
nova Documentation

Release 0.42

Anso Labs, LLC

July 19, 2010

CONTENTS

1	Getting Started with Nova	3
1.1	DEPENDENCIES	3
1.2	Recommended	3
1.3	Installation	4
1.4	Configuration	4
1.5	Running	5
2	nova System Architecture	7
2.1	Components	7
3	nova Networking	9
3.1	Components	9
3.2	Component Diagram	9
3.3	State Model	10
3.4	The Public Traffic Path	10
4	Storage in the Nova Cloud	11
4.1	Volume Documentation	11
4.2	Objectstore Documentation	11
5	Auth Documentation	13
5.1	Introduction	13
5.2	Relationship of US eAuth to RBAC	13
5.3	Basic AWS API call structure	13
5.4	Enhancements	14
5.5	CloudAudit APIs	15
5.6	Type declarations	15
5.7	Request Brokering	15
5.8	Dirty Cloud – Hybrid Data Centers	15
5.9	The Details	16
5.10	System limits	16
5.11	Further Challenges	16
5.12	The rbac Module	16
5.13	The signer Module	16
5.14	The users Module	16
5.15	The users_unittest Module	16
5.16	The access_unittest Module	16
6	Compute Documentation	17
6.1	The disk Module	17

6.2	The exception Module	17
6.3	The model Module	17
6.4	The network Module	17
6.5	The node Module	17
6.6	RELATED TESTS	17
6.7	The node_unittest Module	17
7	Endpoint Documentation	19
7.1	The admin Module	19
7.2	The api Module	19
7.3	The cloud Module	19
7.4	The images Module	19
7.5	RELATED TESTS	19
7.6	The api_unittest Module	19
7.7	The api_integration Module	19
7.8	The cloud_unittest Module	19
7.9	The network_unittest Module	19
8	NOVA Libraries	21
8.1	The crypto Module	21
8.2	The adminclient Module	21
8.3	The datastore Module	21
8.4	The exception Module	21
8.5	The flags Module	21
8.6	The rpc Module	21
8.7	The server Module	21
8.8	The test Module	21
8.9	The utils Module	21
9	Nova Fakes	23
9.1	The fakevirt Module	23
9.2	The fakeldap Module	23
9.3	The fakerabbit Module	23
10	Nova Binaries	25
11	Nova Documentation	27
11.1	Modules:	27
12	nova Packages & Dependencies	35
13	Indices and tables	37

Nova is a cloud computing fabric controller (the main part of an IaaS system) built to match the popular AWS EC2 and S3 APIs. It is written in Python, using the Tornado and Twisted frameworks, and relies on the standard AMQP messaging protocol, and the Redis distributed KVS. Nova is intended to be easy to extend, and adapt. For example, it currently uses an LDAP server for users and groups, but also includes a fake LDAP server, that stores data in Redis. It has extensive test coverage, and uses the Sphinx toolkit (the same as Python itself) for code and user documentation. While Nova is currently in Beta use within several organizations, the codebase is very much under active development - there are bugs!

Contents:

GETTING STARTED WITH NOVA

GOTTA HAVE A nova.pth file added or it WONT WORK (will write setup.py file soon)

Create a file named nova.pth in your python libraries directory (usually /usr/local/lib/python2.6/dist-packages) with a single line that points to the directory where you checked out the source (that contains the nova/ directory).

1.1 DEPENDENCIES

Related servers we rely on

- RabbitMQ: messaging queue, used for all communication between components
- OpenLDAP: users, groups (maybe cut)
- ReDIS: Remote Dictionary Store (for fast, shared state data)
- nginx: HTTP server to handle serving large files (because Tornado can't)

Python libraries we don't vendor

- M2Crypto: python library interface for openssl
- curl

Vendored python libraries (don't require any installation)

- Tornado: scalable non blocking web server for api requests
- Twisted: just for the twisted.internet.defer package
- boto: python api for aws api
- IPy: library for managing ip addresses

1.2 Recommended

- euca2ools: python implementation of aws ec2-tools and ami tools
- build tornado to use C module for evented section

