# Preface

*This is a proposal for (maybe) improving the Logical Architecture paragraph.*

*In my opinion, the logical architecture should list all the software modules an administrator needs to know in order to understand the platform. At the moment the list seems to be incomplete because some modules are not mentioned and some have a very short description.*

*I made an exercise for listing all the modules and to provide very basic information on them.*

*The list comes in a tabular format because it’s usually easier to modify and maintain.*

*It can be converted into an indented list form like follows:*

* *AREA*
  + *Module (Type): description*

*For example*

* *Core*
  + *nova-compute (daemon): It is primarily a worker daemon that creates and terminates virtual machine instances via hypervisor's APIs (XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware, etc.). The process by which it does so is fairly complex but the basics are simple: accept actions from the queue and then perform a series of system commands (like launching a KVM instance) to carry them out while updating state in the database.*
  + *Nova-scheduler (daemon): …..*

*In order to facilitate the reading, I used the following color code:*

* *parts I added/modified are highlighted in green*
* *parts in yellow are meant to be deleted in the final version.*
* *the not highlighted parts were taken from the current documentation and, in some cases, I reported the source.*

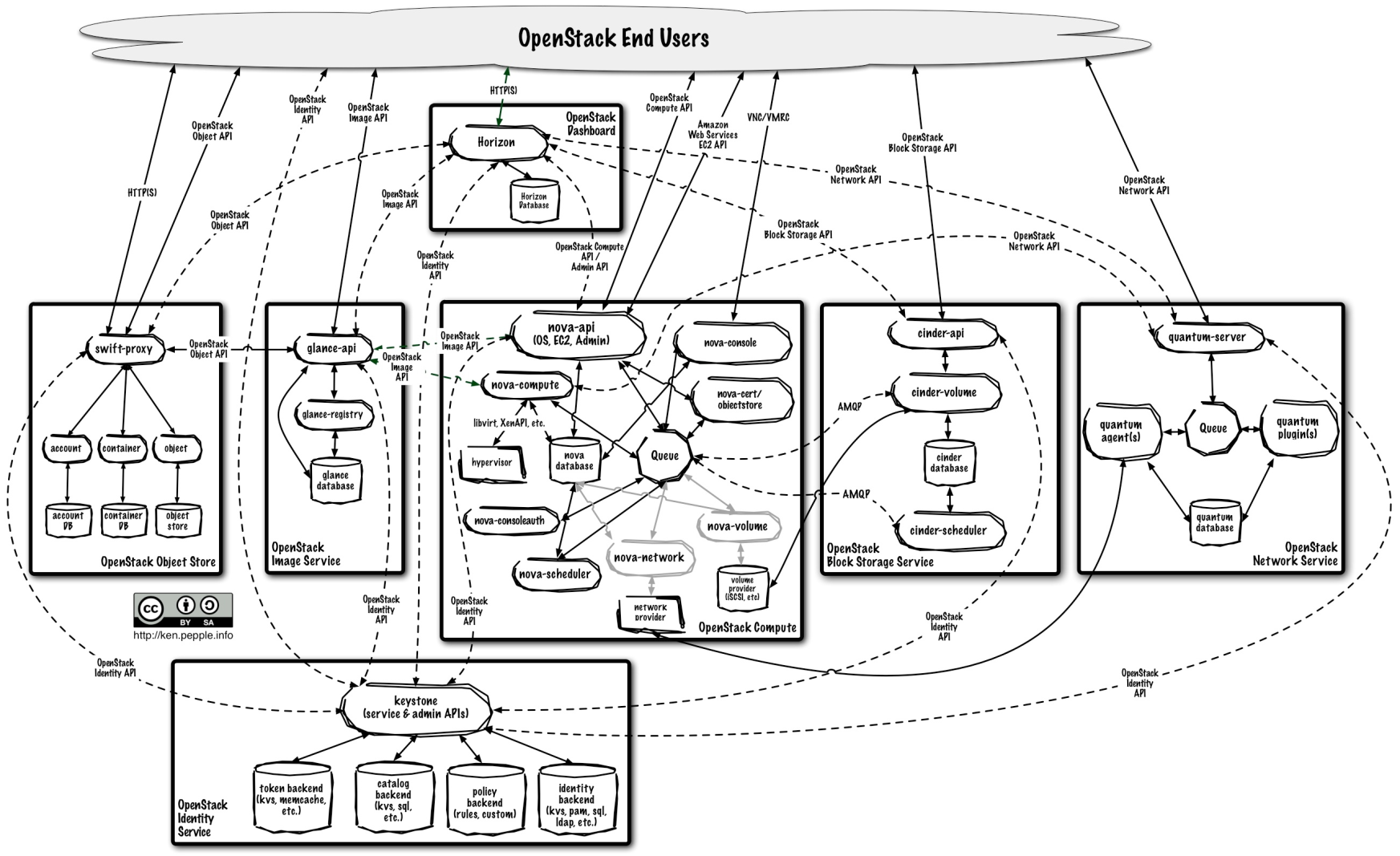
*The color code is aimed to be removed in the final version, of course.*

# Document History

|  |  |
| --- | --- |
| Date | Modifications |
| 2013-03-27 | First Version |
| 2013-03-29 | * Added par. Block Storage * Added par. Enabling Infrastructure (material moved from Compute) * Added Logical Architecture introduction * Terms component and modules used consistently to the manual |

# Logical Architecture

As you can imagine, the logical architecture is far more complicated than the conceptual architecture shown above. As with any service-oriented architecture, diagrams quickly become "messy" trying to illustrate all the possible combinations of service communications. The diagram below, illustrates the most common architecture of an OpenStack-based cloud. However, as OpenStack supports a wide variety of technologies, it does not represent the only architecture possible.



