openATTIC Documentation

Release 2.0.6-201512151436

it-novum GmbH

December 15, 2015

The times when storage was considered a server-based resource and every system needed to have its own hard drives are long gone. In modern data centers central storage systems have become ubiquitous for obvious reasons. Centrally managed storage increases flexibility and reduces the cost for unused storage reserves. With the introduction of a cluster or virtualization solution shared storage becomes a necessity.

This mission-critical part of IT used to be dominated by proprietary offerings. Even though mature open source projects may now meet practically every requirement of a modern storage system, managing and using these tools is often quite complex and is mostly done decentrally.

openATTIC is a full-fledged central storage management system. Hardware resources can be managed, logical storage areas can be shared and distributed and data can be stored more efficiently and less expensively than ever before – and you can control everything from a central management interface. It is no longer necessary to be intimately familiar with the inner workings of the individual storage tools. Any task can be carried out by either using openATTIC's intuitive web interface or via the REST API.

CHAPTER

ONE

INSTALLATION AND GETTING STARTED

This section guides you through the necessary system preparation and the installation process of the openATTIC software.

1.1 System requirements

1.1.1 Operating System Requirements

Note: openATTIC has been designed to be installed on a 64-bit Linux operating system. Installation on 32-bit systems is not supported.

Installable packages of openATTIC are currently available for the following Linux distributions:

- Debian Linux 7 (Wheezy)
- Debian Linux 8 (Jessie)
- Ubuntu Linux 12.04 LTS (Precise)
- Ubuntu Linux 14.04 LTS (Trusty)
- Red Hat Enterprise Linux 7 (RHEL) and derivatives (CentOS 7, Oracle Linux 7 or Scientific Linux 7)

In order to use the ZFS file system, you need to install ZFS on Linux separately. Installation packages for various Linux distributions are available from the ZFS on Linux web site.

FibreChannel support requires at least Linux kernel version 3.5.

1.1.2 Hardware Considerations

openATTIC is designed to run on commodity hardware, so you are not in any way bound to a specific vendor or hardware model.

You need to make sure that your Linux distribution of choice supports the hardware you intend to use. Check the respective hardware compatibility lists or consult your hardware vendor for details.

However, there are a couple of things you should be aware of when designing the system.

1. Buy an enclosure with enough room for disks. The absolute minimum recommendation is twelve disks, but if you can, you should add two hot-spares, so make that fourteen. For larger setups, use 24 disks.

   Warning: Any other number of disks will hinder performance.

2. Are you building a storage backend for virtualization? If so, you will require SAS disks, a very clean setup and a good caching mechanism to achieve good performance.

   Note: Using SSDs instead of SAS disks does not necessarily boost performance. A clean setup on SAS disks delivers the same performance as SSDs, and an unclean SSD setup may even be slower.

3. If the enclosure has any room for hot spare disks, you should have some available. This way a disk failure can be dealt with immediately, instead of having to wait until the disk has been replaced.

   Note: A degraded RAID only delivers limited performance. Taking measures to minimize the time until it can resume normal operations is therefore highly advisable.

4. You should have some kind of hardware device for caching. If you're using a RAID controller, make sure it has a BBU installed so you can make use of the integrated cache. For ZFS setups, consider adding two SSDs.

   Note: When using SSDs for caching, the total size of the cache should be one tenth the size of the device being cached, and the cache needs to be ten times faster. So:

   - only add a cache if you have to - no guessing allowed, measure!
   - don't make it too large
   - don't add an SSD cache to a volume that is itself on SSDs

5. Do you plan on using replication in order to provide failure tolerance? If so, ...

   - you will require the same hardware for all of your nodes, because when using synchronous replication, the slowest node limits the performance of the whole system.
   - make sure the network between the nodes has a low latency and enough bandwidth to support not only the bandwidth your application needs, but also has some extra for bursts and recovery traffic.

   Note: When running VMs, a Gigabit link will get you pretty far. Money for a 10GE card would be better spent on faster disks.

6. You should have a dedicated line available for replication and cluster communication. There should be no other active components on that line, so that when the line goes down, the cluster can safely assume its peer to be dead.
7. Up to the supported maximum of 128GB per node, add as much RAM as you can (afford). The operating system will require about 1GB for itself, everything else is then used for things like caching and the ZFS deduplication table. Adding more RAM will generally speed things up and is always a good idea.

1.1.3 Preparing the Installation

1. Always dedicate two disks to a RAID1 for the system. It doesn't matter if you use hardware or software RAID for this volume, just that you split it off from the rest.

   Note: You can also use other devices to boot from if they fit your redundancy needs.

2. When using hardware RAID:

   (a) Group the other disks into RAID5 arrays of exactly 5 disks each with a chunk size (strip size) of 256KiB. Do not create a partition table on these devices. If your RAID controller does not support 256KiB chunks, use the largest supported chunk size.

(b) Using mdadm, create a Software-RAID0 device on exactly two or four of your hardware RAID devices. Again, do not create a partition table on the resulting MD device. Make sure the chunk size of the RAID0 array matches that of the underlying RAID5 arrays. This way, you will not be able to add more than 20 disks to one PV. This is intentional. If you need to add more disks, create multiple PVs in the same manner.

(c) Using pvcreate, create an LVM Physical Volume on the MD device and add it to a VG using vgcreate or vgextend.

(d) Do not mix PVs of different speeds in one single VG.

3. When using ZFS:

   You will need to specify the complete layout in the zpool create command, so before running it, consider all the following points.

   (a) Group exactly six disks in each raidz2. Use multiple raidz2 vdevs in order to add all disks to the zpool.

   (b) When adding SSDs, add them as mirrored log devices.

   (c) Set the mount point to /media/<poolname> instead of just /<poolname>.

   (d) Do not use /dev/sdc etc, but use /dev/disk/by-id/... paths instead.

