Enhancing Grid Infrastructures with
Virtualization and Cloud Technologies

StratusLab Storage

Technical Note TN-Storage (V0.3)
18 September 2011

Abstract

StratusLab provides a complete, open-source solution for deploying an "Infrastructure as a Service" cloud infrastructure. Storage services are a critical part of the distribution's functionality. StratusLab provides a persistent "disk" service that allows users to create disks and to attach/detach them from virtual machines (at launch or while running). The disk's lifecycle is independent of the virtual machine machine's lifecycle allowing persistent storage of data for later reuse by other virtual machines.

StratusLab is co-funded by the
European Community's Seventh
Framework Programme (Capacities)
Grant Agreement INFSO-RI-261552.

The information contained in this document represents the views of the copyright holders as of the date such views are published.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE MEMBERS OF THE STRATUSLAB COLLABORATION, INCLUDING THE COPYRIGHT HOLDERS, OR THE EUROPEAN COMMISSION BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contributors

Name	Partner	Sections
Alexandre JOSEPH	CNRS-LAL	All
Charles LOOMIS	CNRS-LAL	All

Document History

Version	Date	Comment
0.1	10 August 2011	Initial version for comment.
0.2	26 August 2011	English corrections; added future plans.
0.3	18 Sept. 2011	Update REST API; clean up text.

Contents

List of Tables

1 Introduction

StratusLab provides a complete, open-source solution for deploying an "Infrastructure as a Service" cloud infrastructure. The Persistent Disk Service provides on-site management of persistent storage areas (à la Amazon's Elastic Block Store) allowing users to create disks of a given size, to attach/detach them from virtual machines, and to delete them. The lifecycle of the disk is independent of any particular virtual machine instance, allowing data to be persistent.

The storage component is the key feature for user data management within the StratusLab distribution. It provides block-level storage and associates properties (metadata) with the disks.

1.1 Managing Persistent Disks

Persistent disks are created from scratch, then behave like raw unformatted block devices. Users can create a file system on top of them or use them in any other way they would use a block device (like a hard drive).

The aim of the storage service is to provide a mechanism to users to store their data persistently. They can at any time create a new disk or remove permanently. Creating a disk, offers the following option to the user:

Volume size A disk can hold between 1GB to 1TB of data.

Disk tagging A user can have numerous disks; tags allow users to easily identify a particular disk. The tag is an opaque string which can be used to filter disk listings.

Data visibility The standard policy is that the data are private, meaning that only the user can see the disk, use it, and delete it. The disk owner can choose to make the disk public and share it with others. In this case, other users can see the disk and use it (read and write); the owner is the only one that can delete it.

Disks can be managed either via a web browser or with the user command line interface (CLI). Both interfaces provide the user with the same functionality; both require that the user be authenticated.

1.1.1 Web Interface

The web interface mirrors the REST interface of the service. The main page provides some general information about the service and a link to the disk listing page. All of the features for managing disks can be used through the disk listing and other linked pages. The interface will only show disks owned by the authenticated user.

1.1.2 Command Line Interface

The Command Line Interface (CLI) provides five commands:

`stratus-create-volume` Create a new disk within the persistent disk service,

`stratus-delete-volume` Delete the specified disk,

`stratus-attach-volume` Attach a persistent disk to a running virtual machine,

`stratus-detach-volume` Detach a disk from a running virtual machine, and

`stratus-describe-volumes` List (and optionally filter) the available disks for a user.

Allowing full control of the lifecycle and use of persistent disks.

For the last command, multiple filters can be specified (e.g., the persistent disks is in use, and it is tagged with a particular value). The result includes information for a particular disk only if it matches all the filters. It is possible to use standard regular expression to match the desired persistent disks. Filters can be built with all the properties of the disk.

1.2 Using Persistent Disks

Persistent disks can be attached to a virtual machine in two different ways.

Attach disk at start-up A volume can be linked to an instance when it is created with the `--persistent-disk` option of the `stratus-run-instance` command. The disk will then behave like a standard IDE disk like the root disk of the system. Virtual machines do not any need specific configuration to handle it. The disk cannot be released while the VM is running.

Hot-plug a disk The `stratus-attach-volume` command allows volumes to be attached to a running instance. This means that a disk can be added to or removed from a machine while it is running. There is no need to restart the machine. This feature requires *the operating system to be capable of handling dynamic changes in machine disk configurations*. On Linux systems, the kernel module `acpiphp` needs to be available and loaded. This

is the case for most of the modern distributions such as Ubuntu or CentOS. The name of the device where the persistent disk is attached is returned by the command. At any time a persistent disk can be released with `stratus-detach-volume` command.