1.3 Installation

```
# system libraries and tools
apt-get install -y aoetools vlan curl
modprobe aoe

# python libraries
apt-get install -y python-setuptools python-dev python-pycurl python-m2crypto

# ON THE CLOUD CONTROLLER
apt-get install -y rabbitmq-server dnsmasq nginx
# build redis from 2.0.0-rc1 source
# setup ldap (slap.sh as root will remove ldap and reinstall it)
NOVA_PATH/nova/auth/slap.sh
/etc/init.d/rabbitmq-server start

# ON VOLUME NODE:
apt-get install -y vblade-persist

# ON THE COMPUTE NODE:
apt-get install -y python-libvirt
apt-get install -y kpartx kvm libvirt-bin
modprobe kvm

# optional packages
apt-get install -y euca2ools
```

1.4 Configuration

ON CLOUD CONTROLLER

- Add yourself to the libvirtd group, log out, and log back in
- fix hardcoded ec2 metadata/userdata uri (\$IP is the IP of the cloud), and masqrade all traffic from launched instances

```
iptables -t nat -A PREROUTING -s 0.0.0.0/0 -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j DNAT --to-destination $IP
iptables --table nat --append POSTROUTING --out-interface $PUBLICIFACE -j MASQUERADE
```

- Configure NginX proxy (/etc/nginx/sites-enabled/default)

```
server {
    listen 3333 default;
    server_name localhost;
    client_max_body_size 10m;

    access_log /var/log/nginx/localhost.access.log;

    location ~ /_images/.+ {
        root NOVA_PATH/images;
        rewrite ^/_images/(.*)$ /$1 break;
    }

    location / {
        proxy_pass http://localhost:3334/;
    }
}
```

ON VOLUME NODE

- create a filesystem (you can use an actual disk if you have one spare, default is /dev/sdb)

```
# This creates a 1GB file to create volumes out of
dd if=/dev/zero of=MY_FILE_PATH bs=100M count=10
losetup --show -f MY_FILE_PATH
# replace loop0 below with whatever losetup returns
echo "--storage_dev=/dev/loop0" >> NOVA_PATH/bin/nova.conf
```

1.5 Running

Launch servers

- rabbitmq
- redis
- slapd
- nginx

Launch nova components

- nova-api
- nova-compute
- nova-objectstore
- nova-volume

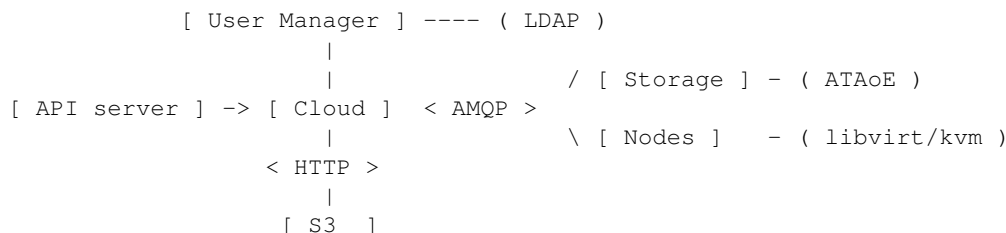
NOVA SYSTEM ARCHITECTURE

Nova is built on a shared-nothing, messaging-based architecture. All of the major nova components can be run on multiple servers. This means that most component to component communication must go via message queue. In order to avoid blocking each component while waiting for a response, we use deferred objects, with a callback that gets triggered when a response is received.

In order to achieve shared-nothing with multiple copies of the same component (especially when the component is an API server that needs to reply with state information in a timely fashion), we need to keep all of our system state in a distributed data system. Updates to system state are written into this system, using atomic transactions when necessary. Requests for state are read out of this system. In limited cases, these read calls are memoized within controllers for short periods of time. (Such a limited case would be, for instance, the current list of system users.)

2.1 Components

Below you will find a helpful explanation.



