WhirrDesign

Goals

Whirr is cloud-neutral

Whirr should provide a cloud-neutral API for running services in the cloud. The choice of cloud provider should be a configuration option, rather than require the user to use a different API. This may be achieved by using cloud-neutral provisioning and storage libraries such as jclouds, libcloud, or fog, or similar. However, Whirr's API should be built on top these libraries and should not expose them to the user directly.

In some cases using cloud-specific APIs may be unavoidable (e.g. EBS storage), but the amount of cloud-specific code in Whirr should be kept to a minimum, and hopefully pushed to the cloud libraries themselves over time.

Whirr prefers minimal image management

Building and maintaining cloud images is a pain, especially across cloud providers where there is no standardization, so Whirr takes the approach of using commonly available base images and customizing them on or after boot. Tools like cloudlets look promising for maintaining custom images, so this is an avenue worth exploring in the future.

Whirr hides provisioning scripts

Whirr doesn't mandate any particular solution. Runurl is a simple solution for running installation and configuration scripts, but some services may prefer to use Chef or Puppet. The details should be hidden from the Whirr user.

Whirr provides a common service API

The Whirr API should not be bound to a particular version of the API of the service it is controlling. For example, you should be able to launch two different versions of Hadoop using the same Whirr API call, just by specifying a different configuration value. This is to avoid combinatorial explosion in version dependencies.

Of course, having launched a particular version of a Hadoop cluster, you will need the correct version of the Hadoop library to interact with it, but that is the client's concern, not Whirr's. We see this in the tests for Whirr - they have dependencies on particular versions of Hadoop, while the Whirr core library has no dependency on Hadoop.

Whirr has smart defaults

Whirr should start a service with sensible defaults, so the user can just start using it. It should be possible to override the defaults too. Selection of good defaults is something that a community-based project should be able to do well. Having the concept of a profile might be a good idea: since the characteristics of a small transient cluster for testing may differ from a large long-lived cluster.

Non-goals

Whirr does not have inter-language service control

It's not an immediate goal to be able to start a cluster using the Python API, then stop it using the Java API, for example.