The following paragraphs give some details on the main modules in the OpenStack components

They are not meant to be exhaustive; the objective is to describe the relevant aspects that administrators need to know for better understanding how to design the deployment, and install and configure the whole platform.

Modules are organized according to the functional area they belong (i.e. the kind of functions they implement or deliver) and classified according to their type.

These are the types:

* **daemon**: it runs as a daemon and, on Linux platforms, is usually installed as a service;
* **script**: it is a script run by some “no OpenStack” module when some event happens (at the moment, it is used as a co-routine of dnsmasq for managing IP Addresses released to instances via DHCP protocol);
* **client**: it’s a client for accessing
* **CLI**: it is a Command Line Interpreter for submitting commands to OpenStack Compute

## Compute

Nova is the most complicated and distributed component of OpenStack. A large number of modules cooperate to turn end user API requests into running virtual machines.

Main modules are implemented in Python: the following table reports the list organized for functional areas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Area** | **Module** | **Type** | **Function** | Reference (**column to be removed**) |
| Core | **nova-compute** | daemon | It is primarily a worker daemon that creates and terminates virtual machine instances via hypervisor's APIs (XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware, etc.). The process by which it does so is fairly complex but the basics are simple: accept actions from the queue and then perform a series of system commands (like launching a KVM instance) to carry them out while updating state in the database. | http://docs.openstack.org/folsom/openstack-compute/admin/content/overview-compute-arch.html |
| **nova-schedule** | daemon | It is conceptually the simplest piece of code in OpenStack Nova: take a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on). | http://docs.openstack.org/folsom/openstack-compute/admin/content/overview-compute-arch.html |
| **nova-conductor** | daemon | It’s a module, introduced in Grizzly release, working as a “mediator” between nova-compute and the database. It is aimed to eliminate all the direct accesses to the cloud database made by nova-compute.  nova-conductor scales horizontally but it shouldn’t be deployed on the same node(s) where nova-compute runs. | http://russellbryantnet.wordpress.com/2012/11/19/a-new-nova-service-nova-conductor/ |
| API | **nova-api** | daemon | It accepts and responds to end user compute API calls. It supports OpenStack Compute API, Amazon's EC2 API and a special Admin API (for privileged users to perform administrative actions).  Depending on the deployment, it can also work as metadata server (see also nova-api-metadata) providing metadata to running instances.  It initiates most of the orchestration activities (such as running an instance) as well as enforces some policy (mostly quota checks). | http://docs.openstack.org/folsom/openstack-compute/admin/content/overview-compute-arch.html |
| **nova-api-metadata** | daemon | It accepts metadata requests from instances (*insert link to Metadata Service paragraph*).  The nova-api-metadata service is generally only used when running in multi-host mode with nova-network installations. | http://docs.openstack.org/trunk/openstack-compute/admin/content/metadata-service.html |
| Block Storage | **nova-volume** | daemon | It manages the creation, attaching and detaching of persistent volumes to compute instances (similar functionality to Amazon’s Elastic Block Storage). It can use volumes from a variety of providers such as iSCSI or [Rados Block Device in Ceph](http://ceph.newdream.net). A new OpenStack projects, Cinder, will eventually replace nova-volume functionality. In the Folsom release, nova-volume and the Block Storage service will have similar functionality.  nova-volume is deprecated and is going to be removed at the release of Grizzly. | <http://ken.pepple.info/openstack/2012/09/25/openstack-folsom-architecture/> |
| Networking | **nova-network** | daemon | It accepts networking tasks then performs tasks to manipulate the network (such as setting up bridging interfaces or changing iptables rules). This functionality is being migrated to Quantum, a separate OpenStack service composed of several modules. | <http://docs.openstack.org/trunk/openstack-compute/admin/content/overview-compute-arch.html> |
| **nova-dhcpbridge** | script | It tracks leases and releases in the  database, using dnsmasq's dhcp-script facility.  This functionality is being migrated to Quantum; a different script is provided. | <http://docs.openstack.org/trunk/openstack-compute/admin/content/configuring-flat-dhcp-networking.html> |
| Console Interface | **nova-novncproxy** | daemon | It’s a proxy for accessing running instances through a VNC connection.  It supports browser-based novnc clients. | <http://docs.openstack.org/developer/nova/runnova/vncconsole.html> |
| **nova-xvpnvcproxy** | daemon | It’s a proxy for accessing running instances through a VNC connection.  It supports a Java client specifically designed for OpenStack. |  |
| **nova-consoleauth** | daemon | It authorizes user’s tokens that console proxies provide (see nova-novncproxy and nova-xvpnvcproxy).  This service must be running in order for console proxies to work. Many proxies of either type can be run against a single nova-consoleauth service in a cluster  configuration. | http://docs.openstack.org/trunk/openstack-compute/admin/content/about-nova-consoleauth.html |
| **nova-console** | daemon | It is a XenAPI-specific service that is not used by the most recent VNC proxy architecture. | http://docs.openstack.org/trunk/openstack-compute/admin/content/about-nova-consoleauth.html |
| **nova-xvpvncviewer** | vnc-client | It is a java client for accessing running instances via VNC protocol |  |
| TBD: I don’t know how to classify | nova-cert | daemon | It manages x509 certificates.  *Not clear what it really does (what functions, what interfaces, etc )Iwas not able to find documentation.* | http://docs.openstack.org/developer/nova/api/nova.cert.manager.html |
| Image Management | **nova-objectstore** | daemon | It provides an S3 interface for registering images onto the image management service (see glance)  It is mainly used for installations that need to support euca2ools. euca2ools talk to nova-objectore in “S3 language” and nova-objectstore translates S3 requests into glance requests | http://docs.openstack.org/trunk/openstack-compute/admin/content/configuring-compute-to-use-the-image-service.html |
| Command Line Interpreter | **nova** | CLI | Client for submitting either tenant administrator’s commands or cloud user’s commands. |  |
| **nova-manage** | CLI | Client for submitting cloud administrator’s commands. |  |
| ***euca2ools*** | CLI | This is not an OpenStack module but it can be supported by OpenStack.  It’s a set of command line interpreter commands form managing cloud resources.  Provided that nova-api is configured for supporting EC2 interface, euca2ools can be used for submitting cloud managing commands.  For more information on euca2ools, see http://www.eucalyptus.com/eucalyptus-cloud/documentation/2.0 | Not being an OpenStack Compute module, this description could be moved below. |