- API: receives http requests from boto, converts commands to/from API format, and sending requests to cloud controller
- Cloud Controller: global state of system, talks to ldap, s3, and node/storage workers through a queue
- Nodes: worker that spawns instances
- S3: tornado based http/s3 server
- User Manager: create/manage users, which are stored in ldap
- Network Controller: allocate and deallocate IPs and VLANs

NOVA NETWORKING

The nova networking components manage private networks, public IP addressing, VPN connectivity, and firewall rules.

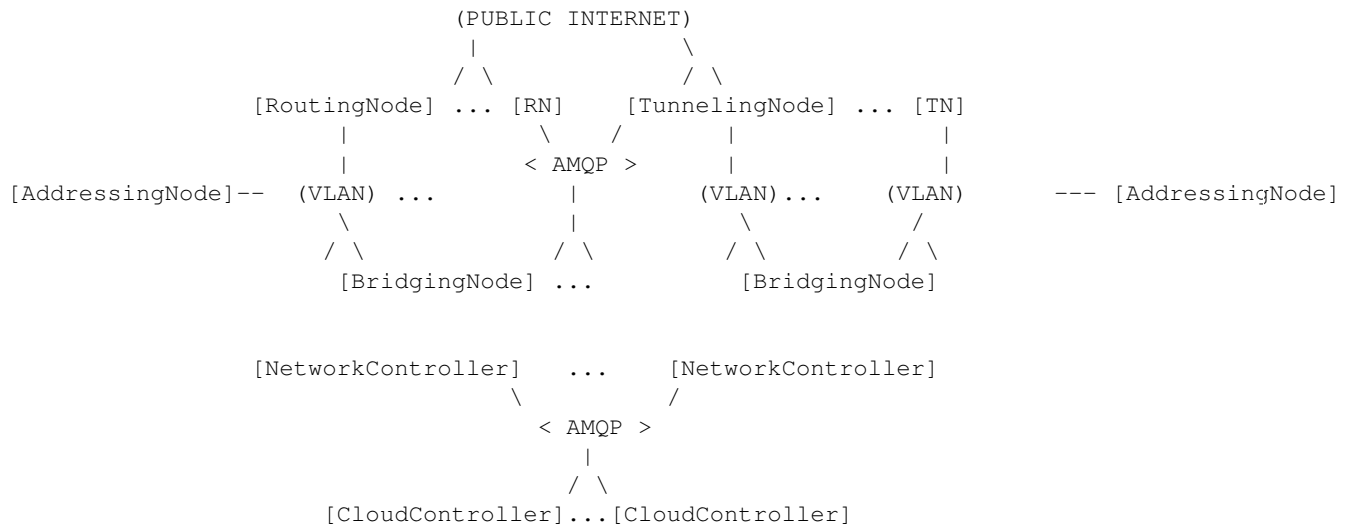
3.1 Components

There are several key components:

- NetworkController (Manages address and vlan allocation)
- RoutingNode (NATs public IPs to private IPs, and enforces firewall rules)
- AddressingNode (runs DHCP services for private networks)
- BridgingNode (a subclass of the basic nova ComputeNode)
- TunnelingNode (provides VPN connectivity)

3.2 Component Diagram

Overview:



While this diagram may not make this entirely clear, nodes and controllers communicate exclusively across the message bus (AMQP, currently).

3.3 State Model

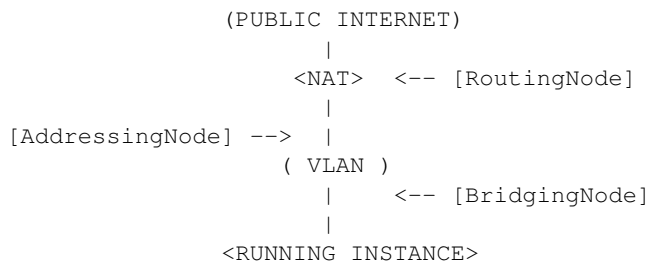
Network State consists of the following facts:

- VLAN assignment (to a project)
- Private Subnet assignment (to a security group) in a VLAN
- Private IP assignments (to running instances)
- Public IP allocations (to a project)
- Public IP associations (to a private IP / running instance)

While copies of this state exist in many places (expressed in IPTables rule chains, DHCP hosts files, etc), the controllers rely only on the distributed “fact engine” for state, queried over RPC (currently AMQP). The NetworkController inserts most records into this datastore (allocating addresses, etc) - however, individual nodes update state e.g. when running instances crash.

