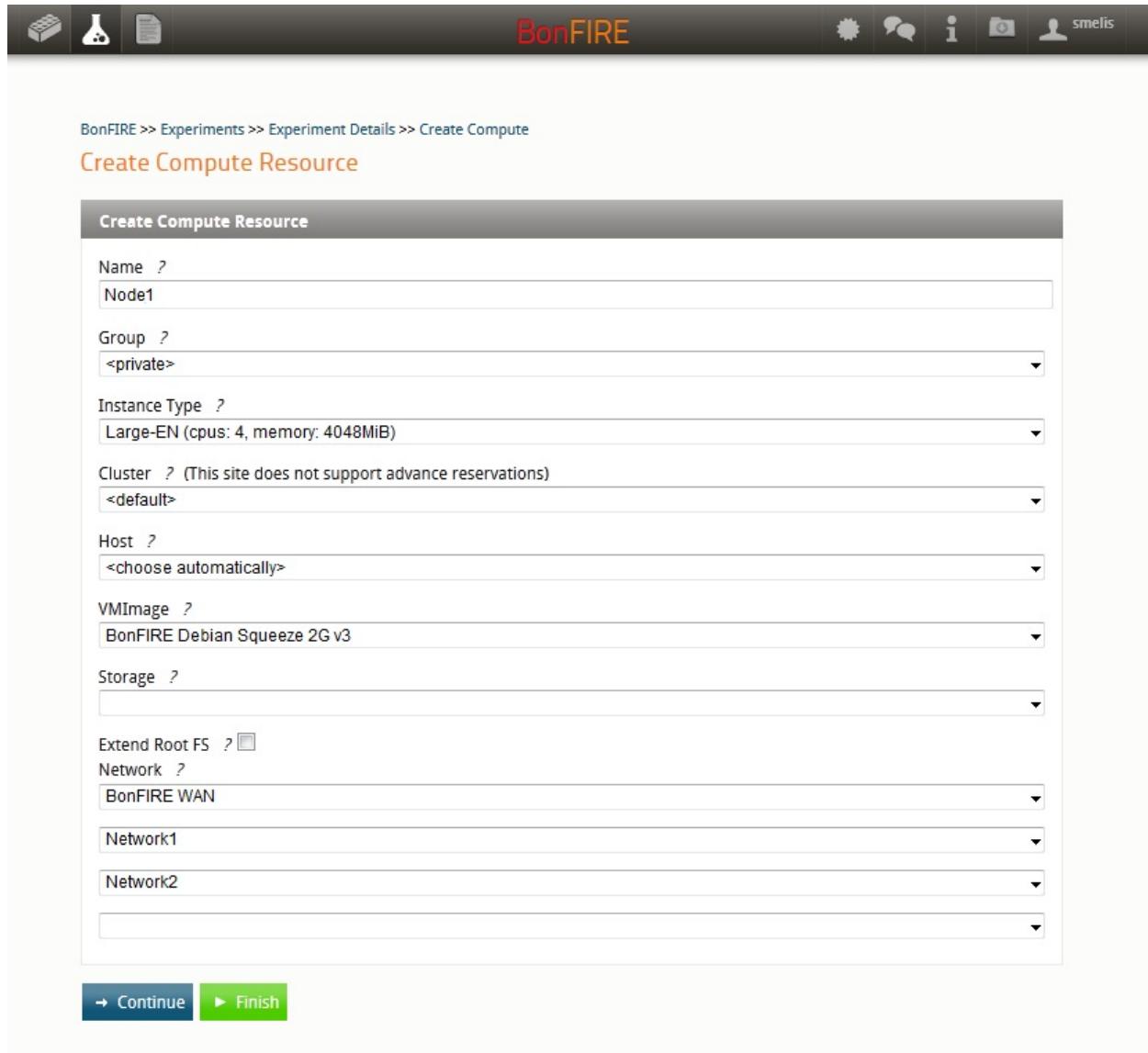
- Name :
- Group :
- Network Address :
- Size :
- Loss Rate (0 - 1) :
- Latency in ms :
- Bandwidth in Mbps :
- Network Type :
- Strategy :
- Queue-in-bytes :
- Limit :
- Maximum Threshold :
- Minimum Threshold :
- Linterm :
- Queue Weight :

At the bottom of the form are two buttons: a blue 'Continue' button with a right-pointing arrow, and a green 'Finish' button with a right-pointing arrow.

WAN network if you want your compute resource to be able to SSH to it. Note that the total size of all datablock storage resources in a compute cannot exceed 60GB.

Once you press Continue, you will be taken to the Review OCCl request page. Here you can add the context to compute resource, by means of a <context> xml element. This xml element should contain only elements which follow this syntax: <key>value</key>.

You can, for example, add a monitoring IP address for the monitoring aggregator as well as a possible startup script you created for your node. This startup script will be called automatically when the compute is swapped in on the VW, and can be used to automatically install the necessary software on your node.



BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Name ?
Node1

Group ?
<private>

Instance Type ?
Large-EN (cpus: 4, memory: 4048MiB)

Cluster ? (This site does not support advance reservations)
<default>

Host ?
<choose automatically>

VMImage ?
BonFIRE Debian Squeeze 2G v3

Storage ?

Extend Root FS ?

Network ?
BonFIRE WAN
Network1
Network2

→ Continue **► Finish**

Starting the experiment

After every network and compute resource is added to the experiment, the experiment can be started. On the Portal this can be done by pressing the GO button.

The screenshot shows the BonFIRE web interface. At the top, there is a dark header bar with icons for cloud, experiment, and user, followed by the text "BonFIRE" and a user profile icon. Below the header, the URL "BonFIRE >> Experiments >> Experiment Details >> Create Compute" is visible. The main content area has a title "Create Compute Resource" and a sub-section "Review OCCII Request for Compute Creation". Inside this section, there is a large text box containing the following XML code:

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Node1</name>
  <instance_type>Large-EN</instance_type>
  <disk>
    <storage href="/locations/be-ibbt/storages/1"/>
    <type>OS</type>
  </disk>
  <nics>
    <nic>
      <network href="/locations/be-ibbt/networks/1"/>
    </nic>
    <nic>
      <network href="/locations/be-ibbt/networks/18"/>
    </nic>
    <nic>
      <network href="/locations/be-ibbt/networks/19"/>
    </nic>
  </nics>
  <link href="/locations/be-ibbt" rel="location"/>
</compute>
```

Below the XML code, there is a checkbox labeled "Ignore XML Validation Errors" with a small error icon next to it. At the bottom of the page, there are two buttons: "Continue" (gray background) and "Finish" (green background).

BonFIRE >> Experiments >> Experiment Details

Compute created (/locations/be-ibbt/computes/60).

Experiment Details - *Test Experiment (ready)*

Experiment ID: /experiments/13623
User ID: smells
Groups: smelis
Creation Time: Fri, 09/07/12 13:06:45 UTC
Last Updated: Fri, 09/07/12 13:06:45 UTC
Expires: Wed, 09/12/12 13:06:45 UTC
[\(Show XML\)](#)

X Delete **Stop** **GO**

Compute Resources										Filter by Site: <all>
Name	Id	Type	Cpus	Mem	VM Image	Wan IP	SSH	State	Del.	
Node7	/locations/be-ibbt/computes/60	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		
Node6	/locations/be-ibbt/computes/59	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		
Node5	/locations/be-ibbt/computes/58	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		
Node4	/locations/be-ibbt/computes/57	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		
Node3	/locations/be-ibbt/computes/56	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		
Node2	/locations/be-ibbt/computes/55	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		
Node1	/locations/be-ibbt/computes/54	Large-EN	4	4048MiB	BonFIRE Debian S	<unassigned>	Not active - Host	PENDING		

Page 1 of 1 | Add Compute at be-ibbt | add

Storage Resources								Filter by Site: <all>	
Name	Id	Description		Type	Size	FS	Pub.	Per.	Del.

Page 1 of 0 | Add Storage at be-ibbt | add

Network Resources									Filter by Site: <all>
Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public	Del.	
Network5	/locations/be-ibbt/networks/22	172.16.4.0/C	100 Mbps	50ms	0.0	managed	false		
Network4	/locations/be-ibbt/networks/21	172.16.4.0/C	100 Mbps	0ms	0.0	managed	false		
Network3	/locations/be-ibbt/networks/20	172.16.3.0/C	10 Mbps	0ms	0.0	managed	false		
Network2	/locations/be-ibbt/networks/19	172.16.2.0/C	1000 Mbps	0ms	0.0	managed	false		
Network1	/locations/be-ibbt/networks/18	172.16.1.0/C	1000 Mbps	0ms	0.0	managed	false		

Page 1 of 1 | Add Network at be-ibbt | add

Elastic Groups			
Name	Elasticity Engine	Service Endpoint	# VMs

Page 1 of 0 | Create Elastic Group

Events			
Timestamp	Type	Id	Status
2012-09-07 13:09:43 UT	network	/locations/be-ibbt/networks/18	created
2012-09-07 13:10:27 UT	network	/locations/be-ibbt/networks/19	created
2012-09-07 13:10:54 UT	network	/locations/be-ibbt/networks/20	created
2012-09-07 13:11:25 UT	network	/locations/be-ibbt/networks/21	created
2012-09-07 13:11:48 UT	network	/locations/be-ibbt/networks/22	created
2012-09-07 13:20:26 UT	compute	/locations/be-ibbt/computes/54	created
2012-09-07 13:21:00 UT	compute	/locations/be-ibbt/computes/55	created
2012-09-07 13:21:42 UT	compute	/locations/be-ibbt/computes/56	created
2012-09-07 13:22:07 UT	compute	/locations/be-ibbt/computes/57	created
2012-09-07 13:22:28 UT	compute	/locations/be-ibbt/computes/58	created

Page 1 of 2 | View Raw Event Data

Once the experiment is started your experiment will be swapped in on the Virtual Wall. Once your experiment has been swapped in, you will be able to see the IP address you can use to SSH into your machines. Be sure to use the one in the BonFIRE WAN.

Once logged in on one of the nodes, you can install the needed software for your experiment, and save this image for later use via the resource's overview page.

Viewing monitoring data

Once the experiment is running, and you have added a monitoring aggregator to your experiment, you can view the monitoring data of your experiment.

Changing network parameters

Once the experiment is running you can change the network parameters if you created a non-default network (this is a prerequisite of the Virtual Wall itself). A default network is a network with no loss rate, no latency and a bandwidth of 1000Mbps.

Stopping the experiment

Once the experiment is finished, you can stop the experiment. Stopping the experiment results in the persistent storages being made persistent and images which you have set to be saved will be saved. After this has been done, you can delete the experiment as well.

Copying data into the Virtual Wall

Virtual Wall compute resources can only access the public internet over HTTP (**no HTTPS**). To download publically accessible data, wget can be used.

```
vw-comp> wget http://www.google.be/images/srpr/logo3w.png
```

If this method does not suffice to copy data into a Virtual Wall compute, one needs to tunnel. Here is a way to do it on Unix, but for this to work properly SSH needs to be configured as presented in section “Putting some Magic into your SSH Configuration” of [SSH Gateway Configuration](#).

```
home_computer> ssh -L1234:localhost:22 -l root <ip of vw-comp>
```

Keep this connection active and start another terminal on home_computer. The following command on the new terminal will copy myfile.txt to the home directory of user root on the IBBT VM:

```
home_computer> scp -P 1234 myfile.txt root@localhost:~/
```

BonFIRE >> Experiments >> Experiment Details

Experiment Details - *Test Experiment (running)*

Experiment ID: /experiments/13623
User ID: smelis
Groups: smelis
Creation Time: Fri, 09/07/12 13:06:45 UTC
Last Updated: Fri, 09/07/12 13:25:47 UTC
Expires: Wed, 09/12/12 13:06:45 UTC
[\(Show XML\)](#)

Compute Resources Filter by Site: <all>

Name	Id	Type	Cpus	Mem	VM Image	Wan IP	SSH	State	Del.
Node7	/locations/be-ibbt/computes/60	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.83	OK	ACTIVE	X
Node6	/locations/be-ibbt/computes/59	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.116	OK	ACTIVE	X
Node5	/locations/be-ibbt/computes/58	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.143	OK	ACTIVE	X
Node4	/locations/be-ibbt/computes/57	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.144	OK	ACTIVE	X
Node3	/locations/be-ibbt/computes/56	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.145	OK	ACTIVE	X
Node2	/locations/be-ibbt/computes/55	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.103	OK	ACTIVE	X
Node1	/locations/be-ibbt/computes/54	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.117	OK	ACTIVE	X

Page 1 of 1 | Add Compute at de-hrs | add

Storage Resources Filter by Site: <all>

Name	Id	Description	Type	Size	FS	Pub.	Per.	Del.

Page 1 of 0 | Add Storage at de-hrs | add

Network Resources Filter by Site: <all>

Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public	Del.
Network5	/locations/be-ibbt/networks/22	172.16.4.0/C	100 Mbps	50ms	0.0	managed	false	X
Network4	/locations/be-ibbt/networks/21	172.16.4.0/C	100 Mbps	0ms	0.0	managed	false	X
Network3	/locations/be-ibbt/networks/20	172.16.3.0/C	10 Mbps	0ms	0.0	managed	false	X
Network2	/locations/be-ibbt/networks/19	172.16.2.0/C	1000 Mbps	0ms	0.0	managed	false	X
Network1	/locations/be-ibbt/networks/18	172.16.1.0/C	1000 Mbps	0ms	0.0	managed	false	X

Page 1 of 1 | Add Network at uk-hplabs | add

Elastic Groups

Name	Elasticity Engine	Service Endpoint	# VMs

Page 1 of 0 | Create Elastic Group

Events

Timestamp	Type	Id	Status
2012-09-07 13:09:43 UT	network	/locations/be-ibbt/networks/18	created
2012-09-07 13:10:27 UT	network	/locations/be-ibbt/networks/19	created
2012-09-07 13:10:54 UT	network	/locations/be-ibbt/networks/20	created
2012-09-07 13:11:25 UT	network	/locations/be-ibbt/networks/21	created
2012-09-07 13:11:48 UT	network	/locations/be-ibbt/networks/22	created
2012-09-07 13:20:26 UT	compute	/locations/be-ibbt/computes/54	created
2012-09-07 13:21:00 UT	compute	/locations/be-ibbt/computes/55	created
2012-09-07 13:21:42 UT	compute	/locations/be-ibbt/computes/56	created
2012-09-07 13:22:07 UT	compute	/locations/be-ibbt/computes/57	created
2012-09-07 13:22:28 UT	compute	/locations/be-ibbt/computes/58	created

Page 1 of 2 | View Raw Event Data

3.5. Tutorials and Experiment Scenarios

No Monitoring Aggregator found for this experiment.

BonFIRE >> Experiments >> Experiment Details >> Resource Details

Details for Compute Resource /locations/be-ibbt/computes/54

ID: /locations/be-ibbt/computes/54
Name: Node1
Groups: smelis
Instance Type: Large-EN
CPUs: 4.0
Memory: 4048MB
OS Image: BonFIRE Debian Squeeze 2G v3 (/locations/be-ibbt/storages/1)
Wan IP: 172.18.4.117
State: ACTIVE
LogFile: Site be-ibbt does not support retrieving VM logs.
(Show XML)

Resource State

The state of this resource is 'ACTIVE'.

The state of Virtual Wall Computes can't be changed individually. Instead, all Computes will be started when the Experiment is set to *running* and shutdown when the Experiment is set to *stopped*. Note that consequently you will have to stop the experiment to apply setting a *save as* target.

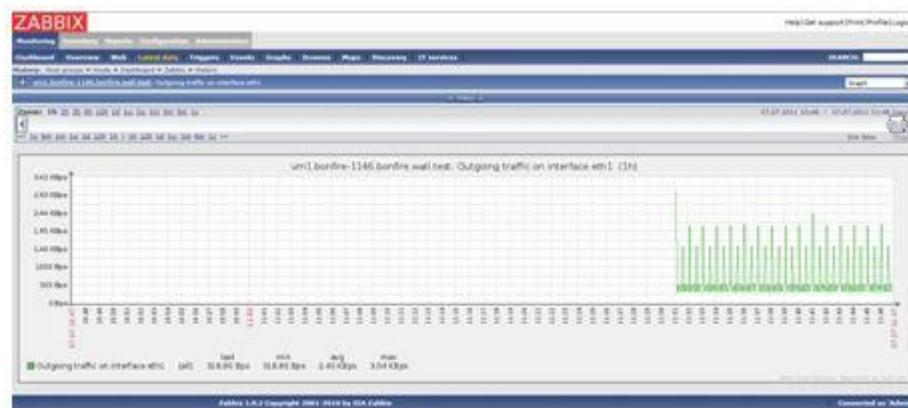
More information about VM states can be found here.

Save As

This compute's VMImage is not yet set to be saved.

Save VMImage as: + Set Save-As

More information on saving VM images can be found here.



BonFIRE >> Experiments >> Experiment Details

Experiment Details - *Test Experiment (running)*

Experiment ID: /experiments/13623
User ID: smelis
Groups: smelis
Creation Time: Fri, 09/07/12 13:06:45 UTC
Last Updated: Fri, 09/07/12 13:25:47 UTC
Expires: Wed, 09/12/12 13:06:45 UTC
[\(Show XML\)](#)

Compute Resources Filter by Site: <all>

Name	Id	Type	Cpus	Mem	VM Image	Wan IP	SSH	State	Del.
Node7	/locations/be-ibbt/computes/60	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.83	OK	ACTIVE	X
Node6	/locations/be-ibbt/computes/59	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.116	OK	ACTIVE	X
Node5	/locations/be-ibbt/computes/58	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.143	OK	ACTIVE	X
Node4	/locations/be-ibbt/computes/57	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.144	OK	ACTIVE	X
Node3	/locations/be-ibbt/computes/56	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.145	OK	ACTIVE	X
Node2	/locations/be-ibbt/computes/55	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.103	OK	ACTIVE	X
Node1	/locations/be-ibbt/computes/54	Large-EN	4	4048MiB	BonFIRE Debian S	172.18.4.117	OK	ACTIVE	X

Page 1 of 1 | Add Compute at de-hrs | add

Storage Resources Filter by Site: <all>

Name	Id	Description	Type	Size	FS	Pub.	Per.	Del.

Page 1 of 0 | Add Storage at de-hrs | add

Network Resources Filter by Site: <all>

Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public	Del.
Network5	/locations/be-ibbt/networks/22	172.16.4.0/C	100 Mbps	50ms	0.0	managed	false	X
Network4	/locations/be-ibbt/networks/21	172.16.4.0/C	100 Mbps	0ms	0.0	managed	false	X
Network3	/locations/be-ibbt/networks/20	172.16.3.0/C	10 Mbps	0ms	0.0	managed	false	X
Network2	/locations/be-ibbt/networks/19	172.16.2.0/C	1000 Mbps	0ms	0.0	managed	false	X
Network1	/locations/be-ibbt/networks/18	172.16.1.0/C	1000 Mbps	0ms	0.0	managed	false	X

Page 1 of 1 | Add Network at uk-hplabs | add

Elastic Groups

Name	Elasticity Engine	Service Endpoint	# VMs

Page 1 of 0 | Create Elastic Group

Events

Timestamp	Type	Id	Status
2012-09-07 13:09:43 UT	network	/locations/be-ibbt/networks/18	created
2012-09-07 13:10:27 UT	network	/locations/be-ibbt/networks/19	created
2012-09-07 13:10:54 UT	network	/locations/be-ibbt/networks/20	created
2012-09-07 13:11:25 UT	network	/locations/be-ibbt/networks/21	created
2012-09-07 13:11:48 UT	network	/locations/be-ibbt/networks/22	created
2012-09-07 13:20:26 UT	compute	/locations/be-ibbt/computes/54	created
2012-09-07 13:21:00 UT	compute	/locations/be-ibbt/computes/55	created
2012-09-07 13:21:42 UT	compute	/locations/be-ibbt/computes/56	created
2012-09-07 13:22:07 UT	compute	/locations/be-ibbt/computes/57	created
2012-09-07 13:22:28 UT	compute	/locations/be-ibbt/computes/58	created

Page 1 of 2 | View Raw Event Data

3.5. Tutorials and Experiment Scenarios

No Monitoring Aggregator found for this experiment.

3.5.5 How to run experiments on HP Cells (uk-hplabs)

This section describes how to run an experiment on HP Cells.

Why use Cells?

One of the most relevant features of HP Cells is the ability to allocate private subnets to aggregate sets of VMs to different networks. This allows for the definition of complex topologies. For instance, a company might want to split its VMs in order to deploy some of them in a private subnet behind a firewall, and some of them in a DMZ, accessible by customers. Another example would be a 3-tier architecture in which three different subnets would contain each a set of VMs for presentation, application processing and data management. Each layer will be associated to a different subnet and are isolated from each other. Some VMs can route traffic from one subnet to another one. One or several VMs can be accessed from the internet and/or from the BonFIRE WAN. Figure *Example scenario* illustrates this scenario.

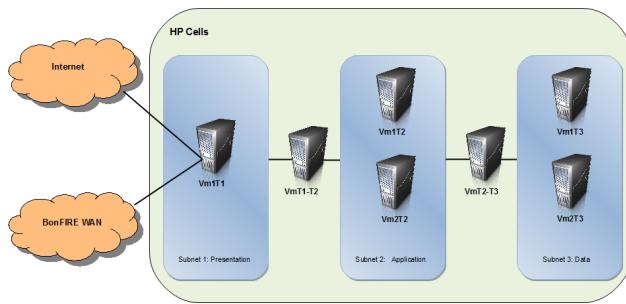


Figure 3.3: *Example scenario*

This latter example will be presented in the following sections as a use case to illustrate how to run an experiment on Cells.

Create an experiment

- In this experiment we will create a private subnet per layer (tier1Network, tier2Network and tier3Network) and will attach a number of VMs to each subnet, as shown in the previous figure. Two additional VMs will interconnect these subnets. First, when a new experiment is requested (Figure *Experiment definition*), the Portal will ask for a name, a description and the lifetime of the experiment in minutes. Optionally, the experiment can be assigned to a group so that it is accessible by all users in that group. By default, only the creator of the experiment will have access to it. In addition, an instance of the monitoring aggregator can be added to the experiment.
- The next screen will show the OCCI request to review it before clicking ‘Continue’ (Figure *OCCI request review*).
- Once in the Experiment Details screen, Cells allows the experimenter to define a private subnet to attach its VMs to. To do that, select ‘uk-hplabs’ in the Network Resources field, and click ‘add’, as shown in Figure *Adding a private subnet on Cells*. Remember that, prior to attach any VM to a subnet, the subnet must already exist.
- The subnet size can be selected depending on the number of VMs that we expect to attach to. In this example, a /29 network is chosen (Figure *Choosing the network size*).
- Again, a screen will prompt the experimenter to review the OCCI request to create the network. Now, back again in the Experiment Details screen, we can create a compute resource at uk-hplabs (Figure *Creating a new Compute Resource*). Note that the just created subnet is listed as well.



Figure 3.4: *Experiment definition*



Figure 3.5: *OCCI request review*

Compute Resources						
Name	ID	Type	Cpus	Mem	VM Image	Wan IP
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)

Storage Resources						
Name	ID	Description	Type	Size	FS	Pub
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)

Network Resources						
Name	ID	Address	Bandwidth	Latency	Loss Rate	Type
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)

Elastic Groups						
Name	Electric Engine	Service Endpoint	# VMs	Public	Del.	...
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	...
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	...
... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	... (redacted)	...

Events			
Timestamp	Type	ID	Status
... (redacted)	... (redacted)	... (redacted)	... (redacted)

Figure 3.6: *Adding a private subnet on Cells*

Figure 3.7: *Choosing the network size*



Figure 3.8: Creating a new Compute Resource

- The Compute Resource Wizard (Figure *New Compute Resource Wizard*) will prompt us to introduce the name of the VM, a group to be accessible by (if any), the instance type, the golden image to use, a block device for additional storage and the networks to attach the VM to. In this case, we choose a VM attached to the public ‘Internet@HP’ network (so that the VM has a public IP) and the tier1Network subnet created in the previous step. Note that the fields ‘Cluster’ and ‘Host’ do not apply to uk-hplabs, and will be removed in future revisions of the Portal.

This is a screenshot of the 'Create Compute Resource' wizard. It shows the configuration for a new VM named 'vm1t1'. The 'Group' is set to '<private>'. The 'Instance Type' is 'Ite (cpu 1, memory: 256MB)'. The 'VMImage' is 'BonFIRE Debian Squeeze v4'. The 'Network' section shows 'Internet@HP' and 'tier1Network' selected. At the bottom, there are 'Continue' and 'Finish' buttons.

Figure 3.9: New Compute Resource Wizard

- The new VM will boot and will be accessible on the public ip shown in the Experiment Details screen (Figure *VM for Tier1 deployed and running*). Note that the public ip can be accessed externally on ports 80, 443 and 22 (inbound and outbound). To use other ports, the requesting computer’s IP must be on the HP Labs whitelist or access through the BonFIRE WAN.



Figure 3.10: VM for Tier1 deployed and running

- Next, we define a tier2Network and create a second VM (vmT1-T2) attached to tier1Network and tier2Network. Since this VM is connected to both subnets, the experimenter can configure it to route traffic between subnets. Finally, we populate the tiers with VMs and create another VM to connect tier2 to tier3. In the end, our topology will look like the one in Figure *Final 3-tier topology*, in which we have also created a client VM in the uk-epcc site:

Compute Resources

Name	ID	Type	Cpus	Mem	VM Image	Wan IP	SSH	Status	Del.
vmT3	/locations/uk-epcc/computes/vm-31-157	lite	1	256MB	BonFIRE Debian Squeeze	192.170.103.180	OK	up	X
vmT1-T2	/locations/uk-epcc/computes/vm-31-158	lite	1	256MB	BonFIRE Debian Squeeze	192.170.103.179	OK	up	X
vmT2-T3	/locations/uk-epcc/computes/vm-31-154	lite	1	256MB	BonFIRE Debian Squeeze	192.170.103.183	OK	up	X
vmT3-T3	/locations/uk-epcc/computes/vm-31-155	lite	1	256MB	BonFIRE Debian Squeeze	192.170.103.184	OK	up	X
client	/locations/uk-epcc/computes/17113	medium	2	2048MB	BonFIRE Debian Squeeze	172.18.6.129	OK	RUNNING	X

Storage Resources

Name	ID	Description	Type	Size	FS	Pri	Pri	QoS

Network Resources

Name	ID	Address	Bandwidth	Latency	Loss Rate	Type	Protocol	QoS
tier1Network	/locations/uk-epcc/networks/subnet-31-	<0>>	<0>>	<0>>	<0>>	default	tcp	X
tier1Network	/locations/uk-epcc/networks/subnet-31-	<0>>	<0>>	<0>>	<0>>	default	tcp	X
tier2Network	/locations/uk-epcc/networks/subnet-31-	<0>>	<0>>	<0>>	<0>>	default	tcp	X

Figure 3.11: *Final 3-tier topology*

- Note that, in our example configuration, only the VM in the Presentation Tier has a public ip. The remaining ones are allocated in private subnets for the sake of modularity and isolation.
- Now, in the defined topology the client, using valid SSH keys, will be able to access vm1T1 in the Presentation Tier through the BonFIRE WAN by addressing the public IP of vm1T1.

Terminate experiment

- Any VM can be deleted at any moment by clicking in the red X next to each VM. To delete the whole experiment, press the button ‘Delete’ in the Experiment Details screen.

3.5.6 How to run experiments on Wellness Telecom (es-wellness)

Experiments can be launched via the Portal, or directly using the API. See respectively BonFIRE Portal and OCCI/API via HTTP(cURL) to learn how to do this step by step.

Why use Wellness Telecom?

Overall, VMware stands out as the cloud leader, and in particular, a leader in enterprise hybrid cloud solutions, due to the reach and maturity of its on-premises and off-premises offerings: IaaS solutions, its broad service provider ecosystem for IaaS, the industry-leading vFabric PaaS suite, Cloud Foundry for open standards-based PaaS, a broad portfolio of virtualization and cloud management solutions, and aggressive development of cross-cloud enabling tools and interfaces to bring them all together. In Gartner market analysis most recent evaluation of cloud service providers, four of the five top service providers ran on technologies based on VMware vCloud. Working with other industry leaders, VMware is helping enterprises gain the benefits of cloud computing while leveraging existing investments and retaining control.

VMware vCloud Director is the VMware solution that enables delivery of Infrastructure as a Service (IaaS). VMware defines cloud as elastic, lightweight entry and exit, available over internet protocols, and running on a shared infrastructure. VMware vCloud Director pools infrastructure resources among multiple clusters into policy-based virtual datacenters. By logically pooling infrastructure capacity into virtual datacenters, Wellness Telecom can manage resources more efficiently with complete abstraction between the delivery of infrastructure and the underlying hardware that supports it.

Creating experiments via OCCI

As mentioned in the beginning of this tutorial, experiments can also be started using the API directly (see OCCI/API via HTTP(cURL)):

```
PUT /experiments/{{experiment_id}} HTTP/1.1
Host: api.bonfire-project.eu
Content-Type: application/vnd.bonfire+xml
Accept: application/vnd.bonfire+xml
<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">action=run</experiment>
```

Creating experiments via the Portal

When creating an experiment on the portal you must provide some information such as:

- **Name:** A name for the experiment. It must begin with a letter and only contain letters, digits, spaces and the minus character.
- **Description:** A description for the experiment.
- **Walltime:** The lifetime of the experiment in minutes
- **Group:** You can optionally choose a BonFIRE user group that this experiment and resources therein will be accessible by. If you do not choose a group the experiment will be accessible only by yourself.
- **Enable AWS Support:** Select this if you would like to enable creating compute resources on Amazon Web Services within this Experiment. Please note that a separate AWS account is required.
- **Add a Monitoring Aggregator at this Site:** Choose if and where to deploy an instance of the BonFIRE monitoring aggregator for this experiment.
- **Add Infrastructure Metrics:** Selecting this will configure your Experiment's monitoring aggregator to include metrics from the physical infrastructure hosting your VM's.
- **Let the Aggregator monitor itself:** Selecting this will configure your Experiment's monitoring aggregator to monitor itself.
- **Elasticity Capable Aggregator:** In order to support the BonFIRE elasticity mechanisms, an experiment's aggregator needs to be configured appropriately as well as reside on a larger instance type than usual. In short: Select this if you ever want to use the BonFIRE elasticity mechanisms within this experiment.

Add resources to the experiment

Once the experiment container has been created, you will need to add resources to it. In order to add computes at WT's infrastructure you must select the option Add compute at es-wellness from the combo box.

Once you click in the add button a new form shows up. You must fill the following information:

- **Name:** The name for this compute. It must begin with a letter and only contain letters, digits and the minus character.
- **Group:** You can optionally choose a BonFIRE user group that this resource will be accessible by. If you do not choose a group the resource will be accessible only by yourself.
- **Instance Type:** This controls the type of VM you will get. For details on these choices please refer to the BonFIRE documentation.

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Create Experiment

Name ?
Test

Description ?
Creating a new experiment

Walltime (Lifetime in Minutes) ?
60

Group ?
wellness

Enable AWS Support ?

Add a Monitoring Aggregator at this Site ?
 don't create an aggregator

Add Infrastructure Metrics ?

Let the Aggregator monitor itself ?

Elasticity Capable Aggregator ?

→ Continue **► Finish**

BonFIRE >> Experiments >> Experiment Details

⚠ Experiment created (/experiments/2738).

Experiment Details - Test(ready)

Experiment ID: /experiments/2738
User ID: csanchez
Groups: wellness
Creation Time: Wed, 06/12/13 07:58:45 UTC
Last Updated: Wed, 06/12/13 07:58:45 UTC
Expires: Wed, 06/12/13 08:58:45 UTC
[\(Show XML\)](#)

X Delete **■ Stop** **► GO**

Resources	Site Interconnection	Elasticity	Statistics	Monitoring	Resource Usage																																																
<p>Compute Resources</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Id</th> <th>Type</th> <th>Cpus</th> <th>Mem</th> <th>VM Image</th> <th>Owner</th> <th>SCN</th> <th>Date</th> </tr> </thead> <tbody> <tr> <td colspan="9">Add Compute at es-wellness <input type="button" value="add"/></td> </tr> </tbody> </table> <p>Storage Resources</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Id</th> <th>Type</th> <th>Size</th> <th>FS</th> <th>State</th> <th>Per.</th> </tr> </thead> <tbody> <tr> <td colspan="7">Add Storage at be-ibbt <input type="button" value="add"/></td> </tr> </tbody> </table> <p>Network Resources</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Id</th> <th>Address</th> <th>Bandwidth</th> <th>Latency</th> <th>Loss Rate</th> <th>Type</th> <th>Public</th> </tr> </thead> <tbody> <tr> <td colspan="8">Add Network at be-ibbt <input type="button" value="add"/></td> </tr> </tbody> </table>						Name	Id	Type	Cpus	Mem	VM Image	Owner	SCN	Date	Add Compute at es-wellness <input type="button" value="add"/>									Name	Id	Type	Size	FS	State	Per.	Add Storage at be-ibbt <input type="button" value="add"/>							Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public	Add Network at be-ibbt <input type="button" value="add"/>							
Name	Id	Type	Cpus	Mem	VM Image	Owner	SCN	Date																																													
Add Compute at es-wellness <input type="button" value="add"/>																																																					
Name	Id	Type	Size	FS	State	Per.																																															
Add Storage at be-ibbt <input type="button" value="add"/>																																																					
Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public																																														
Add Network at be-ibbt <input type="button" value="add"/>																																																					

- **Cluster:** Choose the cluster on which this resource will be deployed. WT does not support advance reservations.
- **Host:** WT does not support the possibility of choosing which physical host to deploy this resource on.
- **VMIImage:** To choose the image that will be used to boot this VM.
- **Storage:** Optionally, you can choose to connect additional storage resources here. These will be available as a block device within your VM.
- **Extend Root FS:** Extend the root file system of this compute resource across the first attached datablock.
- **Network:** WT only provides one network for the computes resources (BonFIRE WAN)

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Create Compute Resource

Name ?
Thor

Group ?
partners

Instance Type ?
Lite (cpus: 1, memory: 256MB)

Cluster ? (This site does not support advance reservations)
<default>

Host ?

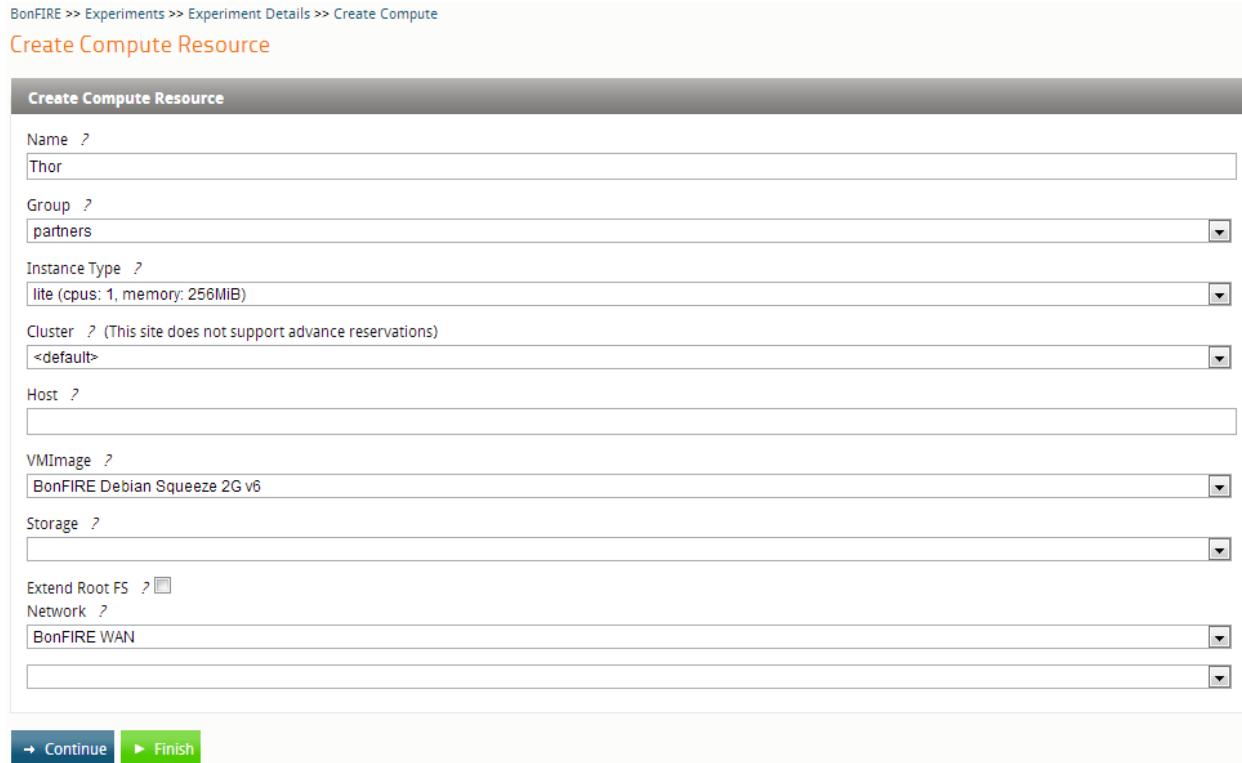
VMIImage ?
BonFIRE Debian Squeeze 2G v6

Storage ?

Extend Root FS ?

Network ?
BonFIRE WAN

→ Continue ▶ Finish



Once you press the **Continue** button the OCCI Request for Compute Creation is showed:

Pressing the **Finish** button creates the compute resources at WT's infrastructure.

Because of VMware limitations, the customization scripts of the virtual machines created at Wellness Telecom have a fixed size. Therefore, the number of SSH public keys that can be transferred to a VM is limited to 70 keys. This number includes all the public keys stored in user's profiles belonging to the group of the virtual machine.

Terminate experiment

Once the experiment is finished, you can stop the experiment by pressing the **stop** button at the top of the page. Any VM can be deleted at any moment by clicking in the red X next to each VM.

To delete the whole experiment, press the button **Delete** in the Experiment Details screen.

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Review OCCI Request for Compute Creation

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Thor</name>
  <groups>partners</groups>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/es-wellness/storages/vappTemplate-b4fa90c3-4c65-479a-b19d-31e903df1bfa"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
  <nics>
    <network href="/locations/es-wellness/networks/8d6b0daa-858b-41e7-b72e-310ac8619b50"/>
  </nics>
  <link href="/locations/es-wellness" rel="location"/>
</compute>
```

Ignore XML Validation Errors

BonFIRE >> Experiments >> Experiment Details

A Compute created (/locations/es-wellness/computes/vm-ee4dd34c-3408-48d8-889d-33fdd5a9c21f).

Experiment Details - *Test* (ready)

Experiment ID: /experiments/2738
User ID: csanchez
Groups: wellness
Creation Time: Wed, 06/12/13 07:58:45 UTC
Last Updated: Wed, 06/12/13 07:58:45 UTC
Expires: Wed, 06/12/13 08:58:45 UTC
 (Show XML)

Resources	Site Interconnection	Elasticity	Statistics	Monitoring	Resource Usage																																																
<p>Compute Resources</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Id</th> <th>Type</th> <th>Cpus</th> <th>Mem</th> <th>VM Image</th> <th>Wan IP</th> <th>SSH</th> <th>State</th> </tr> </thead> <tbody> <tr> <td>BonFIRE Del</td> <td>/locations/es-wellness/computes/vm-ee4dd</td> <td></td> <td>1</td> <td>512MB</td> <td>BonFIRE Debian Squ</td> <td>DHCP</td> <td>Not active - Host is</td> <td>STOPPED</td> </tr> </tbody> </table> <p>Filter by Site: <all> <input type="button" value="refresh"/></p> <p>Page 1 of 1 10 <input type="button" value="next"/> Add Compute at <input type="text" value="be-ibbt"/> <input type="button" value="add"/></p> <p>Storage Resources</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Id</th> <th>Type</th> <th>Size</th> <th>FS</th> <th>State</th> <th>Per.</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <p>Filter by Site: <all> <input type="button" value="refresh"/></p> <p>Page 1 of 0 10 <input type="button" value="next"/> Add Storage at <input type="text" value="be-ibbt"/> <input type="button" value="add"/></p> <p>Network Resources</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Id</th> <th>Address</th> <th>Bandwidth</th> <th>Latency</th> <th>Loss Rate</th> <th>Type</th> <th>Public</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table> <p>Filter by Site: <all> <input type="button" value="refresh"/></p> <p>Page 1 of 0 10 <input type="button" value="next"/> Add Network at <input type="text" value="be-ibbt"/> <input type="button" value="add"/></p>						Name	Id	Type	Cpus	Mem	VM Image	Wan IP	SSH	State	BonFIRE Del	/locations/es-wellness/computes/vm-ee4dd		1	512MB	BonFIRE Debian Squ	DHCP	Not active - Host is	STOPPED	Name	Id	Type	Size	FS	State	Per.								Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public								
Name	Id	Type	Cpus	Mem	VM Image	Wan IP	SSH	State																																													
BonFIRE Del	/locations/es-wellness/computes/vm-ee4dd		1	512MB	BonFIRE Debian Squ	DHCP	Not active - Host is	STOPPED																																													
Name	Id	Type	Size	FS	State	Per.																																															
Name	Id	Address	Bandwidth	Latency	Loss Rate	Type	Public																																														

3.5.7 Example HTTP Request/Response Session

Below you'll find a complete trace of HTTP requests and responses that create an experiment, look up the available storage and networks resources, add three compute resources to it, and then delete the experiment. These traces are given as examples of **raw** requests and responses, so that you can see what needs to be sent to create the various resources, and what is returned in response. You should probably use more advanced tools to interact with the API, such as *Restfully*.

```
Session started at Wed Jun 01 16:59:42 +0200 2011:  
-----  
GET / HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: */*  
Authorization: Basic XXX  
Accept-Encoding: gzip, deflate  
  
HTTP/1.1 200 OK  
Vary: Authorization,Accept  
Transfer-Encoding: chunked  
Etag: "fa2ba873343ba638123b7671c8c09998"  
Content-Type: application/vnd.bonfire+xml; charset=utf-8  
Date: Wed, 01 Jun 2011 14:59:30 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Allow: GET,OPTIONS,HEAD  
Cache-Control: public,max-age=120  
Connection: close  
  
<?xml version="1.0" encoding="UTF-8"?>  
<root xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/">  
  <version>0.8.9</version>  
  <timestampl>1306940370</timestampl>  
  <link rel="experiments" href="/experiments" type="application/vnd.bonfire+xml"/>  
  <link rel="locations" href="/locations" type="application/vnd.bonfire+xml"/>  
  <link rel="users" href="/users" type="application/vnd.bonfire+xml"/>  
</root>  
  
-----  
GET /experiments HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: application/vnd.bonfire+xml  
Authorization: Basic XXX  
Accept-Encoding: gzip, deflate  
  
HTTP/1.1 200 OK  
Vary: Authorization,Accept  
Transfer-Encoding: chunked  
Etag: "272348cca31dbd5e3e29a37b13c57281"  
Content-Type: application/vnd.bonfire+xml; charset=utf-8  
Date: Wed, 01 Jun 2011 14:59:30 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Allow: GET,POST,OPTIONS,HEAD  
Cache-Control: public,max-age=30  
Connection: close  
  
<?xml version="1.0" encoding="UTF-8"?>  
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/experiments">
```

```

<items offset="0" total="6">
<experiment href="/experiments/335">
<id>335</id>
<description>Demo of scenario1 using Restfully</description>
<name>Scenario1</name>
<walltime>1800</walltime>
<user_id>crohr</user_id>
<status>running</status>
<networks>
</networks>
<computes>
<compute href="/locations/fr-inria/computes/424" name="BonFIRE-monitor-experiment335"/>
<compute href="/locations/fr-inria/computes/425" name="server-experiment#335"/>
<compute href="/locations/uk-epcc/computes/288" name="client-experiment#335"/>
</computes>
<storages>
</storages>
<link rel="parent" href="/" />
<link rel="storages" href="/experiments/335/storages"/>
<link rel="networks" href="/experiments/335/networks"/>
<link rel="computes" href="/experiments/335/computes"/>
</experiment>
<experiment href="/experiments/334">
<id>334</id>
<description>Demo of scenario1 using Restfully</description>
<name>Scenario1</name>
<walltime>1800</walltime>
<user_id>crohr</user_id>
<status>running</status>
<networks>
</networks>
<computes>
</computes>
<storages>
</storages>
<link rel="parent" href="/" />
<link rel="storages" href="/experiments/334/storages"/>
<link rel="networks" href="/experiments/334/networks"/>
<link rel="computes" href="/experiments/334/computes"/>
</experiment>
<experiment href="/experiments/332">
<id>332</id>
<description>Demo of scenario1 using Restfully</description>
<name>Scenario1</name>
<walltime>1800</walltime>
<user_id>crohr</user_id>
<status>canceled</status>
<networks>
</networks>
<computes>
</computes>
<storages>
</storages>
<link rel="parent" href="/" />
<link rel="storages" href="/experiments/332/storages"/>
<link rel="networks" href="/experiments/332/networks"/>
<link rel="computes" href="/experiments/332/computes"/>
</experiment>

```

```
<experiment href="/experiments/331">
  <id>331</id>
  <description>Demo of scenario2 using Restfully</description>
  <name>Scenario1</name>
  <walltime>14400</walltime>
  <user_id>crohr</user_id>
  <status>running</status>
  <networks>
    <network href="/locations/be-ibbt/networks/10" name="network-experiment#331"/>
  </networks>
  <computes>
    <compute href="/locations/be-ibbt/computes/15" name="compute1-experiment#331"/>
    <compute href="/locations/be-ibbt/computes/16" name="compute2-experiment#331"/>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/331/storages"/>
  <link rel="networks" href="/experiments/331/networks"/>
  <link rel="computes" href="/experiments/331/computes"/>
</experiment>
<experiment href="/experiments/283">
  <id>283</id>
  <description>SDFsd</description>
  <name>hlrs-tests</name>
  <walltime>40000</walltime>
  <user_id>crohr</user_id>
  <status>running</status>
  <networks>
  </networks>
  <computes>
    <compute href="/locations/de-hlrs/computes/436" name="whatever"/>
    <compute href="/locations/fr-inria/computes/361" name="test"/>
    <compute href="/locations/fr-inria/computes/363" name="test-2g"/>
    <compute href="/locations/de-hlrs/computes/440" name="hlrs_c2"/>
    <compute href="/locations/de-hlrs/computes/441" name="hlrs_c1"/>
    <compute href="/locations/de-hlrs/computes/446" name="hlrs_c1_experiment#283"/>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/283/storages"/>
  <link rel="networks" href="/experiments/283/networks"/>
  <link rel="computes" href="/experiments/283/computes"/>
</experiment>
<experiment href="/experiments/224">
  <id>224</id>
  <description>sfsdf</description>
  <name>sdfdfs</name>
  <walltime>200000</walltime>
  <user_id>crohr</user_id>
  <status>waiting</status>
  <networks>
  </networks>
  <computes>
  </computes>
  <storages>
  </storages>
```

```

<link rel="parent" href="/" />
<link rel="storages" href="/experiments/224/storages" />
<link rel="networks" href="/experiments/224/networks" />
<link rel="computes" href="/experiments/224/computes" />
</experiment>
</items>
<link href="/" rel="parent" type="application/vnd.bonfire+xml" />
</collection>

-----
POST /experiments HTTP/1.1
Host: api.bonfire-project.eu
Accept: */*
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
Content-Type: application/vnd.bonfire+xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Scenario1</name>
  <walltime>1800</walltime>
  <description>Demo of scenario1 using Restfully</description>
</experiment>

HTTP/1.1 201 Created
Vary: Authorization,Accept
Transfer-Encoding: chunked
Location: https://api.bonfire-project.eu/experiments/336
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:31 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: no-cache
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/336" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <id>336</id>
  <description>Demo of scenario1 using Restfully</description>
  <name>Scenario1</name>
  <walltime>1800</walltime>
  <user_id>crohr</user_id>
  <status>running</status>
  <networks>
  </networks>
  <computes>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/336/storages" />
  <link rel="networks" href="/experiments/336/networks" />
  <link rel="computes" href="/experiments/336/computes" />
</experiment>

-----
GET /experiments/336 HTTP/1.1

```

```
Host: api.bonfire-project.eu
Accept: */
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "922928aa66e64e4150d43fd361747ab0"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:32 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,PUT,DELETE,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/336" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <id>336</id>
  <description>Demo of scenario1 using Restfully</description>
  <name>Scenario1</name>
  <walltime>1800</walltime>
  <user_id>crohr</user_id>
  <status>running</status>
  <networks>
  </networks>
  <computes>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/336/storages" />
  <link rel="networks" href="/experiments/336/networks" />
  <link rel="computes" href="/experiments/336/computes" />
</experiment>
```

```
-----
GET /locations HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "e5ecfd8cd78cecd466588356c973e8ec"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:32 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=120
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations">
  <items offset="0" total="5">
```

```

<location href="/locations/be-ibbt">
  <name>be-ibbt</name>
  <url>https://bonfire.test.atlantis.ugent.be</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/be-ibbt/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/be-ibbt/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/be-ibbt/storages" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/de-hlrs">
  <name>de-hlrs</name>
  <url>https://nebulosus.hlrs.de:8443</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/de-hlrs/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/de-hlrs/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/de-hlrs/storages" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/fr-inria">
  <name>fr-inria</name>
  <url>https://bonfire.grid5000.fr:443</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/fr-inria/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/fr-inria/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/fr-inria/storages" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/uk-epcc">
  <name>uk-epcc</name>
  <url>https://crockett.epcc.ed.ac.uk:8443</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/uk-epcc/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/uk-epcc/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/uk-epcc/storages" type="application/vnd.bonfire+xml"/>
</location>
<location href="/locations/uk-hplabs">
  <name>uk-hplabs</name>
  <url>https://occisvr-vif0.occi.ext8.bonfire.hpl.hp.com:443</url>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml"/>
  <link rel="computes" href="/locations/uk-hplabs/computes" type="application/vnd.bonfire+xml"/>
  <link rel="networks" href="/locations/uk-hplabs/networks" type="application/vnd.bonfire+xml"/>
  <link rel="storages" href="/locations/uk-hplabs/storages" type="application/vnd.bonfire+xml"/>
</location>
</items>
<link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>

```

 GET /locations/fr-inria/storages HTTP/1.1
 Host: api.bonfire-project.eu
 Accept: application/vnd.bonfire+xml
 Authorization: Basic XXX
 Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
 Vary: Authorization,Accept
 Transfer-Encoding: chunked
 Etag: "244d68733a2e0c353d4fa2dc4cc11280"
 Content-Type: application/vnd.bonfire+xml; charset=utf-8
 Date: Wed, 01 Jun 2011 14:59:32 GMT

```
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <storage href="/locations/fr-inria/storages/27" name="zabbix-aggregator"/>
    <storage href="/locations/fr-inria/storages/49" name="saved-client-experiment#240-image"/>
    <storage href="/locations/fr-inria/storages/52" name="saved-client-experiment#257-image"/>
    <storage href="/locations/fr-inria/storages/53" name="saved-client-experiment#266-image"/>
    <storage href="/locations/fr-inria/storages/62" name="squeeze-small"/>
    <storage href="/locations/fr-inria/storages/63" name="squeezevm3"/>
    <storage href="/locations/fr-inria/storages/64" name="squeeze 2G 2"/>
  </items>
</collection>
```

```
-----
GET /locations/uk-epcc/storages HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "1fad72ba8fb530cd72e022bd22d4ecdb"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:33 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <storage href="/locations/uk-epcc/storages/44" name="data"/>
    <storage href="/locations/uk-epcc/storages/47" name="data2"/>
    <storage href="/locations/uk-epcc/storages/74" name="whatever name"/>
    <storage href="/locations/uk-epcc/storages/91" name="Squeeze"/>
    <storage href="/locations/uk-epcc/storages/92" name="Squeeze 2G"/>
    <storage href="/locations/uk-epcc/storages/93" name="Zabbix Aggregator"/>
  </items>
</collection>
```

```
-----
GET /locations/fr-inria/networks HTTP/1.1
Host: api.bonfire-project.eu/
Accept: application/vnd.bonfire+xml
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
```

```
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "946f9bd48c594008c3cce0a034d2c9bc"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:33 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <network href="/locations/fr-inria/networks/2" name="WAN Network"/>
    <network href="/locations/fr-inria/networks/20" name="Public Network"/>
  </items>
</collection>

-----
GET /locations/fr-inria/networks/2 HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Cache-Control: no-cache
Authorization: Basic XXX
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "a2c756feabb739e13b346108f9d17f67"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:35 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Cache-Control: public,max-age=15
Connection: close

<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/networks/2">
  <id>2</id>
  <name>WAN Network</name>
  <address />
  <public>YES</public>
</network>

-----
GET /locations/fr-inria/networks/20 HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Cache-Control: no-cache
Authorization: Basic XXX
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
```

```
Etag: "030d9f50e8508b198ccf7603097e502b"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:37 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Cache-Control: public,max-age=15
Connection: close

<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/networks/20"
  <id>20</id>
  <name>Public Network</name>
  <address />
  <public>YES</public>
</network>

-----
GET /locations/uk-epcc/networks HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Authorization: Basic XXX
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "3a23bcd91dea2876d9f2febcb0dc906"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:37 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <network href="/locations/uk-epcc/networks/2" name="Crockett LAN"/>
  </items>
</collection>

-----
GET /locations/uk-epcc/networks/2 HTTP/1.1
Host: https://api.bonfire-project.eu/
Accept: application/vnd.bonfire+xml
Cache-Control: no-cache
Authorization: Basic XXX
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "d2d5b91ca84af00536fc1b2ef63d8ca4"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:38 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
```

```
Allow: GET,DELETE,PUT,OPTIONS,HEAD  
Cache-Control: public,max-age=15  
Connection: close
```

```
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/uk-epcc/networks/2">  
  <id>2</id>  
  <name>Crockett LAN</name>  
  <address>172.18.3.128</address>  
  <size>64</size>  
  <public>YES</public>  
</network>
```

```
-----  
GET /locations/fr-inria/networks/2 HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: application/vnd.bonfire+xml  
Cache-Control: no-cache  
If-None-Match: "a2c756feabb739e13b346108f9d17f67"  
Authorization: Basic XXX  
If-Modified-Since:  
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 304 Not Modified  
Vary: Authorization,Accept  
Etag: "a2c756feabb739e13b346108f9d17f67"  
Date: Wed, 01 Jun 2011 14:59:38 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Cache-Control: public,max-age=15  
Connection: close
```

```
-----  
GET /locations/fr-inria/networks/20 HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: application/vnd.bonfire+xml  
Cache-Control: no-cache  
If-None-Match: "030d9f50e8508b198ccf7603097e502b"  
Authorization: Basic XXX  
If-Modified-Since:  
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 304 Not Modified  
Vary: Authorization,Accept  
Etag: "030d9f50e8508b198ccf7603097e502b"  
Date: Wed, 01 Jun 2011 14:59:40 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Cache-Control: public,max-age=15  
Connection: close
```

```
-----  
GET /experiments/336/computes HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: application/vnd.bonfire+xml  
Authorization: Basic XXX
```

```
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "26d3092a9c5be1783c900af5b63f216f"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:41 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,POST,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/experiments/336/computes">
  <items offset="0" total="0">
  </items>
  <link href="/experiments/336" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

```
POST /experiments/336/computes HTTP/1.1
Host: api.bonfire-project.eu
Accept: */
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
Content-Type: application/vnd.bonfire+xml; charset=utf-8
```

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>BonFIRE-monitor-experiment336</name>
  <location href="https://api.bonfire-project.eu/locations/fr-inria"/>
  <instance_type>small</instance_type>
  <nic id="0">
    <network href="https://api.bonfire-project.eu/locations/fr-inria/networks/20"/>
  </nic>
  <disk id="0">
    <type>OS</type>
    <storage href="https://api.bonfire-project.eu/locations/fr-inria/storages/27"/>
  </disk>
</compute>
```

```
HTTP/1.1 201 Created
Vary: Authorization,Accept
Transfer-Encoding: chunked
Location: https://api.bonfire-project.eu/locations/fr-inria/computes/426
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:41 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: no-cache
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/426">
  <id>426</id>
```

```

<cpu>1</cpu>
<memory>1024</memory>
<name>BonFIRE-monitor-experiment336</name>
<instance_type>small</instance_type>
<state>PENDING</state>
<disk id="0">
  <storage href="/locations/fr-inria/storages/27" name="zabbix-aggregator"/>
  <type>DISK</type>
  <target>xvda</target>
</disk>
<nics>
  <network href="/locations/fr-inria/networks/20" name="Public Network"/>
  <ip>131.254.204.144</ip>
  <mac>02:00:83:fe:cc:90</mac>
</nics>
</compute>

-----
GET /locations/fr-inria/computes/426 HTTP/1.1
Host: api.bonfire-project.eu
Accept: */
Authorization: Basic XXX
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "b960ad75ae9dd3b686b32970ff059149"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:47 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Cache-Control: public,max-age=15
Connection: close

<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/426">
  <id>426</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>BonFIRE-monitor-experiment336</name>
  <instance_type>small</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/27" name="zabbix-aggregator" />
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nics>
    <network href="/locations/fr-inria/networks/20" name="Public Network" />
    <ip>131.254.204.144</ip>
    <mac>02:00:83:fe:cc:90</mac>
  </nics>
</compute>

-----
GET /experiments/336/computes HTTP/1.1

```

```
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
If-None-Match: "26d3092a9c5be1783c900af5b63f216f"
Authorization: Basic XXX
If-Modified-Since:
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "3499b842ff2cbf18eb02f90267fd5b79"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:47 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,POST,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/experiments/336/computes">
  <items offset="0" total="1">
    <compute href="/locations/fr-inria/computes/426" name="BonFIRE-monitor-experiment336"/>
  </items>
  <link href="/experiments/336" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

```
-----
POST /experiments/336/computes HTTP/1.1
Host: api.bonfire-project.eu
Accept: */
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
Content-Type: application/vnd.bonfire+xml; charset=utf-8
```

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>server-experiment#336</name>
  <location href="https://api.bonfire-project.eu/locations/fr-inria"/>
  <instance_type>small</instance_type>
  <context>
    <aggregator_ip>131.254.204.144</aggregator_ip>
  </context>
  <nic id="0">
    <network href="https://api.bonfire-project.eu/locations/fr-inria/networks/20"/>
  </nic>
  <disk id="0">
    <type>OS</type>
    <storage href="https://api.bonfire-project.eu/locations/fr-inria/storages/64"/>
  </disk>
</compute>
```

```
HTTP/1.1 201 Created
Vary: Authorization,Accept
Transfer-Encoding: chunked
Location: https://api.bonfire-project.eu/locations/fr-inria/computes/427
```

```

Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:47 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: no-cache
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/427">
  <id>427</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>server-experiment#336</name>
  <instance_type>small</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/64" name="squeeze 2G 2"/>
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/20" name="Public Network"/>
    <ip>131.254.204.145</ip>
    <mac>02:00:83:fe:cc:91</mac>
  </nic>
</compute>

-----
GET /locations/fr-inria/computes/427 HTTP/1.1
Host: api.bonfire-project.eu
Accept: */*
Authorization: Basic XXX
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "49d507240e4aac83f2b6264666cf6328"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:51 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Cache-Control: public,max-age=15
Connection: close

<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/427">
  <id>427</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>server-experiment#336</name>
  <instance_type>small</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/64" name="squeeze 2G 2" />
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nic>

```

```
<network href="/locations/fr-inria/networks/20" name="Public Network" />
<ip>131.254.204.145</ip>
<mac>02:00:83:fe:cc:91</mac>
</nic>
</compute>

-----
GET /experiments/336/computes HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
If-None-Match: "3499b842ff2cbf18eb02f90267fd5b79"
Authorization: Basic XXX
If-Modified-Since:
Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "e8d854ca601dacb5e18ebff30a0c713c"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:51 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,POST,OPTIONS,HEAD
Cache-Control: public,max-age=30
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/experiments/336/computes">
  <items offset="0" total="2">
    <compute href="/locations/fr-inria/computes/426" name="BonFIRE-monitor-experiment336"/>
    <compute href="/locations/fr-inria/computes/427" name="server-experiment#336"/>
  </items>
  <link href="/experiments/336" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>

-----
POST /experiments/336/computes HTTP/1.1
Host: api.bonfire-project.eu
Accept: */*
Authorization: Basic XXX
Accept-Encoding: gzip, deflate
Content-Type: application/vnd.bonfire+xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>client-experiment#336</name>
  <location href="https://api.bonfire-project.eu/locations/uk-epcc"/>
  <instance_type>small</instance_type>
  <context>
    <server_ip>131.254.204.145</server_ip>
    <aggregator_ip>131.254.204.144</aggregator_ip>
  </context>
  <nics id="0">
    <network href="https://api.bonfire-project.eu/locations/uk-epcc/networks/2"/>
  </nics>
  <disk id="0">
```

```

<type>OS</type>
<storage href="https://api.bonfire-project.eu/locations/uk-epcc/storages/91"/>
</disk>
</compute>

```

HTTP/1.1 201 Created
 Vary: Authorization,Accept
 Transfer-Encoding: chunked
 Location: https://api.bonfire-project.eu/locations/uk-epcc/computes/289
 Content-Type: application/vnd.bonfire+xml; charset=utf-8
 Date: Wed, 01 Jun 2011 14:59:52 GMT
 Server: thin 1.2.11 codename Bat-Shit Crazy
 Cache-Control: no-cache
 Connection: close

```

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/uk-epcc/computes/289">
  <id>289</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>client-experiment#336</name>
  <instance_type>small</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/uk-epcc/storages/91" name="Squeeze"/>
    <type>DISK</type>
    <target>sda</target>
  </disk>
  <nics>
    <network href="/locations/uk-epcc/networks/2" name="Crockett LAN"/>
    <ip>172.18.3.151</ip>
    <mac>00:18:ac:12:03:97</mac>
  </nics>
</compute>

```

 GET /locations/uk-epcc/computes/289 HTTP/1.1
 Host: api.bonfire-project.eu
 Accept: */*
 Authorization: Basic XXX
 Accept-Encoding: gzip, deflate

HTTP/1.1 200 OK
 Vary: Authorization,Accept
 Transfer-Encoding: chunked
 Etag: "8182c48b2092f2f00f989bf15b2ceae"
 Content-Type: application/vnd.bonfire+xml; charset=utf-8
 Date: Wed, 01 Jun 2011 14:59:54 GMT
 Server: thin 1.2.11 codename Bat-Shit Crazy
 Allow: GET,DELETE,PUT,OPTIONS,HEAD
 Cache-Control: public,max-age=15
 Connection: close

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/uk-epcc/computes/289">
  <id>289</id>
```

```

<cpu>1</cpu>
<memory>1024</memory>
<name>client-experiment#336</name>
<instance_type>small</instance_type>
<state>PENDING</state>
<disk id="0">
  <storage href="/locations/uk-epcc/storages/91" name="Squeeze" />
  <type>DISK</type>
  <target>sda</target>
</disk>
<nics>
  <network href="/locations/uk-epcc/networks/2" name="Crockett LAN" />
  <ip>172.18.3.151</ip>
  <mac>00:18:ac:12:03:97</mac>
</nics>
</compute>

```

```

-----
GET /locations/fr-inria/computes/427 HTTP/1.1
Host: api.bonfire-project.eu
Accept: */
Cache-Control: no-cache
If-None-Match: "49d507240e4aac83f2b6264666cf6328"
Authorization: Basic XXX
If-Modified-Since:
Accept-Encoding: gzip, deflate

```

```

HTTP/1.1 200 OK
Vary: Authorization,Accept
Transfer-Encoding: chunked
Etag: "cdf08f573352145bbfd1eac7eb6e07bd"
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Date: Wed, 01 Jun 2011 14:59:54 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Cache-Control: public,max-age=15
Connection: close

```

```

<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/427">
  <id>427</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>server-experiment#336</name>
  <instance_type>small</instance_type>
  <state>ACTIVE</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/64" name="squeeze 2G 2" />
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nics>
    <network href="/locations/fr-inria/networks/20" name="Public Network" />
    <ip>131.254.204.145</ip>
    <mac>02:00:83:fe:cc:91</mac>
  </nics>
</compute>

```

```
-----  
GET /locations/uk-epcc/computes/289 HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: */*  
Cache-Control: no-cache  
If-None-Match: "8182c48b2092f2f00f989bf15b2ceae"  
Authorization: Basic XXX  
If-Modified-Since:  
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 304 Not Modified  
Vary: Authorization,Accept  
Etag: "8182c48b2092f2f00f989bf15b2ceae"  
Date: Wed, 01 Jun 2011 14:59:57 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Cache-Control: public,max-age=15  
Connection: close
```

```
-----  
DELETE /experiments/336 HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: */*  
Authorization: Basic XXX  
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 202 Accepted  
Vary: Authorization,Accept  
Transfer-Encoding: chunked  
Location: https://api.bonfire-project.eu/experiments/336  
Content-Type: text/html; charset=utf-8  
Date: Wed, 01 Jun 2011 15:00:05 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Cache-Control: no-cache  
Connection: close
```

```
-----  
GET /experiments/336 HTTP/1.1  
Host: api.bonfire-project.eu  
Accept: */*  
If-None-Match: "922928aa66e64e4150d43fd361747ab0"  
Authorization: Basic XXX  
If-Modified-Since:  
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 200 OK  
Vary: Authorization,Accept  
Transfer-Encoding: chunked  
Etag: "af68055474f3bb687b8ebb4af186bd40"  
Content-Type: application/vnd.bonfire+xml; charset=utf-8  
Date: Wed, 01 Jun 2011 15:00:06 GMT  
Server: thin 1.2.11 codename Bat-Shit Crazy  
Allow: GET,PUT,DELETE,OPTIONS,HEAD
```

Cache-Control: public,max-age=30
Connection: close

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/336" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <id>336</id>
  <description>Demo of scenario1 using Restfully</description>
  <name>Scenario1</name>
  <walltime>1800</walltime>
  <user_id>crohr</user_id>
  <status>canceling</status>
  <networks>
  </networks>
  <computes>
    <compute href="/locations/fr-inria/computes/426" name="BonFIRE-monitor-experiment336"/>
    <compute href="/locations/fr-inria/computes/427" name="server-experiment#336"/>
    <compute href="/locations/uk-epcc/computes/289" name="client-experiment#336"/>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/336/storages" />
  <link rel="networks" href="/experiments/336/networks" />
  <link rel="computes" href="/experiments/336/computes" />
</experiment>
</source>
```

CLIENT TOOLS

4.1 Overview of Client Tools

One can interact with BonFIRE in many ways, such as via the BonFIRE Portal, Command Line Interface Tools or directly via the BonFIRE API. The API is RESTful, which is exposed over the HTTP protocol and can be accessed via <https://api.bonfire-project.eu/>.

Using the BonFIRE API requires that you are already registered as a BonFIRE user. If not, head over to [Registering a BonFIRE Account](#) and follow the instructions.

To create experiments and Cloud resources in BonFIRE, it is possible to issue raw HTTP commands to the BonFIRE API via, for example, [cURL](#). However, we also offer multiple client tools for interacting easily and effectively with BonFIRE:

- BonFIRE Portal: a graphical client tool that operates via your web browser, which is very intuitive to use. The Portal also gives you the option to set up a Managed Experiment in a step-by-step procedure to define the initial deployment of compute, storage and networking resources for your experiment. As part of this procedure, the Portal will create an *Experiment Descriptor*, which you can save and use again.
- BonFIRE Experiment Descriptor: instead of following a step-by-step procedure on the Portal, you can write/edit an experiment descriptor in either JSON or OVF. These descriptors are interpreted by an Experiment Manager, allowing you to describe all of the resources for an initial deployment of an experiment without needing to know OCCI or how to form HTTP messages.
- Restfully: a general-purpose client library for RESTful APIs, which is written in Ruby. Its goal is to abstract the nitty-gritty details of exchanging HTTP requests between the user-agent and the server. Restfully also allows you to write scripts for your experiment deployment which can be automatically executed.
- Command Line Interface Tools: BonFIRE also offers CLI tools that are meant to provide users with a way to interact with the BonFIRE API from a command line interface. This could be used in an interactive, manual, fashion or programmatically (i.e., scripted).

4.2 Experiment Descriptors

The BonFIRE Experiment Manager allows experimenters to create a managed experiment by uploading a single experimenter descriptor describing all of the resources for an initial deployment of an experiment.

Currently the initial deployment of an experiment can be described in a single JSON document or in a standard way using OVF.

For information about Experiment Manager and how to use JSON descriptor, OVF descriptor, please see the following links:

4.2.1 Introduction to the Experiment Manager

The Experiment Manager allows you to create a managed experiment by uploading an experiment descriptor describing all of the resources for an initial deployment of an experiment. The Experiment Manager will make a series of calls to the BonFIRE API to deploy the resources. The API for the Experiment manager is described in [Experiment Manager API](#) and an example of using it is given in [Command Line Interface Tools](#). The JSON experiment descriptor is described in [JSON Experiment Descriptor](#). You can also generate and submit an experiment descriptor from the BonFIRE web page, as described in [Creating a Managed Experiment From the GUI](#). You can also submit an experiment descriptor in OVF format :doc:`ovf`

4.2.2 Experiment Manager API

Overview

- This interface allows you to create a managed experiment by uploading an experiment descriptor, and then to view some information on the managed experiment, such as its log file. The managed experiment can also be deleted.
- We support batch operations (i.e. sending one big experiment descriptor with all the resources required).
- The experiment description will immediately be parsed and validated. You will immediately be notified of the success or failure of the validation stage.
- The experiment will be deployed in the background and you can check the progress by doing a GET on the managed experiment log.
- Deleting a managed experiment will also delete the experiment created by it.

Create a managed experiment

Create a managed experiment by uploading an Experiment Descriptor in json format.

Request:

```
POST https://api.bonfire-project.eu/managed_experiments

{
    "description": "Experiment description",
    "duration": 120,
    "name": "My Experiment"
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Location: https://api.bonfire-project.eu/managed_experiments/4

<managed_experiment>
    <name>My Experiment</name>
    <description>Experiment description</description>
    <status>valid</status>
    <link rel="experiment" href="" />
    <link rel="log" href="/managed_experiments/{managed_experiment_id}/log" />
</managed_experiment>
```

or

- Invalid experiment descriptors will return:

```
HTTP/1.1 400 Bad Request  
Content-Type: text/plain
```

A description of the validation errors.

View a managed experiment

Request:

```
GET https://api.bonfire-project.eu/managed_experiments/{managed_experiment_id}
```

Response:

Returns an xml formatted description of the managed experiment.

```
HTTP/1.1 200 OK  
Content-Type: application/xml  
Location: https://api.bonfire-project.eu/managed_experiments/4
```

```
<managed_experiment>  
  <name> A managed experiment </name>  
  <description> A managed experiment </description>  
  <status>running</status>  
  <link rel="experiment" href="http://api.bonfire-project.eu/experiments/32" />  
  <link rel="log" href="/managed_experiments/{managed_experiment_id}/log" />  
</managed_experiment>
```

View a managed experiment log file

Request:

```
GET https://api.bonfire-project.eu/managed_experiments/{managed_experiment_id}/log
```

Response: Returns a log file for the managed experiment.

```
HTTP/1.1 200 OK  
Content-Type: text/plain
```

Text of log file here.

View a list of managed experiments

Request: Managed experiments of ALL USERS are returned by default.

```
GET https://api.bonfire-project.eu/managed_experiments/
```

Response:

Returns an xml formatted list of managed experiments.

```
HTTP/1.1 200 OK  
Content-Type: application/xml  
  
<collection href="/experiments?offset=40&limit=20">  
  <items offset="40" total="110">
```

```
<managed_experiment>
...
</managed_experiment>
...
</items>
</collection>
```

Updates of managed experiments are not supported

Request:

```
PUT https://api.bonfire-project.eu/managed_experiments/{managed_experiment_id}
```

Response:

```
HTTP/1.1 405 Method Not Allowed
```

Delete a managed experiment

This will also delete the deployed experiment belonging to the managed experiment.

Request:

```
DELETE https://api.bonfire-project.eu/managed_experiments/{managed_experiment_id}
```

Response:

```
HTTP/1.1 202 Accepted
```

4.2.3 JSON Experiment Descriptor

We'll start with an example of how the JSON experiment descriptor would describe the initial set up of a server and client, and then describe the various options available in more detail.

In this experiment a server VM is set up at EPCC based on an image called "BonFIRE Debian Squeeze v5" and using the "BonFIRE WAN" network. A client VM is set up at Inria using the "BonFIRE Debian Squeeze v5" image and "BonFIRE WAN" network.

In this example the experiment will run for a maximum of 120 minutes ("duration": 120).

```
{
  "name": "My Experiment",
  "description": "Experiment description",
  "duration": 120,
  "resources": [
    {
      "compute": {
        "name": "Server",
        "description": "A description of the server.",
        "instanceType": "small",
        "locations": ["uk-epcc"],
        "resources": [
          { "storage": "@BonFIRE Debian Squeeze v5" },
          { "network": "@BonFIRE WAN" }
        ]
      }
    }
  ]
}
```

```

},
{
  "compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
    "resources": [
      { "storage": "@BonFIRE Debian Squeeze v5" },
      { "network": "@BonFIRE WAN" }
    ]
  }
}
]
}

```

References

There are 3 kinds of references that can be used in the experiment descriptor:

- resources that already exist can be referenced by name, prefixed by an “@” symbol as seen in the example above, e.g. “@BonFIRE WAN”
- resources that already exist can be referenced by uri, e.g.”<https://api.bonfire-project.eu/locations/fr-inria/storages/14>“
- resources that have previously been created in the experiment descriptor document can simply be referenced by name, e.g. “Server”.

Please note that the Experiment Manager processes resources in order of definition in the experiment descriptor. A compute cannot refer to a resource before the resource has been defined in the experiment descriptor.

Contexts

A compute description can contain a set of contextualisation information. There are currently two types of contexts supported: – IP Address Dependency – Contextualisation Element

IP Address Dependencies

The runtime IP address of one VM can be automatically be sent to another as follows:

```

...
"compute": {
  "name": "Client",
  ...
  "contexts": [
    {
      "ServerIP": [ "Server", "BonFIRE WAN" ]
    }
  ]
}

```

The “contexts” section says that the “ServerIP” *Contextualisation* property will be populated with the IP address of the Server compute on the BonFIRE WAN. (ServerIP is an example name.)

A special case is when we want to pass the IP address of an aggregator compute to a client. Then we must name the context “aggregator_ip”. For example, if we have created an aggregator compute called “BonFIRE-Monitor”, then our aggregator client could use the following to obtain the aggregator’s IP address:

```
...
"compute": {
    "name": "AggregatorClient",
    ...
    "contexts": [
        {
            "aggregator_ip": [ "BonFIRE-Monitor", "BonFIRE WAN" ]
        }
    ]
}
```

Cross-references in both directions cannot be handled, i.e. The client can be told the address of the server (or vice versa) but it is not possible to send the client the server’s address **and** send the server the client’s address. This is because a VM must be deployed before its IP address is known, and the contextualisation is set as it is deployed. The json should be written in deployment order. In the example above, the Server must be defined before the client that references it.

Note that the experiment manager *cannot* resolve IP dependencies to both a pre-existing network and a network with the same name defined in the experiment descriptor. For example, if you define a private network called “BonFIRE WAN” in your experiment descriptor, then define a compute with IP dependencies both to the public BonFIRE WAN and the private BonFIRE WAN, then this will not work.

Contextualisation Element

Contextualisation elements, also known as contextualisation variables, can be used to pass initialisation values for an experiment. They are defined as simple name-value pairs. For example:

```
"compute": {
    "name": "Client",
    ...
    "contexts": [
        {
            "experimentmin": "0"
        },
        {
            "experimentmax": "100"
        }
    ]
}
```

This would create parameters EXPERIMENTMIN=0 and EXPERIMENTMAX=100 in the file /etc/bonfire on the created virtual machine.

Ordering

The order in which the VM should be deployed can be defined by listing sets of computes in the order of deployment. All of the VMs in the first group will be deployed before the VMs in the next group.

```
"order": [ [ "Server" ], "created", [ "Client", "OtherClient" ] ]
```

The option to wait until the prior VMs are “running” rather than just “created” will be added in a later version.

Storage

Datablock storage resources can be created as follows:

```
"storage": {
    "name": "Storage block",
    "description": "An extra storage block.",
    "type": "DATABLOCK",
    "size": 512,
    "fstype": "ext3"
}
```

And then referenced from a compute like this:

```
"compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
    "resources": [
        { "storage": "Storage block" },
        { "storage": "@BonFIRE Debian Squeeze v5" },
        { "network": "@BonFIRE WAN" }
    ]
}
```

Or datablocks can be defined within a compute like this:

```
"compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
    "resources": [
        {
            "storage": {
                "name": "Storage block",
                "description": "A place to store the log files of the server.",
                "type": "DATABLOCK",
                "size": 1024,
                "fstype": "ext3"
            }
        },
        {"storage": "@BonFIRE Debian Squeeze v5" },
        {"network": "@BonFIRE WAN" }
    ]
}
```

Normally, storages created this way only exist for the lifetime of the compute that they are attached to. By specifying “`save_as`” on a storage, the storage block will be saved when the compute is shut down. For example:

```
"compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
    "resources": [
        {
            "storage": {
                "name": "Storage block name",
                "type": "DATABLOCK",
                "size": 1024,
                "fstype": "ext3",
                "save_as": "Storage block name"
            }
        }
    ]
}
```

```
        "description": "A place to store the log files of the server.",
        "type": "DATABLOCK",
        "size": 1024,
        "fstype": "ext3"
    },
    "save_as": "my saved storage"
},
{"storage": "@BonFIRE Debian Squeeze v5"},
 {"network": "@BonFIRE WAN"}
```

Storages can also be created as persistent. This means that the storage is created and made available before the compute is created, and then attached. For example:

```
"compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
    "resources": [
        {
            "storage": {
                "name": "Storage block name",
                "description": "A place to store the log files of the server.",
                "type": "DATABLOCK",
                "size": 1024,
                "fstype": "ext3",
                "persistence": "YES"
            }
        },
        {"storage": "@BonFIRE Debian Squeeze v5"},
        {"network": "@BonFIRE WAN"}
```

As the experiment manager creates all its resources inside an experiment, when the experiment is deleted all its resources are deleted. This includes any persistent storages created in the experiment. To prevent this, a storage can be created with a “delete_with_experiment” field. By default this is “YES”. If this is supplied as “NO”, the storage is created as a resource outside the experiment and not deleted with the experiment.

