The 8th DUW

Core, Framework and Configuration System

Federico Stagni

Overview

DIRAC
THE INTERWARE

Components

The DIRAC **components** (what DIRAC runs, and what you install) are:

Services, *Agents*, and *Executors*.

- ***Services***
  - passive components listening to incoming client requests and reacting accordingly by serving requested information (or inserting requests on the *Database* backend).
- ***Agents***
  - active components, similar to cron jobs, which execution is invoked periodically. Agents are animating the whole system by executing actions, sending requests to the DIRAC or third party services.
- ***Executors***
  - similar to consumers of a message queue system. Used in the DIRAC Workload Management System.

Systems and Setups

- *Components* are combined together to form ***Systems*** delivering a complex functionality to the rest of DIRAC, providing a solution for a given class of tasks.
  - E.g.: Workload Management System (WMS) or Configuration System or Data Management System (DMS).
- To achieve a functional DIRAC installation, cooperation of different *Systems* is required. A set of *Systems* providing a complete functionality to the end user form a DIRAC ***Setup***. All DIRAC client installations will point to a particular DIRAC *Setup*. *Setups* can span multiple server installations. Each server installation belongs to a DIRAC *Instance* that can be shared by multiple *Setups*.
- Within a given installation there may be several *Setups*. For example, there can be “Production” *Setup* together with “Test” or “Certification” *Setups* used for development and testing of the new functionality. An instance of a *System* can belong to one or more *Setups*, in other words, different *Setups* can share some *System* instances.

http://dirac.readthedocs.io/en/latest/AdministratorGuide/SetupStructure/index.html

DBs and MQs supported

- ***Databases*** *(MySQL, Oracle, ElasticSearch)*
  - Keep the persistent state of a *System*. They are accessed by Services, Agents, Executors as a kind of shared memory.
    - Most of DBs are in MySQL, which is only hard dependency
    - No Vanilla DIRAC service needs Oracle
    - ElasticSearch is not yet necessary (see the pres from Zoltan tomorrow)
- ***MQs*** *(what talks stomp --> ActiveMQ, RabbitMQ)*
  - No hard dependency, yet (see the pres from Wojciech tomorrow)

DIRAC
THE INTERWARE

DISET

DISET is the communication, authorization and authentication framework on top of which DIRAC services are built

Services expose rpc calls

dips://

```
ALWAYS: Listening at dips://localhost:9170/WorkloadManagement/Matcher
```

DISET implements THIS guy

its “s” is for “secure” (SSL-TLS)
exists also the “dip” version

Support for IPv4 and IPv6 as well

Why?

Long story short: once upon a time, xmlrpc was tried, and it was slow.
So, DIPS was implemented.

DIPS = sockets + SSL + DEncode
(DEncode = DIRAC marshalling library)

Tomorrow afternoon we’ll spend few words on what we’d like to change on this one

AutheNtication

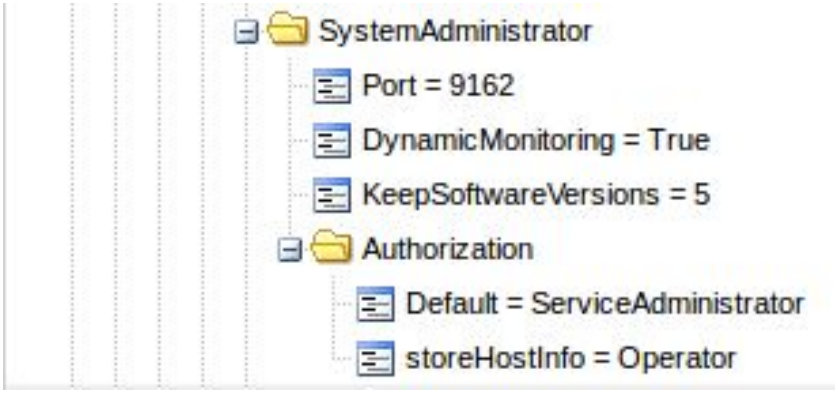
- X509 Certificates
  - openssl at the base
  - pyGSI (part of DIRAC externals) is the current package that creates proxies (and not only)
    - a quite thin layer on top of openssl
    - want to replace it with M2Crypto
- RFC proxies are the default
- DIRAC components by default use the certificate of the host onto which they run
  - Components can be instructed to use a “shifter proxy” for their calls OUT of DIRAC [doc]

AuthoriZation

- **RBAC** (Role Based Access Control) model of AuthZ
  - a *role* (called *property* in DIRAC) carries some authorization
  - a hostname has a DN and some properties
  - a username has a DN, and the *groups* in which it is included
  - a user group has a number of properties

