Enhancing Grid Infrastructures with
Virtualization and Cloud Technologies

StratusLab Marketplace

Technical Note TN-Marketplace (V3.0)
2 March 2011

Abstract

StratusLab provides a complete, open-source solution for deploying an "Infrastructure as a Service" cloud infrastructure. Use of the cloud requires the use of prepared machine and disk images. The StratusLab Marketplace serves as a registry for shared images. It is at the center of the image handling mechanisms in the StratusLab distribution.

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.

Contributors

Name	Partner	Sections
Mohammed AIRAJ	CNRS-LAL	All
Marc-Elian BEGIN	SixSq	All
Stuart KENNY	TCD	All
Charles LOOMIS	CNRS-LAL	All
David O'CALLAGHAN	TCD	All

Document History

Version	Date	Comment
0.1	13 May 2010	Initial version for comment.
1.0	26 Jan. 2011	Collect information in JIRA and previous notes.
1.0	2 Feb. 2011	Update URL mapping.
1.1	2 Feb. 2011	Update version number.
2.0	12 Feb. 2011	Update URL mapping, schemas, and examples.
3.0	2 Mar. 2011	Add use cases and workflows involving the Marketplace.

Contents

List of Tables

1 Introduction

StratusLab provides a complete, open-source solution for deploying an "Infrastructure as a Service" cloud infrastructure. Use of the cloud requires the use of prepared machine and disk images. Although StratusLab provides tools to simplify the creation of these images, the procedure for doing so remains a significant hurdle for use of a cloud. Consequently, StratusLab encourages the sharing and reuse of existing images to reduce this barrier.

The Marketplace it at the center of the image handling mechanisms in the StratusLab cloud distribution. It contains metadata about images and serves as a registry for shared images. There are two primary use cases for images in the cloud: 1) creating new images and making them available and 2) instantiating a referenced image.

1.1 Registering a New Image

New machine or disk images can either be created from scratch following the StratusLab guidelines or be modified versions of existing images. The workflow for creating an image and making it available for other involves the following steps:

Image Creation A new machine or disk image is created from scratch or from an existing image. The `stratus-create-image` command can facilitate this process.

Transfer to Storage The new image is transferred to network accessible storage. This may be storage provided by a StratusLab cloud or storage provided by the user or users' institute. The physical image must be available via one or more URLs.

Create Metadata The metadata entry for the image must be created. This is an XML file containing information about the image itself including the location URLs for the physical image. The `stratus-build-manifest` command may be helpful here.

Sign Metadata All of the metadata entries must be signed with a valid grid certificate. The `stratus-sign-metadata` will sign the metadata file with a given certificate, inserting the certificate information automatically into the metadata file.

Upload Metadata The signed metadata entry can then be uploaded to the StratusLab Marketplace. This is done via an HTTP POST to a the Marketplace's http://mp.example.org/metadata/ URL.

The Marketplace will then validate the entry and verify the email address given in the image metadata. Once all of the checks have been done, the metadata will be visible in the Marketplace and other users can search for the entry.

1.2 Using a Registered Image

Referencing an existing image and instantiating it involves the following steps:

Request New Instance StratusLab users will create new machine instances through the `stratus-run-instance` command. Disk images can be used in the machine description to attach existing, public disks to a new instance. In both cases, the user can supply an image identifier (e.g. "MMZu9WvwKIro-rtBQfDk4PsKO7_"), a Marketplace URL (e.g. http://mp.example.org/metadata/?id=MMZu9WvwKIro-rtBQfDk4PsKO7_, any URL that provides (directly or indirectly) a valid metadata entry (e.g. http://tinyurl.com/my-favorite-image), or the physical location of the image (e.g. http://example.org/myimage).

Resolve Entry The URL given by the user must be resolved into one or more metadata entries associated with the image. For any of the URLs that provide the metadata entry (or entries) directly, this is a simple HTTP GET on the URL. For location based URLs, this will involve a search in the Marketplace either on the location URL or on a checksum calculated on the fly.

Apply Site Policy The command `stratus-policy-image` will apply a configurable site policy to the image metadata. If none of the metadata entries for a particular image pass the site policy, then the image will not be instantiated.

Download Assuming that the image passed the site policy, then the `stratus-download-image` can be used to download a copy of the image based on the location URLs in the metadata entry.

Once the image is available, it will be instantiated and run. All of the work on the cloud side, it expected to happen in the clone hook of OpenNebula.