For example:

```
"compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
    "resources": [
        {
            "storage": {
                "name": "Storage block name",
                "description": "A place to store the log files of the server.",
                "type": "DATABLOCK",
                "size": 1024,
                "fstype": "ext3",
                "persistence": "YES",
                "delete_with_experiment": "NO"
            }
        },
        {"storage": "@BonFIRE Debian Squeeze v5"},
        {"network": "@BonFIRE WAN"}
```

Multiple instances of a VM

Setting the minimum number of instances of a VM to create will create the given number of instances of it. In the example below, three clients will be produced; all attached to the same network and based on the same storage OS image. These will be named Client1, Client2, Client3. If a storage or network is created along with the compute, a new one will be created for each compute.

```
"compute": {
    "name": "Client",
    "description": "A description of the client.",
    "instanceType": "small",
    "locations": ["fr-inria"],
        "min": 3,
    "resources": [
        { "storage": "@BonFIRE Debian Squeeze v5" },
        { "network": "@BonFIRE WAN" }
    ]
}
```

Controlled networks

Custom controlled networks can be created and used within an experiment. The lossrate, bandwidth and latency parameters can be set to simulate slow or noisy networks. The following experiment descriptor shows an example of creating a custom network and using it within a compute.

```
{
  "name": "controlled network experiment",
  "description": "Example of using a controlled network",
  "duration": 120,
  "resources": [
    {
      "network": {
        "name": "customnetwork",
        "locations": [
          "be-ibbt"
        ],
        "address": "192.168.1.1",
        "size": "C",
        "lossrate": 1.0,
        "bandwidth": 700,
        "latency": 21
      }
    },
    {
      "compute": {
        "name": "networkCompute",
        "locations": [
          "be-ibbt"
        ],
        "instanceType": "Large-EN",
        "min": 1,
        "resources": [
          {
            "storage": "@BonFIRE Debian Squeeze 2G v5"
          },
          {
            "network": "@BonFIRE WAN"
          }
        ]
      }
    }
  ]
}
```

```
        },
        {
          "network": "customnetwork"
        }
      ],
      "contexts": []
    }
  }
]
```

Some sites such as IBBT allow further parameters for controlling background traffic simulation. These are protocol, packetsize and throughput. Please note that all three parameters must be specified together if used. Please see :doc:`/networking/virtual-wall` for more information.

Custom computes

As well as using supplied compute instance types, custom compute instances may be defined. These instances allow memory, number of physical cpus and number of virtual cpus to be explicitly defined. The following example shows a compute defining 1 physical cpu, 2 virtual cpus and 256 MB of memory.

```
"compute": {
  "name": "CustomClient",
  "description": "A description of the client.",
  "instanceType": "custom",
  "cpu": "1",
  "vcpu": "2.0",
  "memory": "256",
  "locations": ["fr-inria"],
  "resources": [
    {"storage": "@BonFIRE Debian Squeeze v5"},
    {"network": "@BonFIRE WAN"}
  ]
}
```

Host and Cluster

Computes may be requested to run on specific hosts and clusters by supplying values for host and cluster. Please see :doc:`/compute/compute-resources-bonfire` for further information.

```
"compute": {
  "name": "Client1",
  "description": "A description of the client.",
  "instanceType": "small",
  "cluster": "5ff33cb5ad0e64302ef64f55a08817d818780b81",
  "locations": ["uk-epcc"],
  "resources": [
    {"storage": "@BonFIRE Debian Squeeze v5"},
    {"network": "@BonFIRE WAN"}
  ]
}

"compute": {
  "name": "Client2",
  "description": "A description of the client.",
  "instanceType": "small",
```

```

        "host": "node-1.bonfire.grid5000.fr",
        "locations": ["fr-inria"],
        "resources": [
            {"storage": "@BonFIRE Debian Squeeze v5"},
            {"network": "@BonFIRE WAN"}
        ]
    }
}

```

Aggregators

Aggregator computes can be used to monitor information about other computes within the same experiment. There are two ways to create aggregators: through the “aggregator” keyword; and manually.

Aggregator keyword

In release 4 we have introduced a shortcut for creating monitoring experiments via the experiment manager. Instead of supplying the information below, it is now possible to supply a “aggregator” keyword in an experiment descriptor, and a BonFIRE-Monitor compute of instance type “small” is automatically created, using the most recent version of the Zabbix image available at the specified site. A persistent storage block of size 1024MB, named “Aggregator Storage For Managed Experiment [managed experiment id]”, is automatically created for the BonFIRE-Monitor compute. This is created outside (before) the experiment, so that results are retained beyond the lifetime of the experiment. Also, the aggregator_ip dependencies and usage information are automatically added to all VMs in the experiment during deployment.

In the following example, an Aggregator is created at EPCC, and a “metricsclient” compute is created at fr-inria, with a user-defined metric “users” (for more detail on user metrics see below).

```

{
    "name": "aggregator keyword example",
    "description": "automatically creates a BonFIRE-Monitor aggregator",
    "duration": 600,
    "aggregator": "uk-epcc",
    "resources": [
        {
            "compute": {
                "name": "metricsclient",
                "locations": [ "fr-inria" ],
                "instanceType": "lite",
                "min": 1,
                "resources": [
                    {
                        "storage": "@BonFIRE Debian Squeeze v5"
                    },
                    {
                        "network": "@BonFIRE WAN"
                    }
                ],
                "script": [
                    {
                        "script": "useradd -m -s /bin/bash $user"
                    },
                    {
                        "script": "echo $user:$password | chpasswd"
                    },
                    {
                        "script": "apt-get update"
                    },
                    {
                        "script": "apt-get install -y curl"
                    },
                    {
                        "script": "curl -s https://raw.githubusercontent.com/bonfire-project/bonfire-metrics-client/master/install.sh | bash"
                    }
                ]
            }
        }
    ]
}

```

```
"contexts": [
    {
        "metrics": [
            "users,wc -l /etc/passwd|cut -d\" \" -f1, rate=20, valuetype=3, history=10",
            "system.test, who|wc -l, history=25, valuetype=3"
        ]
    }
]
```

Manual Aggregator Creation

Aggregator computes can be used to monitor information about other computes within the same experiment. Aggregator computes must be created from an appropriate aggregator image containing a Zabbix installation. Aggregators and aggregator clients are created in a similar way to other computes, but some additional usage information must be supplied.

The following example shows an aggregator compute which monitors a client compute. The aggregator is also able to monitor itself. Please note that BonFIRE expects aggregator computes to be named “BonFIRE-Monitor”.

```
{
    "name": "AggregatorExperiment",
    "description": "example aggregator with client",
    "duration": 60,
    "resources": [
        {
            "compute": {
                "name": "BonFIRE-Monitor",
                "locations": [
                    "de-hlrs"
                ],
                "instanceType": "small",
                "min": 1,
                "resources": [
                    {
                        "storage": "@BonFIRE Zabbix Aggregator v7"
                    },
                    {
                        "network": "@BonFIRE WAN"
                    }
                ],
                "contexts": [
                    {
                        "usage": "zabbix-agent;infra-monitoring-init"
                    }
                ]
            }
        }
    ],
    "contexts": [
        {
            "usage": "zabbix-agent;infra-monitoring-init"
        }
    ]
}
```

```
{
  "compute": {
    "name": "aggregatorClientHP",
    "locations": [
      "uk-hplabs"
    ],
    "instanceType": "lite",
    "min": 1,
    "resources": [
      {
        "storage": "@BonFIRE Debian Squeeze v5"
      },
      {
        "network": "@Internet@HP"
      }
    ],
    "contexts": [
      {
        "aggregator_ip": [
          "BonFIRE-Monitor",
          "BonFIRE WAN"
        ]
      },
      {
        "usage": "zabbix-agent"
      }
    ]
  }
}
]
```

In the BonFIRE-Monitor compute, note the usage information specifying that this compute is monitoring its own usage (zabbix-agent); and that it also fetches infrastructure monitoring metrics for all physical hosts on which VMs of this experiment are running (infra-monitoring-init):

```
"contexts": [
  {
    "usage": "zabbix-agent;infra-monitoring-init"
  }
]
```

An additional option when creating an aggregator compute is to allow it to save its monitoring data to an additional disk attached to the compute.

```
"contexts": [
  {
    "usage": "zabbix-agent;zabbix-aggr-extend;infra-monitoring-init"
  }
]
```

In this case a second storage must be specified in the BonFIRE-Monitor compute description, for example (assuming that a storage called ExternalStorageBlock already exists):

```
"resources": [
  {
    "storage": "@BonFIRE Zabbix Aggregator v7"
  },
  {
    "storage": "@ExternalStorageBlock"
  }
]
```

```
{  
    "network": "@BonFIRE WAN"  
}  
]
```

Please note that in R3.1 it is not possible to directly specify creation of a persistent storage block from inside an experiment descriptor. Persistent storage blocks may be created directly via the BonFIRE portal then referenced as in the example above. Alternatively, once an experiment is running, it is possible to select the BonFIRE-Monitor compute in the portal, and manually do a “save_as” on the second disk.

In the client compute, there are two things to note. The first is that the computes must declare an IP dependency called “aggregator_ip”. This allows the computes to obtain the address of the Zabbix monitoring service to which they will publish information. The second is that the usage information declaring that this compute will publish monitoring information.

```
"contexts": [  
    {  
        "aggregator_ip": [  
            "BonFIRE-Monitor",  
            "BonFIRE WAN"  
        ]  
    },  
    {  
        "usage": "zabbix-agent"  
    }  
]
```

User metrics

User defined metrics to be collected by an aggregator may be defined in a compute description. The metrics must be passed as an array called “metrics”, with a separate string for each metric. The strings are automatically placed in CDATA sections to handle characters which may confuse XML parsers. However, any characters which may confuse string processing must be escaped, as in the example below.

```
"contexts": [  
    {  
        "metrics": [  
            "users,wc -l /etc/passwd|cut -d\" \\" -f1, rate=20, valuetype=3, history=10",  
            "system.test, who|wc -l, history=25, valuetype=3"  
        ]  
    }  
]
```

Bandwidth on demand

Bandwidth on demand resources can be created by specifying site link resources and networks which use the site links. Please see :doc:`/networking/autobahn` for further information.

A site link is specified by declaring a site link resource. Currently a site link may only be used between EPCC and PSNC. Bandwidth is specified in megabits per second.

```
"site_link": {  
    "name": "mySiteLink",  
    "description": "autobahn site link",
```

```

    "locations": [ "autobahn" ],
    "endpoints": [ "uk-epcc", "pl-psnc" ],
    "bandwidth": 100
}

```

To create a network using a site link, the network must declare a “vlan” dependency referring to the appropriate site link within the same experiment descriptor. The vlan dependency must contain “vlanFromSiteLink” and the name of the site link. The experiment manager will automatically wait for the site link to become active, then it will obtain the vlan identifier from the site link and use this to create the network.

```

"network": {
    "name": "myEpccAutoBahnNetwork",
    "description": "epcc link to autobahn",
    "locations": [ "uk-epcc" ],
    "address": "192.168.4.0",
    "netmask": "255.255.255.0",
    "vlan": { "vlanFromSiteLink": "mySiteLink" }
}

```

Alternatively, if the vlan identifier for an existing site link is known, then this can be used directly in the experiment descriptor.

```

"network": {
    "name": "myEpccAutoBahnNetwork",
    "description": "epcc link to autobahn",
    "locations": [ "uk-epcc" ],
    "address": "192.168.4.0",
    "netmask": "255.255.255.0",
    "vlan": "10"
}

```

The network can subsequently be used in a compute as per normal:

```

"compute": {
    "name": "server",
    "description": "server at epcc",
    "instanceType": "small",
    "locations": [ "uk-epcc" ],
    "resources": [
        { "storage": "@BonFIRE Debian Squeeze v5" },
        { "network": "myEpccAutoBahnNetwork" },
        { "network": "@BonFIRE WAN" }
    ],
    "contexts": [
        { "iproute": "192.168.3.0/24 dev eth1" }
    ]
}

```

FEDERICA routers

Routers and networks can be created at FEDERICA by defining router resources in an experiment descriptor. Please read the FEDERICA documentation at :doc:`/networking/federica` for further information. Multiple interfaces may be specified per router. Optional additional configuration for each router can supplied in a “config” element.

```

"router": {
    "name": "RouterDFN",
    "host": "dfn.erl.router1",
    "locations": [ "federica" ],

```

```
"interfaces": [
    {
        "name": "ifDFN",
        "physicalInterface": "ge-0/1/0",
        "ip": "192.168.10.10",
        "netmask": "255.255.255.0"
    }
],
"config": "optional configuration information"
}
```

To specify a network based on FEDERICA router end points, a special form of network resource must be specified. The network must contain an array of networkLinks, where each element contains a router name and the name of an interface specified within that router.

```
"network": {
    "name": "myFedericaNetwork",
    "description": "federica network",
    "locations": ["federica"],
    "networkLinks": [
        { "router": "RouterDFN", "interface": "ifDFN" },
        {"router": "RouterPSNC", "interface": "ifPSNC"}
    ]
}
```

Note that this network cannot be used directly within a compute. A further network must be defined which declares the FEDERICA network as a dependency. This must include a vlan dependency, specifying “vlanFromNetwork” and the name of the Federica network. The experiment manager will automatically create the router and network at FEDERICA and wait for them to become active. Once they are active, the experiment manager will obtain the vlan identifier for the network and pass this into the new network at creation time. Alternatively, if the vlan identifier for an existing FEDERICA network is already known, then this can be specified directly in the experiment descriptor.

```
"network": {
    "name": "epccNetwork",
    "description": "epccNetwork",
    "locations": ["uk-epcc"],
    "address": "192.168.1.0/24",
    "netmask": "255.255.255.0",
    "vlan": { "vlanFromNetwork": "myFedericaNetwork" }
}
```

Elasticity

A special case of an experiment descriptor can be used to create an elasticity experiment. This allows the number of active computes to be automatically managed based on cpu usage. This experiment descriptor must contain additonal context information. For further information please see :doc:`/elasticity/using-EaaS`

```
{
    "name": "elasticitytest",
    "description": "elasticitytest",
    "duration": 3600,
    "resources": [
        {
            "compute": {
                "name": "BonFIRE-Monitor",
                "locations": [
                    "fr-inria"
                ]
            }
        }
    ]
}
```

```

        ],
        "instanceType": "small",
        "min": 1,
        "resources": [
            {
                "storage": "@BonFIRE Zabbix Aggregator v7"
            },
            {
                "network": "@BonFIRE WAN"
            }
        ]
    }
},
{
    "compute": {
        "name": "BonFIRE-Elasticity-Engine-webservice",
        "locations": [
            "fr-inria"
        ],
        "instanceType": "small",
        "min": 1,
        "resources": [
            {
                "storage": "@BonFIRE Debian Squeeze v5"
            },
            {
                "network": "@BonFIRE WAN"
            }
        ],
        "contexts": [
            {
                "aggregator_ip": [ "BonFIRE-Monitor", "BonFIRE WAN" ]
            },
            {
                "usage": "elasticity-engine"
            },
            {
                "ELASTICITY_TRIGGER_UPSCALE_EXPRESSION": "{system.cpu.usage.last(0)}>60"
            },
            {
                "ELASTICITY_VMGROUP_MAX": "3"
            },
            {
                "AGGREGATOR_USER": "Admin"
            },
            {
                "ELASTICITY_INSTANCETYPE": "lite"
            },
            {
                "ELASTICITY_LB_SCHEME": "HAProxy"
            },
            {
                "ELASTICITY_VMGROUP_MIN": "1"
            },
            {
                "ELASTICITY_LB_PORT": "80"
            },
            {

```

```
        "ELASTICITY_LB_LOCATION": "/locations/fr-inria"
    },
{
    "ELASTICITY_VMGROUP_NAME": "webservice"
},
{
    "AGGREGATOR_PASSWD": "zabbix"
},
{
    "ELASTICITY_DISKSOURCE": "/locations/fr-inria/storages/1333"
}
]
}

}
]
}
```

Comments

The JSON format does not itself support comments. However, we thought it might be useful to support comments for documentation within BonFIRE experiment descriptors, particularly for complicated scenarios. Therefore we have defined a “comment” element.

```
"comment": "some helpful and illuminating comment"
```

These may be used throughout an experiment descriptor.

4.2.4 Creating a Managed Experiment From the GUI

The following is a step by step process for creating an experiment descriptor and then submitting it to create an experiment.

1. Log onto the BonFIRE Portal using your BonFIRE credentials.
2. Press the Managed Experiments button:

3. Press the Create Managed Experiment button.

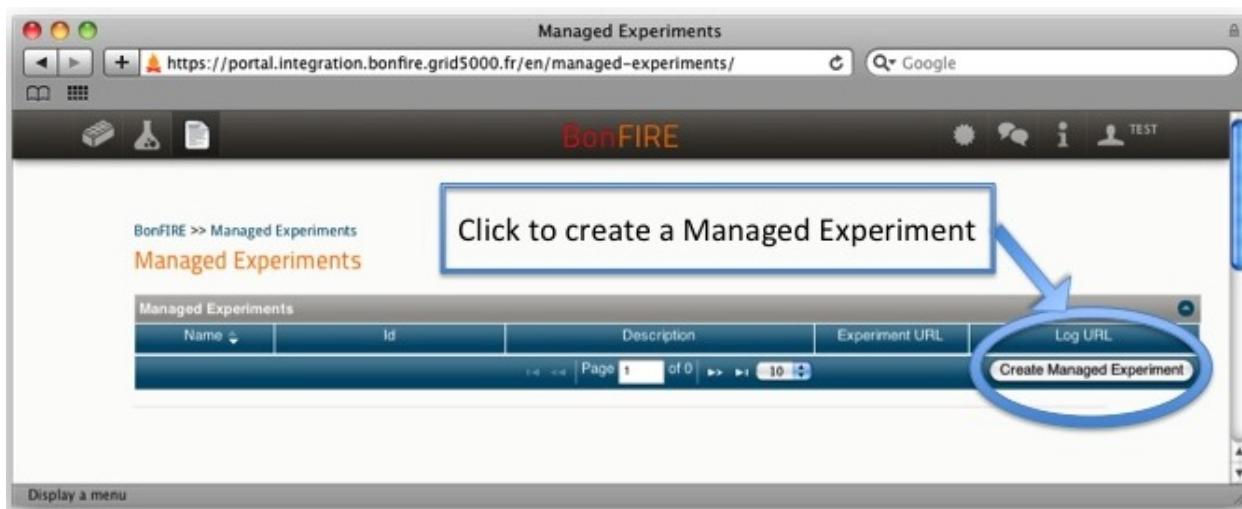
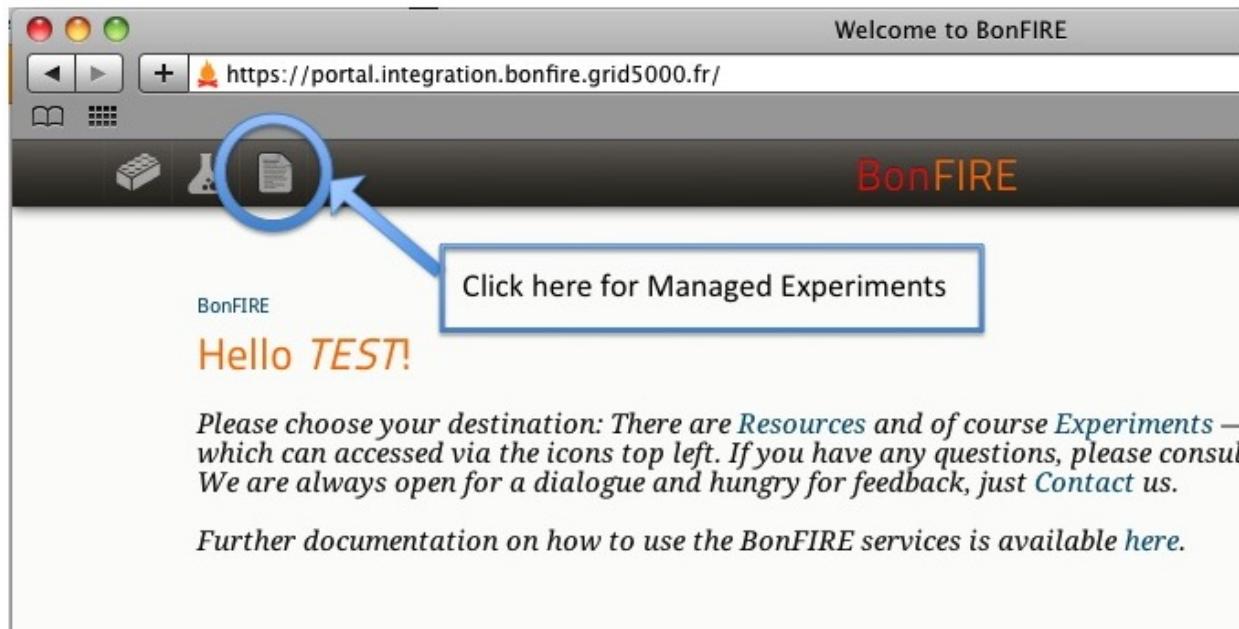
4. Fill in the experiment details and press Continue.

5. Select a Compute, Network or Storage resource to create.

6. Fill in the resource details and press Continue.

7. When all of the resources have been added, select Submit experiment descriptor and press Continue to view the experiment descriptor .

8. View (and optionally edit) the experiment descriptor. Press Finish.



The screenshot shows a web browser window titled 'Create' with the URL <https://portal.integration.bonfire.grid5000.fr/en/managed-experiments/create>. The page is titled 'BonFIRE' and displays the 'Create a Managed Experiment' form. The form fields are as follows:

- Name : A managed experiment
- Description : just testing
- Walltime (Lifetime in Minutes) : 120

At the bottom of the form are three buttons: 'Back', 'Continue' (highlighted in blue), and 'Finish'.

The screenshot shows the same web browser window continuing the process. The title bar still says 'Create'. The main content area now displays the question 'How would you like to proceed?'. There are two options:

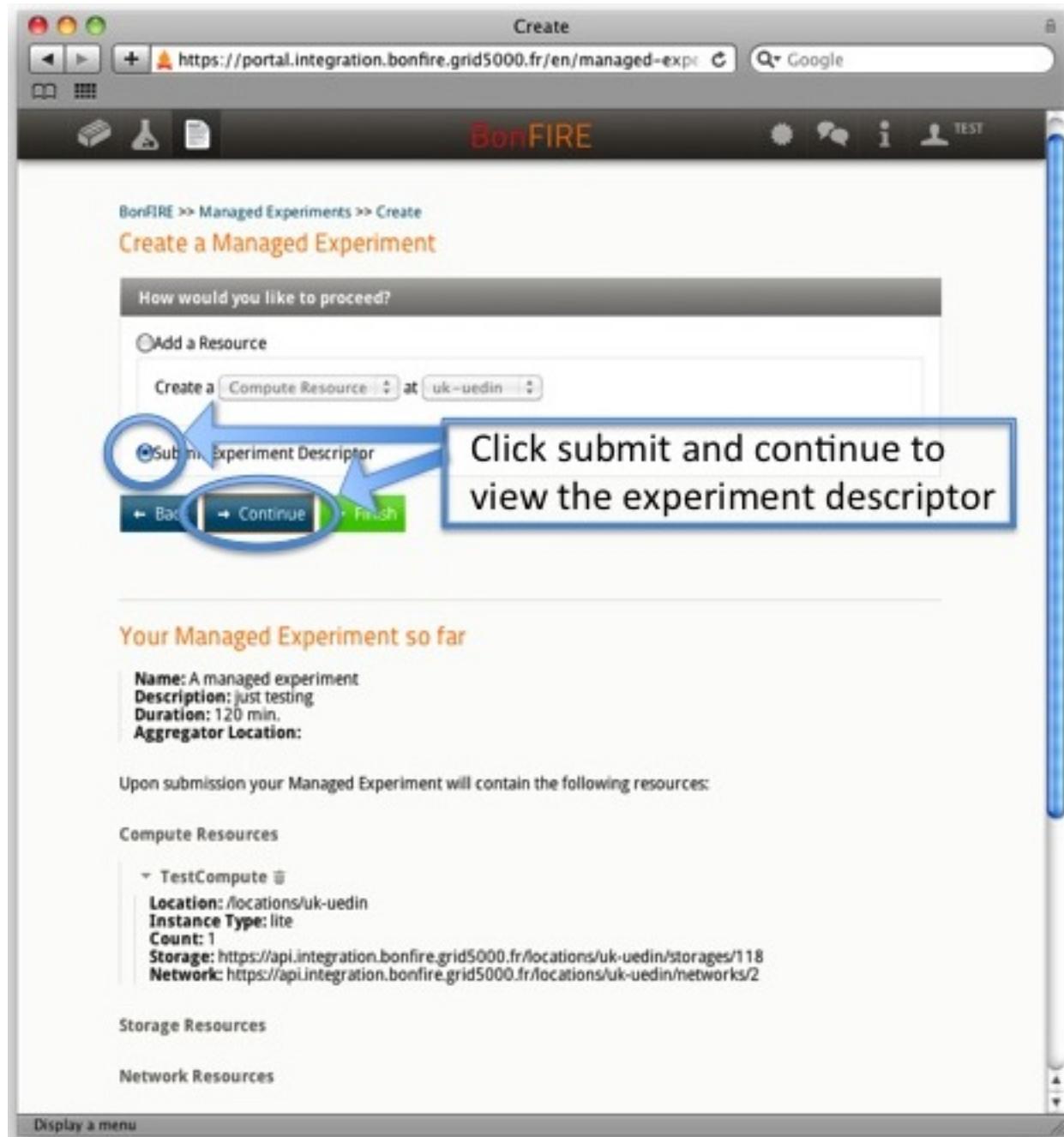
- Add a Resource
Create a Compute Resource at uk-uedin
- Submit Experiment Descriptor

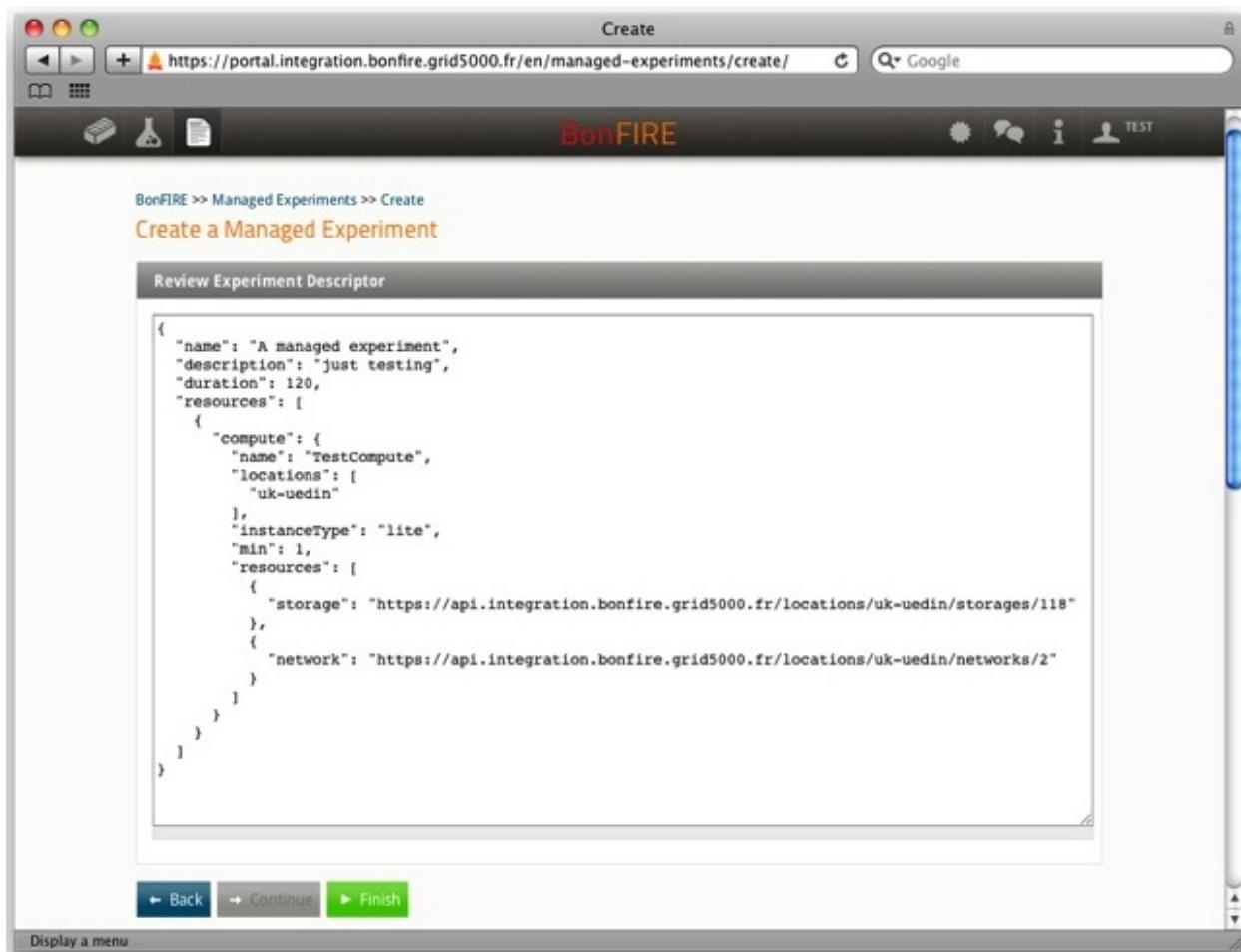
At the bottom of the screen are the same three buttons: 'Back', 'Continue' (highlighted in blue), and 'Finish'.

The screenshot shows a web browser window titled 'Create' with the URL <https://portal.integration.bonfire.grid5000.fr/en/managed-experiments/>. The page is titled 'Create a Managed Experiment'. It contains a form for 'Add a Compute Resource' with the following fields:

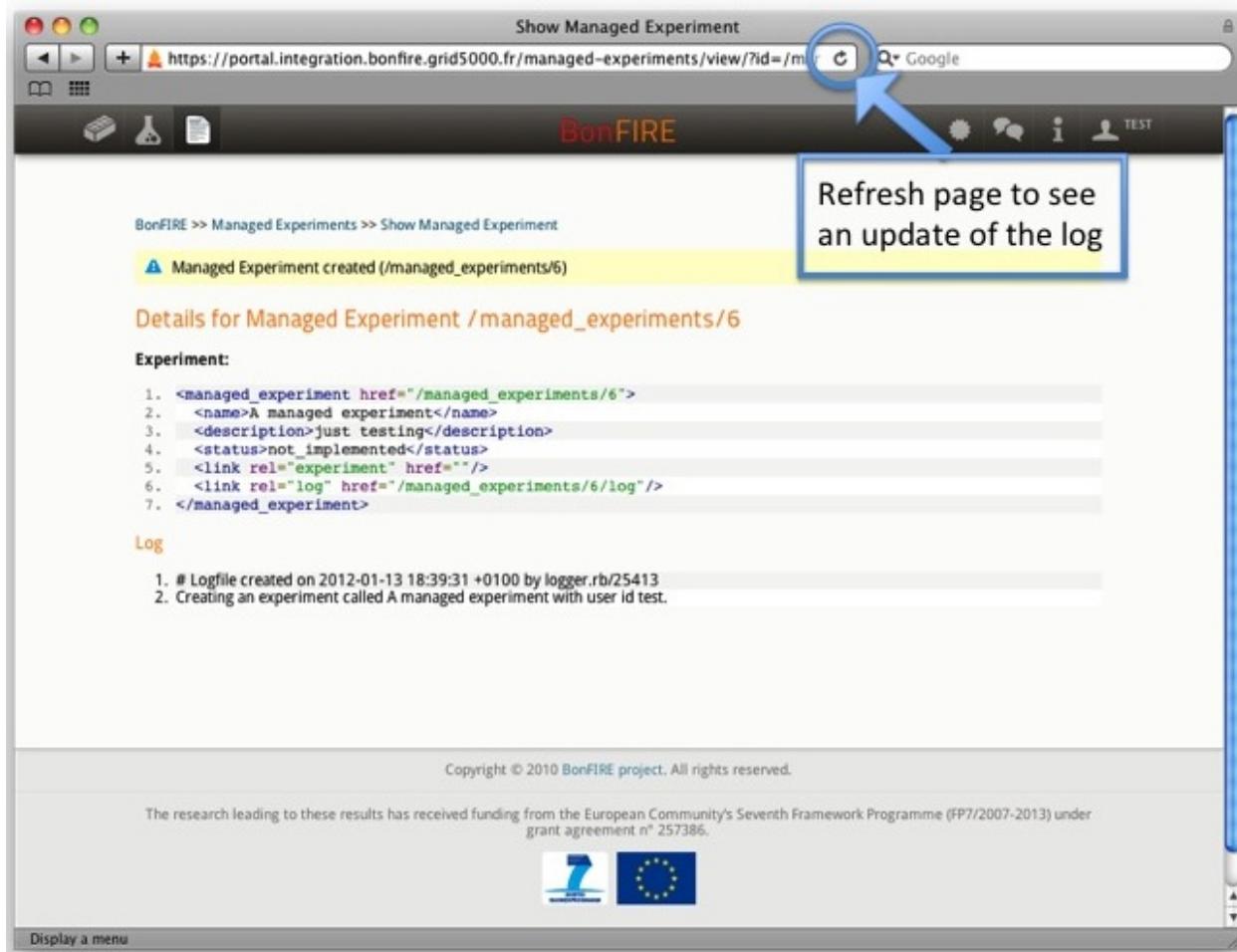
- Name: TestCompute
- Instance Type: lite (cpus: 0.25, memory: 256MiB)
- VMImage: BonFIRE Debian Squeeze v3
- Storage: (empty dropdown)
- Network: BonFIRE WAN
- Count: 1

At the bottom of the form are buttons for Back, Continue, and Finish.





9. View the managed experiment. Refresh the page to see an update of its log file.

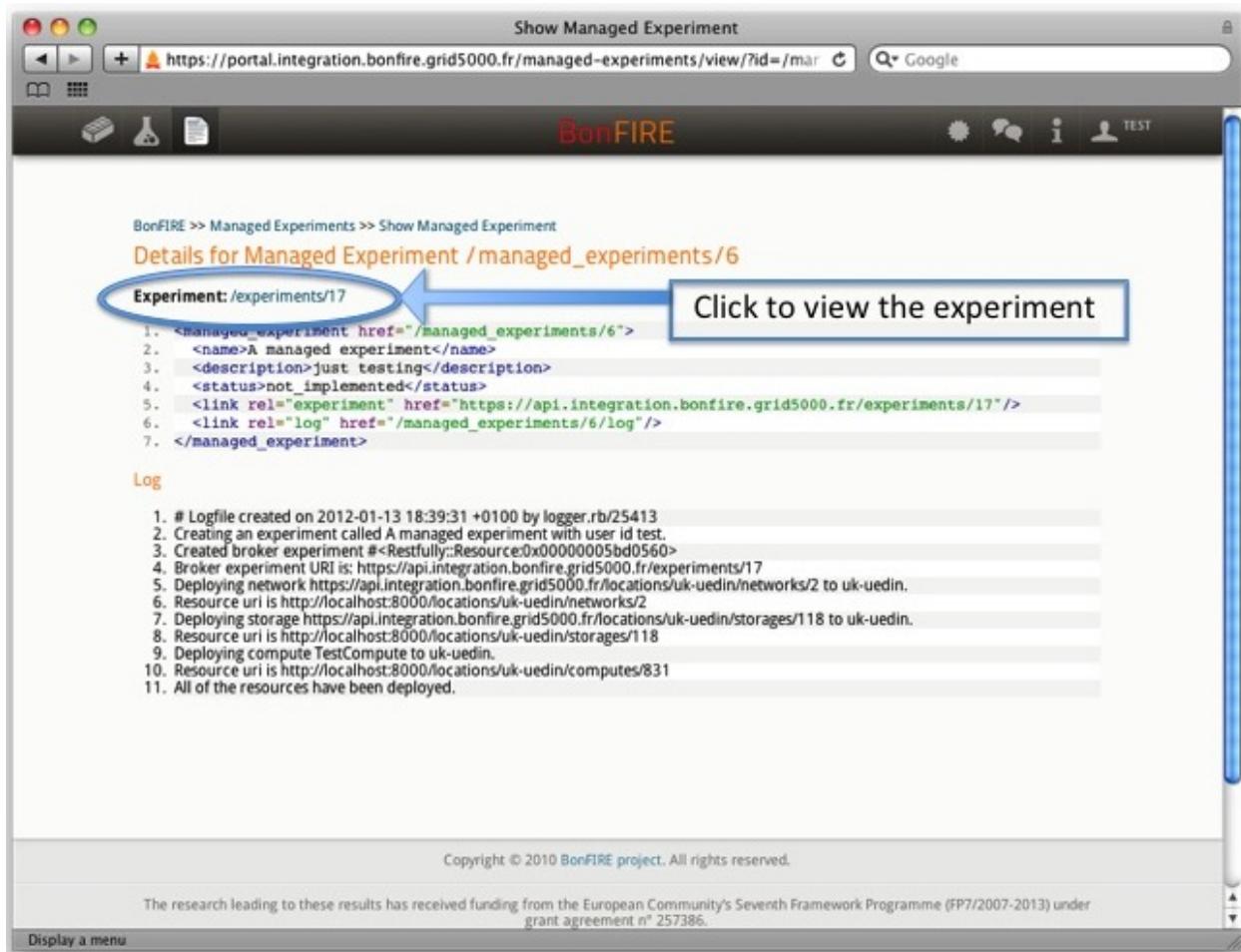


10. The log will list the resources that have been deployed. Click the Experiment link to see the details of the deployed experiment.

11. The experiment page.

Amazon (AWS) Support

Currently, the Experiment Descriptor builder component of the BonFIRE portal only supports specifying the keys necessary for creating an AWS enabled experiment but does not support adding AWS resources. As a temporary workaround when AWS support is required for a Managed Experiment, AWS resources can be specified manually in the Experiment Descriptor. To view and edit the Experiment Descriptor before it is submitted click “View ED”.



The screenshot shows the 'Experiment Details' page for a managed experiment. At the top, it displays the experiment ID (/experiments/17), user ID (test), creation time (Fri, 01/13/12 17:39:34 UTC), last updated time (Fri, 01/13/12 17:39:34 UTC), and expiration time (Fri, 01/13/12 19:39:34 UTC). Below this, there are three main sections: 'Compute Resources', 'Storage Resources', and 'Network Resources'. Each section contains a table with columns for Name, Id, Type, Cpus, Mem, VM Image, Wan IP, SSH, State, and Del. The 'Compute Resources' table has one entry: 'TestCompi /locations/uk-uedin/computes/831' with an 'OK' status and 'RUNNING' state. The 'Storage Resources' and 'Network Resources' tables are currently empty. A 'Monitoring' section below indicates 'No monitoring aggregator found for this experiment.' At the bottom, a 'Display a menu' button is visible.

4.2.5 Command Line Example of Using the Experiment Manager

See BonFIRE API first step for an introduction to using cURL to interact with the BonFIRE API. The authentication for the Experiment Manager is the same as the BonFIRE API. Please note that when creating a managed experiment, a group identifier must be supplied in the header using X-BONFIRE-ASSERTED-SELECTED-GROUP. In the examples a group “user” is used; please use an appropriate group when you run the examples.

Create a managed experiment

```
$ curl -k -i https://api.bonfire-project.eu/managed_experiments
--user BONFIRE_USER -H "Content-Type: application/json"
--header "X-BONFIRE-ASSERTED-SELECTED-GROUP:group"
--data '{
  "description": "Experiment description", "duration": 120, "name": "My Experiment",
  "resources": [{"compute": {"name": "JSON-COMPUTE-TEST",
    "description": "A description of the compute.", "instanceType": "small",
    "locations": ["fr-inria"], "resources": [
      {"storage": "@BonFIRE Debian Squeeze v3"}, {"network": "@BonFIRE WAN"}]}]}]'
```

Note: This command should be entered all on one line.

Alternatively, enter the json into a text file, then use curl to submit it:

ED.json

```
{
  "name": "My Experiment",
  "description": "Experiment description",
  "duration": 120,
  "resources": [
    {
      "compute": {
        "name": "Server",
        "description": "A description of the server.",
        "instanceType": "small",
        "locations": ["uk-epcc"],
        "resources": [
          {"storage": "@BonFIRE Debian Squeeze v3"}, {"network": "@BonFIRE WAN"}
        ]
      }
    },
    {
      "compute": {
        "name": "Client",
        "description": "A description of the client.",
        "instanceType": "small",
        "locations": ["fr-inria"],
        "resources": [
          {"storage": "@BonFIRE Debian Squeeze v3"}, {"network": "@BonFIRE WAN"}
        ],
        "contexts": [
          {
            "ServerIP": [ "Server", "BonFIRE WAN" ]
          }
        ]
      }
    }
  ]
}
```

```
        }
    ]
}
```

Submit using curl

```
$ curl -k -i https://api.bonfire-project.eu/managed_experiments
--header "X-BONFIRE-ASSERTED-SELECTED-GROUP:group"
--user BONFIRE_USER -H "Content-Type: application/json" --data @ED.json
```

This is what you should see in response.

```
HTTP/1.1 201 Created
Date: Tue, 14 Aug 2012 09:46:15 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Location: https://api.bonfire-project.eu/managed_experiments/680
Content-Type: application/xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 3.106216
Set-Cookie: _experimentManager_session=BAh7Bkkid3Nlc3Npb25faWQGogZFRiIlZDkwYTJiNTM2ODc3YTQ2NTUzM2U3OD
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?><managed_experiment href="/managed_experiments/680"><name>My Ex
```

View the managed experiment

```
$ curl -k -i https://api.bonfire-project.eu/managed_experiments/680 --user BONFIRE_USER
```

Response:

```
HTTP/1.1 200 OK
Date: Tue, 14 Aug 2012 09:47:16 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Location: https://api.bonfire-project.eu/managed_experiments/680
Content-Type: application/xml; charset=utf-8
ETag: "d9988ac4dd3848dc9f14004aa6687a56"
Cache-Control: max-age=0, private, must-revalidate
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.945418
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?><managed_experiment href="/managed_experiments/680"><name>My Ex
```

View the managed experiment log file

Successfully creating a managed experiment in the first step does not guarantee that all of the resources will be deployed. To check that they have been you should read the log file and check that there are no errors reported and that the experiment status is reported as “DEPLOYED”.

```
$ curl -k -i https://api.bonfire-project.eu/managed_experiments/680/log --user BONFIRE_USER
```

Response:

```
["2012-08-14 09:46:18:605 UTC: STATUS QUEUED\n",
"2012-08-14 09:46:18:607 UTC: STATUS DEPLOYABLE\n",
"2012-08-14 09:46:18:723 UTC: STATUS DEPLOYING\n",
"2012-08-14 09:46:18:723 UTC: Creating an experiment called My Experiment with user id malcolm.\n",
"2012-08-14 09:46:18:882 UTC: Created broker experiment\n",
"2012-08-14 09:46:19:134 UTC: Broker experiment URI is: https://api.bonfire-project.eu/experiments/12
"2012-08-14 09:46:19:134 UTC: Processing resource BonFIRE WAN\n",
"2012-08-14 09:46:19:134 UTC: Starting to deploying network BonFIRE WAN to fr-inria.\n",
"2012-08-14 09:46:19:191 UTC: Looking up network BonFIRE WAN\n",
"2012-08-14 09:46:21:532 UTC: Deployed resource uri is http://localhost:8000/locations/fr-inria/networks/BonFIRE WAN
"2012-08-14 09:46:21:532 UTC: Processing resource BonFIRE WAN\n",
"2012-08-14 09:46:21:532 UTC: Starting to deploying network BonFIRE WAN to uk-epcc.\n",
"2012-08-14 09:46:21:732 UTC: Looking up network BonFIRE WAN\n",
"2012-08-14 09:46:23:495 UTC: Deployed resource uri is http://localhost:8000/locations/uk-epcc/networks/BonFIRE WAN
"2012-08-14 09:46:23:495 UTC: Processing resource BonFIRE Debian Squeeze v3\n",
"2012-08-14 09:46:23:496 UTC: Starting to deploy storage BonFIRE Debian Squeeze v3 to fr-inria.\n",
"2012-08-14 09:46:23:579 UTC: Looking up storage BonFIRE Debian Squeeze v3.\n",
"2012-08-14 09:46:26:058 UTC: Deployed resource uri is http://localhost:8000/locations/fr-inria/storage/BonFIRE Debian Squeeze v3
"2012-08-14 09:46:26:058 UTC: Processing resource BonFIRE Debian Squeeze v3\n",
"2012-08-14 09:46:26:059 UTC: Starting to deploy storage BonFIRE Debian Squeeze v3 to uk-epcc.\n",
"2012-08-14 09:46:26:153 UTC: Looking up storage BonFIRE Debian Squeeze v3.\n",
"2012-08-14 09:46:28:365 UTC: Deployed resource uri is http://localhost:8000/locations/uk-epcc/storage/BonFIRE Debian Squeeze v3
"2012-08-14 09:46:28:365 UTC: Processing resource Server\n",
"2012-08-14 09:46:28:365 UTC: Starting to deploying compute Server to uk-epcc.\n",
"2012-08-14 09:46:28:475 UTC: Creating compute Server.\n",
"2012-08-14 09:46:31:752 UTC: Deployed resource uri is http://localhost:8000/locations/uk-epcc/computers/Server
"2012-08-14 09:46:31:752 UTC: Processing resource Client\n",
"2012-08-14 09:46:31:752 UTC: Starting to deploying compute Client to fr-inria.\n",
"2012-08-14 09:46:31:839 UTC: Context ServerIP => 172.18.6.149\n",
"2012-08-14 09:46:31:840 UTC: Creating compute Client.\n",
"2012-08-14 09:46:34:269 UTC: Deployed resource uri is http://localhost:8000/locations/fr-inria/computers/Client
"2012-08-14 09:46:34:269 UTC: All of the resources have been deployed.\n",
"2012-08-14 09:46:34:460 UTC: STATUS DEPLOYED\n",
"2012-08-14 09:46:34:621 UTC: A request has been sent to set the experiment status to RUNNING\n
```

Delete a managed experiment

This will also delete the deployed experiment belonging to the managed experiment.

```
$ curl -ki -X DELETE https://api.bonfire-project.eu/managed_experiments/680 --user BONFIRE_USER
```

Response:

HTTP/1.1 202 Accepted
Date: Tue, 14 Aug 2012 10:14:28 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 1.358367
Set-Cookie: _experimentManager_session=BAh7BkkiD3Nlc3Npb25faWQGogZFRiI1YjdkYWU1YmVkOTRmYj1jN2U2M2ZmOW...
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

4.2.6 OVF Experiment Descriptor

Open Virtualization Format (OVF: <http://dmtf.org/standards/ovf>) is used in BonFIRE to specify resources for an initial deployment in a single experiment description.

Standard OVF and BonFIRE Extensions

BonFIRE OVF descriptor uses the standard OVF to specify resources for an initial deployment of an experiment. Although, because the standard OVF is originally designed for self-configuration (customization parameters are known in pre-deployment time) and lacks deployment constraints (e.g. where to deploy a VM, like testbed location), OVF is extended wherever required by BonFIRE according to the OVF extensibility mechanisms proposed by DMTF (through introducing new components, and new attributes for existing components). The following attributes and components are defined as BonFIRE OVF extensions:

New BonFIRE attributes

- walltime: to specify duration of an experiment
- location: to specify location of a BonFIRE testbed site

New BonFIRE components

- InstanceType: to specify a BonFIRE instance type.
- NetworkMap: to map a network definition of OVF onto a concrete BonFIRE testbed site.
- NewNetwork: which is a sub-component of NetworkMap. It is used when a new network will be created.

BonFIRE OVF schema

The BonFIRE OVF schema can be found in OVF schema

Introduction of the OVF sections used in BonFIRE

The following structure and sections of OVF are used currently in BonFIRE:

- Envelope
- References
- DiskSection
- NetworkSection
- VirtualSystemCollection
 - StartupSection
 - VirtualSystemCollection
 - ProductSection
 - VirtualSystem
 - VirtualHardwareSection

Envelope

Envelope is the root element for an OVF document.

References

The file reference part defined by the References element refers the location of an existing storage at a BonFIRE testbed site, e.g.

```
<ovf:References>
  <ovf:File ovf:id="master" ovf:href="https://api.bonfire.grid5000.fr/locations/fr-
    <ovf:File ovf:id="node" ovf:href="https://api.bonfire.grid5000.fr/locations/be-ib-
</ovf:References>
```

DiskSection

A DiskSection describes meta-information about virtual disks.

```
<ovf:DiskSection>
  <ovf:Info> </ovf:Info>
  <ovf:Disk ovf:diskId="master" ovf:capacity="" ovf:fileRef="master"/>
  <ovf:Disk ovf:diskId="node" ovf:capacity="" ovf:fileRef="node"/>
  <ovf:Disk ovf:diskId="master-tmp" ovf:capacity="1024" ovf:fileRef="" ovf:format="ext4">
  <ovf:Disk ovf:diskId="node-tmp" ovf:capacity="1024" ovf:fileRef="" ovf:format="ext4">
</ovf:DiskSection>
```

NetworkSection

The NetworkSection element shall list all logical networks used in OVF.

```
<ovf:NetworkSection>
  <ovf:Info> </ovf:Info>
  <ovf:Network ovf:name="private_net_hlrs"/>
  <ovf:Network ovf:name="private_net_inria"/>
  <ovf:Network ovf:name="private_net_epcc"/>
  <bonfire:NetworkMap ovfNetworkName="private_net_inria" bonfireNetworkRef="BonFIRE">
  <bonfire:NetworkMap ovfNetworkName="private_net_hlrs" bonfireNetworkRef="BonFIRE">
  <bonfire:NetworkMap ovfNetworkName="private_net_epcc" bonfireNetworkRef="BonFIRE">
</ovf:NetworkSection>
```

New created network

In BonFIRE a new network can be created at IBBT.

```
<ovf:NetworkSection>
  <ovf:Info> </ovf:Info>
  <ovf:Network ovf:name="public">
    <ovf:Description>Network to provide administrative access to master</ovf:Description>
  </ovf:Network>
  <ovf:Network ovf:name="private_net, fr-inria">
    <ovf:Description>Private Network</ovf:Description>
  </ovf:Network>
  <ovf:Network ovf:name="private_net, be-ibbt">
    <ovf:Description>Private Network</ovf:Description>
  </ovf:Network>
  <ovf:Network ovf:name="my new network">
    <ovf:Description>Create a new Network</ovf:Description>
  </ovf:Network>
  <bonfire:NetworkMap ovfNetworkName="public" bonfireNetworkRef="https://api.integ-
  <bonfire:NetworkMap ovfNetworkName="private_net, fr-inria" bonfireNetworkRef="Bo-
  <bonfire:NetworkMap ovfNetworkName="private_net, be-ibbt" bonfireNetworkRef="Bon-
  <bonfire:NetworkMap ovfNetworkName="my new network">
    <bonfire:NewNetwork>
      <bonfire:Name>new network at IBBT</bonfire:Name>
      <bonfire:Location>be-ibbt</bonfire:Location>
```

```
<bonfire:Address>192.86.00.00</bonfire:Address>
<bonfire:Size>C</bonfire:Size>
<bonfire:Lossrate>0.5</bonfire:Lossrate>
<bonfire:Bandwidth>100</bonfire:Bandwidth>
<bonfire:Latency>4</bonfire:Latency>
</bonfire:NewNetwork>
</bonfire:NetworkMap>
</ovf:NetworkSection>
```

VirtualSystemCollection

VirtualSystemCollection represents a list of virtual machines, allows one or many virtual machines in different logical grouping, and supports multi-tier applications. This is the upper level VirtualSystemCollection, walltime required for the duration of an experiment is specified here.

```
<ovf:VirtualSystemCollection ovf:id="My Experiment" bonfire:walltime="120">
```

StartupSection

StartupSection specifies the order in which compute resources will be instantiated.

```
<ovf:StartupSection>
  <ovf:Info>Master has to be booted before any Node</ovf:Info>
  <ovf:Item ovf:id="master" ovf:order="0"/>
  <ovf:Item ovf:id="node" ovf:order="1"/>
</ovf:StartupSection>
```

VirtualSystemCollection

This is the low level VirtualSystemCollection which is used to specify the location of a virtual machine.

```
<ovf:VirtualSystemCollection ovf:id="site 1" bonfire:location="fr-inria">
```

VirtualSystem

A virtual machine is a full encapsulation of the virtual hardware, virtual disks, and the metadata associated with it.

ProductSection

ProductSection is used for specifying the required Contextualization information.

```
<ovf:ProductSection ovf:class="" >
  <ovf:Info>contextualization</ovf:Info>
  <ovf:Product>Node</ovf:Product>
  <ovf:Version>0.1</ovf:Version>
  <ovf:Property ovf:key="MasterIP" ovf:type="string" ovf:value="@IP(BonFIRE WAN, 192.86.00.01)" />
</ovf:ProductSection>
```

VirtualHardwareSection

VirtualHardwareSection describes hardware specifications of a virtual machine.

```
<ovf:VirtualHardwareSection>
  <ovf:Info/>
  <ovf:System>
    <vssd:ElementName/>
    <vssd:InstanceID/>
    <bonfire:InstanceType>small</bonfire:InstanceType>
  </ovf:System>
  <ovf:Item>
    <rasd:AutomaticAllocation>true</rasd:AutomaticAllocation>
    <rasd:Connection>private_net, be-ibbt</rasd:Connection>
  </ovf:Item>
</ovf:VirtualHardwareSection>
```

```
<rasd:ElementName>Ethernet adapter on private_net network</rasd:ElementName>
<rasd:InstanceID>3</rasd:InstanceID>
<rasd:ResourceType>10</rasd:ResourceType>
</ovf:Item>
<ovf:Item>
    <rasd:AutomaticAllocation>true</rasd:AutomaticAllocation>
    <rasd:Connection>my new network</rasd:Connection>
    <rasd:ElementName>new created network at IBBT</rasd:ElementName>
    <rasd:InstanceID>3</rasd:InstanceID>
    <rasd:ResourceType>10</rasd:ResourceType>
</ovf:Item>
<ovf:Item>
    <rasd:ElementName>Harddisk 1</rasd:ElementName>
    <rasd:HostResource>ovf://disk/node</rasd:HostResource>
    <rasd:InstanceID>5</rasd:InstanceID>
    <rasd:Parent>4</rasd:Parent>
    <rasd:ResourceType>17</rasd:ResourceType>
</ovf:Item>
<ovf:Item>
    <rasd:ElementName>Harddisk 2</rasd:ElementName>
    <rasd:HostResource>ovf://disk/node-tmp</rasd:HostResource>
    <rasd:InstanceID>5</rasd:InstanceID>
    <rasd:Parent>4</rasd:Parent>
    <rasd:ResourceType>17</rasd:ResourceType>
</ovf:Item>
</ovf:VirtualHardwareSection>
```

4.3 BonFIRE Portal

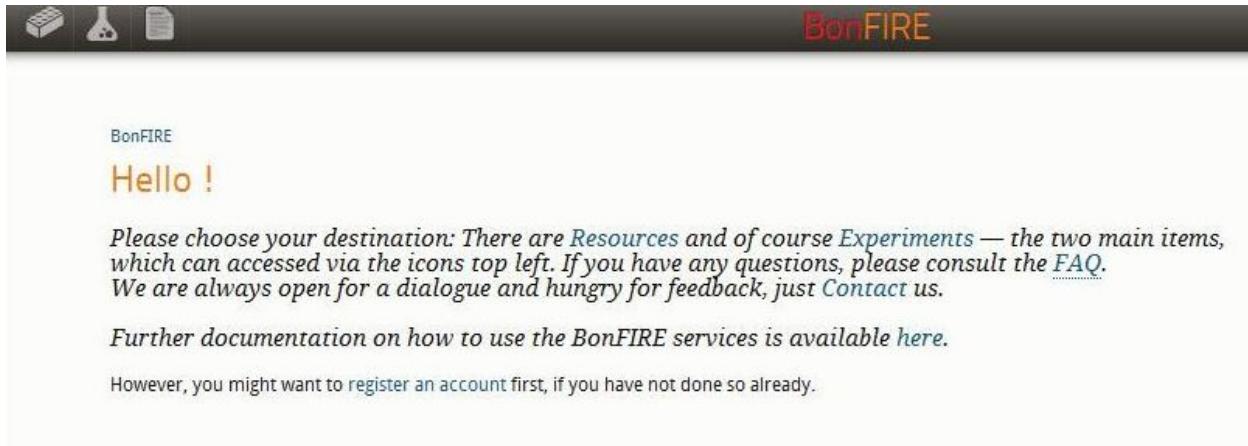
The BonFIRE Portal offers the experimenters a graphical interface to the BonFIRE capabilities, is a simplified entry point for the experimenters, and supports specification of an experiment at both experiment level granularity (through experiment descriptors) and resource level granularity through multiple single resources. The Portal also provides a means of monitoring and has a view of the experimenters' data, the running experiments, the available resources at each testbed site and information for experiment support.

The BonFIRE Portal is available at <https://portal.bonfire-project.eu>.

4.3.1 How to find resources

Available resources in BonFIRE on each infrastructure site can be viewed on the portal. The following is a step by step process for finding resources.

1. log onto the BonFIRE Portal using your BonFIRE credentials.
2. press the button Resources:
3. overview of the BonFIRE resources



Sitewide Resources

Storage Resources								Filter by Site: <all>
Name	Id	Description	Type	Size	FS	Pub.	Per.	Del.
ttylinux	/locations/de-hlrs/storages/817	Zabbix aggregator image created at 2011-11-09 described as a standard Zabbix aggregator image.	OS	40MB		true	false	
BonFIRE Zabbix Aggregator	/locations/de-hlrs/storages/742	Zabbix aggregator image created at 2011-11-09 described as a standard Zabbix aggregator image.		3400MB		true	false	
BonFIRE Zabbix Aggregator	/locations/fr-inria/storages/0	Zabbix aggregator image created at 2011-11-09 described as a standard Zabbix aggregator image.		3400MB		true	false	
BonFIRE Zabbix Aggregator	/locations/uk-epcc/storages/121	Zabbix aggregator image created at 2011-11-09 described as a standard Zabbix aggregator image.		3400MB		true	false	
BonFIRE Zabbix Aggregator	/locations/be-ibbt/storages/2	Zabbix aggregator image created at 2012-01-25 based on the Zabbix 2.0.10 release.	OS	< n/a >			false	
BonFIRE Debian Squeeze v.	/locations/de-hlrs/storages/739	Debian Squeeze created at 2011-09-11, described as a standard Debian Squeeze image.		604MB		true	false	
BonFIRE Debian Squeeze v.	/locations/fr-inria/storages/1	Debian Squeeze created at 2011-06-28, described as a standard Debian Squeeze image.		604MB		true	false	
BonFIRE Debian Squeeze v.	/locations/uk-epcc/storages/118	Debian Squeeze created at 2011-11-24, described as a standard Debian Squeeze image.		604MB		true	false	
BonFIRE Debian Squeeze 2	/locations/de-hlrs/storages/740	Debian Squeeze created at 2011-09-11, with 2G root partition.		2048MB		true	false	
BonFIRE Debian Squeeze 2	/locations/fr-inria/storages/2	Debian Squeeze 2G created at 2011-11-09, described as a standard Debian Squeeze 2G image.		2048MB		true	false	

Page 1 of 2 | Add Storage at de-hlrs | add

Network Resources

Network Resources								Filter by Site: <all>
Name	Id	Address	Bandwidth	Latency	Loss Rate	Public		
Public Network	/locations/fr-inria/networks/27	< n/a >	< n/a >	< n/a >	< n/a >	< n/a >		false
Public IPv4 (EXPERIMENTAL)	/locations/uk-epcc/networks/93	< n/a >	< n/a >	< n/a >	< n/a >	< n/a >		false
BonFIRE WAN	/locations/de-hlrs/networks/90	< n/a >	< n/a >	< n/a >	< n/a >	< n/a >		false
BonFIRE WAN	/locations/fr-inria/networks/0	< n/a >	< n/a >	< n/a >	< n/a >	< n/a >		false
BonFIRE WAN	/locations/uk-epcc/networks/2	< n/a >	< n/a >	< n/a >	< n/a >	< n/a >		false
BonFIRE WAN	/locations/be-ibbt/networks/1	10.0.0.0/16	1000 Mbps		0ms	< n/a >		true

Page 1 of 1 | View 1 - 6 of 6

Sites

Sites						▲	
Name	Id	Instance Types		SSH Gateway	Infrastructure Aggregator		Type
uk-epcc	/locations/uk-epcc	medium, lite, large, xlarge, custom, small		ssh.uk-epcc.bonfire-project.eu	172.18.6.3		ONE
fr-inria	/locations/fr-inria	custom, medium, lite, small		ssh.fr-inria.bonfire-project.eu	131.254.204.19		ONE
de-hlrs	/locations/de-hlrs	xlarge+, custom, large+, large, xlarge, lite, medium, s	< n/a >		172.18.2.128		ONE
be-ibbt	/locations/be-ibbt	Large-EN		ssh.be-ibbt.bonfire-project.eu	< n/a >		VW

Page 1 of 1 | View 1 - 4 of 4

4.3.2 How to create an experiment

We show you here how a new experiment can be created and resources can be added. The following is a step by step process for creating an experiment and adding resources into it.

Creating an experiment container

1. press the button Experiments:



2. press the button Add Experiment:



3. enter information for the experiment, and press Continue:

4. review the details of the experiment data in OCCI format, and click the button Finish to submit it:

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Create Experiment

Name ?
Tutorial

Description ?
used for tutorial

Walltime (Lifetime in Minutes) ?
120

Add a Monitoring Aggregator at this Site ?
don't create an aggregator

Add Infrastructure Metrics ?

→ Continue ► Finish

The screenshot shows a web-based configuration interface for creating an experiment. At the top, a breadcrumb navigation path reads "BonFIRE >> Experiments >> Create Experiment". Below this, the title "Create Experiment" is displayed in orange. A dark grey header bar contains the text "Create Experiment" in white. The main body of the form consists of several input fields and dropdown menus. The "Name" field contains "Tutorial". The "Description" field contains "used for tutorial". The "Walltime (Lifetime in Minutes)" field contains "120". There are two sections for "Monitoring Aggregator": one asking if an aggregator should be created at the site, with "don't create an aggregator" selected, and another asking if infrastructure metrics should be added, with a checked checkbox. At the bottom, there are two buttons: a blue "Continue" button with a right-pointing arrow, and a green "Finish" button with a right-pointing arrow.

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Review OCCI Request for Experiment Creation

```
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Tutorial</name>
  <description>used for tutorial</description>
  <walltime>7200</walltime>
  <status>ready</status>
</experiment>
```

Ignore XML Validation Errors

→ Continue ► Finish

Creating a compute resource

1. select compute resource and location, and click add button on the end of each line to create a compute resource:
2. select instance type and vm image:
3. view the details of the created compute resource:

Note: If the custom instance type is selected, the elements cpu, vcpu, memory shall be added manually into the OCCI request for Compute Creation, and host can be added, but it is optional. The Ignore XML Validation Errors check box shall be activated.

The picture below shows an example with elements: cpu, vcpu, memory and host:

The following figure depicts an example without element host

4.3.3 How to create an experiment

If you want to create an experiment on the Virtual Wall, be sure to add your network resources before you add compute resources. You also have to set the experiment to running by pushing the GO button. More details about creating an

The screenshot shows the 'Experiment Details' page for experiment ID /experiments/3781. It includes sections for Compute Resources, Storage Resources, and Network Resources, each with a table and filtering options. The Compute Resources table has columns: Name, Id, Type, Cpus, Mem, VM Image, Wan IP, SSH, State, Del. The Storage Resources table has columns: Name, Id, Description, Type, Size, FS, Pub., Per., Del. The Network Resources table has columns: Name, Id, Address, Bandwidth, Latency, Loss Rate, Public, Del.

experiment on the Virtual Wall can be found [How to run experiments on Virtual Wall \(ibbt-vw\)](#)

4.4 OCCl/API via HTTP(cURL)

cURL is a command line tool for transferring data with URL syntax, supporting FTP, [...] **HTTP**, **HTTPS**, [...] and TFTP. cURL supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form-based upload, proxies, cookies, user+password authentication (Basic, Digest, NTLM, Negotiate, kerberos...), file transfer resume, proxy tunneling and a busload of other useful tricks.

4.4.1 Installing cURL

MacOS

cURL is installed by default on the MacOS X OS. Nothing to do for you!

Linux

Use your package manager to install cURL.

Either (debian-based distributions):

```
$ sudo apt-get install curl
```

or (redhat-based distributions):

```
$ sudo yum install curl
```

Windows

Download and install the cURL package from the website at <http://curl.haxx.se/download.html>.

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Create Compute Resource

Name ?
vm-test

Instance Type ?
lite (cpus: 0.25, memory: 256MiB)

VMImage ?
BonFIRE Debian Squeeze v3

Storage ?

Network ?
BonFIRE WAN

→ Continue ► Finish

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Review OCCI Request for Compute Creation

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>vm-test</name>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/uk-epcc/storages/118"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
  <nics>
    <network href="/locations/uk-epcc/networks/2"/>
```

Ignore XML Validation Errors

→ Continue ▶ Finish

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Review OCCI Request for Compute Creation

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>custom-vm</name>
  <instance_type>custom</instance_type>
  <cpu>1</cpu>
  <vcpu>1</vcpu>
  <memory>1024</memory>
  <host>floccus06</host>
  <disk>
    <storage href="/locations/uk-epcc/storages/118"/>
    <type>OS</type>
  </disk>
</compute>
```

Ignore XML Validation Errors ←

→ Continue ► Finish

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Review OCCI Request for Compute Creation

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>custom-vm</name>
  <instance_type>custom</instance_type>
  <cpu>1</cpu>
  <vcpu>1</vcpu>
  <memory>1024</memory>
  <disk>
    <storage href="/locations/de-hlrs/storages/739"/>
    <type>OS</type>
  </disk>
</compute>
```

Ignore XML Validation Errors ←

→ Continue ► Finish

4.4.2 Your First Requests

The BonFIRE API entry-point is located at <https://api.bonfire-project.eu/>. Open your Terminal program (or the cURL executable if you're on Windows), and use cURL to fetch the resource located at that URL:

```
$ curl -k -i https://api.bonfire-project.eu/
```

Note: The `-k` flag is necessary to disable the SSL certificate verification (we don't use `-` yet - a trusted SSL certificate).

Hit ENTER to execute the cURL command, and here is what you should see in response:

```
HTTP/1.1 401 Authorization Required
Date: Fri, 08 Jul 2011 08:12:25 GMT
WWW-Authenticate: Basic realm="BonFIRE API"
Content-Length: 489
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Authorization Required</title>
</head><body>
<h1>Authorization Required</h1>
<p>This server could not verify that you
are authorized to access the document
requested. Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
<hr>
<address>Apache/2.2.3 (CentOS) Server at api.bonfire-project.eu Port 444</address>
</body></html>
```

What you see is the HTTP response you received from the server. The first section contains the HTTP headers, and the last section contains the payload (here: `text/html` type). The first line indicates the status of the response, which is 401 Authorization Required. It means that the server requires you to authenticate using your BonFIRE credentials to continue.

Note: The `-i` flag displays the HTTP response headers. If you omit that flag, you will only see the response payload.

Authentication

The BonFIRE API is using the standard [HTTP Basic Authentication](#) mechanism to perform access restriction. You can tell [cURL](#) under which user you want to authenticate using the `-u` flag (replace `BONFIRE_USER` by your BonFIRE username):

```
$ curl -k -i https://api.bonfire-project.eu/ -u BONFIRE_USER
```

cURL will prompt you for your password, and if successful will return the following response:

```
HTTP/1.1 200 OK
Date: Fri, 08 Jul 2011 08:17:55 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Allow: GET,OPTIONS,HEAD
Cache-Control: public,max-age=120
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "4bb22c9cdd9803c44575f93db76e235c"
```

```
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.007996
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/">
  <version>0.11.6</version>
  <timestamp>1310113075</timestamp>
  <link rel="experiments" href="/experiments" type="application/vnd.bonfire+xml"/>
  <link rel="locations" href="/locations" type="application/vnd.bonfire+xml"/>
  <link rel="users" href="/users" type="application/vnd.bonfire+xml"/>
</root>
```

The HTTP status of 200 OK indicates that the server was able to process your request and that everything went fine. You can see that the response is now of type application/vnd.bonfire+xml (the *Media Type*), and that the response payload contains a number of link elements.

Let's fetch the /experiments resource:

```
$ curl -k -i https://api.bonfire-project.eu/experiments -u BONFIRE_USER
```

Here's what you should see:

```
HTTP/1.1 200 OK
Date: Fri, 08 Jul 2011 08:21:31 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Allow: GET,POST,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "f9c146a48119b95d2adbae6aed479a1a"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.011916
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/experiments">
  <items offset="0" total="0">
  </items>
  <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

This is the list of your active experiments. Most probably that list is empty.

Note: If you want to avoid entering your password each time you make a request, cURL can automatically find and send the credentials for a specific website by looking at a special file named `~/.netrc` (on Linux/MacOS systems). Here is how you would create such a file:

```
$ cat <<EOF >> ~/.netrc
machine api.bonfire-project.eu
login your-bonfire-login
password your-bonfire-password
EOF
chmod 600 ~/.netrc
```

Then, you just have to add the `-n` flag to the cURL command to tell it to use the `~/.netrc` file to find your credentials for the BonFIRE API:

```
$ curl -kni https://api.bonfire-project.eu/experiments
```

An interesting point to note in the previous response is that it includes an `Allow` HTTP header, which lists the HTTP verbs (see [Overview of Client Tools](#) for more information) that you can use on the current resource.

For example, the GET method is authorized (that's what we just did, since the cURL tool issues GET requests by default). But you can also see that this resource allows POST requests to be made.

Creating New Resources

Since POST has a semantic of CREATE, it means that you can CREATE a new experiment by POSTing on the `/experiments` URI. Using the [Media Type](#) description to look up how to represent an experiment in BonFIRE+XML, you could come up with the following request to create an experiment:

```
$ curl -k -i https://api.bonfire-project.eu/experiments -u BONFIRE_USER \
-H 'Content-Type: application/vnd.bonfire+xml' -X POST -d \
'<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>My First Experiment using cURL</name>
    <description>Experiment description</description>
    <walltime>3600</walltime>
</experiment>'
```

Hit ENTER, enter your password when prompted for, and you should get back a response like the following:

```
HTTP/1.1 201 Created
Date: Fri, 08 Jul 2011 09:29:32 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Location: https://api.bonfire-project.eu/experiments/1155
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.112345
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/1155" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <id>1155</id>
    <description>Experiment description</description>
    <name>My First Experiment using cURL</name>
    <walltime>3600</walltime>
    <user_id>crohr</user_id>
    <status>ready</status>
    <networks>
    </networks>
    <computes>
    </computes>
    <storages>
    </storages>
    <link rel="parent" href="/" />
    <link rel="storages" href="/experiments/1155/storages" />
    <link rel="networks" href="/experiments/1155/networks" />
    <link rel="computes" href="/experiments/1155/computes" />
</experiment>
```

As indicated by the HTTP status 201 Created, your experiment has been successfully created. You can find the URI of the newly created experiment in the Location HTTP header (<https://api.bonfire-project.eu/experiments/1155>). The response payload contains the experiment description, with links to other resources.