During OpenStack evolution, the functions implemented by some modules have been separated from OpenStack Compute project and allocated to other OpenStack projects for better management and evolution.

This happened to:

* block storage functions, currently implemented in OpenStack Block Storage project (code name Cinder);
* network management functions, originally implemented in OpenStack Compute are currently implemented in OpenStack Virtual Network project (code name Quantum) even though several installations still use the “traditional” services;
* others?

Nova interacts with many other OpenStack services: Keystone for authentication, Glance for images and Horizon for web interface. The Glance interactions are central. The API process can upload and query Glance while nova-compute will download images for use in launching images.

## Block Storage

Cinder separates out the persistent block storage functionality that was previously part of Openstack Compute into it's own service. The OpenStack Block Storage API allows for manipulation of volumes, volume types (similar to compute flavors) and volume snapshots.

Main modules are implemented in Python: the following table reports the list organized for functional areas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Area** | **Module** | **Type** | **Function** | Reference (**column to be removed**) |
| Core | **cinder-volume** | daemon | It receives requests for managing volumes (e.g. creating, removing, attaching, etc…) and translates them into commands for the underlying technologies used to implement block storage elements. It can interact with a variety of storage providers through a driver architecture. | http://docs.openstack.org/folsom/openstack-compute/admin/content/overview-block-storage-arch.html |
| **cinder-scheduler** | daemon | It receives volume creation requests and picks the optimal block storage provider node to create the volume on. | http://docs.openstack.org/folsom/openstack-compute/admin/content/overview-block-storage-arch.html |
| API | **cinder-api** | daemon | It accepts API requests and routes them to cinder-volume for action. | http://docs.openstack.org/folsom/openstack-compute/admin/content/overview-block-storage-arch.html |
| Command Line Interpreter | **cinder** | CLI | Client for submitting either tenant administrator’s commands or cloud user’s commands. |  |
| **cinder-manage** | CLI | Client for submitting cloud administrator’s commands. |  |

Volumes are made available (or presented) to instances as iSCSI devices.

They are implemented as Logical Volumes (LV) hosted on nodes running cinder-volume daemon. Cinder-volume uses the Linux Logical Volume Manager (LVM) lvm2 for managing volumes.

Nodes running VM, on the other hand, in addition to run nova-compute daemon, use open-iSCSI for accessing the volumes.

<Insert a picture to clarifying>

## Enabling Infrastructure

OpenStack uses some software products as enabling infrastructure:

Amessage queue provides a central hub for passing messages between daemons. This is usually implemented with [RabbitMQ](http://www.rabbitmq.com/) today, but could be any AMPQ message queue (such as [Apache Qpid](http://qpid.apache.org/)). New to the Folsom release is support for [Zero MQ](http://www.zeromq.org/);

A SQL database stores most of the build-time and run-time state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available and projects. Theoretically, OpenStack Nova can support any database supported by SQL-Alchemy but the only databases currently being widely used are sqlite3 (only appropriate for test and development work), MySQL and PostgreSQL.