2 Marketplace Implementation

The primary requirement of the Marketplace is that it permits all users to search the range of available images for ones which satisfy their requirements–avoiding the effort to create a customized image. Equally important, the Marketplace provides a repository of image metadata that can be used by cloud administrators to decide if a particular image is trusted and can be run on their cloud.

Although the StratusLab cloud distribution must provide a mechanism for storing and retrieving the image file themselves, a conscious decision was made to separate the storage and transport of image file from the Marketplace implementation. Storing the image files outside of the Marketplace:

- Makes it easier to scale the Marketplace implementation and to create mirrors of it.
- Allows owners of the image to control access to the image itself.
- Makes the Marketplace implementation independent of the transport protocol, allowing many different protocols to be supported.
- Relieves the operator of the Marketplace from concerns related to image copyrights.

Related tools for downloading referenced images and validation of those images must complement the Marketplace service.

2.1 Requirements

1. Allow anyone to upload valid metadata descriptions to the site.
2. Valid descriptions must be signed by a grid certificate. The endorser information must match the information in the certificate itself. It may also be desirable to allow signatures with a PGP key or SSH key.
3. Valid descriptions must contain a valid email address. The service must confirm the email address for every upload of metadata.
4. All valid descriptions must contain a creation date for the endorsement. The server must only accept descriptions with a creation date within a short time window (e.g. 10 minutes) around the current time.

5. There may be several sets of metadata associated with a particular machine or disk image. This allows third parties to endorse images created by someone else. (E.g. VO endorses an image created by StratusLab.)

6. Users can "replace" existing metadata descriptions by uploading a new signed description. Nonetheless, all validated descriptions uploaded to the site must always be available to provide a history of the metadata evolution.

7. Users must be able to search the metadata database on a reasonable subset of the possible keys. Two required keys are the image identifier and the endorser's email address.

8. Users must be able to download the original signed metadata in the RDF/XML format from the registry. This is the only format that allows the metadata to be cryptographically verified.

9. The registry should allow the metadata to be downloaded in alternate formats, notably JSON and HTML.

10. Descriptions of available images should contain at least one location from which the image can be obtained. Descriptions without a location are appropriate only if the image becomes unavailable.

11. The service must be easy to use from all programming languages (including scripting languages) and usable from a web browser.

12. The underlying schema for the metadata descriptions must be flexible and extensible. The accounts for differing needs of users and eventual evolution of those needs.

2.2 Technology Choices

To ensure that the service can be used easily with all programming languages, the Marketplace will expose a RESTful [11] interface over standard HTTP(S). The implementation will use RESTlet [5], a Java framework for implementing RESTful services.

Semantic web technologies were designed to manage metadata about (third-party) resources identified with a URI. Consequently, they are ideally suited to this situation in which the Marketplace must manage metadata about machine and disk images. These technologies already provide standard formats for the metadata (RDF/XML [4, 2, 3]) and query languages (SeRQL [6], SPARQL [8]). The Marketplace implementation will use the Jena [1] or Sesame [7] frameworks to provide search capabilities over the metadata database.

To validate the metadata associated with a particular image, it is necessary to cryptographically sign individual entries. As the raw format used for the metadata entries will be XML, the XML Signature [10] specification can be reused. This is conveniently a standard part of modern Java runtime environments.

Working with RDF also requires an agreed vocabulary to ensure a common semantic meaning of the metadata tags. The Dublin Core Metadata Initiative has published a vocabulary [9] that can be used for much of the image metadata descriptions. This is complemented by a vocabulary specific to StratusLab. Using RDF also allows additional metadata fields to be specified (in separate namespaces) to complement the standard fields described in this document.

2.3 Image Identifiers

To ensure an unambiguous connection between the metadata entries contained in the Marketplace and the described machine and disk images, a unique image identifier must be used that depends solely on the image's contents.

For the purposes of the Marketplace and caching of images, an identifier derived from the SHA-1 hash of the image file is used. The identifier is the 160-bit SHA-1 hash of the image file (treating the file as a binary file), extended with two zero bits to the left to form a 162-bit value. This 162-bit value is then encoded using the base64url encoding to produce a 27 character image identifier. For example, the SHA-1 hash (written in hex):

```
c319bbd5afc0a22ba3eaed0507c39383ec28eeff
```

becomes the image identifier:

```
MMZu9WvwKIro-rtBQfDk4PsKO7_
```

using this algorithm. Note that to avoid inadvertent or malicious collisions of the SHA-1 additional checksum within the metadata description should also be verified.