Deleting Resources

If you fetch the newly created experiment resource, you'll see that the Allow header is different (replace the URI with the one you obtained in the previous response):

```
$ curl -kni https://api.bonfire-project.eu/experiments/1155
```

```
HTTP/1.1 200 OK
Date: Fri, 08 Jul 2011 09:49:31 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Allow: GET,PUT,DELETE,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "50ea93fb5babdbe0a3b98382b11e757f"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.018795
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/1155" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <id>1155</id>
  <description>Experiment description</description>
  <name>My First Experiment using cURL</name>
  <walltime>3600</walltime>
  <user_id>crohr</user_id>
  <status>ready</status>
  <networks>
  </networks>
  <computes>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/1155/storages"/>
  <link rel="networks" href="/experiments/1155/networks"/>
  <link rel="computes" href="/experiments/1155/computes"/>
</experiment>
```

For an Experiment resource, you can see that PUT and DELETE requests are now supported, while POST is not allowed. PUT has the semantic of UPDATE, while DELETE has the semantic of, well, DELETE. Therefore you know you can DELETE the experiment by issuing the following request:

```
$ curl -kni https://api.bonfire-project.eu/experiments/1155 -X DELETE
```

```
HTTP/1.1 202 Accepted
Date: Fri, 08 Jul 2011 09:51:48 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Location: https://api.bonfire-project.eu/experiments/1155
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
```

```
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.026081
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

Here you see another status code: 202 Accepted, which means your request for deletion has been accepted, and will be shortly processed.

4.4.3 Bonus Points

At this point, you should probably take some time to explore the other resources that were advertised at <https://api.bonfire-project.eu/> (e.g. locations, users, etc.). Also, you could add the -v flag (verbose) to see what's being sent over the wire when you're doing a GET or POST request. Then, if you want to see a complete example of the HTTP requests required to create a whole experiment with compute, network, and storage resources, you should have a look at [Example HTTP Request/Response Session](#).

4.5 Command Line Interface Tools

The BonFIRE CLI tools are meant to provide users with a way to interact with the BonFIRE broker API from a command line interface. This usage might happen either manually or programmatically (i.e. scripted).

4.5.1 Introduction

The structure of these tools is inspired by the command line interface offered by the OpenNebula toolkit, in the sense that individual tools exists to interact with resources of a particular type. These individual tools are:

- `bfexperiment` - to interact with experiments
- `bfcompute` - to interact with compute resources
- `bfstorage` - to interact with storage resources
- `bfnetwork` - to interact with network resources
- `bfsitelink` - to interact with Autobahn sitelinks
- `bfrouter` - to interact with Federica routers
- `bffednet` - to interact with Federica networks
- `bfphysicalrouter` - to retrieve information on physical routers available through Federica
- `bfreservation` - to create or retrieve reservations
- `bflocation` - to retrieve information on BonFIRE sites

The general usage pattern is the same for all these tools:

```
# <clitool> <command> <arguments ...> [<options ...>]
```

where `<clitool>` is one of the tools mentioned above, `<command>` is one of {info|list|show|create|update|delete}, `<arguments ...>` are command specific arguments and `[<options ...>]` are additional options. Both generic that apply to all command, as well as command specific options exist. Options are - as the name implies - always optional.

The meaning of the individual commands is as follows:

- `info` - display configuration settings and some system information for diagnostic purposes. The output of this command is the same for all tools and should be included when reporting any issues.
- `list` - list the items in question. (translated to a HTTP GET request on the broker)
- `show` - display information on a particular item (also corresponds to a GET)
- `create` - create an item. (corresponds to a POST)
- `update` - change one or more parameters of an item (corresponds to PUT)
- `delete` - delete a particular item (corresponds to DELETE)
- `ssh` - Open an SSH connection to a BonFIRE compute resource. Only applicable to the `bfc` tool.

4.5.2 Download

The BonFIRE CLI Tools and related libraries can be downloaded from [here](#).

4.5.3 Installation

Installation from source

After obtaining a copy of the source distribution, unpack the downloaded archive (unless you got a copy from SVN), change to its directory and, as the root user, say:

```
# python setup.py install
```

Note that when installing from source you will also need the *BonFIRE Python Libraries* package. It is installed in the same manner.

RPM Installation

Add the BonFIRE RPM repository to your system's configuration and say:

```
# yum install bonfire-cli
```

Windows

Prerequisites

- Python 2.6 or higher
- Make sure that both the python interpreter executable (`python.exe`) and the python scripts directory (e.g. `c:\python27\scripts`) are on `%PATH%`

Installation

Follow the link above and download the relevant package (`bonfire-cli-<version>-win32.exe`). After the download finishes, run the downloaded file and follow the instructions on the screen.

4.5.4 Configuration

Upon startup, each tool will try to subsequently read configuration data from a number of locations. Any configuration key encountered during a later stage of this process will override any previously encountered instance of the same key. Any configuration parameters specified on the command line as options (see below) will override the respective configuration parameter specified in configuration files.

On UNIX like systems (including Linux and Apple), the following locations will be read (in that order):

- /etc/default/bonfire (note that this file will automatically exist on every compute resource created through the broker API, meaning that the BonFIRE CLI tools will work on BonFIRE computes without further configuration)
- /etc/bonfire
- ~/.config/bonfire
- ~/.bonfire

On MS Windows systems, the following locations will be read (in that order):

- %APPDATA%\bonfire
- %USERHOME%\bonfire
- %USERHOME%\bonfire

Configuration parameters are specified on a single line in the form <KEY>=<VALUE>. Any line starting with the hash sign (#) is ignored. The following configuration keys will be recognized:

- BONFIRE_URI - the URI of the BonFIRE broker
- BONFIRE_CREDENTIALS - credentials to use when accessing the broker API in the form <username>:<password>

4.5.5 General Usage Notes

Generic Options

Note that the short form of all generic options consists of a lower-case letter, whereas the short form of tool specific options consists of an upper-case letter. The following is a list of options that are applicable to all tools:

- --version - display version information and exit successfully.
- -h, --help - display a brief usage summary and exit successfully.
- -b <uri>, --broker-uri <uri> - URI of the BonFIRE broker API.
- -u <username>, --username <username> - BonFIRE username.
- -p <password>, --password <password> - BonFIRE password. Will be prompted for if unavailable. Note that specifying the password in this manner is strongly discouraged since it creates a security problem. You should rather consider to specify BONFIRE_CREDENTIALS in a configuration file.
- -f <file>, --configfile <file> - additional config file location to read. Will be read after all builtin locations have been read
- -q, --quiet - suppress any diagnostic output on stderr.
- -s, --show - print requests to stdout before sending them to the broker.
- -d, --dry-run - don't actually send requests to the broker. Implies -s unless -q is given.

Abbreviations

When specifying relations with BonFIRE entities, a number of abbreviations can be used to make specifying IDs more convenient. To be exact, the following applies:

- Location IDs can be specified either as their full ID (`/locations/uk-epcc`) or their short name (`uk-epcc`).
- Experiments can be specified using either their full ID (`/experiments/42`) or their short ID (42).
- Network and storage resources can be specified using either their full ID (`/locations/uk-epcc/networks/42`), their short ID (42) or their name (BonFIRE-WAN). Note that referring to a resource by name will be significantly slower than all other methods.
- When specifying the URI to the Bonfire RM (through `--broker-uri` or `-u`), in addition to give the full URI (e.g. `https://api.bonfire-project.eu`) one can refer to the magic values `%production`, `%qualification` and `%integration` which point to the well known URIs of these particular BonFIRE infrastructures. These values can be abbreviated as `%p`, `%q`, and `%i` respectively.

Boolean Values

On some occasions, boolean values must/can be entered. Boolean value interpretation is not case sensitive. In Addition, the following applies:

- The following values will be interpreted as `true`: `true`, `t`, `yes`, `y`, `1`
- The following values will be interpreted as `false`: `false`, `f`, `no`, `n`, `0`

The `info` Command

The `info` command does not execute any options on the BonFIRE RM. Its purpose is to print information about the environment the BonFIRE CLI tools run in as well as its current configuration. It behaves exactly the same for all tools. It is therefore irrelevant which tool is used to invoke it.

Note: The output of `bfexperiment info` should always be included when reporting bugs in the CLI tools.

The following options apply to the `info` command:

- `-t <format>, --format <format>` - Choose to write output either in a human readable way (`plain`) or in a way that is suitable for use as config file (`config`). Defaults to `config` format when outputting to a known config location and to `plain` format in all other cases.
- `-o <outfile>, --outfile <outfile>` - Write output to this file. When outputting in config file format (`--format config`), the magic values `%globalconfig` and `%userconfig` can be used to refer to the well known config file location. These values can be abbreviated as `%g` and `$u`.
- `-n, --no-purge` - Show plaintext passwords in output. Note that plaintext passwords will always be written when writing config format output to a file.
- `-r, --force` - Force execution even when trying to overwrite a file or trying to write plain output to a known config file.

Tip: A combination of the options above allows us to conveniently write a configuration file without having to use a text editor or other peasantly means:

```
$ bfexperiment info --user cliuser --outfile %userconfig --broker-uri %p --verbose
BonFIRE password:
[bonfire.cli] INFO: Writing output in config format to /home/demouser/.config/bonfire
```

Or in an equivalent but shorter way:

```
$ bfexperiment info -u cliuser -o %u -b %p  
BonFIRE password:
```

Both these commands configure the CLI tools to use the RM on BonFIRE's production infrastructure and the username `cliuser`. Since no password is specified, it is prompted for. The resulting configuration is written to the default user configuration file. Subsequently, the CLI tools can be used without having to specify this information again.

Output

All BonFIRE CLI tools write any data that is the result of the specified operation to `stdout`. All additional (diagnostic and error) information is written to `stderr`. It is therefore safe (and recommended) to redirect `stdout` as appropriate when data should be stored in a file and/or further processing by another program should take place.

Controlling Output

Information that is produced by the `show` and `list` commands can be controlled in various ways. The following options are always applicable to these commands, regardless of which tool is invoked:

- `-t <format>, --format <format>` - Format in which to show output. Possible values are `plain` (a human readable representation), `csv` (comma separated values) and `xml` (The raw XML representation coming from the RM). Defaults to `plain`.
- `-n <num>, --num <num>` - Number of times to retrieve information. Set to 0 to poll indefinitely. Defaults to 1.
- `-d <delay>, --delay <delay>` - Time to sleep between polling attempts in seconds. Obviously only meaningful when `--delay` is not 1.
- `-k, --keep-going` - Keep going in face of errors encountered during polling.
- `-a, --no-labels` - Do not show field labels in csv and plain output.
- `-r, --no-decorations` - Suppress decorations in plain output.
- `-s <separator>, --separator <separator>` - Separator to use in `csv` and `plain` output. Defaults to `,` for `csv` output and `|` for `plain` output.

In addition, the following option is applicable to the `list` command:

- `-e, --details` - By default the output of the `list` command will only contain the ID and name of listed entities. Specifying this option will cause additional information to be included in the output. This option has no impact for `xml` output. Note that this may cause the operation to take significantly longer to complete.

Tip: In combination with any readily available `xpath` tool, we can use the `XML` output option to easily apply `xpath` expressions to filter results, simply by piping the output of the CLI tool into an `xpath` tool. For example, the following expression will select all compute resources that implement BonFIRE services (e.g. monitoring aggregator or elasticity engine):

```
$ bfcompute list -t xml | xpath '/items/compute[starts-with(@name, "BonFIRE")]'
```

Note that common `xpath` expressions tend to contain characters that have special meanings in a UNIX shell (like parenthesis or quotation marks). Thus, it is necessary to either escape them with a backslash or surround the whole expression in single ticks (`'`) as seen above.

Many `xpath` implementations exist, a particularly comprehensive specimen is contained in perl's `XML-XPath` package:
<http://search.cpan.org/dist/XML-XPath/>

4.5.6 Working with Experiments (bfexperiment)

The `bfexperiment` tool can be used to query and manipulate existing experiments as well as to create new ones. Its usage is as follows:

Creating Experiments

```
# bfexperiment create <name> [-D <description>] [-W <walltime>] [-G <group> ...]
```

The following positional argument must be specified when creating experiments:

- `<name>` - The name for this experiment.

The following options are applicable when creating experiments:

- `-D <description>, --description <description>` - description of the experiment. Defaults to “`<no description>`”.
- `-W <walltime>, --walltime <walltime>` - lifetime of the experiment in seconds. Defaults to one day.
- `-G <group>, --group <group>` - A user group this experiment will be accessible by. This option can be specified multiple times.

Updating Experiments

```
# bfexperiment update <id> -S {running/stopped}
```

The status of an experiment can be updated through the `-S` option:

- `-S {running|stopped}, --status {running|stopped}` - Set the status of an existing experiment to either `running` or `stopped`.

4.5.7 Working with Sites (bflocation)

The `bflocation` tool can be used to query existing BonFIRE locations. Only the `show` and `list` commands are available for locations.

4.5.8 Working with Storages (bfstorage)

The `bfstorage` tool can be used to query and manipulate existing storage resources as well as to create new ones. Its usage is as follows:

Creating Storages

```
# bfstorage create <name> <location> [<experiment>] [-D <description>] [-S <size>] [-T {datablock/sha}
```

The following positional arguments must be specified when creating storages (in that order):

- `<name>` - The name for this resource.
- `<location>` - The BonFIRE site where this resource will be created.

Optionally, an experiment this resource will be part of can be specified as a third positional argument. The following options are applicable when creating storages:

- **-D <description>, --description** - The description of this storage resource. Defaults to “<no description>”.
- **-S <size>, --size** - Size of this storage resource in MiB. This is only applicable to datablock resources. Defaults to 1024 MiB.
- **-T {datablock|shared}** - The type of this storage resource. Note that “shared” storages are only available on be-ibbt. Defaults to “datablock”.
- **-F <filesystem>, --fstype <filesystem>** - The filesystem the storage resource will be formatted with. This is only applicable to datablock resources. Defaults to “ext3”.
- **-G <group>, --group <group>** - A user group this resource will be accessible by. This option can be specified multiple times.
- **-P, --public** - Indicates that this storage resource should be publicly available. Defaults to <false>.
- **-R, --persistent** - Indicates that a persistent storage resource should be created. Defaults to <false>.

Updating Storages

```
# bfstorage update <id> -P {true, false, yes, no, 1, 0, t, f} -R {true, false, yes, no, 1, 0, t, f}
```

The following options are applicable when updating storages:

- **-P {true, false, yes, no, 1, 0, t, f}, --public {true, false, yes, no, 1, 0, t, f}** - Changes the “public” flag of this storage resource.
- **-R {true, false, yes, no, 1, 0, t, f}, --persistent {true, false, yes, no, 1, 0, t, f}** - Changes the “persistent” flag of this storage resource.

4.5.9 Working with Networks (bfnetwork)

The bfnetwork tool can be used to query and manipulate existing network resources as well as to create new ones. Its usage is as follows:

Creating Networks

```
# usage: /usr/bin/bfnetwork create <name> <location> <experiment> [-A <address>] [-S {A,B,C}] [-X <prefix>]
```

The following positional arguments must be specified when creating storages (in that order):

- <name> - The name for this resource
- <location> - The BonFIRE site where this resource will be created.
- <experiment> - The experiment this resource will be part of.

The following options are applicable when creating network resources. Note that at present, networks can only be created on uk-hplabs and be-ibbt and that all options except **-G** are only applicable to networks on be-ibbt.

- **-A <address>, --address <address>** - Network address for this network resource. Applicable to networks on ONE and VW sites. (default: 192.168.0.0)
- **-X <prefix>, --prefix <prefix>** - Network size specified as CIDR prefix. Applicable to CELLS sites. (default: 24)
- **-S {A|B|C}, --size {A|B|C}** - Network size specified as network class. Only applicable to VW sites. (default: C)

- `-N <netmask>, --netmask <netmask>` - Network size specified as bit mask. Applicable to ONE sites. (default: 255.255.255.0)
- `-L <latency>, --latency <latency>` - Specifies the network latency in ms with the minimum non-zero value being 2ms.
- `-R <lossrate>, --lossrate <lossrate>` - Specifies the lossrate for this network as a floating point number in the range 0..1.
- `-B <bandwidth>, --bandwidth <bandwidth>` - Specifies the network bandwidth in Mbit/s.
- `-C {TCP, UDP}, --protocol {TCP, UDP}` - Type of generated background traffic. Specifying this implies the creation of an active network and thus requires `--packetsize` and `--throughput` as well. Only applicable to networks on VW. (default: TCP)
- `-K <packetsize>, --packetsize <packetsize>` - Packetsize of generated background traffic in bytes. Specifying this implies the creation of an active network and thus requires `--protocol` and `--throughput` as well. Only applicable to networks on VW.
- `-U <throughput>, --throughput <throughput>` - Number of generated background traffic packets per second. Specifying this implies the creation of an active network and thus requires `--packetsize` and `--protocol` as well. Only applicable to networks on VW.
- `-T {managed, active}, --type {managed, active}` - Force the type of this network to be either managed or active. If this option is missing the type will be derived from other options. Only applicable to networks on VW.
- `-P, --public` - Indicates that this network resource should be publicly available.
- `-G <group>, --group <group>` - A user group this resource will be accessible by. This option can be specified multiple times.

Updating Networks

```
# bfnetwork update <id> [-L <latency>] [-R <lossrate>] [-B <bandwidth>]
```

The following options are applicable when updating network resources. Note that at present, only networks form be-ibbt can be updated.

- `-L <latency>, --latency <latency>` - Specifies the network latency in ms with the minimum non-zero value being 2ms.
- `-R <lossrate>, --lossrate <lossrate>` - Specifies the lossrate for this network as a floating point number in the range 0..1.
- `-B <bandwidth>, --bandwidth <bandwidth>` - Specifies the network bandwidth in Mbit/s.

4.5.10 Working with Computes (bfcompute)

The `bfcompute` tool can be used to query and manipulate existing compute resources as well as to create new ones. Its usage is as follows:

Creating Computes

```
# bfcompute create <name> <os_image> <experiment> [-I <instance type>] [-V <vcpu>] [-P <cpu>] [-M <mem>]
```

Note that contrary to all other resources, compute resources can be created without explicitly specifying their location. That is because their location will be derived from either the OS image, any other involved storage resources and any involved network resources (in that order). Only when this is not possible (because all involved networks and storages were specified in an abbreviated form), specifying the location (through the `-L` option) is necessary.

The following positional arguments must be specified when creating computes (in that order):

- `<name>` - The name for this resource.
- `<os_image>` - Storage resource to use as OS image.
- `<experiment>` - The experiment this resource will be part of.

The following options are applicable when creating compute resources:

- `-I <instance type>, --instance_type <instance type>` - Instance type to use when creating the compute resource. (default: “small”)
- `-V <vcpu>, --vcpu <vcpu>` - Number of virtual CPUs present within the compute resource. Only applicable for the “custom” instance type. (default: 1)
- `-P <cpu>, --cpu <cpu>` - The amount of CPU resources to allocate. Must be a non-negative, non-zero floating point value. Only applicable for the “custom” instance type. (default: 1)
- `-M <memory>, --memory <memory>` - The amount of memory to allocate in MiB. Only applicable for the “custom” instance type. (default: 512MiB).
- `-S <storage>, --storage <storage>` - An additional storage resource to attach. Can be specified either as a full id (/locations/uk-epcc/storages/42), short id (42) or name (My-Storage). This option can be specified multiple times.
- `-N <network>, --network <network>` - A network resource to attach. Can be specified either as a full id (/locations/uk-epcc/networks/42), short id (42) or name (BonFIRE-WAN). This option can be specified multiple times.
- `-R <cluster>, --cluster <cluster>` - The cluster to deploy this compute resource on.
- `-H <host>, --host <host>` - The host to deploy this compute resource on.
- `-C <context>, --context <context>` - Specifies a contextualization parameter in the form “KEY=VALUE”. This option can be specified multiple times.
- `-L <location>, --location <location>` - Location where this resource will be deployed. If this is missing, the location will be derived from other involved resources.
- `-G <group>, --group <group>` - A user group this resource will be accessible by. This option can be specified multiple times.

Updating Computes

```
# bfcompute update <id> [-S {resume,suspended,shutdown}] [-A <save_as target>]
```

The following options are applicable when updating compute resources:

- `-S {resume,suspended,shutdown}, --state {resume,suspended,shutdown}` - Changes the state of the compute resources.
- `-A <save_as target>, --save_as <save_as target>` - Sets the “save_as” target for this compute resource.

4.5.11 Working with Autobahn Sitelinks (bfsitelink)

```
# bfsitelink create <name> <location> <experiment> [-D <description>] [-E ENDPOINT <endpoint>] [-B <
```

4.5.12 Working with Federica Routers (bfrouter)

```
# bfrouter create <name> <host> <location> <experiment> <interface> [<interface>...] [-C <config>] [
```

4.5.13 Working with Federica Networks (bffednet)

```
# bffednet create <name> <location> <experiment> <network_link> [<network_link> ...] [-G <group>...]
```

4.6 Restfully

4.6.1 What is Restfully

Restfully is a general-purpose client library for RESTful APIs. It is written in [Ruby](#). Its goal is to abstract the nitty-gritty details of exchanging HTTP requests between the user-agent and the server. It also discovers resources at runtime, which means should the API change and add a new functionality, the client will automatically discover it.

In the next section we will describe how to use Restfully in the specific case of BonFIRE.

4.6.2 Installation

Please refer to that [Restfully Wiki page](#) for installation instructions for UNIX-based systems and Windows.

Note: On Ubuntu, you might have to install the `ruby1.8-dev` package, otherwise you may have issues installing Ruby gems later:

```
$ apt-get install ruby1.8-dev
```

Note: If you try to install Restfully at IBBT, you will encounter an error, because IBBT nodes don't have direct internet access. HTTP and FTP internet access is available through the use of a proxy: `http://proxy.atlantis.ugent.be:8080`. Therefore you have to install gems with the `-p` flag:

```
$ gem install -p "http://proxy.atlantis.ugent.be:8080" restfully restfully-addons
```

Note: **On Windows**, when Restfully uses your private key to connect to SSH gateways etc it tries to ask your password at the command line but I am not able to type it in. In Unix systems ssh agents can be used eliminate this need to get the password. I have not yet investigated if there is a windows equivalent that will eliminate this need to obtain a password. In the meantime I cannot use the Restfully commands that require my private key. Using a private key with no password may be a way around this problem but that has its own risks.

4.6.3 Getting Started

Start the interactive Restfully shell (replace `LOGIN` and `PASSWORD` by your BonFIRE credentials, see the [FAQ](#) for an alternative way of passing parameters):

```
$ restfully --uri=https://api.bonfire-project.eu/ -u LOGIN -p PASSWORD -r ApplicationVndBonfireXml
```

You should get back a prompt like the following:

```
Restfully/0.8.1 - The root resource is available in the 'root' variable.  
ruby-1.8.7-p249 >
```

As indicated, enter `root` and hit RETURN:

```
ruby-1.8.7-p249 > root  
=> {"timestamp"=>"1305300530", "version"=>"0.5.1"}
```

You just fetched the root of the BonFIRE API. You can see the BonFIRE API version (0.5.1) and the server timestamp. If you want more information of that particular resource, enter `pp` ('pretty print') followed by your request:

```
ruby-1.8.7-p249 > pp root  
#<Resource:0x813f065c uri=https://api.bonfire-project.eu/  
  RELATIONSHIPS  
    experiments, locations, self  
  PROPERTIES  
    "timestamp"=>"1305300530"  
    "version"=>"0.5.1">  
=> nil
```

The `RELATIONSHIPS` header contains links to other API resources (here: `experiments`, `locations`, `self`). You can follow any of these links and see what comes back. For instance, following `locations`:

```
ruby-1.8.7-p249 > pp root.locations  
#<Collection:0x813dd534 uri=https://api.bonfire-project.eu/locations  
  RELATIONSHIPS  
    parent, self  
  ITEMS (0..5)  
    #<Resource:0x813c6b04 uri=https://api.bonfire-project.eu/locations/be-ibbt>  
    #<Resource:0x813b5868 uri=https://api.bonfire-project.eu/locations/de-hlrs>  
    #<Resource:0x813a45cc uri=https://api.bonfire-project.eu/locations/fr-inria>  
    #<Resource:0x81393330 uri=https://api.bonfire-project.eu/locations/uk-epcc>  
    #<Resource:0x81382094 uri=https://api.bonfire-project.eu/locations/uk-hplabs>>  
=> nil
```

It returns the collection of locations available in BonFIRE. If you need to access a specific location, you can either use the `find` function of the [Ruby Enumerable](#) objects:

```
ruby-1.8.7-p249 > pp root.locations.find{|l| l['name'] == 'fr-inria'}  
#<Resource:0x81337e68 uri=https://api.bonfire-project.eu/locations/fr-inria  
  RELATIONSHIPS  
    computes, networks, parent, self, storages  
  PROPERTIES  
    "name"=>"fr-inria"  
    "url"=>"https://bonfire.grid5000.fr:443">  
=> nil
```

Or use the shortcut method `[]`, with a `Symbol` as key:

```
ruby-1.8.7-p249 > pp root.locations[:'fr-inria']  
#<Resource:0x812e9240 uri=https://api.bonfire-project.eu/locations/fr-inria  
  RELATIONSHIPS  
    computes, networks, parent, self, storages  
  PROPERTIES  
    "name"=>"fr-inria"  
    "url"=>"https://bonfire.grid5000.fr:443">  
=> nil
```

Again, you see the `RELATIONSHIPS`, and you can look at any of them by following the links:

```
ruby-1.8.7-p249 > pp root.locations[:'fr-inria'].networks
#<Collection:0x813ffff1c uri=https://api.bonfire-project.eu/locations/fr-inria/networks
 ITEMS (0..2)/2
 #<Resource:0x813f754c uri=https://api.bonfire-project.eu/locations/fr-inria/networks/1>
 #<Resource:0x813f1e6c uri=https://api.bonfire-project.eu/locations/fr-inria/networks/2>>
=> nil
```

A collection includes the [Ruby Enumerable](#) module, so you can use any of the functions defined in that module. For instance, pretty-printing each of the networks can be achieved with:

```
ruby-1.8.7-p249 > root.locations[:'fr-inria'].networks.each{|net| pp net}
#<Resource:0x80fed9d8 uri=https://api.bonfire-project.eu/locations/fr-inria/networks/1
RELATIONSHIPS
self
PROPERTIES
"name"=>"Public Network"
"public"=>"YES"
"id"=>"1">
#<Resource:0x80fdb15c uri=https://api.bonfire-project.eu/locations/fr-inria/networks/2
RELATIONSHIPS
self
PROPERTIES
"name"=>"WAN Network"
"public"=>"YES"
"id"=>"2">
=> [{"name"=>"Public Network", "public"=>"YES", "id"=>"1"}, {"name"=>"WAN Network", "public"=>"YES", "id"=>"2"}]
```

Take your time to follow every relationship you find to see what you get back.

You can also use additional features, like doing the save_as feature.

You must create an experiment with a compute, and the save-as feature can be achieved with:

```
ruby-1.8.7-p249 > pp root.locations[:'fr-inria'].computes[:'vm'].update(:disk => [{:save_as => {:name
ruby-1.8.7-p249 > pp root.locations[:'fr-inria'].computes[:'vm'].update(:state => "shutdown")
```

At be-ibbt testbed, the status needs to be stopped, not shutdown contrary to others testbeds.

Then in the next section we'll see how we can create things.

4.6.4 Creating a BonFIRE Experiment

You could do all the following using the interactive shell, but it's probably easier to write everything in a script file ready to be executed as many times as you need. For this we'll use [Restfully](#) in library mode, and we can write a short script that deploy the resources we need, and SSH into them to configure them. This script uses a configuration file for restfully authentication, see the FAQ entry for [How do I avoid passing my password each time I want to use Restfully?](#) in order to set it up.

```
require 'rubygems'
require 'restfully'
require 'restfully addons/bonfire'

SERVER_IMAGE_NAME = "BonFIRE Debian Squeeze v5"
WAN_NAME          = "BonFIRE WAN"

logger           = Logger.new(STDOUT)
logger.level     = Logger::INFO
```

```
session = Restfully::Session.new(
  :configuration_file => "~/.restfully/api.bonfire-project.eu.yml",
  :gateway             => "ssh.bonfire.grid5000.fr",
  :keys                => ["~/.ssh/id_rsa"],
  :cache               => false,
  :logger              => logger
)

experiment = nil

begin
  # Find an existing running experiment with the same name or submit a new
  # one. This allows re-using an experiment when developing a new script.
  experiment = session.root.experiments.find{|e|
    e['name'] == "Demo SSH" && e['status'] == "running"
} || session.root.experiments.submit(
  :name      => "Demo SSH",
  :description => "SSH demo using Restfully - #{Time.now.to_s}",
  #define the group(s) you want to use if it differs from your username
  #:groups    => "mygroup",
  :walltime   => 8*3600 # 8 hours
)

# Create shortcuts for location resources:
inria = session.root.locations[:'fr-inria']
fail "Can't select the fr-inria location" if inria.nil?

session.logger.info "Launching VM..."
# Find an existing server in the experiment, or set up a new one:
server = experiment.computes.find{|vm|
  vm['name'] == "VM-experiment#{experiment['id']}"
} || experiment.computes.submit(
  :name      => "VM-experiment#{experiment['id']}",
  :instance_type => "small",
  :disk       => [
    :storage => inria.storages.find{|s|
      s['name'] == SERVER_IMAGE_NAME
    },
    :type     => "OS"
  ],
  :nic        => [
    {:network => inria.networks.find{|n| n['name'] == WAN_NAME}}
  ],
  :location   => inria
)
server_ip = server['nic'][0]['ip']
session.logger.info "SERVER IP=#{server_ip}"

# Pass the experiment status to running.
# If it was already running this has no effect.
experiment.update(:status => "running")

# Wait until all VMs are ACTIVE or RUNNING and ssh-able.
# Fail if one of them has FAILED.
until [server].all?{|vm|
  ['RUNNING', 'ACTIVE'].include?(vm.reload['state']) && vm.ssh.accessible?
} do
  fail "One of the VM has failed" if [server].any?{|vm|
```

```

    vm['state'] == 'FAILED'
}
session.logger.info "One of the VMs is not ready. Waiting..."
sleep 20
end

session.logger.info "VMs are now READY!"
# Display VM IPs
session.logger.info "*** Server IP: #{server['nic'][0]['ip']}"

server.ssh do |ssh|
  session.logger.info "Uploading content..."
  # Here is how you would upload a file:
  # sshscp.upload!("/path/to/file", '/tmp/file.log')
  # Here is how you can upload some in-memory data:
  sshscp.upload!(StringIO.new('some data'), '/tmp/file.log')
  # See <http://net-ssh.github.com/scp/v1/api/index.html> for more details.

  session.logger.info "Content of uploaded file:"
  puts ssh.exec!("cat /tmp/file.log")

  session.logger.info "Installing things..."
  output = ssh.exec!("apt-get update && apt-get install curl -y")
  session.logger.debug output

  session.logger.info "Running query against API..."
  puts ssh.exec!("source /etc/default/bonfire && curl -k $BONFIRE_URI/locations/$BONFIRE_PROVIDER/
end

session.logger.warn "Success! Will delete experiment in 10 seconds. Hit CTRL-C now to keep your VMs"
sleep 10
experiment.delete

rescue Exception => e
  session.logger.error "#{e.class.name}: #{e.message}"
  session.logger.error e.backtrace.join("\n")
  session.logger.warn "Cleaning up in 30 seconds. Hit CTRL-C now to keep your VMs..."
  sleep 30
  experiment.delete unless experiment.nil?
end

```

A video Demonstrating Control and Observability on BonFIRE

The script `demo_experiment_using_restfully.rb` launches an experiment on BonFIRE using restfully. (The following files are also needed to run the demo: `web_backend.tar.gz`, `web_frontend-sites-available.tar.gz`, `web_frontend-www.tar.gz`, `load_balancer.tar.gz`)

4 Computers are defined, launched and configured:

- A web backend hosting some video files (debian+nginx)
- A web frontend hosting a web page and a js video player requesting the videos from the web backend (debian+nginx+mediaelement)
- A load balancer proxyfying the web frontend (debian+nginx)
- An aggregator used to collect the metrics of the other computes and their hosts (when the computes are virtual machines) (debian+zabbix)

The load balancer, launched at fr-inria, is available on Public Network, and connects to the web frontend through BonFIRE WAN. The web frontend, launched at be-ibbt, connects to the web backend, launched at be-ibbt, through a VirtualWall network: backend network. All the computes connect to the aggregator, launched at fr-inria, through BonFIRE WAN. The aggregator is also available through the BonFIRE portal. The zabbix API is then used to measure the transmission rate (TX) of the different interfaces involved in serving the video files. A metric measuring the number of hits in the last five minutes is added to the aggregator through the BonFIRE API context.

You can download this demo video here.

4.6.5 FAQ

How do I avoid passing my password each time I want to use Restfully?

```
$ mkdir -p ~/.restfully && echo '  
uri: https://api.bonfire-project.eu/  
username: LOGIN  
password: PASSWORD  
require: ['ApplicationVndBonfireXml']  
' > ~/.restfully/api.bonfire-project.eu.yml && chmod 600 ~/.restfully/api.bonfire-project.eu.yml
```

Then:

```
$ restfully -c ~/.restfully/api.bonfire-project.eu.yml
```

Where can I find examples?

Some BonFIRE-specific examples are maintained at <https://github.com/crohr/restfully-addons/tree/master/examples/bonfire>.

COMPUTE

5.1 Overview of Compute

BonFIRE offers compute resources to be able to deploy *VM Images* on the different testbeds. A compute resource MUST be deployed with a network resource, which could be one of the default networks provided in BonFIRE or one that you create yourself, as discussed in *Overview of Networking in BonFIRE*. A compute resource can, optionally, also be deployed with an additional storage resource if the storage space provided with the VM image is not great enough (maximum 10GB currently). For more information about this, see *Overview of Storage*.

When compute resources are deployed, there's some *Contextualisation* that occurs to configure details such as the hostname, public IP (if any) and bonfire credentials. Such things are provided automatically by BonFIRE, but experimenters can also provide parameters such as aggregator IP (if any), metrics (if any) and post-install scripts. The contextualisation element is generic, so that any key-value pairs could be defined. The contextualisation variables are written to /etc/default/bonfire, so you can source this file to access the variables within your experiment VMs.

To support large-scale experiments, BonFIRE offers *On Request Resources*, which can be reserved in advanced and used as the 'basic' compute resources. For more details and how to deploy the compute resources, see *Deploying Compute Resources in BonFIRE*. It is now also possible to deploy resources on Amazon EC2 within a BonFIRE experiment.

As a facility to conduct empirical experiments, BonFIRE strives to provide maximum control of how you define and where you deploy the compute resources. To deploy the compute resources, BonFIRE offers two ways of specifying the *Instance Types*, following either a 'small', 'medium', 'large' type of taxonomy or your own custom template based on CPU, VCPU and RAM. You can also specify the physical host or cluster that the resource should be deployed on.

5.2 Contextualisation

Contextualisation elements in OCCI can be used to pass initialisation values for a compute resource in the form of simple key-value pairs. This is used by the BonFIRE testbeds and the BonFIRE API, but is also available to the users for specifying, for example:

- The IP address of the Zabbix Aggregator when deploying compute resources (the Portal will do this automatically).
- Any custom monitoring metrics.
- Elasticity trigger rules, if the EaaS is used.
- Post-install scripts, which can run after the VM has been deployed.
- Any additional SSH keys, to allow multiple users access to experiment VMs.

The contextualisation element is generic, so that any key-value pairs could be defined. The contextualisation variables are written to `/etc/default/bonfire`, so you can source this file to access the variables within your experiment VMs. Follow the links below for more information.

5.2.1 List of all Contextualisation Names

Provided by the BonFIRE Testbed

`bonfire_resource_id`

The ID of the compute resource.

`public_ip`

The public IP (if any) that was associated with the compute resource.

`wan_ip`

The IP in the BonFIRE WAN (if any) that was associated with the compute resource.

`public_mac`

The MAC address corresponding to the `public_ip`.

`wan_mac`

The MAC address corresponding to the `wan_ip`.

`public_gateway`

The IP of the testbed's gateway for the public network.

`wan_gateway`

The IP of the testbed's gateway for the WAN network.

`files`

A whitespace-separated list of files that must be copied into the contextualisation image of the compute resource. It looks like the following: `/srv/cloud/context /srv/cloud/context/lib /srv/cloud/context/distributions /srv/cloud/context/sites /srv/cloud/context/common /srv/cloud/context/init.sh`

hostname

The hostname given to the compute resource. In BonFIRE this is `bonfire_resource_name-bonfire_resource_id`.

usage

A flag that specifies the intended usage of the compute resource. By default it is set to `zabbix-agent`.

dns_servers

A whitespace-separated list of DNS servers.

ntp_servers

A whitespace-separated list of NTP servers.

Provided by the BonFIRE API

bonfire_provider

Specifies the testbed where the compute resource is located. Values include one of `fr-inria`, `uk-epcc`, `de-hlrs`, `be-ibbt`, `uk-hplabs`.

bonfire_uri

Specifies the URI to the BonFIRE API.

bonfire_credentials

Specifies the elasticity credentials (`login:password`) that were delivered for the duration of the experiment. Those credentials can be used to access the BonFIRE API from compute resources.

bonfire_experiment_id

Specifies the ID of the experiment the resource belongs to.

bonfire_experiment_routing_key

Specifies the routing key to be used by a client willing to access the message queue dedicated to her experiment.

bonfire_experiment_aggregator_password

Specifies the aggregator password that was generated to be used to connect as `admin` to the experiment's aggregator (if any). Every user that can access to the experiment (through group membership for instance) can retrieve that password and connect to the aggregator (here Zabbix) GUI of the experiment. It also means that all compute resources can use that password to make calls to the Zabbix API (or go through the BonFIRE API).

bonfire_experiment_expiration_date

Specifies the date at which the experiment is expected to be stopped, i.e. when the experiment `walltime` is hit. This is a UNIX timestamp.

bonfire_resource_name

Specifies the name of the resource, as it was given by the user when the resource was created.

Provided by the BonFIRE User

The following contextualisation elements may be set by the user, within the `<context></context>` section of a compute description:

aggregator_ip

The IP of the compute resource that acts as the aggregator. The user **MUST** set this information if she wants to register metrics to the remote aggregator.

Example:

```
<compute>
  ...
  <context>
    <aggregator_ip>131.254.204.12</aggregator_ip>
  ...
</context>
</compute>
```

metrics

A list of metrics that need to be registered from the compute resource to the aggregator. The format is as follows:

```
<metric>metric_name,shell_script_that_returns_metric_value</metric>
```

More than one metric definition can be given.

Example:

```
<compute>
...
<context>
  <aggregator_ip>131.254.204.12</aggregator_ip>
  <metrics>
    <metric>syslog_length,/bin/cat /var/log/syslog | /usr/bin/wc -l</metric>
    <metric>other_metric_name,/path/to/script</metric>
  </metrics>
  ...
</context>
</compute>
```

postinstall

A URI of a file that will be fetched and executed after the compute resource has been initialized.

Example:

```
<compute>
...
<context>
  <postinstall>http://server.ltd/path/to/postinstall/script.sh</postinstall>
  ...
</context>
</compute>
```

authorized_keys

One or more SSH public keys that will be dropped in the authorized_keys file of the root account on the compute resource.

Example:

```
<compute>
...
<context>
  <authorized_keys>
    ssh-rsa AAAAB3NzaC1yc...zBFQh/2GE9w== crohr@parachute
    ssh-dss AAAAB3NzaC1kc...V66BzU= crohr@somewhere
  </authorized_keys>
  ...
</context>
</compute>
```

iproute

Define additional route entries (separated by semicolon ;) for particular compute resource. This value will be used to execute series of ip route add ROUTE commands.

Example: .. code-block:: xml

```
<compute> ... <context>
  <ipRoute>192.168.1.0/24 via 192.168.1.254; 192.168.5.0/24 dev eth1</ipRoute>
  ...
</context>
</compute>
```

```
</context>  
</compute>
```

Note: All the contextualisation variables will be written to the /etc/default/bonfire file, so that you can source this file to access the variables any time you need them (uppercased version).

Example:

```
$ source /etc/default/bonfire  
$ echo $BONFIRE_CREDENTIALS  
crohr:some-password  
  
$ aptitude install curl -y  
$ curl -k $BONFIRE_URI -u $BONFIRE_CREDENTIALS  
<?xml version="1.0" encoding="UTF-8"?>  
<root xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/">  
  <version>0.10.1</version>  
  <timestampl>1311584251</timestampl>  
  <link rel="experiments" href="/experiments" type="application/vnd.bonfire+xml"/>  
  <link rel="locations" href="/locations" type="application/vnd.bonfire+xml"/>  
  <link rel="users" href="/users" type="application/vnd.bonfire+xml"/>  
</root>
```

Please keep in mind that the contextualisation file may be generated **after** the compute resource is SSH-able, so don't be surprised if you log onto a compute resource and don't see the /etc/default/bonfire file immediately. It should be made available shortly thereafter.

If you need the contextualisation to be available before you execute some code in your scripts, the following code will wait until it's available:

```
# Contextualisation  
until [ -f /etc/default/bonfire ]; do  
# echo "Waiting for contextualisation file to be available..."  
sleep 5  
done  
source /etc/default/bonfire
```

5.2.2 Contextualisation How-To

This page gives some hints on how to contextualise a VM that belongs to an experiment. A typical scenario is an experiment made of a set of VMs where there are dependencies among services in terms of knowing particular parameters (e.g. IP) and also in the starting order. Imagine you have 3 VMs, A, B and C. B and C need to register with a service run in A and for that they need to know the IP of A. BonFIRE provides a way to contextualise VMs. For example, in OCCI this is done via the context element:

```
<compute>  
  ...  
  <context>  
    <aggregator_ip>131.254.204.12</aggregator_ip>  
    <A_service_ip>131.254.204.13</aggregator_ip>  
    ...  
  </context>  
</compute>
```

or from RESTfully:

```
:context => {  
  'aggregator_ip' => '131.254.204.12',
```

```

    'A_service_ip' => '131.254.204.13'
}

```

In this case the aggregator and A_service_IP keys and values will be added to the /etc/default/bonfire file of B and C, and may be exposed as environment variables. However, although BonFIRE contextualisation is performed as quickly as possible, it still takes some time. So, how would B and C know when the A_service_IP value is available in this context file so that they can perform their registration? The approach in this page to create an Init Script LSB (Linux Standards Base)-compliant, <http://wiki.debian.org/LSBInitScripts/>

```

#!/bin/bash

## BEGIN INIT INFO
# Provides:          vm-context
# Required-Start:    $all
# Required-Stop:
# Should-Start:     $all
# Default-Start:    2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Contextualize the VM based on /etc/default/bonfire
# Description:       Read the A_service_IP from /etc/default/bonfire, and
#                   update B or C A_service_IP with this value. It waits
#                   until the /etc/default/bonfire and the A_service_IP
#                   are available.
## END INIT INFO

do_start () {
    # wait for the /etc/default/bonfire to be available
    while true
    do
        [ -f /etc/default/bonfire ] && break
        sleep 1
    done

    # wait for the A_service_IP to be available
    until [ `grep -c A_service_IP /etc/default/bonfire` == 1 ]
    do
        sleep 1
    done

    if [ -f /etc/default/bonfire ]; then
        source /etc/default/bonfire

        A_ip=`echo $A_service_IP`
        # register with A_ip
        ...
    fi

    exit 0
}

case "$1" in
    start)
        do_start
        ;;
    stop)
        # clean /etc/default/bonfire so that if the image is saved it will not include this value
        sed -i "s:A_service_IP.*::g" /etc/default/bonfire

```

```
# stop the process when shutting down
kill $(ps ax | awk '/vm-context.sh/ {print $1}')
exit 3
;;
esac

:
```

The example above contains hints on how to wait for the BonFIRE file and also for the appropriate variable to be available. The section between `### BEGIN INIT INFO` and `### END INIT INFO` contains LSB parameters. It should be modified according to the specific needs.

Once the script has been completed run the command

```
update-rc.d vm-context.sh defaults
```

This will add the `vm-context.sh` as a service to be started at boot up.

5.3 On Request Resources

On BonFIRE, some testbeds offer the possibility to reserve additional resources on demand. If you think your experiment needs a lot of compute resources (in the hundreds), or you want a dedicated set of resources so that you can experiment with VM colocation and other placement constraints, then you should use this API. On-request (compute) resources are only available at Inria, HP, USTUTT and IBBT.

For information about the exact infrastructure on offer by the different testbed for on-request resources, please read the [Infrastructure](#) page. Note that, currently, only Inria offers a dynamic advanced reservation system for on-request resources. For other testbeds, this needs to be done manually. See details below.

5.3.1 At IBBT

Requests for on-request resources at IBBT from BonFIRE will compete with requests from other Virtual Wall users and be subject to the same charter on acceptable usage time and duration. In general, this corresponds to standard usage policies applicable on Emulab testbeds (<http://www.protogeni.net/trac/emulab/wiki/Swapping>).

To request additional resources at IBBT, you need to send an email to `vwall-ops@atlantis.ugent.be`.

You need to specify the requests as follows:

- Number of physical resources, each type of physical resources available being completely described. Therefore, the mapping of requirements between instance types and physical hosts is left to the user.
- Start time.
- Duration.
- Experiment project already accepted to run on BonFIRE related to the request.

If accepted, the resources will be scheduled by the Virtual Wall testbed responsible person and enforced on the local platform. Simple and small requests that conform to IBBT's policy for BonFIRE access to the Virtual Wall can be approved quickly (within two working days). Requests for longer durations (several days) or larger numbers of nodes (more than twenty) will be queued for approval by the people responsible for IBBT's testbed. The expected delay in this case is smaller than one week.

5.3.2 At USTUTT

Different to the other testbeds, USTUTT does not offer Cloud resources on-request, but offers access to the HPC cluster at HLRS. BonFIRE users have access to a 545 nodes / 60 TFLOPs cluster.

The process for using the HPC cluster is as follows:

1. Boot the special HPC cluster image at HLRS which includes the Globus Toolkit
2. Contact Michael Gienger <gienger@hlrs.de> to obtain a user certificate (no generic certificates are available)
3. Insert the user certificate into the VM
4. Copy your data into the VM
5. Submit your batch-job to the HPC-cluster
6. Wait until the job finishes

You are not able to install software, but common software tools like compilers or simulation software packages are already installed. If additional software is needed, system admins have to do that. The cluster is not integrated into the BonFIRE WAN, but it can be accessed through the booted VM via `gsi-ssh`. No monitoring data will be available.

5.3.3 At HP

The on-request resources offered by HP must be requested via email. The request must clearly state:

1. The experiment already accepted to run on BonFIRE.
2. A justification of the need for the resources.
3. The number of on-request nodes and/or amount of storage needed.
4. The start time and expected use period.

Please note that the acceptance will be based on availability. The resource allocation process is manual and must be planned on, so expect 2-3 weeks for them to be available. Please send the requests to Juan Lorenzo del Castillo <juan.lorenzo-del-castillo@hp.com>.

5.3.4 At Inria

Inria offers a booking system accessible via the web browser or an API supported by Restfully. How to make the reservations and what goes on under the bonned is explained in sections below.

Please note that any reservations at Inria must comply with the Grid'5000 policy rules and be aware that the resources are also available for other Grid'5000 users. In particular, be aware of the following acceptable usage times and duration:

- Between 9:00 and 19:00 UTC+1, during weekdays, no user should use more than 3600 core-hours (daytime usage).
- Overnight and during week-ends, no restrictions apply (free usage).
- No resource usage should cumulate day time usage and free usage.

Additional conditions apply to BonFIRE users: without additional approval from Inria, no experiment should consume more than 446,400 core-hours (complete usage of the site in Rennes for 4 week-ends).

Using the GUI

It's actually quite simple. Just open the following link: <https://api.bonfire-project.eu/locations/fr-inria/reservations>, click on the 'New reservation' button, and follow the instructions.

Here is what you should see:

- The list of your current and past reservations (empty the first time you log in):

The screenshot shows a web browser window titled "BonFIRE On Request - fr-inria". The address bar displays the URL <https://api.integration.bonfire.grid5000.fr/locations/fr-inria/reservations>. The main content area is titled "List of your reservations (crohr)". Below the title is a button labeled "+ New reservation". The table lists 15 past reservations, each with a unique ID, status, and submission date. To the right of each row is a small circular icon with a right-pointing arrow.

List of your reservations (crohr)	
+ New reservation	
405293 - running	Submitted on Fri Jan 20 15:03:20 +0100 2012
405290 - error	Submitted on Fri Jan 20 14:59:56 +0100 2012
405289 - running	Submitted on Fri Jan 20 14:59:25 +0100 2012
404593 - terminated	Submitted on Thu Jan 12 13:15:40 +0100 2012
404153 - error	Submitted on Thu Jan 05 17:10:29 +0100 2012
404152 - error	Submitted on Thu Jan 05 17:09:18 +0100 2012
402973 - error	Submitted on Tue Dec 20 14:24:17 +0100 2011
402763 - terminated	Submitted on Wed Dec 14 17:10:46 +0100 2011
402723 - terminated	Submitted on Tue Dec 13 11:24:04 +0100 2011
402722 - error	Submitted on Tue Dec 13 11:06:00 +0100 2011
402719 - error	Submitted on Mon Dec 12 17:24:54 +0100 2011
402718 - terminated	Submitted on Mon Dec 12 17:10:50 +0100 2011
402716 - terminated	Submitted on Mon Dec 12 16:59:41 +0100 2011
402715 - terminated	Submitted on Mon Dec 12 16:15:43 +0100 2011
402713 - error	Submitted on Mon Dec 12 15:41:09 +0100 2011

- The form where you select your resources:
- The details of a reservation:

Note: In the future, this GUI will probably be integrated into the BonFIRE portal. For now this is a separate interface, but you can still log in with your existing BonFIRE credentials.

At the end of the process, you'll see the details of your reservation. You will notice a field named CLUSTER UUID, which you'll have to use when you create compute resources. Simply add <cluster>{{cluster_uuid}}</cluster> to your compute creation request (in the XML payload), so that

The screenshot shows a web browser window titled "BonFIRE On Request - fr-inria" with the URL <https://api.integration.bonfire.grid5000.fr/locations/fr-inria/reservations/form>. The page is titled "BonFIRE On Request Reservation Form (crohr)".

The form instructions state: "Fill in the form below to request more resources at the fr-inria BonFIRE testbed. If your reservation fails, try another combination of clusters."

Select the physical machines
Three options are listed with sliders:

- paradent (64 physical machines with CPU: 2x4 cores Intel Xeon 2.5 GHz - RAM: 32 GB)
- paramount (33 physical machines with CPU: 2x2 cores Intel Xeon 2.33 GHz - RAM: 8 GB)
- parapide (25 physical machines with CPU: 2x4 cores Intel Xeon 2.93 GHz - RAM: 24 GB)

Select the starting date and duration of your reservation
Fields for "From" (2012-01-20), "At" (15:12), and "Duration" (0 Day, 08:00:00) are present.

[Toggle Gantt diagram](#)

Who to notify when the reservation has started?
Email address: mailto:cyril.rohr@inria.fr

Make a reservation

[View my reservations](#)

The screenshot shows a web browser window titled "BonFIRE On Request - fr-inria". The URL in the address bar is <https://api.integration.bonfire.grid5000.fr/locations/fr-inria/reservations/405294>. The page displays a form for a reservation with the identifier "Reservation #405294".

The form fields are as follows:

- Submission date: Fri Jan 20 15:12:47 +0100 2012
- State: running
- User: crohr
- Cluster UUID for BonFIRE
(keep it secret): 5ff33cb5ad0e64302ef64f55a08817d818780b81
- Scheduled at: Fri Jan 20 15:12:51 +0100 2012
- Start date: Fri Jan 20 15:12:51 +0100 2012
- End date: Fri Jan 20 16:12:00 +0100 2012
- Assigned nodes: paramount-18.rennes.grid5000.fr

At the bottom of the form, there are three buttons: "Request deletion" (blue), "STDOUT" (gray), and "STDERR" (gray). Below the form, there is a navigation bar with two buttons: "View my reservations" and "New reservation".

the compute resource gets created on your dedicated cluster, and not on the resources shared by all BonFIRE users. Assuming your CLUSTER UUID is 5ff33cb5ad0e64302ef64f55a08817d818780b81, the XML payload that you send to create a resource in the portal or via the API would look like:

```
<compute xmlns="...">
  <name>Compute name</name>
  <cluster>5ff33cb5ad0e64302ef64f55a08817d818780b81</cluster>
  <instance_type>lite</instance_type>
  ...
  <context>
    ...
  </context>
</compute>
```

Using the API

You can also make your reservations through a JSON API. The recommended way is to use [Restfully](#), which automatically supports that kind of API.

Here is how you would do it in a Restfully session:

```
reservations = get("/locations/fr-inria/reservations")
resa = reservations.submit(:clusters => {:paradent => 2, :paramount => 1}, :from => Time.now.to_i, :t
pp resa

puts "***"
puts "CLUSTER ID to use in your reservations = #{resa['name']}
```

Run it like this:

```
$ restfully -c ~/.restfully/config --shell http://doc.bonfire-project.eu/R2/_static/examples/restfull
```

Then, **assuming your reservation is RUNNING** you can use a script like the following, to launch compute resources in your dedicated cluster:

```
logger.info "Starting..."

inria = root.locations[:'fr-inria']
fail "No location" if inria.nil?

image = inria.storages[:'BonFIRE Debian Squeeze v3']
fail "No image" if image.nil?

# For now, you must use a specific network if you wish to SSH into your VMs
# (and you must go through INRIA's SSH gateway). In the near future, you'll be
# able to use the BonFIRE WAN network, as you would expect.
network = inria.networks[:'BonFIRE OnDemand WAN']
fail "No network" if network.nil?

count = (ENV['COUNT'] || 1).to_i
cluster = ENV['CLUSTER'] || "default"

experiment = root.experiments.submit(
  :name => "BonFIRE On Demand",
  :description => "Started on #{Time.now.to_s}.",
  :walltime => 10*3600
)
```

```
computes = experiment.computes

logger.info "I'm going to submit #{count} VMs..."

count.times do |i|
  payload = {
    :name => "VM-#{i}",
    :instance_type => "lite",
    :location => inria,
    :disk => [{:storage => image}],
    :nic => [{:network => network}],
    :cluster => cluster
  }
  payload[:host] = ENV['HOST'] unless ENV['HOST'].nil?

  c = computes.submit(payload)
  logger.info "Submitted VM##{c['id']} with IP=#{c['nic'][0]['ip']}, in cluster #{cluster}."
end
```

Run it like this (replace 5ff33cb5ad0e64302ef64f55a08817d818780b81 with your cluster UUID):

```
$ CLUSTER=5ff33cb5ad0e64302ef64f55a08817d818780b81 COUNT=5 restfully -c ~/.restfully/config -v --shell
```

Here is the kind of output you'll see:

```
I, [2012-01-20T16:42:23.597846 #10716] INFO -- : Loading configuration from /Users/crohr/.restfully...
I, [2012-01-20T16:42:23.598316 #10716] INFO -- : Disabling RestClient::Rack::Compatibility.
I, [2012-01-20T16:42:23.598365 #10716] INFO -- : Enabling Restfully::Rack::BasicAuth.
I, [2012-01-20T16:42:23.598416 #10716] INFO -- : Enabling Rack::Cache.
I, [2012-01-20T16:42:23.598454 #10716] INFO -- : Requiring ApplicationVndBonfireXml...
I, [2012-01-20T16:42:23.640362 #10716] INFO -- : Starting...
cache: [GET ] miss, store
cache: [GET /locations] miss, store
cache: [GET /locations/fr-inria/storages] miss, store
cache: [GET /locations/fr-inria/storages/3] miss, store
cache: [GET /locations/fr-inria/networks] miss, store
cache: [GET /locations/fr-inria/networks/4] miss, store
cache: [GET ] fresh
cache: [GET /experiments] miss, store
cache: [POST /experiments] invalidate, pass
cache: [GET /experiments/59] miss, store
cache: [GET /experiments/59/computes] miss, store
I, [2012-01-20T16:42:26.740140 #10716] INFO -- : I'm going to submit 5 VMs...
cache: [POST /experiments/59/computes] invalidate, pass
cache: [GET /locations/fr-inria/computes/21257] miss, store
I, [2012-01-20T16:42:28.715626 #10716] INFO -- : Submitted VM#21257 with IP=172.18.7.61, in cluster
cache: [POST /experiments/59/computes] invalidate, pass
cache: [GET /locations/fr-inria/computes/21258] miss, store
I, [2012-01-20T16:42:30.846760 #10716] INFO -- : Submitted VM#21258 with IP=172.18.7.62, in cluster
cache: [POST /experiments/59/computes] invalidate, pass
cache: [GET /locations/fr-inria/computes/21259] miss, store
I, [2012-01-20T16:42:33.176797 #10716] INFO -- : Submitted VM#21259 with IP=172.18.7.63, in cluster
cache: [POST /experiments/59/computes] invalidate, pass
cache: [GET /locations/fr-inria/computes/21260] miss, store
I, [2012-01-20T16:42:38.486076 #10716] INFO -- : Submitted VM#21260 with IP=172.18.7.64, in cluster
cache: [POST /experiments/59/computes] invalidate, pass
cache: [GET /locations/fr-inria/computes/21261] miss, store
I, [2012-01-20T16:42:44.105053 #10716] INFO -- : Submitted VM#21261 with IP=172.18.7.65, in cluster
```

You could also specify a specific HOST of a specific CLUSTER. Just pass `<host>hostname</host>` in your compute creation request. For instance, if you got resources at INRIA, you might get nodes such as `paradent-1.rennes.grid5000.fr` (this will be displayed in your reservation details). Therefore, you could ask that 5 VMs be started on the same physical host by specifying the HOST environment variable when you start the script:

```
$ CLUSTER=5ff33cb5ad0e64302ef64f55a08817d818780b81 HOST=paradent-1.rennes.grid5000.fr COUNT=5 restful...
```

Please refer to [Overview of Compute](#) to know how to specify HOST and CLUSTER in the OCCI request.

What this does under the hood

If you're wondering how this feature works, here are some details:

- when you create a reservation, the API will reserve the adequate number of Grid'5000 nodes at INRIA, using the existing Grid'5000 API.
- when your reservation starts on Grid'5000, it executes a script that will deploy a specific image (this is one of the prominent feature of Grid'5000: you fully control the physical machine). This image is the same as the one deployed on the nodes of the permanent BonFIRE infrastructure at INRIA.
- when the deployment is terminated, BonFIRE base images are efficiently propagated on all your nodes (using the mighty `taktuk` tool), so that creating and booting VMs from those base images happens in a snap (less than 10s). This, plus other low-level optimizations allow you to start **hundreds** of VMs in a few minutes. We might allow you to pre-copy your own images as well, so that you are able to get the speed boost for your custom images (otherwise there is a small delay the first time a VM is booted from one of these images).
- finally, your (now properly configured) nodes are added to our local OpenNebula installation, which means there is no difference in treatment between permanent and physical resources, except for the fact that only the people that know the cluster UUID can submit VMs on the cluster's physical machines.
- once the nodes have been registered with OpenNebula, we notify you of the availability of your cluster (if you specified a notification address when you created your reservation).
- you can now start VMs on your dedicated cluster!

5.4 Deploying Compute Resources in BonFIRE

5.4.1 Getting Configuration Options

When you create a compute resource on a BonFIRE testbed, you can choose a specific `instance_type`, which will assign a predefined amount of RAM and CPU to your compute resource. Every testbed offers different configurations, and you can discover them using the API.

Listing available configurations

Listing available configurations is easily achieved with a simple GET request:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/configurations
```

```
HTTP/1.1 200 OK
Date: Thu, 08 Dec 2011 13:30:12 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/1.0.0)
Cache-Control: public,max-age=30
```

```
Allow: GET,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "1a29676c02d17b874bce5cf7010a032a"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.081551
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items offset="0" total="5">
    <configuration href="/locations/fr-inria/configurations/custom">
      <name>custom</name>
    </configuration>
    <configuration href="/locations/fr-inria/configurations/medium">
      <name>medium</name>
      <vcpu>4</vcpu>
      <vmem>4096</vmem>
    </configuration>
    <configuration href="/locations/fr-inria/configurations/large">
      <name>large</name>
      <vcpu>8</vcpu>
      <vmem>8192</vmem>
    </configuration>
    <configuration href="/locations/fr-inria/configurations/lite">
      <name>lite</name>
      <vcpu>0.25</vcpu>
      <vmem>256</vmem>
    </configuration>
    <configuration href="/locations/fr-inria/configurations/small">
      <name>small</name>
      <vcpu>1</vcpu>
      <vmem>1024</vmem>
    </configuration>
  </items>
  <link rel="parent" href="/locations/fr-inria" type="application/vnd.bonfire+xml"/>
  <link rel="self" href="/locations/fr-inria/configurations" type="application/vnd.bonfire+xml"/>
  <link rel="top" href="/locations/fr-inria/configurations" type="application/vnd.bonfire+xml"/>
</collection>
```

The value of the name element is what you need to specify as the value for your instance_type element when you create a compute resource.

5.4.2 Deploying Compute Resources with the Portal

Information about how to use Portal to create compute resource can be found here

5.4.3 Deploying Compute Resources with OCCI

Compute resources are the base resources of BonFIRE. This is where you can run your software. In this tutorial you'll learn how to create compute resources using the BonFIRE API.

Prerequisites

A compute resource has to be linked to at least one network resource, and at least one storage resource of type OS. Consequently, before creating a compute resource, you need to find out an existing storage resource that you can use as image, and an existing network resource.

Using the API, it's easy to list these kind of resources for a given testbed. At fr-inria for example, you would fetch the list of existing storage resources like this:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/storages

HTTP/1.1 200 OK
Date: Wed, 26 Oct 2011 15:00:59 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Allow: GET,POST,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "bd7607657ba7261de6ba67a2f059a5a7"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.327973
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <storage href="/locations/fr-inria/storages/49" name="saved-client-experiment#240-image"/>
    <storage href="/locations/fr-inria/storages/52" name="saved-client-experiment#257-image"/>
    <storage href="/locations/fr-inria/storages/53" name="saved-client-experiment#266-image"/>
    <storage href="/locations/fr-inria/storages/161" name="VM-ipperf"/>
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2"/>
    <storage href="/locations/fr-inria/storages/166" name="BonFIRE Debian Squeeze 2G v2"/>
    <storage href="/locations/fr-inria/storages/167" name="BonFIRE Zabbix Aggregator v4"/>
    <storage href="/locations/fr-inria/storages/184" name="iperf-test4"/>
    <storage href="/locations/fr-inria/storages/236" name="Storage name"/>
    <storage href="/locations/fr-inria/storages/264" name="name for my storage"/>
    <storage href="/locations/fr-inria/storages/273" name="squeeze 2G 2"/>
    <storage href="/locations/fr-inria/storages/281" name="BonFIRE Debian Squeeze 10G v2"/>
  </items>
</collection>
```

Storage images provided by BonFIRE for use as the OS of your compute resources are those whose name starts with BonFIRE Let's assume you'd like to deploy a compute resource based on the Debian Squeeze distribution (BonFIRE Debian Squeeze v2, URI=""/locations/fr-inria/storages/165").

Then, list the available network resources like this:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/networks
```

```
HTTP/1.1 200 OK
Date: Wed, 26 Oct 2011 15:03:45 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Allow: GET,POST,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "23522379bbb92138c5b5e467c4a925fc"
```

```
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.372999
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <network href="/locations/fr-inria/networks/20" name="Public Network"/>
    <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN"/>
  </items>
</collection>
```

Here you can see that there are two network resources available. See [OCCI How-To](#) for more information about the kind of networks. For now let's assume that we want to link our compute resource to the BonFIRE WAN network (URI=''/locations/fr-inria/networks/47').

Now that we know which image and which network to use, we can create the compute resource.

Creating a compute resource

The following assumes that you already created an experiment resource, with ID 4056. Follow [OCCI/API via HTTP\(cURL\)](#) if you don't know how to create an experiment resource.

To create a compute resource, you need the know following things:

- the type of instance you want to create (tiny, small, large, etc.). This specifies how much computing power you need, in terms of number of CPUs and RAM. See [Testbed Specificities](#) for up to date information;
- the location at which you want to create the resource (here: fr-inria);
- the image to deploy (here: BonFIRE Debian Squeeze v2);
- the network to which the resource will be attached (here: BonFIRE WAN).

Then, just send a POST request on your experiment computes URI, like this:

```
$ curl -kni https://api.bonfire-project.eu/experiments/4056/computes \
-X POST -H'Content-Type: application/vnd.bonfire+xml' \
-d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>my-vm</name>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/fr-inria/storages/165" />
  </disk>
  <nics>
    <nic>
      <network href="/locations/fr-inria/networks/47" />
    </nic>
  </nics>
  <location href="/locations/fr-inria" />
</compute>'
```

```
HTTP/1.1 201 Created
Date: Wed, 26 Oct 2011 15:14:55 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Location: https://api.bonfire-project.eu/locations/fr-inria/computes/3585
Content-Type: application/vnd.bonfire+xml; charset=utf-8
```

```
Cache-Control: no-cache
X-UA-Compatible: IE=Edge, chrome=1
X-Runtime: 0.326905
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/3585">
  <id>3585</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <name>my-vm</name>
  <instance_type>lite</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2"/>
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN"/>
    <ip>172.18.1.242</ip>
    <mac>02:00:ac:12:01:f2</mac>
  </nic>
</compute>
```

If everything goes well, you'll receive a 201 Created status code, and you get back the URI of the newly created resource in the Location header.

You can use this URI to regularly poll the compute state, or other properties:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/computes/3585
```

```
HTTP/1.1 200 OK
Date: Wed, 26 Oct 2011 15:16:47 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "386add2827c81615f6cddc8749c9b45e"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.519026
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/3585">
  <id>3585</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <name>my-vm</name>
  <instance_type>lite</instance_type>
  <state>ACTIVE</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2"/>
    <type>DISK</type>
```

```
<target>xvda</target>
</disk>
<nic>
  <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN"/>
  <ip>172.18.1.242</ip>
  <mac>02:00:ac:12:01:f2</mac>
</nic>
<link rel="experiment" href="/experiments/4056" type="application/vnd.bonfire+xml"/></compute>
```

After some time, your compute resource will be active and running, and you'll be able to start an SSH session between your machine and the compute resource. See [SSH Gateway Configuration](#) to learn how to do this. Depending on the testbed used, the published state for machines that are ready may vary between ACTIVE, RUNNING and on.

Placing compute resources on specific hosts or clusters

Note that you can specify placement constraints for your compute resources. For instance, if you're using the [On Request Resources](#) facility you may want to deploy VMs on the cluster you reserved instead of the default one. Also, within a cluster, you may want to specify on which particular host you want to deploy a VM (useful for testing influence of colocation for instance).

Placement constraints are very easily achieved by just adding a `<host>` and/or `<cluster>` element to your creation request. For instance, let's assume I have reserved on-demand resources and that I have now access to cluster `5ff33cb5ad0e64302ef64f55a08817d818780b81`. If I want to submit a VM on this cluster, irrespective of a particular physical host, I would simply do as follows:

```
$ curl -kni https://api.bonfire-project.eu/experiments/4056/computes \
-X POST -H'Content-Type: application/vnd.bonfire+xml' \
-d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>my-vm</name>
  <cluster>5ff33cb5ad0e64302ef64f55a08817d818780b81</cluster>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/fr-inria/storages/165" />
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/47" />
  </nic>
  <location href="/locations/fr-inria" />
</compute>'
```

And then that VM would be scheduled on one of the physical hosts of that cluster.

Similarly, you could specify a particular physical host as follows (if no `<cluster>` element is present, it will try to find node-1 in the default cluster. You must specify `<cluster>` if the host is not in the default cluster):

```
$ curl -kni https://api.bonfire-project.eu/experiments/4056/computes \
-X POST -H'Content-Type: application/vnd.bonfire+xml' \
-d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>my-vm</name>
  <host>node-1</host>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/fr-inria/storages/165" />
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/47" />
  </nic>
</compute>'
```

```
</nic>
<location href="/locations/fr-inria" />
</compute>'
```

In this case the scheduler will try to schedule the VM on node-1 of the default (i.e. permanent) cluster. Please note that if the host is full, your VM might stay in PENDING state for a long time.

Host names can be known by looking at the monitoring infrastructure (for permanent resources) or, for on-demand requests, the names are returned to you when the reservation is created.

Updating a compute resource

You can change a compute state by updating the compute resource representation. For example, if you wanted to shut down the resource, you could send the following request (only works for OpenNebula testbeds):

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/computes/3585 \
-X PUT -H'Content-Type: application/vnd.bonfire+xml' -d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <state>SHUTDOWN</state>
</compute>'
```

```
HTTP/1.1 202 Accepted
Date: Wed, 26 Oct 2011 15:22:34 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Location: https://api.bonfire-project.eu/locations/fr-inria/computes/3585
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge, chrome=1
X-Runtime: 0.322056
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/3585">
  <id>3585</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <name>my-vm</name>
  <instance_type>lite</instance_type>
  <state>ACTIVE</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2" />
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nics>
    <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN" />
    <ip>172.18.1.242</ip>
    <mac>02:00:ac:12:01:f2</mac>
  </nics>
</compute>
```

You'll receive a 202 Accepted status code, meaning your request has been accepted, and will be processed in the near future.

After some time, if you try to fetch the compute resource, you'll see that the state has been updated to the requested state:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/computes/3585

HTTP/1.1 200 OK
Date: Wed, 26 Oct 2011 15:23:47 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Allow: GET,DELETE,PUT,OPTIONS,HEAD
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "759dbe01657cb71bc58a4b8a99ee2c01"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.259581
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/3585">
  <id>3585</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <name>my-vm</name>
  <instance_type>lite</instance_type>
  <state>DONE</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2"/>
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN"/>
    <ip>172.18.1.242</ip>
    <mac>02:00:ac:12:01:f2</mac>
  </nic>
  <link rel="experiment" href="/experiments/4056" type="application/vnd.bonfire+xml"/></compute>
```

Migrating a compute resource

You can change placement of your compute by sending a migration request.

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/computes/3585 \
-X PUT -H'Content-Type: application/vnd.bonfire+xml' -d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <migration_strategy>cold</migration_strategy>
  <host>targethost</host>
</compute>'
```

The `migration_strategy` parameter supports the following values:

- `cold`: The compute will be suspended and migrated to the target host. This requires moving all the storage which will take time. Once it's finished the compute will be resumed to its previous state. This mechanism is currently supported at the EPCC, HLRS and INRIA testbeds.
- `live`: The compute will be livemigrated using the hypervisor functionality. This will allow for an immediate availability in the target host with virtually no down-time. This mechanism is not yet supported by any of the BonFIRE testbeds.

`targethost` is the name of the physical host to migrate to.

After the request, you'll receive a 202 Accepted status code, meaning your request has been accepted, and will be processed in the near future.

Deleting a compute resource

As for all other resources, simply send a DELETE request to the resource URI:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/computes/3585 -X DELETE

HTTP/1.1 204 No Content
Date: Wed, 26 Oct 2011 15:26:18 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.233121
Vary: Authorization,Accept
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Final note

If you're going to script your experiments like this, you should probably have a look at the [Restfully](#) tool, which abstracts most of the work.

Also, there are a number of advanced topics about compute resources, such as saving one of the compute disks into a new storage resource (see [Overview of Storage](#)), [List of all Contextualisation Names](#), and creating [Overview of Monitoring Options in BonFIRE](#).

5.4.4 Deploying Compute Resources with Experiment Descriptors

Creating compute resource using JSON experiment Descripotr

JSON is one of BonFIRE experiment descriptors. It provides information about how to create resource of compute using JSON experiment descriptor

5.4.5 Deploying Compute Resources with Restfully

Restfully Snippets

This is a page that lists all the useful Restfully snippets we can come up with. If you have interesting snippets of your own, please send them to support@bonfire-project.eu.

Creating a VM

Here is the fire.rb snippet:

```
image_name = ENV['IMAGE'] || "BonFIRE Debian Squeeze v3"
network_name = ENV['NETWORK'] || "BonFIRE WAN"
location_name = ENV['LOCATION'] || "fr-inria"
walltime = ENV['WALLTIME'] || 7200
```

```
instance_type = ENV['TYPE'] || "lite"

location = root.locations[location_name.to_sym]
fail "Can't find location #{location_name.inspect}." if location.nil?

image = location.storages[image_name.to_sym]
fail "Can't find image #{image_name.inspect}." if image.nil?

network = location.networks[network_name.to_sym]
fail "Can't find image #{network_name.inspect}." if network.nil?

experiment = root.experiments.submit(
  :name => "Experiment Name",
  :description => "Started on #{Time.now.to_s}",
  :walltime => walltime
)
fail "Can't create experiment" if experiment.nil?

begin
  compute = experiment.computes.submit(
    :name => Time.now.to_i,
    :disk => [
      {:storage => image}
    ],
    :nic => [
      {:network => network}
    ],
    :instance_type => instance_type,
    :location => location
  )
  experiment.update(:status => "running")

  pp compute
rescue => e
  fail e
end
```

Run it like this:

```
$ restfully -c ~/.restfully/config --shell -v http://doc.bonfire-project.eu/R3/_static/examples/restful
```

By default this will launch a lite VM at fr-inria. You can set any of the environment variables to launch on a different location, with a different image, etc.

The --shell flag is very useful since it will give you back a shell at the end of the script, where you'll have access to the full context of the script (i.e. you can do a pp compute to display the compute details, and manipulate any other variable defined in the script).

Creating a VM at HP

For now there is a separate snippet for HP, since it requires a specific private network for the experiment. Here is the content of the hp.rb snippet:

```
#!/usr/bin/env ruby
require 'restfully/addons/bonfire'

Signal.trap("INT") do
```

```

    puts "Deleting experiment, and exiting..."
    @experiment.delete unless @experiment.nil?
    exit
end

hp = root.locations[:'uk-hplabs']
@experiment = root.experiments.submit(:name => "HP experiment", :description => "whatever")
pp @experiment
network = @experiment.networks.submit(:name => "some random network name", :location => hp)
pp network
image = hp.storages.first
pp image
compute = @experiment.computes.submit(:location => hp, :name => "compute name", :disk => [{:storage =>
  pp compute

puts "Hit CTRL-C to terminate experiment."

```

Run it like this:

```
$ restfully -c ~/.restfully/config --shell -v http://doc.bonfire-project.eu/R3/_static/examples/restfully
```

Creating VMs on multiple testbeds

A very simple (and not error-proof) example looks like this:

```

# This is a recorded Restfully session, using the --record option of restfully.
# You can replay it easily with: $ restfully -c ~/.restfully/config.yml --replay --shell FILE
ibbt = root.locations[:'be-ibbt']
inria = root.locations[:'fr-inria']
i1 = ibbt.storages[0]
n1 = ibbt.networks[0]
i2 = inria.storages[0]
n2 = inria.networks[1]
exp = root.experiments.submit(:name => "Federation using IBBT and INRIA", :description => "Quick example")
c1 = exp.computes.submit(:location => ibbt, :instance_type => "lite", :nic => [{:network => n1}], :disk => [{:storage => i1}], :disk_size => 10)
c2 = exp.computes.submit(:location => inria, :instance_type => "lite", :nic => [{:network => n2}], :disk => [{:storage => i2}], :disk_size => 10)
exp.update(:status => "running")
pp exp

```

Run it like this:

```
$ restfully -c ~/.restfully/config --replay http://doc.bonfire-project.eu/R3/_static/examples/restfully
```

And, once again, you'll get a shell where you can interact with the created resources.

STORAGE

6.1 Overview of Storage

When deploying compute resources, a VM image needs to be selected that has a particular OS and storage size. In BonFIRE, the maximum size of a VM image is 10GB, which might not be enough for running a particular experiment. In this case, the VMs can be extended with *Datablock Storage* resources, for which you can define the file system of (e.g., ext3, jfs, reiserfs) and the desired storage size, which are mounted in the VMs. It is also possible to make this storage extend the root partition of your compute node instead of being mounted as a separate partition.

Another special type of storage resource in BonFIRE is the *shared storage*. At the moment this can only be created at the be-ibbt testbed. Unlike the other storage types, shared storages are accessible by multiple computes at the same time. If the storage was created outside the scope of an experiment, it can even be accessed by computes from different experiments.

If you're starting an experiment in BonFIRE, you may also want to have a look at [Advanced Storage Features](#).

6.2 Datablock Storage

If the OS storage provided with the VM images offered in BonFIRE is not enough, VM's storage can be extended with DATABLOCK resources, for which you can define the file system of (e.g., ext3, jfs, reiserfs) and the desired storage size. These need to be *mounted to the VMs* that you deploy. It is also possible to make this storage *extend the root partition* of your compute node instead of being mounted as a separate partition. See the *storage attributes* page for more details about the OCCI configuration options for storage.

By default, storage resources that are created in BonFIRE are duplicated when they are deployed with compute resources, and are destroyed after the experiment duration has finished. Whilst other resources (compute and network) must be created within the context of an experiment, storage resources can be created outside of an experiment and will therefore outlive the duration of any experiments that use that storage resource. Moreover, it is possible to *persist the storage resources* on the OpenNebula testbeds in BonFIRE (see the *Infrastructure* page), so that data written to the storage during the execution of an experiment is saved when the compute resource is shut down.

For information about how to use datablock storage, please see the following links:

6.2.1 Datablock Portal How-To

Storage resources can be created in two ways in BonFIRE. You can either create a storage resource within an Experiment, or you can create a storage resource independently of any Experiment. In the first case the resource will be deleted at the end of the Experiment walltime, while in the second case it will stay alive as long as you wish, and it is your responsibility to delete it when you no longer need it.

Creating a storage resource within an experiment

We assume that you have already created an experiment container. Follow [create experiment](#) if you don't know how to create an experiment container.

The following is a step by step process for creating a storage resource within an experiment:

1. On the Storage Resources panel, select location beside “Add Storage at”, then press the button add:

The screenshot shows the BonFIRE experiment management interface. At the top, there's a summary of the experiment details: Experiment ID: /experiments/2991, User ID: yllang, Groups: yllang, Creation Time: Wed, 11/28/12 11:54:23 UTC, Last Updated: Wed, 11/28/12 11:54:23 UTC, Expires: Wed, 11/28/12 12:24:23 UTC, and a link to (Show XML). Below this is a navigation bar with tabs: Resources (selected), Site Interconnection, Elasticity, and Monitoring. The main content area has three sections: Compute Resources, Storage Resources, and Network Resources. In the Storage Resources section, there's a table with columns: Name, Id, Description, Type, Size, FS, State, Pub., Per., and Del. At the bottom of this section is a button labeled "Add Storage at" with a dropdown menu set to "be.ibbt" and an "add" button next to it. This "add" button is circled in red in the screenshot.

2. fill in the field

3. press the button Continue, you can get an overview about the created resource in OCCI

Creating a storage resource outside an experiment

In this case, no experiment container is required. A storage resource can be created independently.

1. go to the start page, and select Resources:
2. select location beside “Add Storage at”, then press the button add:
3. fill in the field
4. press the button Continue, you can get an overview about the created resource in OCCI

BonFIRE >> Experiments >> Experiment Details >> Create Storage

Create Storage Resource

Create Storage Resource

Name ?
Tutorial Storage

Group ?
yliang

Description ?
used for tutorial

Type
DATABLOCK

Filesystem ?
ext3

Size in MByte ?
10

Persistent ?

→ Continue **► Finish**

BonFIRE >> Experiments >> Experiment Details >> Create Storage

Create Storage Resource

Review OCCI Request for Storage Creation

```
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Tutorial-Storage</name>
  <groups>yliang</groups>
  <description>used for tutorial</description>
  <type>DATABLOCK</type>
  <size>10</size>
  <fstype>ext3</fstype>
  <link href="/locations/be-ibbt" rel="location"/>
</storage>
```

Ignore XML Validation Errors

→ Continue **► Finish**

BonFIRE

Hello *yliang!*

Please choose your destination: There are [Resources](#) and of course [Experiments](#) — the two main items, which can accessed via the icons top left. If you have any questions, please consult the [FAQ](#). We are always open for a dialogue and hungry for feedback, just [Contact us](#).

Further documentation on how to use the BonFIRE services is available [here](#).

— **We've got News**

We've got news. Right on the front page. You might have noticed already.

BonFIRE

BonFIRE >> Resources

Sitewide Resources

(show infrastructure health overview)

Storage Resources

Name	ID	Description	Type	Size	FS	State	Pub.	Per.	Del.
Aggregator-Storage-aggrc /locations/uk-hplabs/storages/vol-41-1-		Could not retrieve data: Error during execution. The broker said: 403 Forbidden - You are not authorized to access this resource							X
BonFIRE Debian Squeeze : /locations/uk-hplabs/storages/vol-74d2e			OS	2049MB		READY	true	false	X
BonFIRE Debian Squeeze : /locations/uk-hplabs/storages/vol-6157f			OS	2048MB		READY	true	false	X
BonFIRE Zabbix Aggregat /locations/uk-hplabs/storages/vol-8ecf9			OS	3401MB		READY	true	false	X
Aggregator-Storage-aggrc /locations/uk-hplabs/storages/vol-41-1-		Could not retrieve data: Error during execution. The broker said: 403 Forbidden - You are not authorized to access this resource							X
BonFIRE Debian Squeeze : /locations/uk-hplabs/storages/vol-6b42f			OS	604MB		READY	true	false	X
Aggregator-Storage-testib /locations/uk-hplabs/storages/vol-41-1-		Could not retrieve data: Error during execution. The broker said: 403 Forbidden - You are not authorized to access this resource							X
BonFIRE Zabbix Aggregat /locations/uk-hplabs/storages/vol-3d6bc			OS	6144MB		READY	true	false	X
BonFIRE Debian Squeeze : /locations/uk-hplabs/storages/vol-7ac0f			OS	2049MB		READY	true	false	X
Aggregator-Storage-aggrc /locations/uk-hplabs/storages/vol-41-1-		Could not retrieve data: Error during execution. The broker said: 403 Forbidden - You are not authorized to access this resource							X

Filter by Site: <all>

Add Storage at: be-ibm ▾ add

BonFIRE >> Resources >> Create Storage

Create Storage Resource

Create Storage Resource

Name ?
tutorial

Group ?
yliang

Description ?
for tutorial

Filesystem ?
ext3

Size in MByte ?
10

Persistent ?

→ Continue ► Finish

BonFIRE >> Resources >> Create Storage

Create Storage Resource

Review OCCI Request for Storage Creation

```
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>tutorial</name>
  <groups>yliang</groups>
  <description>for tutorial</description>
  <type>DATABLOCK</type>
  <size>10</size>
  <fstype>ext3</fstype>
  <persistent>YES</persistent>
  <link href="/locations/uk-epcc" rel="location"/>
</storage>
```

Ignore XML Validation Errors

→ Continue ► Finish

Saving an existing OS or DATABLOCK storage as a new storage

Let's say you created a compute resource with a storage of type OS attached to it. You logged on the machine, installed some software, and you now want to save your changes into a new storage, so that you can use that customized storage from other compute resources. What you need to do is a SAVE_AS action, i.e. update the compute resource to specify which disk you want to save, and under which name. Let's see how this can be done.

Assuming you have created a compute resource whose description is the following, and double click on the name of your resource

The screenshot shows the 'Experiment Details' page for an experiment named 'tutorial'. The 'Compute Resources' section lists one entry:

Name	Id	Type	Cpus	Mem	VM Image	Wan IP	SSH	State	Del.
tutorial-for-save-as	/locations/fr-inria/computes/3	lite	0.5	256MB	BonFIRE Debian	172.18.250.20	OK	RUNNING	

A red dashed circle highlights the 'Name' column for the first row. The 'Storage Resources' and 'Network Resources' sections are also visible below, each with their own tables and filtering options.

You can tell the infrastructure to save the disk as a new storage by giving a name and then press the green button Set Save-As:

Note that the new storage resource will be really created **only after you explicitly shut down** the compute resource. Shutting down a compute resource can be done by updating its state to SHUTDOWN, and then press the green button Switch State:

After some time, the compute resource will be shut down, the new storage resource will appear, meaning the storage resource can now be used.

6.2.2 Datablock OCCI How-To

Storage resources can be created in two ways in BonFIRE. You can either create a storage resource within an Experiment, or you can create a storage resource independently of any Experiment. In the first case the resource will be deleted at the end of the Experiment walltime, while in the second case it will stay alive as long as you wish, and it is your responsibility to delete it when you no longer need it.

Obviously the second case is useful if you want to persist data between experiments. Also, storage resources created from a SAVE_AS command fall into the second case, i.e. they are not deleted at the end of the experiment that contained the compute resource from which it was created.

BonFIRE >> Experiments >> Experiment Details >> Resource Details

Details for Compute Resource /locations/fr-inria/computes/33667

ID: /locations/fr-inria/computes/33667
Name: tutorial-for-save-as
Groups: yliang
Instance Type: lite
CPUs: 0.5
Memory: 256MB
OS Image: BonFIRE Debian Squeeze 2G v5 (/locations/fr-inria/storages/2033)
Wan IP: 172.18.250.206
State: RUNNING
Logfile: <http://frontend.bonfire.grid5000.fr/logs/33667.log>
[\(Show XML\)](#)

Resource State

The state of this resource is 'RUNNING'.

Switch state:

More information about VM states can be found [here](#).

Save As

This compute's VMImage is not yet set to be saved.

Save VMImage as:

More information on saving VM images can be found [here](#).

BonFIRE >> Experiments >> Experiment Details >> Resource Details

Details for Compute Resource /locations/fr-inria/computes/33667

ID: /locations/fr-inria/computes/33667
Name: tutorial-for-save-as
Groups: yliang
Instance Type: lite
CPUs: 0.5
Memory: 256MB
OS Image: BonFIRE Debian Squeeze 2G v5 (/locations/fr-inria/storages/2033)
Wan IP: 172.18.250.206
State: RUNNING
Logfile: <http://frontend.bonfire.grid5000.fr/logs/33667.log>
[\(Show XML\)](#)

Resource State

The state of this resource is 'RUNNING'.

Switch state:

More information about VM states can be found [here](#).

Save As

This compute's VMImage is not yet set to be saved.

Save VMImage as:

More information on saving VM images can be found [here](#).

Creating a storage resource within an experiment

The following assumes that you already created an experiment resource, with ID 1274. Follow [OCCI/API via HTTP\(cURL\)](#) if you don't know how to create an experiment resource.

Here is how you would create a storage resource with the cURL tool:

```
$ curl -kni https://api.bonfire-project.eu/experiments/1274/storages \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Storage name</name>
  <description>Storage description</description>
  <type>DATABLOCK</type>
  <size>1024</size>
  <fstype>ext3</fstype>

  <link rel="location"
        href="/locations/fr-inria" />
</storage>
'
```

Notice the `location` link, which specified on which testbed you want to create the resource. Here is what you would get back:

```
HTTP/1.1 201 Created
Date: Wed, 20 Jul 2011 07:45:20 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 bonfire-api (bonfire-api/0.10.1)
Location: https://api.bonfire-project.eu/locations/fr-inria/storages/234
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.801753
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/234">
  <id>234</id>
  <name>Storage name</name>
  <type>DATABLOCK</type>
  <description>Storage description</description>
  <size>1024</size>
  <fstype>ext3</fstype>
  <public>NO</public>
  <persistent>NO</persistent>
</storage>
```

Also note the `Location` HTTP header, which tells you where the resource can be found (here: <https://api.bonfire-project.eu/locations/fr-inria/storages/234>).

Creating a non-public storage resource outside an experiment

To create a storage resource outside an Experiment you don't need any requirement. Just POST your resource description directly to the location you want it to be at.

Here is how you would do it with the cURL tool:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/storages \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Storage name</name>
  <description>Storage description</description>
  <type>DATABLOCK</type>
  <size>1024</size>
  <fstype>ext3</fstype>
</storage>
'
```

Here is what you would get back:

```
HTTP/1.1 201 Created
Date: Wed, 20 Jul 2011 07:50:04 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 bonfire-api (bonfire-api/0.10.1)
Location: https://api.bonfire-project.eu/locations/fr-inria/storages/236
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.701750
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/236">
  <id>236</id>
  <name>Storage name</name>
  <type>DATABLOCK</type>
  <description>Storage description</description>
  <size>1024</size>
  <fstype>ext3</fstype>
  <public>NO</public>
  <persistent>NO</persistent>
</storage>
```

Again, the Location HTTP header tells you where the resource can be found.

Creating a public storage resource outside an experiment

To create a public storage resource outside an Experiment you don't need any requirement. Just POST your resource description directly to the location you want it to be at.

Here is how you would do it with the cURL tool:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/storages \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>testdone</name>
  <groups>yliang</groups>
  <description>testdone</description>
  <type>DATABLOCK</type>
  <size>1024</size>
  <fstype>ext3</fstype>
  <public>YES</public>
  <link href="/locations/fr-inria" rel="location"/>
</storage>
```

```
</storage>
```

```
,
```

Here is what you would get back:

```
HTTP/1.1 201 Created
Date: Mon, 06 May 2013 14:18:06 GMT
Server: thin 1.5.0 codename Knife
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/1.9.94)
Location: https://api.bonfire-project.eu/locations/fr-inria/storages/340
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.304892
Vary: Authorization,Accept
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/340">
  <id>340</id>
  <name>testdone</name>
  <user_id href="/locations/fr-inria/users/27">yliang</user_id>
  <groups>yliang</groups>
  <state>LOCKED</state>
  <type>DATABLOCK</type>
  <description>testdone</description>
  <size>1024</size>
  <fstype>ext3</fstype>
  <public>YES</public>
  <persistent>NO</persistent>
  <datastore>default</datastore>
  <datastore_id>1</datastore_id>
</storage>
```

Saving an existing OS or DATABLOCK storage as a new storage

Let's say you created a compute resource with a storage of type OS attached to it. You logged on the machine, installed some software, and you now want to save your changes into a new storage, so that you can use that customized storage from other compute resources. What you need to do is a SAVE_AS action, i.e. update the compute resource to specify which disk you want to save, and under which name. Let's see how this can be done.