  All the above defined in CS in /Registry section

  → A user creates a proxy with a group and this guarantees certain properties
- Services exposed calls authorized by properties
  - Can have a default
    - e.g. “authenticated” -- meaning everyone with a proxy or certificate known to DIRAC, or “all”
    - Configuration in `/Systems/<setup>/Services/<ServiceName>/Authorization`



DIRAC
THE INTERWARE

Some properties

```
# CS Administrator - possibility to edit the Configuration Service
CS_ADMINISTRATOR = "CSAdministrator"

# Job Administrator can manipulate everybody's jobs
JOB_ADMINISTRATOR = "JobAdministrator"

# Job Monitor - can get job monitoring information
JOB_MONITOR = "JobMonitor"

#Allow managing production
PRODUCTION_MANAGEMENT = "ProductionManagement"
```

On logging

gLogger

(recently) based on python logging module

For every DIRAC component, and every script.

Several backends are possible, allowing for logs centralization.

Configuration System

Configuration sources

- *Command line options:* for all the DIRAC commands there is option '-o' defined which takes one configuration option setting.

```
dirac-wms-job-submit job.jdl -o
/DIRAC/Setup=Dirac-Production
```

- *Command line argument specifying a CFG file*

```
dirac-wms-job-submit job.jdl my.cfg
```

- *$HOME/.dirac.cfg* file in the user's home directory with the *CFG* format
- *$DIRACROOT/etc/dirac.cfg* configuration file in the root directory of the DIRAC installation
- *Configuration Service* Configuration data available from the global DIRAC Configuration Service

Configuration in a cascade

The client needing a configuration option is:

1. first looking for it in the command line arguments. If the option is not found, the search continues in
2. the cfg file on the command line. If not found, keep looking for it in
3. the user configuration file, then in
4. the DIRAC installation configuration file and finally in
5. the Configuration Service.

These gives a **flexible** mechanism of **overriding** global options by specific local settings.

All managed by **gConfig**

Configuration structure

- tree structure, divided in sections, can be seen as directories
- each section can contain other sections and options (the leafs) which contain the actual configuration data.

Sections at the top level:

DIRAC: the most general information about the DIRAC installation.

Systems: Configuration data for all the DIRAC Systems, their instances and components

Registry: Information about DIRAC users, groups and communities (VOs).

Resources: description of all the resources: include computing, storage elements, third party services.

Operations: operational parameters needed to run the system.

“The CS”

The Configuration System is DIRAC’s backbone

no Configuration System → no DIRAC

NB: we often refer to “**the CS**” as DIRAC’s Configuration Service (not the system…)

→ you want/need: **1** master (rw), **n** slaves (ro)

Framework

Framework: functionalities

- Instantiation of DIRAC *components*

  but also DIRAC commands (scripts)
- Management and monitoring of *components*
- Proxies management

http://dirac.readthedocs.io/en/latest/AdministratorGuide/Systems/Framework/index.html

Components (un)installation

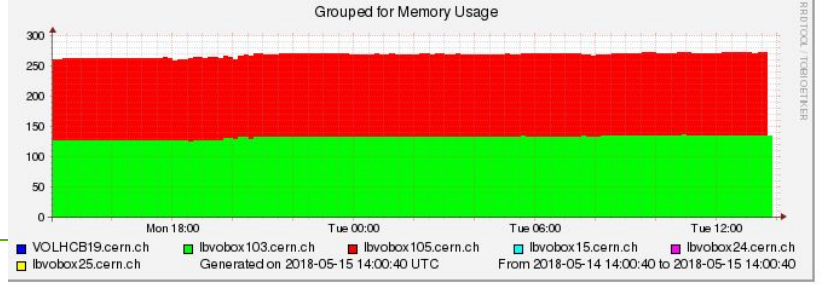
- Framework/SystemAdministrator
  - the only component which is mandatory to run on every host running DIRAC services (on each server)
  - for managing the components on the host
- Framework/ComponentMonitoring:
  - logs information about what components are being installed and uninstalled on which machines, when and by whom.
  - complementary to SystemAdministrator, for a global view
  - Running this service is mandatory