3.4 The Public Traffic Path

Public Traffic:



The RoutingNode is currently implemented using IPTables rules, which implement both NATing of public IP addresses, and the appropriate firewall chains. We are also looking at using Netomata / Clusto to manage NATing within a switch or router, and/or to manage firewall rules within a hardware firewall appliance.

Similarly, the AddressingNode currently manages running DNSMasq instances for DHCP services. However, we could run an internal DHCP server (using Scapy ala Clusto), or even switch to static addressing by inserting the private address into the disk image the same way we insert the SSH keys. (See compute for more details).

STORAGE IN THE NOVA CLOUD

There are three primary classes of storage in a nova cloud environment:

- Ephemeral Storage (local disk within an instance)
- Volume Storage (network-attached FS)
- Object Storage (redundant KVS with locality and MR)

4.1 Volume Documentation

Nova uses ata-over-ethernet (AoE) to export storage volumes from multiple storage nodes. These AoE exports are attached (using libvirt) directly to running instances.

Nova volumes are exported over the primary system VLAN (usually VLAN 1), and not over individual VLANs.

AoE exports are numbered according to a “shelf and blade” syntax. In order to avoid collisions, we currently perform an AoE-discover of existing exports, and then grab the next unused number. (This obviously has race condition problems, and should be replaced by allocating a shelf-id to each storage node.)

The underlying volumes are LVM logical volumes, created on demand within a single large volume group.

4.1.1 The `storage` Module

4.1.2 The `storage_unittest` Module

4.2 Objectstore Documentation

This page contains the Objectstore Package documentation.

4.2.1 The `bucket` Module

4.2.2 The `handler` Module

4.2.3 The `image` Module

4.2.4 The `stored` Module

4.2.5 RELATED TESTS

4.2.6 The `objectstore_unittest` Module

AUTH DOCUMENTATION

Nova provides RBAC (Role-based access control) of the AWS-type APIs. We define the following roles:

Roles-Based Access Control of AWS-style APIs using SAML Assertions “Achieving FIPS 199 Moderate certification of a hybrid cloud environment using CloudAudit and declarative C.I.A. classifications”

5.1 Introduction

We will investigate one method for integrating an AWS-style API with US eAuthentication-compatible federated authentication systems, to achieve access controls and limits based on traditional operational roles. Additionally, we will look at how combining this approach, with an implementation of the CloudAudit APIs, will allow us to achieve a certification under FIPS 199 Moderate classification for a hybrid cloud environment.

5.2 Relationship of US eAuth to RBAC

Typical implementations of US eAuth authentication systems are structured as follows:

```
[ MS Active Directory or other federated LDAP user store ]
  --> backends to...
[ SUN Identity Manager or other SAML Policy Controller ]
  --> maps URLs to groups...
[ Apache Policy Agent in front of eAuth-secured Web Application ]
```

In more ideal implementations, the remainder of the application-specific account information is stored either in extended schema on the LDAP server itself, via the use of a translucent LDAP proxy, or in an independent datastore keyed off of the UID provided via SAML assertion.

5.3 Basic AWS API call structure

AWS API calls are traditionally secured via Access and Secret Keys, which are used to sign API calls, along with traditional timestamps to prevent replay attacks. The APIs can be logically grouped into sets that align with five typical roles:

- System User
- System Administrator
- Network Administrator

- Project Manager
- Cloud Administrator
- (IT-Sec?)

