HAGGIS
Accounting Group Management @ CERN

European HTCondor Workshop
Nikolaos Petros Triantafyllidis
ntrianta@cern.ch, CERN

Introduction

Haggis was first created in order to help inject into HTCondor fairshare and user mapping information. Initially, Haggis would read the Accounting Group management database to retrieve group quota information. Moreover it would continuously query CERN's LDAP server in order to retrieve CERN user information. Haggis would then expose this information via a simple REST API.

Two command-line utilities, haggis-fairshare and haggis-maps, would consume the API and generate the necessary files that Condor Negotiator and Condor Schedd need to determine user submission rights and fairshares.

Haggis has since evolved to be a full-fledged information system, aiming to become the ultimate Source of Truth for Accounting Group resource management and charging information for our HTCondor deployment.

Haggis' code follows a layered, pluggable architecture, providing enough abstraction to make it easily modifiable for any HTCondor deployment.

Resource Management

Haggis enforces a data model that has the notion of the resource pool at its base. At CERN we wish to allocate to our Accounting Groups different resources under different pools. Thus the 'Pool' entity forms the root of Haggis' resource tree. An example of this model can be seen below:

- Batch Pool
    - Group ALICE, Quota: 15000
        - Group ALICE Production, Quota: 10000
        - Group ALICE Grid, Quota: 5000
        - …
        - ...
    - Group CMS, Quota: 10000
        - Group CMS Production, Quota: 5000
        - Group CMS Pilot, Quota: 5000
    - ...
- HPC Pool
    - Group ALICE, Quota: 12000
        - Group ALICE Production, Quota: 7000
        - Group ALICE Pilot, Quota: 5000
    - Group CMS, Quota: 6000
        - Group CMS Production, Quota: 2000
        - Group CMS Pilot, Quota: 4000
    - ...
- Tier 0 Pool
    - Group ATLAS, Quota: 20000
        - Group Atlas Production, Quota: 12000
        - Group Atlas Pilot, Quota: 8000
    - …

Haggis enforces this model by manipulating these three entities: Pools, Accounting Groups, and Pool to Group mappings. Pools and Accounting Groups exist as separate entities containing only basic information such as IDs and names. An Accounting Group makes sense in production only when it is mapped under a Pool, since this is when it gets its quota. The resource tree is built on the Pool to Group mapping information.

All users of each Accounting Group have submission rights to every Pool where that Accounting Group is mapped, but with different resources on each of them.

Top level admins have the right to increase or decrease the quota of an Accounting Group under a Pool. Haggis also exposes the functionality to group admins to distribute the quota among the groups they manage as they see fit. Haggis responds to a quota change by automatically distributing the change among the groups under a subtree.

User Management

CERN has a pretty big and frequently changing number of users. Haggis needs to be able to obtain this information dynamically and expose a mapping between the users and the Accounting Groups.

To that aim, Haggis defines a Compute Group entity, which represents a user group. In the case of CERN these groups are either standard Unix groups or user mailing lists (called e-groups). Each Compute Group is mapped to an Accounting Group. More than one Compute Groups can be mapped to the same Accounting Group.

Haggis concurrently queries a directory service (e.g. LDAP), at a regular interval, and retrieves all the users of a given Compute Group. It then adds these users to the Accounting Group mapped to this Compute Group, by creating two in-memory lookup structures; one from users to groups and one from groups to users.

Accounting Management

Since Haggis is aimed to serve as a Source of Truth for Accounting Group management information, we found it useful to augment the model with additional accounting (charging) information. We have defined two new entities, Charge Group and Charge Role.

Charge Group is defined per top level Accounting Group and is inherited by all its child nodes. It states the top level organisation entity (experiment, project, etc…) that will be charged for the use of the resources of that Accounting Group.

Charge Role is defined per pool to group mapping and it contains information (properties) on how to account the related resource.

Charge Groups and Roles can, as well, be used to account other services, unrelated to HTCondor.

Code

Haggis is written in Go. It is structured in a layered and pluggable architecture which strictly adheres to the the Dependency Inversion Principle. It provides several layers of indirection and abstraction allowing for easy configuration, adaptation and testing. We can distinguish the following layers:

- Core:
    - Entities: Entities are business objects with embedded some standard business logic. On creation each object will be validated against this logic.
    - Use cases: Use cases contain application specific business logic as well as enforce the previously describe data model. External tools necessary for each use case are abstracted as interfaces which define the 'contract' that needs to be satisfied by the external tools. Use cases follow the interactor pattern.
- Adapters: The Adapters layer is the 'glue' between the core logic and the external tools and libraries. It implements the interfaces defined in the core layer as well as handles the input taken from the external environment before it is passed to the use case interacts.
    - Repositories: Repositories are the implementation of a storage interface. They basically implement CRUD functionality over a resource.
    - Source: A Source is a user directory abstraction with a search function.
    - Controllers: Controllers are input/output handlers that serialise and deserialise input and output messages to an from the core layer.
- Infrastructure: The infrastructure layer holds anything external to the core logic. For example database and protocol drivers, web router and middleware, etc.

An additional layer we have implemented is the Authorization layer. We feel that any authz concern is irrelevant to a use case, thus any authz logic is external to Haggis. The basic principle is that only authorized requests are allowed to call the I/O handler that is bound to a specific use case. That way we ensure that different implementations can exist with different authz models and policies. Authz management is exposed as a separate, yet embedded into Haggis, application.

For the time being Haggis does not provide any support for Authentication or encryption. The software assumes that it will be deployed in an authenticated and

ssl terminated environment. For example in our deployment we use an Apache Web Server providing Kerberos authentication as well as HTTPs support.

We hope that Haggis can be useful to other teams using HTCondor. We are striving to make the software as pluggable and configurable as possible in order for it to be easily tailored to different needs. For example one might use the Core layer as a library that enforces the resource management data model and provide their own implementation on top of it.

Contact us for any question:

Nick Triantafyllids, ntrianta@cern.ch, Developer
Florentia Protopsalti, fprotops@cern.ch, Developer