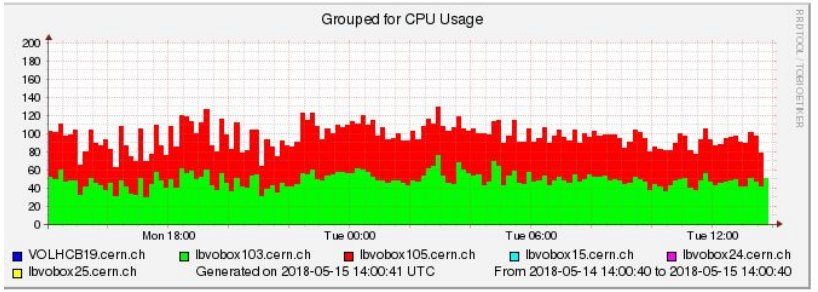
Interaction via:

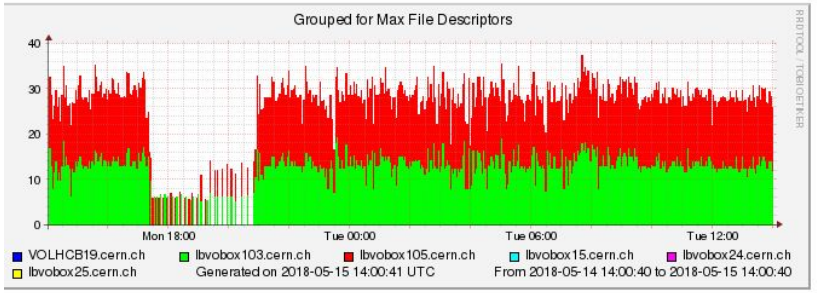
- `dirac-admin-sysadmin-cli`
- SystemAdministrator web app
  - start/stop/restart components
  - view logs
  - update version

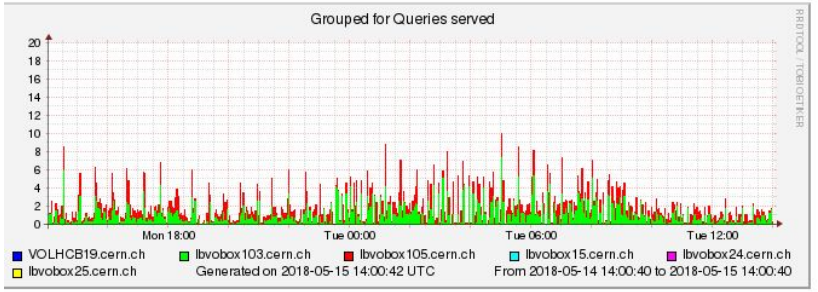
Components monitoring

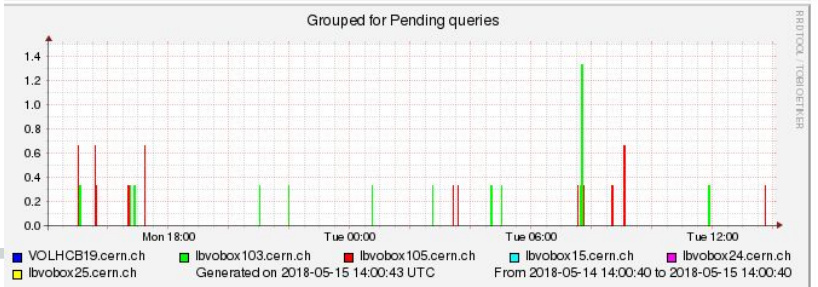
- Framework/Monitoring service
  - Another mandatory service
  - system based on RRD
    - which one day would like to replace/improve
- ActivityMonitor/System Overview Plots web app











CAs update

- multi -server installations required to keep the CA's data up to date
- some DIRAC component require CA's and CRL's such as Elasticsearch, WebAppDIRAC
- Mandatory for creating proxy (dirac-proxy-init)
- BundleDeliveryClient for:
  - synchronizing and downloading:
    - CA's
    - CRL's
  - CA's and CRL's downloaded, if the file can not be created

Other Framework components

- UserProfile for storing user related data
  - widely used by the WebAppDIRAC
- SecurityLogging for security traceability
  - keeping who accessed to given DIRAC service
- SystemLogging for storing the errors of each DIRAC components

Proxies management

- Framework/ProxyManagement service
  - For storing/retrieving proxies in ProxyDB
    - `dirac-proxy-init --upload`
    - **security-wise, you better treat this DB in a bit special way**

Be careful with authorization properties:

- FullDelegation → permits full delegation of proxies
- LimitedDelegation → permits downloading only limited proxies
- PrivateLimitedDelegation → permits downloading only limited proxies for one self

Your pilot jobs will access the ProxyManagement for running the payloads

→ Your pilot group needs the LimitedDelegation property

DIRAC
THE INTERWARE

Questions/comments

?