Note that disks cannot be deleted while they are in use. They must be first detached from running machine instances if the disk was attached dynamically or by shutting down the machine if it was attached at launch.

2 Storage implementation

The primary requirement of the storage service is that it permits all uses to keep data independently from a Virtual Machine and its life cycle. It means that a persistent disk is not linked to a specific instance, instance type, or appliance. It is totally independent and can be attached at any time to a machine. This requirement also allows users to share data with others.

2.1 Requirements

1. Cloud users can create persistent disks from 1GB to 1TB that can be mounted on a StratusLab machine instance.
2. Allow cloud users to manage disks on the site via a browser and with a command line interface.
3. User data can be shared with others but only on user request.
4. A persistent disk can only be accessed via an instance. This hides from users the underlying technologies and data location. They cannot access it directly, they need to request it through a hypervisor.
5. Persistent disks behave like raw, unformatted block devices. Users can create a file system on top of this disk or use them in any other way a block device could be used (like a hard drive).
6. Disks can be attached to an instance at any time if the operating system supports it. No need to restart the machine (hot-plug feature).
7. Persistent disks are created on a specific storage area but can be used in every StratusLab cloud if the site policy allows it.
8. The storage service is independent from other cloud services. Having a storage service does not require having a other cloud services installed on the infrastructure.
9. Cloud users can attach as many disk as they want on an instance. This means that it can stripe data across them for increased I/O and throughput performance. This can be is particularly helpful for database style applications that frequently encounter many random reads and writes across the dataset.

10. Cloud system-administrators can chose how data are shared between the storage service and the nodes according to the cloud site requirements.

11. Cloud administrators can configure the persistent disk storage to use device partitions (e.g. LVM) or files on a shared file system.

12. Persistent disk metadata storage has to be fault tolerant to avoid losing information about existing disks, which would result in lost data. Redundancy and other fault tolerant mechanism should be integrated by default in the service, allowing system administrator to setup it automatically.

2.2 Technology choices

To ensure that the service can be used easily with all programming languages, the storage will expose a RESTful [5] interface over standard HTTP(S). The implementation will use RESTlet [1], a Java framework for implementing RESTful services.

Disk sharing between service and nodes must use standard disk sharing mechanisms to ensure interoperability with all the existing systems. Implementing a new sharing protocol as S3 is not required and will add an unnecessary complexity. Not having an abstraction layer like S3 will expose directly service sharing technology. That is why users should only be authorized to access data through a running instance, to hide technical details of the service. Data security has to be an important concern in service implementation as it can be very sensitive. New disk sharing methods should be easy to add to existing sharing protocols. The first release of the service will implement iSCSI [3] and NFS [2] (shared file system) share types which should fit to a majority of cloud site requirements.

Sharing protocols like iSCSI support many types of devices such as hard drives, block files, tapes, CDROMs or LVM partitions. Block files are the simplest way to create and share disks without special configuration. It will allow most cloud administrators to setup the service even in a very restricted site. This disk type allows iSCSI and NFS sharing, although NFS not the most efficient way to manage disks. First, file creation can be long depending of the disk size; this is not ideal for RESTful client because of connection timeout. Second, it may result in the exposure of user data if block files are not correctly and completely zeroed. Third, block files do not permit manipulation after creation: they cannot be reduced or increased in size. In addition to block files, LVM partitions are available for creating persistent disks. LVM is a now standard feature in most distributions.

Disk properties are an important part of the storage service. It is the heart of the component, allowing disks to be discovered and used. A storage service should not let a user use a disk just because he knows the disk identifier. Accessibility must be governed by the disk's owner and the defined visibility of the disk. Without disk properties, user data are lost. That is why this information should be stored in a fault tolerant system to increase service availability. As properties are simple

key/value pairs, they do not require a complex database. The most important feature is scalability. The storage service uses Zookeeper which is a highly reliable distributed system primarily designed for maintaining configuration information, naming, synchronizing distributed processes, and providing group services.

2.3 Disk identifiers

To ensure an unambiguous connection between the disk contained in the storage and its properties, a unique disk identifier must be used. This identifier is generated at disk creation and never changes throughout the disk's lifecycle.

For the purposes of the persistent disk service, an UUID [4] identifier is used. The identifier is pseudo-random 128-bit hash. In its canonical form, a UUID is represented by 32 hexadecimal digits, displayed in 5 groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 digits and 4 hyphens). For example, a disk UUID can be:

```
b3dad761-a10e-44c4-b221-1644a0a32809
```

The total number of UUIDs is about 3×10^{38} which should be enough for any storage system with neglible chance of identifier collisions.