2.4 Verification of Uploaded Metadata

When new metadata entries are uploaded to the Marketplace, the service must validate the entry before accepting it. This involves three separate steps:

1. The metadata must be a valid RDF/XML file, following the defined schemas and conventions (details provided in Chapter 3).

2. To avoid having old metadata entries being reposted (for example to make a deprecated image appear to be valid), the server must ensure that entries have a timestamp within a narrow window (e.g. 10 minutes) around the current time.

3. To ensure that the included email address is valid, all additions to the Marketplace must be confirmed by email.

Only validated metadata entries should be visible from the standard Marketplace interface. The service should provide immediate feedback for entries which are not timely or do not conform to standard conventions. Unconfirmed additions should be dropped after an appropriate delay (e.g. 48 hours).

2.5 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 Marketplace. (The 'delete' and 'put' methods are not supported by any URLs.) Within the table "identifier" refers to the 27 character image identifier, "email" refers to the endorser's email address, and "date" refers to endorsement date written in the format `yyyy-MM-ddThh:mm:ssZ`. All of the URLs can have a query string attached to refine the search criteria. All of the URLs support an XML and HTML formats. They may additionally also provide JSON or other representations.

2.6 Storage and Query of Metadata

The metadata entries will be stored using the Jena [1] interfaces (using behind the scenes a relational database). This allows simple queries and queries with SPARQL [8] (or other query languages supported by Jena) to be easily supported. Unsigned representations (e.g. HTML) will be generated via Jena.

Unfortunately, Jena appears unable to accept the signed metadata. Consequently, the raw, signed metadata files will have to be saved on a file system as well. These will be returned directly when XML is requested. On disk the files should be stored with a filename like ⟨identifier⟩-⟨date⟩-⟨email⟩, guaranteeing a unique filename. These may be divided into subdirectories based on the identifier to avoid having too many entries in a single subdirectory.

Table 2.1: *REST URL Mapping for the Marketplace*

URL	get	post	options
/	landing page with information about service and internal links	X	last update time
/slreq	list of metadata terms in slreq namespace	X	X
/slreq/⟨name⟩	description of particular term in slreq namespace	X	X
/slterms	list of metadata terms in slterms namespace	X	X
/slterms/⟨name⟩	description of particular term in slterms namespace	X	X
/endorsers	list of endorsers in database	X	number, last update time
/endorsers/⟨email⟩	statistics about particular endorser	X	number, last update time
/metadata/?⟨query⟩	list of image identifiers and selected fields (query terms of id, email, and date are supported to refine list)	add another metadata entry	number, last update time
/metadata/⟨identifier⟩/⟨email⟩/⟨date⟩	unique metadata entry	X	X
/metadata/query	form for simple query of service	submission of complete query document	X
/confirmations/⟨UUID⟩?action=⟨action⟩	confirmation of upload (action can be 'CONFIRM', 'ABORT', or 'ABUSE')	X	X

3 Metadata Schema and Format

Sharing machine and disk images requires standardized, trusted metadata to allow users to find appropriate images and to allow system administrators to judge the suitability of them.

As stated in the previous chapter, the metadata descriptions are in RDF/XML [4] format and cryptographically signed following the XML Signature [10] specification. The connection between the described image and the metadata description is the image identifier based on the SHA-1 hash.

Figure 3.1 shows an (unsigned) example of the metadata description. The first element is the description of the image containing information about the image file, contained operating system, and location. It also contains the endorsement of the information with information on who endorsed the image and when. The email of the endorser is used as the key and is consequently a required element of the description. A digital signature element ("xmldsig:Signature") follows the "rdf:Description" element for signed metadata entries. (Relevant XML namespaces are listed in Table 3.1.)

The entries in the Marketplace deal with *individual* images. If it is desired that collections of images are signed, then one possibility is to include in each individual entry references to the other image descriptions in the collection. This allows the full collection to be reconstructed from any individual entry. One method of doing this is shown in the example metadata description.

Table 3.1: *XML Namespaces and Prefixes*

rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
dcterms	`http://purl.org/dc/terms/`
slreq	`http://mp.stratuslab.eu/slreq#`
slterms	`http://mp.stratuslab.eu/slterms#`

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:slreq="http://mp.stratuslab.eu/slreq#"
  xmlns:slterms="http://mp.stratuslab.eu/slterms#"
  xmlns:ex="http://example.org/"
  xml:base="http://mp.stratuslab.eu/">