Assuming you have created a compute resource whose description is the following:

```
$ curl -kn https://api.bonfire-project.eu/locations/fr-inria/computes/1469

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/1469">
  <id>1469</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>my vm</name>
  <instance_type>small</instance_type>
  <state>ACTIVE</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2"/>
    <type>DISK</type>
    <target>xvda</target>
  </disk>
</compute>
```

```

</disk>
<nic>
  <network href="/locations/fr-inria/networks/20" name="Public Network"/>
  <ip>131.254.204.187</ip>
  <mac>02:00:83:fe:cc:bb</mac>
</nic>
<link rel="experiment" href="/experiments/1349" type="application/vnd.bonfire+xml"/></compute>
```

You can tell the infrastructure to save the first disk as a new storage by issuing the following request:

```
$ curl -kn https://api.bonfire-project.eu/locations/fr-inria/computes/1469 \
-X PUT -H 'Content-Type: application/vnd.bonfire+xml' -d '
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <disk id="0"><save_as name="name for my storage" /></disk>
</compute>
'
```

In the response you receive, you can see the URI at which the newly created storage will be available (here: /locations/fr-inria/storages/264):

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/1469">
  <id>1469</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>my vm</name>
  <instance_type>small</instance_type>
  <state>ACTIVE</state>
  <disk id="0">
    <save_as href="/locations/fr-inria/storages/264" />
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2" />
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/20" name="Public Network" />
    <ip>131.254.204.187</ip>
    <mac>02:00:83:fe:cc:bb</mac>
  </nic>
</compute>
```

Note that the new storage resource will be really created **only after you explicitly shut down** the compute resource. Until then, you can get the description of the storage resource but its size element will be absent (meaning it was not physically created yet):

```
$ curl -kn https://api.bonfire-project.eu/locations/fr-inria/storages/264

<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/264">
  <id>264</id>
  <name>name for my storage</name>
  <type>OS</type>
  <public>NO</public>
  <persistent>NO</persistent>
</storage>
```

Shutting down a compute resource can be done by updating its state to SHUTDOWN:

```
$ curl -kn https://api.bonfire-project.eu/locations/fr-inria/computes/1469 \
-X PUT -H 'Content-Type: application/vnd.bonfire+xml' -d '
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
```

```
<state>SHUTDOWN</state>
</compute>
'

<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/1469">
  <id>1469</id>
  <cpu>1</cpu>
  <memory>1024</memory>
  <name>my vm</name>
  <instance_type>small</instance_type>
  <state>ACTIVE</state>
  <disk id="0">
    <save_as href="/locations/fr-inria/storages/264" />
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2" />
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/20" name="Public Network" />
    <ip>131.254.204.187</ip>
    <mac>02:00:83:fe:cc:bb</mac>
  </nic>
</compute>
```

After some time, the compute resource will be shut down, and the size element of the new storage resource will appear, meaning the storage resource can now be used:

```
$ curl -kn https://api.bonfire-project.eu/locations/fr-inria/storages/264
```

```
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/264">
  <id>264</id>
  <name>name for my storage</name>
  <state>LOCKED</state>
  <type>OS</type>
  <size>604</size>
  <public>NO</public>
  <persistent>NO</persistent>
</storage>
```

Note: This method is a one-shot backup. If you create a new compute resource with the storage resource created by the SAVE_AS command, and add new data on it, then the additional data will not be automatically persisted when you shut down the compute resource. See the next section to learn how to do this.

Has my SAVE_AS succeeded?

In order to find out whether a SAVE_AS operation successfully created a valid disk image, the created STORAGE resource should be polled until the STATE goes to READY (success) or ERROR (the SAVE_AS failed).

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/storages/264
```

```
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/264">
  <id>264</id>
  <name>name for my storage</name>
  <state>READY|ERROR</state>
  <type>OS</type>
  <size>604</size>
  <public>NO</public>
  <persistent>NO</persistent>
</storage>
```

Saving Storages under a different Group

When saving storage resources, a group by which the newly created storage will be accessible can optionally be specified. If the group specification is omitted, the resource manager will place the new storage resource under the group the respective experiment belongs to (Note: this might be different from the group the compute resource belongs to).

The group for a saved storage is specified by adding an additional `groups` element to the `save_as` element in the request XML:

```
$ curl -kn https://api.bonfire-project.eu/locations/fr-inria/computes/1469 \
-X PUT -H 'Content-Type: application/vnd.bonfire+xml' -d '
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <disk id="0">
    <save_as name="name for my storage">
      <groups>MyGroup</groups>
    </save_as>
  </disk>
</compute>
'
```

Note that currently, the group specified under `save_as` will not be returned in the resulting OCCI of the compute resource.

6.2.3 Datablock Experiment Descriptor How-To

Storage resources can be created in two ways in BonFIRE. You can either create a storage resource within an Experiment, or you can create a storage resource independently of any Experiment. In the first case the resource will be deleted at the end of the Experiment walltime, while in the second case it will stay alive as long as you wish, and it is your responsibility to delete it when you no longer need it.

Creating a storage resource using JSON experiment Descripotr

JSON is one of BonFIRE experiment descriptors. It provides information about how to create storage using JSON experiment descriptor

6.2.4 Download Datablock Images How-to

This tutorial explains how to download an entire stored datablock image from BonFIRE.

Scenario 1

We have a small datablock attached to our VM. It is smaller than the free space available in the root filesystem. We're going to download the image to an external (to BonFIRE) machine and mount it there so we can inspect its contents.

1. Check the device name, whether the image is mounted and, if so, where

```
root@test-13972:~# mount
/dev/mapper/vgroot-rootfs on / type ext3 (rw,errors=remount-ro)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
udev on /dev type tmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
```

```
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)
/dev/xvda1 on /boot type ext3 (rw)
/dev/xvde on /tmp/db type ext3 (rw)
root@test-13972:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vgroot-rootfs
                                         460M  407M   54M  89% /
tmpfs           127M    0  127M   0% /lib/init/rw
udev            109M   76K  109M   1% /dev
tmpfs           127M    0  127M   0% /dev/shm
/dev/xvda1       36M   30M   5.9M  84% /boot
/dev/xvde        31M   4.5M   25M  15% /tmp/db
```

In this case we see that our image is /dev/xvde, mounted on /tmp/db. It is 31MB in size and we have 54MB available in / so we have sufficient room to store a temporary copy of it in the root filesystem.

2. Unmount the datablock

```
root@test-13972:~# umount /tmp/db
```

3. Create a temporary copy in /tmp

```
root@test-13972:~# dd if=/dev/xvde of=/tmp/db.img bs=4k
8192+0 records in
8192+0 records out
33554432 bytes (34 MB) copied, 0.373705 s, 89.8 MB/s
```

4. On the external machine, use scp to download the image. This requires that you have previously configured the external machine to connect to the BonFIRE WAN address through the SSH gateway (see [SSH Gateway Configuration](#) for details).

```
[root@foo ~]# scp root@172.18.6.17:/tmp/db.img ./db.img
db.img                                100%    32MB   16.0MB/s   00:02
```

5. Mount the downloaded image using the loop device and inspect its contents.

```
[root@foo ~]# mkdir db
[root@foo ~]# mount -o loop ./db.img db
[root@foo ~]# ls ./db
hello.txt  lost+found
[root@foo ~]# cat ./db/hello.txt
Hello, world.
```

Scenario 2

As scenario 1, but this time the datablock image is too large to store in the root filesystem (1GB in this case) so we use a slightly larger temporary datablock to store a copy of it.

```
root@test-13974:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vgroot-rootfs
                                         460M  375M   86M  82% /
tmpfs           127M    0  127M   0% /lib/init/rw
udev            109M   80K  109M   1% /dev
tmpfs           127M    0  127M   0% /dev/shm
/dev/xvda1       36M   30M   5.9M  84% /boot
/dev/xvdb        49M   49M     0 100% /mnt
```

```
/dev/xvde           1.2G   38M  1.1G   4% /tmp/db
/dev/xvdf           1.5G   35M  1.4G   3% /tmp/temp_db
```

`/dev/xvde` is the datablock we're downloading, `/dev/xvdf` is the temporary storage space.

```
root@test-13974:~# umount /dev/xvde
root@test-13974:~# dd if=/dev/xvde of=/tmp/temp_db/db.db.img bs=4k
319488+0 records in
319488+0 records out
1308622848 bytes (1.3 GB) copied, 115.864 s, 11.3 MB/s
root@test-13974:~# logout
Connection to 172.18.6.19 closed.
[root@foo ~]# scp root@172.18.6.19:/tmp/temp_db/db.db.img ./db.db.img
db.db.img                                         100% 1248MB 25.5MB/s 00:49
[root@foo ~]# mount -o loop db.db.img db
[root@foo ~]# ls db
lost+found test.dat test.dat.shal test.txt
[root@foo ~]# cat db/test.txt
This is a test.
```

6.3 Network File System

Network File System (NFS) is a special shared storage solution available in BonFIRE. At the moment shared storage can only be created at the `be-ibbt` testbed, however. Unlike the other storage types, shared storages are accessible by multiple computes at the same time. If the storage was created outside the scope of an experiment, it can even be accessed by computes from different experiments.

For information about how to use NFS, please see the following links:

6.3.1 NFS OCCI How-To

Creating and using a SHARED storage resource

This type of storage resource can only be created on `be-ibbt`. The big difference between the Shared Storages and Datablock Storages, is that Shared storages can be connected to multiple compute resources at the same time and each compute works the same version (i.e. not a clone). These computes can be located on other testbeds than `be-ibbt`. Even more, the computes that connect to a Shared Storage even do not have to be in the same experiment. For this to work, the Shared Storage has to be created outside the scope of an experiment.

The implementation of Shared Storages is built around a dedicated NFS server (http://en.wikipedia.org/wiki/Network_File_System) located at the IBBT premises. This NFS server is connected to the BonFIRE WAN, which makes that it is accessible from every server connected to the BonFIRE WAN. Be aware of all the security considerations implied by using NFS storages: <http://www.linuxsecurity.com/content/view/117705/49/>.

When for instance the following storage resource is created (either outside or inside the context of an experiment), it results in the creation of a directory on the NFS server.

```
<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>shared</name>
    <description>test shared storage</description>
    <type>SHARED</type>
    <link href="/locations/be-ibbt" rel="location"/>
</storage>
```

The directory then can be mounted on any server connected to the BonFIRE WAN. For be-ibbt compute resources this mounting procedure is automated for the experimenter. By default on be-ibbt computes the shared storage is mounted to the directory /mount/bonfire-shared/{storage-id}. If you want to change this default target, you can specify it in the XML when creating the compute:

```
...
<disk id="...">
    <storage name="shared" href="/locations/be-ibbt/storages/6"/>
    <target>/mnt/shared</target>
    <type>DATABLOCK</type>
</disk>
...
```

On other testbed's computes the experimenter has to mount the SHARED storage manually. This can be easily done with the following command.

```
root@c2-2749:~# mount {mountpoint} {target-directory}
```

The target-directory is a directory that exists on the compute. If it does not exist make sure to create it first (mkdir).

The mountpoint parameter can be retrieved from a GET to the SHARED storage resource. It is located in the mountpoint XML element:

```
<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" name="shared" href="/locations/be-ibbt/storages/6">
    <description>test shared storage</description>
    <groups>gvseghbr</groups>
    <mountpoint>172.18.4.253:/mnt/bonfire-iscsi/gvseghbr/shared/6</mountpoint>
    <name>shared</name>
    <type>SHARED</type>
    <link rel="experiment" href="/experiments/12204" type="application/vnd.bonfire+xml"/>
</storage>
```

If you receive errors when executing the mount command, it is possible that the compute is not yet capable of mounting NFS filesystems. By executing the following commands, you can fix this issue:

```
root@c2-2749:~# apt-get update
root@c2-2749:~# apt-get install nfs-common portmap
```

Once the storage is mounted, it can be used as a regular directory. All files are synchronized in real-time.

6.4 Advanced Storage Features

BonFIRE provides the following advanced storage features.

6.4.1 Persisting data

A common use case is to install software or copy data on a persistent datablock storage, so that this storage can be reused at a later stage by another compute resource (this is what is done in the previous section). An additional requirement may be that any additional data written to the storage resource by subsequent compute resources must be automatically persisted.

This can be done on OpenNebula testbeds with the following method: create a **persistent** datablock storage **outside** of any experiment, then attach it to a new compute resource and write data to it. The new data will then be persisted when you shut down the compute resource.

Here is how you would do it with the cURL tool:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/storages \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>data</name>
  <persistent>YES</persistent>
  <size>1024</size>
  <type>DATABLOCK</type>
  <fstype>ext3</fstype>
</storage>
'

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/392">
  <id>392</id>
  <name>data</name>
  <type>DATABLOCK</type>
  <size>1024</size>
  <fstype>ext3</fstype>
  <public>NO</public>
  <persistent>YES</persistent>
</storage>
```

Then you would attach the newly created storage resource as a disk of a compute resource:

```
$ curl -kni https://api.bonfire-project.eu/experiments/1234/computes \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>c2</name>
  <location href="https://api.bonfire-project.eu/locations/fr-inria"/>
  <disk>
    <storage href="https://api.bonfire-project.eu/locations/fr-inria/storages/165"/>
    <type>OS</type>
  </disk>
  <disk>
    <storage href="https://api.bonfire-project.eu/locations/fr-inria/storages/392"/>
    <type>DATABLOCK</type>
  </disk>
  <nic>
    <network href="https://api.bonfire-project.eu/locations/fr-inria/networks/47"/>
  </nic>
  <instance_type>lite</instance_type>
</compute>

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/2749">
  <id>2749</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <name>c2</name>
  <instance_type>lite</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/165" name="BonFIRE Debian Squeeze v2"/>
    <type>DISK</type>
    <target>xvda</target>
```

```
</disk>
<disk id="1">
  <save_as href="/locations/fr-inria/storages/392"/>
  <storage href="/locations/fr-inria/storages/392" name="data"/>
  <type>DISK</type>
  <target>xvde</target>
</disk>
<nic>
  <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN"/>
  <ip>172.18.1.249</ip>
  <mac>02:00:ac:12:01:f9</mac>
</nic>
</compute>
```

Notice how a persistent storage resource is just a normal storage resource that is ‘saved-as’ under the same name.

Once the compute resource will be running, you’ll be able to mount the datablock storage. This can be achieved like this:

```
$ ssh 172.18.1.249
root@c2-2749:~# mkdir /tmp/data
root@c2-2749:~# mount -t ext3 /dev/xvde /tmp/data
```

You can then list the files in the /tmp/data directory, and list all currently mounted file systems:

```
root@c2-2749:~# ls -al /tmp/data
root@c2-2749:~# df -ah
```

Note: WARNING: as for ‘saved-as’ resource, you have to properly shut down the compute resource for the data to be persisted! If you just delete the compute resource, you’ll lose the data that has been written from this compute resource.

6.4.2 Extend the VM filesystem with datablock storage

Datablock concept

A datablock is an extra disk stored apart from the system disk image. You can create a datablock and use it as an extra drive inside the VM instance, then save it independently, and reuse it with another instance, etc ...

Note: Datablocks are **not** shared filesystem (like nfs), so you can’t share one at the same time between different instances.

One interesting property of datablock is the *persistent* flag. By default, a disk image is cloned each time you want to use it, with a new name / new image id. With the *persistent* flag set, the image is not cloned. When you use it and save it, the initial image is replaced with the new one. It means that if you use a *persistent* datablock on more than one instance at same time, only the last saved will remain.

Using datablocks to extend your root filesystem

The base images provided by BonFIRE are using LVM[1] to deploy the root filesystem. LVM makes it easy to extend the root filesystem with an extra drive.

Basically, you attach a datablock on your VM, which is seen as an extra drive inside the instance. A script will then add it to LVM, and extend the root partition and the filesystem without unmounting it.

This section explains how you can create an instance of any size with datablocks.

1. Create a datablock

BonFIRE >> Experiments >> Experiment Details >> Create Storage
Create Storage Resource

Create Storage Resource

Name ?

Group ?

Description ?

Filesystem ?

Size in MByte ?

Persistent ?

→ Continue **► Finish**

When you create a datablock, choose the desired size; note that your filesystem will have this size *added* to the default one. Important point is to **check the persistent flag** !

2. Create a compute resource

BonFIRE >> Experiments >> Experiment Details >> Create Compute
Create Compute Resource

Create Compute Resource

Name ?

Group ?

Instance Type ?

Cluster ? (Click here to reserve resources on this site)

Host ?

VMImage ?

Storage ?

Extend Root FS ?

Network ?

→ Continue **► Finish**

Create your resource as usual, and select your datablock on the appropriated field.

3. Add extend options: make sure that you check the Extend Root FS flag.

On the BonFIRE Portal, this generates the following xml:

```
<context>
  <usage>extend-root;zabbix-agent</usage>
</context>
```

The zabbix-agent usage is only needed if you have an aggregator on your experiment.

4. A few seconds after the boot, you will be able to login on your instance and see that the root filesystem has been extended.
5. One requirement is to **never** use the datablock on another instance (or the instance with another datablock), or you will lose data.

6.4.3 Storage attributes

A storage resource can have a number of attributes:

name

The storage name.

description

The storage description.

type

The storage type. Can be one of:

- OS: This type of storage contains a working operative system. Every VM template must define one DISK referring to an image of this type. Only one image of this type can be used in each compute description.
- FS: A file system is used to control how information is stored and retrieved. Basically, any fs understood by mkfs can be used. By default ext2,ext3,ext4,xfs can be supported in BonFIRE.
- DATABLOCK: A datablock storage is a storage for data, which can be accessed and modified from different compute resource. This type of storage can be created from previous existing data, or as an empty drive. You can use as many DATABLOCK storage resource in your compute resource descriptions as you need.
- SHARED: A shared storage is unique to be-ibbt. This type of storage can be accessed from any server in the BonFIRE WAN - this includes evidently also the compute resources connected to the BonFIRE WAN.

size

The storage size. In MB.

public

A flag that specifies if that storage resource can be seen and used by anyone. Possible values are YES or NO. Defaults to NO. A public storage resource cannot be persistent at the same time.

persistent

A flag that specifies if data changes in that storage resource will persist after the VM it is attached to is shut down. Setting **persistent** to YES means that the original storage resource will be modified if changes are made on the disk which uses that storage resource. Defaults to NO. Persistence is achieved by not cloning the data storage prior to use.

fstype

The type of file system used by the storage. This attribute is only required for storage resource of type DATABLOCK.

By default: ext2, ext3, ext4, xfs

6.4.4 Mount a datablock storage

Datablock concept

A datablock is an extra disk stored apart from the system disk image. You can create a datablock and use it as an extra drive inside the VM instance, then save it independently, and reuse it with another instance, etc.

Note: Datablocks are **not** shared filesystem (like nfs), so you can't share one at the same time between different instances.

One interesting property of datablock is the *persistent* flags. By default, a disk image is cloned each time you want to use it, with a new name / new image id. With the *persistent* flag set, the image is not cloned. When you use it and save it, the initial image is replaced with the new one. It means that if you use a *persistent* datablock on more than one instance at same time, only the last saved will remain.

Using datablocks as separate drives

You attach a datablock on your VM, which is seen as an extra drive inside the instance. This section explains how you can create an instance of any size with datablocks.

1. Create a datablock

BonFIRE >> Experiments >> Experiment Details >> Create Storage
Create Storage Resource

Name ?

Group ?

Description ?

Filesystem ?

Size in MByte ?

Persistent ?

→ Continue **Finish**

When you create a datablock, choose the desired size. You may or may not want to check the persistent flag.

2. Create a compute resource

BonFIRE >> Experiments >> Experiment Details >> Create Compute
Create Compute Resource

Name ?
vm

Group ?
demo

Instance Type ?
lite (cpus: 0.5, memory: 256MB)

Cluster ? (Click here to reserve resources on this site)
<default>

Host ?

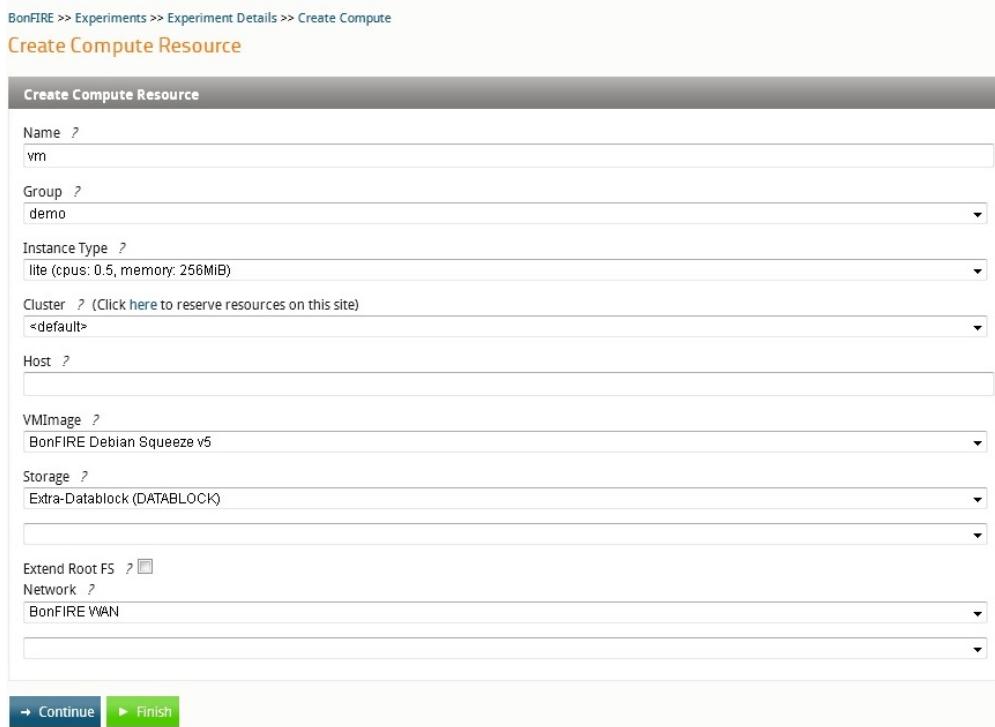
VMImage ?
BonFIRE Debian Squeeze v5

Storage ?
Extra-Datablock (DATABLOCK)

Extend Root FS ?

Network ?
BonFIRE WAN

→ Continue **► Finish**



Create your resource as usual, and select your datablock on the appropriated field. Make sure that you **do not check** the Extend Root FS flag.

3. A few seconds after the boot, you will be able to login on your instance. The datablock is not mounted yet.
4. Mount the new device to your desired location. Before you mount, you need to create the target directory.

4a. Create the location:

```
$ mkdir -p /my/mountpoint
```

4b. Mount the datablock:

```
$ mount /dev/xvdc /my/mountpoint
```

4c. Use the new directory:

```
$ ls /my/mountpoint
lost+found
```

NETWORKING

7.1 Overview of Networking in BonFIRE

This section aims to introduce the different networking options (basic resources, emulated network at Virtual Wall, AutoBAHN and FEDERICA, BonFIRE WAN and public internet), and to provide a high level overview of the inter-site networking capabilities, and limitations of BonFIRE.

7.1.1 Network Infrastructure

For details on the network resources offered within each site see the *Infrastructure* documentation.

By default (but see *Advanced Network Features*) the sites are interconnected solely via the public internet. Achieved network performance between sites is therefore highly variable depending on both the time of measurement and the particular pair of sites in question. Network performance is further affected by VPN issues - see *below*.

7.1.2 The BonFIRE WAN

Testbed providers are not able to provision a public IPv4 address for all virtual machines running on their infrastructure. Because of this, virtual machines running on different sites can only communicate between themselves if they all have a IPv4 address that is not public but that is reachable from the other sites. These private (RFC 1918) addresses that are routable outside local testbed boundaries are part of what we call the BonFIRE WAN. In theory, we potentially have three classes of addresses in the facility:

- Public IPv4 addresses
- BonFIRE addresses: they are private in the RFC 1918 sense of the term, but are uniquely assigned in BonFIRE
- Private addresses: they are unambiguous only inside a local testbed (or potentially even more narrowly).

In order to implement this, BonFIRE sites operate a VPN which tunnels BonFIRE WAN traffic between sites.

Table 7.1: List of BonFIRE WAN Addresses

Site Name	WAN Address
EPCC	172.18.3.0/24; New infrastructure BonFIRE WAN: 172.18.6.0/24
HLRS	
iMinds	172.18.4.0/24
HP	
PSNC	
INRIA	

7.1.3 Implications for Experiments

VM Networks

In order for a VM to be reachable by ssh, monitoring, etc. it **must** be reachable from the BonFIRE WAN. [Describe how this is achieved in R3 (routing via WAN or otherwise) and what the experimenter has to do. Include something about Cells when we know what's happening for R3.]

SSH Gateways

Because BonFIRE VMs do not, in general, have a public IP address, they cannot be reached directly from outside the BonFIRE WAN. An experimenter wishing to ssh to a VM must therefore do so via one of the [SSH Gateway Configuration](#).

Network Performance

Experimenters should be aware that, because of the VPN tunnelling, network performance between BonFIRE sites will be substantially lower than would otherwise be the case. Note that this is likely to be the case *even when one or both VMs are using a public IP*.

7.1.4 User Defined Networks

The be-ibbt, uk-hplabs and uk-epcc sites allow the creation of user defined networks. See [Basic network resources](#) for details.

7.1.5 VPN for users

OpenVPN service should enable the users to access the BonFIRE IPs at least on ssh and http(s) without limitation.

7.1.6 Advanced Network Features

- *Emulated Network at the Virtual Wall*
- *Controlled Bandwidth with AutoBAHN*
- *Controlled public networking with FEDERICA*

7.1.7 Future Developments

The following network features are not yet available, but are planned for future releases of BonFIRE.

IPv6

It is anticipated that some sites (EPCC, HLRS and PSNC) will be able to offer VMs with a native IPv6 connection. It is **not** intended to move any BonFIRE services to IPv6, therefore such VMs will still require an IPv4 connection to the BonFIRE WAN (for monitoring in particular).

7.1.8 Public IPs Available on Each Site

Different infrastructure sites can support a number of public IPs. The following table shows the number of public IPs available on each site:

Table 7.2: List of public IPs on each site

Site Name	Accessibility to Internet with WAN or private addresses	Access Restriction	Routing	Public Internet Address Pool Size	Public Internet Address Pool Size (BonFIRE Production Environment)
EPCC	Yes	Whitelist. Ports 80, 443 and 22 allowed by default. Others may be possible by request.	Direct (for VMs with public IPs).	6	6
HLRS	Yes	No restrictions	Directly	0 (not possible with current infrastructure)	0
iMinds	only via http_proxy	no restrictions (within BonFIRE WAN)	all traffic that CAN be routed via the WAN, IS currently routed over the WAN (according to Tinc config SVN)	not available	only few public addresses available for servers (where needed), only very limited availability
HP	on 80/443/22 tcp	Whitelist: only 80/443/22 tcp are opened. Inside the whitelist all ports are accessible	Directly through the Internet (for VMs with public IPs)	128 shared between both BonFIRE instances	128 shared between both BonFIRE instances
PSNC	Yes	No restrictions	routed through the gateway through the WAN	5	5
INRIA	on 80/443/22 tcp	only 80/443/22 tcp are opened		/24	/25 (131.254.204.128/25) (128 ips)

7.2 Basic network resources

Every compute resource has to be linked to at least one network resource, via a declaration of at least one `nic`. Unless you're doing experiments that are network-oriented, you'll probably want to use one of the public network resources, declared on each site. If you're operating on `fr-inria`, `uk-epcc`, `de-hlrs`, and `be-ibbt`, you'll be able to link your compute resources to the BonFIRE WAN network, which will give you an internal IP routed through the entire BonFIRE WAN (i.e. any VM can contact any other VM irrespective of the site). If you need public IPs, a few sites offer networks which will give your compute resources a public IP: `fr-inria`.

On `be-ibbt`, controllable network resources are offered in order to support network-oriented experiments. Additional network attributes can be set to control link impairments (latency, bandwidth, lossrate) and background traffic (throughput, protocol, packetsize). If you're operating on `uk-hplabs`, there is not "default" network, and you must always have created a network there first. See `uk-hplabs` specifics below.

It is also possible to create networks on `uk-epcc` using the same method. These local networks offer greater - although not complete - isolation from the effect of other experiments on the same site.

7.2.1 Network attributes

A network resource can have a number of attributes:

name

The network name.

description

The network description.

address

The address you wish for your network subnet. This is ignored by `uk-hplabs`.

size

The size of the network subnet. Possible values include A, B, C. These letters refer to the IANA-reserved private IPv4 network ranges. In the future address and size may be replaced with proper CIDR notation. Defaults to C. For `uk-hplabs`, the size is specified as a CIDR prefix in the range 20 - 30 inclusive.

public

A flag that specifies if that network resource can be seen and used by anyone. Possible values are YES or NO. Defaults to NO.

latency

The controlled latency (in ms) of the network. Only available on `be-ibbt`. Defaults to 0 (zero - no controlled latency).

bandwidth

The controlled bandwidth (in Mbps) of the network. Only available on `be-ibbt`. Defaults to 1000Mbps.

lossrate

The controller loss rate (value between 0 and 1) to introduce in the network. Only available on `be-ibbt`. Defaults to 0 (zero).

throughput

A traffic generator (TG) can be added to the network. The throughput of the TG is specified in #packets/s. Only available on `be-ibbt`. Optional, but if one of the TG parameters is specified, all three of them need to be there (`throughput`, `protocol` and `packetsize`).

protocol

The protocol of the TG. Should be either UDP or TCP. Only available on `be-ibbt`. Optional, but if one of the TG parameters is specified, all three of them need to be there (`throughput`, `protocol` and `packetsize`).

packetsize

The packet size used by the TG (in bytes). Only available on `be-ibbt`. Optional, but if one of the TG parameters is specified, all three of them need to be there (`throughput`, `protocol` and `packetsize`).

strategy

The queue strategy used. Only available on `be-ibbt`. Optional. Default value is `DropTail`, while other possible values are `RED` and `GRED`.

queue-in-bytes

Use bytes instead of packets in the following parameters. Only available on `be-ibbt`. Optional, but this parameter needs to be present when selected either the `RED` or `GRED` strategy. Value must be either 0 or 1.

limit

The queue size in packets. Only available on `be-ibbt`. Optional, but this parameter needs to be present when selected either the `RED` or `GRED` strategy.

maxthresh

The maximum threshold for the average queue size in packets. Only available on `be-ibbt`. Optional, but this parameter needs to be present when selected either the `RED` or `GRED` strategy.

thresh

The minimum threshold for the average queue size in packets. Only available on `be-ibbt`. Optional, but this parameter needs to be present when selected either the `RED` or `GRED` strategy.

linterm

As the average queue size varies between `thresh` and `maxthresh`, the packet dropping probability varies between 0 and $1/linterm$. Only available on `be-ibbt`. Optional, but this parameter needs to be present when selected either the `RED` or `GRED` strategy.

q_weight

The queue weight, used in the exponential-weighted moving average for calculating the average queue size. Only available on `be-ibbt`. Optional, but this parameter needs to be present when selected either the RED or GRED strategy.

7.2.2 UK-HPLabs (Cells) specifics

Experimenters are required to explicitly configure a network to which compute resources will be attached. When using the Portal or the API, the values for the address are ignored by the HP site; however the size must be specified by a CIDR prefix in the range 20-30 inclusive.

On the HP site, Internet access is provided as a publicly accessible subnet with a range of public IP addresses. There is no “by default” network, a VM must be explicitly attached to this network and/or any desired private networks at the point of creation. Private subnets by default are visible only to each experiment, and provide internal IP addresses on a subnet of the `10.0.0.0/8` network. These private subnets can be created by the “create network” function of the Portal or API.

Notwithstanding the above, a user may choose to share a network between several experiments or create it as a public resource. The use of the “public” attribute solely means that all BonFIRE users will be able to use the resource; this network will still be instantiated as a subnet allocated in the private `10.0.0.0/8` address space. This is not a recommended practice unless the user has specific reasons to use such a configuration.

7.2.3 How To Deploy Network Resources

OCCI How-To

Listing network resources

Listing network resources is easily achieved with a simple GET request:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/networks
```

```
HTTP/1.1 200 OK
Date: Wed, 12 Oct 2011 13:17:56 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Cache-Control: public,max-age=30
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Allow: GET,POST,OPTIONS,HEAD
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "23522379bbb92138c5b5e467c4a925fc"
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.386735
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <items>
    <network href="/locations/fr-inria/networks/20" name="Public Network"/>
    <network href="/locations/fr-inria/networks/47" name="BonFIRE WAN"/>
  </items>
</collection>
```

Creating network resources

Network resources MUST be created within an experiment. Assuming you have a running experiment with ID 3600, then you can create a network as follows:

```
$ curl -kni https://api.bonfire-project.eu/experiments/3600/networks \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Network name</name>
  <description>Network description</description>
  <address>192.168.0.0</address>
  <size>C</size>
  <bandwidth>1000</bandwidth>
  <latency>0</latency>
  <lossrate>0</lossrate>
  <throughput>200</throughput>
  <protocol>UDP</protocol>
  <packetsize>10</packetsize>
  <link rel="location"
        href="/locations/be-ibbt" />
</network>
'
```

```
HTTP/1.1 201 Created
Date: Wed, 12 Oct 2011 13:26:13 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Location: https://api.bonfire-project.eu/locations/be-ibbt/networks/27
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.212910
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/be-ibbt/networks/27">
  <address>192.168.0.0</address>
  <bandwidth>1000</bandwidth>
  <id>27</id>
  <latency>0</latency>
  <lossrate>0</lossrate>
  <name>Network name</name>
  <public>No</public>
  <size>C</size>
  <throughput>200</throughput>
  <protocol>UDP</protocol>
  <packetsize>10</packetsize>
</network>
```

See below for the complete list of attributes you can give when creating a network.

Note: OpenNebula testbeds are currently NOT configured to properly create private networks, therefore creation of network resources is disabled on the fr-inria, de-hlrs, and uk-epcc testbeds. This means for now you have to use the default networks.

Deleting network resources

As with every resource, just send a DELETE request to the resource URI:

```
$ curl -kni https://api.bonfire-project.eu/locations/be-ibbt/networks/27 -X DELETE
```

If everything went well, you'll receive a 204 No Content status code:

```
HTTP/1.1 204 No Content
Date: Wed, 12 Oct 2011 13:44:30 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/0.10.1)
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.173453
Vary: Authorization,Accept
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Note that in the specific case of be-ibbt, deleting individual resources is a no-op, i.e. be-ibbt resources are only deleted when you delete the experiment.

Portal How-To

Under construction

Experiment Descriptor How-To

JSON is one of BonFIRE experiment descriptors, and provides information about how to create network resource using JSON experiment descriptor

7.3 Emulated Network at the Virtual Wall

If the influence of the network is important in your experiment, the Virtual Wall testbed at IBBT offers you capabilities to control different network-oriented aspects. The functionality can be summarized into three main categories:

- **Network topologies:** the Virtual Wall allows you to setup any network topology you want and, by default, provides shortest path routing between any compute resources installed on the network nodes, with Gigabit connectivity.
- **Network impairments:** if you want to study the influence of network impairments on the performance of your system under test, you can specify link characteristics such as bandwidth, latency and loss rate on each link, both statically and dynamically.
- **Background traffic:** it is also possible to introduce TCP or UDP streams on the links to represent background traffic, with parameters to identify the protocol used, the packet size and the throughput.

7.3.1 Network Topologies

On the Virtual Wall infrastructure, a lossless switch with 1000 1Gbps ports interconnects all 100 nodes, each with 4 or 6 1Gbps ports, via VLANs. Each experiment on the Virtual Wall can use a subset of these nodes and connections to create a network topology. On each of these nodes default or customized images can be swapped in. The links have

a default bandwidth of 1Gbps. Details on how to start an experiment on the Virtual Wall can be found on [How to run experiments on Virtual Wall \(ibbt-vw\)](#).

7.3.2 Network Impairments

During experiment preparation, different values can be set to the parameters of each network link, to introduce impairments. The bandwidth of the link can be reduced from 100Mbps to any lower value. The latency can be increased from 0ms to several tens (e.g. access network link) or hundreds (e.g. Internet backbone link) of ms. Packet loss can be introduced as well, with typical values of several % for access network links or several tens of % for Internet backbone links.

These impairments can dynamically be updated during the experiment run. Details on network link configuration can be found in [Overview of Networking in BonFIRE](#).

You can also set a specific queue strategy. By default the DropTail strategy is used, but you can also use either RED or GRED. Both these strategies expect a few other parameters, which can be found in Basic network resources

7.3.3 Background Traffic

In order to determine the influence of background traffic on the tested system, transport layer streams can be created on each of the network links. These streams can use the UDP or TCP protocol, which can be chosen through the “protocol” parameter. The bandwidth of the stream can be controlled by setting the other two parameters for packet size (in bytes) and throughput (in packets per second). Details on background traffic configuration can also be found in [Overview of Networking in BonFIRE](#).

7.3.4 How to Emulate Network at the Virtual Wall

There are different possibilities to emulate network at Virtual Wall.

- via Portal : here is information about creating network resource via Portal for Virtual Wall
- via OCCI : here is information about creating network resource via OCCI for Virtual Wall
- via experiment descriptor : here is information about creating network resource via JSON for Virtual Wall

7.4 Controlled public networking with FEDERICA

7.4.1 IMPORTANT MESSAGE

At this moment, we do not have access to enough FEDERICA infrastructure resources to offer you the service described below. You will be notified when enough resources are available and you can start experimenting with FEDERICA.

Apologies for the inconveniences.

7.4.2 FEDERICA offering brief description

FEDERICA is an e-Infrastructure based on virtualization in both computers and network resources. FEDERICA interconnection allow BonFIRE experimenters to request an isolated network slice to connect VMs between EPCC and PSNC site, with the capability to:

- Design a network topology.
- Decide IP configuration of the interfaces.
- Configure of routing protocols (i.e: OSPF, BGP, static routes).

7.4.3 Network slice in BonFIRE: the router and network resource

To incorporate in BonFIRE the FEDERICA service some extensions are required. The <router> resource, a new type of OCCI resource has been added, and <network> resource has been extended. FEDERICA resources introduce a remarkable consideration when requesting for resources. For FEDERICA resources, experimenters have to explicitly indicate the physical resources on top of which they aim to create the logical partitions, which are also known as bounded requests. Moreover, the experimenter has to bear in mind the following:

- The experimenter provides sensible IP configuration within the network. There is not checking about the IP coherence in the submission process.
- The experimenter has knowledge of the routers' physical configuration since he has to provide the configuration when the logical router is requested.
- FEDERICA resources cannot be modified during the experimenter lifetime.

OCCI ‘router’ resource:

FEDERICA is composed by JUNIPER equipment that provides virtualization capabilities. Experimenters can request for logical routers, which are partition of the physical router into multiple logical devices that perform independently and replicate hardware functionality. Brief descriptions of the OCCI tags are:

- **<location>**: Reference the BonFIRE location where this router is going to be created. Initially, in routers case, it will be always /locations/federica/routers/
- **<host>**: Indicates the physical router where the logical router is requested.
- **<name>**: The name given to the logical router.
- **<description>**: A free field for experimenter convenience.
- **<interface>**: Interfaces represent the logical ports requested in the logical router. A logical router can have as many interfaces as required by the experimenter. Interfaces have the following parameters to define:
 - **<name>**: Identifies the logical interface within the logical router.
 - **<ip>, <netmask>**: IP configuration assigned to the logical interface.
 - **<physical_interface>**: Physical interface where is going to be created the logical interface.
 - **<config>**: The experimenter provides the desired router configuration as plain text.

The OCCI to request a logical router resource is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<router xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <location href="/locations/federica"/>
  <host></host>
  <name></name>
  <description></description>
  <interface>
    <name></name>
    <physical_interface></physical_interface>
    <ip></ip>
    <netmask></netmask>
  </interface>
```

```
<config></config>
</router>
```

OCCI ‘network‘ resource extension:

After adding FEDERICA resources, <network> also describes network topologies (links between logical routers) . To describe this, the <network_link> tag have been added to the <network> description. Brief descriptions of the OCCI tags are:

- **<location>**: Reference the BonFIRE location where this network is going to be created.
- **<description>**: A free field for experimenter convenience.
- **<network_link>**: Specifies the link between two endpoints. There are as many network_links as required by the topology.
- **<endpoint>**: One of the ends of a link connecting two resources.
- **<router href="">**: References a logical router created previously.
- **<router_interface>**: Name of the logical interface that comprise the link

The OCCI to request a network resource in FEDERICA is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <location href="/locations/federica"/>
  <description></description>
  <network_link>
    <endpoint>
      <router href="" />
      <router_interface></router_interface>
    </endpoint>
    <endpoint>
      <router href="" />
      <router_interface></router_interface>
    </endpoint>
  </network_link>
</network>
```

Creating FEDERICA experiment via OCCI

IMPORTANT MESSAGE

At this moment, we do not have access to enough FEDERICA infrastructure resources to offer you the service described below. You will be notified when enough resources are available and you can start experimenting with FEDERICA.

Apologies for the inconveniences.

Create a FEDERICA experiment

Creating a FEDERICA experiment via OCCI consists in these steps:

1. Create an experiment.
2. Create logical routers.
3. Create the FEDERICA network.

4. Put experiment to RUN.
5. Wait until the resources change to RUNNING state.
6. Create Network at EPCC and PSNC sites.
7. Create VMs at EPCC and PSNC sites.

Step 1. Create an experiment To create a new Experiment, send the following OCCI to BonFIRE Resource Manager:

Request:

```
POST /experiments HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>My FedericaExperiment</name>
    <description> First Experiment description</description>
    <walltime>7200</walltime>
</experiment>
```

Response:

```
HTTP/1.1 201 Created
Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/experiments/{{experiment_id}}">
    <name>My Experiment</name>
    <description>Experiment description</description>
    <walltime>7200</walltime>
    <computes>
        ...
    </computes>
    <storages>
        ...
    </storages>
    <networks>
        ...
    </networks>
    <link rel="parent" href="/" type="application/vnd.bonfire+xml" />
</experiment>
```

Step 2. Create a logical router To create a logical router, send the following OCCI to BonFIRE Resource Manager:

Request:

```
POST /experiments/{{experiment_id}}/routers HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<router xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
```

```

<location href = "locations/federica"/>
<host>dfn.erl.router1</host>
<name>RouterDFN</name>
<interface>
  <name>ifDFN</name>
  <physical_interface>ge-0/1/0</physical_interface>
  <ip>192.168.10.10</ip>
  <netmask>255.255.255.0</netmask>
</interface>
<config> edit routing-options autonomous-system 55;edit protocols bgp group G1 type external peer-a
</router>

```

Response:

HTTP/1.1 201 Created
 Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}/routers/{{router_id}}
 Content-Type: application/vnd.bonfire+xml

```

<?xml version="1.0" encoding="UTF-8"?>
<router xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/federica/router/9">
  <link href="/locations/federica" rel="location"/>
  <id>9</id>
  <status>PENDING</status>
  <host>dfn.erl.router1</host>
  <name>RouterDFN</name>
  <interface>
    <name>ifDFN</name>
    <physical_interface>ge-0/1/0</physical_interface>
    <ip>192.168.10.10</ip>
    <netmask>255.255.255.0</netmask>
  </interface>
  <config>edit routing-options autonomous-system 55;edit protocols bgp group G1 type external peer-a
</router>

```

Step 3. Create a network in FEDERICA To create a network in FEDERICA, send the following OCCI to BonFIRE Resource Manager:

Request:

POST /experiments/{{experiment_id}}/networks HTTP/1.1
 Host: api.bonfire-project.eu
 Accept: application/vnd.bonfire+xml
 Content-Type: application/vnd.bonfire+xml

```

<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Network name</name>
  <location href = "locations/federica"/>
  <network_link>
    <endpoint>
      <router href="locations/federica/routers/9"/>
      <router_interface>ifDFN</router_interface>
    </endpoint>
    <endpoint>
      <router href="locations/federica/routers/10"/>
      <router_interface>ifPSNC</router_interface>
    </endpoint>
  </network_link>

```

```
</network>'
```

Response:

```
HTTP/1.1 201 Created
Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}/networks/{{network_id}}
Content-Type: application/vnd.bonfire+xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/federica/network/3">
  <link href="/locations/federica" rel="location"/>
  <id>3</id>
  <name>Network name </name>
  <status>PENDING</status>
  <description/>
  <network_link>
    <endpoint>
      <router href="/locations/federica/router/9"/>
      <router_interface>ifDFN</router_interface>
    </endpoint>
    <endpoint>
      <router href="/locations/federica/router/10"/>
      <router_interface>ifPSNC</router_interface>
    </endpoint>
  </network_link>
</network>
```

Step 4. Put Experiment to RUN To put your experiment to RUN, send the following OCCI to BonFIRE Resource Manager:

```
PUT /experiments/{{experiment_id}} HTTP/1.1
Host: api.bonfire-project.eu
Content-Type: application/vnd.bonfire+xml
Accept: application/vnd.bonfire+xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">action=run</experiment>
```

You need to wait until all the resources change to RUNNING before continue with the steps below.

Step 5. Create network in EPCC To create a network in FEDERICA, send the following OCCI to BonFIRE Resource Manager: (watch out, do not forget to type the VLAN id obtained from the network in FEDERICA).

Request:

```
POST /experiments/{{experiment_id}}/networks HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>myNetEPCC</name>
  <address>192.168.0.0</address>
  <netmask>255.255.255.0</netmask>
  <vlan>2946</vlan>
```

```
<link href="/locations/uk-epcc" rel="location"/>
</network>
```

Response:

HTTP/1.1 201 Created
 Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}/networks/{{network_id}}
 Content-Type: application/vnd.bonfire+xml

```
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/uk-epcc/networks/141"
  <id>145</id>
  <name>myNetEPCC</name>
  <uname href="/locations/uk-epcc/users/31">jordi_jofre</uname>
  <groups>jordi_jofre</groups>
  <address>192.168.0.0</address>
  <netmask>255.255.255.0</netmask>
  <public>NO</public>
  <vlan>2946</vlan>
  <link rel="experiment" href="/experiments/2580" type="application/vnd.bonfire+xml"/>
</network>
```

Step 6. Create compute in EPCC To create a compute in EPCC, and attach it to the FEDERICA network, send the following OCCI to BonFIRE Resource Manager:

Request:

POST /experiments/{{experiment_id}}/computes HTTP/1.1
 Host: api.bonfire-project.eu
 Accept: application/vnd.bonfire+xml
 Content-Type: application/vnd.bonfire+xml

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>myVM</name>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/uk-epcc/storages/0"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
  <nics>
    <nic>
      <network href="/locations/uk-epcc/networks/0"/>
    </nic>
    <nic>
      <network href="/locations/uk-epcc/networks/141"/>
    </nic>
  </nics>
  <link href="/locations/uk-epcc" rel="location"/>
</compute>
```

Response:

HTTP/1.1 201 Created
 Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}/computes/{{network_id}}
 Content-Type: application/vnd.bonfire+xml

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/uk-epcc/computes/3328"
  <id>3328</id>
  <cpu>0.25</cpu>
  <memory>256</memory>
```

```
<name>myVM</name>
<instance_type href="/locations/uk-epcc/instance_type/lite">lite</instance_type>
<state>PENDING</state>
<disk id="0">
    <storage href="/locations/uk-epcc/storages/0" name="BonFIRE Debian Squeeze v3"/>
    <type>FILE</type>
    <target>sda</target>
</disk>
<nic>
    <network href="/locations/uk-epcc/networks/0" name="BonFIRE WAN"/>
    <ip>172.18.3.159</ip>
    <mac>02:00:ac:12:03:9f</mac>
</nic>
<nic>
    <network href="/locations/uk-epcc/networks/141" name="myEPCCnet-2580-54864305-ef08-49fc-94ac-ef31">
    <ip>192.168.0.1</ip>
    <mac>02:00:c0:a8:00:01</mac>
</nic>
<context></context>
</compute>
```

Creating a FEDERICA experiment through the Portal

IMPORTANT MESSAGE

At this moment, we do not have access to enough FEDERICA infrastructure resources to offer you the service described below. You will be notified when enough resources are available and you can start experimenting with FEDERICA.

Apologies for the inconveniences.

Create a FEDERICA experiment

To create a FEDERICA experiment, you need to follow these steps in order:

1. Create an experiment.
2. Create logical routers.
3. Create the FEDERICA network.
4. Put experiment to RUN.
5. Wait until the resources change to RUNNING state.
6. Create Network at EPCC and PSNC sites.
7. Create VMs at EPCC and PSNC sites.

Step 1. Create an experiment Connect and log to the BonFIRE portal. Go to the Experiment page and click on “Add Experiment”. In the “Create Experiment” page (*Adding a private subnet on Cells*), specify the desired parameters and click on the “Finish” button.

The screenshot shows the 'Create Experiment' page. It includes fields for 'Name' (set to 'Federica Experiment'), 'Description' (set to 'M-M Federica Experiment'), and several configuration options. These options include checkboxes for 'Enable AWS Support', 'Add a Monitoring Aggregate at this Site', 'Add a Compute Aggregate at this Site', 'Add Storage Aggregates', 'Let the Aggregate monitor itself', and 'Enable Capable Aggregate'. At the bottom, there are 'Continue' and 'Finish' buttons.

Figure 7.1: Experiment creation

Step 2. Create Logical Routers From the “Experiment Details” page (see *Choosing the network size*) select the “Site Interconnection” tab (see *Creating a new Compute Resource*) and click on the “add” button in the first table to add a router resource.

The screenshot shows the 'Experiment Details' page with the 'Resources' tab selected. It features three tables: 'Compute Resources', 'Storage Resources', and 'Network Resources'. Each table has columns for Name, Id, Type, Description, Address, Bandwidth, Latency, Loss Rate, and various status indicators. Each table also has an 'add' button. Above the tables, there are tabs for 'Resources', 'Site Interconnection' (which is highlighted in red), 'Elasticity', and 'Monitoring'.

Figure 7.2: Experiment details - resources

In the “Create Logical Router” form, specify the parameters of your logical router and click on the “Finish” button to create it.

Step 3. Create Network From the “Experiment Details” page (see *Choosing the network size*) select the “Site Interconnection” tab (see *Creating a new Compute Resource*) and click on the “add” button in the second table to add a network resource.

In the “Create Network” form, specify the parameters of your logical router and click on the “Finish” button to create it.

Step 4. Put Experiment to RUN In order to successfully finish the creation of the experiment, after all the resources have been submitted, the experimenter need to push the “GO” button.



Figure 7.3: Experiment details - create logical router



Figure 7.4: Experiment details - logical router form

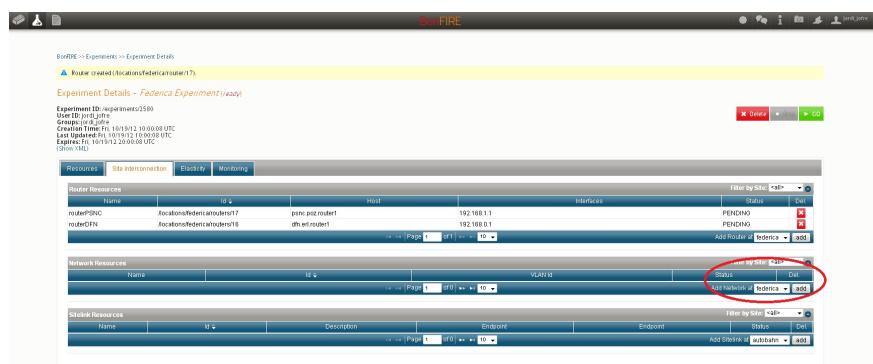


Figure 7.5: Experiment details - create network in FEDERICA

BonFIRE > Experiments > Experiment Details > Create Federica network
Create Federica network Resource

Name: mfFedericaNetworkSlice
Please specify a number of network links between Federica routers that will be part of this network:
Network link
First Endpoint: Router routerDFN -> Interface: IDFN (192.168.0.1)
Second Endpoint: Router routerPSNC -> Interface: IPSONC (192.168.1.1)

Add Network Link

Continue Finish

Figure 7.6: Experiment details - network form

BonFIRE > Experiments > Experiment Details
Experiment Details - Federica Experiment ready

Experiment ID: experiments/250
User ID: jordi.jofre
Group ID: jofre
Creation Time: Fri, 10/09/12 10:00:00 UTC
Last Update: Fri, 10/09/12 10:00:00 UTC
Expires: Fri, 10/09/12 20:00:00 UTC
Owner: jofre

Resources Site Interconnection Elasticity Monitoring

Router Resources

Name	ID	Host	Interfaces	Status	Det.
routerPSNC	locations/federicanetworks/7	psnc psn router	192.168.1.1	PENDING	
routerDFN	locations/federicanetworks/8	dfn dfn router	192.168.0.1	PENDING	

Network Resources

Name	ID	VLAN	Status	Det.
mfFedericaNetworkSlice	locations/federicanetworks/9	2946	PENDING	

Sitemesh Resources

Name	ID	Description	Endpoint	Endpoint	Status	Det.

Filter by State: All ▾

Run Stop

Figure 7.7: Experiment details - put Experiment to RUN

Step 5. Wait until the resources change its status to RUNNING When the resources change to RUNNING, the network slice is ready and the experiment can start using it.

BonFIRE > Experiments > Experiment Details
Experiment Details - Federica Experiment running

Experiment ID: experiments/250
User ID: jordi.jofre
Group ID: jofre
Creation Time: Fri, 10/09/12 10:00:00 UTC
Last Update: Fri, 10/09/12 10:00:00 UTC
Expires: Fri, 10/09/12 20:00:00 UTC
Owner: jofre

Resources Site Interconnection Elasticity Monitoring

Router Resources

Name	ID	Host	Interfaces	Status	Det.
routerPSNC	locations/federicanetworks/7	psnc psn router	192.168.1.1	RUNNING	
routerDFN	locations/federicanetworks/8	dfn dfn router	192.168.0.1	RUNNING	

Network Resources

Name	ID	VLAN	Status	Det.
mfFedericaNetworkSlice	locations/federicanetworks/9	2946	RUNNING	

Sitemesh Resources

Name	ID	Description	Endpoint	Endpoint	Status	Det.

Filter by State: All ▾

Run Stop

Figure 7.8: Experiment details - Experiment in RUNNING status.

Step 6. Create networks at EPCC and PSNC sites We need to wait until the Federica slice is ready before creating VMs at EPCC and PSNC sites. This is because we need the Federica network details to be configured in the VMs (namely the VLAN of the FEDERICA slice) to make sure the VMs traffic goes through the user-configured slice.

To do that, we need to create a network in EPCC and PSNC sites, introducing in the form the VLAN id obtained from the FEDERICA network. This is mandatory and important to configure correctly the VM.

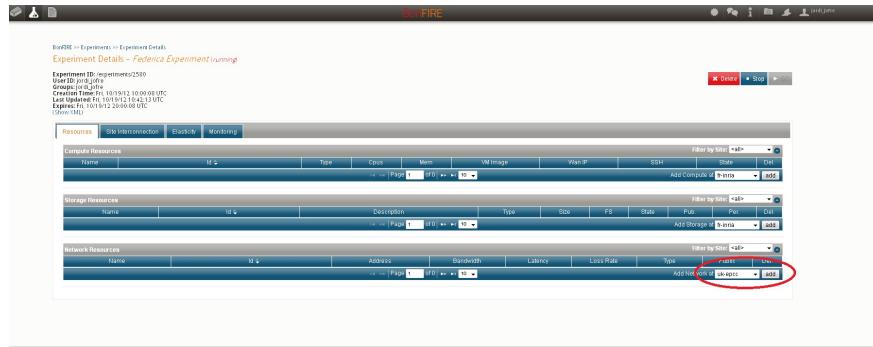


Figure 7.9: Experiment details - Create network in EPCC



Figure 7.10: Experiment details - network form in EPCC

Do the same for PSNC site.

Step 7. Create VMs at EPCC and PSNC sites After the network in EPCC and PSNC are ready. Is the moment to create VMs in EPCC and PSNC. To create a compute you need to do the following:



Figure 7.11: Experiment details - Create compute

In the “Create compute” form, specify the parameters of your VM. Do not forget to attach to your VM the FedericaNetwork you have created. Otherwise your traffic will not go through FEDERICA.

Do the same for PSNC site.

Figure 7.12: Experiment details - compute form

Creating a FEDERICA experiment through Experiment Descriptor

A FEDERICA experiment can be created using Experiment Descriptor. Please have a look at JSON for FEDERICA

7.5 Controlled Bandwidth with AutoBAHN

BonFIRE, starting from release 3.1, provides the experimenters with the possibility to request QoS-enabled network connectivity services with a guaranteed bandwidth (Bandwidth on Demand – BoD – services) to interconnect BonFIRE sites. Instead of relying on the best effort Internet connectivity, the cloud resources (storage, VMs) located in different BonFIRE sites can be interconnected through a dedicated network service with the bandwidth requested by the experimenter.

This feature is mainly targeted to the network-aware experiments that may need guaranteed bandwidth to achieve more than best-effort connections between computational resources. In fact, experimenters can obtain guarantees for the minimum bandwidth reserved to their experiment and be sure it will not be impacted by other coexisting experiments; similarly, they can obtain a maximum bandwidth guarantee, thus being capable of emulating the behaviour of a given link capacity or link congestions.

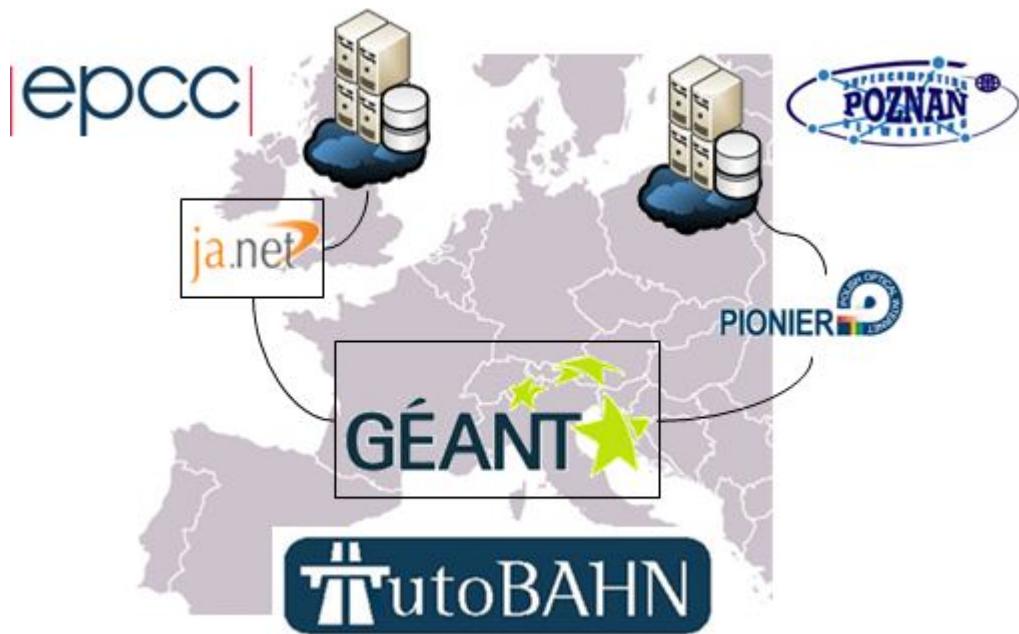
In BonFIRE the inter-site BoD services are provided by a third party network provider connecting the BonFIRE sites. In particular BonFIRE adopts the GÉANT Bandwidth on Demand (BoD) system (AutoBAHN BoD version 2.1.1), since GÉANT and the National R&E Networks (NRENs) are the primary network providers for most of the BonFIRE facility owners. In release 3.1, the offer for BoD services is limited to the interconnection between the EPCC and PSNC sites.

7.5.1 BoD in BonFIRE: the site_link resource

In BonFIRE a BoD service is considered as a new type of OCCI resource, like a storage or a network resource. The BoD service is represented through a site_link resource and can be manipulated following the common CRUD mechanisms adopted in BonFIRE to create, remove and query resources via OCCI interfaces. The only limitation is that the modification of an already existing site_link resource is not allowed.

The following list of parameters is included in a site_link resource:

- <location>, <name>, <id>, <description> - self explaining names of commonly used parameters in BonFIRE OCCI elements



- <endpoint> - this attribute references to a BonFIRE site. For each BonFIRE site connected to GÉANT through a NREN deploying AutoBAHN for the automatic provisioning of BoD services (i.e. currently only PSNC and EPCC), a specific client port has been defined and configured in order to act as end-point for the BoD services in the associated site.
- <bandwidth> - value of bandwidth (in Mbps) for symmetric network connections from “start” to “end” end-point
- <status> - identifies the current status of the BoD service. The following values are possible: PENDING, RUNNING, FAILED, CANCELLED, SCHEDULED

OCCI site_link resource:

```
<site_link href ="/locations/autobahn/site_links/{<ID>}">

    <name></name>
    <id>xx</id>
    <description></description>

    <endpoint>
        <location href="/locations/{{siteA_id}}">
            <vlan>xxx1</vlan>
        </location>
    </endpoint>

    <endpoint>
        <location href="/locations/{{siteB_id}}">
            <vlan>xxx2</vlan>
        </location>
    </endpoint>

    <bandwidth>yy</bandwidth>
    <status>zz</status>
    <link rel="location" href="/locations/autobahn"/>

</site_link>
```

The `site_link` resource is the first type of resource to be created in an experiment. Then, for each site, the user must create a new network resource using the VLAN Identifiers reported in the `site_link` description. Finally, he/she can create the compute and storage resources, connecting them to these networks. As result, the resources will interact using the QoS-enabled network service established between the sites.

The following tutorial provides a brief overview of the main actions that can be performed over `site_link` resources following two main perspectives. The section 1 explains how to create an experiment with resources interconnected through a QoS-enabled network service using the BonFIRE Portal and is mainly targeted to the experimenters. The section 2 provides details about the OCCI messages to be exchanged with the enactor in order to create/remove/query the `site_link` resources and addresses the BonFIRE developers.

7.5.2 Site link performance optimization and troubleshooting

Due to applied bandwidth policing over site links and their long distant layer 2 connectivity it is required to optimize networking on the VMs interfaces attached to the site links to achieve the best possible performance. There are two aspects which have to be covered:

1. Switching off any offload parameters for ethernet devices (interfaces) connected to the site links
2. Traffic control for shaping outgoing traffic (mainly for improving TCP performance).

Understanding and proper application of these steps will reduce potential issues related to the possible degradation of performance over users' site links.

Segment Offload parameters

By default BonFIRE VMs have enabled some of large segment offload (LSO) parameters that are responsible for increasing outbound high bandwidth network traffic while CPU overhead is reduced. Through this technique it is possible to send jumbo frames up to 64KB over the network. However using very large packets causes lots of packet segmentation over site links and when we combine this with the policing over these links there will be poor network performance due to frequent packet drops by the network devices along the path. The solution is to turn off any LSO features for the interfaces attached to site links using following command:

```
# ethtool -K ethX tso off ufo off gso off
```

where

- `ethX` is an interface assigned to the particular site link.

It might be required to install this tool with `apt-get install ethtool` command.

Traffic control using tc command tool

Traffic control is used to provide shaping of any outgoing network traffic for the particular interfaces which eliminates influence of the site link policing. It is very important to notice that `tc` command does not work properly with any LSO turned on so please make sure that offload features are disabled first.

```
# tc qdisc add dev ethX root tbf rate YYYmbit latency 70ms burst 15k
```

where:

- `ethX` is an interface attached to the site link
- `latency 70ms` - as the site link RTT between EPCC and PSNC varies between 54-65 ms depending on the site link route
- `burst 15k` - it is $10 * \text{MTU size} = 15\text{KB}$ - calculated with one of standard methods for setting burst size limits
- `YYYmbit` is the site link bandwidth

It is important to know that `tc` command uses rate in *mbit* ($1\text{mbit} = 1024 * 1024 \text{ bits}$) while site link bandwidth is defined in *Mbps* ($1\text{Mbps} = 1000 * 1000 \text{ bits}$). Bandwidth rate for `tc` command has to be recalculated according to the following equation:

$$\text{tc_rate} = \frac{\text{site_link_bandwidth_in_Mbps} * 1000 * 1000}{1024 * 1024}$$

Example for 10 Mbps site link assigned to `eth0`:

$$\text{tc_rate} = (10 * 1000 * 1000) / (1024 * 1024) = \sim 9.53$$

```
tc qdisc add dev eth0 root tbm rate 9.53mbit latency 70ms burst 15k
```

7.5.3 How to Manage Experiments with BoD

Managing experiments with BoD services through the BonFIRE Portal

The creation of an experiments with controlled network services can be easily performed using the BonFIRE portal and can be summarized in 4 main steps:

1. Creation of a new experiment.
2. Creation of the BoD service between the BonFIRE sites.
3. Creation of the network resource within the BonFIRE sites.
4. Creation of the compute/storage resource within the BonFIRE sites.

These steps are detailed in the following paragraphs.

Step 1. Create an experiment

Connect and log to the BonFIRE portal. Go to the Experiment page and click on “Add Experiment”. In the “Create Experiment” page (*Adding a private subnet on Cells*), specify the desired parameters and click on the “Finish” button.

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Name ?
BoD-experiment

Description ?
Test BoD

Waittime (Lifetime in Minutes) ?
120

Group ?
minus

Enable AWS Support ?

Add a Monitoring Aggregator at this Site ?

don't create an aggregator

Add Infrastructure Metrics ?

Let the Aggregator monitor itself ?

Elasticity Capable Aggregator ?

→ Continue ▶ Finish

Figure 7.13: Experiment creation

Step 2. Add the interconnection between the BonFIRE sites

From the “Experiment Details” page (see [Choosing the network size](#)) select the “Site Interconnection” tab (see [Creating a new Compute Resource](#)) and click on the “add” button in the last table to add a site_link resource.

Figure 7.14: Experiment details - resources

Figure 7.15: Experiment details - interconnections

In the “Create site_link” page (see [New Compute Resource Wizard](#)) specify the parameters of the BoD service (name, description, requested bandwidth in Mbps and the two BonFIRE sites to be interconnected) and click on the “Finish” button to create the site_link.

Once the site_link resource has been created, it is shown in the “Site Interconnection” tab of the “Experiment Details” page (see the following figure).

Click on the resource name to visualize the site_link details (see the following figure). The VLAN IDs will be shown as soon as made available, after the computation of the service path.

BonFIRE >> Experiments >> Experiment Details >> Create Sitelink
Create Sitelink Resource

Create Sitelink Resource

Name ? sitelink1

Group ? mikus

Description ? my link

Bandwidth in Mbps ? 10

First Endpoint ? pl-psnc

Second Endpoint ? uk-epcc

→ Continue ▶ Finish

Figure 7.16: site_link creation

Experiment ID: /experiments/2835
User ID: mikus
Groups: mikus
Created Time: Fri, 01/18/13 15:15:51 UTC
Last Updated: Fri, 01/18/13 15:15:51 UTC
Expires: Fri, 01/18/13 17:15:51 UTC
(Show XML)

Resources **Site Interconnection** **Elasticity** **Monitoring**

Router Resources					
Name	Id	Host	Interfaces	Status	Del
Page 1 of 0					
Add Router at federica [add]					

Network Resources					
Name	Id	VLAN Id	Interfaces	Status	Del
Page 1 of 0					
Add Network at federica [add]					

Sitelink Resources					
Name	Id	Description	Endpoint	Endpoint	Status
siteLink1	/locations/autobahn/site	my link	VLAN 2004 @ pl-psnc	VLAN 2004 @ uk-epcc	RUNNING
Page 1 of 1					
Add Sitelink at autobahn [add]					

Figure 7.17: Experiment details - site interconnections (1)

BonFIRE >> Experiments >> Experiment Details >> Resource Details
Details for Sitelink Resource /locations/autobahn/site_links/TestGeantBoD@1358522791525

ID: /locations/autobahn/site_links/TestGeantBoD@1358522791525
Name: sitelink1
Groups: mikus
Description: my link
Bandwidth in Mbps: 10
Status: RUNNING

Endpoint

Location: pl-psnc VLAN Id: 2004

Location: uk-epcc VLAN Id: 2004

(Show XML)

Figure 7.18: site_link details

Step 3. Add the network resources within the BonFIRE sites

Once the BoD service has been established, you need to create a network resource within each BonFIRE site to be interconnected. These networks must have the same VLAN IDs selected for the `site_link` resource and reported in the `site_link` description - 2008 for both the EPCC and PSNC sites in our example.

In order to add a network resource you can follow the common BonFIRE procedure: from the “Resources” tab in the “Experiment Details” page (see the following figure) in the “Network Resources” table select the BonFIRE site and click on “add”.

Name	Id	Description	Type	Size	FS	State	Pub.	Per.	Del.
pi-psnc			be-psnc						Add

Figure 7.19: Resources summary

In the “Create Network Resource” page (see the following figure), specifies all the requested parameters and click on Finish. Remember to set the Virtual LAN identifier using the value reported in the `site_link` description.

Name	Man-on-psnc
Group	mikus
Network Address	192.168.0.0
Subnetmask	255.255.255.0
Virtual LAN Id	2004 (SiteLink1 to uk-epcc)

[Continue](#) [Finish](#)

Figure 7.20: Network resource creation

This procedure must be repeated for both the BonFIRE sites. It is important to define different network ranges on both sites. This will prevent from assigning the same ip addresses on both sites, so to provide routing between sites additional contextualization has to be provided for VMs (see next step).

Step 4. Add the compute/storage resources within the BonFIRE sites

As final step, you can create the compute or storage resources to be instantiated on the BonFIRE sites, following the usual BonFIRE procedures. In the resource description you need to specify the network where the resource itself must be attached, selecting one of the networks created before.

BonFIRE >> Experiments >> Experiment Details >> Create Compute
Create Compute Resource

Create Compute Resource

Name ?
vm-on-psnc

Group ?
mikus

Instance Type ?
lite (cpus: 0.25, memory: 256MB)

Cluster ? (This site does not support advance reservations)
<default>

Host ?

VMImage ?
BonFIRE Debian Squeeze for KVM v1

Storage ?

Extend Root FS ?

Network ?
BonFIRE WAN

vlan-on-psnc

Figure 7.21: Compute resource creation

To provide routing on attached network resource one has to edit compute resource and add context section to OCCI descriptor.

BonFIRE >> Experiments >> Experiment Details >> Create Compute
Create Compute Resource

Review OCCI Request for Compute Creation

```
</nic>
<nics>
<nic href="/locations/pl-psnc/networks/27"/>
</nics>
</nic>
<network href="/locations/pl-psnc/networks/27"/>
</networks>
<context>
<ipRoute>192.168.1.0/24 dev eth1</ipRoute>
</context>
<link href="/locations/pl-psnc" rel="location"/>
</compute>
```

Ignore XML Validation Errors

Figure 7.22: Compute resource context

In the above example `ipRoute` will cause to run `ip route add 192.168.1.0/24 dev eth1` command during VM contextualization process. In this case `eth1` is the interface created for attached network resource.

This step has to be performed for other compute resources on both sites.

Managing BoD services via OCCI

The global workflow to create an experiment with a `site_link` between two sites (only EPCC and PSNC in R3.1) is shown in [Procedures to create an experiment with a BoD service](#). The specific OCCI messages are described in section [OCCI messages for site_link creation/query/deletion](#).

Once the experimenter has declared the experiment (step 1), he/she can start to specify and request the different resources associated to this experiment.

The first resource to be requested (step2) is the `site_link` between the two BonFIRE sites. At the Enactor the `site_link` specification is translated into a BoD service request sent to AutoBAHN through the UAP interface in a SOAP message. AutoBAHN starts the procedures for the creation of the end-to-end network path between the two BonFIRE sites and provides the response. At this stage, the enactor reply returns the service ID that must be used for the entire lifecycle of the BoD service to univocally identify the OCCI `site_link` resource.

It should be noted that in this initial phase, the BoD service is not yet fully established and some parameters (e.g. the VLAN IDs selected for the traffic flows) could be not yet available. Consequently, it is required a further step (step 3) to request the full set of information related to the `site_link` resource through a GET request. This request is translated in a SOAP message addressed to AutoBAHN and the returned description is provided back to the user that becomes aware of the VLAN IDs. These VLAN IDs must be used to tag the traffic flow on both the BonFIRE sites (PSNC and EPCC in this example).

At this point, the experimenter has all the information required to create the remaining set of resources on the source (step 4) and destination (step 5) BonFIRE sites. In particular, the experimenter requests first the network resource specifying the VLAN ID returned in the `site_link` description and then the compute resource to be attached to network just created.

The deletion phase, not shown in the picture, does not require a specific order (i.e. the experimenter can decide to remove the various resources in a different order).

OCCI messages for site_link creation/query/deletion

This section provides some examples of curl commands to create, query and delete a `site_link` resource sending OCCI requests to the BonFIRE Resource Manager. The related replies are also reported for reference.

1. Request to create a new `site_link` (step 2) - In the request both the end-points and a not-zero bandwidth value must be declared. The end-points must be different (currently only uk-epcc and pl-psnc are supported). The reply includes the resource id, and some parameters (e.g. VLAN IDs) are still missing, since the network service is still under signalling and not yet fully established (status=PENDING).

Request:

```
POST
https://api.bonfire-project.eu/locations/autobahn/site_links
```

```
<site_link xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>link1</name>
    <description>sample_link</description>
    <endpoint>
        <location href="/locations/uk-epcc"/>
    </endpoint>
    <endpoint>
        <location href="/locations/pl-psnc"/>
    </endpoint>
    <bandwidth>1000000</bandwidth>
```

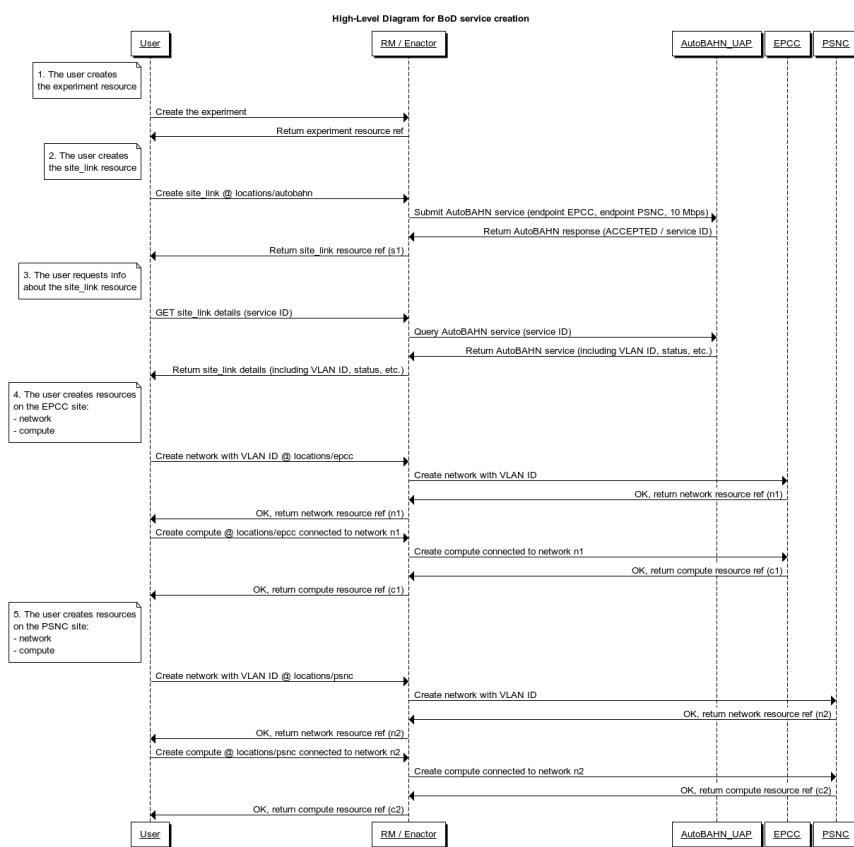


Figure 7.23: Procedures to create an experiment with a BoD service

```

<link rel="location" href="/locations/autobahn"/>
</site_link>
```

Response:

```

< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/vnd.bonfire+xml
< Transfer-Encoding: chunked
< Date: Thu, 02 Aug 2012 13:28:03 GMT
<
<?xml version="1.0" encoding="UTF-8"?>
<site_link href="/locations/autobahn/site_links/TestGeantBoD@1343914053450">
```

2. Query of the site_link to retrieve VLAN IDs (step 3) - the site_link can be queried using the resource id returned in the previous response. The response provides now the full description of the site_link resource, including the VLANs, and the network service is now completely established (status=RUNNING).

Request:

```
GET
https://api.bonfire-project.eu/locations/autobahn/site_links/TestGeantBoD@1343914053450
```

Response:

```

< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/vnd.bonfire+xml
< Transfer-Encoding: chunked
< Date: Thu, 02 Aug 2012 13:28:19 GMT
<

<?xml version="1.0" encoding="UTF-8"?>
<site_link href="/locations/autobahn/site_links/TestGeantBoD@1343914053450">
    <id>TestGeantBoD@1343914053450</id>
    <name>link1</name>
    <description>sample_link</description>
    <endpoint>
        <location href="/locations/uk-epcc">
            <vlan>2004</vlan>
        </endpoint>
        <endpoint>
            <location href="/locations/pl-psnc">
                <vlan>2004</vlan>
            </endpoint>
            <bandwidth>1000000</bandwidth>
            <status>RUNNING</status>
            <link rel="location" href="/locations/autobahn"/>
    </site_link>
```

3. Deletion of the site_link.

Request:

```
DELETE
https://api.bonfire-project.eu/locations/autobahn/site_links/TestGeantBoD@1343914053450
```

Response:

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/vnd.bonfire+xml
< Transfer-Encoding: chunked
< Date: Thu, 02 Aug 2012 13:28:30 GMT
```

4. Query of the deleted site_link - once the site_link has been deleted its status is CANCELLED.

Request:

```
GET
https://api.bonfire-project.eu/locations/autobahn/site_links/TestGeantBoD@1343914053450
```

Response:

```
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/vnd.bonfire+xml
< Transfer-Encoding: chunked
< Date: Thu, 02 Aug 2012 13:28:33 GMT
```

```
<?xml version="1.0" encoding="UTF-8"?>
<site_link href="/locations/autobahn/site_links/TestGeantBoD@1343914053450">
    <id>TestGeantBoD@1343914053450</id>
    <name>link1</name>
    <description>sample_link</description>
    <endpoint>
        <location href="/locations/uk-epcc">
            <vlan>2004</vlan>
        </location>
    </endpoint>
    <endpoint>
        <location href="/locations/pl-psnc">
            <vlan>2004</vlan>
        </location>
    </endpoint>
    <bandwidth>1000000</bandwidth>
    <status>CANCELLED</status>
    <link rel="location" href="/locations/autobahn"/>
</site_link>
```

Managing BoD services via Experiment Descriptor

Managing AutoBAHN BoD services can be done using an Experiment Descriptor. The following code (also available to download) creates a site-link between EPCC and PSNC; an associated network on each site; and a VM on each site, which can route traffic through AutoBAHN:

```
{
    "name": "AutoBAHN simple",
    "description": "Creates a site-link, site networks and one VM on each site",
    "duration": 60,
    "resources": [
        {
            "site_link": {
                "name": "link",
                "description": "link-epcc-psnc",
                "locations": [
                    "autobahn"
                ],
                "endpoints": [
```

```

        "uk-epcc",
        "pl-psnc"
    ],
    "bandwidth": 10
}
},
{
    "network": {
        "name": "vlan-epcc",
        "locations": [
            "uk-epcc"
        ],
        "address": "192.168.1.0",
        "netmask": "255.255.255.0",
        "vlan": {
            "vlanFromSiteLink": "link"
        }
    }
},
{
    "network": {
        "name": "vlan-psnc",
        "locations": [
            "pl-psnc"
        ],
        "address": "192.168.0.0",
        "netmask": "255.255.255.0",
        "vlan": {
            "vlanFromSiteLink": "link"
        }
    }
},
{
    "compute": {
        "name": "vm-psnc",
        "locations": [
            "pl-psnc"
        ],
        "instanceType": "lite",
        "min": 1,
        "resources": [
            {
                "storage": "@BonFIRE Debian Squeeze v6"
            },
            {
                "network": "@BonFIRE WAN"
            },
            {
                "network": "vlan-psnc"
            }
        ],
        "contexts": [
            {
                "iproute": "192.168.1.0/24 dev eth1"
            }
        ]
    }
},

```

```
{  
    "compute": {  
        "name": "vm-epcc",  
        "locations": [  
            "uk-epcc"  
        ],  
        "instanceType": "lite",  
        "min": 1,  
        "resources": [  
            {  
                "storage": "@BonFIRE Debian Squeeze v6"  
            },  
            {  
                "network": "@BonFIRE WAN"  
            },  
            {  
                "network": "vlan-epcc"  
            }  
        ],  
        "contexts": [  
            {  
                "iproute": "192.168.0.0/24 dev eth1"  
            }  
        ]  
    }  
}
```

Performance optimization will still be necessary.

