**Revision History**

| **Version #** | **Date** | **Author / Editor** | **Version / Revision Comments** |
| --- | --- | --- | --- |
| 1.0 | 11/20/2018 | Vinod Durisala |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**:

* Architecture of the Nimbus framework
* Goals of Nimbus
* Detailing on two main technologies used
* How @Config annotations are used in the framework with examples

**Architecture of the Nimbus framework:**

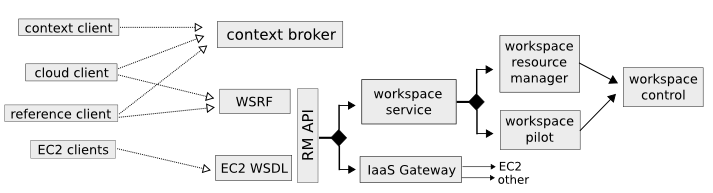
1. **What is Nimbus:**

Nimbus is a set of open source tools that together provide an "**Infrastructure-as-a-Service**" (IaaS) cloud computing solution. Our mission is to evolve the infrastructure with emphasis on the needs of science, but many non-scientific use cases are supported as well.

Nimbus allows a client to lease remote resources by deploying virtual machines (VMs) on those resources and configuring them to represent an environment desired by the user.

It was formerly known as the "Virtual Workspace Service" (VWS) but the "workspace service" is technically just one the components in the software

**Nimbus Architecture**:



**Components**:

**Workspace**: The Workspace Service as site manager

**WSRF**: A based remote protocol implementation

**EC2**: A based remote protocol implementation (partial)

**RM API**: The bridge between remote protocols/security and specific site manager implementations.

**Cloud client**: Aims to get users up and running in minutes with instance launches and one-click clusters.

**Reference Client**: exposes the entire feature set in the WSRF protocol as a commandline client (with underlying Java client library). For advanced uses, scripting, portal integration, etc

**Workspace Pilot**: allows you to integrate VMs with resources already configured to manage jobs (i.e., already using a batch scheduler like PBS)

**Workspace-Control**: agent implements VMM and network specific tasks on each hypervisor

**Context Broker**: allows clients to coordinate large virtual cluster launches automatically and repeatably

**Context Agent**: lives on VMs and interacts with the Context Broker at VM boot

**EC2 backend**: allows the service to turn around and secure remote resources from off-site

**Goals of Nimbus:**

* Allow providers to build clouds
  + Private clouds (privacy, expense considerations)
  + Workspace Service: open source EC2 implementation
* Allow users to use cloud computing
  + Do whatever it takes to enable scientists to use IaaS
  + Context Broker: turnkey virtual clusters
  + IaaS Gateway: interoperability
* Allow developers to experiment with Nimbus
  + For research or usability/performance improvements
  + Community extensions and contributions
* Two main technologies in Nimbus

**What is the EC2 Frontend**:

This is an implementation of the Amazon Elastic Compute Cloud (EC2) web services description (WSDL) that allows you to use clients developed for the real EC2 system against Nimbus based clouds.

**EC2 Operations**:

Nimbus provides a partial protocol implementation of EC2's WSDL (namespace http://ec2.amazonaws.com/doc/2008-05-05/, a previous version supported 2008-02-01). The operations behind these EC2 commandline clients are currently provided:

* + ec2-describe-images - See what images in your personal cloud directory you can run.
  + ec2-run-instances - Run images that are in your personal cloud directory.
  + ec2-describe-instances - Report on currently running instances.
  + ec2-terminate-instances - Destroy currently running instances.
  + ec2-reboot-instances - Reboot currently running instances.
  + ec2-add-keypair [\*] - Add personal SSH public key that can be installed for root SSH logins
  + ec2-delete-keypair - Delete keypair mapping.

There are two options for add-keypair implementations that can be chosen by the administrator in the conf file:

* + - One is the normal implementation where the server-side generates a private and public key (using jsch) and delivers the private key to you.
    - The other (configured by default) is a break from the regular semantics. It allows the keypair "name" you send in the request to be the name AND the public key value. This means there is never a private key server-side and also that you can use keys you aready have created on your system. (In a sense, this is add-keypair as opposed to the normal behavior which should perhaps be named create-keypair).

**What is the EC2 backend:**

This is a workspace service backend that serves as a portal to the Amazon Elastic Compute Cloud (EC2).

It allows clients to boot virtual machines in the Amazon cloud using grid protocols and their X509 credentials, first passing through the service's authorization and accounting layers.

EC2 gateway provides:

* + The ability to run any public Amazon Machine Image (AMI) image on Amazon as well as whatever AMIs the workspace service's credentials have access to privately.
  + Asynchronous WSN notifications about status (EC2 does not, it relies on polling)
  + Adjustment of the root account's SSH pubkey authorized\_keys ("personalization") on the VM

Running time enforcement:

* + It makes the public IP address of the VM known to you via resource property when the address becomes available (on EC2 this is known only after it begins to run).
  + Detailed accounting that the authorization layer can use to make decisions based on a client's current aggregate and reserved usage.

This code is not in a current release, but it is currently deployed.

* **How @Config annotations are used in the framework with examples**
  + **@Config annotation:**

A class with @Config annotation is used to perform an action on button click. In most cases, the action is to retrieve values via HTTP Rest calls from database (MongoDB), and display on the web page.

In the example shown above, when the button is clicked, the control will be navigated to the specified url.  
*nav* is the http call for navigation

**Attributes**:

**url** takes String value

**The possible Actions are**:

1. get for HTTP GET
2. new for HTTP post
3. update for HTTP update
4. delete for HTTP delete
5. search for searching
6. nav for navigation
7. process for custom process/ work-flow definitions

If we have multiple url’s needs to be configured then we have @Configs annotation. It has attribute ‘values’ and it takes value as ‘Config[]’

Example on Config Annotation:

@Config(url="/pageAddEditGoal/tileEditGoal/sectionEditGoal/goalDetailsForm/\_nav?pageId=pageCarePlanSummary")

@Button(type = Button.Type.PLAIN)

Private String cancel;

Example with Configs Annotation:

@Configs({

@Config(url="~/client/org\_name/\_update"),

@Config(url="~/client/org\_name/\_process?fn=\_set&amp;url=/p/cmcase/\_search?fn=query&amp;where=cmcase.patientReferred.firstName.eq('<!/.m/patientReferred/firstName!>').and(cmcase.patientReferred.lastName.eq('<!/.m/patientReferred/lastName!>'))"),

@Config(url="~/client/org\_name/\_nav?pageId=vpAdvancedCaseSearch")

         })