2.4 REST Resource URLs

The mapping between URLs and service resources is the central part of any RESTful service. The resource mapping must provide convenient access via a browser but also facilitate automated interactions with tools. Table 2.1 provides the URL mapping for the persistent storage service. Within the table, "uuid" refers to the 36 character disk identifier. Most of the URLs support both HTML and JSON representations. Parameters to POST requests must always be provided via a standard web-encoded form.

2.5 Disk user registration

The storage service is not directly aware that a user is using a disk; it must be informed by an outside agent that a disk is being attached to or detached from a VM. There is two distinct ways to do it 1) by daemon attaching the disk and 2) by the client attaching the disk. Sending attachment or detachment notifications by the daemon which is mounting the disk has the advantage that if the instance fails before attaching the disk, it will remain free. It is also simpler to implement this behavior with existing clients as they do not need changes; everything isis done in the attach/detach daemon. The negative point is that the daemon needs to access to the storage service in order to register new disk. Consequently it need a account with some particular rights to register disks.

For start-up disk attachment, the storage service relies on the first method, this avoids making changes in the `stratus-run-instance` command. It also simplifies extension of storage service. It means that in the future it will be possible to

Table 2.1: *REST URL Mapping for the storage service*

URL	GET	POST	DELETE
/	Landing page with service information and links (HTML)	X	X
/disks/	List available disks and properties (HTML, JSON); HTML contains form to create disk	Create new disk (HTML, JSON); parameters: size, tag, visibility	
/disks/⟨uuid⟩/	Display properties for a disk (HTML, JSON)	X	Remove disk with the given UUID (HTML, JSON)
/disks/⟨uuid⟩/mounts	List mount locations for a disk (HTML, JSON)	Attach a disk to a VM (HTML, JSON); parameters: node, vm_id	X
/disks/⟨uuid⟩/mounts/⟨mountid⟩	Information for a particular mount (HTML, JSON)	X	Detach disk from machine (HTML, JSON)

store appliances in the storage service, register it into a Marketplace and it will be possible to start an instance directly from the storage service. This will be significantly faster as it does not require file copies during the initialization.

When stopping a virtual machine, the storage daemon on the node also informs the service that the disk is no longer in use. This avoids not releasing disks if the storage configuration is incorrect when killing an instance. In this case there is no difference between disks (hot plugged or not).

When hot-plugging a disk, a query is directly made by the client to the storage service. It knows whether a disk is in use or not; there is no need to query another service.

2.6 Disk user quota

Limiting use of disks to a defined number of users is necessary to avoid I/O errors. Cloud system administrators can set a disk user quota on disk. By default this is one, because most services do not handle having shared block devices.

User quotas for a disk are sent by the service when requesting disk properties. This limit is sent with to additional HTTP headers: `X-DiskUser-Limit` defining how many users can use a disk at the same time and `X-DiskUser-Remaining` defining how many users can still attach the disk.

Disk user quotas will not be verified by the storage service, the client has to do those check to ensure it can attach a disk. The official Stratuslab command line distribution is doing those verifications.

3 Component Interactions

The storage service is a core component of the StratusLab distribution and interacts with other services:

- The first interaction is with the virtual machine manager, it must handle persistent disks in the VM template and know what to do when a user requests that a disk to be attached. For this, the service uses URLs like

  ```
  pdisk:<service_name>:<service_port>:<disk_uuid>
  ```

  This lets internal scripts have enough information to attach the disk. This URL is handled by the OpenNebula TM clone script which invokes a script on the node where the machine will be started.

- The second interaction is with the node where the virtual machine has been launched. A script needs to retrieve the shared disk on the service to make it available to the VM. There are two scripts on each node: `attach-persistent-disk` and `detach-persistent-disk`. These are able to attach disks at start-up but also to hot plug it. Those script are also responsible for registering a disk attachment (detachment) when VM boots (shuts down).

4 Use cases

TODO

5 Future work

- Allow direct download of initial disk data (e.g. for caching of disk images)
- External access to data via http(s) and perhaps gsiftp.
- Direct booting from persistent disk image.
- Quotas for total disk space.
- ACLs for disk access with users and groups.
- Snapshot management.
- Asynchronous operation of the create and delete methods.
- Refactoring to allow plugins for different storage and sharing methods.

References

[1] RESTlet–RESTful web framework for Java. http://www.restlet.org/.

[2] Network Working Group. RFC 1094 - NFS: Network File System Protocol Specification. http://tools.ietf.org/html/rfc1094.

[3] Network Working Group. RFC 3720 - Internet Small Computer Systems Interface (iSCSI). http://tools.ietf.org/html/rfc3720.

[4] Network Working Group. RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace. http://tools.ietf.org/html/rfc4122.html/.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.