   So, the command you're going to use will look something like this:

```
# zpool create -m /media/tank tank \
  raidz2 /dev/disk/by-id/scsi-3500000e1{1,2,3,4,5,6} \
  raidz2 /dev/disk/by-id/scsi-350000392{1,2,3,4,5,6} \
  log mirror /dev/disk/by-id/scsi-SATA_INTEL_SSD{1,2}
```

1.1.4 Operating System Configuration Hints

1. Disable swap.
2. Make sure the output of `hostname --fqdn` is something that makes sense, e.g. `srvopenattic01.example.com` instead of `localhost.localdomain`. If this doesn't fit, edit `/etc/hostname` and `/etc/hosts` to contain the correct names.
3. In a two-node cluster, add a variable named `$PEER` to your environment that contains the hostname (not the FQDN) of the cluster peer node. This simplifies every command that has something to do with the peer. Exchange SSH keys.
4. In pacemaker-based clusters, define the following Shell aliases to make your life easier:

```
alias maint="crm configure property maintenance-mode=true"
alias unmaint="crm configure property maintenance-mode=false"
```

5. After setting up MD raids, make sure `mdadm.conf` is up to date. This can be ensured by running these commands:

```
# /usr/share/mdadm/mkconf > /etc/mdadm/mdadm.conf
# update-initramfs -k all -u
```

6. Install and configure an NTP daemon on every host.
7. You may want to install the `ladvd` package, which will ensure that your switches correctly identify your system using LLDP.
8. Make sure `/etc/drbd.d/global_common.conf` contains the following variables:

```
disk {
 no-disk-barrier;
 no-disk-flushes;
 no-md-flushes;
}

net {
 max-buffers 8000;
 max-epoch-size 8000;
}

syncer {
 al-extents 3389;
}
```

1.2 Installation on Debian/Ubuntu Linux

1.2.1 Enabling the openATTIC Apt package repository

In order to use enable the openATTIC Apt repository, create a file named `/etc/apt/sources.list.d/openattic.list`, and put the following lines into it:

For Debian 7 (Wheezy)

```
deb     http://apt.openattic.org/ wheezy   main
deb-src http://apt.openattic.org/ wheezy   main
```

For Debian 8 (Jessie)

Note: Currently, only the nightly packages are supported for Debian Jessie.

```
deb     http://apt.openattic.org/ jessie   main
deb-src http://apt.openattic.org/ jessie   main
deb     http://apt.openattic.org/ nightly  main
deb-src http://apt.openattic.org/ nightly  main
```

For Ubuntu 14.04 LTS (Trusty)

```
deb     http://apt.openattic.org/ trusty   main
deb-src http://apt.openattic.org/ trusty   main
```

Enabling Nightly Builds (for Debian Jessie or Ubuntu Trusty)

In addition to the offical releases, we also provide nightly builds, build off the current development branch.

Add the following to the existing `/etc/apt/sources.list.d/openattic.list` file:

```
deb     http://apt.openattic.org/ nightly  main
deb-src http://apt.openattic.org/ nightly  main
```

Importing the openATTIC Keyfile

The openATTIC packages are signed using a cryptographic key. You can import the key's public key from the download site using the following command:

```
# wget http://apt.openattic.org/A7D3EAFA.txt -q -O - | apt-key add -
```

1.2.2 Installation (Debian Jessie)

```
# apt-key adv --recv --keyserver hkp://keyserver.ubuntu.com A7D3EAFA
# echo deb http://apt.open-attic.org/ jessie main > /etc/apt/sources.list.d/openattic.list
# apt-get update
# apt-get install openattic
# oaconfig install
```

1.2.3 Installation (Ubuntu Trusty 14.04)

```
# apt-key adv --recv --keyserver hkp://keyserver.ubuntu.com A7D3EAFA
# echo deb http://apt.open-attic.org/ trusty main > /etc/apt/sources.list.d/openattic.list
# apt-get update
# apt-get install openattic
# oaconfig install
```

1.3 Installation on Red Hat Enterprise Linux (and Derivatives)

Starting with version 2.0, openATTIC is also available for RPM-based Linux distributions, namely Red Hat Enterprise Linux 7 (RHEL) and derivatives (e.g. CentOS 7, Oracle Linux 7 or Scientific Linux 7). For the sake of simplicy, we refer to these distributions as Enterprise Linux 7 (EL7).

The software is delivered in the form of RPM packages via dedicated yum repositories.

1.3.1 Preliminary Preparations on RHEL 7

To install on RHEL 7, be sure to disable the "EUS" and "RT" yum repos, and enable the "Optional" repo:

```
# subscription-manager repos --disable=rhel-7-server-eus-rpms
# subscription-manager repos --disable=rhel-7-server-rt-rpms
# subscription-manager repos --enable=rhel-7-server-optional-rpms
```

Afterwards, just follow the installation steps as outlined for EL7.

1.3.2 Yum Repository Configuration

Download and install the `openattic-release` RPM package located in the following directory:

```
# yum install http://repo.openattic.org/rpm/openattic-2.x-el7-x86_64/openattic-release.rpm
```

To enable the nightly RPM builds, edit `/etc/yum.repos.d/openattic.repo` and enable the `[openattic-nightly]` yum repository by setting `enabled` to `1`.

1.3.3 openATTIC Installation

To install the packages on CentOS 7, run the following commands:

1. Disable SELinux:

   ```
   # setenforce 0
   ```

   Edit `/etc/sysconfig/selinux` and set `SELINUX` to `disabled`.