Please visit JSON for AutoBAHN for further discussion.

MONITORING

8.1 Overview of Monitoring Options in BonFIRE

The monitoring solution offered by BonFIRE is based on the open source monitoring software [Zabbix](#), with the exception of timestamping. The software comprises two major software components: Zabbix server and Zabbix agent. The server is referred to as ‘aggregator’ in BonFIRE, which would be deployed in a separate resource, whilst the agent resides in the deployed resource image (one per resource image). The Zabbix aggregator collects monitoring information reported by the Zabbix agents in a database. The database can be stored either inside the aggregator image or stored into an external storage resource that is attached to the aggregator as an additional, external disk.

BonFIRE monitoring System enables experimenters to store monitoring data in a flexible way. The data storage size can be offered on-demand, and thus, the experimenter can get the desired storage size. Furthermore, depending on the experiment setup, monitoring data is not only available while experimenting but can even be maintained persistent after the deletion or the expiration of the experiment.

8.1.1 VM Monitoring

Many metrics are being monitored in VM level. Over than 100 metrics are being monitored by default and can be activated or deactivated on request. Some of these, but not limited to, per VM are: total memory, used memory, free memory, total swap memory, used swap memory, free swap memory, total storage, used storage, free storage, CPU load, CPU utilization (%), CPU count, network metrics (e.g. incoming and outgoing traffic in interfaces, OS-related metrics, processes-related metrics (e.g. number of running processes), services-related metrics (e.g. FTP-/Email-/SSH-server is running), etc. On the page of how to set up monitoring you can find more information and guidelines.

8.1.2 Application Monitoring

Application monitoring addresses metrics which provide information about the state of the running application, its performance and other application specific information. These metrics are not provided by default by Zabbix, the experiments needs to explicitly configure them at the agent and the server.

8.1.3 Infrastructure Monitoring

BonFIRE provides its experimenters the ability to get monitoring data about the physical machines that run their VMs. We refer to this service as infrastructure monitoring. Most requested infrastructure metrics are: CPU load, total and free memory, free swap memory, the number of VMs on a physical node, disk IO, disk read/write, incoming and outgoing traffics on interfaces, energy usage and CO₂ estimation, etc.

8.1.4 Other Observability Options

The BonFIRE OpenNebula (ONE) testbed sites (EPCC, Inria, HLRS, PSNC) provide experimenters with the deployment log files of their VMs and the system status of the worker nodes so that the experimenters can check the status of their VMs in real time and can have an overview about the current workload of the infrastructure.

ONE VM Logs

The deployment log files of each VM can be exposed by experimenters. There are two possibilities to get the log files:

- Using the BonFIRE Portal
- Using the URL at each testbed site:

Testbed	VM logs at each testbed
HLRS	http://nebulosus.rus.uni-stuttgart.de/logs/{vm-id}.log
Inria	http://frontend.bonfire.grid5000.fr/logs/{vm-id}.log
EPCC	http://bonfire.epcc.ed.ac.uk/logs/{vm-id}/vm.log
PSNC	http://bonfire.psnc.pl/logs/{vm-id}/vm.log

ONE Status Pages

An overview of the current workload about an infrastructure can help plan and schedule future experiments. Two versions of worker node status are provided:

- .txt format: it is human-readable version
- .xml format: it is machine-readable version

Testbed	worker node status at each testbed (txt)	worker node status at each testbed (xml)
HLRS	http://nebulosus.rus.uni-stuttgart.de/one-status.txt	http://nebulosus.rus.uni-stuttgart.de/one-status.xml
EPCC	http://bonfire.epcc.ed.ac.uk/one-status.txt	http://bonfire.epcc.ed.ac.uk/one-status.xml
Inria	http://frontend.bonfire.grid5000.fr/one-status.txt	http://frontend.bonfire.grid5000.fr/one-status.xml
PSNC	http://bonfire.psnc.pl/one-status.txt	http://bonfire.psnc.pl/one-status.xml

Virtual Wall Status page

An overview of the current workload about Virtual Wall at iMinds can be found: <http://ssh.be-ibbt.bonfire-project.eu/production/availability>

8.1.5 Timestamping

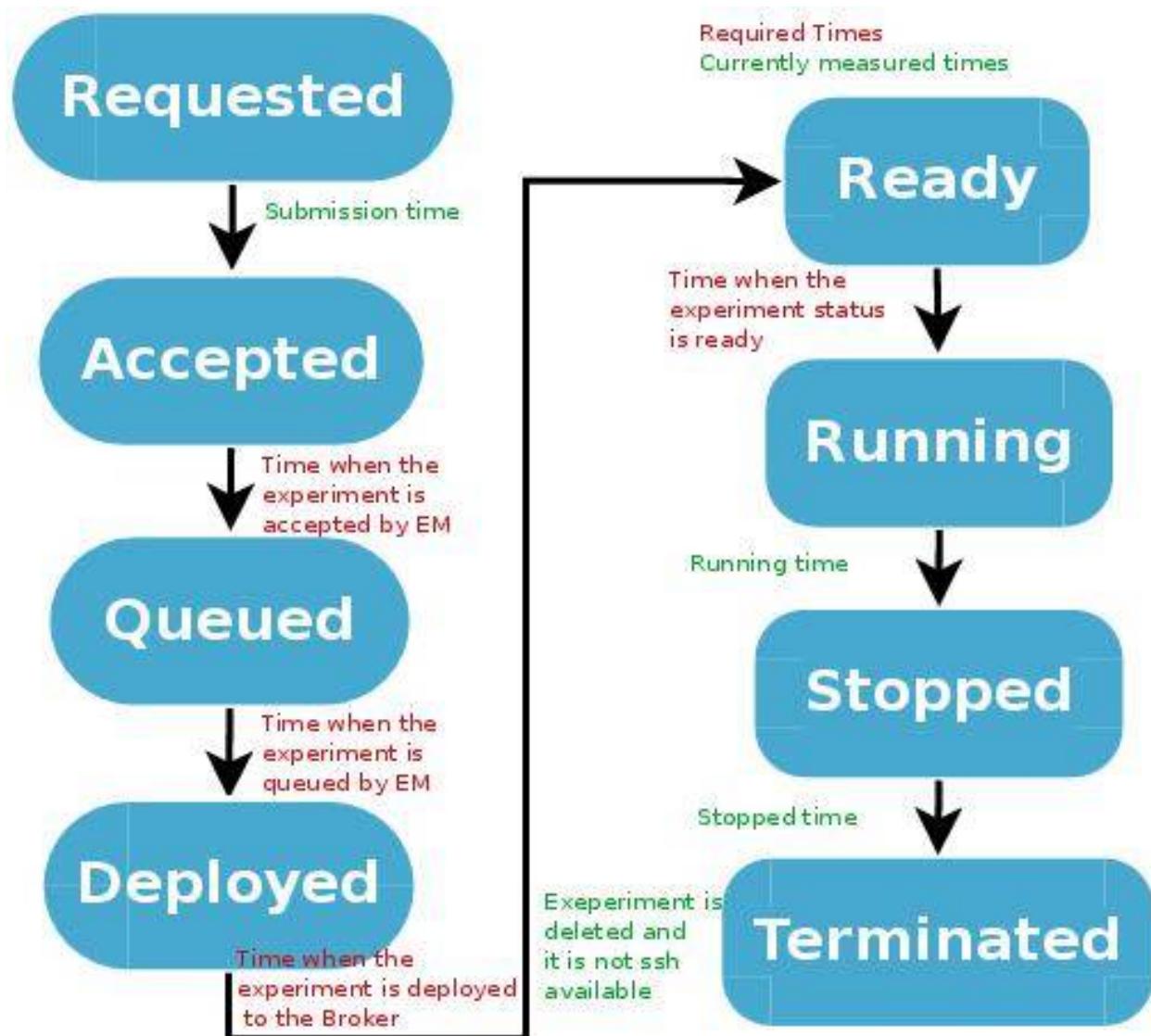
BonFIRE exposes to experimenters the times at which certain events occur in the BonFIRE stack.

Event States

The following experiment states have been defined

Please note the following:

- Requested, Accepted, Queued, Deploying and Deployed states only exist for managed experiments, ie submitted via the Experiment Manager



- “Ready” can be calculated by establishing the time the last VM in an experiment goes into the RUNNING state

Attention is drawn to the case of elasticity-generated VMs; the experimenters should decide whether and how they will affect the calculation of “Ready”.

In BonFIRE Release 3, each component within an experiment is expected to log its own state changes. These logs should be accessible from the computers hosting each component. An exception to this is the Experiment Manager, which logs the state changes and times during deployment to a log file for each managed experiment. On successfully deploying the components within a managed experiment, the Experiment Manager automatically requests that the experiment state be set to RUNNING. The log file for a typical deployment is shown below, demonstrating the QUEUED, DEPLOYABLE, DEPLOYING and DEPLOYED states.

Experiment Manager

GET api.bonfire-project.eu/managed_experiments/680/log

```
"2012-08-14 09:46:18:605 UTC: STATUS QUEUED",
"2012-08-14 09:46:18:607 UTC: STATUS DEPLOYABLE",
"2012-08-14 09:46:18:723 UTC: STATUS DEPLOYING",
"2012-08-14 09:46:18:723 UTC: Creating an experiment called My Experiment ",
"2012-08-14 09:46:18:882 UTC: Created broker experiment",
"2012-08-14 09:46:19:134 UTC: Broker experiment URI is: https://api.bonfire-project.eu/experiments/12
"2012-08-14 09:46:19:134 UTC: Processing resource BonFIRE WAN",
"2012-08-14 09:46:19:134 UTC: Starting to deploying network BonFIRE WAN to fr-inria.",
"2012-08-14 09:46:19:191 UTC: Looking up network BonFIRE WAN",
"2012-08-14 09:46:21:532 UTC: Deployed resource uri is http://localhost:8000/locations/fr-inria/netwo
"2012-08-14 09:46:21:532 UTC: Processing resource BonFIRE WAN",
"2012-08-14 09:46:21:532 UTC: Starting to deploying network BonFIRE WAN to uk-epcc.",
"2012-08-14 09:46:21:732 UTC: Looking up network BonFIRE WAN",
"2012-08-14 09:46:23:495 UTC: Deployed resource uri is http://localhost:8000/locations/uk-epcc/netwo
"2012-08-14 09:46:23:495 UTC: Processing resource BonFIRE Debian Squeeze v3",
"2012-08-14 09:46:23:496 UTC: Starting to deploy storage BonFIRE Debian Squeeze v3 to fr-inria.",
"2012-08-14 09:46:23:579 UTC: Looking up storage BonFIRE Debian Squeeze v3.",
"2012-08-14 09:46:26:058 UTC: Deployed resource uri is http://localhost:8000/locations/fr-inria/stora
"2012-08-14 09:46:26:058 UTC: Processing resource BonFIRE Debian Squeeze v3",
"2012-08-14 09:46:26:059 UTC: Starting to deploy storage BonFIRE Debian Squeeze v3 to uk-epcc.",
"2012-08-14 09:46:26:153 UTC: Looking up storage BonFIRE Debian Squeeze v3.",
"2012-08-14 09:46:28:365 UTC: Deployed resource uri is http://localhost:8000/locations/uk-epcc/stora
"2012-08-14 09:46:28:365 UTC: Processing resource Server",
"2012-08-14 09:46:28:365 UTC: Starting to deploying compute Server to uk-epcc.",
"2012-08-14 09:46:28:475 UTC: Creating compute Server.",
"2012-08-14 09:46:31:752 UTC: Deployed resource uri is http://localhost:8000/locations/uk-epcc/comput
"2012-08-14 09:46:31:752 UTC: Processing resource Client",
"2012-08-14 09:46:31:752 UTC: Starting to deploying compute Client to fr-inria.",
"2012-08-14 09:46:31:839 UTC: Context ServerIP => 172.18.6.149",
"2012-08-14 09:46:31:840 UTC: Creating compute Client.",
"2012-08-14 09:46:34:269 UTC: Deployed resource uri is http://localhost:8000/locations/fr-inria/compu
"2012-08-14 09:46:34:269 UTC: All of the resources have been deployed.",
"2012-08-14 09:46:34:460 UTC: STATUS DEPLOYED",
"2012-08-14 09:46:34:621 UTC: A request has been sent to set the experiment status to RUNNING"
```

8.2 Managing Data Storage for Monitoring

Three options for managing the storage of monitoring data are provided for the experimenter during the creation of an experiment through BonFIRE Portal. These are as follow, see Figure below:

BonFIRE >> Experiments >> Create Experiment

Create Experiment

Choose the Storage for your Experiment's Aggregator (Step 2 of 5)

Do not use external storage

Reuse existing storage
aggregatordatabase

Create new storage

Name ?
Aggregator-Storage-Test

Description ?
Aggregator Storage

Type
DATABLOCK

Persistent ?

Filesystem ?
ext3

Size in MByte ?
1024

→ Continue ▶ Finish

1. **Do not use external storage:** creation of an aggregator without an external storage resource, this means that the database will be stored inside the aggregator image (the VM hosting the aggregator). In this option BonFIRE will offer a fixed aggregator image size (e.g. 6Gb) and the experimenter cannot increase this size for a large expected database storage demand.
2. **Create new storage:** the aggregator will be created with an external storage resource that is mounted as an additional disk while booting the VM image that host the aggregator and the aggregator's database will be then stored into the external storage resource. This storage is permanently available after the deletion or the expiration of the experiment.
3. **Reuse existing storage:** the aggregator will be created with an external storage resource that was already used in the past by a previously deleted experiment. This storage resource includes usually data from that experiment. This option is offered in order to give the experimenter the ability to get/read monitoring data collected from previous experiments.

The experimenter can delete the external storage resource any time through Portal, click on the “Resources” button, a list of all your deployed resources belonging to your experiments will be shown, delete the storage resource, see Figure below.



In addition to these options, the experimenter also has the ability to enable the aggregator to monitor itself (the VM hosting the aggregator image). This is also supported by the BonFIRE Portal while creating the aggregator.

The creation of an experiment is done in five steps. In each step, multiple options are provided for the experimenter. In the first step, the experimenter has the ability to create an experiment with or without an aggregator.

The experimenter has the ability to create an experiment and skip going through all these steps by clicking on FINISH button. This will lead to create an experiment with the default setting. If an experiment is created in the first step with an aggregator, following default setting regarding monitoring will be as follow: the aggregator will be created with an external, additional storage resource that has 1Gb size, the aggregator will also monitor itself.

8.3 How To Set Up Monitoring

8.3.1 Creating an Experiment with Monitoring using the Portal

The following guidelines will describe how to create an experiment with monitoring enabled using the Bonfire portal. Please note this tutorial will focus mostly on monitoring in an experiment and not on the other necessary steps needed when creating an experiment. For information about how to add new metrics when compute resources are deployed, or after an experiment is running, please see [Adding Monitoring Metrics](#):

Log into the portal and create an experiment. At this step one can choose whether to enable monitoring or not, this is done by selecting to deploy an aggregator or not. One can select also the site where the aggregator to be provisioned. In the example of this tutorial, the aggregator is deployed on INRIA site. Furthermore, experimenters have the ability to get monitoring data about the physical machines that run their VMs. We refer to this service as infrastructure monitoring. To get benefit from this additional service, the “Add Infrastructure Metrics” box should be checked with a tick mark as shown in the picture below. It is checked by default.

The screenshot shows the 'Create Experiment' form in the BonFIRE portal. The top navigation bar includes icons for Home, Experiments, and Create Experiment, along with the BonFIRE logo. The main form fields are:

- Name**: Monitoring Experiment
- Description**: Monitoring Test
- Walltime (Lifetime in Minutes)**: 1000
- Add a Monitoring Aggregator at this Site**: uk-epcc (checkbox checked)
- Add Infrastructure Metrics**: (checkbox checked, indicated by a checked box icon)

At the bottom of the form are two buttons: a blue 'Continue' button and a green 'Finish' button.

On clicking the “Finish” button, the experiment will be created with monitoring support. While the “Continue” button will lead to further configuration steps. The second steps is monitoring related step, in which experimenters can identify where to store the monitoring data. For more details, see Monitoring Data Storage.

Once the experiment with monitoring service is created, an aggregator VM will be listed as a normal compute resource of the experiment. This is because the existence of a Zabbix agent running in the aggregator image to monitor itself as well. To start enjoying the monitoring service, click on “Open Aggregator Interface” button that is to be found at the bottom of the page. It is a link to the aggregator GUI.

The screenshot shows the BonFIRE Portal's Experiment Details interface. At the top, there are icons for experiment, monitoring, and dashboard. The main title is "Experiment Details - Monitoring Experiment (ready)". Below this, there is a summary of experiment details:

- Experiment ID:** /experiments/213
- User ID:** alhazmi
- Groups:** users
- Creation Time:** Tue, 02/14/12 10:18:22 UTC
- Last Updated:** Tue, 02/14/12 10:18:22 UTC
- Expires:** Wed, 02/15/12 02:58:22 UTC

Buttons for "Delete" and "GO" are available. Below the summary are three resource tables:

- Compute Resources:** Shows one entry: BonFIRE-monitor /locations/uk-epcc/computes/1343 (small, 1 CPU, 1024MB, BonFIRE Zabbix Aggregate, IP 172.18.6.225, OK, RUNNING). Buttons include "Add Compute at" and "add".
- Storage Resources:** No entries listed.
- Network Resources:** No entries listed.

A "Monitoring" section follows, displaying a message: "Found monitoring aggregator: BonFIRE-monitor - /locations/uk-epcc/computes/1343" and a "Open Aggregator Interface" button.

The portal will automatically log you in to Zabbix Server GUI using the username Admin and the password BONFIRE_EXPERIMENT_AGGREGATOR_PASSWORD. This password is generated by BonFIRE broker one per experiment and to be found on the aggregator machine in the contextualisation file located at /etc/default/bonfire, or from the xml description of the aggregator machine that can be shown through the BonFIRE Portal. Using this credential, you can login to Zabbix Server through zabbix-API as well.

The screenshot shows the Zabbix 1.8.6 login screen. The title bar says "ZABBIX". The menu bar includes "Monitoring", "Inventory", "Reports", "Configuration", and "Administration". The main area is titled "Login" and contains fields for "Login name" (Admin) and "Password" (*****), with an "Enter" button. Below the login form, the text "Zabbix 1.8.6 Copyright 2001-2011 by SIA Zabbix" is visible.

On clicking the “configuration” and then the “Hosts” tabs, the experiment resources will be listed. At the top of the page on the right side, one can filter the shown resources based on groups (physical machines “PM”, virtual machines “VM” or even both “all”). Based on the created experiment of this tutorial, the aggregator VM and the physical machine hosting it are to be seen in the picture below.

The screenshot shows the Zabbix Configuration > Hosts page. The top navigation bar includes "Help", "Get support", "Print", "Profile", and "Logout". The menu bar has "Monitoring", "Inventory", "Reports", "Configuration", and "Administration". The search bar is labeled "SEARCH:". The host list table shows two hosts:

Name	Applications	Items	Triggers	Graphs	DNS	IP	Port	Templates	Status	Availability
BonFIRE-monitor-1343	Applications (13)	Items (102)	Triggers (0)	Graphs (0)	BonFIRE-monitor-1343	127.0.0.1	10050	BonFIRE Template	Monitored	
vmhost1	Applications (0)	Items (16)	Triggers (0)	Graphs (0)	-	0.0.0.0	-	BonFIRE Template PM	Monitored	

Buttons at the bottom include "Export selected" and "Go (0)". The footer displays "Zabbix 1.8.6 Copyright 2001-2011 by SIA Zabbix" and "Connected as 'Admin'".

More than 100 metrics of a VM are monitored. Those are the ones monitored by Zabbix by default. In Zabbix, items refer to metrics. On clicking the “Items” button, all items will be listed, a part of them are to be seen in the following

shortcut.

ITEMS										
Displaying 1 to 50 of 102 found										
Hosts list		Applications (13)		Triggers (0)		Graphs (0)		Host: BonFIRE-monitor-1343		
								DNS: BonFIRE-monitor-1343		
								IP: 127.0.0.1		
								Port: 10050		
								Status: Monitored		
								Availability: Unknown		
1 2 3 Next >										
Wizard	Description	Triggers	Key	Interval	History	Trends	Type	Status	Applications	Error
<input type="checkbox"/>	BonFIRE Template:Buffers memory		vm.memory.size[buffers]	30	7	365	Zabbix agent (active)	Active	Availability	✓
<input type="checkbox"/>	BonFIRE Template:Cached memory		vm.memory.size[cached]	30	7	365	Zabbix agent (active)	Active	Memory	✓
<input type="checkbox"/>	BonFIRE Template:Checksum of /usr/sbin/sshd		vfs.file_cksum[/usr/sbin/sshd]	600	7	365	Zabbix agent (active)	Active	Integrity	✓
<input type="checkbox"/>	BonFIRE Template:Checksum of /usr/bin/ssh		vfs.file_cksum[/usr/bin/ssh]	600	7	365	Zabbix agent (active)	Active	Integrity	✓
<input type="checkbox"/>	BonFIRE Template:Checksum of /vmlinuz		vfs.file_cksum[/vmlinuz]	600	7	365	Zabbix agent (active)	Active	Integrity	✓
<input type="checkbox"/>	BonFIRE Template:Checksum of /etc/services		vfs.file_cksum[/etc/services]	600	7	365	Zabbix agent (active)	Active	Integrity	✓
<input type="checkbox"/>	BonFIRE Template:Checksum of /etc/inetd.conf		vfs.file_cksum[/etc/inetd.conf]	600	7	365	Zabbix agent (active)	Not supported	Integrity	✗
<input type="checkbox"/>	BonFIRE Template:Checksum of /etc/passwd		vfs.file_cksum[/etc/passwd]	600	7	365	Zabbix agent (active)	Active	Integrity	✓
<input type="checkbox"/>	BonFIRE Template:CPU system time (avg1)		system.cpu.util[,system,avg1]	10	90	365	Zabbix agent (active)	Active	Performance	✓
<input type="checkbox"/>	BonFIRE Template:CPU nice time (avg1)		system.cpu.util[,nice,avg1]	10	90	365	Zabbix agent (active)	Active	Performance	✓
<input type="checkbox"/>	BonFIRE Template:CPU idle time (avg1)		system.cpu.util[,idle,avg1]	10	90	365	Zabbix agent (active)	Active	Performance	✓
<input type="checkbox"/>	BonFIRE Template:CPU iowait time (avg1)		system.cpu.util[,iowait,avg1]	10	90	365	Zabbix agent (active)	Active	Performance	✓
<input type="checkbox"/>	BonFIRE Template:CPU user time (avg1)		system.cpu.util[,user,avg1]	10	90	365	Zabbix agent (active)	Active	Performance	✓
<input type="checkbox"/>	BonFIRE Template:Email (SMTP) server is running		net.tcp.service[smtbp]	60	7	365	Zabbix agent (active)	Active	Services	✓
<input type="checkbox"/>	BonFIRE Template:Free disk space on /usr		vfs.fs.size[/usr,free]	30	7	365	Zabbix agent (active)	Active	Availability	✓
<input type="checkbox"/>	BonFIRE Template:Free disk space on /var		vfs.fs.size[/var,free]	30	7	365	Zabbix agent (active)	Active	Availability	✓
<input type="checkbox"/>	BonFIRE Template:Free disk space on /tmp		vfs.fs.size[/tmp,free]	30	7	365	Zabbix agent (active)	Active	Availability	✓
<input type="checkbox"/>	BonFIRE Template:Free disk space on /home		vfs.fs.size[/home,free]	30	7	365	Zabbix agent (active)	Active	Availability	✓
<input type="checkbox"/>	BonFIRE Template:Free disk space on /opt		vfs.fs.size[/opt,free]	30	7	365	Zabbix agent (active)	Active	Availability	✓

Items are also categorized as “Applications” according to their relevance.

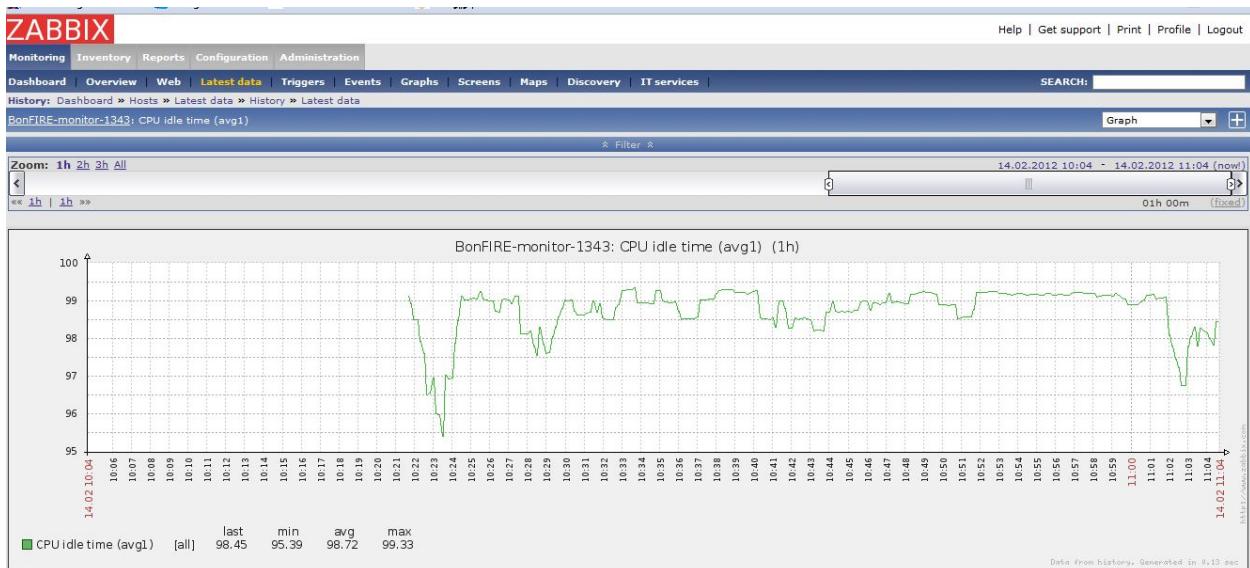
ZABBIX									
Help Get support Print Profile Logout									
Monitoring Inventory Reports Configuration Administration									
Host groups Templates Hosts Maintenance Web Actions Screens Slides Maps Discovery IT services									
History: Dashboard > Nodes > Host groups > Hosts > Configuration of items									
SEARCH: <input type="text"/>									
APPLICATIONS									
Displaying 1 to 13 of 13 found									
Hosts list	Items (102)	Triggers (0)	Graphs (0)	Host: BonFIRE-monitor-1343	DNS: BonFIRE-monitor-1343	IP: 127.0.0.1	Port: 10050	Status: Monitored	Availability: Unknown
<input type="checkbox"/>	Application							Show	
<input type="checkbox"/>	BonFIRE Template:Availability							Items (47)	
<input type="checkbox"/>	BonFIRE Items							Items (0)	
<input type="checkbox"/>	BonFIRE Template:CPU							Items (0)	
<input type="checkbox"/>	BonFIRE Template:Filesystem							Items (0)	
<input type="checkbox"/>	BonFIRE Template:General							Items (7)	
<input type="checkbox"/>	BonFIRE Template:Integrity							Items (6)	
<input type="checkbox"/>	BonFIRE Template:Log files							Items (1)	
<input type="checkbox"/>	BonFIRE Template:Memory							Items (4)	
<input type="checkbox"/>	BonFIRE Template:Network							Items (6)	
<input type="checkbox"/>	BonFIRE Template:OS							Items (6)	
<input type="checkbox"/>	BonFIRE Template:Performance							Items (8)	
<input type="checkbox"/>	BonFIRE Template:Processes							Items (10)	
<input type="checkbox"/>	BonFIRE Template:Services							Items (7)	

To see the monitoring data of any resource, click on the “Monitoring” then on the “Latest data” tabs. The latest data of each monitored metric will be shown. In the following picture, latest values of the performance-related metrics of the “BonFIRE-monitor-1343” VM are shown.

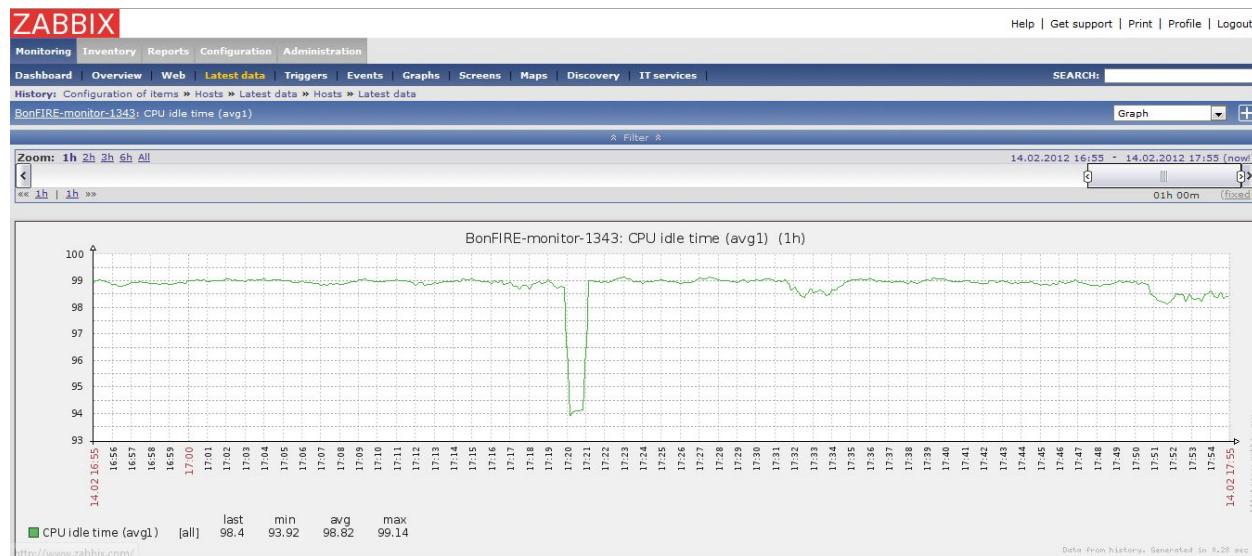
The screenshot shows the BonFIRE monitoring interface. At the top, there's a navigation bar with tabs for Monitoring, Inventory, Reports, Configuration, and Administration. Below that is a secondary navigation bar with links for Dashboard, Overview, Web, Latest data, Triggers, Events, Graphs, Screens, Maps, Discovery, and IT services. A search bar labeled 'SEARCH:' is also present. The main content area is titled 'ITEMS' and displays a table of monitoring items. The table has columns for Description, Last check, Last value, Change, and History. There are filters at the top of the table. On the left side, there's a sidebar with a tree view of item categories: Availability (47 items), General (6 items), Integrity (5 items), Log files (1 items), Memory (4 items), Network (4 items), OS (3 items), and Performance (8 items). The 'Performance' category is expanded, showing metrics like CPU idle time (avg1), CPU iowait time (avg1), etc.

Furthermore, one can show the monitoring data in graphics as well. To see the results in graphical views for any period of time, click on the “Graph” of the target metric.

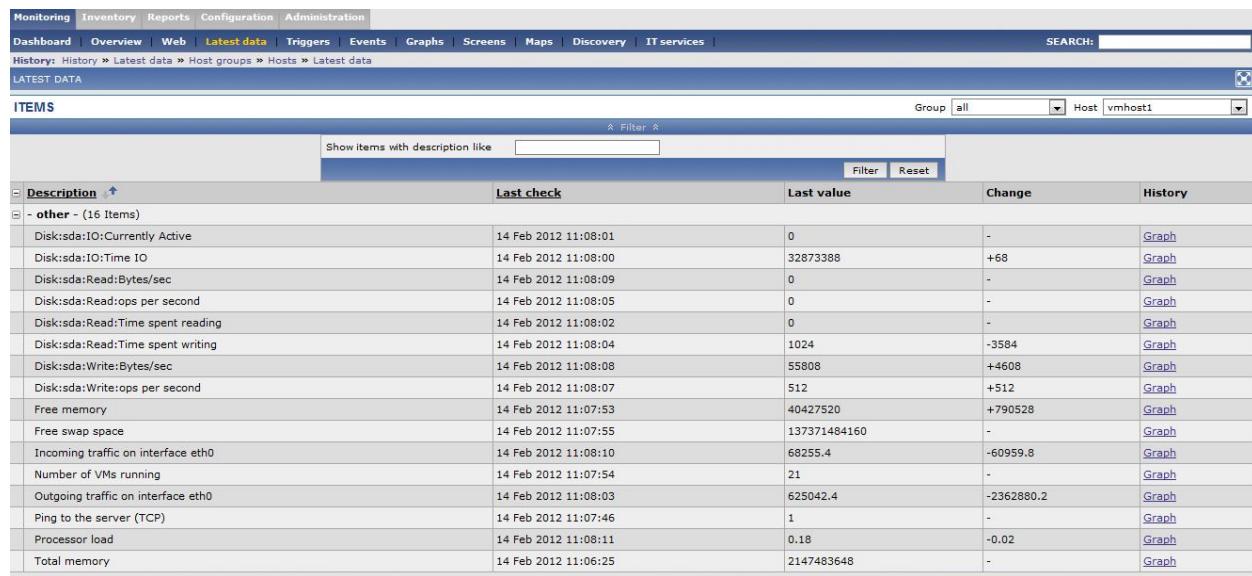
Example 1: Monitoring data of the “CPU idle time” metric for one hour within the time period between 10:04-11:04 am, Feb 14, 2012 is shown below.



Example 2: Monitoring data of the same metric but for approx. five hours, within the time period between 12:44 am and 05:58 pm, Feb 14, 2012 is shown below.



Regarding the infrastructure monitoring, only partial information is provided to experimenters about the physical machines hosting their VMs. Currently, data about 16 metrics are provided, see the following picture that shows the latest data of these metrics about the physical machine “vmhost1” that hosts the “BonFIRE-monitor-1343” VM.



8.3.2 Creating an Experiment with Monitoring using OCCI

BonFIRE offers its users the possibility to create their experiments through the BonFIRE Portal and through the BonFIRE API as well.

Since the monitoring is one of the key features of BonFIRE, it is possible for BonFIRE users to create their experiments with monitoring as a service not only through the Portal but also through the BonFIRE API.

Creation of an Experiment with Monitoring-as-a-Service at API Level:

To create a new experiment resource through the API, follow [Creating Experiments](#). The following assumes that you already created an experiment resource, with ID 11415.

For having monitoring service, an aggregator is required. Therefore an aggregator compute resource belong to the experiment should be deployed. In which site the aggregator is created is left to the experimenter.

Before creating an aggregator, the experimenter has to get the available aggregator images in the target site. Using the API, it's easy to list these kinds of resources for a given testbed. At `fr-inria` for example, you would fetch the list of existing storage resources (from type `datablock` used for instance as storage for aggregator database, from type `OS` as aggregator images) like this:

```
$ curl -ki -u USER_NAME https://api.bonfire-project.eu/locations/fr-inria/storages
```

Now the aggregator can be created as a compute resource that boots using the BonFIRE Zabbix Aggregator v5 image for instance. The monitoring data is stored in the aggregator database. The aggregator database can be either stored inside the aggregator VM or outside as a datablock that is attached to the aggregator VM as an external disk. An old datablock from previous experiment can be reused again. The later could be needed if an experimenter wants to make comparison between an old experiment and a new one, or to enable getting the data from a deleted/expired experiment, since the datablocks in BonFIRE are persistent resources and permanently available for their owners.

Three Options of Creating an Aggregator

1. Without external storage:

An aggregator is deployed as a normal compute resource with exception that it includes Zabbix software (server and agent).

To create a compute resource, you need the know following things:

- the experiment ID is required,
- the type of instance you want to create (`tiny`, `small`, `large`, etc.). This specifies how much computing power you need, in terms of number of CPUs and RAM. See [Instance Types](#) for up to date information;
- the type of instance should be at least `small`, but not `lite`, except for some testbed with only `lite` instance-type,
- the location at which you want to create the resource (here: `fr-inria`);
- the image to deploy (here: BonFIRE Zabbix Aggregator v5);
- the network to which the resource will be attached (here: BonFIRE WAN).

In addition to these, in the case of creating a compute resource for the aggregator, the aggregator compute resource must be called “BonFIRE-monitor”.

To find the compute resource location, use the following command line:

```
curl -i -k -u USER_NAME ~/.ssh/id_bonfire https://api.bonfire-project.eu/locations/
```

or refer to [Monitoring Zabbix API](#) and [ZabbixAPI](#). A creation request is sent to create the aggregator compute resource using an aggregator image (e.g. the BonFIRE Zabbix Aggregator v5 image at `fr-inria`) as follow: (note that the experiment ID 11415 will be used)

```
$ curl -kni https://api.bonfire-project.eu/experiments/11415/computes \
-X POST -H'Content-Type: application/vnd.bonfire+xml' \
-d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>BonFIRE-monitor</name>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/fr-inria/storages/0"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
</compute>'
```

```
</disk>
<nics>
  <network href="/locations/fr-inria/networks/53"/>
</nics>
<context>
  <usage>zabbix-agent;infra-monitoring-init</usage>
</context>
<link href="/locations/fr-inria" rel="location"/>
</compute>'
```

Here is what you would get back:

```
HTTP/1.1 201 Created
Date: Wed, 08 Aug 2012 13:13:52 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/1.9.5)
Location: https://api.bonfire-project.eu:443/locations/fr-inria/computes/24188
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 1.316646
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/24188">
  <id>24188</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <uname href="/locations/fr-inria/users/16">alhazmi</uname>
  <groups>users</groups>
  <name>BonFIRE-monitor</name>
  <instance_type href="/locations/fr-inria/instance_type/lite">lite</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/0" name="BonFIRE Zabbix Aggregator v5"/>
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <nics>
    <network href="/locations/fr-inria/networks/53" name="BonFIRE WAN"/>
    <ip>172.18.248.39</ip>
    <mac>02:00:ac:12:f8:27</mac>
  </nics>
  <context>
    <authorized_keys>ssh-rsa AAAAB3NzaC1yc2EAAAQAAIEAj3OvWrZq4cqc4puHGCBPq/gDgKXprFt+wa/N7PG1JfS</authorized_keys>
    <bonfire_experiment_aggregator_password>xq9tjw</bonfire_experiment_aggregator_password>
    <bonfire_experiment_expiration_date>1344461395</bonfire_experiment_expiration_date>
    <bonfire_experiment_id>11415</bonfire_experiment_id>
    <bonfire_experiment_routing_key>d2ac0352d437fe11b490</bonfire_experiment_routing_key>
    <bonfire_provider>fr-inria</bonfire_provider>
    <bonfire_resource_id>24188</bonfire_resource_id>
    <bonfire_resource_name>BonFIRE-monitor</bonfire_resource_name>
    <bonfire_uri>https://api.bonfire-project.eu</bonfire_uri>
    <cluster>default</cluster>
    <dns_servers>131.254.204.4</dns_servers>
    <files>/srv/cloud/context /srv/cloud/context/lib /srv/cloud/context/distributions /srv/cloud/cont</files>
  </context>
</compute>
```

```

<hostname>BonFIRE-monitor-24188</hostname>
<ntp_servers>ntp.rennes.grid5000.fr</ntp_servers>
<public_gateway>131.254.204.8</public_gateway>
<target>xvdb</target>
<usage>zabbix-agent;infra-monitoring-init</usage>
<wan_gateway>172.18.248.1</wan_gateway>
<wan_ip>172.18.248.39</wan_ip>
<wan_mac>02:00:ac:12:f8:27</wan_mac>
</context>
</compute>

```

The usage element in the context tag is used for setting the required contextualisation information needed for configuring monitoring services.

In this case it set to zabbix-agent;infra-monitoring-init. The first zabbix-agent means that the aggregator should monitor itself, while infra-monitoring-init means that the infrastructure monitoring should be supported.

All combinations of setting the usage element are presented in the contextualisation usage below.

2. With external storage:

If external data storage for monitoring data will be used, it should be created before the creation of the aggregator compute resource. If it is required only to include that data during the experiment lifetime, it should be created outside the experiment. If the storage is created as a part of the experiment resource, it will not be permanently available after the experiment lifetime or deletion, and therefore if the user wants his data to be permanently available, the used external storage should be created outside the experiment. This storage is mounted to the aggregator compute resource as an external storage. In BonFIRE the type of this kind of storage is called datablock. In this case an additional disk should be added to the OCCI request as shown below:

```

<disk>
  <storage href='/locations/site_name/storages/datablock_id' />
  <type>disk</type>
  <target>hdb</target>
</disk>

```

The zabbix database is moved into this storage and the mysql server is configured accordingly. This is done through the contextualisation script called zabbix-aggr-extend that is executed after booting the aggregator compute resource. This script is passed through the context section of the OCCI request.

To create an external storage for an experiment, besides the information above, you should know the following:

- the storage name is not fixed, you can choose your own name,

To check the compute, storage information, use the following command,

```
curl -i -k -u USER_NAME ~/.ssh/id_bonfire https://api.bonfire-project.eu/locations/
```

and choose your testbed, to create the storage, the command lines:

```
$ curl -kni https://api.bonfire-project.eu/locations/fr-inria/storages \
-X POST -H 'Content-Type: application/vnd.bonfire+xml' -d '
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Aggregator-Storage</name>
  <description>Aggregator Storage</description>
  <type>DATABLOCK</type>
  <size>1024</size>
  <fstype>ext3</fstype>
```

```
<persistent>YES</persistent>
<link rel="location" href="/locations/fr-inria" />
</storage> '
```

Here is what you would get back:

```
HTTP/1.1 201 Created
Date: Wed, 08 Aug 2012 14:53:38 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/1.9.5)
Location: https://api.bonfire-project.eu:443/locations/fr-inria/storages/1148
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 1.602890
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/storages/1148">
  <id>1148</id>
  <name>alhazmi_Aggregator-Storage</name>
  <user_id href="/locations/fr-inria/users/16">alhazmi</user_id>
  <groups>users</groups>
  <state>READY</state>
  <type>DATABLOCK</type>
  <description>Aggregator Storage</description>
  <size>1024</size>
  <fstype>ext3</fstype>
  <public>NO</public>
  <persistent>YES</persistent>
</storage>
```

A storage from type datablock with the ID 1148 as an external storage is created at fr-inria, /locations/fr-inria/storages/1148.

This datablock with the ID 1148 is then added as external disk to the aggregator. The creation request of the aggregator with the external datablock is:

Then, just send a POST request on your experiment computes URI, like this:

```
$ curl -kni https://api.bonfire-project.eu/experiments/11417/computes \
-X POST -H'Content-Type: application/vnd.bonfire+xml' \
-d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>BonFIRE-monitor</name>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/fr-inria/storages/0"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
  <disk>
    <storage href="/locations/fr-inria/storages/1148"/>
    <type>disk</type>
    <target>hdb</target>
  </disk>
  <nic>
```

```

<network href="/locations/fr-inria/networks/53"/>
</nic>
<context>
  <usage>zabbix-agent;zabbix-aggr-extend;infra-monitoring-init</usage>
</context>
<link href="/locations/fr-inria" rel="location"/>
</compute>

```

Here is what you would get back:

```

HTTP/1.1 201 Created
Date: Wed, 08 Aug 2012 14:55:09 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/1.9.5)
Location: https://api.bonfire-project.eu:443/locations/fr-inria/computes/24190
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 1.535398
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/24190">
  <id>24190</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <uname href="/locations/fr-inria/users/16">alhazmi</uname>
  <groups>users</groups>
  <name>BonFIRE-monitor</name>
  <instance_type href="/locations/fr-inria/instance_type/lite">lite</instance_type>
  <state>PENDING</state>
  <disk id="0">
    <storage href="/locations/fr-inria/storages/0" name="BonFIRE Zabbix Aggregator v5"/>
    <type>DISK</type>
    <target>xvda</target>
  </disk>
  <disk id="1">
    <storage href="/locations/fr-inria/storages/1148" name="alhazmi_Aggregator-Storage"/>
    <type>DISK</type>
    <target>xvde</target>
  </disk>
  <nics>
    <nic>
      <network href="/locations/fr-inria/networks/53" name="BonFIRE WAN"/>
      <ip>172.18.248.42</ip>
      <mac>02:00:ac:12:f8:2a</mac>
    </nic>
  </nics>
  <context>
    <authorized_keys>ssh-rsa AAAAB3NzaC1yc2EAAAQAAIEAj3OvWrZq4cqcz4puHGCBPq/gDgKXprFt+wa/N7PG1JfS...
    <bonfire_experiment_aggregator_password>afmehq</bonfire_experiment_aggregator_password>
    <bonfire_experiment_expiration_date>1344462544</bonfire_experiment_expiration_date>
    <bonfire_experiment_id>11417</bonfire_experiment_id>
    <bonfire_experiment_routing_key>a5f2f8cfdf18ab565904</bonfire_experiment_routing_key>
    <bonfire_provider>fr-inria</bonfire_provider>
    <bonfire_resource_id>24190</bonfire_resource_id>
    <bonfire_resource_name>BonFIRE-monitor</bonfire_resource_name>
    <bonfire_uri>https://api.bonfire-project.eu</bonfire_uri>
  </context>
</compute>

```

```
<cluster>default</cluster>
<dns_servers>131.254.204.4</dns_servers>
<files>/srv/cloud/context /srv/cloud/context/lib /srv/cloud/context/distributions /srv/cloud/cont
<hostname>BonFIRE-monitor-24190</hostname>
<ntp_servers>ntp.rennes.grid5000.fr</ntp_servers>
<public_gateway>131.254.204.8</public_gateway>
<target>xvdb</target>
<usage>zabbix-agent; zabbix-aggr-extend; infra-monitoring-init</usage>
<wan_gateway>172.18.248.1</wan_gateway>
<wan_ip>172.18.248.42</wan_ip>
<wan_mac>02:00:ac:12:f8:2a</wan_mac>
</context>
</compute>
```

The usage element in the context tag in this case set to zabbix-agent; zabbix-aggr-extend; infra-monitoring-init, zabbix-aggr-extend is needed to configure the aggregator in a way to store its database and of course the monitoring data into the external storage.

All combinations of setting the usage element are presented in the contextualisation usage below.

3. Reuse an old external storage:

An existing datablock is reused. The creation request of the aggregator with an existing external datablock is as the one with an external storage, the only difference here is that you don't need to create a new external storage rather than using an existing one. The usage element in the context tag in this case is set as well to zabbix-agent; zabbix-aggr-extend; infra-monitoring-init.

The contextualisation usage of the OCCI request to create an aggregator:

BonFIRE provides its user multiple tools to use BonFIRE, BonFIRE Portal, Experiment Descriptors, OCCI/API via HTTP(cURL), Command Line Interface Tools, and Restfully. Using any of these the user can create experiments and resources. In order for enjoying monitoring services, an aggregator is required.

If the user wants the aggregator compute resource to be monitored, the monitoring agent inside the aggregator should be run; this is done by setting the usage element of the context tag of the OCCI request to zabbix-agent.

BonFIRE users can get monitoring information about the physical machines that host their VMs. This service is activated if the usage element of the context tag contains the infra-monitoring-init script for post installation.

BonFIRE experimenter might need to track all the events of his experiment such as create, delete, stop, suspend, etc. resources, and have all this information stored in the aggregator. This is done through setting the usage element to log-MQevents-in-zabbix.

All these services are made optional for BonFIRE users. The user while creating an experiment with an aggregator can request any or all of them. The portal provides these optional services in form of check boxes; the user can either check or uncheck any. If the user uses BonFIRE Experiment Descriptor or any other user tools to create experiments he has to be aware on the fact that the usage element should be set correctly. To this end, there are various setting combinations of the usage element based on the required monitoring services to be discussed here.

- If the aggregator does not monitor itself, and no need for using an external storage and for infrastructure monitoring: the usage element is not added at all.
- **If the aggregator monitors itself only:**

```
<usage>zabbix-agent</usage>
```

- If an external storage is used:

```
<usage>zabbix-aggr-extend</usage>
```

- If infrastructure monitoring is requested:

```
<usage>infra-monitoring-init</usage>
```

- If the aggregator monitors itself and an external storage is used:

```
<usage>zabbix-agent; zabbix-aggr-extend</usage>
```

- If an external storage is used and infrastructure monitoring is requested:

```
<usage>zabbix-aggr-extend; infra-monitoring-init</usage>
```

- If the aggregator monitors itself, an external storage is used, and infrastructure monitoring is requested:

```
<usage>zabbix-agent; zabbix-aggr-extend; infra-monitoring-init</usage>
```

- If log experiment events is requested:

```
<usage>zabbix-agent; log-MQevents-in-zabbix</usage>
```

((in this case the aggregator should monitor itself to include under its monitored metrics the new logging metric that shows the experiment events through zabbix))

- If the aggregator monitors itself and log experiment events is requested:

```
<usage>zabbix-agent; log-MQevents-in-zabbix</usage>
```

- If an external storage is used and log experiment events is requested:

```
<usage>zabbix-agent; zabbix-aggr-extend; log-MQevents-in-zabbix</usage>
```

((as before, the aggregator should monitor itself to include under its monitored metrics the new logging metric that shows the experiment events through zabbix))

- If infrastructure monitoring and log experiment events are requested:

```
<usage>zabbix-agent; infra-monitoring-init; log-MQevents-in-zabbix</usage>
```

((as before, the aggregator should monitor itself to include under its monitored metrics the new logging metric that shows the experiment events through zabbix))

- If the aggregator monitors itself, an external storage is used, and log experiment events is requested:

```
<usage>zabbix-agent; zabbix-aggr-extend; log-MQevents-in-zabbix</usage>
```

- If the aggregator monitors itself, and infrastructure monitoring and log experiment events are requested:

```
<usage>zabbix-agent; infra-monitoring-init; log-MQevents-in-zabbix</usage>
```

- If an external storage is used, and infrastructure monitoring and log experiment events are requested:

```
<usage>zabbix-agent; zabbix-aggr-extend; infra-monitoring-init; log-MQevents-in-zabbix</usage>
```

- If the aggregator monitors itself, an external storage is used, infrastructure monitoring and log experiment events are req

```
<usage>zabbix-agent;zabbix-aggr-extend;infra-monitoring-init;log-MQevents-in-zabbix</usage>
```

Creation of Compute Resource to be Monitored

Till now an experiment is created with monitoring support. The user is now able to create the required compute resource required in the experiment.

All BonFIRE images include Zabbix agent already preinstalled. While creating a compute resource, the IP address of the aggregator should be configured. Let's create a compute resource:

```
$ curl -kni https://api.bonfire-project.eu/experiments/11417/computes \
-X POST -H'Content-Type: application/vnd.bonfire+xml' \
-d '<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Compute-Resource</name>
  <instance_type>lite</instance_type>
  <disk>
    <storage href="/locations/fr-inria/storages/1"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
  <nic>
    <network href="/locations/fr-inria/networks/53"/>
  </nic>
  <context>
    <usage>zabbix-agent</usage>
    <aggregator_ip>172.18.248.42</aggregator_ip>
  </context>
  <link href="/locations/fr-inria" rel="location"/>
</compute>'
```

Note that the image that is deployed for this compute resource (here: BonFIRE Debian Squeeze v3) is different than that used for the aggregator.

In the `<context>` tag, the `usage` element is set to `zabbix-agent` that is responsible for starting an agent on the compute resource, while the `aggregator_ip` element is set to the IP-address of the aggregator.

Here is what you would get back:

```
HTTP/1.1 201 Created
Date: Wed, 08 Aug 2012 15:04:40 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Via: 1.1 enactor:8080 (bonfire-enactor/0.5),1.1 bonfire-api (bonfire-api/1.9.5)
Location: https://api.bonfire-project.eu:443/locations/fr-inria/computes/24191
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 1.514117
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/fr-inria/computes/24191">
  <id>24191</id>
  <cpu>0.5</cpu>
  <memory>256</memory>
  <uname href="/locations/fr-inria/users/16">alhazmi</uname>
  <groups>users</groups>
```

```

<name>Compute-Resource</name>
<instance_type href="/locations/fr-inria/instance_type/lite">lite</instance_type>
<state>PENDING</state>
<disk id="0">
  <storage href="/locations/fr-inria/storages/1" name="BonFIRE Debian Squeeze v3"/>
  <type>DISK</type>
  <target>xvda</target>
</disk>
<nics>
  <network href="/locations/fr-inria/networks/53" name="BonFIRE WAN"/>
  <ip>172.18.248.43</ip>
  <mac>02:00:ac:12:f8:2b</mac>
</nics>
<context>
  <aggregator_ip>172.18.248.42</aggregator_ip>
  <authorized_keys>ssh-rsa AAAAB3NzaC1yc2EAAAQABJQAAIEAj3OvWrZq4cqc4puHGCBPq/gDgKXprFt+wa/N7PG1JfS
  <bonfire_experiment_aggregator_password>afmehq</bonfire_experiment_aggregator_password>
  <bonfire_experiment_expiration_date>1344462544</bonfire_experiment_expiration_date>
  <bonfire_experiment_id>11417</bonfire_experiment_id>
  <bonfire_experiment_routing_key>a5f2f8cfdf18ab565904</bonfire_experiment_routing_key>
  <bonfire_provider>fr-inria</bonfire_provider>
  <bonfire_resource_id>24191</bonfire_resource_id>
  <bonfire_resource_name>Compute-Resource</bonfire_resource_name>
  <bonfire_uri>https://api.bonfire-project.eu</bonfire_uri>
  <cluster>default</cluster>
  <dns_servers>131.254.204.4</dns_servers>
  <files>/srv/cloud/context /srv/cloud/context/lib /srv/cloud/context/distributions /srv/cloud/cont
  <hostname>Compute-Resource-24191</hostname>
  <ntp_servers>ntp.rennes.grid5000.fr</ntp_servers>
  <public_gateway>131.254.204.8</public_gateway>
  <target>xvdb</target>
  <usage>zabbix-agent</usage>
  <wan_gateway>172.18.248.1</wan_gateway>
  <wan_ip>172.18.248.43</wan_ip>
  <wan_mac>02:00:ac:12:f8:2b</wan_mac>
</context>
</compute>

```

Identify Metrics to be Monitored:

The experimenter has the ability to add customer metrics to be monitored within the compute resource. How these metrics are deployed, can be found at [Creating an Experiment with Monitoring using the Portal](#).

How to get monitoring data:

There are several options for getting monitoring data, see [Accessing and Downloading Monitoring Data](#).

8.3.3 Creating an Experiment with Monitoring using Experiment Descriptors

BonFIRE offers its users the possibility to create their experiments through Experiment Descriptor. Please check the page how to set up monitoring in JSON for more information.

8.3.4 Adding Monitoring Metrics

Adding Metrics with OCCI Contextualisation

The following steps show how to create a new resource to the experiment with additional monitoring features.

1. Add compute resources to the experiment by selecting the site where the compute resource will be provisioned. Also one can set up the name of the resource, its descriptions, the underlying VM base image, the storage and the network resources. The VM base image contains a preinstalled monitoring agent configured for the Bonfire infrastructure. On clicking on the “Finish” button, a new resource will be normally created, but while clicking on the “Continue” button, you will come to the next step. Where, experimenter has the ability to edit the xml-based resource creation request.

2. In this step the user will be able to edit the xml-based OCCI request before it is send to the Bonfire infrastructure (Bonfire broker). At this level one can specify monitoring metrics which will be registered at the agent and aggregator side by configuring them as part of the OCCI contextualization tag (<context>). The following snippet depicts the metrics configured in the OCCI context :

```
<context>
  ...
  <metrics>
    <metric>key1, command1, rate=rate1, valuetype=valuetype1, history=history1</metric>
    <metric>keyi, commandi, rate=ratei, valuetype=valuetypei, history=historyi</metric>
  </metrics>
  ...
</context>
```

One can insert as many metrics, [<metric>key1, command1, rate=rate1, valuetype=valuetype1, history=history1</metric> <metric>keyi, commandi, rate=ratei, valuetype=valuetypei, history=historyi </metric>], where i =2..., between the CDATA tags, while the metrics tag should be unique. To be noticed is that CDATA should be written in large letters; otherwise, an error “Error validating OCCI XML” will be reported. However, CDATA is used about text data that should not be parsed by the XML parser, since

some characters like “<” and “&” are illegal in XML elements. For BonFIRE contextualization proposes, it might be that such characters are used as parts of the sent commands of the added metrics, therefore CDATA is used. A key of a metric represents the unique identifier of the key over the set of configured keys at the agent and aggregator. The command of a key is a bash script which is executed by the agent and of which result is reported to the aggregator. These two properties of the metric (key and command) and their order are mandatory when creating metrics. The metrics definition is fully compatible with the Zabbix definition of user parameters. For further details please consult the [Zabbix user parameters documentation](#). In addition to this, it's currently possible for the experimenter to optionally configure its personalized added metrics through the OCCI request, by defining the update rate of monitoring information in second, the suitable type of the return values of the monitored metric (character, numeric unsigned, float, etc.) and also the history in days. The value types defined in zabbix as follow: float=0, string= 1, log= 2, numeric (unsigned integer)= 3 and Text=4. These three properties (rate, valuetype and history) are optional ones. One, two or the three can be used. The order of them is not important.

An example of an OCCI request for a compute resource with monitoring metrics is depicted in the following listing:

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
<name>Agent</name>
<description>monitoring agent</description>
<instance_type>small</instance_type>
<disk>
    <storage href="/locations/fr-inria/storages/11"/>
    <type>DISK</type>
    <target>hda</target>
</disk>
<nic>
    <network href="/locations/fr-inria/networks/1"/>
</nic>
<context>
    <aggregator_ip>172.18.3.144</aggregator_ip>
    <metrics>
        <![CDATA[<metric>users,wc -l /etc/passwd|cut -d" " -f1, rate=20, valuetype=3
                <![CDATA[<metric>system.test, who|wc -l, history=25, valuetype=3 </metric>
        </metrics>
    </context>
    <link href="/locations/fr-inria" rel="location"/>
</compute>
```

The first added metric is called “users” and its command calculates the number of lines in the file “/etc/passwd”. The monitoring information is updated each 20 second, the return values are numeric and the history is assigned to 10 days. The second added metric is called “system.test” and its command calculates the number of users currently on the local system. The return values are numeric and the history is assigned to 25 days. The rate of the updated monitoring information will be set to the default value which is 30 second. Note also the IP of the aggregator to which the agent will report the value of the “users” metric. Note the presence of the aggregator IP is crucial, otherwise the agent installed on this VM will not know where to report the metric values (default is localhost).

In this example, one can leave using CDATA, since the added metric doesn't include any character which is illegal in XML elements. The context part of the OCCI request could look as follow:

```
<context>
    <aggregator_ip>172.18.3.144</aggregator_ip>
    <metrics>
        <metric>users,wc -l /etc/passwd|cut -d" " -f1, rate=20, valuetype=3, history=10
                <metric>system.test, who|wc -l, history=25, valuetype=3 </metric>
    </metrics>
</context>
```

- At the portal, the experimenter will have access to the monitoring aggregator graphical interface. On the Latest Data page of the frontend one can see the last reported values of the monitoring metrics. The hosts are identified based on their host name. Note that beside the explicitly defined metrics, predefined metrics about the system status (e.g. CPU, Memory, Performance etc) appear in the frontend. These metrics values are implicitly provided by the agent.

Adding New Metrics After the Experiment Has Been Deployed

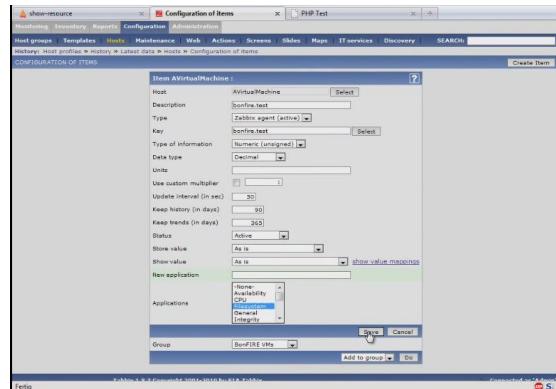
In case the user would like to add new metrics after the experiment has been deployed, two ways are possible, manually or automatically. In the manual case, the following steps should be followed:

- Log in into the specific VM where the metric should be registered.
- Add the metric into the configuration file of the agent /etc/zabbix/zabbix_agentd.conf. Please follow the indications on the documentation page of [Zabbix user arguments](#) on how to define such arguments (metrics). The following screenshot displays an example of how to configure a new metric to the agent. In Figure below add a new metric (UserParameter) to the agent configuration file a new metric called bonfire.test is added and the value of this metric is calculated by counting the number of rows in a file (see the command after the “,”).

```
#####
# Format: UserParameter<key>,<shell command>
# Note that shell command must not return empty string or EOL only
UserParameter=system,test,whoami -l
## Set of parameter for monitoring MySQL server (v3.23.42 and later)
# Note: MySQL 5.6 and add --password if required
UserParameter=mysql.ping,mysqladmin -uroot ping|grep alive|wc -l
UserParameter=mysql.uptime,mysqladmin -uroot status|cut -f2 -d":"|cut -f1 -d":"
UserParameter=mysql.threads,mysqladmin -uroot status|cut -f3 -d":"|cut -f1 -d":"
UserParameter=mysql.questions,mysqladmin -uroot status|cut -f4 -d":"|cut -f1 -d":"
UserParameter=mysql.connections,mysqladmin -uroot status|cut -f5 -d":"|cut -f1 -d":"
UserParameter=mysql.qps,mysqladmin -uroot status|cut -f9 -d":"
UserParameter=mysql.version,mysql -V

# Start Bonfire generated user parameters; please note the changes will be overwritten, in case of rebooting
# End Bonfire generated user parameters, and begin the personalised ones
UserParameter=bonfire.test,/tmp/testFile.txt | cut -d " " -f1
```

- Restart the zabbix agent by running /etc/init.d/zabbix-agent restart.
- The metrics defined at step 1 need to be manually configured also at the server side. The reason behind is that the agent does not provide any functionality to push the new metrics (user parameters) to the server. Figure below depicts how to create a new item. The type of the item needs to be compatible with the agent type. Therefore, one needs to select the type “Zabbix agent (active)”. The properties of the added metric can be further configured through the aggregator GUI, like the update rate, the type of information (which is referring to the value type), history and others.



- The metric’s values can be seen at the aggregator front end available in the portal at Hosts / Latest Data. Please note the metric (item) will appear in this list as soon as values have been provided by the agent. As a consequence

this may take up to 1 minute. Also note there are available also predefined metrics which offer information about the VM status.

In the automatic case, the following steps should be followed:

1. The user has firstly to log in into the specific VM where the metric should be registered.
2. Then execute the script zabbix-add-metric which is located at /usr/local/bin/ directory. The user will be asked to enter the metrics. This script will then configure the agent and the server automatically by conducting the steps 2-4 in the manual case.

```
root@AgentMachine-1259:/usr/local/bin# ./zabbix-add-metric-multiplemetrics
zabbix_context[12488]: AGGREGATOR_IP is 172.18.2.153
-----
Enter your metrics one by one using the following pattern, otherwise, the added metrics will not work:
<key>, command, rate=X, valuetype=Y, history=Z
Where,
key: represents the name of the metric
command: a Linux command that is executed on the agent
rate: the update interval in second
valuetype: the data type of the monitoring values sent to zabbix server (float=0,
string=1, log=2), metric (unsigned integer)=3 and text=4)
history: history of the monitoring information in days
The first two properties (key and command) and their order are mandatory, while the last three properties are optional and their order is not important
EXAMPLE-1: users, wc -l /etc/passwd|cut -d " " -f1, rate=20, valuetype=3, history=80
EXAMPLE-2: testitem, echo testing a new metric, history=45, valuetype=1

Enter the metric you want to add following the pattern described above:
system.test, who|wc -l, rates=20, history=24, valuetype=3
Do you want to add further metrics, yes/no?
YES
Error: you have to correctly enter either 'yes' or 'no'
yes
Enter the metric you want to add following the pattern described above:
testMetric, echo "the last test metric", history=15, rate=30
Do you want to add further metrics, yes/no?
no
no more metric properties
Stopping Zabbix agent: zabbix_agentd.
Starting Zabbix agent: zabbix_agentd.
```

As shown in Figure above, the first two properties (key and command) and their order are mandatory, while the last three properties are optional and their order is not important. If any of the last three optional properties is not used, it will be set to the default value.

One more advantage of using the automatic case is that, if user doesn't have the ability to use the aggregator GUI to create metrics and configure the zabbix server.

8.3.5 How to configure monitoring metrics

BonFIRE monitoring solutions provides its own template that includes many metrics that have default configurations.

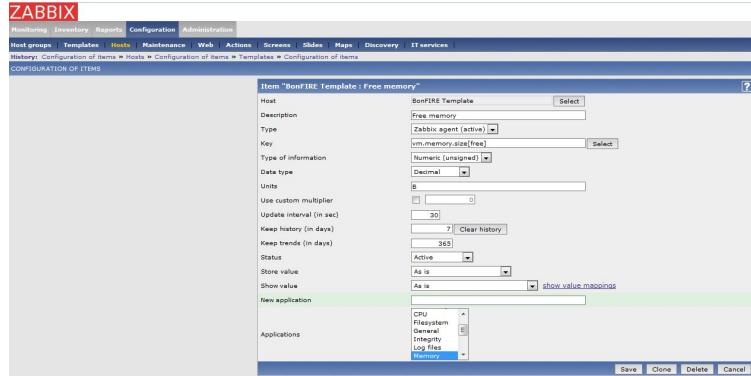
BonFIRE users can reconfigure these metrics as needed. This can be manually done either through *Zabbix API* or easier through the aggregator zabbix GUI by editing metrics (called items in zabbix). Many attributes of a metric can be reconfigured, e.g. update interval (how often the metric is measured), keep history in days, etc.

Creation of a template with own configuration:

You can create your own template including the required metrics with your configurations. This template can be saved and used any time by any VM.

This can be done by creating a new experiment with an aggregator but ALL monitoring services should be disabled (don't let the Aggregator monitor itself, don't add infrastructure metrics, and do not use external storage). This to ensure creating an aggregator without any monitoring data. Then open the aggregator GUI from the portal. Now you can configure any item in the VM template. Or even add new items.

The figure below shows how the Free Memory item for instance can be reconfigured through the aggregator GUI. Under Configuration tab, click on Templates, then click on the items of the “BonFIRE Template”. Now, all metrics are listed, you can edit any metric you would like to reconfigure. For instance, the history attribute can be changed to 2 days to reduce the stored data, then click on save.



Thus, BonFIRE user can reconfigure all or some metrics, and even enable or disable metrics on demand. The figure below shows that the status of the “Buffered Memory” item is disabled. By clicking on “Disabled”, it will be enabled and the status will be changed to “Active”, and vice versa.

Item	Description	Triggers	Rate	Interval	History	Trends	Index	Status	Applications	Funer
vm.memory.size[free]			30	1	300	300	active	Enabled	Memory	green
vm.memory.size[cache]			30	1	300	300	active	Enabled	Memory	green

Once the configuration is done, you can save the aggregator image. Go to the BonFIRE portal, select the created experiment, then click on the aggregator compute resource (usually called BonFIRE-monitor). Then save the image as a storage resource under any name. Note that the image will be really created only after you explicitly shut down the aggregator compute resource. This is done through changing the state “Switch State” after saving the image.

So far, your aggregator image with your configuration is saved and can be used any time.

Creation of an experiment with your preconfiguration aggregator:

Create an experiment without an aggregator either through the portal or BonFIRE API at the OCCI level. Once, the experiment is running, you can now create a new compute resource for the aggregator, this means that your saved image will be used for this compute resource.

In case you wanted to have monitoring data (aggregator database) stored permanently and externally (as datablock), then you have to create a storage resource from type datablock outside the experiment, this to ensure having the data permanently available after the deletion or expiration of the experiment.

Given that an external storage is desired, you have to select your external storage to be added to the aggregator VM as an additional disk, see the figure below.

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Create Compute Resource

Name ?

BonFIRE-monitor

Group ?

<private>

Instance Type ?

medium (cpus: 2, memory: 2048MiB)

Cluster ? (This site does not support advance reservations)

<default>

Host ?

<choose automatically>

VMIImage ?

MyAggregatorImage

Storage ?

alhazmi_Aggregator-Storage-MyExperiment (DATABLOCK)

Extend Root FS ?

Network ?

BonFIRE WAN

Enable Monitoring ?

→ Continue

► Finish

In the following figure you see the XML-based OCCI request to create the aggregator VM. It indicates that the VM will have two disks, one from type OS with the aggregator image, the other one from type datablock with the external storage.

BonFIRE >> Experiments >> Experiment Details >> Create Compute

Create Compute Resource

Review OCCI Request for Compute Creation

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>BonFIRE-monitor</name>
  <instance_type>medium</instance_type>
  <disk>
    <storage href="/locations/uk-epcc/storages/1851"/>
    <type>OS</type>
    <target>hda</target>
  </disk>
  <disk>
    <storage href="/locations/uk-epcc/storages/1900"/>
    <type>DATABLOCK</type>
    <target>hdb</target>
  </disk>
  <nic>
    <network href="/locations/uk-epcc/networks/105"/>
  </nic>
  <context>
    <usage>zabbix-agent;zabbix-aggr-extend;infra-monitoring-init</usage>
    <aggregator_ip>172.18.6.20</aggregator_ip>
  </context>
  <link href="/locations/uk-epcc" rel="location"/>
</compute>
```

Ignore XML Validation Errors

The usage element in the context tag is used for setting the required contextualisation information needed for configuring monitoring services.

In case the aggregator should monitor itself, the usage element should be set to:

<usage>zabbix-agent</usage>

which is used by the portal as a default value for the usage element while creating a new compute resource.

If an external storage is used to store the aggregator database, then the usage element should be set to:

<usage>zabbix-aggr-extend</usage>

If the infrastructure monitoring service is also required in addition to having an external storage, then:

<usage>zabbix-aggr-extend;infra-monitoring-init</usage>

The usage element will be set to the following if the aforementioned three monitoring services are required:

<usage>zabbix-agent;zabbix-aggr-extend;infra-monitoring-init</usage>

None of the three monitoring services will be supported, if you don't add the usage element to the context tag at all.

8.3.6 How to Create Aggregated Metrics

Calculated items are virtual data sources, which are calculated from one or more real items. The resulting calculation of the data is stored in Zabbix as a new item.

Zabbix distinguishes between two types of calculated items: calculated items and aggregated items.

In this How-To we will show how to create an aggregated item in zabbix, which is more suitable than calculated item if we want to aggregate values across a host group.

Aggregated item is mostly useful for clusters where overall state is more important than state of individual machines.

To find out what we can use in such a situation, open zabbix GUI and then go to Configuration | Hosts, select all in the Group dropdown and click on Items next to A Test Host, i.e. the aggregator hostname, which normally called BonFIRE-monitor-XXX

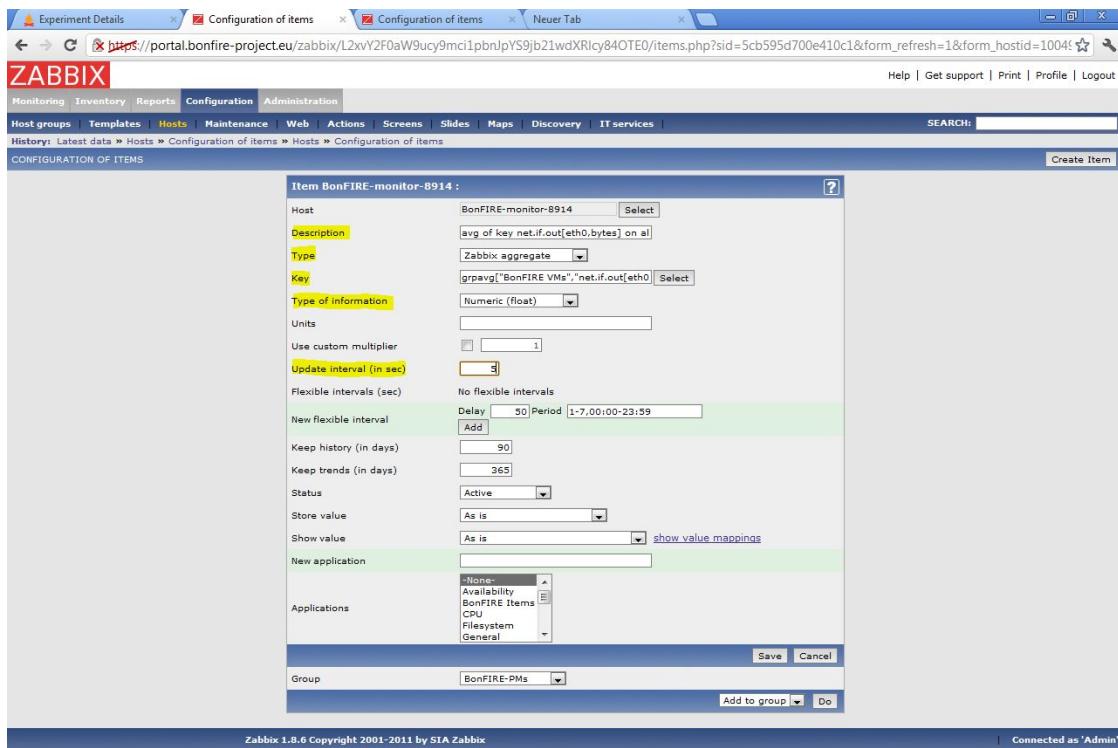
Name	Applications	Items	Triggers	Graphs	DNS	IP	Port	Templates	Status	Availability
BonFIRE-monitor-8914	Applications (13)	Items (104)	Triggers (0)	Graphs (0)	BonFIRE-monitor-8914	127.0.0.1	10050	BonFIRE_Template	Monitored	[] [] []
hirsagent-6053	Applications (13)	Items (104)	Triggers (0)	Graphs (0)	hirsagent-6053	127.0.0.1	10050	BonFIRE_Template	Monitored	[] [] []
node-3.bonfire.grid5000.fr	Applications (0)	Items (14)	Triggers (0)	Graphs (0)	-	0.0.0.0	-	BonFIRE_Template.PM	Monitored	[] [] []

And then click on “Create Item”

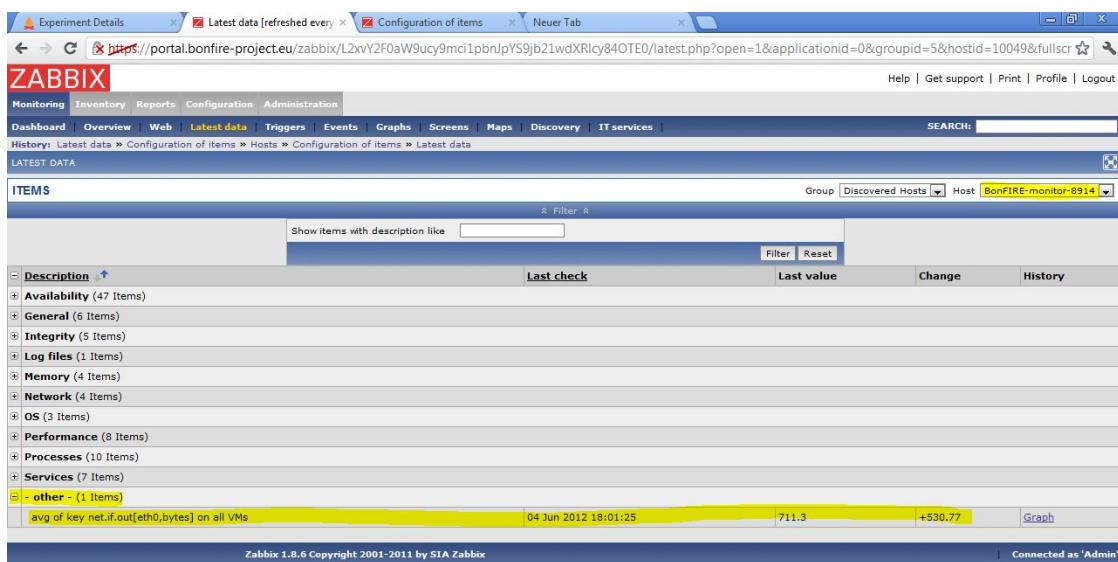
Hosts list	Applications (13)	Triggers (0)	Graphs (0)	Host: BonFIRE-monitor-8914	DNS: BonFIRE-monitor-8914	IP: 127.0.0.1	Port: 10050	Status: Monitored	Availability: Unknown	
Wizard	Description ↑	Triggers	Key	Interval	History	Trends	Type	Status	Applications	Error
	BonFIRE generated: bonfire.count9[/tmp/testFile.txt]		bonfire.count9[/tmp/testFile.txt]	30	7		Zabbix agent (active)	Not supported	BonFIRE Items	X
	BonFIRE generated: bonfire.count10[/tmp/testFile.txt]		bonfire.count10[/tmp/testFile.txt]	30	7		Zabbix agent (active)	Not supported	BonFIRE Items	X
	BonFIRE Template:Buffers memory		vm.memory.size[buffers]	30	7	365	Zabbix agent (active)	Active	Availability	✓
	BonFIRE Template:Cached memory		vm.memory.size[cached]	30	7	365	Zabbix agent (active)	Active	Memory	✓
	BonFIRE Template:Checksum of /user/shin/sehrf1		vfs.file.cksumf[/user/shin/sehrf1]	600	7	365	Zabbix agent (active)	Active	Intensity	✓

Here we are going to create an aggregated item, which calculate the average of Outgoing traffic on interface eth0 in all hosts in the hostgroup “BonFIRE VMs”. Notice: select the same or a longer update interval than in the real items

Description:	avg of key net.if.out[eth0,bytes] on all VMs
Type:	Zabbix aggregate
Key:	grpavg["BonFIRE VMs","net.if.out[eth0,bytes]", "last", "0"]
Type of information:	Numeric (float)
Update interval (in sec)	5



After entering these information click save and then go to Monitoring | Latest data and you will find the latest data of the new aggregated item there



8.3.7 Monitoring Zabbix API

Curl-based examples of using Zabbix JSON RPC

This document includes some examples which are based on cURL to invoke Zabbix API through JSON RPC. Through command-line-based scripts, Zabbix API is invoked to provide resource monitoring information. The output of the executed shell scripts are the monitoring information results as responses to JSON RPC queries. To be noticed that a resource is called “item” by Zabbix API. Before monitoring any resource, user authentication information is required.