There is an additional, conceptual end-user that may or may not have API access:

- (EXTERNAL) End-user / Third-party User

Basic operations are available to any System User:

- Launch Instance
- Terminate Instance (their own)
- Create keypair
- Delete keypair
- Create, Upload, Delete: Buckets and Keys (Object Store) – their own
- Create, Attach, Delete Volume (Block Store) – their own

System Administrators:

- Register/Unregister Machine Image (project-wide)
- Change Machine Image properties (public / private)
- Request / Review CloudAudit Scans

Network Administrator:

- Change Firewall Rules, define Security Groups
- Allocate, Associate, Deassociate Public IP addresses

Project Manager:

- Launch and Terminate Instances (project-wide)
- CRUD of Object and Block store (project-wide)

Cloud Administrator:

- Register / Unregister Kernel and Ramdisk Images
- Register / Unregister Machine Image (any)

5.4 Enhancements

- SAML Token passing
- REST interfaces
- SOAP interfaces

Wrapping the SAML token into the API calls. Then store the UID (fetched via backchannel) into the instance metadata, providing end-to-end auditability of ownership and responsibility, without PII.

5.5 CloudAudit APIs

- Request formats
- Response formats
- Stateless asynchronous queries

CloudAudit queries may spawn long-running processes (similar to launching instances, etc.) They need to return a ReservationId in the same fashion, which can be returned in further queries for updates. RBAC of CloudAudit API calls is critical, since detailed system information is a system vulnerability.

5.6 Type declarations

- Data declarations – Volumes and Objects
- System declarations – Instances

Existing API calls to launch instances specific a single, combined “type” flag. We propose to extend this with three additional type declarations, mapping to the “Confidentiality, Integrity, Availability” classifications of FIPS 199. An example API call would look like:

```
RunInstances type=m1.large number=1 secgroup=default key=mykey confidentiality=low integrity=low ava
```

These additional parameters would also apply to creation of block storage volumes (along with the existing parameter of ‘size’), and creation of object storage ‘buckets’. (C.I.A. classifications on a bucket would be inherited by the keys within this bucket.)

5.7 Request Brokering

- Cloud Interop
- IMF Registration / PubSub
- Digital C&A

Establishing declarative semantics for individual API calls will allow the cloud environment to seamlessly proxy these API calls to external, third-party vendors – when the requested CIA levels match.

See related work within the Infrastructure 2.0 working group for more information on how the IMF Metadata specification could be utilized to manage registration of these vendors and their C&A credentials.

5.8 Dirty Cloud – Hybrid Data Centers

- CloudAudit bridge interfaces
- Anything in the ARP table

A hybrid cloud environment provides dedicated, potentially co-located physical hardware with a network interconnect to the project or users’ cloud virtual network.

This interconnect is typically a bridged VPN connection. Any machines that can be bridged into a hybrid environment in this fashion (at Layer 2) must implement a minimum version of the CloudAudit spec, such that they can be queried to provide a complete picture of the IT-sec runtime environment.

Network discovery protocols (ARP, CDP) can be applied in this case, and existing protocols (SNMP location data, DNS LOC records) overloaded to provide CloudAudit information.

5.9 The Details

- Preliminary Roles Definitions
- Categorization of available API calls
- SAML assertion vocabulary

5.10 System limits

The following limits need to be defined and enforced:

- Total number of instances allowed (user / project)
- Total number of instances, per instance type (user / project)
- Total number of volumes (user / project)
- Maximum size of volume
- Cumulative size of all volumes
- Total use of object storage (GB)
- Total number of Public IPs

5.11 Further Challenges

- Prioritization of users / jobs in shared computing environments
- Incident response planning
- Limit launch of instances to specific security groups based on AMI
- Store AMIs in LDAP for added property control

5.12 The `rbac` Module

5.13 The `signer` Module

5.14 The `users` Module

5.15 The `users_unittest` Module

5.16 The `access_unittest` Module

COMPUTE DOCUMENTATION

This page contains the Compute Package documentation.

6.1 The `disk` Module

6.2 The `exception` Module

6.3 The `model` Module

6.4 The `network` Module

6.5 The `node` Module

6.6 RELATED TESTS

6.7 The `node_unittest` Module

ENDPOINT DOCUMENTATION

This page contains the Endpoint Package documentation.