2. Install packages:

   ```
   # yum install epel-release
   # yum install openattic
   ```

3. If you have installed your system's root and swap file systems on Logical Volumes, you can tag them to prevent openATTIC from using them:

   ```
   # lvchange --addtag @sys /dev/centos/root
   # lvchange --addtag @sys /dev/centos/swap
   ```

4. Create a Volume Group for openATTIC to use:

   ```
   # pvcreate /dev/sdb
   # vgcreate vgdata /dev/sdb
   ```

5. Install the database:

   ```
   # oaconfig install
   ```

6. Install the GUI

   The GUI is not installed automatically when using `yum install openattic`, as it might not be required on each node of an openATTIC cluster. Instead, it should be installed with the following command:

   ```
   # yum install openattic-gui
   ```

1.4 Getting started

In order to use the openATTIC GUI, you need to run one last command in your shell:

```
# oaconfig add-disk /dev/<sdX> <vgname>
```

After running this command, the whole storage system can be managed by the user interface - have fun!

1.4.1 Accessing the Web UI

Note: HTTP access to the Web UI might be blocked by the default firewall configuration. In order to allow external HTTP requests execute the following command:

```
# firewall-cmd --zone=public --add-port=80/tcp --permanent
```

Open a web browser and navigate to http://openattic.yourdomain.com/openattic

1.4.2 Installing additional openATTIC Modules

After installing openATTIC, you can install additional modules by using `oaconfig install openattic-module-<module-name>`, i.e.:

```
# oaconfig install openattic-module-drbd
# oaconfig install openattic-module-btrfs
# oaconfig install openattic-module-lio
```

Note: `oaconfig install` currently works on Debian/Ubuntu only. On EL7, use `yum install openattic-module-<module-name>` instead.

1.5 Enabling Ceph Support in openATTIC

To set up openATTIC with Ceph you first have to copy the Ceph administrator keyring and configuration from your Ceph admin node to your openATTIC system.

From your Ceph admin node, you can perform this step by using `ceph-deploy` (assuming that you can perform SSH logins from the admin node into the openATTIC host):

```
# ceph-deploy admin openattic.yourdomain.com
```

On the openATTIC node, you should then have the following files:

```
/etc/ceph/ceph.client.admin.keyring
/etc/ceph/ceph.conf
```

Alternatively, you can copy these files manually.

The next step is to install the openATTIC Ceph module on your system:

```
# oaconfig install openattic-module-ceph
```

The last step is to recreate your openATTIC configuration:

```
# oaconfig install
```

CHAPTER

TWO

USER MANUAL

This section covers the openATTIC web user interface (GUI), focusing on storage tasks like adding volumes and shares, system management tasks like the configuration of users and API credentials, and the integrated monitoring system.

2.1 Administration Guide

2.1.1 Introducting the New Graphical User Interface

The new user interface is now based on Bootstrap to make it look more modern, realizing this was a great advantage when we switched from the ExtJS to the AngularJS JavaScript framework.

We restructured the openATTIC user interface in order to make it more intuitive and user-friendly. This included a clean-up of the menu tree as well. Actions like snapshots and shares are now directly available in the volumes panel - by selecting a volume those options get activated and will only display useful actions, depending on the volume type.

Also, we have integrated wizards on the dashboard so that users can be guided through the single steps based on specific use cases like **VM storage** or **Raw Block Storage**.

2.2 How to Perform Common Tasks

- Dashboard
  - overview of the system (disk load, cpu load)
  - cluster/host status (written data, network traffic)
  - wizards
- Disks
  - displays all disks
  - create pool
- Pools
  - all existing pools
  - add pool
- Volumes
  - volumes overview

  - actions
    - add
    - delete
    - set deletion protection for volume
    - clone
    - resize
  - more options (detail-view)
    - click volume and
      - make a snapshot
      - create clone from snapshot
      - create a share
      - automatically only shows available options for volume type
    - without filesystem
      - only iSCSI/FibreChannel
    - with filesystem
      - http
      - NFS
      - CIFS
      - check performance
- Hosts
  - host overview
  - actions
    - add
      - add attribute (peer, initiator for iSCSI share/FibreChannel WWN for FC share)
- System
  - Users
    - add
    - edit
    - delete
    - update: field “is superuser” was changed to “has all privileges” | “is staff” was changed to “is administrator”
  - Command Logs
    - all nagios logs
    - options
      - delete by date
      - delete

  - CRUSH Map

Removed: API-Keys

CHAPTER

THREE

DEVELOPER DOCUMENTATION

openATTIC consists of a set of components built on different frameworks, which work together to provide a comprehensive storage management platform.

When an application (e.g. the openATTIC Web UI, a command line tool or an external application), wants to perform an action, the following happens:

- The REST API receives a request in form of a function call, decides which host is responsible for answering the request, and forwards it to the core on that host.
- The *openATTIC Core* consists of two layers:
  - Django Models, the brains. They keep an eye on the whole system and decide what needs to be done.
  - File system layer: Decides which programs need to be called in order to implement the actions requested by the models, and calls those programs via the openATTIC `systemd` background process (not to be confused with the systemd System and Service Manager).
- The openATTIC *systemd* executes commands on the system and delivers the results.

First of all, start off by *Setting up a Development System*. Then code away, implementing whatever changes you want to make. See *Contributing Code to openATTIC* for details on how to submit your changes to the upstream developers. Follow the *openATTIC Contributing Guidelines* to make sure your patches will be accepted.

3.1 Setting up a Development System

In order to begin coding on openATTIC, you need to set up a development system, by performing the following steps. The instructions below assume a Debian "Jessie" or Ubuntu "Trusty" Linux environment. The package names and path names likely differ on other Linux distributions.

The openATTIC source code is managed using the Mercurial distributed source control management tool. Mercurial offers you a full-fledged version control, where you can commit and manage your source code locally and also exchange your modifications with other developers by pushing and pulling change sets across repositories.

If you're new to Mercurial, take a look at the Learn Mercurial web site. This will teach you the basics of how to get started.

3.1.1 Create Your own openATTIC Fork

The openATTIC source code repository is publicly hosted in a Mercurial Repository on BitBucket.

A "fork" is a remote Mercurial clone of a repository. Every openATTIC developer makes code modifications on a local openATTIC fork before they are merged into the main repository. See *Contributing Code to openATTIC* for instructions on how to get your code contributions included in the openATTIC main repository.