To get user authentication information, “`user.authenticate`” method is used in the JSON RPC query. The following shell script is used to get user authentication information.

The username is Admin and the password BONFIRE_EXPERIMENT_AGGREGATOR_PASSWORD is generated by BonFIRE broker one per experiment and to be found on the aggregator machine in the contextualisation file located at /etc/default/bonfire, or from the xml description of the aggregator machine that can be shown through the BonFIRE Portal. Using this credential, you can login to Zabbix Server through zabbix-API.

getUserAuth.sh

```
#!/bin/bash
curl -i -X POST -H 'Content-Type:application/json' -d'{"jsonrpc": "2.0", "method": "user.auth",
```

The response to this query has the following result, which is the output of the executed shell script “getUserAuth.sh”:

```
{"jsonrpc":"2.0", "result":"6705db5c5bdbd1d02a1f53857be6b200", "id":0}
```

This result is used for further JSON RPC queries. To get information about a specific host, in addition to user authentication information, the name of the host should be given in the JSON RPC query. “host.get” method is used to get information about a specific host. Use the parameter “host” to give the host name, in the following example the host name is “Zabbix server”:

getHost.sh

```
#!/bin/bash
curl -i -X POST -H 'Content-Type: application/json' -d '{"jsonrpc":"2.0","method":"host.get"}'
```

After executing the “getHost.sh” script, the response to the JSON RPC query has the following result:

```
{"jsonrpc":"2.0","result":[{"maintainances":[{"maintenanceid":"0"}]},{"hostid":"10017","proxy_
```

This result includes information about the host. The “hostid”:“10017”, is one of the information parameters, which can be used for further request along with user authentication results.

To monitor any resource (item), “item.get” is used as the method of JSON RPC query. The authentication result and the host ID received from the previous JSON RPC queries are used as input to some parameters of the JSON RPC query in the following getItemID.sh script, which is used to get item ID. However, when using JSON RPC to invoke Zabbix API, the parameter “key_” is used for filtering monitoring information. For instance, to get only the monitoring information about the item “Used disk space on /usr”, the “key_” should be set to “vfs.fs.size[/usr,used]”.

getItemID.sh

```
#!/bin/bash
curl -i -X POST -H 'Content-Type: application/json' -d '{"jsonrpc":"2.0","method":"item.get"}'
```

After executing the “getItemID.sh” script, the response to the JSON RPC query has the following result:

```
{"jsonrpc":"2.0", "result":[{"hosts":[{"hostid":"10017"}]}, {"itemid":"18527", "type":0, "snmp_
```

If one needs to know a list of all available items and their corresponding keys before looking for a specific item, one can find this in two ways:

1. On the Zabbix GUI: go to Configuration/Hosts/ then click on the items of the target host, you will find the description of items (e.g. “Used disk space on /usr”) with their corresponding keys (e.g. “vfs.fs.size[/usr,used]”).
 2. If you don’t have the ability to see the GUI, which means that you only use remote calls (JSON-RPC) through Zabbix API using curl, you can execute, for example, the following shell script:

getAvailableItems.sh

```
#!/bin/bash
curl -i -X POST -H 'Content-Type: application/json' -d '{"jsonrpc":"2.0","method":"item.get"}'
```

When you execute this shell script, you will get all items (their descriptions and keys) as a response to the JSON-RPC request.

Since the returned result as a response to the JSON RPC query is too long, the following is the last part of that result:

```
{"jsonrpc": "2.0", "result": [ {".....": "..."}, {"hosts": [{"host": "....", "status": 1, "last_update": "2011-06-26T10:10:00", "last_error": null}], "value_type": "float", "key": "Used disk space on /usr", "description": "Used disk space on /usr", "last_update": "2011-06-26T10:10:00", "last_error": null}], "id": 1}
```

To monitor the history of the any item, the “history.get” method is used in the JSPN RPC queries. If we want to monitor the history of, for instance, the item “Used disk space on /usr”, the JSON RPC query should include user authentication result, host ID and item IDs gained from previous JSON RPC queries. The “getHistory.sh” script can be executed to get history information about the “Used disk space on /usr” within the period of time [26.06.2011/01:00 pm, 26.06.2011/01:10 pm] (Note: this period of time is converted to UNIX timestamps through UNIX timestamp converter). Moreover, the parameter “history” of the “history.get” JSON RPC query should set to the item value type. This value is included in the response to the “item.get” JSON RPC query under the name “value_type”.

In Zabbix, the following value_types are defined:

1. define('ITEM_VALUE_TYPE_FLOAT', 0);
2. define('ITEM_VALUE_TYPE_STR', 1);
3. define('ITEM_VALUE_TYPE_LOG', 2);
4. define('ITEM_VALUE_TYPE_UINT64', 3);
5. define('ITEM_VALUE_TYPE_TEXT', 4);

getHistory.sh

```
#!/bin/bash
curl -i -X POST -H 'Content-Type: application/json' -d '{"jsonrpc": "2.0", "method": "history.get", "params": {"itemids": ["18527"], "start": "1309086017", "end": "1309086107", "history": 1}, "id": 1}' http://zabbix:10050/api_jsonrpc.php
```

The response to “history.get” query after executing “getHistory.sh” has the following result:

```
{"jsonrpc": "2.0", "result": [{"item": {"itemid": "18527", "name": "Used disk space on /usr", "last_update": "2011-06-26T10:10:00", "last_error": null, "value_type": "float"}, "clock": "1309086017", "value": 9675649024}], "id": 1}
```

Important things to be noticed regarding “history.get”:

1. “history.get” functionality is not implemented only since version: 1.8.3. If you use, lower version of Zabbix, like 1.8.2, the response to the “history.get” JSON RPC query will be:

```
{"jsonrpc": "2.0", "error": {"code": -32602, "message": "Invalid params.", "data": "Resource (histo...)"}}
```

2. Fetching history info about multiple metrics is not possible using one JSON RPC query. In Zabbix documentation, it's clear written that “Getting values for multiple items of different types (history parameter) is not supported at this time”.
3. Link to these infos: <http://www.zabbix.com/documentation/1.8/api/history/get>

8.3.8 Estimation of Monitoring Data Storage Size

This documentation helps the experimenter to estimate the required storage size for monitoring data. The required storage size for monitoring data of an experiment depends on the number of monitored resources (VMs), the number of monitored metrics (e.g. CPU load, memory, etc.) in each VM and their configuration, and the lifetime of the whole experiment or the individual resources. BonFIRE monitoring system enables experimenters to store monitoring data in a flexible way. The data storage size can be offered on-demand, and thus, the experimenter can get the desired storage size. Furthermore, depending on the experiment setup, monitoring data is not only available while experimenting but can even be maintained persistent after the deletion or the expiration of the experiment.

BonFIRE monitoring solution used the open source software Zabbix. Its configuration data requires a fixed amount of disk space (normally 10MB) and does not grow much [1]. The collected/reported monitoring data are stored in Zabbix

database. However, it is not easy to have an accurate estimation on the required storage size for Zabbix database due to the fact that the monitored metrics (items as called in Zabbix) are configured differently. The configuration parameters that have an effect on the size of the needed storage are as follow:

Refresh rate: how often a metric (e.g. CPU load) is measured and the obtained value is then stored into the Zabbix server's database. If the refresh rate of the CPU load metric is set to 30 second (default value in Zabbix), the load of the CPU is then measured every 30 second and the obtained value is sent to the server.

Type: represents the data type of the measured value (floats, integers, strings, log files, etc.). The needed disk space for keeping a single value may vary from 40 bytes to hundreds of bytes depending on the data type.

History: representing the period of time (in days) in which the monitoring data of a metric are maintained and available for the experimenter before deletion. Monitoring data of a metric could be maintained for 10 days and 100 days for another one depending on the housekeeper setting for history.

Trends: Zabbix server hourly calculates the (min, avg, and max) of the measured values for each metric using history tables and stores them on a table, called trends. Trends information is stored for a specific period of time depending on the housekeeper setting for trends. Trends are not deleted due to the history housekeeping, i.e. trends remain even if the history parameter is set.

Monitoring a large number of metrics (e.g. 200) configured with different rates (varying from few seconds up to minutes), data types, history and trends setting, demands very long calculations for estimating the needed storage size. However, some experiments, with different number of deployed compute resources (VMs), have been conducted to observe the growth behaviour of Zabbix database size. Around 100 metrics are monitored in each of the deployed VMs. Metrics are configured using Zabbix default setting, i.e. various rates, types, history, trends, etc. These experiments are as follow:

1. An experiment with only one resource is deployed. The observed growth of Zabbix database is as follow:

Clock	Value
08/11/11 10:47 AM	29.56 MB
18/11/11 03:24 PM	223.94 MB
25/11/11 03:24 PM	265.88 MB
02/12/11 03:24 PM	299.44 MB
09/12/11 03:24 PM	332.99 MB
12/12/11 03:24 PM	349.77 MB

2. An experiment with two resources is deployed. The observed growth of Zabbix database is as follow:

Clock	Value
09/11/11 09:43 AM	29.56 MB
18/11/11 03:24 PM	391.71 MB
25/11/11 03:24 PM	475.60 MB
02/12/11 03:24 PM	542.71 MB
09/12/11 03:24 PM	618.20 MB
12/12/11 03:24 PM	651.76 MB

3. An experiment with five resources is deployed. The observed growth of Zabbix database is as follow:

Clock	Value
22/11/11 04:34 PM	29.56 MB
29/11/11 04:34 PM	773.56 MB
05/12/11 16:07 PM	1005,56 MB
06/12/11 11:58 AM	1021.56 MB

However, to give a simplified example on the estimation, let assume having the following scenario. An experiment comprises of three resources (compute, storage or network), and 100 metrics are addressed to be monitored in each resource. All metrics have the same configuration: 30 second refresh-rate, the same number of bytes (e.g. 50 bytes)

required to keep a single value, 50 days history, and no trends are used. Given that the available storage for Zabbix database is only 1GB, after how many days this storage will be full?

Answer: 100 metrics with refresh rate of 30 seconds, the number of values per second is calculated as 100/30. Monitoring data is needed to be kept for 50 days ($50*24*3600$) of history. The total number of values will be around = $100/30*50*24*3600 = 144.000.000$. But the required disk space per value is around 50 bytes. In our example, 14,4M of values will require $14,4M * 50 \text{ bytes} = 720\text{MB}$ of disk space. Based on this result, in addition to the fixed amount of disk space (normally 10MB) that is needed for Zabbix configuration, the dedicated storage 1GB will theoretically never be full and no additional disk space will be required even if the experiment lifetime is unlimited.

But in case 200 metrics are monitored with the same aforementioned configuration, the available storage size will be full after 34,72 days of experimenting, and in this case the housekeeping setting for history in this case will be useless.

$$200/30*\text{days}*24*3600*50\text{B} = 1\text{GB} \Rightarrow \text{days} = 34,72$$

For more details on Zabbix database size, please see the Zabbix documentation (in 1.8 version, 2.6 Database size) [1].

[1] <http://www.zabbix.com/documentation/1.8/manual/installation/requirements>

8.4 Accessing and Downloading Monitoring Data

There are several ways of accessing and downloading data. Assuming you have deployed a Zabbix Aggregator for an experiment, the BonFIRE Portal will display a button for the Zabbix Web GUI where you can access and view the monitoring data. This GUI also allows configuration, if desired. However, the web GUI does not lend itself very well to downloading the monitoring data, which is covered in different ways in the different pages below:

8.4.1 Fetching Metric Values Programmatically

Assuming you have correctly created an experiment with ID 4056 with an aggregator and at least one compute resource (VM), you can now access the Zabbix API of your aggregator at the following URI:

`https://api.bonfire-project.eu/experiments/4056/zabbix`.

On this URI you can send normal JSON-RPC requests, as you would do to interact with the [Zabbix API](#).

For example, here is how you would fetch the values for the metric `vm.memory.size[free]` (free memory) for our `my-vm` compute resource:

- First, you must authenticate against the Zabbix API to receive an authentication token, to be used with further requests.
- Hint: The password is generated by BonFIRE broker one per experiment and to be found on the aggregator machine in the contextualisation file located at `/etc/default/bonfire`, or from the xml description of the aggregator machine that can be shown through the BonFIRE Portal. Using this credential, you can login to Zabbix Server through zabbix-API:

```
$ curl -kni https://api.bonfire-project.eu/experiments/4056/zabbix \
-X POST -H 'Content-Type: application/json' -d '{
    "jsonrpc": "2.0", "auth": null, "id": 1,
    "method": "user.authenticate",
    "params": {"user": "Admin", "password": "BONFIRE_EXPERIMENT.Aggregator_PASSWORD"}
}'

HTTP/1.1 200 OK
Date: Wed, 26 Oct 2011 14:18:11 GMT
Server: Apache/2.2.16 (Debian)
X_POWERED_BY: PHP/5.3.3-7+squeeze1
```

```

SET_COOKIE: zbx_sessionid=a072e27410fa5c7524285f04610c513c
CONTENT_LENGTH: 68
CONTENT_TYPE: application/json
Via: 1.1 bonfire-api (bonfire-api/0.10.1)
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.465800
Content-Length: 68
Vary: Authorization,Accept
Connection: close
Content-Type: text/plain; charset=UTF-8

{"jsonrpc":"2.0","result":"a072e27410fa5c7524285f04610c513c","id":1}

```

- Then, use the returned token to ask for the internal ID of our metric, with the item.get command:

```

$ curl -kni https://api.bonfire-project.eu/experiments/4056/zabbix \
-X POST -H 'Content-Type: application/json' -d '{
    "jsonrpc": "2.0", "auth": "a072e27410fa5c7524285f04610c513c", "id": 2,
    "method": "item.get",
    "params": {"filter": {"host": ["my-vm-3584"], "key_": "vm.memory.size[free]"}, "output": "ex"
}'

```

HTTP/1.1 200 OK

Date: Wed, 26 Oct 2011 14:21:49 GMT

Server: Apache/2.2.16 (Debian)

X_POWERED_BY: PHP/5.3.3-7+squeeze1

CONTENT_LENGTH: 796

CONTENT_TYPE: application/json

Via: 1.1 bonfire-api (bonfire-api/0.10.1)

X-UA-Compatible: IE=Edge,chrome=1

X-Runtime: 0.4000773

Content-Length: 796

Vary: Authorization,Accept

Connection: close

Content-Type: text/plain; charset=UTF-8

```

{
    "jsonrpc": "2.0",
    "result": [
        {
            "hosts": [
                {
                    "hostid": "10050"
                }
            ],
            "itemid": "22569",
            "type": "7",
            "snmp_community": "",
            "snmp_oid": "",
            "snmp_port": "161",
            "hostid": "10050",
            "description": "Free memory",
            "key_": "vm.memory.size[free]",
            "delay": "30",
            "history": "7",
            "trends": "365",
            "lastvalue": "208412672",
            "lastclock": "1319638881",
            "prevvalue": "208326656",
            "status": "0",
            "value_type": "3",

```

```
"trapper_hosts": "",  
"units": "B",  
"multiplier": "0",  
"delta": "0",  
"prevorgvalue": null,  
"snmpv3_securityname": "",  
"snmpv3_securitylevel": "0",  
"snmpv3_authpassphrase": "",  
"snmpv3_privpassphrase": "",  
"formula": "0",  
"error": "",  
"lastlogsize": "0",  
"logtimefmt": "",  
"templateid": "22261",  
"valuemapid": "0",  
"delay_flex": "",  
"params": "",  
"ipmi_sensor": "",  
"data_type": "0",  
"authtype": "0",  
"username": "",  
"password": "",  
"publickey": "",  
"privatekey": "",  
"mtime": "0"  
}],  
"id": 2  
}
```

- We had to make this request to fetch the `itemid` (here: "22569") corresponding to our metric. Now we can ask for the 5 latest metric values with the `history.get` command:

```
$ curl -kni https://api.bonfire-project.eu/experiments/4056/zabbix \
-X POST -H 'Content-Type: application/json' -d '{
    "jsonrpc": "2.0", "auth": "a072e27410fa5c7524285f04610c513c", "id": 3,
    "method": "history.get",
    "params": {"itemids": ["22569"], "output": "extend", "limit": 5}
}'

HTTP/1.1 200 OK
Date: Wed, 26 Oct 2011 14:26:00 GMT
Server: Apache/2.2.16 (Debian)
X_POWERED_BY: PHP/5.3.3-7+squeeze1
TRANSFER_ENCODING: chunked
CONTENT_TYPE: application/json
Via: 1.1 bonfire-api (bonfire-api/0.10.1)
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.441692
Content-Length: 9455
Vary: Authorization,Accept
Connection: close
Content-Type: text/plain; charset=UTF-8

{
    "jsonrpc": "2.0",
    "result": [
        {
            "itemid": "22569",
            "clock": "1319634468",
            "value": 100
        }
    ]
}
```

```
        "value": "209043456"
    },
    {
        "itemid": "22569",
        "clock": "1319634498",
        "value": "209043456"
    },
    {
        "itemid": "22569",
        "clock": "1319634529",
        "value": "208949248"
    },
    {
        "itemid": "22569",
        "clock": "1319634559",
        "value": "208973824"
    },
    {
        "itemid": "22569",
        "clock": "1319634589",
        "value": "208949248"
    }],
    "id": 3
}
```

As you can see, you get back a list of timestamped values, which can be easily processed.

See <http://www.zabbix.com/documentation/1.8/api/history/get> for the list of all options available for the history.get command.

8.4.2 Exporting Monitoring Data to CSV

This section describes how to get monitoring data from Zabbix server through the command line and export these to a CSV formatted file.

The python script “fetch_items_to_csv.py” is used for this purpose and you can find it for download here: [fetch_items_to_csv.py](#)

Prerequisites

Before using this script you have to install first the python extension `python-argparse`:

```
apt-get install python-argparse
```

The “fetch_items_to_csv.py” script depends also on the python library for Zabbix API. Download from here: [zabbix_api.py](#)

Usage

Before running these scripts make sure to make them executable:

```
chmod +x fetch_items_to_csv.py
chmod +x zabbix_api.py
```

After that just run the script “fetch_items_to_csv.py” with the `-h` option to see how it should be used:

```
./fetch_items_to_csv.py -h
usage: fetch_items_to_csv.py [-h] -s SERVER_IP -n HOSTNAME -k KEY
                             [-u USERNAME] [-p PASSWORD] [-o OUTPUT]
                             [-t1 DATETIME1] [-t2 DATETIME2] [-v DEBUGLEVEL]

Fetch history from aggregator and save it into CSV file

optional arguments:
  -h, --help            show this help message and exit
  -s SERVER_IP          aggregator IP address
  -n HOSTNAME           name of the monitored host
  -k KEY                zabbix item key, if not specified the script will fetch all
                        keys for the specified hostname
  -u USERNAME           zabbix username, default Admin
  -p PASSWORD           zabbix password
  -o OUTPUT              output file name, default hostname.csv
  -t1 DATETIME1         begin date-time, use this pattern '2011-11-08 14:49:43' if
                        only t1 specified then time period will be t1-now
  -t2 DATETIME2         end date-time, use this pattern '2011-11-08 14:49:43'
  -v DEBUGLEVEL         log level, default 0
```

A more detailed description of the aforementioned arguments is as follow:

SERVER_IP: the IP address of the aggregator that includes the monitoring data.

HOSTNAME: is referring to any monitored host, both virtual and physical ones. BonFIRE monitoring offers VM-level monitoring as well as physical infrastructure-level monitoring.

KEY: in Zabbix, the key refers to the description of the item. The key of the monitored items are to be found on the aggregator GUI, by just clicking on the items button. If not specified the script will fetch all keys for the specified hostname

DATETIME1 & DATETIME2 represent the time period in which the monitoring history is fetched.

DATETIME1 represents the begin time (date and clock as shown in the pattern) from which the monitoring data are fetched till:

DATETIME2 which represents the end time, or now if the **DATETIME2** is not used.

Example

The monitoring history of the item “incoming traffic on interface eth0” on the monitored host is fetched. The key of this item is “net.if.in[eth0,bytes]”. The monitored host in this example has the name “BonFIRE-monitor-5539”. To fetch the monitoring data, since a specific time (e.g. November the 1st 2011 at 10:30 AM) until now, then run:

```
./fetch_items_to_csv.py -s 127.0.0.1 -n BonFIRE-monitor-5539 -k net.if.in[eth0,bytes] -t1 '
```

Output:

```
3681 records has been fetched and saved into: BonFIRE-monitor-5539.csv
```

If you need the monitoring history for a specific time period e.g. between November the 1st 2011 at 10:30 AM and November the 13th 2011 at 04:00 PM then run:

```
./fetch_items_to_csv.py -s 127.0.0.1 -n BonFIRE-monitor-5539 -k net.if.in[eth0,bytes] -t1 '
```

Output:

```
2718 records has been fetched and saved into: BonFIRE-monitor-5539.csv
```

If you need the monitoring history for all keys(items) then don't specify the -k option and the script will fetch the monitoring history for all keys(items) for the specified host

```
./fetch_items_to_csv.py -s 127.0.0.1 -n vmhost1 -t1 "2012-06-08 16:00:00" -t2 "2012-06-08 1
```

Output:

```
552 records has been fetched and saved into: vmhost1.csv
```

8.4.3 Getting Monitoring Data After Experiment Lifetime

BonFIRE monitoring System enables experimenters to store monitoring data in a flexible way. The monitoring data can be stored in an external storage, and thus, the data can be maintained persistent after the deletion or the expiration of the experiment. The whole aggregator database including the monitoring data is stored in the external storage. To get the data, the experimenter can either create a new experiment and reuse this external storage as an additional external disk attached to the aggregator of the new experiment, see the [Managing Data Storage For Monitoring](#) page or use any Zabbix server and replace its database with the one stored in the external storage.

ELASTICITY

9.1 BonFIRE EaaS - Elasticity as a Service

The elasticity as a service is a feature that allows experimenters to run an elastic experiment. Elasticity means that is possible to dynamically increase or decrease the number of computing resources to cope with different situation of load.

The experimenter only need to prepare an image that contains his service and configure the Elasticity Engine in order to use this disk during an elastic experiment. The elasticity engine will use this disk to create the virtual machine that will serve users requests. In order to distribute the load among those virtual machine there are two different kind of load balancers: HAProxy and Kamailio (using the dispatcher module). Those are enough for HTTP and SIP application stateless load balancing.

The EaaS in BonFIRE uses the monitoring aggregator for retrieving information regarding the load of the virtual machines. It interoperates with the broker for dynamically adding and removing compute resources based on some “rules” specified by the user. For more information about how to do this, please see the [How To Use the BonFIRE EaaS](#) page.

9.2 How To Use the BonFIRE EaaS

9.2.1 How to create an elastic group

For creating an elastic group is basically needed to configure some parameters using the contextualization of the Elasticity Engine virtual machine. Those parameter are necessary for providing some basic information to the Elasticity Engine that will be used during the execution. This is possible in two different ways: using the portal or the API.

9.2.2 Configuring the EaaS using the API

For using the elasticity as a service you need to create an experiment with an aggregator. Before you start an elastic group is important that the aggregator is up and running. In order to create an elastic service you should prepare the image that contains the service itself. To do this you can read this documentation [Datablock OCCI How-To](#). it is important to understand that the image that you will prepare during that process is the base image that will be used every time is necessary to create a new virtual machine to scale up. This image will be cloned every time your upscale threshold will be exceeded and a new VM will be created using that disk as a source. After you create your disk image containing the preconfigured service, you need to start configuring some properties for the EaaS. There are different kind of configuration parameters: some are necessary for the Group of VMs, other for specifying the rules.

First of all you need to specify the minimum number of virtual machines that you want will be part of your elastic group. Then you have also to put a maximum number of instances to create during the execution. Furthermore you have also to specify a name that will be assigned to the VMs part of your group:

```
<ELASTICITY_VMGROUP_MIN>1</ELASTICITY_VMGROUP_MIN>
<ELASTICITY_VMGROUP_MAX>5</ELASTICITY_VMGROUP_MAX>
<ELASTICITY_VMGROUP_NAME>elastic_service</ELASTICITY_VMGROUP_NAME>
```

Then you need to specify the Type of the instance that you would like to use and the disk source of your prepared image. You need to specify the locations where you want to deploy your VMs. Supported locations are HLRS, INRIA, and EPCC. Basically the elasticity engine will choose randomly the location where to deploy the necessary VMs :

```
<ELASTICITY_INSTANCETYPE>lite</ELASTICITY_INSTANCETYPE>
<ELASTICITY_DISKSOURCE>disk_name</ELASTICITY_DISKSOURCE>
<ELASTICITY_LOCATIONS>fr-inria;uk-epcc;de-hlrs</ELASTICITY_LOCATIONS>
<ELASTICITY_NETWORK>BonFIRE WAN</ELASTICITY_NETWORK>
```

In case you need to have a custom instance Type you can use the following tags:

```
<ELASTICITY_INSTANCETYPE>custom</ELASTICITY_INSTANCETYPE>
<ELASTICITY_MEMORY>1024</ELASTICITY_MEMORY>
<ELASTICITY_CPU>0.5</ELASTICITY_CPU>
<ELASTICITY_VCPU>1</ELASTICITY_VCPU>
```

For additional context parameters the following tag should be added:

```
<ELASTICITY_CONTEXT>key1=value1;key2=value2</ELASTICITY_CONTEXT>
```

If you want you can also use the reservation on INRIA for reserving a cluster of machines and run your experiment in a specific cluster. In this case is necessary to insert the cluster_uuid and if you want you can also specify the hostname where to deploy your virtual machines created dynamically by the elasticity engine.

```
<ELASTICITY_CLUSTER>cluster_uuid</ELASTICITY_CLUSTER>
<ELASTICITY_HOST>hostname</HOSTNAME>
```

9.2.3 Configuring the trigger

Before explaining you the way to configure the trigger I will introduce the upscale/downscale mechanism. Think about a group of virtual machines that are serving user requests distributed among them by a load balancer with a weight based round robin algorithm. Their load, in terms of CPU or MEMORY, should be the same if the load balancer does well his job. You need only to introduce the upscale threshold and for doing this you can use a standard zabbix expression:

```
{<key>.<function>(<parameter>)}<operator><constant>
```

For example if you want that the Elasticity Engine starts another instance after the cpu average of your group overcomes the 60% you should write an expression like this:

```
{system.cpu.usage.last(0)}>70
```

It's highly recommended to use this item for the CPU threshold, since it will get the cpu usage of the whole system. This is the context parameter that you need to insert during the creation of the compute resource of the Elasticity Engine.

```
<ELASTICITY_TRIGGER_UPSCALE_EXPRESSION>{system.cpu.usage.last(0)}>70</ELASTICITY_TRIGGER_UPSCALE_EXPRESSION>
```

The utilized down-scaling algorithm is based on the assumption that a VM can be removed from the group, only if its removal does not bring other VMs into a trigger state. So you don't need to specify any kind of trigger for downscaling if you introduce an upscale expression.

9.2.4 Getting a load balancer

If you create an elastic experiment you might want to create a load balancer in order to distribute the load among your instances. For doing this, you need to specify three parameters:

```
<ELASTICITY_LB_SCHEME>HAProxy</ELASTICITY_LB_SCHEME>
<ELASTICITY_LB_PORT>80</ELASTICITY_LB_PORT>
<ELASTICITY_LB_LOCATION>/locations/fr-inria</ELASTICITY_LB_LOCATION>
```

Basically you can specify two different kind of load balancers: “HAProxy” or “kamailio”. The first attribute will allow the creation of an HAProxy instance. HAProxy is a layer 4/7 load balancer. Kamailio is a SIP load balancer. Both of them are configured for stateless services. The elasticity engine automatically will add/remove instances to the load balancer. You need to specify the PORT number where your service is reachable and the location where you want to deploy this load balancer.

9.2.5 Complete OCCI request

Here you can find an example of a complete OCCI request that could be used for creating an elastic group. Basically you need to create a compute resource using a standard Debian 2G image (so you have to specify the STORAGE_ID and NETWORK_ID) and then include the context variables in the request:

```
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>BonFIRE-Elasticity-Engine-elenService</name>
    <instance_type>lite</instance_type>
    <cluster>uuid</cluster>
    <disk>
        <storage href="STORAGE_ID"/>
        <type>OS</type>
        <target>hda</target>
    </disk>
    <nic>
        <network href="NETWORK_ID"/>
    </nic>
    <context>
        <usage>zabbix-agent</usage>
        <aggregator_ip>aggregator_ip</aggregator_ip>
        <ELASTICITY_TRIGGER_UPSCALE_EXPRESSION>{system.cpu.usage.last(0)}>70</ELASTICITY_TRIGGER_UPSCALE_EXPRESSION>
        <ELASTICITY_VMGROUP_MAX>5</ELASTICITY_VMGROUP_MAX>
        <AGGREGATOR_USER>Admin</AGGREGATOR_USER>
        <AGGREGATOR_PASSWD>zabbix</AGGREGATOR_PASSWD>
        <ELASTICITY_INSTANCETYPE>lite</ELASTICITY_INSTANCETYPE>
        <ELASTICITY_DISKSOURCE>disk_name</ELASTICITY_DISKSOURCE>
        <ELASTICITY_LOCATIONS>fr-inria;uk-epcc;de-hlrs</ELASTICITY_LOCATIONS>
        <ELASTICITY_NETWORK>BonFIRE WAN</ELASTICITY_NETWORK>
        <ELASTICITY_LB_SCHEME>HAProxy</ELASTICITY_LB_SCHEME>
        <ELASTICITY_VMGROUP_MIN>1</ELASTICITY_VMGROUP_MIN>
        <ELASTICITY_LB_PORT>80</ELASTICITY_LB_PORT>
        <ELASTICITY_LB_LOCATION>/locations/fr-inria</ELASTICITY_LB_LOCATION>
        <ELASTICITY_VMGROUP_NAME>eaas</ELASTICITY_VMGROUP_NAME>
        <usage>elasticity-engine</usage>
        <ELASTICITY_CLUSTER>uuid</ELASTICITY_CLUSTER>
    </context>
    <link href="/locations/fr-inria" rel="location"/>
</compute>
```

9.2.6 Configuring the EaaS using the Portal

For creating an elastic experiment through the portal, you need to create a new experiment and select the check box for enabling elasticity capable aggregator.

When you have an experiment with a running aggregator, you need to create an elastic group configuring all the parameters necessary.

If you want to use the reservation offered by INRIA, you can do it by clicking on `continue` instead of `finish` and adding those parameters into the context section:

```
<ELASTICITY_CLUSTER>uuid</ELASTICITY_CLUSTER>
<ELASTICITY_HOST>host</ELASTICITY_HOST>
```

After you create your experiment you should see the service endpoint in the elastic group section.

EXPERIMENT SERVICES

10.1 Notifications

10.1.1 Notification Service

BonFIRE offers a way for clients to subscribe to notifications sent when the experiment state changes, and when resources are created, updated, or destroyed.

For example, your compute resources may be interested to know when the experiment state transition into the stopped state (see [Experiment Lifecycle](#)), so that they can start a backup process or some custom shutdown procedure before the experiment ends.

The way BonFIRE does this is to publish state transition events into a message queue (we're using [RabbitMQ](#)), to which you (meaning: some script written by you) can subscribe.

Event format

All messages that you will receive are [JSON](#), formatted in either a simple or more detailed structure:

Simple messages:

timestamp	the UNIX timestamp at which the event occurred
type	type of the resource (experiment, compute, network, storage)
status	the updated status of the target resource
path	the path to the resource in the BonFIRE API (e.g. /experiments/1234)

Detailed messages:

timestamp	the UNIX timestamp at which the event occurred
objectType	type of the resource (experiment, compute, network, storage, site_link)
eventType	the updated status of the target resource (create, state, update, delete). See State section
objectId	the path to the resource in the BonFIRE API (e.g. /experiments/1234)
source	module or site from which the event is sent (e.g. res-mng, be-ibbt)
groupId	group that triggers the event notification
userId	user that triggers the event notification
objectData	extra information to recreate the entire object. See objectData section

Check the [Event structure](#) section for further details and sample messages.

Prerequisites

You should find a library that talks the AMQP protocol, for your language of choice. Here are a few suggestions:

- Ruby – [bunny](#) or [amqp](#)
- Java – [RabbitMQ Java client](#)
- Python – [pika](#)

You'll also need to know the following connection parameters:

host	mq.bonfire-project.eu
port	5672
username	eventReader
password	reader1348
vhost	bonfire
exchange	experiments
routing_key	experiment specific

Note that the guest user is only allowed to create server-generated queues (i.e. queues with empty names in most of the AMQP libraries).

Your queue must bind to the `experiments` exchange and must specify a `routing_key` specific to your experiment. You may check [Using the RabbitMQ Notification Service](#) for connection examples on various languages.

10.1.2 Event structure

Almost every CRUD operation (`create`, `state`, `update`, `delete`) on resources triggers different types of events on the Experiment Message Queue. The format of the events on the EMQ vary: there are both *simple* and more *detailed* messages.

Important: EMQ does not currently listen to the resource events generated from the *Resource Manager* (`res-mng`), so you'll obtain that information directly from the sites these resources belong to. As a side effect, the events whose status is `delete` may not exist in the queue and you **should not rely on them** when developing any script that reads from the queue.

Simple messages

Correspond to those events directly sent from the *Resource Manager*. These contain a minimum subset of information, which are in fact *mapped* to some properties from the [Detailed messages](#).

Sample event

```
{  
    "timestamp":1374243305  
    "type":"compute",  
    "status":"created",  
    "path":"/locations/uk-epcc/computes/1263",  
}
```

Fields

The following fields are mandatory for every message:

timestamp Same as `timestamp`. However, the time varies because it is crafted and sent at a different time, from a different origin.

type Same as *objectType*.

status Almost similar to *eventType*. However, it may vary slightly.

path Same as *objectId*.

Detailed messages

Correspond to those events that are forwarded from the *sites*, the *enactor adapters* and also from the *Resource Manager*. These contain a more extended subset of information compared to the *Simple messages*.

Sample event

```
{
    "timestamp":1373881015,
    "eventType":"create",
    "objectType":"experiment",
    "objectId":"/experiments/257",
    "source":"res-mng",
    "groupId":"crohr",
    "userId":"crohr",
    "objectData":{
        "name":"test",
        "duration":7200
    }
}
```

Fields

The following fields are mandatory for every message (except for *objectData*, which applies only to `create` and `update` events):

timestamp Epoch timestamp.

eventType Indicates the operation that fired the event. Can be one of the following:

- `create`: indicates that an object is created. The definition of that object MUST be available in the *objectData* (more information in *objectData* section)
- `delete`: indicates that an object is removed from the BonFIRE runtime
- `update`: indicates that a particular object changes. Again, the new definition of that object MUST be available in the *objectData* field
- `state.{generic_state} (".testbed_specific_state")`: indicates that an object changes its state. More on this in *State*

State The “state” event defines a change in the object’s state. Evidently, the states depend on the *objectType*. Since BonFIRE does not want to intervene (much) in the lifecycle management of the resources on the different testbeds, it can happen that for similar object states each testbed has its own representation. This is not very workable when we want to use the [topic exchange mechanism](#) for easy event filtering. That is why we should have a very small set of generic object states. Because we do not want to loose the detailed (and in most cases, more fine grained) state information of the individual testbeds, the *eventType* of a “state” event consists of 3 parts:

```
"state."{generic_state} ("." "testbed_specific_state")
```

- state: to denote it is a state event
- generic_state: the options depend on the *objectType*. Every partner should feel comfortable adding interesting states or change states and exposing these generic states
 - experiment
 - * pending: a resource’s description has reached the site, but it still needs to be deployed
 - * reserved: a resource is reserved from the moment it is taking away resources from other users
Read: when do you want the experimenter to start paying for the resource
 - * active: a resource is active when all of its functionality (also BonFIRE specific functionality) can be used by the user
 - * stopping
 - * stopped: a resource is stoped when it stops using physical resources
 - * deleting
 - * deleted: a resource is deleted when all references to the resource are removed from the site.
 - * failed
 - Federica router
 - * pending: a router is in pending state when the slice is still not ready. It is different from a reserved state since there is no resource booking
 - * active: a router resource when is ready to be used by the user
 - * deleted: the router is deleted from the experiment. The router must be in PENDING state to perfom this operation
 - * failed: the router creation failed
 - Federica network
 - * pending: a Federica network is in pending state when the slice is still not ready. It is different from a reserved state since there is no resource booking
 - * active: a Federica network resource when is ready to be used by the user.
 - * deleted: the Federica network is deleted from the experiment. The Federica network must be in PENDING state to perfom this operation
 - * failed: the Federica network failed during its creation
 - Autobahn site_link
 - * pending
 - * active
 - * deleted

- * failed
- testbed_specific_state: a number of specific states, separated by a “.” (period). For example, disabling a Managed Network at the VirtualWall causes the following state event: *state.active.down*

objectType The type of the object this event applies to. E.g. *experiment, compute, network, router, network_link, reservation*, etc.

objectId Full object ID. E.g. *experiments/345, /locations/uk-epcc/computes/453*. For objects that have a location it will be represented in this object ID so there is no need for a locations field.

source The unique identification of the entity that fired this event.

userId The user who caused this event, e.g. in the example: the user that created the experiment.

groupId The group on behalf of which the user triggered the event.

objectData **Note:** Only mandatory for create and update events.

This field contains a *sub-message* inside. It should hold sufficient information to be able to recreate the entire object. However, *sufficient* does not mean all the information one would get by doing a GET on a resource. For example, information to identify a session, like sequence numbers, session ids etc, should not be in the objectData.

Follows a sample for each kind of resource.

Reservation

```
"objectData": {
  "name": "myReservation",
  "duration": 120,
  "allocationUnits": 23
}
```

A brief explanation about some fields:

- duration: in minutes
- allocationUnits: total number of allocation units in the reservation. Consumers may multiply this by the duration to get allocation unit hours.

OCCI

```
"objectData": {
  "relatedId": "/experiments/1234",
  "httpRequestLine": "GET /experiments/1234 HTTP/1.1",
  "headers": { "X-BONFIRE-ASSERTED-ID": "gvseghbr", ... },
  "occi": "...",
  "tool": "..."
}
```

A brief explanation about some fields:

- relatedId: full reference to the object
- occi: complete XML message

- tool: first element of the USER-AGENT header

Experiment

```
"objectData": {  
    "name": "My Experiment",  
    "duration": 120,  
    "reservationId" : "/reservations/345"  
}
```

A brief explanation about some fields:

- reservationId: optional reservation id if experiment has an associated reservation

Managed experiment Any of the following:

```
"objectData": {  
    "name": "My Experiment",  
    "experimentId": "/experiments/657",  
    "type": "JSON",  
    "descriptor": {...}  
}  
  
"objectData": {  
    "name": "My Experiment",  
    "experimentId": "/experiments/657",  
    "type": "OVF",  
    "descriptor": "..."  
}
```

A brief explanation about some fields:

- descriptor: full JSON descriptor (1st message), full OVF descriptor in XML (2nd message)

Storage

```
"objectData": {  
    "name": "storage-name",  
    "type": "DATABLOCK",  
    "persistent": false  
    "size": 1024,  
    "fstype": "ext3",  
    "experimentId": "/experiments/657"  
}
```

Network

```
"objectData": {  
    "name": "network-name",  
    "address": "192.168.0.0".  
    "size": "C",  
    "type": "active"  
    "lossrate": 0.1,  
    "latency": 10,  
    "bandwidth": 100,  
    "protocol": "TCP",  
    "packetsize": 50,  
    "throughput": 30,
```

```

    "experimentId": "/experiments/657"
}

```

Compute

```

"objectData": {
    "name": "compute-name",
    "instanceType": "/locations/fr-inria/instance_type/medium",
    "cpu": 1,
    "vcpu": 4,
    "memory": 1024,
    "cluster": null,
    "host": "node-1.integration.bonfire.grid5000.fr",
    "osimage": "/locations/fr-inria/storages/571",
    "disks": [
        {"id": "0",
            "storage": "/locations/fr-inria/storages/571",
            "type": "FILE",
            "target": "sda"}, ...
    ],
    "networks": [
        {"network": "/locations/uk-epcc/networks/0",
            "ip": "172.18.3.57",
            "mac": "02:00:ac:12:03:39"}, ...
    ],
    "experimentId": "/experiments/657",
    "bestEffortAllocation" : true,
    "allocationUnits" : 4
}

```

A brief explanation about some fields:

- `bestEffortAllocation`: optional (only present in *res-mng* events). Defaults to *false*. Is this a best effort allocation (rather than reservation)?
- `allocationUnits`: optional (only present in *res-mng* events). Must be present if `bestEffortAllocation` is *true*.

Federica Resources

Router

```

"objectData": {
    "name": "RouterName",
    "host": "HostName",
    "interfaces" : [
        { "name" : "if_a1",
            "physicalInterface" : "ge-0/2/2",
            "ip" : "192.168.1.2",
            "netmask" : "255.255.255.0"
        },
        { "name" : "if_a2",
            "physicalInterface" : "ge-0/1/0",
            "ip" : "192.168.2.1",
            "netmask" : "255.255.255.0"
        },
        ...
    ]
}

```

```
],
"experimentId": "/experiment/1"
"config": "myRouterConfig"
}
```

Network

```
"objectData": {
  "name" : "myFedericaNetwork",
  "vlan": "network vlan id"
  "networkLinks" : [
    { "src_router" : "RouterA",
      "src_interface" : "if_a1",
      "dst_router" : "RouterB",
      "dst_interface" : "if_b1"
    },
    { "src_router" : "RouterB",
      "src_interface" : "if_b2",
      "dst_router" : "RouterC",
      "dst_interface" : "if_c1"
    },
    ...
  ],
  "experimentId": "/experiment/1"
}
```

Autobahn resources

Site_links

```
"objectData": {
  "name": "site-link-name",
  "description": "site-link-description",
  "end_point_A": "/locations/uk-epcc",
  "vlan_A": "2001",
  "end_point_Z": "/locations/pl-psnc",
  "vlan_Z": "2002",
  "bandwidth": "1000000",
  "experimentId": "/experiments/657"
}
```

Amazon resources BonFIRE only allows to create compute, network, and storage resources on Amazon. These resources use the same objectData scheme as already presented, with the location being the only difference.

Virtual Wall Resources The Virtual Wall does not have special resources in BonFIRE. The special thing the Virtual-Wall will do (soon) is launching a create event for computes on behalf of the user, when the experiment uses Managed Networks. This is done in order to correctly define the number of computes being used by the experimenter.

10.1.3 Using the RabbitMQ Notification Service

Getting the code

It will be explained here the procedure to bind or subscribe to a queue and which kind of messages you may expect from some operations on the experiment.

Ruby

Install the following Ruby gems:

```
$ gem install bunny
```

Then, copy the following script into a `subscribe-mq.rb` file (source can be downloaded from `subscribe-mq.rb`):

```
require 'rubygems'
require 'bunny'

ENV['BONFIRE_AMQP_USER'] ||= 'eventReader'
ENV['BONFIRE_AMQP_PASS'] ||= 'reader1348'
ENV['BONFIRE_AMQP_HOST'] ||= 'mq.bonfire-project.eu'
ENV['BONFIRE_AMQP_PORT'] ||= '5672'

routing_key = ARGV[0]

# Create a new AMQP connection:
bunny = Bunny.new(
  :user => ENV['BONFIRE_AMQP_USER'],
  :pass => ENV['BONFIRE_AMQP_PASS'],
  :host => ENV['BONFIRE_AMQP_HOST'],
  :port => ENV['BONFIRE_AMQP_PORT'],
  :vhost => 'bonfire'
)

# Start the channel:
bunny.start

# Create a new queue, with a server-generated name:
queue = bunny.queue('')

# Bind the queue to the ''experiments'' direct exchange:
bind = queue.bind('experiments', :routing_key => routing_key)
puts "Bind result: #{bind}"

# Subscribe to the queue (blocking call, set a timeout > 0 if you want to
# listen for a limited amount of time):
queue.subscribe(:timeout => 0) do |msg|
  puts "Received message: #{msg[:payload]}"
end

bunny.stop
```

Java

First you will need to download the needed libraries from the [RabbitMQ Java client](#) web page.

```
$ wget http://www.rabbitmq.com/releases/rabbitmq-java-client/v2.7.0/rabbitmq-java-client-bin-2.7.0.t
```

Unzip the libraries where you want to keep them. You need the path where the libraries are stored when the client is compiled and executed.

Then, copy the following script into a `SubscribeMQ.java` file (source can be downloaded from `SubscribeMQ.java`):

```
import java.io.File;
import java.io.FileInputStream;

import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.QueueingConsumer;
import com.rabbitmq.client.AMQP.Queue.BindOk;

public class SubscribeMQ {

    private static final String EXCHANGE_NAME = "experiments";
    private static final String HOST = "mq.bonfire-project.eu";
    private static final String VIRTUAL_HOST = "bonfire";
    private static final String USER_NAME = "eventReader";
    private static final String PASSWORD = "reader1348";

    public static void main(String[] argv) throws Exception {
        if (argv.length < 1){
            System.err.println("Please, introduce the routing key");
            System.exit(1);
        }

        // Getting the routing key
        String routingID = argv[0];

        // Create the connection:
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(HOST);
        factory.setVirtualHost(VIRTUAL_HOST);
        factory.setUsername(USER_NAME);
        factory.setPassword(PASSWORD);
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        // Get the name of the queue in order to bind the channel to it.
        String queueName = channel.queueDeclare().getQueue();
        BindOk bound = channel.queueBind(queueName, EXCHANGE_NAME, routingID);

        System.out.println(" [*] Waiting for messages...");

        QueueingConsumer consumer = new QueueingConsumer(channel);
        channel.basicConsume(queueName, true, consumer);

        // We will listen to the channel indefinitely. The while could be changed if desired.
        while (true) {
            QueueingConsumer.Delivery delivery = consumer.nextDelivery();
            String message = new String(delivery.getBody());
            String routingKey = delivery.getEnvelope().getRoutingKey();
            System.out.println(" [*] Received message: " + message);
        }
    }
}
```

```

        }
    }
}
```

For compiling the client, it is needed to reference to the rabbitmq-client packet:

```
$ javac -cp rabbitmq-java-client-bin-2.7.0/rabbitmq-client.jar SubscribeMQ.java
```

This client will accept the routing key as input. The routing key is an alphanumeric string, related to your experiment and returned when it is created.

Python

Copy the following script into a `subscribe-mq.py` file (source can be downloaded from `subscribe-mq.py`):

```

import pika
import sys

host = 'mq.bonfire-project.eu'
port = '5672'
vhost = 'bonfire'
exchange = 'experiments'
user = 'eventReader'
password = 'reader1348'

if len(sys.argv) < 1:
    print 'Please add the experiment_routing_key(s) !'
    sys.exit(2)

routing_keys = sys.argv[1:]

# Create a new AMQP connection:
credentials = pika.PlainCredentials(user, password)
connection = pika.BlockingConnection(pika.ConnectionParameters(
    host=host, virtual_host=vhost, credentials=credentials))

# Create channel and queue
channel = connection.channel()

result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue

# Bind the queue to the different routing_keys
for experiment_routing_key in routing_keys:
    channel.queue_bind(exchange=exchange,
                       queue=queue_name,
                       routing_key=experiment_routing_key)

print ' [*] Waiting for messages. To exit press CTRL+C'

def callback(ch, method, properties, body):
    print " [x] %r:%r" % (method.routing_key, body,)

channel.basic_consume(callback,
                      queue=queue_name,
                      no_ack=True)
```

```
channel.start_consuming()
```

For more information, you can read the [RabbitMQ tutorial](#).

Getting queue notifications

In order to see the notifications you may follow these steps. First of all, the client must bind to a queue. Later on, an experiment will be created, updated and finally deleted so you can check on the queue the different notifications or events triggered by those actions.

Subscribing to a queue

Create first a new experiment. This will be done in a terminal window by using CURL:

```
$ curl -kni https://api.bonfire-project.eu/experiments \
-H 'Content-Type: application/vnd.bonfire+xml' -X POST -d \
'<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>Message queue experiment</name>
    <description>Experiment description</description>
    <walltime>7200</walltime>
</experiment>'

HTTP/1.1 201 Created
Date: Thu, 27 Oct 2011 13:54:27 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Location: https://api.bonfire-project.eu:443/experiments/82
Content-Type: application/vnd.bonfire+xml; charset=utf-8
Cache-Control: no-cache
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.060483
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked

<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/82" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <id>82</id>
    <description>Experiment description</description>
    <name>Message queue experiment</name>
    <walltime>7200</walltime>
    <user_id>crohr</user_id>
    <status>ready</status>
    <routing_key>e82</routing_key>
    <created_at>2011-10-27T13:54:27Z</created_at>
    <updated_at>2011-10-27T13:54:27Z</updated_at>
    <networks>
    </networks>
    <computes>
    </computes>
    <storages>
    </storages>
    <link rel="parent" href="/" />
    <link rel="storages" href="/experiments/82/storages"/>
    <link rel="networks" href="/experiments/82/networks"/>
```

```
<link rel="computes" href="/experiments/82/computes"/>
</experiment>
```

You can see here in the response the routing key that was assigned to your experiment (e82 in this case). Note that you should not make any assumption regarding the format of this key: it should be considered as an opaque string.

Now start the Ruby, Java or Python process in a terminal window with the routing key as the first argument, so it can bind to the appropriate queue:

Ruby

```
$ ruby subscribe-mq.rb e82
```

If the connection to the message queue is successful, you should see the following result:

```
$ ruby subscribe-mq.rb e82
Bind result: bind_ok
```

And the process should be still running, waiting for a message to appear in the message queue.

Java It is needed to specify where are the libraries used in the code when you run the client. For that, you can create a variable pointing to the libraries downloaded before and use it as a classpath:

```
$ export CP=.:./root/rabbitmq-java-client-bin-2.7.0/commons-cli-1.1.jar:/root/rabbitmq-java-client-bin
$ java -cp $CP SubscribeMQ e82
$ [*] Waiting for messages...
```

The client will be listening the queue indefinitely, waiting for messages.

Python

```
$ python subscribe-mq.py e82
```

If the connection to the message queue is successful, you should see the following result:

```
$ python subscribe-mq.py e82
[*] Waiting for messages. To exit press CTRL+C
```

The client will be listening the queue indefinitely, waiting for messages.

Listening to queue notifications

Keep the previous terminal open and do open a new one for the next step. Update now the experiment status to running:

```
$ curl -kni https://api.bonfire-project.eu/experiments/82 \
-H 'Content-Type: application/vnd.bonfire+xml' -X PUT -d \
'<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <status>running</status>
</experiment>'
```

```
HTTP/1.1 200 OK
Date: Thu, 27 Oct 2011 14:05:39 GMT
Server: thin 1.2.11 codename Bat-Shit Crazy
Content-Type: application/vnd.bonfire+xml; charset=utf-8
ETag: "5680a774cb03cc4aa1729dded2be5a25"
```

```
Cache-Control: max-age=0, private, must-revalidate
X-UA-Compatible: IE=Edge,chrome=1
X-Runtime: 0.057566
Vary: Authorization,Accept
Connection: close
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8"?>
<experiment href="/experiments/82" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <id>82</id>
  <description>Experiment description</description>
  <name>Message queue experiment</name>
  <walltime>7200</walltime>
  <user_id>crohr</user_id>
  <status>ready</status>
  <routing_key>e82</routing_key>
  <created_at>2011-10-27T13:54:27Z</created_at>
  <updated_at>2011-10-27T14:05:39Z</updated_at>
  <networks>
  </networks>
  <computes>
  </computes>
  <storages>
  </storages>
  <link rel="parent" href="/" />
  <link rel="storages" href="/experiments/82/storages"/>
  <link rel="networks" href="/experiments/82/networks"/>
  <link rel="computes" href="/experiments/82/computes"/>
</experiment>
```

After some time, the experiment status will transition into the running state, and your listening client process should receive something similar to:

```
{"type": "experiment", "status": "running", "path": "/experiments/82", "timestamp": 1382290322}

{"timestamp": 1382290323, "eventType": "state.running", "objectType": "experiment", "objectId": "/experiment/82"}
```

If you change the experiment status to stopped:

```
$ curl -kni https://api.bonfire-project.eu/experiments/82 \
-H 'Content-Type: application/vnd.bonfire+xml' -X PUT -d \
'<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <status>stopped</status>
</experiment>'
```

Then the client process should receive something similar to:

```
{"type": "experiment", "status": "stopped", "path": "/experiments/82", "timestamp": 1382290338}

{"timestamp": 1382290398, "eventType": "state.stopped", "objectType": "experiment", "objectId": "/experiment/82"}
```

Now delete the experiment:

```
$ curl -kni https://api.bonfire-project.eu/experiments/82 -X DELETE
```

And you should receive something similar to:

```
{
  "type": "experiment",
  "status": "terminated",
  "path": "/experiments/82",
  "timestamp": 1382290691
}

{
  "timestamp": 1382290691,
  "eventType": "delete",
  "objectType": "experiment",
  "objectId": "/experiments/82",
  "version": 1
}

{
  "timestamp": 1382290698,
  "eventType": "state.terminated",
  "objectType": "experiment",
  "objectId": "/experiments/82",
  "version": 2
}
```

You can see that the payload format is **JSON**. Additional properties may be added in the future. You can change the client in order to act as you want when the message is received.

10.2 COCOMA

10.2.1 Controlled Contentious and Malicious framework

The aim of COCOMA framework is to create, monitor and control contentious and malicious system workload. By using this framework experimenters are able to create operational conditions for their System under Test (SuT) under which tests and experiments can be carried out. This allows more insight into the testing process of the SuT so that various scenarios of the cloud infrastructure behaviour can be analysed by collecting and correlate metrics of the emulated environment with the test results.

A common approach is to setup COCOMA within a VM and deploy it in a landscape. As this VM is intended to run mainly COCOMA resources patterns, other processes in the VM should have almost no impact on the desired patterns.

10.2.2 Contents

Introduction

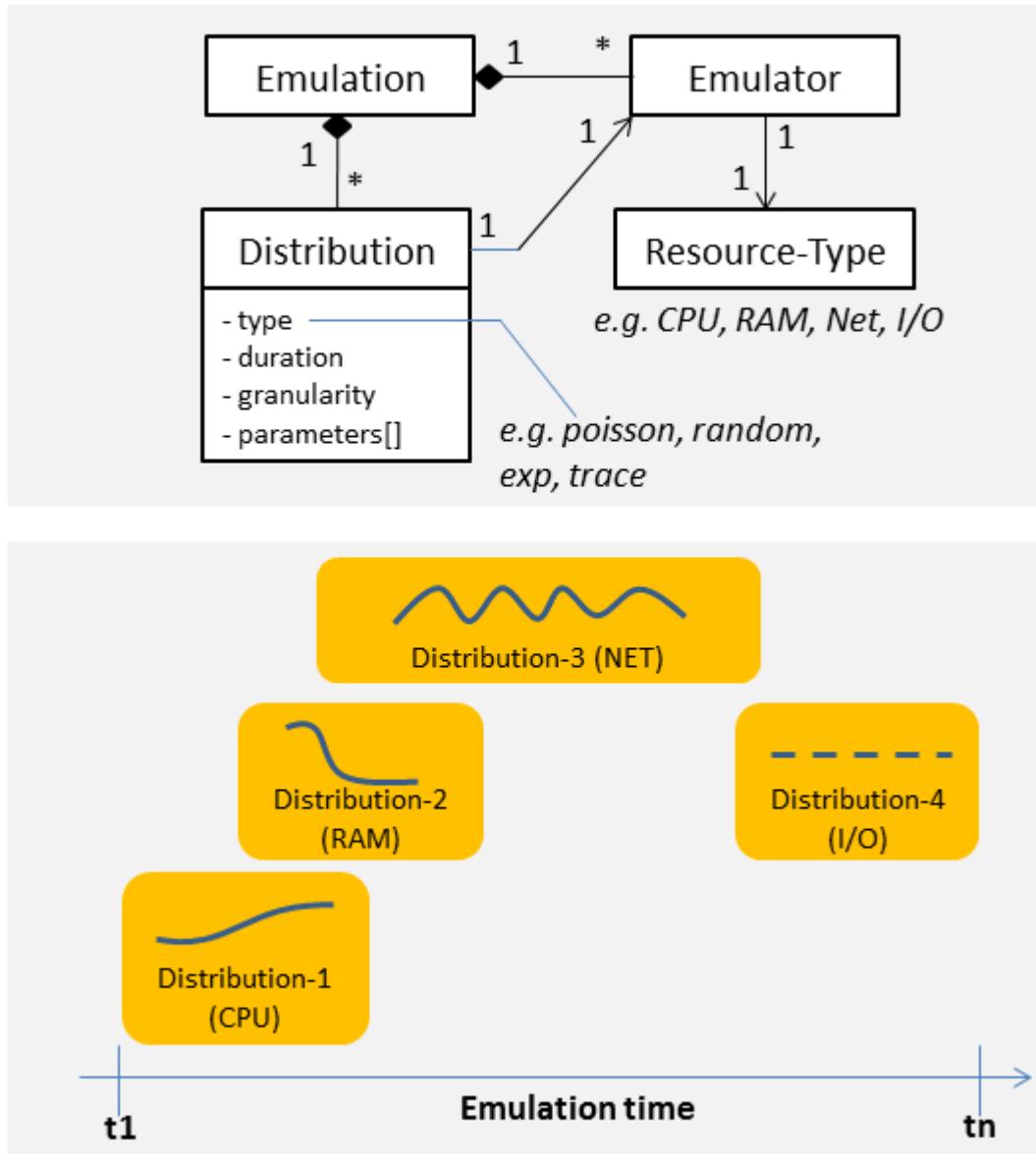
One of the main common characteristics of cloud computing is resource sharing amongst multiple users, through which providers can optimise utilization and efficiency of their system.

However, at the same time this raises some concerns for performance predictability, reliability and security:

- Resource (i.e. CPU, storage and network) sharing inevitably creates contention, which affects applications' performance and reliability
- Workloads and applications of different users residing on the same physical machine, storage and network are more vulnerable to malicious attacks

Studying the effect of resource contention and maliciousness in a cloud environment can be of interest for different stakeholders. Experimenters may want to evaluate the performance and security mechanisms of their system under test (SuT). On the other hand cloud providers may want to assess their mechanisms to enforce performance isolation and security.

In order to use COCOMA, an experimenter defines an *emulation* which embeds all environment operational conditions as shown in the figure below. The actual operational conditions are defined in what are called *distributions*, which create specific workloads over the targeted resource of a specific resource type. For example, *distribution 1* targets the CPU creating an exponential trend over a specific time range within the whole emulation. Each distribution time is divided into multiple time-slots based on the distribution granularity then broken down into multiple runs each one injecting a different load level per time slot, which depends on the discrete function of the distribution.



Getting Started

COCOMA is installed in “`/usr/share/pyshared/cocoma`”. In this section we provide information about the components that have to be running in order to fully use the framework, and how can a user interact with it.

Starting Components

The two main components of COCOMA are:

- Scheduler daemon (mandatory, needs to be the first started)
- API Daemon (optional, if the REST API functionality is required)

The **Scheduler daemon** - runs in the background and executes workload with differential parameters at the time defined in the emulation properties. To check if the scheduler is running use the following:

```
$ ccmsh --show scheduler
```

To start the scheduler on a specific network interface and port, use the command:

```
$ ccmsh --start scheduler eth1 55555
```

If the network interface and port are omitted, the default values are respectively `eth0` and `51889`

If more detailed output information is needed, the *Scheduler* can also be started in *DEBUG* mode:

```
$ ccmsh --start scheduler eth1 55555 debug
```

The **API daemon** - provides the RESTfull web API, which exposes COCOMA resources to be used over the network. It follows the same command structure as the Scheduler. To check if the API daemon is running use the following:

```
$ ccmsh --show api
```

To start the api on a specific network interface and port, use the command:

```
$ ccmsh --start api eth2 77777
```

If the network interface and port are omitted, the default values are respectively `eth0` and `5050`

The log level is the same specified for the *Scheduler*.

CLI

The COCOMA CLI is called `ccmsh`, and provides the following options:

-h, -help

Display help information of the available options

-v, -version

Display installed version information of COCOMA

-q, -mq

add configuration parameters for message queue: enabled vhost exchange user password host topic

-m, -rmq

remove configuration parameters for message queue

-a, -enl

enable configuration parameters for message queue

-s, -smq
show configuration parameters for message queue

-b, -bfz
Update>Show location of Backfuzz emulator (stored in database)

-l, -list <emulation name>
Display list of all emulations that are scheduled or already finished. If emulation name is provided then it lists information for that specific emulation

-r, -results <emulation name>
Display list of results of all emulations that are scheduled or already finished. If emulation name is provided, then it lists information for that specific emulation

-j, -list-jobs
Queries the scheduler for the list of jobs that are scheduled to be executed and are currently executing. For each one, it gives the job name, the planned execution time, and the duration (when available)

-i, -dist <distribution name>
Scans the “*/usr/share/pyshared/cocoma/distributions*” folder and displays all available distribution modules. If a distribution name is provided, then it shows the help information for that specific distribution

-e, -emu <emulator name>
Scans the “*/usr/share/pyshared/cocoma/emulators*” folder and displays all available emulator wrapper modules. If an emulator name is provided, then it shows the help information for that specific emulator wrapper

-x, -xml <file name>
It create and emulation based on the XML provided

-n, -now (used with -x option only)
Override any start date in the local XML emulation file without modifying the file, i.e. `ccmsh -x <file name> -n`

-d, -delete <emulation name>
Deletes a specific emulation from the database

-p, -purge
Remove all DB entries and scheduled jobs

-c, -clear-logs
Remove all log files

-start <api interface port>, <scheduler interface port>
Start Scheduler or API daemon by specifying network interface and port number i.e. `ccmsh --start api eth0 2020` or `ccmsh --start scheduler eth0 3030`. By default if the network interface is not specified, the Scheduler daemon will run on *eth0* and *51889*, and the API daemon will run on *eth0* and *5050*.

-stop <api>, <scheduler>
Stop Scheduler or API daemon

-show <api>, <scheduler>
Show Scheduler or API daemon processes and their PID numbers

Most of these options are used by themselves, with the exception of ‘-x’ (-xml) which can be used in conjunction with ‘-f’ (-force) and/or ‘-n’ (-now)

Logging

Event Logging COCOMA provides two levels of event logging, *INFO* which is set by default, and *DEBUG* which can be enabled if more detailed feedback is required:

```
$ ccmsh --start scheduler eth1 5555 debug
```

Timestamped events are then saved in the logs/COCOMAlogfile.csv file.

Both *Scheduler* and *API* also save their CLI output in the logs/COCOMAlogfile_Scheduler_sout.txt and logs/COCOMAlogfile_API_sout.txt accordingly. All above files can be retrieved as ZIP archive through COCOMA web API URI:

```
* /logs/system
```

Resource Consumption Logging Optionally the CPU, Memory and Network consumption of COCOMA VM can be recorded during each emulation. To enable resource consumption logging just add this XML code block to the payload:

```
<log>
  <!-- Use value "1" to enable logging(by default logging is off) -->
  <enable>1</enable>
  <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
  <frequency>1</frequency>
  <!-- Set the level of information detail -->
  <logLevel>debug</logLevel>
</log>
```

If the logging was enabled, once the emulation is finished, COCOMA will produce two files named after the emulation (e.g. logs/4-CPU_EMU-res_2013-04-09T18:51:09.csv and logs/4-CPU_EMU-config_2013-04-09T18:51:09.xml). The file with CSV extention will contain the system resource data, and the file with XML extention will have the exact copy of the payload used for that emulation.

Both files can be retrieved as ZIP archive through COCOMA web API URI by emulation name:

```
* /logs/emulations
* /logs/emulations/{name}
```

REST API

Index The API URIs summary list is as follow:

```
* /
* /emulations
* /emulations/{name}
* /distributions
* /distributions/{name}
* /emulators
* /emulators/{name}
* /jobs
* /results
* /results/{name}
* /tests
* /tests/{name}
* /logs
* /logs/system
* /logs/emulations
* /logs/emulations/{name}
```

Description http:method:: GET /

The **root** returns a *collection* of all the available resources. Example of a XML response:

```
<?xml version="1.0" ?>
<root href="/">
    <version>0.1.1</version>
    <timestamp>1365518303.44</timestamp>
    <link href="/emulations" rel="emulations" type="application/vnd.bonfire+xml"/>
    <link href="/emulators" rel="emulators" type="application/vnd.bonfire+xml"/>
    <link href="/distributions" rel="distributions" type="application/vnd.bonfire+xml"/>
    <link href="/tests" rel="tests" type="application/vnd.bonfire+xml"/>
    <link href="/results" rel="results" type="application/vnd.bonfire+xml"/>
    <link href="/logs" rel="logs" type="application/vnd.bonfire+xml"/>
</root>
```

http:method:: GET /emulations

The **emulations** returns a *collection* of all the available emulation resources. Example of a XML response:

```
<?xml version="1.0" ?>
<collection href="/emulations" xmlns="http://127.0.0.1/cocoma">
    <items offset="0" total="3">
        <emulation href="/emulations/1-Emu-CPU-RAM-IO" id="1" name="1-Emu-CPU-RAM-IO" state="inac">
            <emulation href="/emulations/2-CPU_EMU" id="2" name="2-CPU_EMU" state="inac">
                <emulation href="/emulations/3-CPU_EMU" id="3" name="3-CPU_EMU" state="inac">
            </emulations>
            <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
        </items>
    </collection>
```

http:method:: GET /emulations/{name}

Displays information about emulation by name. The returned 200-OK XML is:

```
<?xml version="1.0" ?>
<emulation href="/emulations/1-Emu-CPU-RAM-IO" xmlns="http://127.0.0.1/cocoma">
    <id>1</id>
    <emulationName>1-Emu-CPU-RAM-IO</emulationName>
    <emulationType>mix</emulationType>
    <resourceType>mix</resourceType>
    <emuStartTime>2013-04-09T13:00:01</emuStartTime>
    <emuStopTime>180</emuStopTime>
    <scheduledJobs>
        <jobseempty>No jobs are scheduled</jobseempty>
    </scheduledJobs>
    <distributions ID="1" name="Distro1">
        <startTime>5</startTime>
        <granularity>3</granularity>
        <duration>30</duration>
        <startload>10</startload>
        <stopload>90</stopload>
    </distributions>
    <distributions ID="2" name="Distro2">
        <startTime>5</startTime>
        <granularity>3</granularity>
        <duration>30</duration>
        <startload>10</startload>
        <stopload>90</stopload>
    </distributions>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
```

```

        <link href="/emulations" rel="parent" type="application/vnd.bonfire+xml"/>
</emulation>
```

The returned *404 – Not Found* XML is:

```
<error>Emulation Name: 1-Emu-CPU-RAM-IO1 not found. Error:too many values to unpack</error>
```

http:method:: POST /emulations

```
:param string XML: Emulation parameters defined via XML as shown in the examples section.
```

The returned *201-Created* XML:

```

<?xml version="1.0" ?>
<emulation href="/emulations/4-CPU_EMU" xmlns="http://127.0.0.1/cocoma">
    <ID>4-CPU_EMU</ID>
    <EmuNotes>OK</EmuNotes>
    <DistroNotes>OK</DistroNotes>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
    <link href="/emulations" rel="parent" type="application/vnd.bonfire+xml"/>
</emulation>
```

The returned *400 – Bad Request* XML:

```
<?xml version="1.0" ?>
<error>XML is not well formed Error: syntax error: line 1, column 0</error>
```

http:method:: GET /emulators

Displays emulators list. The returned *200- OK* XML:

```

<?xml version="1.0" ?>
<collection href="/emulators" xmlns="http://127.0.0.1/cocoma">
    <items offset="0" total="3">
        <emulator href="/emulators/lookbusy" name="lookbusy"/>
        <emulator href="/emulators/stressapptest" name="stressapptest"/>
        <emulator href="/emulators/iperf" name="iperf"/>
    </items>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /emulators/{name}

```
:arg name: Name of emulator that you want to get more info
```

Displays information about emulator by name. It returns parameters that can be used with emulator and the values limits (where applicable). The returned *200- OK* XML:

```

<?xml version="1.0" ?>
<emulator href="/emulators/lookbusy" xmlns="http://127.0.0.1/cocoma">
    <info>
        <help>
            Emulator lookbusy can be used for following resources:
            1) Loads CPU with parameters:
                ncpus - Number of CPUs to keep busy (default: autodetected)

            2) Loads Memory (MEM) with parameters:
                memSleep - Time to sleep between iterations, in usec (default 1000)

            3) Changing size of files to use during IO with parameters:
                ioBlockSize - Size of blocks to use for I/O in MB

```

ioSleep - Time to sleep between iterations, in msec (default 100)

XML block example:

```
&lt;emulator-params&gt;
    &lt;resourceType&gt;CPU&lt;/resourceType&gt;
    &lt;ncpus&gt;0&lt;/ncpus&gt;
&lt;/emulator-params&gt;

</help>
<resources>
    <cpu>
        <ncpus>
            <upperBound>100</upperBound>
            <lowerBound>100</lowerBound>
        </ncpus>
    </cpu>
    <io>
        <iосleep>
            <upperBound>999999999</upperBound>
            <lowerBound>999999999</lowerBound>
        </iосleep>
        <ioblocksize>
            <upperBound>9999999</upperBound>
            <lowerBound>9999999</lowerBound>
        </ioblocksize>
    </io>
    <mem>
        <memsleep>
            <upperBound>999999999</upperBound>
            <lowerBound>999999999</lowerBound>
        </memsleep>
    </mem>
</resources>
</info>
<link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
<link href="/emulators" rel="parent" type="application/vnd.bonfire+xml"/>
</emulator>
```

http:method:: GET /distributions

Displays distributions list. The returned 200- OK XML:

```
<?xml version="1.0" ?>
<collection href="/distributions" xmlns="http://127.0.0.1/cocoma">
    <items offset="0" total="3">
        <distribution href="/distributions/linear" name="linear"/>
        <distribution href="/distributions/linear_incr" name="linear_incr"/>
        <distribution href="/distributions/trapezoidal" name="trapezoidal"/>
    </items>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /distributions/{name}

:arg name: Name of distributions that you want to get more info

Displays information about distributions by name. It returns parameters that can be used with distributions and the values limits(where applicable). The returned 200- OK XML:

```

<?xml version="1.0" ?>
<distribution href="/distributions/linear_incr" xmlns="http://127.0.0.1/cocoma">
  <info>
    <help>Linear Increase distribution takes in start and stop load (plus malloclimit for M
    increasing resource workload by spawning jobs in parallel. Can be used with MEM,IO,NET
    <resources>
      <mem>
        <startload>
          <upperBound>3895</upperBound>
          <lowerBound>3895</lowerBound>
        </startload>
        <stopload>
          <upperBound>3895</upperBound>
          <lowerBound>3895</lowerBound>
        </stopload>
        <malloclimit>
          <upperBound>4095</upperBound>
          <lowerBound>4095</lowerBound>
        </malloclimit>
      </mem>
      <io>
        <startload>
          <upperBound>999999</upperBound>
          <lowerBound>999999</lowerBound>
        </startload>
        <stopload>
          <upperBound>999999</upperBound>
          <lowerBound>999999</lowerBound>
        </stopload>
      </io>
      <net>
        <startload>
          <upperBound>1000000</upperBound>
          <lowerBound>1000000</lowerBound>
        </startload>
        <stopload>
          <upperBound>1000000</upperBound>
          <lowerBound>1000000</lowerBound>
        </stopload>
      </net>
    </resources>
  </info>
  <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
  <link href="/distributions" rel="parent" type="application/vnd.bonfire+xml"/>
</distribution>

```

http:method:: GET /tests

Displays tests list. The returned 200- OK XML:

```

<?xml version="1.0" ?>
<collection href="/tests" xmlns="http://127.0.0.1/cocoma">
  <items offset="0" total="20">
    <test href="/tests/01-CPU-Linear-Lookbusy_10-95.xml" name="01-CPU-Linear-L
    <test href="/tests/03-NET-Linear_incr-Iperf-100-1000.xml" name="03-NET-Line
    <test href="/tests/02-IO-Linear-Stressapptest_1-10.xml" name="02-IO-Lin
    <test href="/tests/02-IO-Linear_incr-Stressapptest_1-10.xml" name="02-IO-Li
    <test href="/tests/02-MEM-Linear_incr-Stressapptest_100-1000.xml" name="02-
    <test href="/tests/01-CPU-Trapezoidal-Lookbusy_10-95.xml" name="01-CPU-Trap

```

```

<test href="/tests/01-IO-Trapezoidal-Lookbusy_1-10.xml" name="01-IO-Trapezo
<test href="/tests/01-NET_TEST.xml" name="01-NET_TEST.xml"/>
<test href="/tests/03-MEM-500-1000MB-overlap.xml" name="03-MEM-500-1000MB-o
<test href="/tests/01-CPU-Linear_incr-Lookbusy_10-95.xml" name="01-CPU-Line
<test href="/tests/01-IO-Linear_incr-Lookbusy_1-10.xml" name="01-IO-Linear_
<test href="/tests/02-IO-Trapezoidal-Stressapptest_1-10.xml" name="02-IO-Tr
<test href="/tests/03-CPU-opposite.xml" name="03-CPU-opposite.xml"/>
<test href="/tests/01-MEM-Linear_incr-Lookbusy_100-1000.xml" name="01-MEM-L
<test href="/tests/03-MEM-500-1000MB.xml" name="03-MEM-500-1000MB.xml"/>
<test href="/tests/03-MEM-Linear-Stressapptest_500-1000MB.xml" name="03-MEM
<test href="/tests/01-MEM-Trapezoidal-Lookbusy_100-1000.xml" name="01-MEM-T
<test href="/tests/02-MEM-Trapezoidal-Stressapptest_100-1000.xml" name="02-
<test href="/tests/03-NET-Trapezoidal-Iperf-100-1000.xml" name="03-NET-Trap
<test href="/tests/01-IO-Linear-Lookbusy_1-10.xml" name="01-IO-Linear-Lookb

</items>
<link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /tests/{name}

:arg name: Name of tests that you want to get more info

Displays Content of XML file.

http:method:: POST /tests

:param string: name of the test that is located on COCOMA machine

Create emulation from available tests. The returned 201- *Created XML*:

```

<?xml version="1.0" ?>
<test href="/tests/5-CPU_EMU" xmlns="http://127.0.0.1/cocoma">
    <emulationName>5-CPU_EMU</emulationName>
    <startTime>2013-04-09T18:57:32</startTime>
    <durationSec>60</durationSec>
</test>
```

The returned 400- *Not Found* reply XML:

```

<?xml version="1.0" ?>
<error>error message</error>
```

http:method:: GET /results

Displays results list. The returned 200- *OK XML*:

```

<?xml version="1.0" ?>
<collection href="/results" xmlns="http://127.0.0.1/cocoma">
    <items offset="0" total="5">
        <results failedRuns="0" href="/results/1-Emu-CPU-RAM-IO" name="1-Emu-CPU-RA
        <results failedRuns="0" href="/results/2-CPU_EMU" name="2-CPU_EMU" state="i
        <results failedRuns="0" href="/results/3-CPU_EMU" name="3-CPU_EMU" state="i
        <results failedRuns="0" href="/results/4-CPU_EMU" name="4-CPU_EMU" state="i
        <results failedRuns="0" href="/results/5-CPU_EMU" name="5-CPU_EMU" state="i

    </items>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /results/{name}

:arg name: Name of tests that you want to get more info

Displays information about results by name. The returned 200- *OK* XML:

```
<?xml version="1.0" ?>
<results href="/results/1-Emu-CPU-RAM-IO" xmlns="http://127.0.0.1/cocoma">
    <emulationName>1-Emu-CPU-RAM-IO</emulationName>
    <totalRuns>6</totalRuns>
    <executedRuns>6</executedRuns>
    <failedRuns>0</failedRuns>
    <emuState>inactive</emuState>
</results>
```

http:method:: GET /jobs

```
<?xml version="1.0" ?>
<collection href="/jobs" xmlns="http://127.0.0.1/cocoma">
    <items offset="0" total="3">
        <Job>Job: 1-CPU_EMU-Emulation (trigger: date[2013-12-18 10:48:27], next run at: 2013-12-18 10:48:29.000000)
        <Job>Job: {"Duration": 10.0, "EndTime": "2013-12-18 10:48:29.000000", "StressValue": 100}
        <currentlyRunningJob>Job: 1-CPU_EMU-1-0-cpu_distro { startTime: 2013-12-18 10:48:09.000000}
    </items>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /logs

Displays logs list. The returned 200- *OK* XML:

```
<?xml version="1.0" ?>
<logs href="/logs">
    <link href="/logs/emulations" rel="emulations" type="application/vnd.bonfire+xml"/>
    <link href="/logs/system" rel="system" type="application/vnd.bonfire+xml"/>
</logs>
```

http:method:: GET /logs/system

Return Zip file with system logs.

http:method:: GET /logs/emulations

Displays emulations logs list. The returned 200- *OK* XML:

```
<?xml version="1.0" ?>
<collection href="/logs/emulations" xmlns="http://127.0.0.1/cocoma">
    <items offset="0" total="3">
        <emulationLog href="/logs/emulations/3-CPU_EMU" name="3-CPU_EMU"/>
        <emulationLog href="/logs/emulations/5-CPU_EMU" name="5-CPU_EMU"/>
        <emulationLog href="/logs/emulations/4-CPU_EMU" name="4-CPU_EMU"/>
    </items>
    <link href="/" rel="parent" type="application/vnd.bonfire+xml"/>
    <link href="/logs" rel="parent" type="application/vnd.bonfire+xml"/>
</collection>
```

http:method:: GET /logs/{name}

:arg name: Name of emulation logs that you want to get

Return Zip file with emulation logs.

XML payload structure

A COCOMA emulation is specified in XML. The user can directly create the XML and send it to COCOMA through the CLI client or any REST client. An Emulation must contain all the necessary information about starting time, duration, target resource and required resource usage. Once the XML document is received by COCOMA, the framework automatically schedules and executes the required workload on the chosen resource(s), CPU, IO, Memory or Network.

Consider this sample XML document code:

```
1 <emulation>
2   <emuname>CPU_EMU</emuname>
3   <emuType>Mix</emuType>
4   <emuresourceType>CPU</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>60</emustopTime>
9
10  <distributions>
11
12    <name>CPU_Distro</name>
13    <startTime>0</startTime>
14    <!--duration in seconds -->
15    <duration>60</duration>
16    <granularity>20</granularity>
17    <!--Minimum time a job can last (seconds)-->
18    <minJobTime>2</minJobTime>
19    <distribution href="/distributions/linear" name="linear" />
20    <!--cpu utilization distribution range-->
21    <startLoad>10</startLoad>
22    <stopLoad>95</stopLoad>
23
24    <emulator href="/emulators/lookbusy" name="lookbusy" />
25    <emulator-params>
26      <!--more parameters will be added -->
27      <resourceType>CPU</resourceType>
28      <!--Number of CPUs to keep busy (default: autodetected)-->
29      <ncpus>0</ncpus>
30    </emulator-params>
31
32  </distributions>
33
34  <log>
35    <!-- Use value "1" to enable logging(by default logging is off) -->
36    <enable>1</enable>
37    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
38    <frequency>1</frequency>
39    <logLevel>debug</logLevel>
40  </log>
41
42 </emulation>
```

The XML document defines the emulation experiment details, which consists of three blocks:

- **Emulation**