  <rdf:Description rdf:about="#MMZu9WvwKIro-rtBQfDk4PsKO7_">

    <dcterms:identifier>MMZu9WvwKIro-rtBQfDk4PsKO7_</dcterms:identifier>

    <slreq:bytes>100</slreq:bytes>

    <slreq:checksum rdf:parseType="Resource">
      <slreq:algorithm>SHA-1</slreq:algorithm>
      <slreq:value>c319bbd5afc0a22ba3eaed0507c39383ec28eeff</slreq:value>
    </slreq:checksum>

    <slreq:endorsement rdf:parseType="Resource">
      <dcterms:created>2011-01-24T09:59:42Z</dcterms:created>
      <slreq:endorser rdf:parseType="Resource">
        <slreq:email>jane.tester@example.org</slreq:email>
        <slreq:subject>CN=Jane Tester,OU=...</slreq:subject>
        <slreq:issuer>CN=Jane Tester,OU=...</slreq:issuer>
      </slreq:endorser>
    </slreq:endorsement>

    <dcterms:type>machine</dcterms:type>

    <dcterms:valid>2011-07-23T10:59:42Z</dcterms:valid>

    <dcterms:publisher>StratusLab</dcterms:publisher>
    <dcterms:title>linux-with-my-apps</dcterms:title>
    <dcterms:description>A 32-bit ttylinux...</dcterms:description>

    <slterms:location>http://example.org/...</slterms:location>

    <slterms:serial-number>0</slterms:serial-number>
    <slterms:version>1.0</slterms:version>

    <slterms:hypervisor>kvm</slterms:hypervisor>

    <slterms:inbound-port>443</slterms:inbound-port>
    <slterms:outbound-port>25</slterms:outbound-port>
    <slterms:icmp>8</slterms:icmp>

    <slterms:os>ttylinux</slterms:os>
    <slterms:os-version>9.7</slterms:os-version>
    <slterms:os-arch>i486</slterms:os-arch>

    <slterms:deprecated>security issue with app</slterms:deprecated>

    <ex:other-info>additional metadata</ex:other-info>
    <ex:yet-more>still more info</ex:yet-more>

    <ex:relatedImages rdf:parseType="Resource">
      <dcterms:identifier>MMZu9WvwKIro-rtBQfDk4PsKO7_</dcterms:identifier>
      <dcterms:identifier>NMZu9WvwKIro-rtBQfDk4PsKO7_</dcterms:identifier>
      <dcterms:identifier>OMZu9WvwKIro-rtBQfDk4PsKO7_</dcterms:identifier>
      <dcterms:identifier>PMZu9WvwKIro-rtBQfDk4PsKO7_</dcterms:identifier>
    </ex:relatedImages>