It is possible to create a local clone of the openATTIC repository by simply running `hg clone https://bitbucket.org/openattic/openattic`.

However, if you would like to collaborate with the openATTIC developers, you should consider creating a user account on BitBucket and create a "Fork".

Take a look at the BitBucket Documentation for instructions on how to create a free BitBucket account. We require real user names over pseudonyms when working with contributors.

Once you are logged into BitBucket, go to the openATTIC main repository and click **Fork** on the left side under **ACTIONS**. Now you should have your own openATTIC fork, which will be used to create a local copy (clone). You can find your repository's SSH or HTTPS URL in the top right corner of the repository overview page.

3.1.2 Installing the Development Tools

openATTIC requires a bunch of tools and software to be installed and configured, which is handled automatically by the Debian packages. While you could of course configure these things manually, doing so would involve a lot of manual work which isn't really necessary. Set up the system just as described in *Installation and Getting Started*, but **do not yet execute** `oaconfig install`.

We recommend installing a nightly build for development systems, which is based on the latest commit in the `default` branch.

1. Set the installed packages on `hold` to prevent Apt from updating them:

   ```
   # apt-mark hold 'openattic-.*'
   ```

2. Install Mercurial:

   ```
   # apt-get install mercurial
   ```

3. Install Node.JS and the Node Package Manager `npm`:

   ```
   # apt-get install nodejs npm
   # ln -s /usr/bin/nodejs /usr/bin/node
   ```

4. Install Bower and Grunt (to build the Web UI):

   ```
   # npm install -g bower
   # npm install grunt
   # npm install -g grunt-cli
   ```

5. Go to the `/srv` directory, and create a local clone of your openATTIC fork there, using the current `development` branch as the basis:

   ```
   # cd /srv
   # hg clone -u development https://hg@bitbucket.org/<Your user name>/openattic
   ```

6. Customize the Apache configuration by editing `/etc/apache2/conf-available/openattic.conf` and replace `/usr/share/openattic` with `/srv/openattic/backend`. Also add the following directive:

   ```
   <Directory /srv/openattic>
     Require all granted
   </Directory>
   ```

7. In file `/etc/default/openattic`, change the `OADIR` variable to point to the local Mercurial clone:

   ```
   OADIR="/srv/openattic/backend"
   ```

8. In file `/etc/apache2/conf-available/openattic`, change the `WSGIScriptAlias` line to point to the local clone:

   ```
   WSGIScriptAlias  /openattic/serverstats  /srv/openattic/backend/serverstats.wsgi
   WSGIScriptAlias  /openattic              /srv/openattic/backend/openattic.wsgi
   ```

9. Now build the Web UI:

   ```
   # cd /srv/openattic/webui
   # npm install
   # bower install --allow-root
   # grunt build
   ```

   If you intend to make changes to the web interface, it may be useful to run `grunt dev` as a background task, which watches the project directory for any changed files and triggers an automatic rebuild of the web interface code (including the jshint output), if required.

10. Run `oaconfig install` and start openATTIC by running `oaconfig start`.

The openATTIC web interface should now be accessible from a local web browser via <http://localhost/openattic/>_ . The default username and password is "openattic".

You can now start coding by making modifications to the files in `/srv/openattic`. The openATTIC daemons, GUI and the `oaconfig` tool will automatically adapt to the new directory and use the code located therein.

See chapters *Contributing Code to openATTIC* and *openATTIC Contributing Guidelines* for further details on how to prepare your code contributions for upstream inclusion.

3.2 Contributing Code to openATTIC

This is an introduction on how to contribute code or patches to the openATTIC upstream project. If you intend to submit your code upstream, please also review and consider the guidelines outlined in chapter *openATTIC Contributing Guidelines*.

3.2.1 Keeping Your Local Repository in Sync

If you have followed the instructions in *Setting up a Development System*, you should already have a local openATTIC instance that is based on the current development branch.

You should update your repository configuration so that you will always pull from the main openATTIC repository and push to your openATTIC fork by default. This ensures that your fork is always up to date, by tracking the upstream development.

In your local clone, edit the Mercurial configuration file `.hg/hgrc`. It should contain the following three lines:

```
[paths]
default = https://hg@bitbucket.org/openattic/openattic
default-push = https://hg@bitbucket.org/<Your user name>/openattic
```

The `default-push` location is the URL that you can obtain from your fork's repository overview page on BitBucket.

If you want to push via SSH, you just have to modify your `default-push` URL:

```
default-push = ssh://hg@bitbucket.org/<Your user name>/openattic
```

This requires uploading your public SSH key to BitBucket first. Check the BitBucket documentation for details on how to accomplish this.

If you want to use SSH behind a proxy you may use corkscrew. After the installation, append the following two lines to your `$HOME/.ssh/config` file:

```
Host bitbucket.org
    ProxyCommand corkscrew <proxy name or ip> <port number> %h %p
```

Now you can use SSH behind the proxy, because corkscrew now tunnels your SSH connections through the proxy to bitbucket.org.

3.2.2 Working With Branches

It is strongly recommended to separate changes required for a new feature or for fixing a bug in a separate Mercurial branch. Please refer to the Mercurial documentation for a detailed introduction into working with branches.

If you intend to submit a patch to the upstream openATTIC repository via a pull request, please make sure to follow the *openATTIC Contributing Guidelines*.

To create a new feature branch update your repository, change to the development branch and create your new branch on top of it, in which you commit your feature changes:

```
# hg pull
# hg update development
# hg branch <branchname>
< Your code changes >
# hg commit
```

To list your branches type:

```
# hg branches
```

To see the current branch you are working with type:

```
# hg branch
```

After you are done with your changes, you want to push them to your fork:

```
# hg push
```

If you can't push them because a new remote branch would be created use:

```
# hg push --new-branch
```

3.2.3 Submitting Pull Requests

Now that your fork contains your local changes in a separate branch, you can create a pull-request on Bitbucket to request an inclusion of the changes you have made into the `development` branch of the main openATTIC repository.