7.1 The `admin` Module

7.2 The `api` Module

7.3 The `cloud` Module

7.4 The `images` Module

7.5 RELATED TESTS

7.6 The `api_unittest` Module

7.7 The `api_integration` Module

7.8 The `cloud_unittest` Module

7.9 The `network_unittest` Module

NOVA LIBRARIES

8.1 The `crypto` Module

8.2 The `adminclient` Module

8.3 The `datastore` Module

8.4 The `exception` Module

8.5 The `flags` Module

8.6 The `rpc` Module

8.7 The `server` Module

8.8 The `test` Module

8.9 The `utils` Module

NOVA FAKES

9.1 The `fakevirt` Module

9.2 The `fakeldap` Module

9.3 The `fakerabbit` Module

NOVA BINARIES

- nova-api
- nova-compute
- nova-manage
- nova-objectstore
- nova-volume

The configuration of these binaries relies on “flagfiles” using the google gflags package. If present, the nova.conf file will be used as the flagfile - otherwise, it must be specified on the command line:

```
$ python node_worker.py --flagfile flagfile
```


NOVA DOCUMENTATION

This page contains the Nova Modules documentation.

11.1 Modules:

11.1.1 Auth Documentation

Nova provides RBAC (Role-based access control) of the AWS-type APIs. We define the following roles:

Roles-Based Access Control of AWS-style APIs using SAML Assertions “Achieving FIPS 199 Moderate certification of a hybrid cloud environment using CloudAudit and declarative C.I.A. classifications”

Introduction

We will investigate one method for integrating an AWS-style API with US eAuthentication-compatible federated authentication systems, to achieve access controls and limits based on traditional operational roles. Additionally, we will look at how combining this approach, with an implementation of the CloudAudit APIs, will allow us to achieve a certification under FIPS 199 Moderate classification for a hybrid cloud environment.

Relationship of US eAuth to RBAC

Typical implementations of US eAuth authentication systems are structured as follows:

```
[ MS Active Directory or other federated LDAP user store ]
  --> backends to...
[ SUN Identity Manager or other SAML Policy Controller ]
  --> maps URLs to groups...
[ Apache Policy Agent in front of eAuth-secured Web Application ]
```

In more ideal implementations, the remainder of the application-specific account information is stored either in extended schema on the LDAP server itself, via the use of a translucent LDAP proxy, or in an independent datastore keyed off of the UID provided via SAML assertion.

Basic AWS API call structure

AWS API calls are traditionally secured via Access and Secret Keys, which are used to sign API calls, along with traditional timestamps to prevent replay attacks. The APIs can be logically grouped into sets that align with five typical roles:

- System User
- System Administrator
- Network Administrator
- Project Manager
- Cloud Administrator
- (IT-Sec?)

There is an additional, conceptual end-user that may or may not have API access:

- (EXTERNAL) End-user / Third-party User

Basic operations are available to any System User:

- Launch Instance
- Terminate Instance (their own)
- Create keypair
- Delete keypair
- Create, Upload, Delete: Buckets and Keys (Object Store) – their own
- Create, Attach, Delete Volume (Block Store) – their own

System Administrators:

- Register/Unregister Machine Image (project-wide)
- Change Machine Image properties (public / private)
- Request / Review CloudAudit Scans

Network Administrator:

- Change Firewall Rules, define Security Groups
- Allocate, Associate, Deassociate Public IP addresses

Project Manager:

- Launch and Terminate Instances (project-wide)
- CRUD of Object and Block store (project-wide)

Cloud Administrator:

- Register / Unregister Kernel and Ramdisk Images
- Register / Unregister Machine Image (any)

Enhancements

- SAML Token passing
- REST interfaces
- SOAP interfaces

Wrapping the SAML token into the API calls. Then store the UID (fetched via backchannel) into the instance metadata, providing end-to-end auditability of ownership and responsibility, without PII.

CloudAudit APIs

- Request formats
- Response formats
- Stateless asynchronous queries

CloudAudit queries may spawn long-running processes (similar to launching instances, etc.) They need to return a ReservationId in the same fashion, which can be returned in further queries for updates. RBAC of CloudAudit API calls is critical, since detailed system information is a system vulnerability.