  </rdf:Description>
</rdf:RDF>
```

Figure 3.1: *Example (Abbreviated) Metadata Description*

3.1 Signing and Validating StratusLab Metadata files

For signing and validating metadata files we are using XML Signature [10] specification. Commands to support metadata signatures have been written in Java as recent Java virtual machines contain an API implementing this standard.

Metadata files can be signed using grid certificates (in PKCS12 format), PGP key pairs, or DSA/RSA key pairs. Verification and validation automatically detects signature algorithm and type of private key used for signing metadata files, verifies the metadata file and prints, for grid certificates, the DN of the user who signed the metadata file.

3.2 Metadata Elements

Where possible the Dublin Core metadata vocabulary [9] has been used for the metadata description. Table 3.2 shows the terms taken from the Dublin Core specification. Additional terms have been defined by StratusLab to complete the metadata description (see Table 3.3).

Additional terms can be added to the metadata descriptions, but they should appear in their own XML namespaces. *This allows for application-specific metadata and also evolution of the standard schema.* These should appear after the endorsement element in the description.

Table 3.2: *Metadata Elements from Dublin Core*

NS	qname	freq.	XSD	Constraints	Notes
dcterms	identifier	1	string	valid identifier	image identifier
dcterms	isReplacedBy	?	string	valid identifier	image identifier for replacement image
dcterms	replaces	?	string	valid identifier	image identifier for image replaced by this one
dcterms	isVersionOf	?	string	valid identifier	image identifier for parent image
dcterms	valid	?	dateTime	XML DateTime format	expiration date for image metadata
dcterms	title	?	string		short title for humans
dcterms	description	1	string		longer description of the image
dcterms	type	1	string	“machine” or “disk”	type of the described image
dcterms	creator	?	string		name of image or metadata record creator
dcterms	created	?	string		date when metadata record was created
dcterms	publisher	?	string		publisher (group, experiment, project) of image
dcterms	format	1	string		format of machine or disk image

Table 3.3: *Metadata Elements from StratusLab Schema*

NS	qname	freq.	XSD	Constraints	Notes
slreq	endorsement	1	complex		endorsement information
slreq	endorser	1	complex		endorser information
slreq	bytes	1	positive integer		bytes of described image
slreq	checksum	+	string	lowercase hex digits only	checksum in hex with algorithm prefix
slreq	email	1	string		email address of the metadata record creator
slreq	subject	1	string		certificate subject
slreq	issuer	+	string		certificate issuer
slterms	location	*	URI		location hint for download (none if unavailable)
slterms	serial-number	?	non-negative integer		numeric index of image within a series
slterms	version	?	string		version of the image
slterms	hypervisor	?	string		appropriate hypervisors for machine image
slterms	inbound-port	*	unsigned short	0 for all	required inbound port
slterms	outbound-port	*	unsigned short	0 for all	required outbound port
slterms	icmp	*	unsigned byte		ICMP packet types
slterms	os-arch	?	string		OS architecture
slterms	os-version	?	string		OS version
slterms	os	?	string		OS
slterms	deprecated	?	string		reason that image is deprecated (missing if OK)

4 Security Considerations

4.1 Replay of Signed Metadata

As all of the signed metadata data entries are publicly available, there is a risk that an older entry would be "replayed" to replace a newer entry. This would be done, for instance, to make an image that has been deprecated to appear to still be valid.

The implementation makes this difficult to do by:

- Requiring that all metadata entries explicitly include the endorsement timestamp as part of the signed content.
- Only allowing uploaded entries that have a timestamp close to the current time.
- Confirming all changes via the email address in the metadata entry.

To successfully replay a previous entry would require that the endorser's private key be compromised (to update timestamp) as well as his email account.

4.2 Confidentiality of Data

As all of the metadata descriptions on the servers are considered public, there are no concerns about confidentiality. Thus the Marketplace implementation does not need to do anything special.

4.3 Validity and Completeness of Data

If there were an open communication channel between the Marketplace and the user, then responses from the server could be altered in transit. To prevent this, the server must be deployed with a valid certificate and communications must take place over a connection using TLS.

4.4 Altered Images

As the images will be downloaded from other sites, there is a danger that they will be altered (either intentionally or maliciously). As images are selected and evaluated based on the associated metadata, altered images must not correspond to the unaltered image's metadata or to another valid image's metadata.

The image identifier is based on the SHA-1 hash of the image. Although difficult to modify an image while maintaining the SHA-1 hash is difficult, it is possible. Hence an altered image could masquerade as a valid image if only the SHA-1 hash information were used to validate a downloaded image file.

To ensure that altered images are detected in the validation additional information is provided in the metadata descriptions:

- Length of the file in bytes.
- MD5, SHA-1, SHA-256, and SHA-512 hash values.

All of these should be verified with a newly downloaded image. The likelihood that someone can create an altered image with exactly the same length and multiple checksums is miniscule.

4.5 Compromised Marketplace Server

If someone were to take control of the Marketplace server, he could not alter individual metadata entries as those are signed by the endorser's whose keys are not available on the server. However, he could delete entries making, for instance, deprecated images appear valid. This is a significant risk and the server should be operated according to modern best practices.

5 Related Components

To integrate the Marketplace with cloud deployments, three additional components are necessary:

Resolver This resolves a given URL into a set of metadata entries. Currently only Marketplace metadata URLs are supported, so this is simply an HTTP GET on the URL and not a separate component.

Download This is implemented in the `stratus-download-image` command. It supports any transport supported natively by python.

Policy Engine This is implemented in `stratus-policy-image` command. It will validate the metadata entries against a site policy. It currently implements signature validation, endorser white and black lists, and checksum blacklist.

All of these are used on the cloud side and are not visible to end users. Cloud administrators, however, may find them useful in certain circumstances.

References

[1] Jena–A Semantic Web Framework for Java. http://jena.sourceforge.net/.

[2] RDF Primer. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[3] RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[4] RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

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

[6] SeRQL–Sesame RDF Query Language. http://www.openrdf.org/doc/sesame2/users/ch09.html.

[7] Sesame is an open source framework for storage, inferencing and querying of RDF data. http://www.openrdf.org/.

[8] SPARQL Query Language for RDF. http://www.w3.org/TR/2008/REC-rdf-sparql-query-2008011.

[9] DCMI Usage Board. DCMI Metadata Terms. http://dublincore.org/documents/2010/10/11/dcmi-terms/.

[10] Mark Bartel, et al. XML Signature Syntax and Processing (Second Edition). http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/.

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