To do this, go to your fork on Bitbucket and click `Create pull request` in the left panel. On the next page, choose the branch with your changes as source and the main openATTIC development branch as target.

Below the **Create pull request** button, first check out the **Diff** part if there are any merge conflicts. If you have some, you have go back into your branch and update it:

```
# hg pull
# hg merge development
<test and review changes>
# hg commit -m "Merged development"
# hg push
```

After you have resolved the merge conflicts and pushed them into your fork, retry submitting the pull-request. If you already created a pull request, BitBucket will update it automatically.

After the pull-request was reviewed and accepted, your feature branch will be merged into the main repository. The merged feature branch will then be closed in the main openATTIC repository by the maintainer.

Please do not close the branch yourself, because after pulling from the main repository, you'll also receive a changeset which closes your local branch.

To push the merge and closing of your branch into your fork again you have to run the following command:

```
# hg pull -u
# hg push
```

3.2.4 Collaborating With Other Developers

The distributed nature of Mercurial makes it possible to collaborate with other developers on the same set of changes, by pulling and pushing change sets between the personal forks of the openATTIC repository.

To pull changes from another developer's branch, type the following:

```
# hg pull <alias or fork URL> <branch name>
```

If you plan to contribute something to the branch you have to push your changes to your fork. The other developer can pull the changes the other way round, see hg command above.

To create and use an alias you have to edit your `.hg/hgrc` and add a new alias beneath `[paths]`:

```
<alias name> = <fork clone URL>
```

The following images illustrate this concept:

To sum it up

Work on a specific branch:

```
# hg update <branch name>
```

Fetch new revisions from openATTIC:

```
# hg pull -u
```

Merge your branch to the latest revision:

```
# hg pull -u
# hg merge development
```

Create a new branch on top of the current working branch:

```
# hg branch <branch name>
```

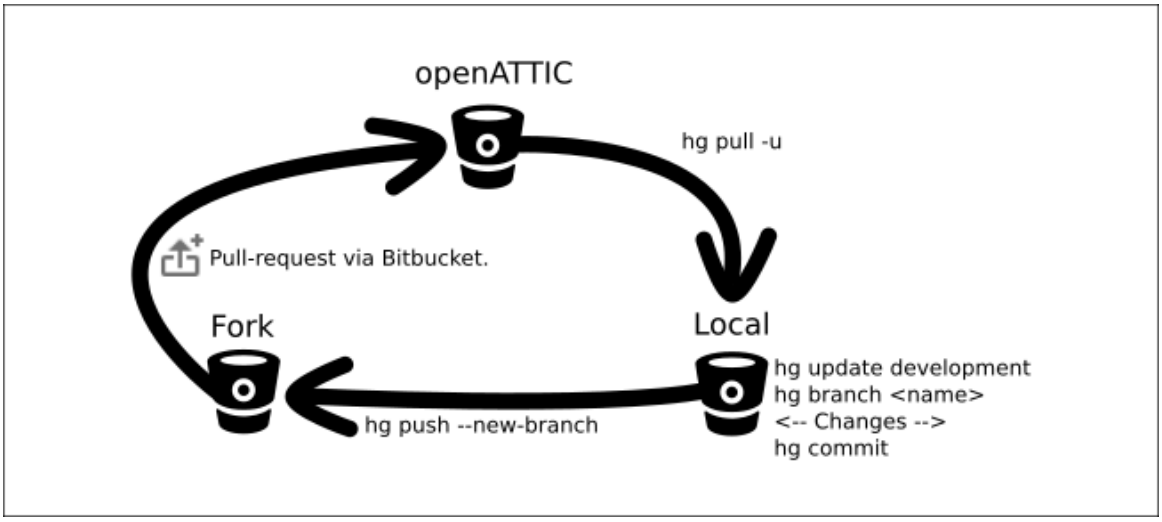

Figure 3.1: Workflow between the main openATTIC repository and your fork.

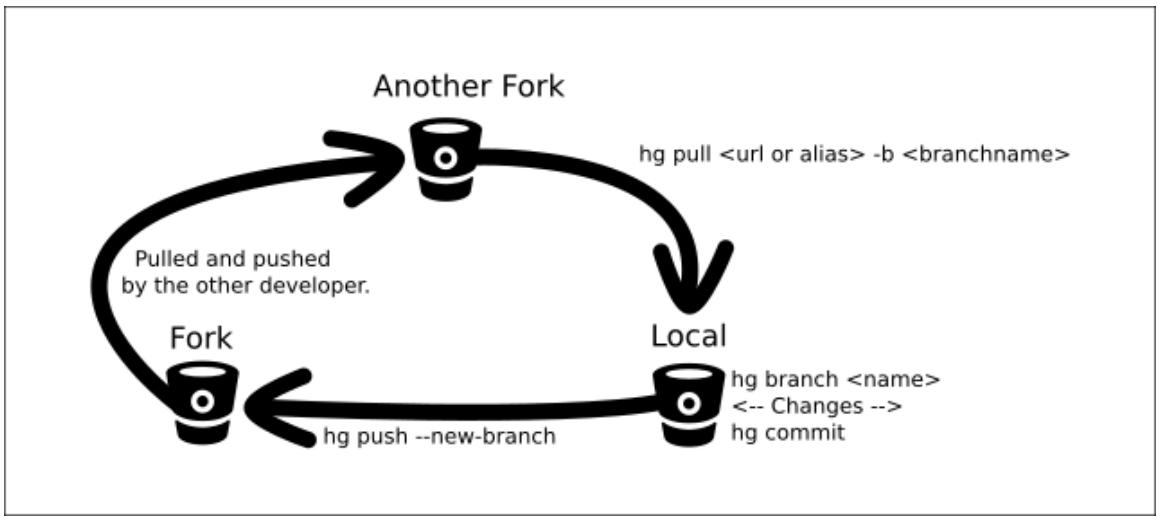

Figure 3.2: A collaborative workflow between two forks.