Type declarations

- Data declarations – Volumes and Objects
- System declarations – Instances

Existing API calls to launch instances specific a single, combined “type” flag. We propose to extend this with three additional type declarations, mapping to the “Confidentiality, Integrity, Availability” classifications of FIPS 199. An example API call would look like:

```
RunInstances type=m1.large number=1 secgroup=default key=mykey confidentiality=low integrity=low ava
```

These additional parameters would also apply to creation of block storage volumes (along with the existing parameter of ‘size’), and creation of object storage ‘buckets’. (C.I.A. classifications on a bucket would be inherited by the keys within this bucket.)

Request Brokering

- Cloud Interop
- IMF Registration / PubSub
- Digital C&A

Establishing declarative semantics for individual API calls will allow the cloud environment to seamlessly proxy these API calls to external, third-party vendors – when the requested CIA levels match.

See related work within the Infrastructure 2.0 working group for more information on how the IMF Metadata specification could be utilized to manage registration of these vendors and their C&A credentials.

Dirty Cloud – Hybrid Data Centers

- CloudAudit bridge interfaces
- Anything in the ARP table

A hybrid cloud environment provides dedicated, potentially co-located physical hardware with a network interconnect to the project or users’ cloud virtual network.

This interconnect is typically a bridged VPN connection. Any machines that can be bridged into a hybrid environment in this fashion (at Layer 2) must implement a minimum version of the CloudAudit spec, such that they can be queried to provide a complete picture of the IT-sec runtime environment.

Network discovery protocols (ARP, CDP) can be applied in this case, and existing protocols (SNMP location data, DNS LOC records) overloaded to provide CloudAudit information.

The Details

- Preliminary Roles Definitions
- Categorization of available API calls
- SAML assertion vocabulary

System limits

The following limits need to be defined and enforced:

- Total number of instances allowed (user / project)
- Total number of instances, per instance type (user / project)
- Total number of volumes (user / project)
- Maximum size of volume
- Cumulative size of all volumes
- Total use of object storage (GB)
- Total number of Public IPs

Further Challenges

- Prioritization of users / jobs in shared computing environments
- Incident response planning
- Limit launch of instances to specific security groups based on AMI
- Store AMIs in LDAP for added property control

The `rbac` Module

The `signer` Module

The `users` Module

The `users_unittest` Module

The `access_unittest` Module

11.1.2 Compute Documentation

This page contains the Compute Package documentation.

The `disk` Module

The `exception` Module

The `model` Module

The `network` Module

The `node` Module

RELATED TESTS

The `node_unittest` Module

11.1.3 Endpoint Documentation

This page contains the Endpoint Package documentation.

The `admin` Module

The `api` Module

The `cloud` Module

The `images` Module

RELATED TESTS

The `api_unittest` Module

The `api_integration` Module

The `cloud_unittest` Module

The `network_unittest` Module

11.1.4 Nova Fakes

The `fakevirt` Module

The `fakeldap` Module

The `fakerabbit` Module

11.1.5 NOVA Libraries

The `crypto` Module

The `adminclient` Module

The `datastore` Module

The `exception` Module

The `flags` Module

The `rpc` Module

The `server` Module

The `test` Module

The `utils` Module

11.1.6 Volume Documentation

Nova uses ata-over-ethernet (AoE) to export storage volumes from multiple storage nodes. These AoE exports are attached (using libvirt) directly to running instances.

Nova volumes are exported over the primary system VLAN (usually VLAN 1), and not over individual VLANs.

AoE exports are numbered according to a “shelf and blade” syntax. In order to avoid collisions, we currently perform an AoE-discover of existing exports, and then grab the next unused number. (This obviously has race condition problems, and should be replaced by allocating a shelf-id to each storage node.)

The underlying volumes are LVM logical volumes, created on demand within a single large volume group.

The storage Module

The storage_unittest Module

NOVA PACKAGES & DEPENDENCIES

Nova is being built on Ubuntu Lucid.

The following packages are required:

apt-get install python-ipy, python-libvirt, python-boto, python-pycurl, python-twisted, python-daemon,
python-redis, python-carrot, python-lockfile

In addition you need to install python:

- python-gflags - <http://code.google.com/p/python-gflags/>

INDICES AND TABLES

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