```
1 <emulation>
2   <emuname>CPU_EMU</emuname>
```

```

3   <emuType>Mix</emuType>
4   <emuresourceType>CPU</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>60</emustopTime>
9   ...
10  </emulation>

```

The ‘emuresourceType’ value is used for a check to ensure that all distributions in an emulstion are of the specified type (CPU in this case). For Emulations with multiple distributions over different resource types use ‘MIX’

- **Distribution**

```

1   <distributions>
2
3   <name>CPU_Distro</name>
4   <startTime>0</startTime>
5   <!--duration in seconds -->
6   <duration>60</duration>
7   <granularity>20</granularity>
8   <!--Minimum time a job can last (seconds) -->
9   <minJobTime>2</minJobTime>
10  <distribution href="/distributions/linear" name="linear" />
11  <!--cpu utilization distribution range-->
12  <startLoad>10</startLoad>
13  <stopLoad>95</stopLoad>
14
15  <emulator href="/emulators/lookbusy" name="lookbusy" />
16  <emulator-params>
17      <!--more parameters will be added -->
18      <resourceType>CPU</resourceType>
19      <!--Number of CPUs to keep busy (default: autodetected) -->
20      <ncpus>0</ncpus>
21  </emulator-params>
22
23  </distributions>

```

- **Log (optional)**

```

1  <log>
2      <!-- Use value "1" to enable logging(by default logging is off) -->
3      <enable>1</enable>
4      <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
5      <frequency>1</frequency>
6      <logLevel>debug</logLevel>
7  </log>

```

In plain english it means - create an emulation named *CPU_EMU* starting *now* and running for *60* sec. The Emulation includes one distribution called *CPU_Distro*, which starts at the same time as emulation, runs for *60* sec, using *linear* pattern. The pattern increases the workload of the *CPU* from *10%* to *95%* in *20* steps by using the *lookbusy* emulator (with none of the steps lasting less than *2* seconds). The workload produced by the application is logged every second with *debug* level information.

Creating Emulation via CLI

To create an emulation via CLI, a local XML *emulation.xml* file as the following can be used:

```
1  <emulation>
2      <emuname>CPU_Emulation</emuname>
3      <emuType>Mix</emuType>
4      <emuresourceType>Mix</emuresourceType>
5      <emustartTime>now</emustartTime>
6      <!--duration in seconds -->
7      <emustopTime>180</emustopTime>
8
9      <distributions>
10         <name>Distro1</name>
11         <startTime>5</startTime>
12         <!--duration in seconds -->
13         <duration>30</duration>
14         <granularity>3</granularity>
15         <minJobTime>2</minJobTime>
16         <distribution href="/distributions/linear" name="linear" />
17         <!--cpu utilization distribution range-->
18         <startLoad>90</startLoad>
19         <stopLoad>10</stopLoad>
20         <emulator href="/emulators/lookbusy" name="lookbusy" />
21         <emulator-params>
22             <!--more parameters will be added -->
23             <resourceType>CPU</resourceType>
24             <!--Number of CPUs to keep busy (default: autodetected)-->
25             <ncpus>0</ncpus>
26
27             </emulator-params>
28         </distributions>
29
30         <distributions>
31             <name>Distro2</name>
32             <startTime>5</startTime>
33             <!--duration in seconds -->
34             <duration>30</duration>
35             <granularity>3</granularity>
36             <minJobTime>2</minJobTime>
37             <distribution href="/distributions/linear" name="linear" />
38             <!--cpu utilization distribution range-->
39             <startLoad>10</startLoad>
40             <stopLoad>90</stopLoad>
41             <emulator href="/emulators/lookbusy" name="lookbusy" />
42             <emulator-params>
43                 <!--more parameters will be added -->
44                 <resourceType>CPU</resourceType>
45                 <!--Number of CPUs to keep busy (default: autodetected)-->
46                 <ncpus>0</ncpus>
47
48                 </emulator-params>
49             </distributions>
50
51             <log>
52                 <!-- Use value "1" to enable logging(by default logging is off) -->
53                 <enable>1</enable>
54                 <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
55                 <frequency>3</frequency>
56             </log>
57
58         </emulation>
```

The command to start the emulation via CLI is:

```
$ ccmsh -x emulation.xml
```

Once sent, the list of scheduled jobs is shown on screen as follow:

Each line from 3-8 shows information of a single scheduled emulation job. Each line provides job's information, for example line 3:

- **INFO:Distribution Manager:Scheduler reply:** -just a generic logger part
- **6-CPU_Emotion** - emulation name, which is a combined string of emulation ID from the DB and emuname value in the XML file
- **7** - database ID number for distribution
- **0** - run number of this distribution
- **Distro1** - name of the distribution taken from XML file
- **lookbusy** - distribution module used to calculate each run parameters
- **cpu** - the target resource used by this run
- **90** - stress value applied to this run
- **Duration 10.0sec.** - how long the job run
- **Start Time: 2013-04-10 09:43:01 End Time: 09:43:11** - time interval when the run is/was executed

More generally, the run/job notation is as follow:

```
(logger reply) - (emulationID-name) - (distribution ID) - (run number) -  
(distribution name) - (distribution module) - (resource) - (stress value) -  
(run duration) - (execution time)
```

Line 10 shows another job which was created for the logger. This job appears only if the optional *log* section is stated in the XML. The logger job executes for the duration of the whole emulation and collects system resource usage information. The logger job name notation can be described in this way:

```
(logger reply) - (emulationID-name) - (logger mark) - {poll interval} - (start  
time)
```

Creating Emulation via API Client (Restfully)

This section provides examples on how to use the REST API via the restfully client.

First you need to create a configuration file for restfully api.cocoma.yml, containing the public IP address of COCOMA:

```
uri: http://131.254.204.223/  
require: [ApplicationVndBonfireXml]
```

The example below creates an emulation with two distributions over the MEM resource. The file can be saved as a .rb and used by restfully. It contains the XML payload for COCOMA and a reference to the config file to connect to the COCOMA VM:

```
require 'rubygems'  
require 'restfully'  
require 'logger'  
  
session = Restfully::Session.new(  
:configuration_file => "~/api.cocoma.yml"
```

```
)  
  
session.logger.level = Logger::INFO  
  
emulation = nil  
  
begin  
  emulation = session.root.emulations.submit(  
    :emuname => "MEM-emulation",  
    :emutype => "Contention",  
    :emuresourceType => "RAM",  
    :emustartTime => "now",  
    :emustopTime => "240",  
    :distributions => [{  
      :name => "MEM-increase",  
      :startTime => "0",  
      :duration => "120",  
      :granularity => "10",  
      :minJobTime => "2",  
      :distribution => {  
        :href => "/distributions/linear_incr",  
        :name => "linear_incr"  
      },  
      :startLoad => "10%",  
      :stopLoad => "80%",  
      :emulator => {  
        :href => "/emulators/stressapptest",  
        :name => "stressapptest"},  
      :'emulator-params' => {  
        :resourceType => "MEM",  
        :memThreads => "1"  
      }  
    },  
    {  
      :name => "MEM-decrease",  
      :startTime => "121",  
      :duration => "119",  
      :granularity => "10",  
      :minJobTime => "2",  
      :distribution => {  
        :href => "/distributions/linear_incr",  
        :name => "linear_incr"  
      },  
      :startLoad => "80%",  
      :stopLoad => "10%",  
      :emulator => {  
        :href => "/emulators/stressapptest",  
        :name => "stressapptest"  
      },  
      :'emulator-params' => {  
        :resourceType => "MEM",  
        :memThreads => "1"  
      }  
    }]  
  )  
  
end
```

The script can be executed as:

```
$ restfully emulation.rb
```

You can access the COCOMA VM interactively through the `restfully` client, and check if the emulation was created successfully:

```
$ restfully -c cocoma.yml
```

```
>> pp root.emulations
>> #<Collection:0x45f9f3e uri="/emulations"
>>   RELATIONSHIPS
>>     parent, self
>>   ITEMS (0..2)/2
>>     #<Resource:0x45b5d3e name="7-CPU_Stress" uri="/emulations/7-CPUStress">
>>     #<Resource:0x4489eb0 name="8-MEM-emulation" uri="/emulations/8-MEM-emulation">>
>> => nil
```

To get more client tutorials check the `restfully` page.

The Web UI

Here we explain how to use the web UI to create and manage emulations and view their results.

Note: For best results only use the Web UI on an up to date browser (may not function as intended on an out of date or older browser)

Opening the UI

Provided that the API is running, the web UI will be accessible at

```
http://[COCOMA API IP]:[COCOMA API PORT]/index.html
```

The *COCOMA API IP* refers to the IP of the interface on which the API have been started. The page is compatible with Chrome, Firefox and Safari web browsers, while it is not with Internet Explorer. The page will automatically load in available emulators, distributions and resources. It will detect which distributions and resources are compatible with the given emulator so that the user needs not worry about creating XML the framework cannot process. Any emulations which already exist will also be displayed in the right hand bar.

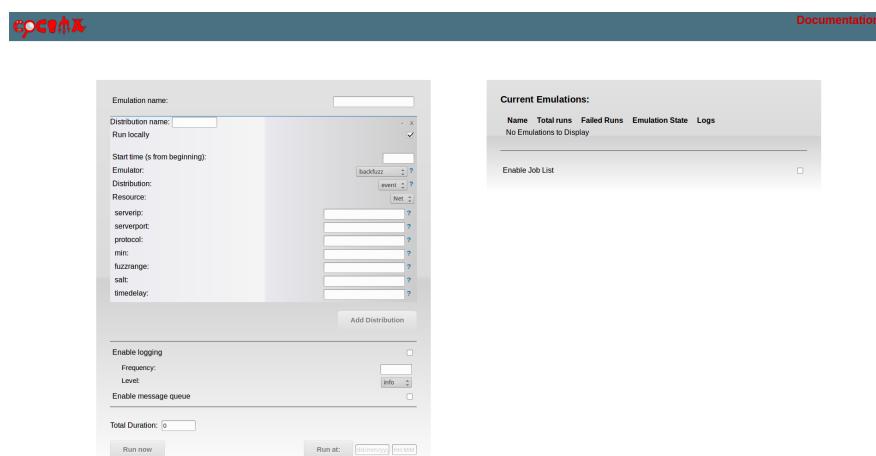


Figure 10.1: COCOMA webUI

Creating an emulation

Each emulation requires a name and at least one distribution, although as many distributions as required can be added. Each distribution requires a name and all required fields to be filled, this data will vary as per the emulation, distribution or resource selected. Some help is provided to explain what each of the arguments are and (if relevant) what their units are. Distribution windows can be minimized for overall readability or removed entirely (not added to the emulation) by clicking the ‘x’ in the top right corner:

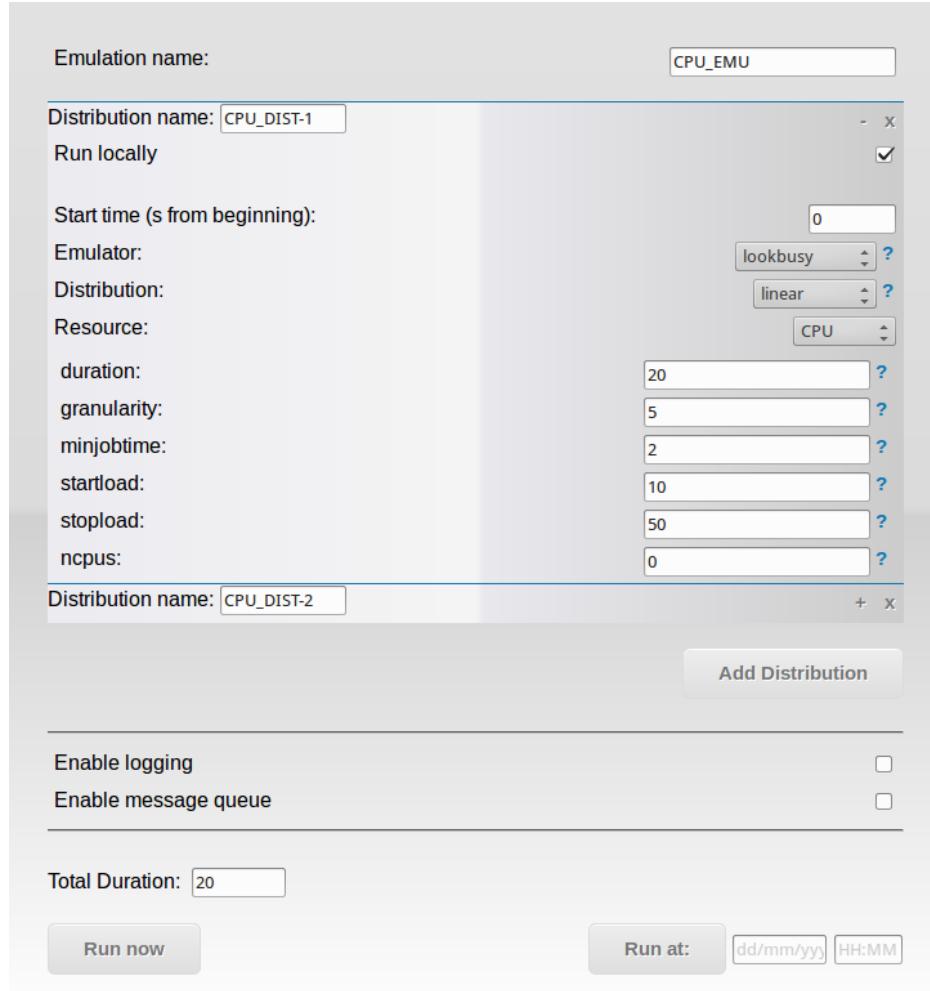


Figure 10.2: Multi-distribution interface

Distribution Parameters

Where possible, help has been added for each distribution parameter. This can be viewed by hovering over the ? beside a parameter; This should display a pop-up with some help and the units (if relevant) of that parameter

Logging and Message Queue

After the distributions have been created and specified, there is an option to enable or disable logging. Enabling logging give 2 more options, the *frequency* in seconds and the *level*, which dictates the amount of output the logs will

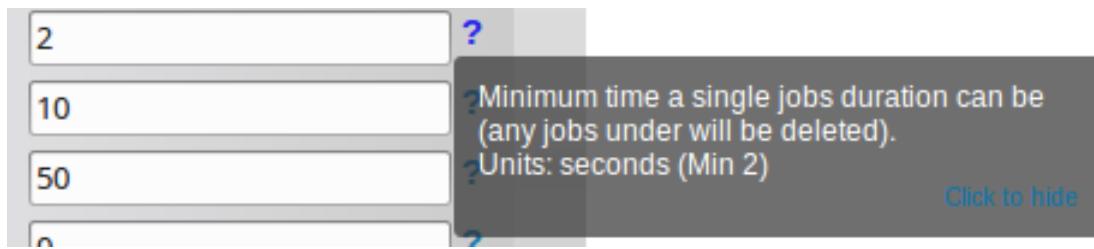


Figure 10.3: Help pop-ups

contain. Below this is the option to enable or disable the message queue followed by various parameters allowing for it's setup

A screenshot of a configuration interface. It includes sections for 'Enable logging' (checkbox checked), 'Frequency' (input field with value 3, dropdown menu showing 'debug'), 'Level' (checkbox checked), 'Enable message queue' (checkbox checked), and several input fields for 'vhost', 'exchange', 'user', 'password', 'host', and 'topic'.

Figure 10.4: Logging and EMQ settings

Running the emulation

Once all the parameters are set there are two options; run the emulation right away by clicking the *Run now* button, or schedule the emulation to begin running at a set time in the future by clicking the *Run at* button:



Figure 10.5: Set time for emulation

Working with existing emulations

Any existing emulations in your system will be listed on the right hand side of the screen. The UI also displays the total number of runs, how many of those failed and the current state of the emulation (*Pending Execution*, *Running*, *Executed*, *Unexecuted Run(s)* or *Failed Run(s)*). Hovering over the emulation name will display the information on that emulation in a popup.

Clicking the small download icon to the right of each emulation will prompt the download of a zip file to your system. This zip file contains the .xml used to create the emulation as well as .csv files with the system logs and the logs for that specific emulation.

The screenshot shows a web-based interface for managing emulations. At the top, a header reads "Current Emulations:". Below it is a table with the following data:

Name	Total runs	Failed Runs	Emulation State	Logs
1-CPU_EMU	2	1	Failed run(s)	
2-CPU_EMU	2	0	Unexecuted run(s)	
4-CPU_EMU	18	0	Pending Execution	
6-CPU_EMU	2	0	Executed	
7-CPU_EMU	2	0	Running	

A tooltip for the row "Emulation name: 7-CPU_EMU" provides the following details:

- Start time: 17/12/2013
- End time: undefined
- Duration: 20 s
- cpu_distro:
 - startTime: 0
 - granularity: 2
 - distribution: linear
 - duration: 20
 - minjobtime: 2
 - startload: 10
 - stopload: 20
 - emulator: lookbusy
- Emulator parameters:
 - resourceType: cpu
 - ncpus: 0

At the bottom right of the interface is a "Refresh Job list" button.

Figure 10.6: Emulations interface

Hello World Example

In this section we provide an “Hello World” example to show the contention effects on the RAM on a colocated VM. The scenario consists of 2 VMs, 1 COCOMA and 1 SuT, and 1 aggregator VM. The 2 VMs are deployed in the same physical node in the Inria onrequest cluster, while the aggregator is deployed in the normal Inria production site. The figure below shows this scenario:

Onrequest resource booking

In order to have esclusive access to the onrequest resource at Inria, a reservation has to be made. Please check how it works at [On Request Resources](#). For example the following restfully script can be used:

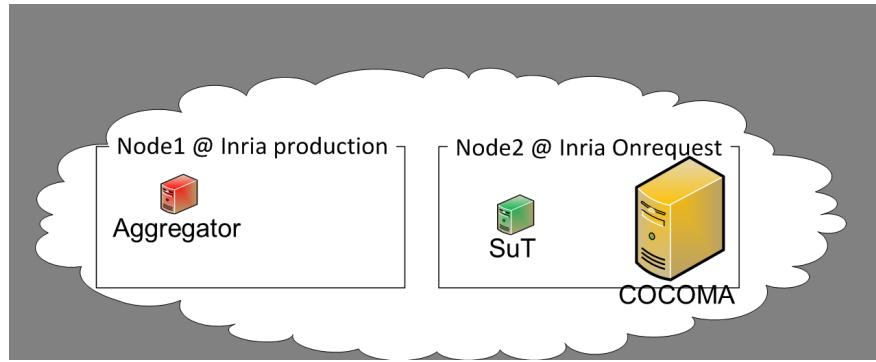


Figure 10.7: Contention scenario

```

reservations = get("/locations/fr-inria/reservations")
resa = reservations.submit(:clusters => { :paradent => 1 }, :from => Time.now.to_i, :to => Time.now.to_i)
pp resa

puts "***"
puts "CLUSTER ID to use in your reservations = #{resa['name']} "

```

This can be started through the following command:

```
$ restfully .restfully/api.bonfire.yml onrequest_reservation.rb
```

Scenario deployment

Once the reservation has been made, in order to deploy the scenario various approaches can be used (please see the relative documentation). In our case we use again a restfully script, which can be seen here

```

require 'rubygems'
require 'restfully'
require 'restfully addons/bonfire'

START_TIME = Time.now.to_i * 1000

COCOMA_NAME = "SAP_COCOMA_2GB_inria_v69"
AGGREGATOR_IMAGE_NAME = "BonFIRE Zabbix Aggregator v8"
WAN_NAME = "BonFIRE WAN"
PUB_NAME = "Public Network"

session = Restfully::Session.new(
  :configuration_file => "~/.restfully/api.bonfire-project.eu.yml",
  :cache => false,
  :gateway => "ssh.bonfire.grid5000.fr",
  :keys => ["~/.ssh/id_bonfire"]
)
session.logger.level = Logger::INFO

cluster = ENV['CLUSTER']

host1 = ENV['HOST1']
#host2 = ENV['HOST2']

count = (ENV['COUNT'] || 1).to_i

```

```
experiment = nil

begin
    # Find an existing running experiment with the same name or submit a new
    # one. This allows re-using an experiment when developing a new script.
    experiment = session.root.experiments.find{|e|
        e['name'] == "SAP-COCOMA-inria-1-HOST-scenario" && e['status'] == "running"
    } || session.root.experiments.submit(
        :name => "SAP-COCOMA-inria-1-HOST-scenario",
        :description => "COCOMA deployed in 2 HOSTS at Inria onrequest - #{Time.now.to_s}",
        :status => "ready",
        :groups => "demos",
        :walltime => 3*3600 # 5 hrs
    )

    # Create shortcuts for location resources:
    inria = session.root.locations[:'fr-inria']
    fail "Can't select the fr-inria location" if inria.nil?
    #hlrs = session.root.locations[:'de-hlrs']
    #fail "Can't select the de-hlrs location" if hlrns.nil?
    #epcc = session.root.locations[:'uk-epcc']
    #fail "Can't select the uk-epcc location" if epcc.nil?

    # In this array we'll store the clients launched at each site:
    #locations = [[epcc, []], [hlrs, []], [inria, []]]
    locations = [[inria, []]]
    # -----
    # Aggregator section
    # -----
```



```
session.logger.info "Launching aggregator..."
#logger.info "Submitting Aggregator in host #{host1}"
logger.info "Submitting Aggregator at Inria ..."
aggregator = experiment.computes.find{|vm|
    vm['name'] == "BonFIRE-monitor-#{experiment['id']}"
} || experiment.computes.submit(
    #payload = {
        :name => "BonFIRE-monitor-#{experiment['id']}",
        :instance_type => "small",
        :disk => [],
        :storage => inria.storages.find{|s|
            s['name'] == AGGREGATOR_IMAGE_NAME
        }, :type => "OS"
    },
    :nic => [],
    :network => inria.networks.find{|n|
        n['name'] == WAN_NAME
    }
},
:location => inria,
#:cluster => cluster,
:context => {
    #:usage => "infra-monitoring-init"
    #:usage => "zabbix-agent;infra-monitoring-init;log-MQevents-in-zabbix"
    #:usage => "zabbix-agent,infra-monitoring-init",
    #'metrics' => XML::Node.new_cdata('<metric>mem-used, free | grep Mem | awk \'print $3\'')
}
#}
```

```

#payload[:host] = host1 unless host1.nil?
#aggregator = experiment.computes.submit(payload)

aggregator_ip = aggregator['nic'][0]['ip']
session.logger.info "Submitted AGGREGATOR##{aggregator['id']} with IP=#{aggregator_ip}, in cluster

def create_cocoma_instance(session, experiment, location, cluster, aggregator_ip, count, host)
computes = experiment.computes

logger.info "Submitting #{count} large COCOMA VMs... in host #{host}"
hostarray = host.split('-')
h = hostarray[0..1].join('-')

count.times do |i|
  payload = {
    :name => "COCOMA-L-#{i}-#{h}",
    #:instance_type => "small",
    :instance_type => "custom",
    :cpu => 4,
    :vcpu => 4,
    :memory => 24*1024,
    :location => location,
    :disk => {
      :storage => location.storages.find{|s|
        s['name'] == COCOMA_NAME}
    },
    #:nic => [{:network => inria.networks.find{|n| n['name'] == WAN_NAME}}],
    :nic => [{:network => location.networks.find{|n| n['name'] == WAN_NAME}}, {:network => loc
    :cluster => cluster,
    :context => {
      :usage => "zabbix-agent",
      :aggregator_ip => aggregator_ip,
      #'metrics' => XML::Node.new_cdata('<metric>cocoma-io, cat /tmp/cocoma-ioprobe.out, rate
      'metrics' => XML::Node.new_cdata('<metric>disk-throughput, dstat -d --float --noheaders
      <metric>cpu-usage, cat /usr/share/pyshared/cocoma/scripts/cpu, rate=1, valuetype=0, his
      <metric>mem-percentage-used, cat /usr/share/pyshared/cocoma/scripts/mem-percentage, rat
      <metric>mem-bytes-used, cat /usr/share/pyshared/cocoma/scripts/mem-bytes, rate=1, value
      ')
    }
  }
  payload[:host] = host unless host.nil?

  c = computes.submit(payload)
  logger.info "Submitted COCOMA-L-#{c['id']} with IP=#{c['nic'][0]['ip']}, in cluster #{clust
end

end

def create_small_cocoma (session, experiment, location, cluster, aggregator_ip, count, host)
computes = experiment.computes
logger.info "Submitting #{count} SUT VM... in host #{host}"
hostarray = host.split('-')
h = hostarray[0..1].join('-')
count.times do |i|
  payload = {
    :name => "SUT-#{i}-#{h}",
    #:instance_type => "small",
    :instance_type => "custom",
    :cpu => 1,

```

```
:vcpu => 1,
:memory => 2*1024, # 2GB
:location => location,
:disk => {
  :storage => location.storages.find{|s|
    s['name'] == COCOMA_NAME}
},
:nic => [[:network => location.networks.find{|n| n['name'] == WAN_NAME}],{:network => location.networks.find{|n| n['name'] == LAN_NAME}}]
:cluster => cluster,
:context => {
  :usage => "zabbix-agent",
  'aggregator_ip' => aggregator_ip,
  # Register metric on the server
  'metrics' => XML::Node.new_cdata('<metric>appmembench, cat /out/appmembenchprobe.out, rate=1, valuetype=0, history=3</metric>
<metric>disk-throughput, dstat -d --float --noheaders 1 1 | awk -F \' \' '{print $2}'</metric>
<metric>cpu-usage, ps afuxw | awk \' { if ($3 ~ /^[0-9]/) {SUM +=$3} } END {print SUM}'</metric>
<metric>appcpubench, cat /out/appcpubenchprobe.out, rate=1, valuetype=0, history=3</metric>
')
}
}
payload[:host] = host unless host.nil?

c = computes.submit(payload)
logger.info "Submitted SuT-#{c['id']} with IP=#{c['nic'][0]['ip']}, in cluster #{cluster}."
end

end

cocoma1 = create_cocoma_instance(session, experiment, inria, cluster, aggregator_ip, count, host1)
sut1 = create_small_cocoma(session, experiment, inria, cluster, aggregator_ip, count, host1)

experiment.update(:status => "running")

session.logger.info "VMs are now READY!"
session.logger.info "*** Aggregator IP: #{aggregator_ip}"

rescue Exception => e
  session.logger.error "#{e.class.name}: #{e.message}"
  session.logger.error e.backtrace.join("\n")
  session.logger.warn "Exception in experiment creation ..."
end
```

The important parts to notice are where the metrics for the SuT and COCOMA are set:

```
<metric>appmembench, cat /out/appmembenchprobe.out, rate=1, valuetype=0, history=3</metric>\n<metric>mem-percentage-used, cat /usr/share/pyshared/cocoma/scripts/mem-percentage, rate=1, valuetype=0</metric>
```

Those metrics will be used in zabbix to show the contention. Also, note that the amount of memory and CPUs allocated to the VMs depends from the node they are deployed to. So, for example for the onrequest *Paradent* machine, which has 8 cores and 31GB in each node, the allocation should be

- 4 cores and ~24GB for COCOMA
- 1 core and ~4GB for the SuT

The assumption here is that COCOMA represents all the other VMs colocated in the same node by the cloud service provider, which consume the majority of the physical resources. In the case of other onresource nodes, such as

Parapluie and *Parapide*, please check their specs and work out how to partition the resources for COCOMA and the SuT.

Zabbix custom graph

After the deployment is completed, we need to create a custom graph in zabbix to show the contention. Open the Zabbix web interface from your experiment in the portal and follow the steps below:

- go to Configurations->Hosts
- click on “Graphs” of 1 of the 2 HOSTS (doesn’t matter which one)
- click on “Create Graph” (on the right side)
- enter a name, e.g COCOMA-SUT-RAM-contention and click add
- click “select”, select the SuT-Benchmark-... host, select the “BonFIRE generated:appmembench”, select “right” on the “Y Axis side” and click “Add”
- click add again, and click “select”, select the COCOMA-LARGE-... host, select the “BonFIRE generated:mem-used”, select a color like red and click “Add”
- You should see both graphs displayed in the preview below. Once happy, click “Save”
- Go to Monitoring->Graphs
- Select the Graph you created from the drop down menu on the right and the graph should appear

Starting the benchmark in the SuT

The SuT VM is in this case a COCOMA image which we’ll use to run our benchmark. The benchmark reserves an amount of free RAM (400MB) and tries to write with the requested throughput over the reserved memory. The metric measures the percentage of success writes over time. In order to start the benchmark use the command:

```
$ /root/appmembenchprobe start 6
```

This starts the benchmark with a throughput of 6GB/s. You should check now the graph in zabbix and you should see the benchmark scores reported. The important thing is that to show the contention the benchmark score should be as close as possible to 100%. In this case the 6GB/s is when the SuT is deployed in *Paradent*. However, for other nodes, such as Parapide, having different specs, the value should be different. In that case 14GB/s seems the most appropriate.

Starting a COCOMA emulation

Check the public IP of the COCOMA VM and open the browser at <**COCOMA-public-IP>/index.html**. This will open the COCOMA web UI. Now you can create a memory contention distribution by selecting *stressapptest* as emulator, *linear_incr* as distribution, and setting a startLoad of 10% and stopLoad of 50%. Once started you can go to the zabbix graph, and should see the benchmark score dropping significantly when the memory distribution kicks in. The image below shows examples of such contention for various distributions:

Note: Similarly to the RAM contention example provided in this section, IO, CPU and Network examples can be created.

XML Examples

This section provides XML payload examples for creating different emulations over various resources.

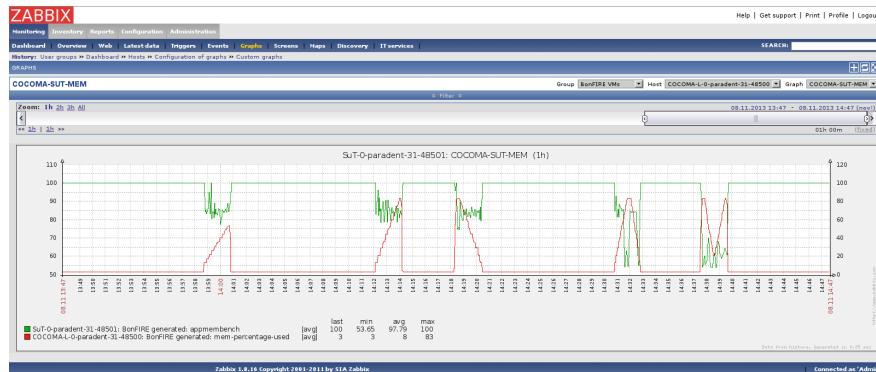


Figure 10.8: Contention examples

CPU

Emulation XML for CPU contention:

```

1 <emulation>
2   <emuname>CPU_EMU</emuname>
3   <emuType>TIME</emuType>
4   <emuresourceType>CPU</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>120</emustopTime>
9
10  <distributions>
11    <name>CPU_Distro</name>
12    <startTime>0</startTime>
13    <!--duration in seconds -->
14    <duration>120</duration>
15    <granularity>24</granularity>
16    <minJobTime>2</minJobTime>
17    <distribution href="/distributions/linear" name="linear" />
18    <!--cpu utilization distribution range-->
19    <startLoad>10</startLoad>
20    <stopLoad>95</stopLoad>
21    <emulator href="/emulators/lookbusy" name="lookbusy" />
22
23    <emulator-params>
24      <resourceType>CPU</resourceType>
25      <!--Number of CPUs to keep busy (default: autodetected)-->
26      <ncpus>1</ncpus>
27    </emulator-params>
28  </distributions>
29
30  <log>
31  <!-- Use value "1" to enable logging(by default logging is off) -->
32  <enable>1</enable>
33  <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
34  <frequency>1</frequency>
35  <logLevel>debug</logLevel>
36  </log>
37
38 </emulation>
```

I/O

Emulation XML for I/O contention:

```

1 <emulation>
2   <emuname>IO_EMU</emuname>
3   <emuType>TIME</emuType>
4   <emuresourceType>IO</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>60</emustopTime>
9
10  <distributions>
11
12    <name>IO_Distro</name>
13    <startTime>0</startTime>
14    <!--duration in seconds -->
15    <duration>60</duration>
16    <granularity>5</granularity>
17    <minJobTime>2</minJobTime>
18    <distribution href="/distributions/linear_incr" name="linear_incr" />
19    <startLoad>1</startLoad>
20    <stopLoad>10</stopLoad>
21    <emulator href="/emulators/stressapptest" name="stressapptest" />
22
23    <emulator-params>
24      <resourceType>IO</resourceType>
25      <!--Size of mem in MB used-->
26      <memsize>1000</memsize>
27      <!--Number of threads-->
28      <memThreads>10</memThreads>
29    </emulator-params>
30
31  </distributions>
32
33  <log>
34    <!-- Use value "1" to enable logging(by default logging is off) -->
35    <enable>1</enable>
36    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
37    <frequency>3</frequency>
38    <logLevel>debug</logLevel>
39  </log>
40
41 </emulation>
```

In this example we use a different distribution called *trapezoidal*:

```

1 <emulation>
2   <emuname>IO_EMU</emuname>
3   <emuType>TIME</emuType>
4   <emuresourceType>IO</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>60</emustopTime>
9
10  <distributions>
```

```
12 <name>IO_Distro</name>
13 <startTime>0</startTime>
14 <!--duration in seconds -->
15 <duration>60</duration>
16 <granularity>5</granularity>
17 <minJobTime>2</minJobTime>
18 <distribution href="/distributions/trapezoidal" name="trapezoidal" />
19 <startLoad>1</startLoad>
20 <stopLoad>10</stopLoad>
21 <emulator href="/emulators/stressapptest" name="stressapptest" />
22
23   <emulator-params>
24     <resourceType>IO</resourceType>
25     <!--Size of mem in MB used-->
26     <memsize>1000</memsize>
27     <!--Number of threads-->
28     <memThreads>10</memThreads>
29   </emulator-params>
30
31 </distributions>
32
33 <log>
34   <!-- Use value "1" to enable logging(by default logging is off) -->
35   <enable>1</enable>
36   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
37   <frequency>3</frequency>
38   <logLevel>debug</logLevel>
39 </log>
40
41 </emulation>
```

Memory

Emulation XML for memory contention:

```
1 <emulation>
2   <emuname>MEM_EMU</emuname>
3   <emuType>TIME</emuType>
4   <emuresourceType>MEM</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>60</emustopTime>
9
10  <distributions >
11    <name>MEM_Distro</name>
12    <startTime>0</startTime>
13    <!--duration in seconds -->
14    <duration>60</duration>
15    <granularity>5</granularity>
16    <minJobTime>2</minJobTime>
17    <distribution href="/distributions/linear_incr" name="linear_incr" />
18    <!--Memory usage (Megabytes) -->
19    <startLoad>100</startLoad>
20    <stopLoad>1000</stopLoad>
21    <emulator href="/emulators/stressapptest" name="stressapptest" />
22    <emulator-params>
```

```

23      <resourceType>MEM</resourceType>
24      <memThreads>0</memThreads>
25    </emulator-params>
26  </distributions>
27
28  <log>
29    <!-- Use value "1" to enable logging(by default logging is off) -->
30    <enable>1</enable>
31    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
32    <frequency>3</frequency>
33    <logLevel>debug</logLevel>
34  </log>
35
36 </emulation>

```

Example for memory emulation using *trapezoidal* distribution:

```

1  <emulation>
2    <emuname>MEM_EMU</emuname>
3    <emuType>TIME</emuType>
4    <emuresourceType>MEM</emuresourceType>
5    <!--date format: 2014-10-10T10:10:10 -->
6    <emustartTime>now</emustartTime>
7    <!--duration in seconds -->
8    <emustopTime>60</emustopTime>
9
10   <distributions >
11     <name>MEM_Distro</name>
12     <startTime>0</startTime>
13     <!--duration in seconds -->
14     <duration>60</duration>
15     <granularity>5</granularity>
16     <minJobTime>2</minJobTime>
17     <distribution href="/distributions/trapezoidal" name="trapezoidal" />
18     <!--Megabytes for memory -->
19     <startLoad>100</startLoad>
20     <stopLoad>1000</stopLoad>
21     <emulator href="/emulators/stressapptest" name="stressapptest" />
22     <emulator-params>
23       <resourceType>MEM</resourceType>
24       <memThreads>0</memThreads>
25     </emulator-params>
26   </distributions>
27
28  <log>
29    <!-- Use value "1" to enable logging(by default logging is off) -->
30    <enable>0</enable>
31    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
32    <frequency>3</frequency>
33    <logLevel>debug</logLevel>
34  </log>
35
36 </emulation>

```

Network

The newtork emulation needs two COCOMA VM's, one that acts as a client and the other as a server. Normally those two VMs are placed in different nodes. The SuT should be composed of at least two VMs placed on the same two nodes of COCOMA. The emulation XML for network contention looks like:

```
1 <emulation>
2   <emuname>NET_emu</emuname>
3   <emuType>TIME</emuType>
4   <emuresourceType>NET</emuresourceType>
5   <!--2014-02-02T10:10:10-->
6   <emustartTime>now</emustartTime>
7   <!--duration in seconds -->
8   <emustopTime>155</emustopTime>
9
10  <distributions>
11    <name>NET_distro</name>
12    <startTime>0</startTime>
13    <!--duration in seconds -->
14    <duration>150</duration>
15    <granularity>10</granularity>
16    <minJobTime>2</minJobTime>
17    <distribution href="/distributions/linear_incr" name="linear_incr" />
18    <!--set target bandwidth to bits per sec-->
19    <startLoad>100</startLoad>
20    <stopLoad>1000</stopLoad>
21    <emulator href="/emulators/iperf" name="iperf" />
22    <emulator-params>
23      <resourceType>NET</resourceType>
24      <serverip>172.18.254.234</serverip>
25      <!--Leave "0" for default 5001 port -->
26      <serverport>5001</serverport>
27      <clientip>172.18.254.236</clientip>
28      <clientport>5001</clientport>
29    </emulator-params>
30  </distributions>
31
32  <log>
33    <!-- Use value "1" to enable logging(by default logging is off) -->
34    <enable>0</enable>
35    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
36    <frequency>3</frequency>
37  </log>
38
39 </emulation>
```

Note: Some network emulators require that you be targeting a port with a web service active on it to work

Multiple distributions emulation

An important feature of COCOMA is the ability to combine multiple distributions within the same emulation. This allows to specify contention properties for multiple resources or create different patterns for the same resource. Distributions can overlap, meaning two distributions can run at the same time frame. If distributions for the same resource overlap and they exceed the available resources, the runs might crash.

- CPU and Memory example

```

1 <emulation>
2   <emuname>CPU_and_Mem</emuname>
3   <emutype>TIME</emutype>
4   <emuresourceType>MIX</emuresourceType>
5   <emustartTime>now</emustartTime>
6   <!--duration in seconds -->
7   <emustopTime>80</emustopTime>
8
9   <distributions>
10    <name>CPU_distro</name>
11    <startTime>0</startTime>
12    <!--duration in seconds -->
13    <duration>60</duration>
14    <granularity>1</granularity>
15    <minJobTime>2</minJobTime>
16    <distribution href="/distributions/linear" name="linear" />
17    <!--cpu utilization distribution range-->
18    <startLoad>10</startLoad>
19    <stopLoad>95</stopLoad>
20    <emulator href="/emulators/lookbusy" name="lookbusy" />
21    <emulator-params>
22      <resourceType>CPU</resourceType>
23      <!--Number of CPUs to keep busy (default: autodetected) -->
24      <ncpus>0</ncpus>
25    </emulator-params>
26  </distributions>
27
28  <distributions>
29    <name>MEM_Distro</name>
30    <startTime>20</startTime>
31    <!--duration in seconds -->
32    <duration>60</duration>
33    <granularity>5</granularity>
34    <minJobTime>2</minJobTime>
35    <distribution href="/distributions/linear_incr" name="linear_incr" />
36    <!--Megabytes for memory -->
37    <startLoad>100</startLoad>
38    <stopLoad>1000</stopLoad>
39    <emulator href="/emulators/stressapptest" name="stressapptest" />
40    <emulator-params>
41      <resourceType>MEM</resourceType>
42      <memThreads>0</memThreads>
43    </emulator-params>
44  </distributions>
45
46  <log>
47    <!-- Use value "1" to enable logging(by default logging is off) -->
48    <enable>1</enable>
49    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
50    <frequency>3</frequency>
51  </log>
52</emulation>

```

- CPU, MEM and IO example

```

1 <emulation>
2   <emuname>CPU_and_Mem</emuname>
3   <emutype>TIME</emutype>

```

```
4   <emuresourceType>MIX</emuresourceType>
5   <emustartTime>now</emustartTime>
6   <!--duration in seconds -->
7   <emustopTime>80</emustopTime>
8
9   <distributions>
10    <name>CPU_distro</name>
11    <startTime>0</startTime>
12    <!--duration in seconds -->
13    <duration>60</duration>
14    <granularity>1</granularity>
15    <minJobTime>2</minJobTime>
16    <distribution href="/distributions/linear" name="linear" />
17    <!--cpu utilization distribution range-->
18    <startLoad>10</startLoad>
19    <stopLoad>95</stopLoad>
20    <emulator href="/emulators/lookbusy" name="lookbusy" />
21    <emulator-params>
22      <resourceType>CPU</resourceType>
23      <!--Number of CPUs to keep busy (default: autodetected)-->
24      <ncpus>0</ncpus>
25    </emulator-params>
26  </distributions>
27
28  <distributions >
29    <name>MEM_Distro</name>
30    <startTime>20</startTime>
31    <!--duration in seconds -->
32    <duration>60</duration>
33    <granularity>5</granularity>
34    <minJobTime>2</minJobTime>
35    <distribution href="/distributions/linear_incr" name="linear_incr" />
36    <!--Megabytes for memory -->
37    <startLoad>100</startLoad>
38    <stopLoad>1000</stopLoad>
39    <emulator href="/emulators/stressapptest" name="stressapptest" />
40    <emulator-params>
41      <resourceType>MEM</resourceType>
42      <memThreads>0</memThreads>
43    </emulator-params>
44  </distributions>
45
46  <distributions>
47    <name>IO_Distro</name>
48    <startTime>0</startTime>
49    <!--duration in seconds -->
50    <duration>60</duration>
51    <granularity>5</granularity>
52    <minJobTime>2</minJobTime>
53    <distribution href="/distributions/linear_incr" name="linear_incr" />
54    <startLoad>1</startLoad>
55    <stopLoad>10</stopLoad>
56    <emulator href="/emulators/lookbusy" name="lookbusy" />
57    <emulator-params>
58      <resourceType>IO</resourceType>
59      <!--Size of blocks to use for I/O, in MB-->
60      <ioBlockSize>10</ioBlockSize>
61      <!--Time to sleep between iterations, in msec-->
```

```

62      <ioSleep>100</ioSleep>
63    </emulator-params>
64  </distributions>
65
66  <log>
67    <!-- Use value "1" to enable logging(by default logging is off) -->
68    <enable>1</enable>
69    <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
70    <frequency>3</frequency>
71  </log>
72</emulation>

```

Event Based Scheduling

In addition to the regular, time based, scheduling there is Event based scheduling. In Event based scheduling the order of jobs in the XML is used to determine which order jobs will run in. Below is a short explanation of how jobs are scheduled when using events:

- Run time based jobs as normal (if there are any) until an Event is reached
- Stop scheduling any further jobs until the Event finishes
- Resume Scheduling jobs, using their start time as a delay after the event finishes. (A job with a start time of 5 would start 5 seconds after the event finishes)
- Repeat until all distributions are scheduled or emuStopTime expires (at which point all running jobs will be killed, and scheduling will stop)

Event Based Emulation example:

```

1  <emulation>
2    <emuName>MAL_EMU</emuName>
3    <emuType>MIX</emuType>
4    <emuResourceType>MIX</emuResourceType>
5    <!--date format: 2014-10-10T10:10:10 -->
6    <emuStartTime>now</emuStartTime>
7    <!--duration in seconds -->
8    <emuStopTime>35</emuStopTime>
9
10   <distributions>
11     <name>MAL_Distro1</name>
12     <startTime>0</startTime>
13     <minJobTime>2</minJobTime>
14     <distribution href="/distributions/event" name="event" />
15     <emulator href="/emulators/backfuzz" name="backfuzz" />
16     <emulator-params>
17       <resourceType>NET</resourceType>
18       <min>100</min>
19       <fuzzRange>900</fuzzRange>
20       <serverip>10.55.168.142</serverip>
21       <serverport>5050</serverport>
22       <protocol>TCP</protocol>
23       <timedelay>1</timedelay>
24       <salt>100</salt>
25     </emulator-params>
26   </distributions>
27
28   <distributions>

```

```
29      <name>CPU_Distro</name>
30      <startTime>5</startTime>
31      <!--duration in seconds -->
32      <duration>10</duration>
33      <granularity>2</granularity>
34      <minJobTime>2</minJobTime>
35      <distribution href="/distributions/linear" name="linear" />
36      <startLoad>10</startLoad>
37      <stopLoad>50</stopLoad>
38      <emulator href="/emulators/lookbusy" name="lookbusy" />
39      <emulator-params>
40          <resourceType>CPU</resourceType>
41          <ncpus>0</ncpus>
42      </emulator-params>
43  </distributions>
44
45  <log>
46      <!-- Use value "1" to enable logging(by default logging is off) -->
47      <enable>0</enable>
48      <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
49      <frequency>3</frequency>
50      <logLevel>debug</logLevel>
51  </log>
52
53  </emulation>
```

Adding a new emulator

In order to add a new emulator, a new wrapper has to be implemented. This needs to inherit from the relative abstract class, which can be found in the same *emulators* directory. The class needs 3 different methods:

- **`__init__`**: Used to accept the emulators parameters and pass them to whatever methods are used to load the desired resource
- **`emulatorHelp`**: Used for displaying help about an emulator (eg. what parameters it needs)
- **`emulatorArgNames`**: Used for returning the names of the arguments that a given emulator takes. If any of the arguments aren't numerical then they must have their name added to the *textBasedArgs* list in the *boundsCompare* method contained in *bin/Library.py*

As well as the following code:

```
sys.path.insert(0, getHomepath() + '/emulators/')
from abstract_emu import *

class run_yourEmulatorName(abstract_emu):
    pass
```

Specific methods to execute the wanted emulator instance with the relative needed parameters will have to be added. Checking the existing emulator wrappers should give a clear view on how the wrapping process can be carried out.

Note: The `zombieBuster` method (from the abstract class) **must** be called at some point in the emulator wrapper (passing in the PID and name of the emulator being ran). One of the purposes of this method is to terminate any remaining processes after the emulation time expires; If it is not used then jobs may be created after the emulation finishes, or jobs created during the emulation may not terminate properly.

Adding a new distribution

In order to add a new distribution, it needs to inherit from the relative abstract class, which can be found in the same *distributions* directory. The class needs 3 different methods:

- **distHelp**: Used for displaying help about a distribution (eg. what Resources types it can use)
- **functionCount**: Used for getting values for: stressValues, runStartTimeList, runDurations. The actual algorithm (which calculates those values) goes in this function
- **argNames**: Used for returning the names of the arguments that a given resource takes. If any of the arguments aren't numerical then they must have their name added to the *textBasedArgs* list in the *boundsCompare* method contained in *bin/Library.py*

As well as the following code:

```
sys.path.insert(0, getHomepath() + '/distributions/')
from abstract_dist import *

class dist_yourDistributionName(abstract_dist):
    pass
```

Additional Features

In this section the additional features of COCOMA will be discussed

Testing

COCOMA has two main sets of tests supplied with it; **API Tests** and **Command Line Interface (CLI) Tests**. Both of which are implemented using python's unit testing framework (pyUnit)

The test files (**TestAPI** and **TestCLI**) are located in “*/usr/share/pyshared/cocoma/unitTest*”

To run a set of tests on the API or CLI, navigate to the *unitTest* folder and use one of the commands:

```
$ python -m unittest -v TestAPI
$ python -m unittest -v TestCLI
```

The *-v* argument gives more verbose output, and may be omitted if required

Individual test results are output to the terminal in the format *test_Emulators* (*TestAPI.TestAPI*) ... *ok* if the test was successful An unsuccessful test will produce the same output, with *ERROR* or *FAIL* instead of *ok*

Once all the tests in the file have run, a summary of the results will be printed. This will indicate which (if any) tests were unsuccessful, and attempt to give a reason why the test failed

CLI Testing Individual tests can be run on the CLI using the syntax

```
$ python -m unittest TestCLI.TestCLI.test_Name
```

Where *test_Name* is replaced by one of the following:

```
test_EMU_CPU
test_EMU_IO
test_EMU_MEM
test_EMU_MEMTrap
test_EMU_MULTI1
test_EMU_MULTI2
```

```
test_EMU_NETWORK  
test_EMU_NowOperator  
test_EMU_Force  
test_EMU_Logging  
  
test_Scheduler_Start  
test_Scheduler_Show  
test_Scheduler_Stop  
  
test_API_Start  
test_API_Show  
test_API_Stop  
  
test_Help  
test_List  
test_Result  
test_Distributions  
test_Emulators  
test_Purge  
test_Jobs
```

This will produce output similar to running the entire set of tests

API Testing Individual tests can be run on the API using the syntax

```
$ python -m unittest TestAPI.TestAPI.test_Name
```

Where test_Name is replaced by one of the following:

```
test_List_Emulation  
test_List_Emulator  
test_List_Distributions  
test_List_Results  
test_List_Jobs  
test_List_Logs  
test_List_SysLogs  
test_List_EmuLogs  
test_List_Tests  
  
test_EMU_Logs  
test_EMUcr_IO  
test_EMUcr_MEM  
test_EMUcr_MEMtrap  
test_EMUcr_NETWORK  
test_EMUcr_MULTI1  
test_EMUcr_MULTI2
```

This will produce output similar to running the entire set of tests

Resource Overloading

In order to prevent resources from becoming Overloaded (using more than 100% of a resource at a point in time,) the system calculates the resource usage before any Emulation is run.

If an Emulation would cause any of the resources to become overloaded, then that emulation will not run and an exception will be raised with the format:

```
Unable to create distribution:  
CPU resource will become Overloaded: Stopping execution
```

If a distribution would cause a resource to run near its maximum value, then the emulation will not run. Instead the user will be informed of this, and asked to re-send the job with force if they want it to run. For the CLI this process would look like:

```
ccmsh -x CPU.xml  
...  
CPU close to maximum value. Re-send with force ('-f') to run  
...  
ccmsh -x CPU.xml -f
```

To add the force argument to a emulation, ran via a REST client, a parameter called `runIfOverloaded` needs to be added to the request. This is done by sending an **encoded dictionary** from the client:

```
XML = XML_GOES_HERE & runIfOverloaded = Y
```

Note: Spaces should not be included in the request (they are shown here for readability). You may need to manually select an option in order to encode the dictionary.

Bounds Compare

In the ‘argNames’ section of each emulator wrapper and distribution there are a number of different arguments required by that class. These are contained in a dictionary in the form “`ARG_NAME`”: {“`upperBound`”: 0, “`lowerBound`”: 100}; where `ARG_NAME` is the name of the required argument. The nested dictionary contains values for the arguments upper and lower bounds (the maximum and minimum value that argument can be). These values are used to check if the value supplied for that argument in the XML is inside the bounds. If the XML value is outside of the bounds then the value will be changed to whatever boundary value it is closest to.

Note: Some arguments (such as trace and serverIP) require text based values; These values aren’t checked by the boundsCompare method, and so the values for their bounds can be any number. It is possible to have text-based values checked by adding a list called “*accepted*” into an arguments dictionary. The dictionary entry for the `packType` argument of *iperf* is shown below as an example:

```
"serverip", {"upperBound":10000, "lowerBound":1, "argHelp":"Server IP to connect to"}
```

Message queue use

COCOMA writes messages to the EMQ, which are used by the provenance service. Each message contains a timestamp, the message content and the component that created it. The message contains further information as the type of action, and various parameters depending on the specific action. The format adopted is to have key starting in capital, and use the *camel* notation in case of multi-words. Below is the set of messages:

```
{"Timestamp": 1378893008.422242, "Message": {"Action": "Scheduler started",  
"Interface": "eth0", "Port": "51889"}, "From": "Scheduler"}  
  
{"Timestamp": 1378893809.897368, "Message": {"Action": "USER REQUEST Create  
Emulation", "File": "tests/02-MEM-Linear_incr-Stressappstest_100-1000.xml"},  
"From": "ccmsh"}  
  
{"Timestamp": 1378893810.206373, "Message": {"Action": "Emulation request  
received", "UserEmulationName": "MEM_EMU"}, "From": "Emulation Manager"}  
  
{"Timestamp": 1378893810.744948, "Message": {"ResourceTypeDist": "mem",
```

```
"JobName": "2-MEM_EMU-2-0-mem_distro-lookbusy-mem", "DistributionName":  
"mem_distro", "Emulator": "lookbusy", "Action": "Job Created", "RunNo":  
"0", "EndTime": "2013-09-11 10:04:31", "EmulationName": "2-MEM_EMU",  
"DistributionID": 2, "StressValue": 100, "StartTime": "2013-09-11 10:03:31",  
"Duration": 60.0}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893811.128323, "Message": {"ResourceTypeDist": "mem",  
"JobName": "2-MEM_EMU-2-1-mem_distro-lookbusy-mem", "DistributionName":  
"mem_distro", "Emulator": "lockbusy", "Action": "Job Created", "RunNo":  
"1", "EndTime": "2013-09-11 10:04:31", "EmulationName": "2-MEM_EMU",  
"DistributionID": 2, "StressValue": 75, "StartTime": "2013-09-11 10:03:43",  
"Duration": 48.0}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893811.479812, "Message": {"ResourceTypeDist": "mem",  
"JobName": "2-MEM_EMU-2-2-mem_distro-lookbusy-mem", "DistributionName":  
"mem_distro", "Emulator": "lookbusy", "Action": "Job Created", "RunNo": "2",  
"EndTime": "2013-09-11 10:04:31", "EmulationName": "2-MEM_EMU",  
"DistributionID": 2, "StressValue": 75, "StartTime": "2013-09-11 10:03:55",  
"Duration": 36.0}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893811.838568, "Message": {"ResourceTypeDist": "mem",  
"JobName": "2-MEM_EMU-2-3-mem_distro-lookbusy-mem", "DistributionName":  
"mem_distro", "Emulator": "lockbusy", "Action": "Job Created", "RunNo":  
"3", "EndTime": "2013-09-11 10:04:31", "EmulationName": "2-MEM_EMU",  
"DistributionID": 2, "StressValue": 75, "StartTime": "2013-09-11 10:04:07",  
"Duration": 24.0}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893812.189469, "Message": {"ResourceTypeDist": "mem",  
"JobName": "2-MEM_EMU-2-4-mem_distro-lookbusy-mem", "DistributionName":  
"mem_distro", "Emulator": "lockbusy", "Action": "Job Created", "RunNo":  
"4", "EndTime": "2013-09-11 10:04:31", "EmulationName": "2-MEM_EMU",  
"DistributionID": 2, "StressValue": 75, "StartTime": "2013-09-11 10:04:19",  
"Duration": 12.0}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893812.621874, "Message": {"Action": "Emulation created",  
"EmulationName": "MEM_EMU"}, "From": "Emulation Manager"}  
  
{ "Timestamp": 1378893871.00535, "Message": {"Action": "Emulation finished",  
"EmulationName": "2-MEM_EMU"}, "From": "Logger"}  
  
{ "Timestamp": 1378893871.163372, "Message": {"Action": "Job Executed Successfully",  
"StartTime": "2013-09-11 10:04:07", "Duration": 24.0,  
"EndTime": "2013-09-11 10:04:31", "StressValue": 75, "JobName":  
"2-MEM_EMU-2-3-mem_distro-lookbusy-mem"}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893871.274156, "Message": {"Action": "Job Executed Successfully",  
"StartTime": "2013-09-11 10:04:19", "Duration": 12.0, "EndTime":  
"2013-09-11 10:04:31", "StressValue": 75, "JobName":  
"2-MEM_EMU-2-4-mem_distro-lookbusy-mem"}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893871.398665, "Message": {"Action": "Job Executed Successfully",  
"StartTime": "2013-09-11 10:03:55", "Duration": 36.0, "EndTime":  
"2013-09-11 10:04:31", "StressValue": 75, "JobName":  
"2-MEM_EMU-2-2-mem_distro-lookbusy-mem"}, "From": "Scheduler"}  
  
{ "Timestamp": 1378893871.493218, "Message": {"Action": "Job Executed Successfully",  
"StartTime": "2013-09-11 10:03:43", "Duration": 48.0,  
"EndTime": "2013-09-11 10:04:31", "StressValue": 75, "JobName":
```

```

"2-MEM_EMU-2-1-mem_distro-lookbusy-mem"}, "From": "Scheduler"}

{ "Timestamp": 1378893871.628944, "Message": {"Action": "Job Executed Successfully",
"StartTime": "2013-09-11 10:03:31", "Duration": 60.0, "EndTime":
"2013-09-11 10:04:31", "StressValue": 100, "JobName":
"2-MEM_EMU-2-0-mem_distro-lookbusy-mem"}, "From": "Scheduler"}

{ "Timestamp": 1378893913.604134, "Message": {"Action":
"USER REQUEST list all Emulations"}, "From": "ccmsh"}

{ "Timestamp": 1378893929.615051, "Message": {"Action":
"USER REQUEST list Emulation", "EmulationName": "2-MEM_EMU"}, "From": "ccmsh"}

{ "Timestamp": 1378894024.729127, "Message": {"Action":
"USER REQUEST delete Emulation", "EmulationName": "2-MEM_EMU"}, "From": "ccmsh"}

{ "Timestamp": 1378894042.969776, "Message": {"Action":
"USER REQUEST purge all Emulations"}, "From": "ccmsh"}

```

Real trace parse

This feature allows a user to create a distribution from a real trace file. The format of the trace file has to be as follow:

```

NCPUS 2
MEMTOTAL 2074448
TIMESTAMP 1378900076312
POLLFR 1
CPU%      MEMUSED%
2          34
2          34
2          34
2          34

```

The first 4 lines provide information about the machine the trace was recorded from. This allows to scale the usage to the machine that has to reply it. As it can be seen, for now only **CPU** and **MEM** are supported. In the future, **IO** and **NET** might be supported too. Below is a xml snippet showing a new tag called **trace** which provides the path to the trace file from which the distribution **real_trace** creates the runs:

```

<distributions>
    <name>realTrace</name>
    <startTime>0</startTime>
    <distribution href="/distributions/real_trace" name="real_trace" />
    <trace>/path/to/real-trace_1.txt</trace>
    <emulator href="/emulators/lookbusy" name="lookbusy" />
    <emulator-params>
        <resourceType>MEM</resourceType>
        <malloclimit>4004</malloclimit>
        <!--time between iterations in usec (default 1000)-->
        <memSleep>0</memSleep>
        <!-- value to group jobs (+/- %)-->
        <groupingRange>5</groupingRange>
    </emulator-params>
</distributions>

```

The **duration** section is not needed as the actual duration is calculated from the trace itself. So if the emulation ends before the distribution, all jobs left (running and scheduled) will be stopped.

The **groupingRange** section is used to group stress values from the supplied trace file; this by grouping together consecutive, and averaging, values which are within the specified range (So if we wanted to group the values ‘10, 14, 18, 22’ with a grouping range of 5; then the first 3 values would be grouped and averaged to give 14, though 22 would not be grouped as it is outside the range).

As the concept of distribution in COCOMA relates to a single resource (CPU, RAM, IO, NET), if a mixed (CPU and RAM) real trace emulation wanted to be performed, 2 distributions can be added in the xml, each targeting one of the resources, but having the same *startTime* and *trace*.

Recording a real trace COCOMA ships with a script called *rec_res_usage.sh* which can be used to create a trace file with CPU and MEM used. The script can get as option the recording frequency, which by default is 1 sec. As the script can be used also as a live monitoring tool, in order to save the data into a file, the output redirection should be used, such as:

```
$ rec_res_usage.sh 2 > trace_file.txt
    this uses a polling time of 2 seconds

$ timeout 30s rec_res_usage.sh 2 > trace_file.txt
    this uses the command *timeout* in front of the script so that
    it will run for the specified (30 seconds) amount of time
```

Event Based Scheduling

In addition to the regular, time based, scheduling COCOMA offers Event based scheduling (Only usable with the backfuzz emulator at present). This was introduced as the amount of time it takes for a network distribution to run can vary depending on a number of unknown factors (for example a slower network would take longer to send data over). In Event based scheduling the order of distributions in the supplied XML is used to determine which order distributions will run in. Below is a short explanation of how distributions are scheduled when using events:

- Run time based distributions as normal (if there are any) until an Event is reached
- Stop scheduling any further distributions until the Event finishes
- Resume Scheduling distributions, using their start time as a delay after the event finishes. (A distribution with a start time of 5 would start 5 seconds after the event finishes)
- Repeat until all distributions are scheduled or *emuStopTime* expires (at which point all running jobs will be killed, and scheduling will stop)

Event Based Emulation example:

```
1  <emulation>
2      <emuname>MAL_EMU</emuname>
3      <emuType>MIX</emuType>
4      <emuresourceType>MIX</emuresourceType>
5      <!--date format: 2014-10-10T10:10:10 -->
6      <emustartTime>now</emustartTime>
7      <!--duration in seconds -->
8      <emustopTime>35</emustopTime>
9
10     <distributions>
11         <name>MAL_Distro1</name>
12         <startTime>0</startTime>
13         <distribution href="/distributions/event" name="event" />
14         <emulator href="/emulators/backfuzz" name="backfuzz" />
15         <emulator-params>
16             <resourceType>NET</resourceType>
```

```

17      <min>100</min>
18      <fuzzRange>900</fuzzRange>
19      <serverip>10.55.168.142</serverip>
20      <serverport>5050</serverport>
21      <protocol>TCP</protocol>
22      <timedelay>1</timedelay>
23      <salt>100</salt>
24  
```

`</emulator-params>`
`</distributions>`
`<distributions>`
 `<name>CPU_Distro</name>`
 `<startTime>5</startTime>`
 `<!--duration in seconds -->`
 `<duration>10</duration>`
 `<granularity>2</granularity>`
 `<distribution href="/distributions/linear" name="linear" />`
 `<startLoad>10</startLoad>`
 `<stopLoad>50</stopLoad>`
 `<emulator href="/emulators/lookbusy" name="lookbusy" />`
 `<emulator-params>`
 `<resourceType>CPU</resourceType>`
 `<ncpus>0</ncpus>`
 `</emulator-params>`
 `</distributions>`
`<log>`
 `<!-- Use value "1" to enable logging(by default logging is off) -->`
 `<enable>0</enable>`
 `<!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->`
 `<frequency>3</frequency>`
 `<logLevel>debug</logLevel>`
`</log>`
`</emulation>`

In the above example the Event based distribution would first run to completion, then the time (CPU) distribution would be run 5 seconds after the event finishes

Malicious module

The malicious module allows users to create distributions that can target a specific machine by sending fuzzing data over a chosen protocol. As the emulator supporting our malicious module is *backfuzz*^{1 2}, it offers fuzzing over various known protocol such as *HTTP*, *SSH*, *FTP*, *IMAP*, etc. The nice thing that all protocols are added to the tool as plugins, so if a new protocol wants to be tested, a new plugin for it can be created and added to the tool for the purpose. The fuzzing process time cannot be known a priori as it depends from factors out of the user control, such as the network between COCOMA and the SuT to target. Therefore, the **event-driven** approach was introduced to support this. The xml snippet below (the same of the event-driven section) shows a malicious distribution using backfuzz:

```

1 <emulation>
2   <emuuname>MAL_EMU</emuuname>
3   <emuType>NET</emuType>
4   <emuresourceType>NET</emuresourceType>
5   <!--date format: 2014-10-10T10:10:10 -->
```

¹ <https://github.com/localh0t/backfuzz>

² <http://www.darknet.org.uk/2012/03/backfuzz-multi-protocol-fuzzing-toolkit-supports-httplibimap-etc/>

```
6  <emustartTime>now</emustartTime>
7  <!--duration in seconds -->
8  <emustopTime>120</emustopTime>
9
10 <distributions>
11   <name>MAL_Distro</name>
12   <startTime>0</startTime>
13   <distribution href="/distributions/event" name="event" />
14   <emulator href="/emulators/backfuzz" name="backfuzz" />
15   <emulator-params>
16     <resourceType>NET</resourceType>
17     <min>100</min>
18     <fuzzRange>900</fuzzRange>
19     <serverip>10.55.168.142</serverip>
20     <serverport>5050</serverport>
21     <protocol>TCP</protocol>
22     <!-- Timeout (default 0.8s) -->
23     <timedelay>1</timedelay>
24     <salt>10</salt>
25   </emulator-params>
26 </distributions>
27
28 <log>
29   <!-- Use value "1" to enable logging(by default logging is off) -->
30   <enable>0</enable>
31   <!-- Use seconds for setting probe intervals(if logging is enabled default is 3sec) -->
32   <frequency>3</frequency>
33   <logLevel>debug</logLevel>
34 </log>
35
36 </emulation>
```

In the emulator parameters part we can specify the server IP and its port, the minimum and maximum lenght of the fuzzing string sent, the type of protocol and the time after which the fuzz starts.

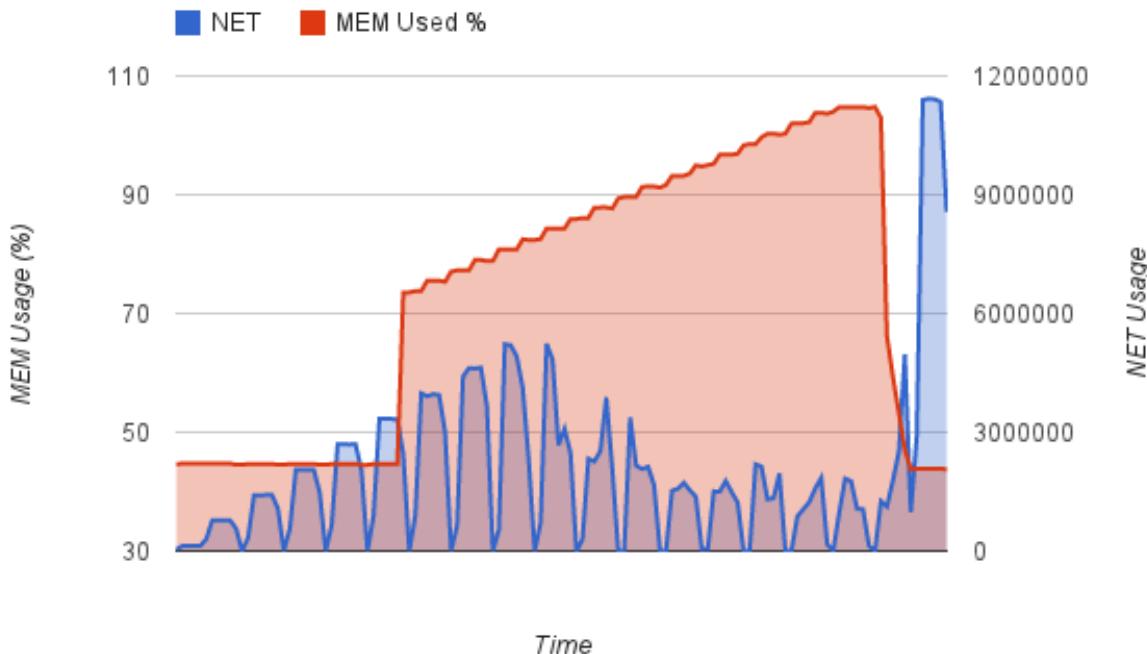
Known Issues

The interaction of the various emulators used in COCOMA can cause unexpected issues. Some of these issues are listed below (This is *not* an exhaustive list, and will be updated as new issues are discovered)

- Stressapptest: it uses ~100% CPU, regardless of what resource it is being ran on. Also, when using it with high memory consumption, it can create issues to processes running in the COCOMA VM, such as causing them to crash. The user should be aware of how it is used;
- If a Linear increase distribution is run on memory using stressapptest at the same time as Iperf is being used to load the Network, then the Network resource may not reach its target load. This problem is usually encountered when the memory usage reaches over ~80% (as shown in the graph below)
- When running an Emulation containing an Event based distribution, then the list of jobs (seen by using the command ‘ccmsh -j’) may not be complete as some of the jobs have yet to be scheduled
- The Web UI may not work properly on older or out of date browsers

10.2.3 Indices and tables

- *genindex*



- Glossary

10.3 Provenance

REFERENCE DOCS

11.1 BonFIRE Architecture

The BonFIRE architecture has been designed to support experiments over multiple heterogeneous cloud testbeds with key functionalities such as monitoring at infrastructure and virtual machine level, experiment management with a single declarative experiment descriptor, elasticity, and resource management for deployment of application software over a variety of differently configured resources (compute, storage, and network)

The BonFIRE architecture has five layers and a set of cross cutting capabilities for monitoring and identity management. The layers are:

1. a Portal
2. an Experiment Manager
3. a Resource Manager
4. an Enactor; and
5. a Testbed site layer

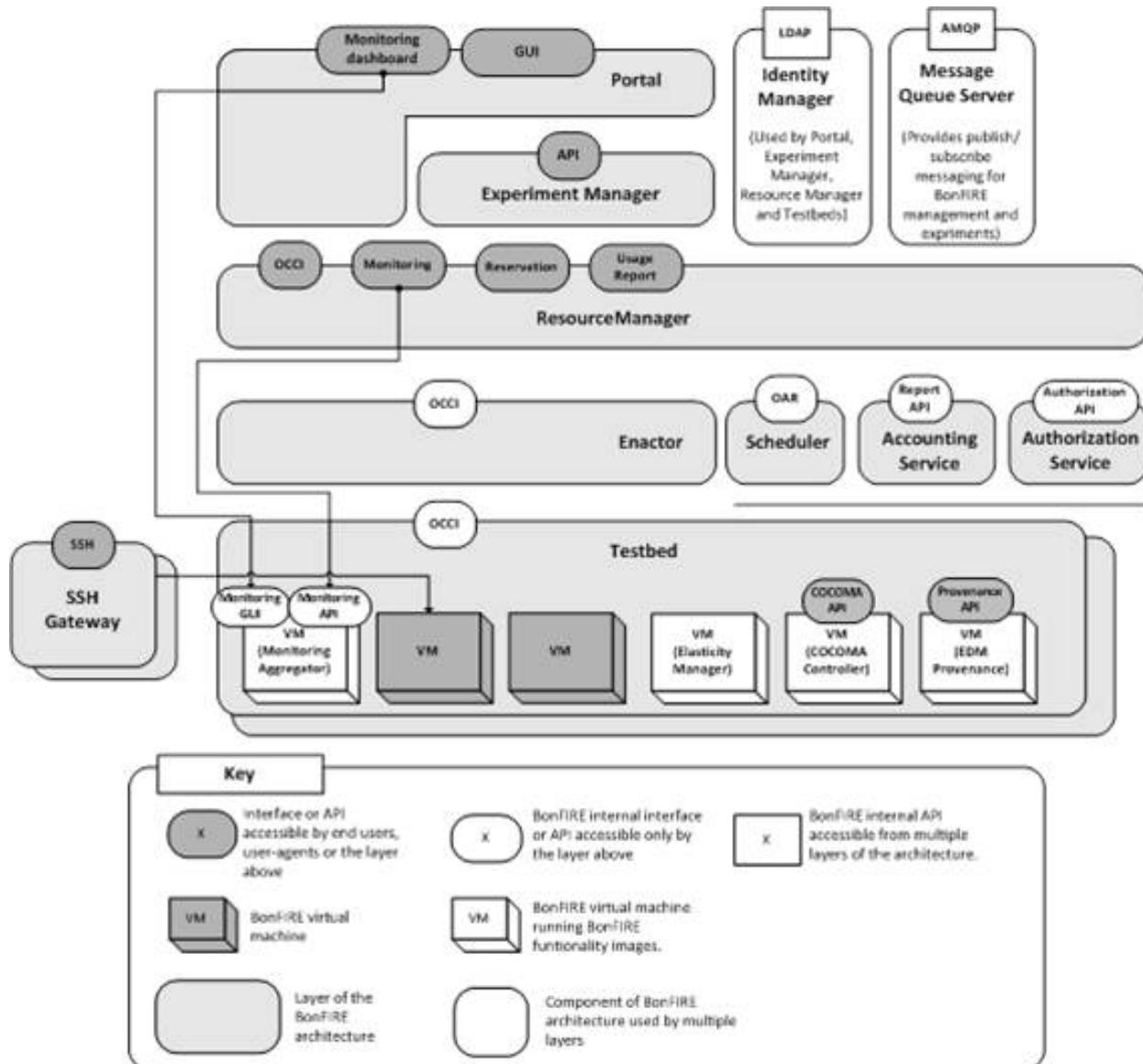
Each layer exposes its functionality via a set of APIs. The APIs of the lowest two layers, the Testbeds layer and the Enactor layer, can only be used by the layer above them. The APIs of the next two layers, the Experiment Manager and the Resource Manager, can be used by experimenters, agents acting on behalf of experimenters, and the layer above them.

11.1.1 The Portal

The Portal offers the experimenters a graphical interface to the BonFIRE capabilities, is a simplified entry point for the experimenters, and supports specification of an experiment at both experiment level granularity (through experiment descriptors) and resource level granularity through multiple single resources. The Portal also provides a means of monitoring and has a view of the experimenters' data, the running experiments, the available resources at each testbed site and information for experiment support.

11.1.2 Experiment Manager

The Experiment Manager provides an API to the Portal or other user agents. A user agent is program acting on behalf of the user (experimenter), to pursue the user's goals. The main task of the Experiment Manager is to schedule, plan and orchestrate the execution of an experiment as described by an experiment description. An experiment can be



specified with all resources for an initial deployment in a single experiment description, and the Experiment Manager manages its execution to the Resource Manager.

11.1.3 Resource Manager

The Resource Manager provides API to the Portal, Experiment Manager and other user agents. Its main task is to provide a resource level interface from which compute, network and storage resources can be created and managed. These resources may physically reside at any testbed in the BonFIRE system. The Resource Manager is in charge of controlling the life-cycle of an experiment and terminating the experiment when its ‘walltime’ expires.

11.1.4 Enactor

The Enactor shields the implementation details of how to interact with various BonFIRE testbeds from the Resource Manager. It provides API exclusively for use by the Resource Manager. It allows the decoupling of the specific implementations and versions of the testbed API from the BonFIRE Resource Manager.

11.1.5 Testbed Sites

The internal BonFIRE cloud testbed sites use a common OCCI API to expose their resources to the Enactor. There are three types of resources: network, compute and storage. Compute resources refers to virtual machines created for the experiment. Network resources connect these VMs and storage resources are disk blocks that can be attached to VMs but whose lifetime can extend beyond that of the VM.

11.2 BonFIRE API Specification

11.2.1 Overview

An introduction to the API concepts can be found in *Overview of Client Tools*. Please note that:

- The API does not support batch operations (i.e. sending one big experiment descriptor with all the resources required). User has to request resources one by one. It is the responsibility of the user to create resources in the right order.
- Creating a resource at the broker layer means that the resource will be immediately instantiated at the testbed layer. The user is immediately notified of the success or failure of her request.
- It is the responsibility of the user to start/stop resources based on the needs of her experiment.
- The API does not provide a *monitor* resource. The user has to start a compute resource with a predefined image containing the BonFIRE monitoring solution, and properly set up the context in all the other compute resources that need to be monitored.
- To ensure a limited execution time, a maximum `walltime` will be enforced for an experiment. When the experiment hits its `walltime`, the deletion of the experiment will be automatically requested.
- A new password will be generated for each experiment created. This password will be automatically added to the context of each compute resource, and automatically revoked when the experiment reaches a final state (cancelation/termination).

11.2.2 Interface

Note: The following specification is given for **information purposes only**. If you are a developer, please note that you should NOT hardcode URIs and HTTP verbs in your client libraries. Use the semantics of the *Media Type* to derive that kind of information at runtime.

- The API must have an entry-point.

```
GET / HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
```

=>

```
HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/" >
  <version>0.10.1</version>
  <timestamp>1311582412</timestamp>
  <link rel="experiments" href="/experiments" type="application/vnd.bonfire+xml"/>
  <link rel="locations" href="/locations" type="application/vnd.bonfire+xml"/>
  <link rel="users" href="/users" type="application/vnd.bonfire+xml"/>
</root>
```

- The API must provide the list of active experiments for the currently logged in user. Use query parameters if you want to filter out results.

```
GET /experiments?offset={{offset}}&limit={{limit}}&sort_by={{sort_by}}&order={{ASC|DESC}} HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
```

=>

```
HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
             href="/experiments?offset=40&limit=20">
  <items offset="40" total="110">
    <experiment>
      ...
    </experiment>
    <experiment>
      ...
    </experiment>
    ...
  </items>
  <link rel="next" href="/experiments?offset=60&limit=20" type="application/vnd.bonfire+xml" />
  <link rel="prev" href="/experiments?offset=20&limit=20" type="application/vnd.bonfire+xml" />
  <link rel="top" href="/experiments?limit=20" type="application/vnd.bonfire+xml" />
  <link rel="parent" href="/" type="application/vnd.bonfire+xml" />
</collection>
```

- The API must allow a user to create a new experiment container.