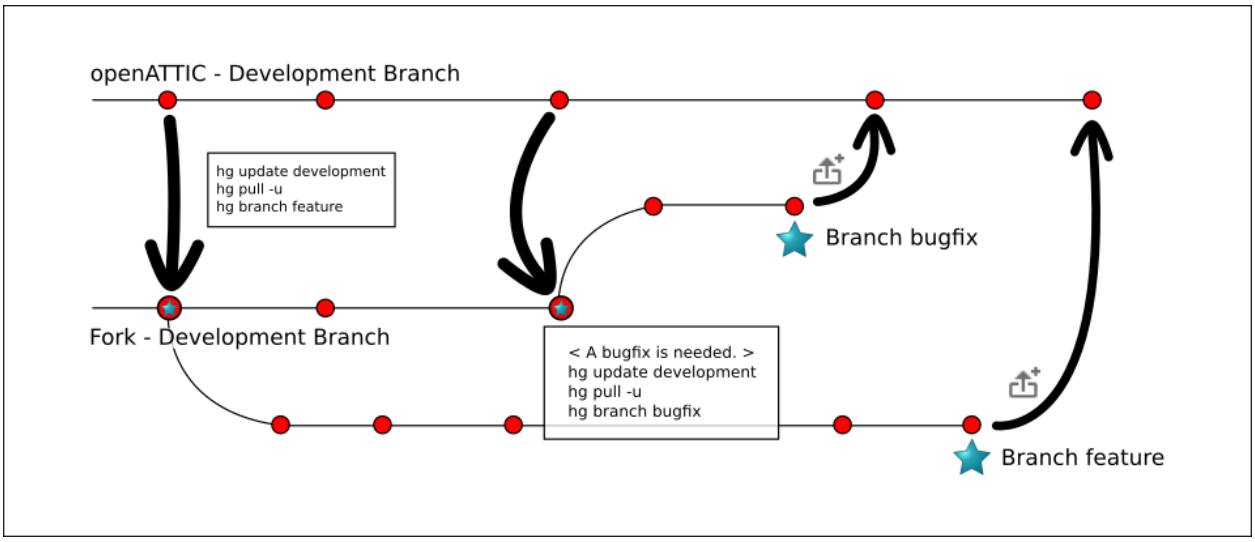

Figure 3.3: The workflow with branches.

Lists all open branches:

```
# hg branches
```

Show current working branch:

```
# hg branch
```

Merges a branch into the current working branch:

```
# hg merge <branch name>
```

Push your changes on your fork:

```
# hg push
```

Does the above, but creates a new branch or deletes an old one:

```
# hg push --new-branch
```

3.3 openATTIC Contributing Guidelines

The following recommendations should be considered when working on the openATTIC Code Base.

While adhering to these guidelines may sound more work in the first place, following them has multiple benefits:

- It supports the collaboration with other developers and others involved in the product development life cycle (e.g. documentation, QA, release engineering).
- It makes the product development life cycle more reliable and reproducible.
- It makes it more transparent to the user what changes went into a build or release.

Some general recommendations for documenting/tracking your changes:

- Every bug fix or notable change made to a release branch must be accompanied by a bug report (JIRA issue).

- New features and other larger changes also require a related JIRA issue that provides detailed background information about the change.
- Code and the related changes to the documentation should be committed in the same change set, if possible. This way, both the code and documentation are changed at the same time.
- Write meaningful commit messages. Commit messages should include a detailed description of the change, including a reference to the related JIRA issue, if appropriate. "Fixed OP-xxx" is not a valid or useful commit message! For details on why this matters, see The Science (or Art?) of Commit Messages and How to Write a Git Commit Message (this applies to Mercurial as well).
- When resolving a JIRA issue as fixed, include the resulting Mercurial Change Set Revision ID or add a link to the ChangeSet or related pull request on BitBucket for reference. This makes it easier to review the code changes that resulted from a bug report or feature request.

3.3.1 Signing Your Patch Contribution

To improve tracking of who did what, we use the "sign-off" procedure introduced by the Linux kernel. The sign-off is a simple line at the end of the explanation for the patch, which certifies that you wrote it or otherwise have the right to pass it on as an open-source patch.

The rules are pretty simple: if you can certify the following:

```
Developer Certificate of Origin
Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.
660 York Street, Suite 102,
San Francisco, CA 94110 USA

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
    have the right to submit it under the open source license
    indicated in the file; or

(b) The contribution is based upon previous work that, to the best
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.

(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

then you just add the following line below your commit message and pull request saying:

```
Signed-off-by: Random J Developer <random@developer.example.org>
```

using your **real name and email address** (sorry, no pseudonyms or anonymous contributions).

If you want to automate the task of adding that tag line, consider installing the signoff.py Mercurial hook.

If you like, you can put extra tags at the end:

1. `Reported-by:` is used to credit someone who found the bug that the patch attempts to fix.
2. `Acked-by:` says that the person who is more familiar with the area the patch attempts to modify liked the patch.
3. `Reviewed-by:`, unlike the other tags, can only be offered by the reviewer and means that she is completely satisfied that the patch is ready for application. It is usually offered only after a detailed review.
4. `Tested-by:` is used to indicate that the person applied the patch and found it to have the desired effect.

You can also create your own tag or use one that's in common usage such as `Thanks-to:`, `Based-on-patch-by:`, or `Mentored-by:`.

3.3.2 Documenting Your Changes

Depending on what you have changed, your modifications should be clearly described and documented. Basically, you have two different audiences that have different expectations on how and where you document your changes:

- **Developers** that need to review and comment on your changes from an architectural and code quality point of view. They are primarily interested in the descriptions you put into the Mercurial commit messages and the description of your pull request, but will also review and comment on any other documentation you provide.
- **End users or administrators** that use openATTIC and need to be aware of potential changes in behaviour, new features or important bug and security fixes. They primarily consult the official documentation, release notes and the `CHANGELOG`.

Note: Note that you should not update the `CHANGELOG` directly. Instead, add a note to your pull request that includes the text that should be added to the `CHANGELOG` by the developer that merges your pull request. See chapter *Merging Pull Requests* for details.

Changes that should be user-visibly documented in the `CHANGELOG`, release notes or documentation include:

- Bug/security fixes on a release branch.
- User-visible changes or changes in behavior on a release branch. Make sure to review and update the documentation, if required.
- Major changes / new features. In addition to the `CHANGELOG`, these must be described in the documentation as well.

Minor or "behind the scene" changes that have no user-visible impact or do not cause changes in behavior/functionality (e.g. improvements to build scripts, typo fixes, internal code refactoring) usually don't have to be documented in the `CHANGELOG` or the release notes.

Trust your judgment or ask other developers if you're unsure if something should be user-visibly documented or not.

Don't worry too much about the wording or formatting, the `CHANGELOG` and Release Notes will be reviewed and improved before a final release build anyway. It's much more important that we keep track of all notable changes without someone having to trawl JIRA or the commit messages prior to a release.

3.3.3 Merging Pull Requests

The following steps should be performed when you're reviewing and processing a pull request on BitBucket:

1. A developer fixes a bug or implements a new feature in a dedicated feature branch. If required, he documents the changes in the documentation (for end-users) and the Mercurial commit messages (including the related Jira issue ID and a `Signed-off by:` line as outlined in chapter *Signing Your Patch Contribution*)
2. The developer creates a new Pull Request on BitBucket as described in chapter *Submitting Pull Requests*. The Pull Request description should include a detailed description of the change in a form suitable for performing a code review, summarizing the necessary changes. The description should also include a text suitable for inclusion into the `CHANGELOG`, describing the change from an end-user perspective.
3. After the pull request has been reviewed and approved, you perform the merge into the main development branch using the BitBucket Merge functionality.
4. If the merge was successful, update the `CHANGELOG` in the `development` branch based on the description provided by the developer that submitted the pull request. This can be performed by using the built-in editor on BitBucket.
5. Close the feature branch that this pull request has been merged from, by using the BitBucket web frontend.

3.4 openATTIC Core

The openATTIC core makes heavy use of the Django framework and is implemented as a Django project, consisting of several apps, one for each supported functionality or backend system.

Each app bundles a set of submodules. Models are used to represent the structure of the objects an app is supposed to be able to manage. The REST API (based on the Django REST Framwork is used for interaction with the models. And lastly, the System API can be used in order to run other programs on the system in a controlled way.

3.4.1 Models

Models are used to provide an abstraction for the real-world objects that your app has to cope with. They are responsible for database communication and for keeping an eye on the state of the whole system, being able to access any other piece of information necessary.

Please check out Django at a glance for more information.

3.4.2 Filesystem API

The filesystem API abstracts handling different file systems, translates actions initiated by the model into commands to be executed and calls Systemd accordingly.

3.5 openATTIC E2E Tests

This section describes how our test environment is set up, as well as how you can run our existing tests on your openATTIC system and how to write your own tests.

By continuously writing E2E-tests, we want to make sure that our graphical user interface is stable and acts the way it is supposed to be - that offered functionalities really do what we expect them to do. We want to deliver a well-tested application, so that you - as users and community members - do not get bothered with a buggy user interface. Instead, you should be able to get started with the real deal - MANAGING storage with openATTIC.

3.5.1 About Protractor

Protractor is a end-to-end test framework, which is especially made for AngularJS applications and is based on WebDriverJS. Protractor will run tests against the application in a real browser and interacts with it in the same way a user would.

For more information, please refer to the protractor documentation.

3.5.2 System Requirements

Testing VM:

- Based on our experience,the system on which you want to run the tests needs at least 4GB RAM to prevent it from being laggy or very slow!

3.5.3 Install Protractor

- `npm install -g protractor`
- `apt-get install open-jdk-7-headless`
- `npm install -g jasmine-beforeAll` (in case this package is not available, try `npm install -g jasmine-before-all`)
- Choose/Install your preferred browser (Protractor supports the two latest major versions of Chrome, Firefox, Safari, and IE)
- Please adapt the `protractor.conf.js` file which can be found in `/openattic/webui/` to your system setup - see instructions below

3.5.4 Protractor Configuration

Before starting the tests, you need to configure and adapt some files Here's what you have to do in `protractor.conf.js`:

3.5.5 Enable BeforeAll / AfterAll

In order to use `beforeAll` and `afterAll` you need to tell protractor to use `jasmine2` as framework (protractor uses an older version by default, which does not support beforeAll/afterAll).

Add the following line to your protractor.conf:

```
exports.config = {

  seleniumAddress: ...

  jasmineNodeOpts: {
   ....
  },

``framework: 'jasmine2',``

  suites: {
    ...
    ...
```

```
  },
  ....
}
```

3.5.6 Maximize Browser Window

If the browser windows in which the tests will be executed is too small, it occurs that protractor can't click an element and tests will fail. To prevent this, you can maximize your browser window by default by adding the following line to `webui/protractor.conf.js`:

```
exports.config = {

  seleniumAddress: ...
  jasmineNodeOpts: {
    ....
  },

  framework: 'jasmine2',

  suites: {
    ...
    ...
    ..