```
POST /experiments HTTP/1.1
Host: api.bonfire-project.eu
```

```

Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>My Experiment</name>
  <description>Experiment description</description>
  <walltime>7200</walltime>
</experiment>

=>

HTTP/1.1 201 Created
Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
            href="/experiments/{{experiment_id}}">
  <name>My Experiment</name>
  <description>Experiment description</description>
  <walltime>7200</walltime>
  <computes>
    ...
  </computes>
  <storages>
    ...
  </storages>
  <networks>
    ...
  </networks>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml" />
</experiment>

```

- The API must allow a user to fetch an experiment she owns:

```

GET /experiments/{{experiment_id}} HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml

```

```

=>

HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<experiment xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
            href="/experiments/{{experiment_id}}">
  ...
  <computes>
    <compute href="/locations/{{location_id}}/computes/{{compute_id}}" name="Compute1" />
    <compute href="/locations/{{location_id}}/computes/{{compute_id}}" name="Compute2" />
  </computes>
  <storages>
    <storage href="/locations/{{location_id}}/storages/{{storage_id}}" name="Storage1" />
    <storage href="/locations/{{location_id}}/storages/{{storage_id}}" name="Storage2" />
  </storages>
  <networks>
    <network href="/locations/{{location_id}}/networks/{{network_id}}" name="Network1" />
  
```

```

<network href="/locations/{{location_id}}/networks/{{network_id}}" name="Network2" />
</networks>
<link rel="parent" href="/" type="application/vnd.bonfire+xml" />
</experiment>

```

- The API must allow a user to delete an experiment she owns (experiment resources will be deleted in a near future, and the experiment status set to *terminated*):

```

DELETE /experiments/{{experiment_id}} HTTP/1.1
Host: api.bonfire-project.eu
Accept: */*

```

=>

```

HTTP/1.1 202 Accepted
Content-Length: 0
Location: https://api.bonfire-project.eu/experiments/{{experiment_id}}

```

- The API must allow a user to add a compute resource to an experiment she owns (see the XML Schema Definition for the list of available elements in a compute description):

```

POST /experiments/{{experiment_id}}/computes HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

```

```

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <name>Compute name</name>
    <description>Compute description</description>
    <link rel="location"
        href="/locations/{{location_id}}" />
    ...
</compute>

```

=>

```

HTTP/1.1 201 Created
Content-Type: application/vnd.bonfire+xml
Location: https://api.bonfire-project.eu/locations/{{location_id}}/computes/{{compute_id}}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
        href="/locations/{{location_id}}/computes/{{compute_id}}">
    <name>Compute name</name>
    <description>Compute description</description>
    <link rel="location"
        href="/locations/{{location_id}}" />
    ...
</compute>

```

- The API must allow a user to update a compute resource of an experiment she owns:

```

PUT /locations/{{location_id}}/computes/{{resource_id}} HTTP/1.1
Host: api.bonfire-project.eu
Content-Type: application/vnd.bonfire+xml
Accept: application/vnd.bonfire+xml

```

```

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi">

```

```

...
</compute>

=>

HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
          href="/locations/{{location_id}}/computes/{{compute_id}}">
  ...
</compute>

```

- The API must allow a user to add a network resource to an experiment she owns (see XML Schema Definition for the list of available elements):

```

POST /experiments/{{experiment_id}}/networks HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Network name</name>
  <description>Network description</description>
  <link rel="location"
        href="/locations/{{location_id}}"/>
  ...
</network>

=>

HTTP/1.1 201 Created
Content-Type: application/vnd.bonfire+xml
Location: https://api.bonfire-project.eu/locations/{{location_id}}/networks/{{network_id}}

<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
          href="/locations/{{location_id}}/networks/{{network_id}}">
  <name>Network name</name>
  <description>Network description</description>
  <link rel="location"
        href="/locations/{{location_id}}"/>
  ...
</network>

```

- The API must allow a user to add a storage resource to an experiment she owns (see XML Schema Definition for the list of available elements):

```

POST /experiments/{{experiment_id}}/storages HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <name>Storage name</name>
  <description>Storage description</description>
  <link rel="location"

```

```
    href="/locations/{{location_id}}" />
...
</storage>

=>

HTTP/1.1 201 Created
Content-Type: application/vnd.bonfire+xml
Location: https://api.bonfire-project.eu/locations/{{location_id}}/storages/{{storage_id}>

<?xml version="1.0" encoding="UTF-8"?>
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
          href="/locations/{{location_id}}/storages/{{storage_id}}">
  <name>Storage name</name>
  <description>Storage description</description>
  <link rel="location"
        href="/locations/{{location_id}}" />
...
</storage>
```

- The API must allow a user to delete a resource of an experiment she owns:

```
DELETE /locations/{{location_id}}/(computes|networks|storages)/{{resource_id}} HTTP/1.1
Host: api.bonfire-project.eu
Accept: */*
```

=>

```
HTTP/1.1 202 Accepted
Content-Length: 0
Location: https://api.bonfire-project.eu/locations/{{location_id}}/(computes|networks|storages)
```

- The API must provide the list of participating testbeds:

```
GET /locations HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml
```

=>

```
HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations">
  <items offset="0" total="3">
    <location>
      ...
    </location>
    <location>
      ...
    </location>
    ...
  </items>
  <link rel="parent" href="/" type="application/vnd.bonfire+xml" />
</collection>
```

- The API must allow a user to see a location.

```

GET /locations/{{location_id}} HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml
Content-Type: application/vnd.bonfire+xml

=>

HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<location xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
            href="/locations/{{location_id}}">
    ...
    <link rel="computes"
          href="/locations/{{location_id}}/computes"
          type="application/vnd.bonfire+xml" />
    <link rel="networks"
          href="/locations/{{location_id}}/networks"
          type="application/vnd.bonfire+xml" />
    <link rel="storages"
          href="/locations/{{location_id}}/storages"
          type="application/vnd.bonfire+xml" />
    <link rel="parent"
          href="/"
          type="application/vnd.bonfire+xml" />
</location>

```

- The API must allow a user to fetch her profile:

```

GET /users/{{user_id}} HTTP/1.1
Host: api.bonfire-project.eu
Accept: application/vnd.bonfire+xml

=>

HTTP/1.1 200 OK
Content-Type: application/vnd.bonfire+xml

<?xml version="1.0" encoding="UTF-8"?>
<user href="/users/{{user_id}}" xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <id>{{user_id}}</id>
    <uid>2019</uid>
    <gid>2000</gid>
    <name>User Name</name>
    <email>user@server.ltd</email>
    <homedir>/home/{{user_id}}</homedir>
    <shell>/bin/sh</shell>
    <keys>ssh-rsa AAAAB3Nz...</keys>
    <link rel="parent" href="/" />
</user>

```

11.2.3 XML Schema Definition

The *XML Schema Definition* formally describes the elements contained in each resource representation.

11.2.4 Status Codes

The previous section shows the status code returned when the request is successful. A user-agent programmed against the BonFIRE API must also support the following status codes and the semantic associated (see [Status Code Definitions](#)).

For 2xx and 3xx codes, standard semantics apply. Please refer to the HTTP specification for more information.

400 Bad Request

The request could not be understood by the server due to malformed syntax. The client **SHOULD NOT** repeat the request without modifications.

401 Authorization Required

The request requires user authentication. Please retry by adding your credentials to the request.

403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request **SHOULD NOT** be repeated.

404 Not Found

The server has not found anything matching the given Request-URI. You **MAY** retry the request if you think a resource will be available at some point at this URI.

405 Method Not Allowed

The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response will contain an *Allow* header specifying which methods are allowed.

406 Not Acceptable

You asked for a media type that we do not support. Please change your *Accept* header.

412 Precondition Failed

The precondition given in one or more of the request-header fields (on conditional PUT/GET/etc.) evaluated to false when it was tested on the server.

415 Unsupported Media Type

The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

500 Internal Server Error

Something went wrong from our side. You may retry the request. If the error persists, please report to an administrator if you can.

502 Bad Gateway

The server failed at forwarding the request to a subsequent server. Please retry and report to the administrator if the issue persists.

503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. Please retry later.

504 Gateway Timeout

The server was unable to fulfill the request in a reasonable amount of time. You may retry the request.

11.2.5 Non Functional Aspects

Cacheability

Every response includes at least validation headers (*ETag*) and cache control headers (*Cache-Control*). API clients should use the validation headers to make conditional requests when GETting or PUTting a resource.

Performance

Responses will be returned in less than 60 seconds at most.

Security

The API uses SSL/TLS encryption over HTTP (HTTPS). Only authenticated clients can access the API. The Authentication Scheme that is used is [HTTP Basic Authentication](#).

11.3 Media Type

A Media Type (also called MIME Type, or Content-Type) is an identifier for file formats on the Internet. When a Client (we'll call it a *User-Agent* in the rest of the documentation) receives an HTTP response from a Server, it needs to know the content-type of the response to be able to interpret it, and make use of it. For instance, your browser knows how to deal with formats such as `text/html` (HTML source file), `image/jpeg` (JPEG Image), `application/pdf` (PDF Document), etc.

The Media Type we are using in BonFIRE is a custom type called BonFIRE+XML, identified by the `application/vnd.bonfire+xml` Media Type. As the name implies, it is based on the XML format, but `application/vnd.bonfire+xml` payloads have elements and attributes with specific semantics that we'll describe below.

A User-Agent that knows about these specifics will then be able to handle that kind of payloads, and interact with the BonFIRE API.

11.3.1 Elements

Please refer to the [XML Schema Definition](#) file.

11.3.2 Attributes

Please refer to the [XML Schema Definition](#) file.

11.3.3 Link Relations

Most if not all of the BonFIRE payloads will contain an unbounded number of `link` elements, with three important attributes. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi"
  href="/locations/uk-epcc/computes/1">
  ...
  <link
```

```
    rel="self"
    href="/locations/fr-inria/computes/123"
    type="application/vnd.bonfire+xml" />
<link
    rel="location"
    href="/locations/fr-inria"
    type="application/vnd.bonfire+xml" />
</compute>
```

Valid Link Attributes

In that example you can see that the payload contains two `link` elements. Each `link` has three attributes:

rel specifies the relationship between the current resource, and the resource targeted by the `link`. [MANDATORY]

href specifies the URI at which the target resource can be accessed. This is most of the time an HTTP URI (`http:` scheme), but it could be based on other schemes (`xmpp:`, `mailto:`, `ftp:`). [MANDATORY]

type specifies the content-type of the target resource. The value can be a comma-separated list of content-types. [OPTIONAL]

Valid Link Relationships

The valid link relation values for the `rel` attribute of the BonFIRE+XML media-type document are listed below:

self refers to the current resource.

parent refers to the first parent resource in the resource hierarchy.

next for a Collection, it refers to the list of items immediately next to the last item of the current representation.

prev for a Collection, it refers to the list of items immediately previous to the last item of the current representation.

users refers to the list of Users.

experiments refers to the list of Experiments.

locations refers to the list of Locations.

computes refers to the list of Compute resources.

networks refers to the list of Network resources.

storages refers to the list of Storage resources.

configurations refers to the list of compute Configuration types available at a specific Location.

services refers to the list of Service resources available at a specific Location.

location refers to the Location that resource belongs to.

experiment refers to the Experiment that resource belongs to.

11.3.4 Extensibility

This document describes the BonFIRE+XML markup vocabulary. Markup from other vocabularies (“foreign markup”) can be used in a BonFIRE+XML document. Any extensions to the BonFIRE+XML vocabulary MUST not redefine any elements, attributes, or link relations defined in this document. Clients that do not recognize extensions to the BonFIRE+XML vocabulary SHOULD ignore them.

The details of designing and implementing BonFIRE+XML is beyond the scope of this document.

Note: It is possible that future forward-compatible modifications to this specification will include new elements, attributes, and link-relations. Extension designers should take care to prevent future modifications from breaking or redefining those extensions. The safest way to do this is to use a unique XML Namespace for each extension.

11.4 Testbed Specificities

11.4.1 Deviations of the infrastructure sites for OCCI implementation

BonFIRE infrastructure sites provide cloud interface for the management and monitoring of the virtual resources. This interface is based on the Open Cloud Computing Interface ([OCCI](#)). OCCI describes only three kinds relating to management of cloud infrastructures: compute, network and storage. In BonFIRE there are three sorts of underlying infrastructure which are homogenized via OCCI on top of them. They are:

- OpenNebula: provided by * The University of Edinburgh, United Kingdom (EPCC), * Institut National de Recherche en Informatique et en Automatique, France (INRIA), * Universität Stuttgart, Germany (USTUTT-HLRS).
- Virtual Wall: provided by * Interdisciplinary Institute for Broadband Technology, Belgium (IBBT);
- Cells: provided by * Hewlett-Packard Labs, United Kingdom (HP);

BonFIRE decided to adopt OCCI as its standard interface to access the different infrastructure sites in a seamless manner for experimenters. However because of different semantics or working model in each underlying system, some deviations between them for OCCI implementation need to be taken into account by experimenters in describing their experiments. Since OpenNebula's OCCI implementation can be considered as the *de facto* standard implementation with regards to data types, the following three tables present OCCI/OpenNebula, Cells' and VW's deviations from this *de facto* standard OCCI in storage, network and compute resource.

Table 11.1: Storage

OCCI Element/Attribute	Cells vs OpenNebula	VW vs Open-Nebula
href	Not used	NA
id	Mandatory unique id of the disk	NA
name	Same usage	NA
description	Same usage	NA
type	Same usage. OS tells cells to mount the partition in /dev/hda making it bootable. DATABLOCK AND CDROM are equally treated (a device in the specified VM will be filled with data the users will have to mount themselves).	NA
size	Same usage	NA
fstype	Not used	NA
url	Used in "standard" OpenNebula 2.0. Not included in Bonfire's OCCI schema but needed when an image repository is not offered by the site. Users need to provide an image from a downloadable location for Cells to get it and create the volume.	NA

Table 11.2: Network

OCCI Element/Attribute	Cells vs OpenNebula	VW vs OpenNebula
id	Mandatory unique ID of the network	Not used - generated by the database
size	Not used	Same usage
name	Not used	Same usage
address	Not used	Same usage
public	Overwritten: in order to comply with the network architecture and the fact that a VM can only have a single vNIC in cells all the networks are public for Bonfire users	Not used
lossrate (VW specific)	NA	Value between 0 and 1
latency (VW specific)	NA	Value in ms
bandwidth (VW specific)	NA	Value in Mbps

Table 11.3: Compute

OCCI Element/Attribute	Cells vs OpenNebula	VW vs OpenNebula
href	Not used	Not used
id	Mandatory unique ID of the VM	Same usage - generated by the database
name	Same usage	Same usage
instance_type	Same usage	Not used
disk	Same usage	Not used
disk storage	Same usage	Same usage
href		
disk type	Not used	Not used
disk target	Not used	Not used
nic	Same usage	Same usage
nic network	Same usage	Same usage
href		
nic ip	Not used	Same usage
nic mac	Not used	Not used
state	Same usage. “suspended, cancel, shutdown, off” map into off state in cells and “on and resume” map into “on”.	It is there but does not yet reflect the actual state of the node.
context (key-value pairs)	Same usage. Accessible as a file in a virtual CDROM with no “interpolation” capabilities.	Same usage.

11.5 Deploying resources on Amazon EC2

11.5.1 Overview

Since the release 3, BonFIRE supports the creation of resources in Amazon EC2. To do it, the Enactor's Amazon connector transforms the OCCI data it receives to calls to the Amazon API.

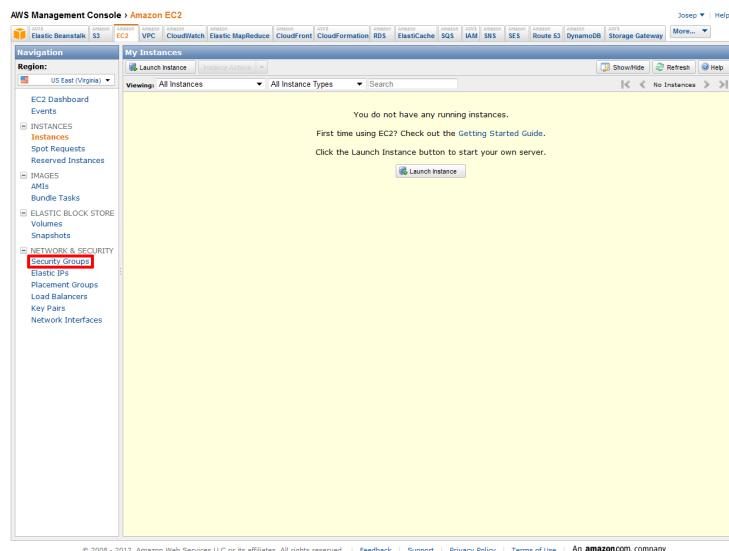
Since the launch of the third release of BonFIRE, an Amazon connector to interact to EC2 services is available. BonFIRE includes a subset of the features of that service providing a way to manage VMs in the same way you could do with the other testbeds. It eases the task of managing various resources distributed on many services providing a single interface.

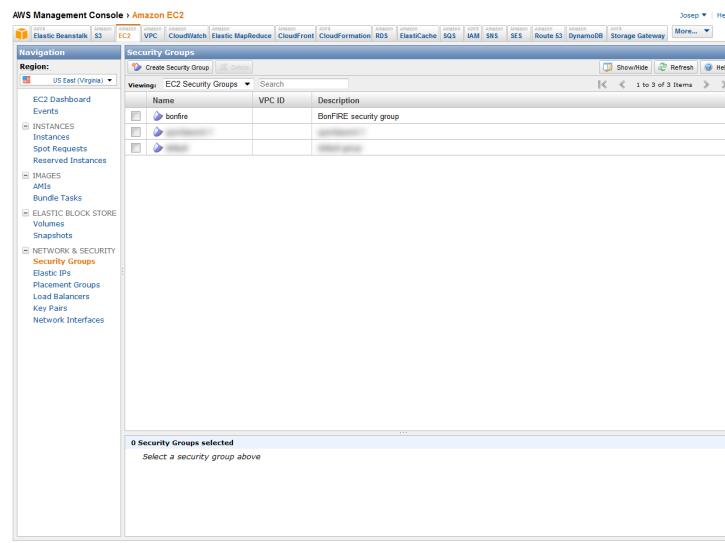
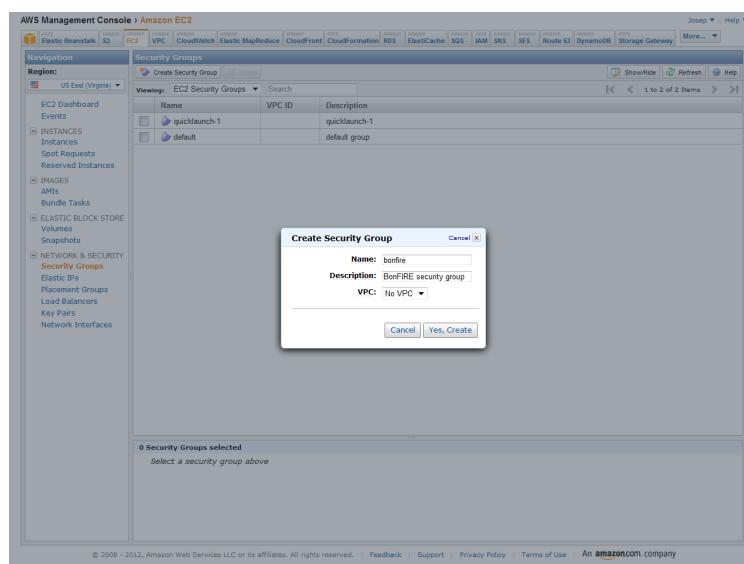
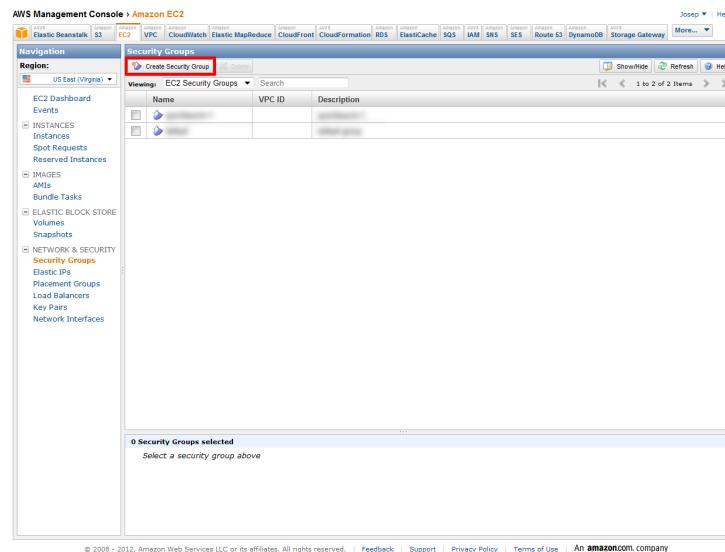
The set of features provided by the BonFIRE's Amazon connector includes creating compute instances, create storages (Amazon volumes) to its later use in compute instances, save snapshots of running compute instances and view detailed information of compute and storage resources.

11.5.2 Previous steps

BonFIRE does not provide Amazon's credential management. So, In order to be able to use the interconnector, you have to be registered in Amazon Web Services.

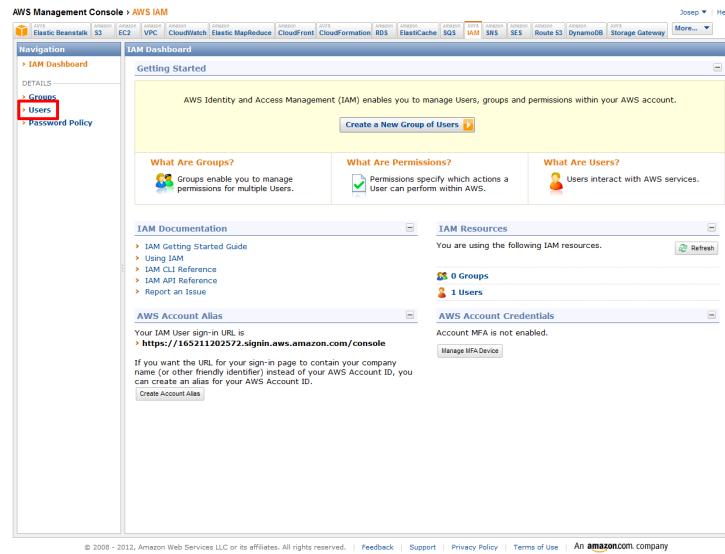
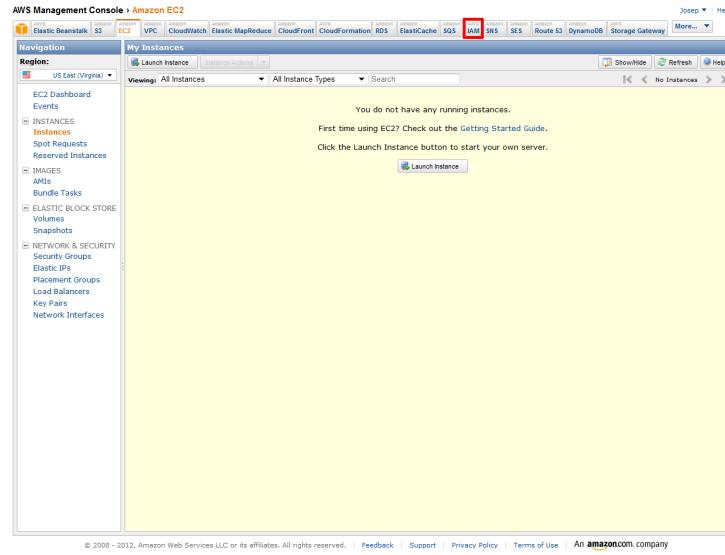
Then, you have to create a security group called "bonfire" from the AWS console:

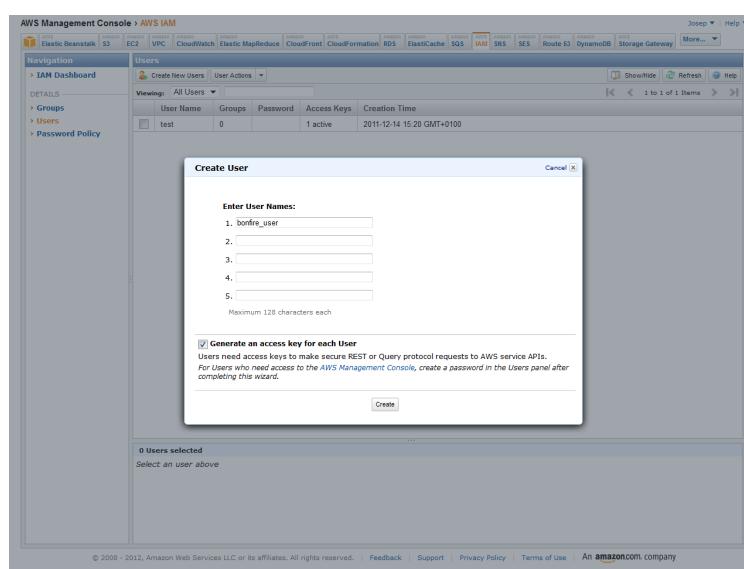
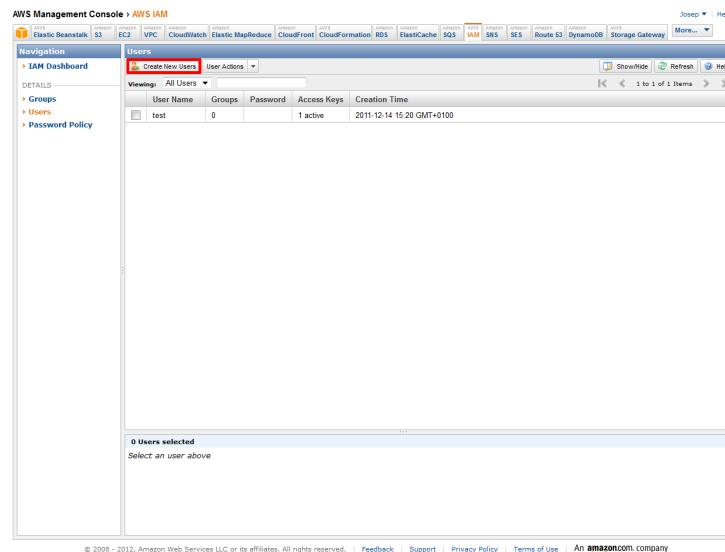




The next step to get your Amazon Web Services account configured to get petitions from BonFIRE is to create a user with full access to EC2, write down the access keys (both public and secret) and append the following code to the policy document:

```
{
  "Effect": "Allow",
  "Action": "iam:GetUser",
  "Resource": "*"
}
```

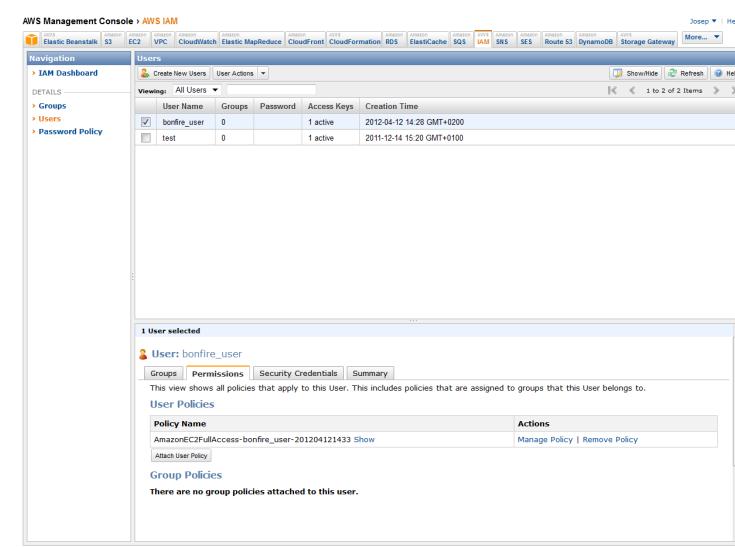
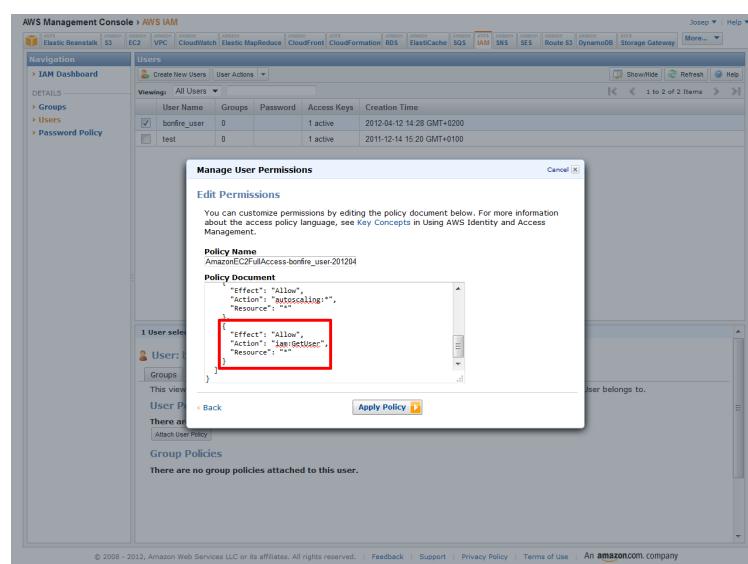
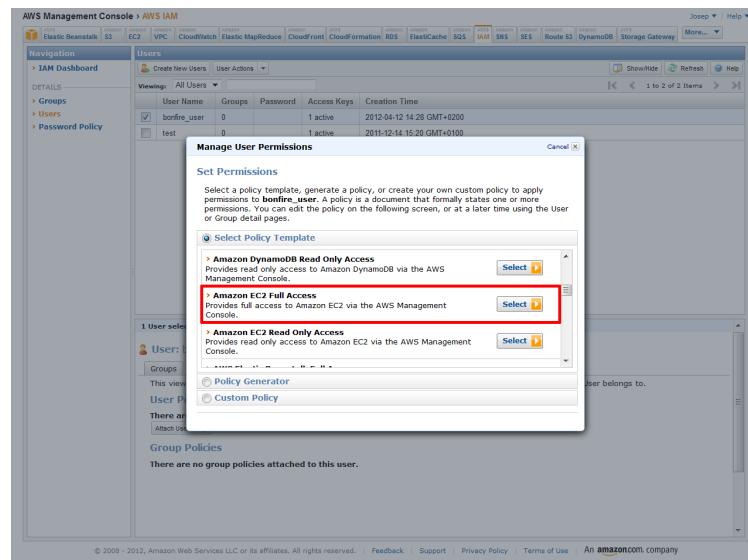




The screenshot shows the AWS Management Console IAM Users page. There are two users listed: 'bonfire_user' and 'test'. The 'bonfire_user' row has a checkmark next to it. A modal window titled 'Create User' is overlaid on the page, containing the message 'Your 1 User has been created successfully.' and a section for 'User Security Credentials' which lists 'bonfire_user' with 'Access Key Id:' and 'Secret Access Key:' fields. A 'Download Credentials' button is present.

User Name	Groups	Password	Access Keys	Creation Time
bonfire_user	0	1 active	2012-04-12 15:05 GMT+0200	
test	0	1 active	2011-12-14 15:20 GMT+0100	

The screenshot shows the AWS Management Console IAM User details page for 'bonfire_user'. The 'Permissions' tab is selected. It displays the message 'There are no policies attached to this user.' and features a prominent red-bordered 'Attach User Policy' button. Other tabs include 'Groups', 'Security Credentials', and 'Summary'.



With these steps, you will have your account set up to work with BonFIRE. Take in account that you will be asked for the user's keys when creating an experiment in BonFIRE.

11.5.3 Interface

The Amazon connector uses the Java API to interact with the infrastructure and manage resources. Anyway, to fully fulfill the BonFIRE requirements, the interaction with the rest of the project's infrastructure is handled by OCCI-compliant messages. Below some examples of the OCCI interaction with the connector to manage resources in Amazon are presented.

Storages

List storages

Storage resources in Amazon are persistent, and can be public or private. This has made that a lot of public storage resources are available, each one with its own customization. Within this large set of items the "official" Amazon storages can be found. The whole list of public storages is so huge (11813 objects at the moment of writing this), that it was decided to provide a filter to distinguish the Amazon images and the user's ones.

The usual way of retrieving storages in all the sites:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/storages
```

...returns the whole list of storages:

```
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
<items>
  <storage href="/locations/useast-aws/storages/ami-000af969" name="null" platform="null" owner="n...
  <storage href="/locations/useast-aws/storages/ami-0011e069" name="clovr-standard-2011-01-07-16-0...
  <storage href="/locations/useast-aws/storages/ami-0017b369" name="clovr-standard-2012-05-15-03-0...
  <storage href="/locations/useast-aws/storages/ami-0022c769" name="null" platform="null" owner="n...
  <storage href="/locations/useast-aws/storages/ami-002bf169" name="null" platform="null" owner="n...
  <storage href="/locations/useast-aws/storages/ami-002dd269" name="xsd.web.server" platform="windo...
  ...
  <storage href="/locations/useast-aws/storages/ami-fff83296" name="Windows2003_Apply" platform="wi...
  <storage href="/locations/useast-aws/storages/ami-ffffd3796" name="inspection2" platform="null" o...
  <storage href="/locations/useast-aws/storages/ami-ffffe2a96" name="vnoc-images-2012-01-01_04-48-5...
</items>
</collection>
```

In order to get only the Amazon images the following command is needed:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/storages/...
```

And, for the user's own images:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/storages/...
```

And, of course, the information retrieving of a single one is performed as usual:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/storages/...
```

Retuning:

```
<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/useast-aws/storages/...
  <id>ami-ffffe2a96</id>
  <name>vnoc-images-2012-01-01_04-48-52/new_name</name>
```

```
<type>OS</type>
<description>10-82-43-216//Daily</description>
<public>YES</public>
<persistent>YES</persistent>
</storage>
```

Create storages

The creation of storages in Amazon is managed by the Elastic Block Storage system. The tools Amazon provides, allows the specification of the size, the availability zone where it will be created, the possibility of using a previously saved snapshot (public images) and some other options.

In order to maximise simplicity, we decided to limit this creation options and make them match, as much as possible, the storage creation in the rest of BonFIRE sites. A curl command to the enactor for creating a storage looks as follows:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/experiments/<experiment id>/storage
      -X POST -d '<storage xmlns="http://api.bonfire-project.eu/doc/schemas/occi">\n        <name>AmazonStorage02</name>\n        <groups>sgarcia</groups>\n        <description>AmazonStorage description</description>\n        <type>DATABLOCK</type>\n        <size>2048</size>\n        <fstype>ext3</fstype>\n        <persistent>YES</persistent>\n        <link href="/locations/useast-aws" rel="location"/>\n    </storage>' \
      --header Content-Type:application/vnd.bonfire+xml'
```

Where the units for the size are MB and the host is optional (if no host is provided, the first one available is chosen).

Computes

In the same way than the storage creation, the management of Amazon computes in BonFIRE tries to follow the most the management in other testbeds.

List configurations

The retrieving of the compute types can be achieved with the following command:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/configurations
```

Then, a OCCI payload as the following one is returned:

```
<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
  <configuration>
    <vcpu>1</vcpu>
    <vmem>1740.8</vmem>
    <name>m1.small</name>
  <configuration>
  <configuration>
    <vcpu>2</vcpu>
    <vmem>3840</vmem>
    <name>m1.medium</name>
  <configuration>
```

```

<configuration>
    <vcpu>4</vcpu>
    <vmem>7680</vmem>
    <name>m1.large</name>
<configuration>
<configuration>
    <vcpu>8</vcpu>
    <vmem>15360</vmem>
    <name>m1.xlarge</name>
<configuration>
<configuration>
    <vcpu>1</vcpu>
    <vmem>613</vmem>
    <name>t1.micro</name>
<configuration>
    ...
</collection>

```

List of computes

As well as the rest of BonFIRE site, a command like the following one allows the retrieving of the current computes at Amazon:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/computes
```

Then, a OCCI payload as the following one is returned:

```

<collection xmlns="http://api.bonfire-project.eu/doc/schemas/occi">
    <items>
        <compute href="/locations/useast-aws/computes/i-e9e1c592" name="Amazon instance"/>
    </items>
</collection>

```

If, moreover, we provide the id of an specific one:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/computes/i-e9e1c592
```

We get a payload with its information:

```

<compute xmlns="http://api.bonfire-project.eu/doc/schemas/occi" href="/locations/useast-aws/computes/i-e9e1c592">
    <id>i-e9e1c592</id>
    <name>Amazon instance</name>
    <type>m1.small</type>
    <state>RUNNING</state>
    <disk>
        <storage href="/locations/useast-aws/storages/ami-31814f58" />
        <type>OS</type>
        <target>/dev/sda1</target>
    </disk>
    <disk>
        <storage href="/locations/useast-aws/storages/vol-c60cecb" />
        <type>DATABLOCK</type>
        <target>/dev/sdb</target>
    </disk>
    <nic>
        <network href="/locations/useast-aws/networks/dummy" name="Public Network" />
        <ip>ec2-67-202-15-241.compute-1.amazonaws.com</ip>
    </nic>
</compute>

```

```
</nic>
</compute>
```

Here we have to note an important point. Amazon provides automatically a public IP for each of the deployed machines, but no specific network is defined.

To match also as much as possible the rest of the infrastructure, the enactor maps the Amazon compute states to the following ONE ones: PENDING, RUNNING, SHUTDOWN, DONE, EPILOG-STOP and STOPPED.

Creation of computes

To create a compute the following command can be sent to the enactor:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/experiments/<experiment id>/computes
-X POST -d "<?xml version='1.0' encoding='UTF-8'?>\n    <compute xmlns='http://api.bonfire-project.eu/doc/schemas/occi'>\n        <name>Amazon instance</name>\n        <description>Description</description>\n        <instance_type>t1.micro</instance_type>\n        <host>us-east-1a</host>\n        <disk>\n            <storage href='/locations/useast-aws/storages/ami-82fa58eb' />\n            <type>OS</type>\n        </disk>\n        <disk>\n            <storage href='/locations/useast-aws/storages/vol-922bc9e8' />\n            <target>/dev/sdc1</target>\n            <type>DATABLOCK</type>\n        </disk>\n        <disk>\n            <storage href='/locations/useast-aws/storages/vol-2028ca5a' />\n            <target>/dev/sdd1</target>\n            <type>DATABLOCK</type>\n        </disk>\n        <link href='/locations/useast-aws' rel='location' />\n    </compute>" \
--header Content-Type:application/vnd.bonfire+xml
```

Noting that the DATABLOCK storages, the description and the host are optional. If no host is provided, the first one available is chosen.

Saving VMs images

As well as the other BonFIRE sites, the Amazon connector allows the user to save an image of a running VM to its future use. To request this, a command like the next one should be used:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/computes/
-X PUT -d "<compute xmlns='http://api.bonfire-project.eu/doc/schemas/occi'>\n    <disk>\n        <save_as name='TEST_SAVE_AS_OCCI' />\n    </disk>\n</compute>" \
--header Content-Type:application/vnd.bonfire+xml
```

Shutdown a compute

To shutdown a compute in Amazon, the only action needed is changing its state with a PUT command:

```
curl -vki --user <user_id> http://api.integration.bonfire.grid5000.fr/locations/useast-aws/computes/
-X PUT -d "<compute xmlns='http://api.bonfire-project.eu/doc/schemas/occi'>\n
<state>SHUTDOWN<state>\n
</compute>" \
--header Content-Type:application/vnd.bonfire+xml
```

11.6 Instance Types

The different infrastructure sites in BonFIRE offer heterogeneous resources. Therefore, to allow a uniform taxonomy of the specification of the VMs that can be deployed, we have adopted a common set of instance types as listed below. To see how these instance types are used when you deploy compute resources in BonFIRE, have a look at the [Portal documentation](#).

Table 11.4: Taxonomy of Instance Types in BonFIRE

Name	CPU cores	Mem-ory	Features	
lite	0.5	256MB		
small	1	1GB		
medium	2	2GB		
large	2	4GB		
large+	2	4GB	Higher CPU clock speed (over 3GHz)	
large-en	4	4GB	Emulated network	It is a minimum and may not be exactly what the user gets
xlarge	4	8GB		
xlarge+	4	8GB	Higher CPU clock speed (over 3GHz)	
Custom	user defined	user defined	VCPUs must be an integer	

Each site/testbed in BonFIRE can support multiple instance types, as well as being able to specify custom instance types. An overview of what is supported for each site is provided below.

Table 11.5: Instance types supported on each testbed

Site	lite	small	medium	large	large+	large-en	xlarge	xlarge+	custom	CPU=VCPUs?
EPCC	+	+	+	++			++		+	Yes (except lite CPUs)
HLRS	+	+	+	+	++		+	++	+	Yes
HP CELLS	+	+	+	+			+			
IBBT VW						++				Yes
INRIA	+	+	+						+	Yes
PSNC	+	+	+	+			+			

Legend +: Possible; ++: Preferred; blank: Not possible.

11.7 VM Images

The BonFIRE testbeds provide the following virtual machine images:

- BonFIRE Debian Squeeze v5
- BonFIRE Debian Squeeze 2G v5
- BonFIRE Debian Squeeze 10G v5
- BonFIRE Zabbix Aggregator v7
- BonFIRE Load Balancer v2
- Customized vm image

11.7.1 BonFIRE Debian Squeeze Virtual Machine Images

These images are based on Debian Squeeze.

BonFIRE Debian Squeeze v5

- Type: OS
- Size: 820MB
- Filesystem: None

BonFIRE Debian Squeeze 2G v5

- Type: OS
- Size: 2049MB
- Filesystem: None

BonFIRE Debian Squeeze 10G v5

- Type: OS
- Size: 10300MB
- Filesystem: None

11.7.2 BonFIRE Zabbix Aggregator Virtual Machine Image

The Zabbix Aggregator is a monitoring server provided by BonFIRE. The Aggregator image is based on the BonFIRE Debian Squeeze base image with preinstalled Zabbix software.

BonFIRE Zabbix Aggregator v7

- Type: OS
- Size: 6144MB
- Filesystem: None

11.7.3 BonFIRE Load Balancer VMI

BonFIRE provides a load balancer vm image for the users who want to use Elasticity as a service.

BonFIRE Load Balancer v2

- Type: OS
- Size: 1500MB
- Filesystem: None

11.7.4 Customized Virtual Machine Image

Users can create a new VM image based on a BonFIRE Debian Squeeze. For more information about how to do this, please see the [Configuring Software and Saving VMs](#) page.

11.8 XML Schema Definition

Here is the XSD schema that describe all the elements and attributes supported in the *Media Type*. The schema is splitted into 2 files. You can download the first file, and second file .

The following is the first file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified" elementFormDefault="qualified">
  <xs:element name="experiment">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:user_id" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:walltime" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:status" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:routing_key" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:aggregator" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:networks" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:computes" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:storages" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:routers" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:site_links" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:aws_access" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:aws_secret" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:groups" minOccurs="0" maxOccurs="1"/>
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:reservation" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="href" use="optional"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="network">
    <xs:complexType mixed="true">
      <xs:choice maxOccurs="unbounded" minOccurs="0">
        <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:address">
          <xs:annotation>
```

```
<xs:documentation>&gt; <cidr>10.0.0.1/24</cidr> //</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:size">
  <xs:annotation>
    <xs:documentation>&gt; <cidr>/24</cidr> //</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:visibility">
  <xs:annotation>
    <xs:documentation>&lt;cidr>10.0.0.1/24</cidr> &lt;cidr>10.0.0.2/24</cidr>
  </xs:annotation>
</xs:element>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:id"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:state"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:network_link"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:vlan"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:location">
  <xs:annotation>
    <xs:documentation>name&gt; Network name</name</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:groups"/>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:lossrate">
  <xs:annotation>
    <xs:documentation>VW specific - must be a percentage between 0 and 1</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:latency">
  <xs:annotation>
    <xs:documentation>VW specific - milliseconds</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:bandwidth">
  <xs:annotation>
    <xs:documentation>VW specific - Mbps</xs:documentation>
  </xs:annotation>
</xs:element>
</xs:choice>
<xs:attribute type="xs:string" name="href" use="optional"/>
<xs:attribute type="xs:string" name="name" use="optional"/>
</xs:complexType>
</xs:element>
<xs:element name="storage">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:groups" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:type" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:size" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:fstype" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:persistent" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:public" minOccurs="0"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:state" minOccurs="0"/>
```

```

<xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" minOccurs="0">
  </xs:sequence>
  <xs:attribute type="xs:string" name="href" use="optional"/>
  <xs:attribute type="xs:string" name="name" use="optional"/>
</xs:complexType>
</xs:element>
<xs:element name="compute">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" minOccurs="0">
        <xs:annotation>
          <xs:documentation>http://tracker.bonfire-project.eu/issues/469</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" minOccurs="0">
        <xs:annotation>
          <xs:documentation>This can contain any free-form element This can contain any free-form element</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:instance_type" minOccurs="0">
        <xs:annotation>
          <xs:documentation>VWall specific: maybe this could be included in &lt;context&gt; element</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" minOccurs="0">
        </xs:sequence>
        <xs:attribute type="xs:string" name="href" use="optional"/>
        <xs:attribute type="xs:string" name="name" use="optional"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="location">
      <xs:complexType mixed="true">
        <xs:sequence>
          <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" minOccurs="0">
            <xs:annotation>
              <xs:documentation>http://api.bonfire-project.eu/doc/schemas/occi</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" minOccurs="0">
            <xs:annotation>
              <xs:documentation>http://api.bonfire-project.eu/doc/schemas/occi</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:longitude" minOccurs="0">
            <xs:annotation>
              <xs:documentation>http://api.bonfire-project.eu/doc/schemas/occi</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:latitude" minOccurs="0">
            <xs:annotation>
              <xs:documentation>http://api.bonfire-project.eu/doc/schemas/occi</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" maxOccurs="1" minOccurs="0">
            <xs:annotation>
              <xs:documentation>http://api.bonfire-project.eu/doc/schemas/occi</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
        <xs:attribute type="xs:string" name="href" use="optional"/>
        <xs:attribute type="xs:string" name="name" use="optional"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:element>

```

```
<xs:element name="router">
  <xs:complexType mixed="true">
    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:id"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:state"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:host"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:interface"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:config"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link"/>
    </xs:choice>
    <xs:attribute type="xs:string" name="href" use="optional"/>
    <xs:attribute type="xs:string" name="name" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="site_link">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:id" minOccurs="0" maxOccurs="1" name="id"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" minOccurs="0" maxOccurs="1" name="name"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" minOccurs="0" maxOccurs="1" name="description"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:endpoint" minOccurs="0" maxOccurs="1" name="endpoint"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:bandwidth" minOccurs="0" maxOccurs="1" name="bandwidth"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:state" minOccurs="0" maxOccurs="1" name="state"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" minOccurs="0" maxOccurs="1" name="link"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="href" use="optional"/>
    <xs:attribute type="xs:string" name="name" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="items">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:experiment" minOccurs="0" maxOccurs="1" name="experiment"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:network" minOccurs="0" maxOccurs="1" name="network"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:storage" minOccurs="0" maxOccurs="1" name="storage"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:compute" minOccurs="0" maxOccurs="1" name="compute"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:location" minOccurs="0" maxOccurs="1" name="location"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:router" minOccurs="0" maxOccurs="1" name="router"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:site_link" minOccurs="0" maxOccurs="1" name="site_link"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:configuration" minOccurs="0" maxOccurs="1" name="configuration"/>
    </xs:sequence>
    <xs:attribute type="xs:byte" name="offset" use="optional"/>
    <xs:attribute type="xs:byte" name="total" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="link">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="rel" use="optional"/>
        <xs:attribute type="xs:string" name="href" use="optional"/>
        <xs:attribute type="xs:string" name="type" use="optional"/>
        <xs:attribute type="xs:string" name="ref" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```

<xs:element name="collection">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:items"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" maxO
    </xs:sequence>
    <xs:attribute type="xs:string" name="href" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="save_as">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="name" use="optional"/>
        <xs:attribute type="xs:string" name="href" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="type" type="xs:string"/>
<xs:element name="target">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="href" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="device" type="xs:string"/>
<xs:element name="ip" type="xs:string"/>
<xs:element name="mac" type="xs:string"/>

<xs:element name="name" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="instance_type" type="xs:string"/>
<xs:element name="disk">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:save_as"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:storage"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:type"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:target"/>
    </xs:choice>
    <xs:attribute type="xs:byte" name="id" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="nic">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:network"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:device" minO
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:ip" minOcu
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:mac" minOcu
    </xs:sequence>
    <xs:attribute type="xs:byte" name="id" use="optional"/>
  </xs:complexType>
</xs:element>

```

```
</xs:element>
<xs:element name="context"><xs:complexType><xs:sequence><xs:any processContents="lax" minOccurs="0">
<xs:element name="startup">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="href" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="state" type="xs:string"/>
<xs:element name="cpu" type="xs:float"/>
<xs:element name="vcpu" type="xs:float"/>
<xs:element name="memory" type="xs:short"/>
<xs:element name="groups" type="xs:string"/>
<xs:element name="cluster" type="xs:string"/>
<xs:element name="host">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:location" minOccurs="0">
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:cluster" minOccurs="0">
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" minOccurs="0">
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:count" minOccurs="0">
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="resource_set" type="xs:string"/>
<xs:element name="shared_allocation" type="xs:string"/>
<xs:element name="best_effort_allocation" type="xs:string"/>
<xs:element name="scheduling_id" type="xs:string"/>
<xs:element name="vmem" type="xs:short"/>
<xs:element name="configuration">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:vcpu" minOccurs="0">
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:vmem" minOccurs="0">
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="user_id" type="xs:string"/>
<xs:element name="walltime" type="xs:string"/>
<xs:element name="status" type="xs:string"/>
<xs:element name="routing_key" type="xs:string"/>
<xs:element name="aggregator_password" type="xs:string"/>
<xs:element name="networks">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:network" minOccurs="0">
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="computes">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:compute" minOccurs="0">
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
</xs:complexType>
</xs:element>
<xs:element name="storages">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:storage" maxOccurs="unbounded" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="routers">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:router" maxOccurs="unbounded" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="site_links">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:site_link" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="aws_access_key_id" type="xs:string"/>
<xs:element name="aws_secret_access_key" type="xs:string"/>
<xs:element name="reservation">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:id" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:status" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:user" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:group" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:walltime" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:starttime" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:resources" type="xs:string"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="href" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="longitude" type="xs:float"/>
<xs:element name="latitude" type="xs:float"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="size" type="xs:string"/>
<xs:element name="visibility" type="xs:string"/>
<xs:element name="router_interface" type="xs:string"/>
<xs:element name="endpoint">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:router" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:router_interface" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:location" minOccurs="1" maxOccurs="1" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:vlan" minOccurs="1" maxOccurs="1" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="id" type="xs:string"/>
```

```
<xs:element name="network_link">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:endpoint" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="vlan">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute type="xs:string" name="rel" use="optional"/>
        <xs:attribute type="xs:string" name="href" use="optional"/>
        <xs:attribute type="xs:string" name="type" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="lossrate" type="xs:float"/>
<xs:element name="latency" type="xs:short"/>
<xs:element name="bandwidth" type="xs:string"/>
<xs:element name="physical_interface" type="xs:string"/>
<xs:element name="connected_to">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:host" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:physical_interface" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="interface">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:connected_to" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:physical_interface" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:ip" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:netmask" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="physical_router">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:interface" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="physical_node">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:name" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:description" type="xs:string"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:interface" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

</xs:element>
<xs:element name="physical_infrastructure">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:physical_r...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:physical_n...
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="count" type="xs:byte"/>
<xs:element name="scheduling_ids">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:scheduling...
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="instance">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:resource_se...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:instance_ty...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:cpu" minOco...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:memory" minO...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:location"/>...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:cluster" minO...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:host" minO...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:count"/>...
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:scheduling_...
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="user" type="xs:string"/>
<xs:element name="group" type="xs:string"/>
<xs:element name="starttime" type="xs:dateTime"/>
<xs:element name="resources">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:instance" r...
        <xs:annotation>
          <xs:documentation>instances      create two instance on different hosts      create two instan...
        </xs:annotation>
      </xs:element>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:host" maxO...
        <xs:annotation>
          <xs:documentation>hosts      reserve 3 hosts at Inria      reserve 3 hosts at Inria on a specific...
        </xs:annotation>
      </xs:element>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:core" maxO...
        <xs:annotation>
          <xs:documentation>cores      reserve 3 cores at Inria      reserve 3 cores at Inria on a specific...
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="core">
  <xs:complexType>

```

```
<xs:sequence>
  <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:location"/>
  <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:cluster" minOc
  <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:host" minOc
  <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:count"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="version" type="xs:string"/>
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:version"/>
      <xs:element xmlns:occi="http://api.bonfire-project.eu/doc/schemas/occi" ref="occi:link" maxOc
    </xs:sequence>
    <xs:attribute type="xs:string" name="href"/>
  </xs:complexType>
</xs:element>
<xs:element name="netmask" type="xs:string"/>
<xs:element name="config" type="xs:string"/>
<xs:element name="fstype" type="xs:string"/>
<xs:element name="public" type="xs:string"/>
<xs:element name="persistent" type="xs:string"/>
</xs:schema>
```

The second file is below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="location">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="href" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="description" type="xs:string"/>
  <xs:element name="endpoint">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="location"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="bandwidth" type="xs:string"/>
  <xs:element name="link">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="rel"/>
          <xs:attribute type="xs:string" name="href"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="site_link">
```

```

<xs:complexType>
  <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="description"/>
    <xs:element ref="endpoint" maxOccurs="unbounded" minOccurs="0"/>
    <xs:element ref="bandwidth"/>
    <xs:element ref="link"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

11.9 Experiment Lifecycle

In BonFIRE, an experiment has the following lifecycle:

- All experiments start in the `ready` state.
- When you're finished adding resources for your initial deployment configuration, you should pass the experiment status to `running`. This is mandatory if you're using VirtualWall resources (`be-ibbt` testbed), otherwise your experiment won't be swapped in there.
- When the experiment `walltime` is expired, the experiment will transition into the `stopped` state, which is a transitory state between `running` and `terminated`. At this point no resources are deleted, but you should use the time interval between `stopped` and `terminated` states to launch backup or specific shutdown procedures.
- Let T_0 be the time at which the experiment goes into the `stopped` state. At $T_1 = T_0 + 300$ seconds, a shutdown procedure will be called so that all compute resources are cleanly shut down before they are deleted (to ensure proper image backup if you chose to save a VM image as a new one – see [Configuring Software and Saving VMs](#)). This means you have 300 seconds for your specific backup or shutdown procedure to occur, before the forced shutdown procedure. Let us know if this is too short.
- The shutdown procedure goes on for another 300 seconds, after which experiment resources are destroyed, and the experiment will transition into the `terminated` state.
- If the user chooses to cancel the experiment (via a `DELETE` request), then the experiment goes directly into the `terminated` state, i.e. all resources are destroyed. The `stopped` event is not generated, and the shutdown procedure is not launched.

11.10 States of OCCI Resources

In BonFIRE OCCI, the resource descriptions for the COMPUTE, NETWORK (only at the Virtual Wall) and STORAGE resources include a state attached. The content of this is determined by the virtualization manager and so will differ between testbeds. Where log files are available, the same states will be used. The following is a list of the states on the different virtualization managers used in BonFIRE:

11.10.1 OpenNebula

The COMPUTE resource can be in the following states:

(*) where the ACTIVE state get unfolded in

PROLOG	--> The images of the VM are being staged to the node
BOOT	--> The VM is booting
RUNNING	--> The VM is running
MIGRATE	--> The VM is being migrated
SAVE_STOP	--> The VM is in process of being stopped
SAVE_SUSPEND	--> The VM is in process of being suspended
SAVE_MIGRATE	--> The VM is in process of saving because of it being migrated
PROLOG_MIGRATE	--> The VM is having its files staged in the target host due to migration
PROLOG_RESUME	--> The VM is having its files staged in the target host due to resuming
EPILOG_STOP	--> The VM is having its files staged into the front-end due to stopping
EPILOG	--> The VM is having its files staged into the front-end due to ending lifecycle
SHUTDOWN	--> The VM is in process of shutting down
CANCEL	--> The VM is in process of being canceled
FAILURE	--> The VM has reached a fail state in its VM lifecycle manager
CLEANUP	--> Files of the VM are being cleaned
UNKNOWN	--> The VM has disappeared from the hypervisor

The STORAGE resource can be in the following states:

INIT	--> This state is no longer used, but may appear in some old log files.
LOCKED	--> The storage is being copied or created.
READY	--> The storage is ready to be used.
USED	--> The storage is used by at least one VM. It may still be used by other VMs.
USED_PERS	--> The persistent storage is used by a VM. It cannot be used by other VMs.
DISABLED	--> The storage is disabled at the site level and cannot be used.
ERROR	--> An error occurred. The storage cannot be used.
DELETE	--> The image is being deleted.

11.10.2 VirtualWall

A quick remark. The Virtual Wall deploys all resources involved with a particular EXPERIMENT atomically. The states that are involved in this atomic action are indicated by an asterisk (*). The italic part between brackets - in the state's description - points to how this state is represented in the Virtual Wall log files (<http://ssh.be-ibbt.bonfire-project.eu/logs>). The COMPUTE resource can be in the following states:

PENDING	--> The COMPUTE is described by the experimenter and waiting for the scheduler or the EXPERIMENT
SWAPIN [1]	--> The Virtual Wall is preparing the EXPERIMENT
BOOTING [2]	--> The physical nodes that will harbor the COMPUTEs are prepared (*os_setup started ...)
ISUP	--> The COMPUTE is fully booted and accessible via the BonFIRE WAN or via the EXPERIMENT
BFSETUP	--> BonFIRE specific init scripts and the experimenter's postinstall scripts are executed
ACTIVE	--> Everything is set and running, the real EXPERIMENT can start
TERMINATING	--> The COMPUTE is shutting down as part of the EXPERIMENT shutdown, the COMPUTE is still available
TERMINATED	--> The COMPUTE is removed from the system and not accessible anymore
*Pre run	--> The EXPERIMENT description is validated and prepared for 'swap-in'. (*Beginning pre run*)
*Swap-in	--> The EXPERIMENT is being deployed to the Virtual Wall. (*Beginning swap-in*)
*Assign	--> The different resources are mapped to physical reality. (*assign_wrapper started ... as*)
*Save-as	--> When necessary the persistent STORAGES are saved and new images are created for the SAVING
*Swap-out	--> All physical resources (COMPUTES, NETWORKS and STORAGES) are freed. (*tbswap out started ... as*)

The NETWORK resource can be in the following states:

PENDING	--> The NETWORK is described by the experimenter and waiting for the scheduler or the EXPERIMENT
*SWAPIN [1]	--> The Virtual Wall is preparing the EXPERIMENT
UP	--> The NETWORK is available for packages to be sent by the connected COMPUTEs
DOWN	--> The NETWORK is brought down, i.e. every communication using this NETWORK will fail

```
*TERMINATING [2] --> The NETWORK is shutting down as part of the EXPERIMENT shutdown, communication
*TERMINATED --> The NETWORK is removed from the Virtual Wall gigabit switch
```

The STORAGE resource can be in the following states:

```
PENDING --> The STORAGE is described by the experimenter. STORAGES created outside the context
LINKED --> The STORAGE is linked to a COMPUTE
*SWAPIN [1] --> The Virtual Wall is preparing the EXPERIMENT
READY --> The STORAGE is mounted on its linked COMPUTE
*TERMINATING [2] --> The EXPERIMENT is shutting down. STORAGES with ``persistent=YES`` are finalized
*TERMINATED --> The STORAGE is removed from the system and not accessible anymore
```

11.10.3 HP Cells

The COMPUTE resource can be in the following states:

```
CREATED --> The COMPUTE has just been requested by the experimenter and Cells is in the pro
UP --> The COMPUTE is fully booted and accessible via it's attached networks. This auto
OFF --> The COMPUTE is shutdown.
STANDBY --> The COMPUTE is in a suspended state but can be resumed.
ERROR --> The COMPUTE is in an error state.
```

It possible for the user to request the SHUTDOWN, STANDBY and UP COMPUTE resource states.

The STORAGE resource can be in the following states:

```
READY --> The STORAGE is ready to be used.
LOCKED --> The STORAGE is currently being imaged following a save-as request. It's state w
ERROR --> The STORAGE is in an error state.
```

11.10.4 Wellness Telecom

The COMPUTE resource can be in the following states:

```
STOPPED --> The VM is stopped, waiting for resume
RUNNING --> The VM is running
SUSPENDED --> The VM is suspended, waiting for resume
PENDING --> The VM is performing an action.
```

There is only one possible state for STORAGES:

```
READY --> The storage has been successfully created.
```

However, as commented in the different Testbeds WT does not support DATABLOCKS. Therefore only OS disks can be used as storage. In this way, the maximum storage possible on VMs is 10 GB and on maximum storage possible on Aggregators is 6GB

11.11 BonFIRE OVF Schema Definition

Here is the XSD file that describe the BonFIRE OVF elements and attributes:

```
<?xml version="1.0" encoding="UTF-8"?>

<xss:schema xmlns:bonfire="http://api.bonfire-project.eu/doc/schemas/ovf/extension" xmlns:ovf="http://
  <xss:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="http://www.w3.org/2001/XMLSchema.xsd">
```

```
<xs:import namespace="http://schemas.dmtf.org/ovf/envelope/1" schemaLocation="http://schemas.dmtf.org/ovf/envelope/1.xsd" />
<xs:element name="NetworkMap">
    <xs:annotation>
        <xs:documentation>Usage: This element shall be used only at the extension point ovf:NetworkMap</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element name="NewNetwork" type="bonfire:NewNetworkType" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
        <xs:attribute name="ovfNetworkName" type="xs:string" />
        <xs:attribute name="bonfireNetworkRef" type="xs:string" />
        <xs:attribute name="location" type="xs:string" />
    </xs:complexType>
</xs:element>
<xs:complexType name="NewNetworkType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="Location" type="xs:string" />
        <xs:element name="Address" type="xs:string" />
        <xs:element name="Size" type="xs:string" />
        <xs:element name="Lossrate" type="xs:float" minOccurs="0" maxOccurs="1" />
        <xs:element name="Bandwidth" type="xs:int" minOccurs="0" maxOccurs="1" />
        <xs:element name="Latency" type="xs:int" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
<xs:element name="InstanceType" type="xs:string">
    <xs:annotation>
        <xs:documentation>Usage: This element shall be used only at the extension point ovf:InstanceType</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:attribute name="walltime" type="xs:int">
    <xs:annotation>
        <xs:documentation>Usage: This attribute shall be used only at the extension point ovf:walltime</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="location" type="xs:string">
    <xs:annotation>
        <xs:documentation>Usage: This attribute shall be used only at the extension point ovf:location</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="storagetype" type="xs:string">
    <xs:annotation>
        <xs:documentation>Usage: This element shall be used only at the extension point ovf:storagetype</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="ordertype">
    <xs:annotation>
        <xs:documentation>Usage: This element shall be used only at the extension point ovf:ordertype</xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:schema>
```

You can download this file [here](#).

11.12 Default Monitoring Metrics

11.12.1 VM Metrics

The following metrics are actively measured by default:

```
Buffers memory
Cached memory
CPU system time (avg1)
CPU nice time (avg1)
CPU idle time (avg1)
CPU iowait time (avg1)
CPU user time (avg1)
Free disk space on /
Free memory
Free swap space
Host boot time
Host status
Host uptime (in sec)
Incoming traffic on interface lo
Incoming traffic on interface eth1
Incoming traffic on interface eth0
Number of processes
Number of running processes
Number of users connected
Outgoing traffic on interface lo
Outgoing traffic on interface eth0
Outgoing traffic on interface eth1
Ping to the server (TCP)
Processor load
Shared memory
System cpu usage average
Total disk space on /
Total memory
Total swap space
Used disk space on /
Used disk space on / in %
```

The following metrics are disabled (are not measured by default), but can be enabled by BonFIRE user any time:

```
Checksum of /usr/sbin/sshd
Checksum of /usr/bin/ssh
Checksum of /vmlinuz
Checksum of /etc/services
Checksum of /etc/inetd.conf
Checksum of /etc/passwd
Email (SMTP) server is running
Free disk space on /usr
Free disk space on /var
Free disk space on /tmp
Free disk space on /home
Free disk space on /opt
Free disk space on /tmp in %
Free disk space on /var in %
```

Free disk space on /usr in %
Free disk space on / in %
Free disk space on /home in %
Free disk space on /opt in %
Free number of inodes on /usr
Free number of inodes on /tmp
Free number of inodes on /home
Free number of inodes on /
Free number of inodes on /opt
Free number of inodes on /tmp in %
Free number of inodes on / in %
Free number of inodes on /usr in %
Free number of inodes on /opt in %
Free number of inodes on /home in %
Free swap space in %
FTP server is running
Host information
Host local time
Host name
IMAP server is running
Maximum number of opened files
Maximum number of processes
News (NNTP) server is running
Number of running processes zabbix_server
Number of running processes zabbix_agentd
Number of running processes apache
Number of running processes inetd
Number of running processes mysqld
Number of running processes sshd
Number of running processes syslogd
POP3 server is running
Processor load5
Processor load15
Size of /var/log/syslog
SSH server is running
Temperature of CPU 1of2
Temperature of CPU 2of2
Temperature of mainboard
Total disk space on /home
Total disk space on /usr
Total disk space on /tmp
Total disk space on /opt
Total number of inodes on /usr
Total number of inodes on /
Total number of inodes on /opt
Total number of inodes on /home
Total number of inodes on /tmp
Used disk space on /usr
Used disk space on /var
Used disk space on /home
Used disk space on /tmp
Used disk space on /opt
Used disk space on /usr in %
Used disk space on /var in %
Used disk space on /tmp in %
Used disk space on /opt in %
Version of zabbix_agent(d) running
WEB (HTTP) server is running

11.12.2 Infrastructure Metrics

Information about a number of predefined metrics that are provided to BonFIRE experimenter about the physical machines hosting their VMs. BonFIRE solution allows each testbed to dynamically provide their own templates. Those of common interest are for example as follow:

```
Eth0 outgoing traffic
Eth0 incoming traffic
Running VMs
Processor load
Free swap space
Total memory
Free memory
Disk sda Write Bytes/sec
Disk sda Write: Ops/second
Disk sda IO ms time spent performing IO
Disk sda IO currently executing
Disk sda Read: Milliseconds spent reading
Disk sda Read: Ops/second
Disk sda Write: Milliseconds spent writing
Disk sda Read Bytes/sec
Ping to the server (TCP)
```

In addition to these, Inria, EPP, and HLRS sites provide further monitoring information about the energy usage and CO2 estimation. The related metrics are:

```
Aggregate energy usage / Wh
Apparent power usage / VA
CO2 produced
Power Consumption in W
Real power usage / W
```

11.13 Release Notes

The BonFIRE facilities in Release 1, Release 2, Release 3, Release 3.1 and Release 4.0.5 include the following features:

11.13.1 Portal

Features in Release 1

- The Portal can generate OCCI requests for the user
- The Portal publishes the resources of the BonFIRE sites
- The Portal lists the available Instance Types (i.e. categorised resource specifications)
- The Portal lists the available base VM images (i.e. preconfigured VM images with the Debian Squeeze OS and Zabbix monitoring framework)
- The Monitoring Aggregator VM can be auto-created for experiments defined on the BonFIRE Portal
- The Portal tunnels through to the monitoring data webserver

Features in Release 2

- Greatly improved user experience

Features in Release 3

- Support for JSON Experiment Descriptor, including builder
- Support for OVF
- Display of per-experiment usage statistics
- Linking of site-status and VM-logging information
- Support for Virtual Wall background traffic emulation
- Numerous usability changes

Features in Release 4.0.5

- The Portal has been revamped to better present user and group experiments and give users control over refresh of the output. It also exploits the new caching mechanism for faster resource listings and better support of large experiments.

11.13.2 Experiment Descriptor

Features in Release 1

- The user can interact with BonFIRE through the BonFIRE OCCI API
- The BonFIRE Portal can generate OCCI requests for the user
- Restfully can generate OCCI requests for the user, and also script the creation and steering of an experiment
- Elasticity: VMs can trigger VM instance creation and deletion through OCCI calls to the BonFIRE API

Features in Release 2

- Support for simple, domain-specific, Experiment Descriptors
- Support for user specifiable, near-end-of-experiment triggers to run scripts to e.g archive data
- Broker message-queue subscription for the VMs to run scripts

Features in Release 3

- Support for OVF
- Extended support for simple, domain-specific, Experiment Descriptor
 - Groups
 - IP dependencies
 - State reporting
 - Contextualisation

- Better error logging through provided URI
- Specify target cluster and host
- Specify Virtual Wall background traffic emulation
- Custom instance-type

Features in Release 4.0.5

- The new “aggregator” keyword simplifies initiating monitoring on managed experiments
- JSON Experiment Descriptor now supports the creation of persistent storage outside an experiment

11.13.3 Monitoring

Features in Release 1

- BonFIRE VM images come pre-installed with the Zabbix monitoring tool
- Zabbix collects standard monitoring data (e.g. CPU and memory utilisation) for each BonFIRE VM
- User-specified, application monitoring data (e.g. webserver hit-count) can be collected for each VM
- **Monitoring metrics can be set:** when creating a VM, through the OCCI interface on a running VM, using standard Zabbix procedures
- The Monitoring Aggregator VM can be auto-created for experiments defined on the BonFIRE Portal
- Clients can access the experiment monitoring data through the Zabbix json-rpc API
- The Portal tunnels through to the experiment’s Zabbix monitoring data webserver

Features in Release 2

- Extended specification of metric parameters through OCCI
- Use of separate storage for Aggregator
- User level infrastructure monitoring
- Monitoring API available at Broker

Features in Release 3

- Support for Zabbix Aggregator at iMinds
- user-specified application monitoring
- in depth-VM monitoring
- Timestamp internal processes (not though OCCI)
- Use Resource Manager-generated, experiment-specific password

Features in Release 4.0.5

- Experiment Message Queue includes events about user resources, which are generated by the sites to provide more detailed, more accurate information about the resources
- HLRS, Inria and EPCC expose a new set of infrastructure metrics about energy consumption and the CO2 profile of this energy in the BonFIRE-based STREP ECO2Clouds
- ECO-metrics together with the new events are available through Zabbix
- Numerous improvements for stability of Zabbix in our images, including upgrade to version 1.8.16 of Zabbix

11.13.4 Data access

Features in Release 1

- The user can specify storage resources for the duration of the experiment

The user must transfer all relevant experiment data out of their BonFIRE VMs prior to experiment expiry

Features in Release 2

- Distributed storage of persistent data (beyond experiment life)
- Distributed storage of VM Images

Features in Release 3

- Support for user quota
- Support for shared storage on BonFIRE NFS (IBBT)
- Support for datablocks (and external storage) on Virtual Wall (IBBT)

11.13.5 Network

Features in Release 1

- Extensive network emulation functionality is available on the BonFIRE Virtual Wall site
- All BonFIRE sites are connected with each other through best-effort internet
- VMs on different sites can communicate between them using the BonFIRE VPN, with the exception that VMs on the Virtual Wall cannot communicate with VMs on the HP Cells site

Features in Release 2

- Support for Virtual Wall network QoS parameters through OCCI

Features in Release 3

- VW background traffic emulation (via Emulab)

Features in Release 3.1

- Support for bandwidth on demand through GEANT AutoBAHN, between EPCC and PSNC

Features in Release 4.0.5

- Supports VPN access to its resources
- Easier to use Unix tools for network impairment on all infrastructures
- The Virtual Wall allows to select the queue strategy
- Increased number of public IP addresses at EPCC and Inria, along with removal of most of the restrictive firewall rules

11.13.6 Authentication, Authorisation and Accounting

Features in Release 1

- Access to the Portal and Broker is available through username-password authentication over SSL
- The user can access their VMs on all BonFIRE sites through SSH with a single set of public-key credentials
- The BonFIRE Broker uploads SSH credentials to VM
- Root access is available on all VMs running on the BonFIRE sites

Features in Release 3.1

- Support for accounting

11.13.7 Experiment Support

Features in Release 2

- Stopped state between Running and Terminated, to allow users to get data, clean up etc.
- Consistent contextualisation API for apps to read context
- Improved OCCI error handling
- Use of POSTINSTALL to execute a user-specified set-up script

Features in Release 3

- Groups-based shared access to resources
- Custom instance type specification
- Usage statistics
- Command line tools
- Rule-based elasticity (pre-configured images and doc)
- Support for VM deployment on user-specified physical host

Features in Release 3.1

- Cells and OpenNebula upgrades
- Improved groups functionality
- Improved Elasticity as a service (cross-site Elasticity)
- Interconnection with Amazon EC2
- Support for Open Access

Features in Release 4.0.5

- Support for Trial accounts, a much simpler, even faster process to get access to BonFIRE within a working day
- Improved performance of collection queries via a new Collections Cache component; also improves site performance via collection data from asynchronous site events
- Inria and iMinds (be-ipbb) include the new COCOMA image for allowing perform controlled experiments of multi-tenant scenarios (effects of VMs co-located on the same physical machine)
- COCOMA allows to generate COntrolled, COntentious and MAlicious pattern loads on the physical nodes where it is deployed, affecting resource such as CPU, RAM, disk and network
- Allows to study in a controlled and repeatable manner how this affects experimenters' own application

11.13.8 More capacity

Features in Release 4.0.5

- Wellness Telecom is on line
- HLRS has more than doubled its capacity to 344 cores
- EPCC has also all but doubled its capacity to 176 cores
- Inria upgrades to 96 cores

FAQ

12.1 FAQ

12.1.1 Are the clocks of the various compute resources synchronized? How?

The short answer is **yes**. We take reasonable measures to ensure that clocks of compute resources among any testbed are within 0.5 seconds (500 milliseconds) of a reference time obtained through the Network Time Protocol ([NTP](#)). Therefore you can assume that all your VMs are synchronized at plus/minus 0.5 seconds of the NTP server BonFIRE is using. BonFIRE central services are also synchronized using the same NTP server, which ensures that the whole BonFIRE infrastructure is in sync.

12.1.2 Can I describe my whole experiment deployment in a single document rather than creating it in the portal or via multiple OCCI calls?

Yes, this is possible with [Experiment Descriptors](#), using the Experiment Manager in BonFIRE. It is also possible by creating [Restfully](#) scripts if you are familiar with Ruby.

12.1.3 How do I control my experiment once it is running?

This depends what you mean by control. Some particular features in BonFIRE include the support for elasticity rules and dynamically changing the QoS of emulated networks on the Virtual Wall. Controlling when programs start and stop on the VMs in your experiment(s) is something that must explicitly be coded by you. For example, [Restfully](#) supports this in that it allows you to SSH into your VMs and run commands.

12.1.4 How can I pass on configuration parameters to VMs at deploy time?

This can be done via the contextualisation mechanism that is available via the OCCI interface used in BonFIRE. You can add any key-value pairs of configuration parameters you wish, as well as specifying any custom monitoring metrics, elasticity trigger rules, additional SSH keys and post-install scripts that can be run when the VM has been deployed. For more information about how to do this, see the [Contextualisation](#) page.

12.1.5 Can I get my experiment VMs execute scripts automatically when they have deployed?

Yes, we refer to this as post-install, which can be set up via the [contextualisation](#) of the VM when you deploy it.

12.1.6 How can I get my VM to wait until contextualisation is finished before continuing with initialisation?

On ONE sites, check for the absence of /mnt/init.sh. On other sites, test for the presence of /tmp/contextualisation_script_done. In bash, can be tested with

```
[ ! -f /mnt/init.sh ] || [ -f /tmp/contextualisation_script_done ]
```

For more information, see the [contextualisation](#) page.

12.1.7 How can I create/save new VM images?

Firstly, you will have to use one of the base images in BonFIRE - you cannot create your own VM images on your local machine and load into BonFIRE. If you do require a particular image/OS that is not currently available in BonFIRE, please [contact us and make a request](#).

When you deploy one of the [base VMs](#) in BonFIRE and SSH into it, you can install software and configure it as you like. You can save a new VM image based on this, so you don't have to repeat this process. This is done by issuing a SAVE_AS command and then shutting down the VM. For more information about how to do this, please see the [Configuring Software and Saving VMs](#) page.

Please note that VM images you create on one testbed cannot be used on another testbed. Therefore, if you are doing a federated experiment in BonFIRE, you may need to save a VM image for each testbed you will use.

12.1.8 How can I reserve resources for running my experiment on?

Advanced reservation of resources in BonFIRE is not possible for the permanent resources, but can be done for the [on-request resources](#) in BonFIRE. Inria offers an online portal for doing this automatically, whilst the other testbeds need to do this manually. Please see the [On Request Resources](#) page for more details on this process.

12.1.9 Where can I find details of physical hosts and current usage?

General details of the infrastructures can be found [here](#) and more specific testbed details [here](#).

There is some statistics of usage for sites, which you can read more about [here](#).

12.1.10 How can I deploy a VM to a specific physical host?

The BonFIRE OCCI interface allows you to specify the host, but this is only possible on the following Open Nebula testbeds: EPCC, HLRS and Inria. For more information about how to specify the host, see for example the following [OCCI tutorial](#).

An overview of the physical hosts available at the different testbeds is provided on the [Testbed Specificities](#) page.

12.1.11 How do I see which physical host my VM is running on?

If you didn't specify the physical host to deploy your VM on, or you simply want to validate that it deployed where you specified, you can see this information in the OCCI XML you receive when deploying the resource. If you don't use OCCI directly via cURL (for example), you can still see this information by clicking on your experiment VMs on the [BonFIRE Portal](#).

12.1.12 How can I deploy multiple instances of the same VM?

The BonFIRE OCCI interface currently only supports operations for single compute resources, i.e., OCCI does not support calls to deploy X number of a particular VM. However, this functionality is supported in client tools such as the *Experiment Descriptors* or *Restfully scripts*.

12.1.13 Why are some basic Linux tools (e.g., less) missing from my VMs OS?

We have tried to provide as lightweight default operating system images as possible. You can easily install more applications yourself and then save the image for further use. The apt-get tool can be used to install software. For example:

```
apt-get update  
apt-get install less
```

After doing these for the applications or tools you wish to install save your image and use this as your OS image for subsequent work.

12.1.14 How can I get monitoring data for my experiments?

First, you will need to have enabled monitoring for your experiment when you created it, which involves deploying a Zabbix Aggregator and passing on its IP address to the Zabbix Agents in the VMs you deploy in your experiment. For more information about how to set up the monitoring, please see the [monitoring how-to](#) page.

Asssuming you have set up monitoring for your experiments, you can always view the monitoring information via the Zabbix UI - you will find a link to this on the BonFIRE portal in the experiment view. You can download the monitoring information [programmatically](#) or [exported to CSV](#). To be able to get the monitoring data after your experiment has finished (resources, including monitoring resources will have been destroyed), please see [this page](#).

12.1.15 Can I dynamically scale the CPU and memory assigned to my VMs?

No, BonFIRE does currently not support vertical elasticity.

12.1.16 How much space do storage resources occupy?

Currently all storage resources (including OS type images) occupy their full size, irrespective of the amount of data stored in them.

12.1.17 If the storage is created within the experiment: is it created at the target hosting node directly?

OpenNebula creates the storage on the front-end and then transfers to the worker node.

12.1.18 Is qcow2 supported on BonFIRE?

No, the OpenNebula version used in BonFIRE does not support qcow2.

12.1.19 How do I create a network with controlled bandwidth, latency etc?

Currently, you can do this with emulated networking at the [Virtual Wall](#). In the next release of BonFIRE, it will also be possible to use AutoBAHN to configure the bandwidth between your VMs on the EPCC and PSNC testbeds.

INDEX

Symbols

-show <api>, <scheduler>
 ccmsh command line option, 282
-start <api interface port>, <scheduler interface port>
 ccmsh command line option, 282
-stop <api>, <scheduler>
 ccmsh command line option, 282
-a, -enl
 ccmsh command line option, 281
-b, -bfz
 ccmsh command line option, 282
-c, -clear-logs
 ccmsh command line option, 282
-d, -delete <emulation name>
 ccmsh command line option, 282
-e, -emu <emulator name>
 ccmsh command line option, 282
-h, -help
 ccmsh command line option, 281
-i, -dist <distribution name>
 ccmsh command line option, 282
-j, -list-jobs
 ccmsh command line option, 282
-l, -list <emulation name>
 ccmsh command line option, 282
-m, -rmq
 ccmsh command line option, 281
-n, -now (used with -x option only)
 ccmsh command line option, 282
-p, -purge
 ccmsh command line option, 282
-q, -mq
 ccmsh command line option, 281
-r, -results <emulation name>
 ccmsh command line option, 282
-s, -smq
 ccmsh command line option, 281
-v, -version
 ccmsh command line option, 281
-x, -xml <file name>
 ccmsh command line option, 282

C

ccmsh command line option
-show <api>, <scheduler>, 282
-start <api interface port>, <scheduler interface port>, 282
-stop <api>, <scheduler>, 282
-a, -enl, 281
-b, -bfz, 282
-c, -clear-logs, 282
-d, -delete <emulation name>, 282
-e, -emu <emulator name>, 282
-h, -help, 281
-i, -dist <distribution name>, 282
-j, -list-jobs, 282
-l, -list <emulation name>, 282
-m, -rmq, 281
-n, -now (used with -x option only), 282
-p, -purge, 282
-q, -mq, 281
-r, -results <emulation name>, 282
-s, -smq, 281
-v, -version, 281
-x, -xml <file name>, 282