  },

``onPrepare: function(){``
  ``browser.driver.manage().window().maximize();``
``},``
}
```

3.5.7 Use multiple browsers

To use Chromium and Firefox to run each test, you have to append the following to your `protractor.conf.js`:

```
exports.config.multiCapabilities = [
    {'browserName': 'chrome'},
    {'browserName': 'firefox'}
];
exports.config.maxSessions = 1; // To prevent running both browsers on the same time.
```

3.5.8 Set up configs.js

Create a `configs.js` file in folder `e2e` and add the URL to you openATTIC system as well as login data - see below:

```
(function() {
  module.exports = {
    url      : 'http://IP-to-your-oA-test-sys/openattic/#/login',
    //leave this if you want to use openATTIC's default user for login
    username: 'openattic',
    password: 'openattic',
  };
}());
```

In order to run our graphical user interface tests, please make sure that your openATTIC system at least has:

- one volume group
- one zpool

and add them to `e2e/configs.js`. .. note:: For more information have a look at `e2e/example_config.js`.

It is important that the first element in this config file is your volume group.

If you do not have a zpool configured and you do not want to create one, you can of course skip those tests by removing the suite from `protractor.conf.js` or putting them in to the comment section.

3.5.9 Start webdriver manager environment

use a separate tab/window to run the following command:

```
webdriver-manager start
```

3.5.10 Make Protractor Execute the Tests

Go to `/srv/openattic/webui/` and type `protractor protractor.conf.js` in order to run the tests:

```
$ protractor protractor.conf.js (--suite <suiteName>)
```

Important: Without a given suite protractor will execute all tests (and this will probably take a while!)

3.5.11 Start Only a Specific Test Suite

If you only want to test a specific action, you can run i.e. `protractor protractor.conf.js --suite snapshot_add` Available test cases can be looked up in protractor.conf.js, i.e.:

```
suites: {
  //suite name       : '/path/to/e2e-test/file.e2e.js'
  snapshot_add       : '../e2e/snapshots/add/**/*.e2e.js',
}
```

Note: When running protractor.conf and the browser window directly closes and you can see something like "user-data error" (i.e. when using chrome) in your console just create a dir (i.e. in /home/) and do `chromium --user-data-dir=/path/to/created/dir`

3.5.12 How to Cancel the Tests

When running the tests and you want to cancel them, rather press `CTRL+C` on the commandline (in same window in which you've started `protractor.conf.js`) than closing the browser. Just closing the browser window causes every single test to fail because protractor now tries to execute the tests and can not find the browser window anymore.

3.5.13 E2E-Test Directory and File Structure

In directory `/srv/openattic/e2e/` the following directories can be found:

```
+-- auth
+-- commandLogs
+-- dashboard
|   `-- todoWidget
+-- disks
+-- general
+-- hosts
+-- pools
+-- shares
|   +-- cifs
|   +-- http
|   +-- lun
|   `-- nfs
+-- snapshots
|   +-- add
|   `-- clone
+-- users
+-- volumes
|   +-- add
|   +-- protection
|   +-- resize
|   `-- zvol
`-- wizards
    +-- block
    +-- file
    `-- vm
```

Most of the directories contain a `..._workflow.e2e.js` in which we only test things like validation, the number of input fields, the title of the form etc. Actions like `add`, `clone` etc. are always in a separate file. This makes it better to get an overview and prevents the files from getting very huge and confusing.

3.5.14 Writing Your Own Tests

Please include `common.js` in every `.e2e.js` file by adding `var helpers = require('../common.js');`. In some cases (depending on how you've structured your tests) you may need to adapt the path.

By including it as `var helpers` you can now make use of helper functions from `common.js`, i.e. the `create_volume` function, you just have to add `helpers.` to the function: `helpers.create_volume( name , type [, size ] )`.

The following helper functions are implemented:

- `create_volume`
- `delete_volume`
- `create_snapshot`
- `delete_snapshot`
- `create_snap_clone`
- `delete_snap_clone`
- `create_host`
- `delete_host`

So if you want to write a test and you need a volume to test an action which is based on a volume (i.e. creating a share), you can use the following lines to create a new volume:

```
beforeAll(function(){
  helpers.login();

  //create an xfs volume before executing any test
  helpers.create_volume("volumename_here","xfs");

});
```

You can also specify the size as a string as third argument, otherwise the volume will always be initiated with 100MB by default.

Depending on which volume type you need, you can set the parameter to:

- `xfs`
- `btrfs`
- `zfs` (if `openattic-module-zfs` is installed)
- `lun`

Every helper function which is based on a volume needs to get the volume object passed.:

```
//var volumename = 'demo_volume';
//volume: var volume = element(by.cssContainingText('tr', volumename));

* ``create_snap_clone(volume)``
* ``helpers.delete_volume(volume, volumename);``
* ``helpers.create_snapshot(volume);``
* ``helpers.delete_snapshot(volume);``
```

When using more than one helper function in one file, please make sure that you use the right order of creating and deleting functions in `beforeAll` and `afterAll`.

Example:

If you put `helpers.delete_volume();` before `helpers.delete_snapshot();` the snapshot will be deleted with the volume and the second one (`delete_snapshot();`) will search for an element which does not longer exist. A second option is to only use `helpes.delete_volume();` so everything which relates to this volumes (like snapshots, shares) will be deleted with the deletion of the volume automatically.

If you need to navigate to a specific menu entry (every time!) where your tests should take place, you can make use of:

```
beforeEach(function(){

  //always navigates to menu entry "Volumes" before executing the actions defined in 'it('', functio
  element.all(by.css('ul .tc_menuitem')).get(3);

});
```

3.5.15 Tips to write tests that also support Firefox

Let protractor only click on clickable elements, like `a`, `button` or `input`.

If you want to select an option element use the following command to make sure that the item is selected (issue #480):

```
browser.actions().sendKeys( protractor.Key.ENTER ).perform();
```

3.5.16 Debugging your tests

To set a breakpoint use `browser.pause()` in your code.

After your test pauses, go to the terminal window where you started the test.

You can type `c` and hit enter to continue to the next command or you can type `rep` to enter the interactive mode, here you can type commands that will be executed in the test browser.

To continue the test execution press `ctrl + c`.

See also:

This documentation is also available as a PDF file.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*